CSE - 546 Project Report

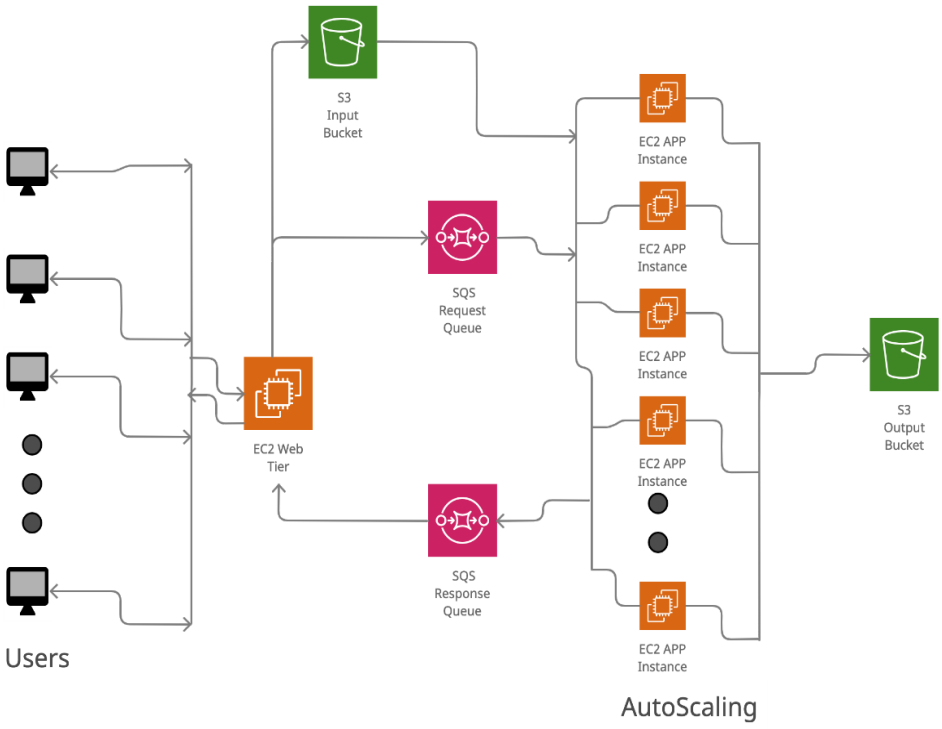
Venkata Surya Shandilya (1219412519)
Gnana Prakash Avvaru (1218707191)
Arun Raj Deva (1218410609)

Problem Statement:

In project 1 we built an elastic application that can automatically scale in and out on demand and cost-effectively by using IaaS cloud. FOr implementing this application we used the Iaas resources provided by Amazon Web Services (AWS). As part of the project we utilized the resources such as EC2 instances, SQS (Simple Queue Service) and S3 (Simple Storage service). Our cloud application provides an image recognition service to users to perform a deep learning model analysis on the user images. Scaling leads to lower maintenance costs, better user experience, and higher agility. Scalability in the cloud effectively facilitates performance, simpler model and with the continuous growth of business the storage space in cloud grows as well.

Design and Implementation

Architecture:



Architectural design of our Implementation

We used the following services of Amazon AWS in our project:

AWS EC2:

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides compute capacity in the cloud. We are using Amazon EC2 instances to run the Web and App tiers of our application. In our application we are using 1 instance for web tier and 0 - 19(Minimum-Maximum) instances for App tier. The number of App instances id determined by the AutoScaling logic.

AWS SQS:

SQS is a managed message queuing service that enables decoupling the application. Using SQS in our application we are sending, storing, and receiving messages between web tier and app tier. In our application we are using FIFO queues which guarantees that messages (requests) are processed exactly once, in the exact order that they are sent.

AWS S3:

Amazon S3 is an object storage service used to store data of images and output classified labels. In our application we created 2 buckets, one an input bucket to store the images and an output bucket to store image names and output labels.

