# Developing Solution for Image Processing

# (LAB-M10-01)

| Version Control | |
|---|---|
| Document | Developing Solution for Image Processing |
| Owner | Ahmad Majeed Zahoory |
| Version | 2.2 |
| Last Change | 24th May 2024 |
| Description of Change | Task steps updated |

**Lab duration**: **30 minutes**

**Lab scenario**

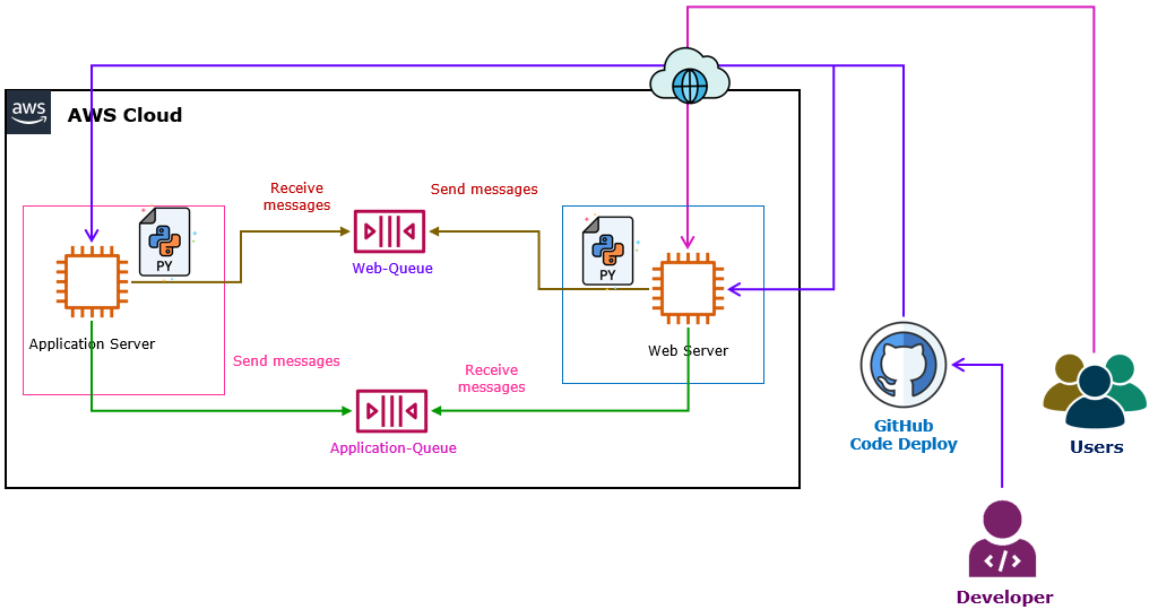
You're preparing to host a Frontend and backend application in AWS for Image processing. As a development group, your team has decided to make this application as asynchronous (loosely coupled).

You need to develop the solution to set up SQS to store the messages for image processing.

**Objectives**

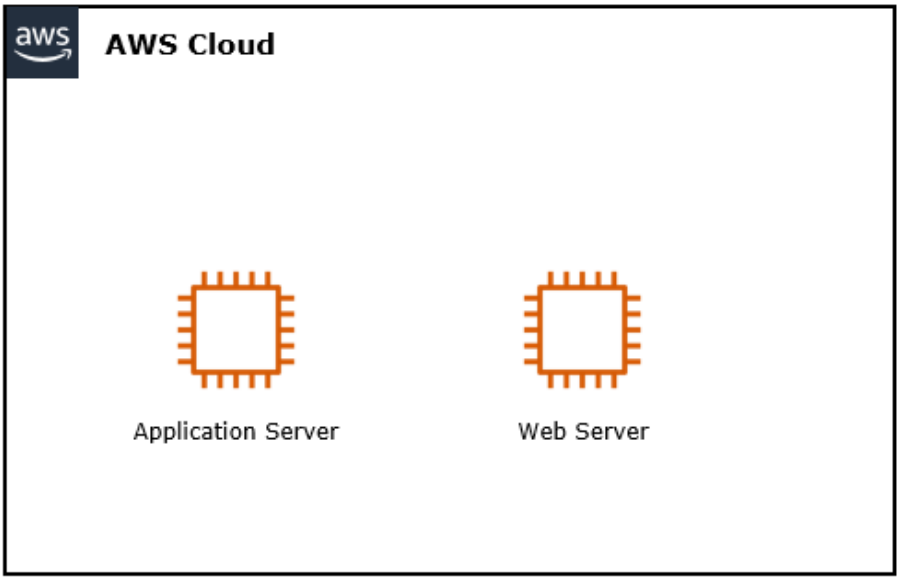
After you complete this lab, you will be able to:

- Set up an SQS queue.
- Build Web Server.
- Build Application Server.
- Send messages to an SQS queue.
- Receive messages from an SQS queue.



# Task 1: Deploy Web servers

In this task, you will create ec2 instances and IAM role with permission to manage SQS and attach the role to ec2 instances.




## Step 1: Create Web Server

1. In the **AWS Management Console**, on the **Services** menu, search and Select **CloudFormation**.
2. Choose the **YOUR ALLOCATED REGION** list to the right of your account information on the navigation bar.
3. Select **Create stack** and configure:
   a. In the **Create stack** page:
      i. **Prepare template**: Select **Template is ready**.
      ii. **Template source**: Select **Upload a template file**.
      iii. **Choose file**: Click on **Choose file**.
         a) **Navigate** and **select** the **LAB-SQS**.yaml file.

**Note**: **LAB-SQS**.yaml template is provided with the Lab manual.

**Note**: AWS template **performing** the **following** tasks:

1. **Creating Two Linux instances**.
2. **Creating t2.micro** instance.
3. **Set** the **ec2-user's Password**.
4. **Creating IAM role** with **SQS permission**.
5. **Attach** the **IAM role** to virtual machine.

iv. Select **Next**.

b. In the **Specify stack details** page:

i. **Stack name**: Write **LAB-SQS**.

**Note**: Leave the other details as default.

ii. Select **Next**.

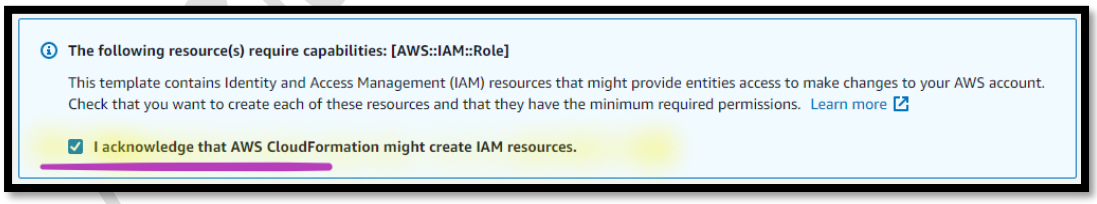
c. In the **Configure stack options** page:

**Note**: Leave all the details as default.

i. Select **Next**.

d. In the **Review LAB-SQS** page:

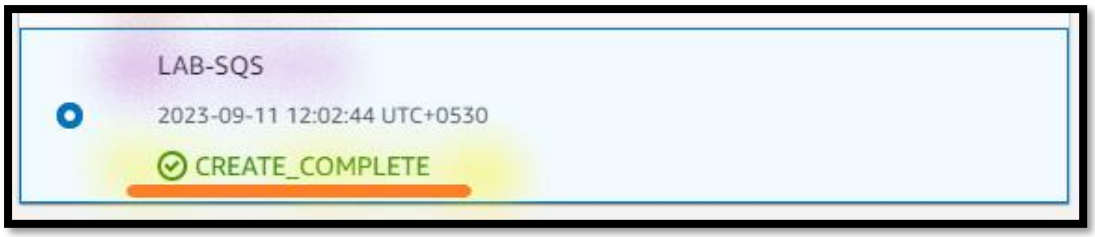
i. **I acknowledge that AWS CloudFormation might create IAM resources**: **Enable** the **Checkmark**.




ii. Select **Submit**.

**Note**: You can see the **Stack** status as **CREATE_IN_PROGRESS**.

**Note**: **Wait**, till you can see the **Stack** status as **CREATE_COMPLETE**. You can **Refresh** your screen




**Step 2: View the Output**

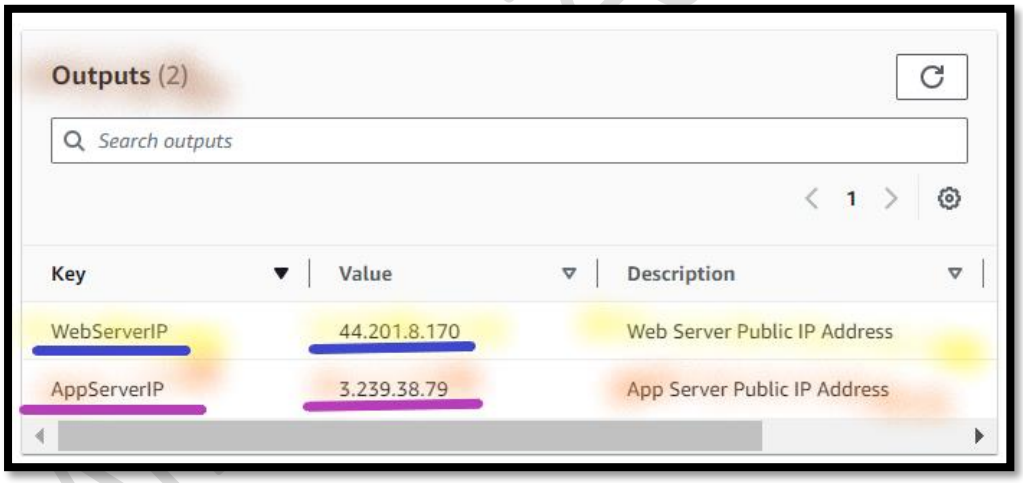
4. **From** the **LAB-SQS** **CloudFormation** console:

   a. Select **Outputs**.

**Note**: You can see the **resources details**.

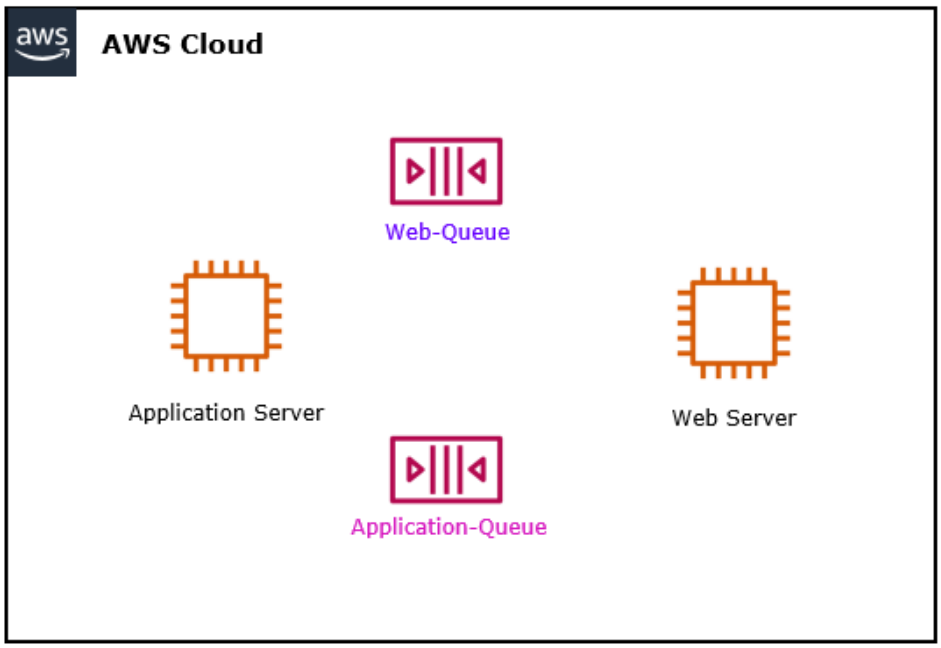
**Note**: **Copy** the **WebServer Public IP** in the **Notepad**.