Implementation:

We implemented an auto scalable application which classifies the user uploaded image. We provided an user interface through which a user can upload the images that need to be classified. Once the images are uploaded the web tier adds these uploaded images to the S3 input bucket and also adds them to the SQS request queue. Load balancing is used in the web tier to decide if a new instance is required to classify the new image. We used one instance as a web tier and we used 19 other instances to scale in and out the app instances. Using the load balancer we are deciding if we need to invoke a new instance or not to scale out the application. When the request queue does not have any more requests to process we are terminating and scaling in the app tier instances. Once an app tier is started we are processing the user image using the deep learning model which classifies the image. For this we are accessing the S3 input bucket to get the image, once we access the image we are adding it to the app tier instance and executing the deep learning model using the path of the image. Once the image is classified we are passing the classified output to the response queue and S3 output bucket. Once the output is parsed to S3 and response queue we are deleting the request from the request queue to avoid duplicate

execution of the image. Through the response queue we are parsing the output and displaying it on the user interface using a browser.

Autoscaling:

Through our application we are achieving the scaling up in the web tier and scaling out in the app tier. When a user uploads the images through the user interface we are adding these requested images to the S3 input bucket and adding the image name to the SQS request queue. Whenever a new request is added to the queue, we are checking if an existing instance can process the image or to create a new instance to classify the image. If there is no instance available to classify the image then we are invoking a new app tier instance. If the number of app tier instances is less than 19 and we have requests to process in the SQS request queue then we are invoking a new app tier instance and classifying the requested image. If the number of app tier instances reaches 19 then we are waiting for an existing instance to complete the classifying of the image and accept the new requests from the request queue. This way we are achieving the scaling out of our application. In our application we are using FIFO queues for both request and response. We are using FIFO queues which guarantees that requests are processed exactly once, in the exact order that they are sent. Once the image classification is completed we are adding the image name and output label to the response queue and S3 output bucket. When an app tier instance is created and classification of the image is successfully completed then we are checking if there are any more requests in the request queue that need to be processed. If there are any more requests in the request queue then we are accepting the request from the queue and using the existing instance to classify the new image and if the app tier instance cannot read any new requests from the request queue then we are terminating the instance and achieving the scaling in of our application. Once all the requests are processed by the app tier we are terminating all the app tier instances.

Testing and Evaluation:

Exhaustive and comprehensive testing is done both on the Web and App tiers individually. We then integrated the entire application end-to-end and performed the integration testing to check the correctness of our application. We tested our application with a varying number of user requested images to test the correctness of auto scaling and of the Image Recognition tasks. To crash test our application, we uploaded 500 images to the WebTier and then the Application's response was tested based on the following criteria:

1. **Support for Image upload:** We rigorously tested and validated that 500 images are being correctly uploaded to the input bucket and to the input S3 bucket.
2. **AutoScaling:** This involved a combination of manual and stress testing where the Scaling up of instances is critically evaluated against a large number of concurrent image requests.
3. **Image classification:** Correctness of the image classification produced by our app tier is validated against the correct entries provided to us in the google sheet.
4. **Scaling out of instances:**

   4.1: The instance scale out when there are no more requests is rigorously tested at multiple stages and time stamps during the entire instance's lifetime.

   4.2: We made sure that no instance will terminate as long as there are requests in the input SQS.

Manual Testing of User Interface: The user interface which provides facilities for image upload and output display is critically evaluated for performance and correctness.

Performance Metrics: The following are the average values of time taken by different parts of our application on 10 trials:

Image upload to input SQS and input S3 (100):	9 seconds.
LoadBalancer Scale out:	1minute16 seconds.
Image Recognition and upload to Output SQS and Output S3:	2 minute 43 seconds.
Display of the output:	34 seconds.

Total time for 100 images: 4 minutes, 42 seconds.

Code:

Web-Tier:

launcher.php :

This presents a user interface where the user can upload multiple images. The user can enter the DNS of the launcher into the browser and can upload images.

WebTierMultiple.php:

This uploads the user uploaded images to input SQS and also to the input S3 bucket.

LoadBalancer.php :

This contains the AutoScaling logic which determines the number of EC2 instances to be created depending on the number of requests in the SQS and on the number of running EC2 instances.

sqstester.php:

This will display the results from the Output SQS onto the output page in a tabular format.