**Note**: **Copy** the **AppServer Public IP** address in the **Notepad**.



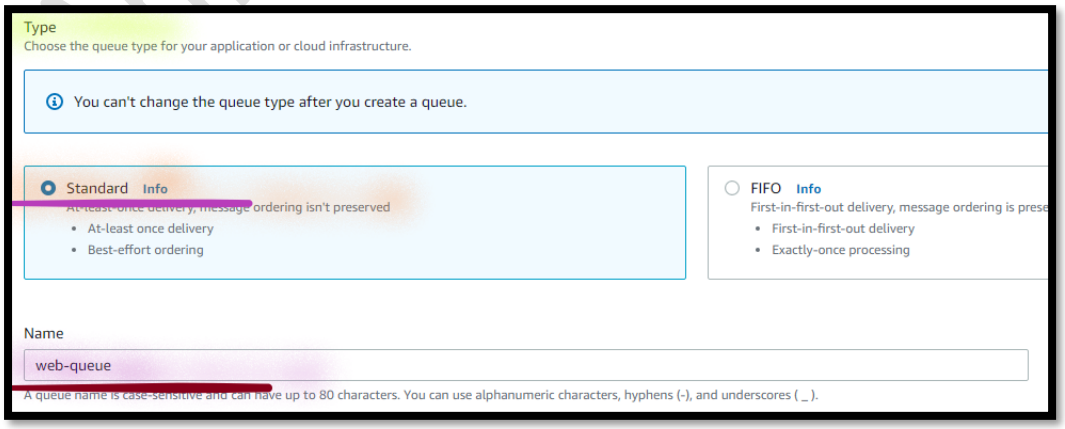
# Task 2: Create Amazon SQS Queues

In this task you create an SQS queue for Web and Application Servers.




## Step 1: Create the Web SQS Queue

5. In the AWS Management Console, on the **Services** menu, click **Simple Queue Service**.
6. Choose the **YOUR ALLOCATED REGION** list to the right of your account information on the navigation bar.
7. Click **Create queue** and configure:
    a. **Type**: Select **Standard**.
        i. **Name**: Write **web-queue**.



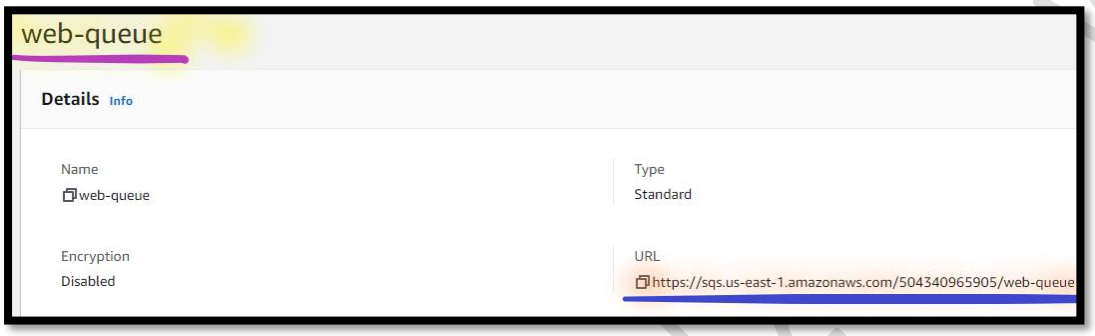
**Note**: Leave the other details as default.

b. Click **Create queue**.

## Copy the web-queue URL

8. **From** the **web-queue** queue:

   a. **Copy** the **web-queue** **URL** in the **Notepad**.




## Step 2: Create the Application SQS Queue

9. **From** the **Simple Queue Service** console.

   a. Select **Create queue** and configure:

      i. **Type**: Select **Standard**.

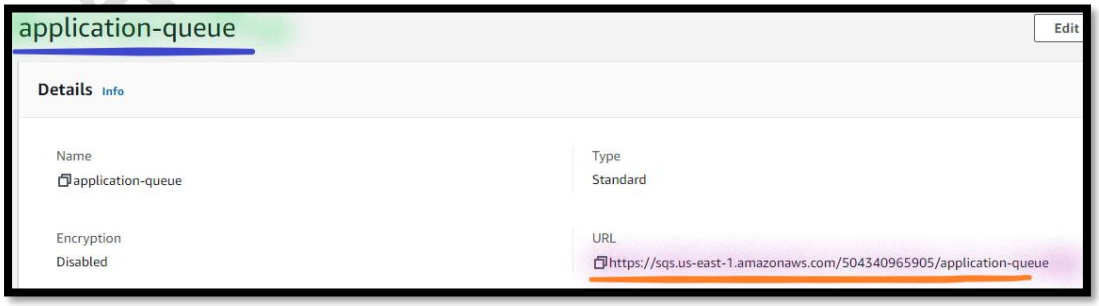
      ii. **Name**: Write **application-queue**.

         a) Click **Create queue**.