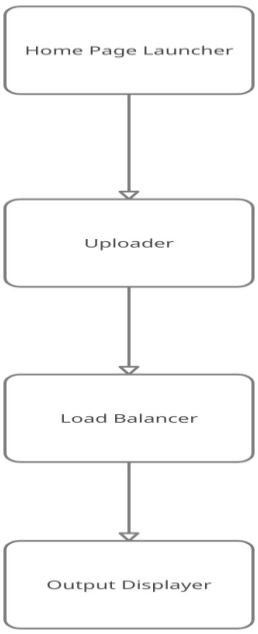
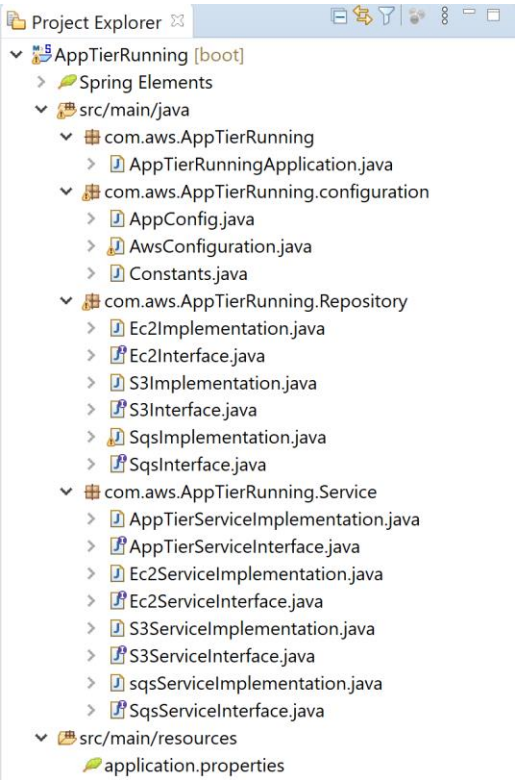


Fig: Sequence of actions performed in Web Tier

App-Tier:

- App tier receives the image name from the SQS queue.
- App tier gets the image from the S3 and pastes it in the Ec2 instance local classifier directory.
- Deep learning model starts executing by taking the image path as an input.
- After image classification the image name and output label will be stored in the S3 bucket and parsed to the SQS response queue in “imageName | outputLabel” format.
- The request is removed from the SQS request queue.
- If the app tier does not get any new requests from the SQS request queue the instance will be terminated, else the new request from SQS request queue will be processed by the app tier instance.

Below screenshot is the structure of the app tier application.



We have used Java Spring Boot for implementing app tier application.

The file names are as follows:

Constants.java : Here, we are defining all the constans that we use over the application at several places . Those are access key, security key, region, queue names and bucket names.

AppConfig.java : Here we are defining the spring beans for all the java classes that we use over the application.

AwsConfiguration.java : Here we are defining the amazon clients for EC2, S3 and SQS so that we can use these clients directly wherever we need over the application.

Ec2Implementation.java : Here we have written the code to end the EC2 App tier instance.

S3Implementation.java : Here we have written the code to create the bucket , get the bucket by bucket name and put the data into S3 bucket as key, value pairs.

SqsImplementation.java : Here we have written the code to delete the message from SQS queue, create a new SQS queue, send the message to SQS queue, receive the message from SQS queue and retrieve the classifier prediction through calling deep learning model in the given

AMI. Here, Iam using the below java runtime exec function to run the image classifier python program given in the AMI from the java program during runtime.

Runtime.getRuntime().exec("python3 image_classification.py test_1.png”);

AppTierServiceImplementation.java : This is where our main auto scale in logic has been written. Here, we will read the messages from the input request queue continuously and store the predicted results from the deep learning model in the S3 output bucket and Response queue. After that, we will delete the message from the input request queue. So, whenever an instance doesn't get a message from the input request queue then the instance will be auto terminated.

Similarly, remaining are the other service classes to call the respective repository functionalities.

AppTierRunningApplication.java : This is the main class of our app tier application where the execution of app tier starts and will call the main functionality in App tier service implementation.

autoStartAppTierRunning.sh : In this shell script, we are writing the command to change the directory having the jar file and writing the command to run that java jar file.

Steps to Execute application:

Setup of Web Tier:

Our Web Tier uses html for markup and PHP for server side scripting. As our AMI is an Ubuntu image, we installed an Ubuntu version of XAMPP PHP server on our Web EC2 instance.

Steps for installing the XAMPP server from your host machine:

Linux & MAC

By executing the following command on a Linux or MAC terminal, you can directly access your box.

ssh -i keyname.pem ubuntu@IP_ADDRESS

Sometimes you will get the following error.

Permissions 0640 for 'keyname.pem' are too open.

It is required that your private key files are NOT accessible by others.

This private key will be ignored.

Load key "keyname.pem": bad permissions

Permission denied (publickey).

sudo chmod 400 keyname.pem

XAMPP Installation Commands for Ubuntu

Download XAMPP for 64 bit

```
wget https://www.apachefriends.org/xampp-files/7.0.23/xampp-linux-x64-7.0.23-0-installer.run
```

Make Execute Installation

```
sudo chmod +x xampp-linux-x64-7.0.23-0-installer.run
```

Run Installation

```
sudo ./xampp-linux-x64-7.0.23-0-installer.run
```

XAMPP instructions

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

XAMPP Core Files : Y (Cannot be edited)

XAMPP Developer Files [Y/n] : **Y**

Is the selection above correct? [Y/n]: **Y**

Installation Directory

XAMPP will be installed to /opt/lampp

Press [Enter] to continue:

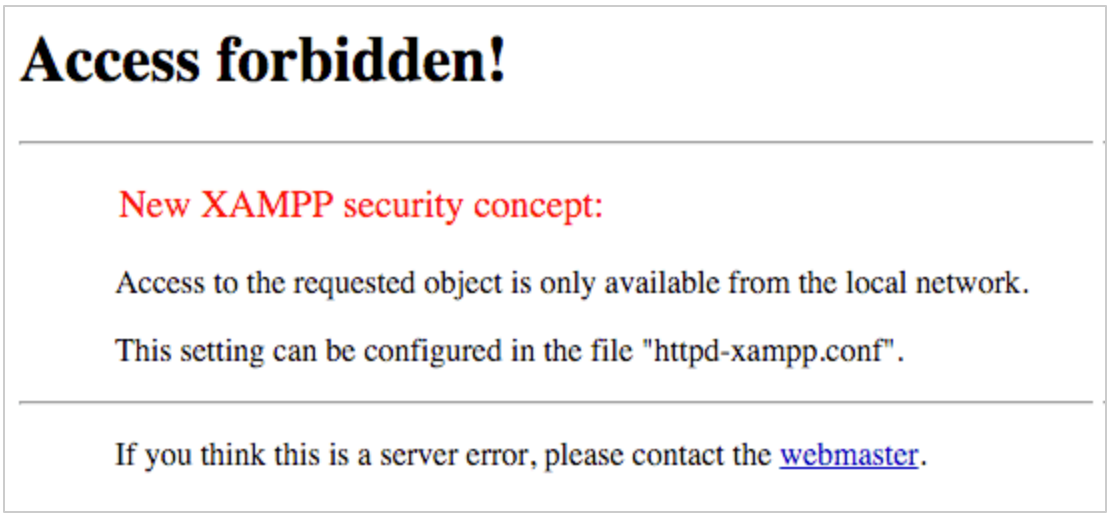
Do you want to continue? [Y/n]:**Y**

Run XAMPP

```
sudo /opt/lampp/lampp start
```

XAMPP Access Forbidden

Open your browser and access http://IP-ADDRESS/ you will find this Access forbidden screen.