## Copy the Application-queue URL

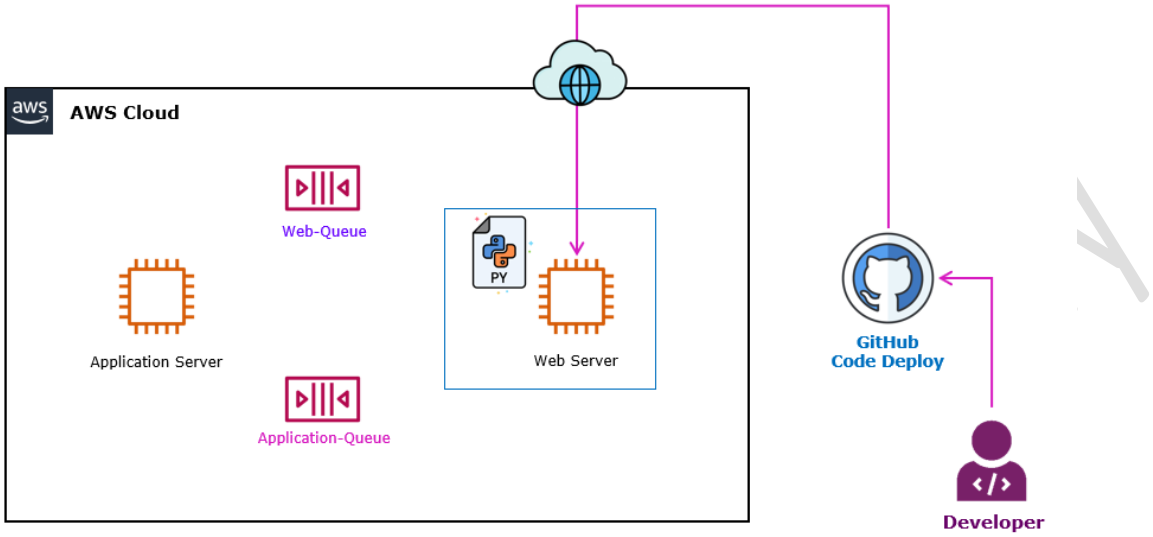
10. **From** the **application-queue** queue:

    a. **Copy** the **application-queue** **URL** in the **Notepad**.



# Task 3: Develop and Deploy the Application for Web Server

In this task, you will develop and deploy the web application for Web Server.




## Step 1: Develop the Code for Web Server

11. **Unzip** the **LAB-M10-01-Code-A.zip** (*Python code*).

**Note**: **lab-m10-01-code-a.zip** code file is available with the **Lab manual**.

**Info:** Code is **performing** the following:

**Part A** (***Upload the Image***)

1. **Upload** the **Raw image**.
2. **Send** the Raw image **message** to Image Input **Queue**.

**Part B** (***Retrive the Image***)

3. **Retrive** the Converted (thumbnail) image **message** from Image Output **Queue**.
4. **Display** the Converted (thumbnail) **image**.

a. Open the **app.py** in the **Notepad**.

i. **Replace** the **WEB-QUEUE-URL** with the **web-queue URL** (*which you have copied in the previous step*).

**Note**: **Don't replace** the starting and end quote [" "].

ii. **Replace** the **APPLICATION-QUEUE-URL** with the **application-queue URL** (*which you have copied in the previous step*).

**Note**: **Don't replace** the starting and end quote [" "].

iii. **Replace** the **REGION-IDENTIFIER** with the **region-identifier**.

**Note**: **Replace** the **region-identifier**.

**Refer** the **link** to know your **respective region region identifier** https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html

app - Notepad

File Edit Format View Help

```
import base64
from io import BytesIO

import boto3
from PIL import Image
from flask import Flask, render_template, request, redirect, make_response, Response

app = Flask(__name__)

# SQS queue to receive messages
img_input_queue_url = "https://sqs.us-east-1.amazonaws.com/504340965905/web-queue"
# SQS queue to send messages
img_output_queue_url = "https://sqs.us-east-1.amazonaws.com/504340965905/application-queue"

region_name = "us-east-1"
```

b. Save the **app.py** file:

i. Select **File**.

a) Select **Save**.

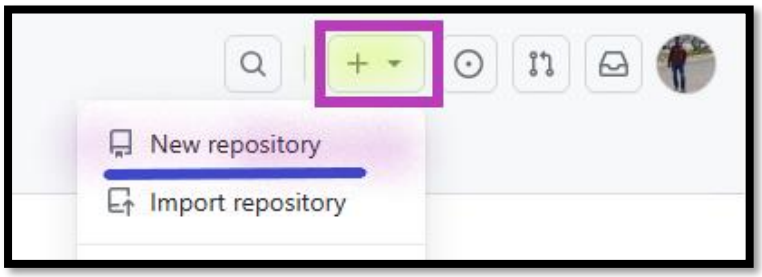
## Step 2: Create GitHub Repository

12. **Login** into your **GitHub account**.

13. To **Create repository**:

    a. Select **+** sign.

        i. Select **New repository**.




        a) **Repository name**: Write **webserver**.

        b) Select **Public**.

            1) Select **Create repository**.