XAMPP Configurations

Edit XAMPP configurations.

vi /opt/lampp/etc/extra/httpd-xampp.conf

```
<LocationMatch "^/(?i:(?:xampp|security|licenses|phpmyadmin|webalizer|server-status|server-info))">
Require local
ErrorDocument 403 /error/XAMPP_FORBIDDEN.html.var
</LocationMatch>

to

<LocationMatch "^/(?i:(?:xampp|security|licenses|phpmyadmin|webalizer|server-status|server-info))">
Order deny,allow
Allow from all
Allow from ::1 127.0.0.0/8 \
fc00::/7 10.0.0.0/8 172.16.0.0/12 192.168.0.0/16 \
fe80::/10 169.254.0.0/16
ErrorDocument 403 /error/XAMPP_FORBIDDEN.html.var
</LocationMatch>
```

Restart XAMPP

sudo /opt/lampp/lampp restart

Security Settings

sudo /opt/lampp/xampp security

XAMPP: Your XAMPP pages are NOT secured by a password.

XAMPP: Do you want to set a password? [yes]

XAMPP: Your XAMPP pages are NOT secured by a password.

XAMPP: Do you want to set a password? [yes] **no**

XAMPP: MySQL is accessable via network.

XAMPP: Normaly that's not recommended. Do you want me to turn it off? [yes] **yes**

XAMPP: Turned off.

XAMPP: Stopping MySQL...ok.

XAMPP: Starting MySQL...ok.

XAMPP: The MySQL/phpMyAdmin user pma has no password set!!!

XAMPP: Do you want to set a password? [yes] **yes**

XAMPP: Password:*******

XAMPP: Password (again):*******

XAMPP: Setting new MySQL pma password.

XAMPP: Setting phpMyAdmin's pma password to the new one.

XAMPP: MySQL has no root passwort set!!!

XAMPP: Do you want to set a password? [yes] **yes**

XAMPP: Write the password somewhere down to make sure you won't forget it!!!

XAMPP: Password:*******

XAMPP: Password (again):*******

XAMPP: Setting new MySQL root password.

XAMPP: Change phpMyAdmin's authentication method.

XAMPP: The FTP password for user 'daemon' is still set to 'xampp'.

XAMPP: Do you want to change the password? [yes] **no**

XAMPP: Done.

Once you are done setting up the XAMPP server, you can start the server by running the following command:

sudo /opt/lampp/lampp start

Now install the AWS php SDK on the Web instance for interacting with the AWS services.

All your source files can be saved at the path: /opt/lampp/htdocs. This will be your root folder for storing all the source files.

A file can be created by the following command:
vi filename.extension
Open four such files with each of the Web Tier file names with a php extension and copy the code from the submitted text files into these php files.

Once you connect to your instance via the public IP, just place the filename after the homepage to run the specific php code.

Ex: launcher.php can be run by directly going to the following address in your browser: yourInstanceIP/launcher.php

Setup of Ec2 App Tier:

1) First, we have used the AMI given in the project description to create an Ec2 instance.
2)Change access key and security key in constant.java file in AppTierRunning application submitted in canvas. After that create a runnable jar file of AppTierRunning application through selecting all the package required libraries into the jar.
3)Once the jar file has been created in a local machine, send this to the classifier folder in the Ec2 instance that we created in step1 using the below scp command.
scp -i key_pair.pem path_of_jar_file_in_local_machine ubuntu@ec2-public-dns:~/classifier/
4)Now, send autoStartAppTierRunning.sh shell script submitted in canvas into the Ubuntu folder of the Ec2 instance using the scp command.
scp -i key_pair.pem path_of_shell_file_in_local_machine ubuntu@ec2-public-dns:~
5) Now connect to the Ec2 instance using the below command.
ssh -i key_pair.pem ubuntu@ec2-public-dns

6)Now, make the jar and shell file executable using the below command in Ec2 Instance.
chmod +x filename
7)Now create a cron job to run the script automatically whenever an instance reboots or launches using this AMI.
Open crontab using the command “crontab -e” and write the following reboot command.
@reboot path_of_shell_script_in_ec2_instance.
Save the cron job and exit.
8) Now we have all the code setup required to run app tier functionality automatically. So create an AMI using this instance so that we can use this new AMI in web tier load balancer logic to launch up to 19 Ec2 App tier Instances whenever we need.