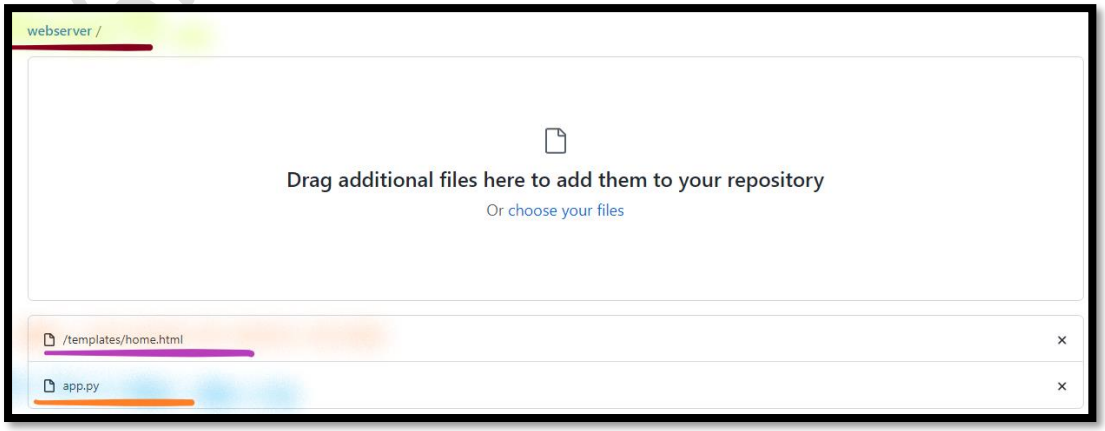
**Note**: You can see the **webserver repository** page.

    b. **From** the **webserver** repository:

        i. Select **uploading an existing file**.

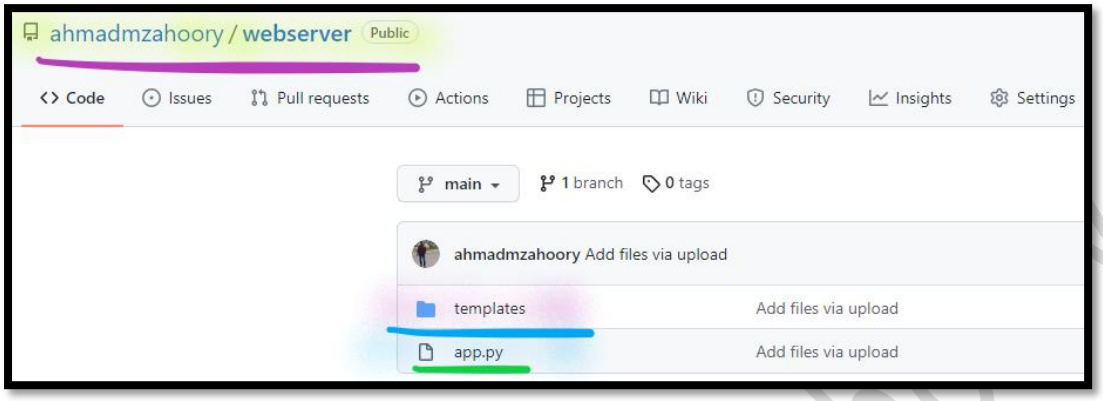
            a) **Drag** and **drop** the **template folder** and **app.py** file.

**Note**: **Upload** the **template folder not** the **.zip** file.



ii. Select **Commit Changes**.

**Note**: You can see the **templates folder** and **app.py** under **webserver repository**.

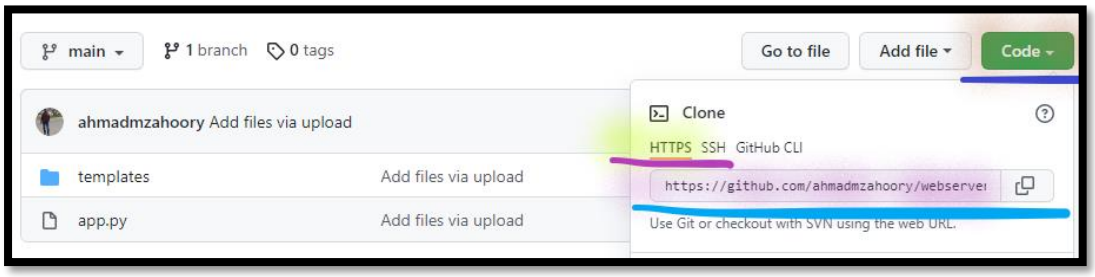



c. **From** the **webserver** repository:

i. Select **Code**.

a) Select **HTTPS**.

1) **Copy** the **Clone URL** in the **Notepad**.




## Step 3: Connect to Web Server

14. From the **Local Desktop/ Laptop** (*Windows desktop*), **Open** the **MobaXterm**.

15. From the **MobaXterm**.

a. Select **Session**.

b. Select **SSH**.

i. **Remote host**: Write **Public IP address** of the **Web Server**.

ii. **Specify username**: **Enable** the **Checkmark**.

a) **Specify username**: Write **ec2-user**.

iii. Select **Ok**.

iv. **Password**: Write the **lab-password** (*which you have set using the user data*).

**Note**: You can see the **Linux Console**.

## Step 4: Install Runtime Environment

16. **From** **Web Server** instance.

17. **From** the **Linux terminal**:

a. **Execute** the ***below command*** to **create** the **requirements.txt** file:

```
cat >> requirements.txt
```

**Info**: In Python requirement.txt file is a type of file that stores information about all the libraries, modules, and packages on which the project is dependent or requires to run.

i. In the **terminal**, type **boto3** and Press **Enter** key.

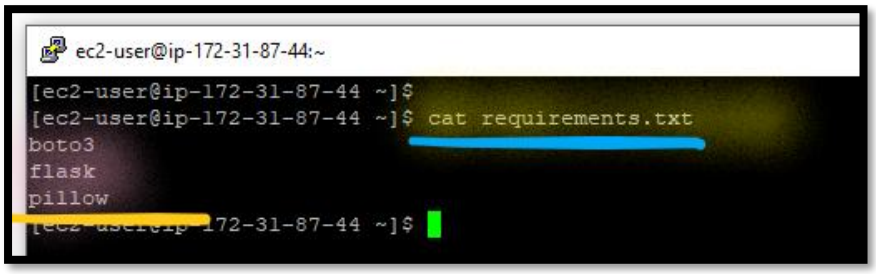
ii. In the **terminal**, type **flask** and Press **Enter** key.

iii. In the **terminal**, type **pillow** and Press **Enter** key.

a) Press **CTRL + C** (*to save the file*).

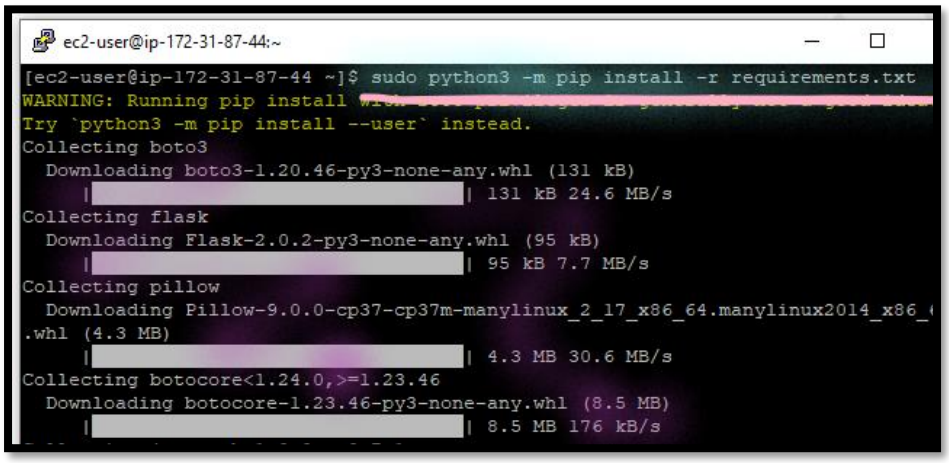
b. **Execute** the ***below command*** to **view** the **requirements.txt** file:

```
cat requirements.txt
```




c. **Execute** the ***below command*** to **install** the **packages** as per the **requirements.txt** file:

```
sudo python3 -m pip install -r requirements.txt
```




d. **Execute** the ***below command*** to **install** the **git package**:

```
sudo yum install -y git
```

e. **Execute** the ***below command*** to **clone** the **git** (*webserver repository*):

```
sudo git clone WEBSERVER-CLONE-URL
```

**Note:** **Replace** the **WEBSERVER-CLONE-URL** with the **webserver repository URL**, which you have copied in the previous step.

f. **Execute** the ***below command*** to **list** the **files and directories**:

```
ls -l
```

**Note:** You can see the **webserver folder**.

g. **Execute** the ***below command*** to **change** the **directory**:

```
cd webserver
```

h. **Execute** the ***below command*** to **list** the **files and directories**:

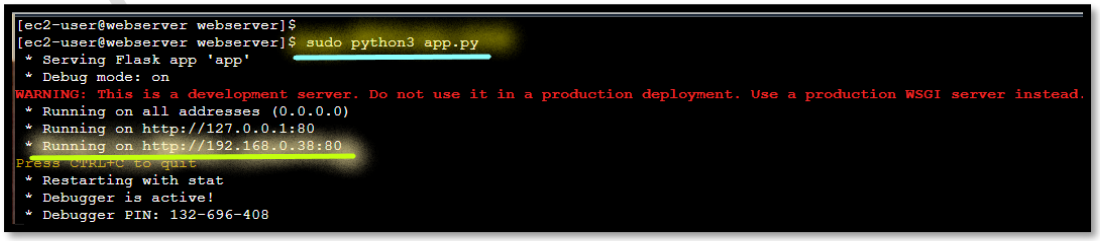
```
ls -l
```

**Note:** You can see the **templates** folder and **app.py** file.

i. **Execute** the ***below command*** to **execute** the **flask application**:

```
sudo python3 app.py
```

**Note:** You can see the **flask application** gets **started**.

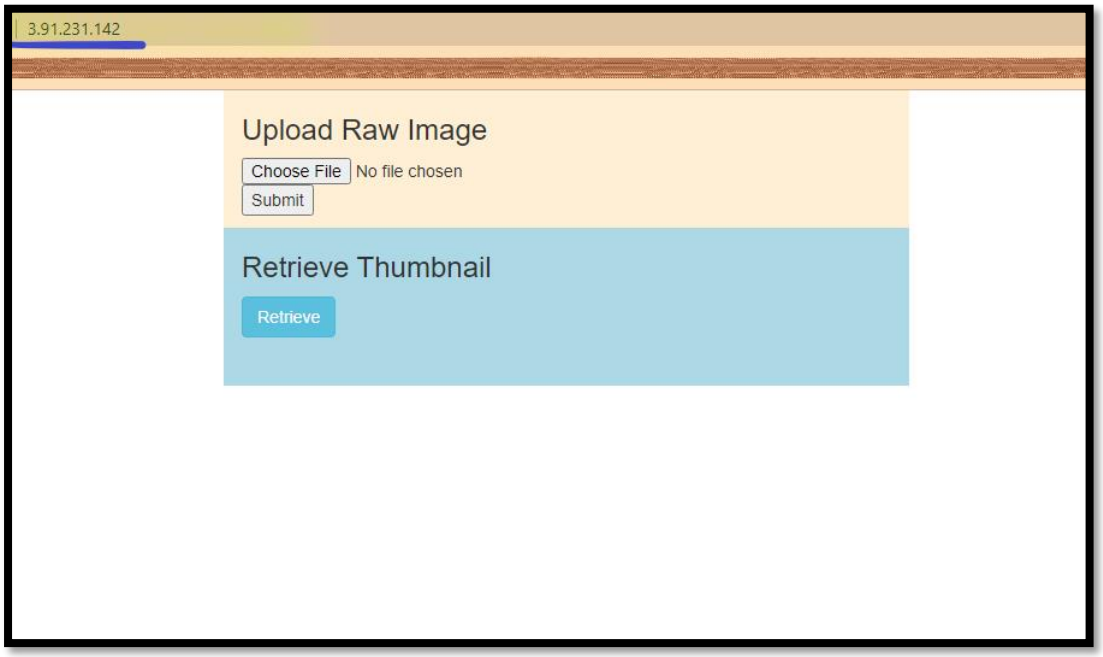


**Note**: **Go to the next task**. But **Don't close** the **Web server terminal**.

## Step 5: Access the Web Server

18. **From** the **Local Desktop/ Laptop** **Web browser**, type **Public IP Address** of **Web Server** and access your **website**.

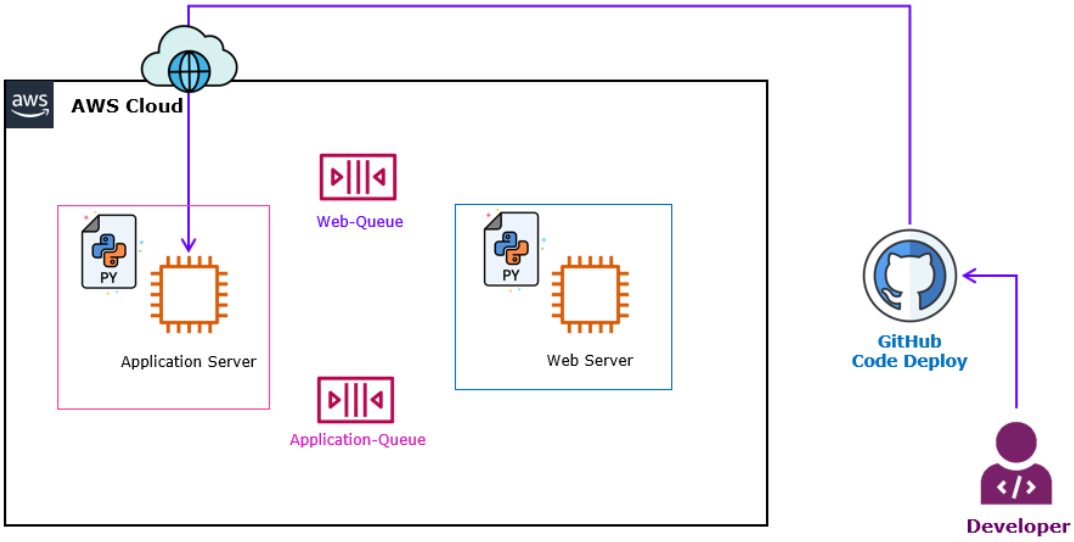
**Note:** You can see the below **web page**.




**Note**: **Go to the next task**. But **Don't close** the **Web page**.

# Task 4: Develop and Deploy the Application for Application Server

In this task, you will develop and deploy the application for Application Server.




## Step 1: Develop the Code for Application Server

19. **Unzip** the **LAB-M10-01-Code-B.zip** (*Python code*).

**Note**: **lab-m10-01-code-b.zip** code file is available with the **Lab manual**.

**Info:** Code is **performing** the following:

1. **Retrive** the Raw image **message** from Image Input **Queue**.
2. **Convert** the **Raw image** into **Thumbnail** (120x120).
3. **Send** the converted (thumbnail) image **message** to Image Output **Queue**.

a. Open the **app.py** in the **Notepad**.

i. **Replace** the **WEB-QUEUE-URL** with the **web-queue URL**, *which you have copied in the previous step*.

**Note**: **Don't replace** the starting and end quote [" "].

ii. **Replace** the **APPLICATION-QUEUE-URL** with the **application-queue URL**, *which you have copied in the previous step*.

**Note**: **Don't replace** the starting and end quote [" "].

iii. **Replace** the **REGION-IDENTIFIER** with the **region-identifier**.

**Note**: **Replace** the **region-identifier**.
**Refer** the **link** to know your **respective region region identifier**
https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html

b. Save the **app.py** file:

i. Select **File**.

a) Select **Save**.

## Step 2: Create GitHub Repository

20. **Login** into your **GitHub account**.

21. **To Create repository**.

a. Select **+** sign.

i. Select **New repository** and configure:

a) **Repository name**: Write **appserver**.

b) Select **Public**.

1) Select **Create repository**.

**Note**: You can see the **appserver repository** page.

b. **From** the **appserver** repository:

i. Select **uploading an existing file**.

a) Select **Choose your files**.

1) **Navigate** and select the **app.py** file.

I. Select **Commit Changes**.

**Note**: You can see the **app.py** under **appserver** **repository**.

c. **From** the **appserver** repository:

i. Select **Code**.

a) Select **HTTPS**.

I. **Copy** the **Clone URL** in the **Notepad**.

## Step 3: Connect to App Server

22. From the **Local Desktop/ Laptop** (Windows Desktop), **Open** the **MobaXterm**.

23. From the **MobaXterm**.

a. Select **Session**.

b. Select **SSH**.

i. **Remote host**: Write **Public IP address** of the **App Server**.

ii. **Specify username**: **Enable** the **Checkmark**.

a) **Specify username**: Write **ec2-user**.

iii. Select **Ok**.

iv. **Password**: Write the **lab-password** (*which you have set using the user data*).

**Note**: You can see the **Linux Console**.

## Step 4: Install Runtime Environment

24. **From** App Server instance.

25. **From** the Linux terminal:

    a. Execute the ***below command*** to create the **requirements.txt** file:

```
cat >> requirements.txt
```

> **Info**: In Python requirement.txt file is a type of file that stores information about all the libraries, modules, and packages on which the project is dependent or requires to run.

    i. In the **terminal**, type **boto3** and Press **Enter** key.

    ii. In the **terminal**, type **flask** and Press **Enter** key.

    iii. In the **terminal**, type **pillow** and Press **Enter** key.

        a) Press **CTRL + C** (*to save the file*).

    b. Execute the ***below command*** to view the **requirements.txt** file:

```
cat requirements.txt
```

    c. Execute the ***below command*** to install the **packages** as per the **requirements.txt** file:

```
sudo python3 -m pip install -r requirements.txt
```

    d. Execute the ***below command*** to install the **git package**:

```
sudo yum install -y git
```

e. **Execute** the ***below command*** to **clode** the **git** (***webserver***) repository:

```
sudo git clone APPSERVER-CLONE-URL
```

**Note: Replace** the **APPSERVER-CLONE-URL** with the **appserver** Github **URL**, which you have copied in the previous step.

f. **Execute** the ***below command*** to **list** the **files and directories**:

```
ls -l
```

**Note:** You can see the **appserver folder**.

g. **Execute** the ***below command*** to **change** the **directory**:

```
cd appserver
```

h. **Execute** the ***below command*** to **list** the **files and directories**:

```
ls -l
```

**Note:** You can see the **app.py** file.

i. **Execute** the ***below command*** to **execute** the **flask application**:

```
sudo python3 app.py
```

**Note:** You can see the **flask application** is **started**.

```
[ec2-user@appserver appserver]$
[ec2-user@appserver appserver]$ sudo python3 app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:80
 * Running on http://192.168.0.139:80
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 867-271-029
```

**Note**: **Go to the next task**. But **Don't close** the **App server terminal**.

## Step 5: Access the Application Server

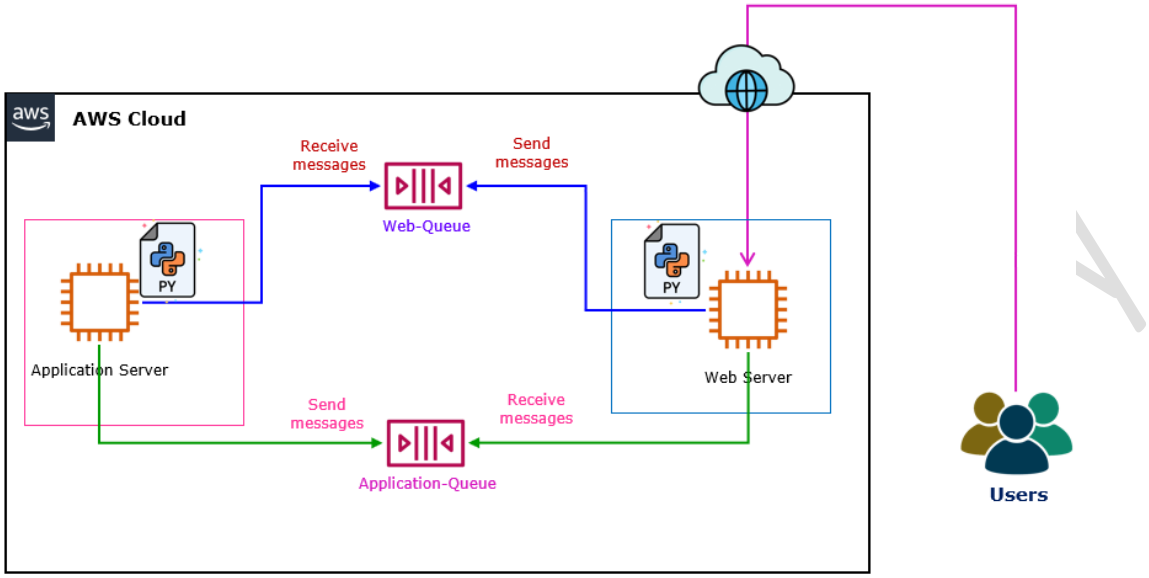
26. **From** the **Local Desktop/ Laptop** **Web browser**, type **Public IP Address** of **Application Server** and access your **website.**

**Note:** You can see the below **web page**.

**Note:** Currently **Application** is **Stopped**.

# Task 5: Validate the Deployment

In this task, you will test and validate the deployment.




## Step 1: Upload the Raw Image

27. **Return** to the **Web Server** **web page**.

    a. Select the **Choose file**.

        i. **Navigate** and select the **Image-01.jpg**.

**Note**: **Images** are **provided** with the **Lab manual**.

    a) Select the **Submit**.

## Step 2: View the Raw Image Details

28. **Return** to the **Web Server** **terminal**.

**Note:** You can see the **Uploaded Image name**.

**Note:** You can see the **Message Id** send to **Input Queue**.

```
Image-01.jpg
4096/cc4-138f-4a76-bd82-fd65bc5dd4fe
                                    "POST / HTTP/1.1" 302 -
183.82.153.227 - - [03/Feb/2022 11:53:32] "GET / HTTP/1.1" 200 -
```

## Step 3: View the Messages in web-queue

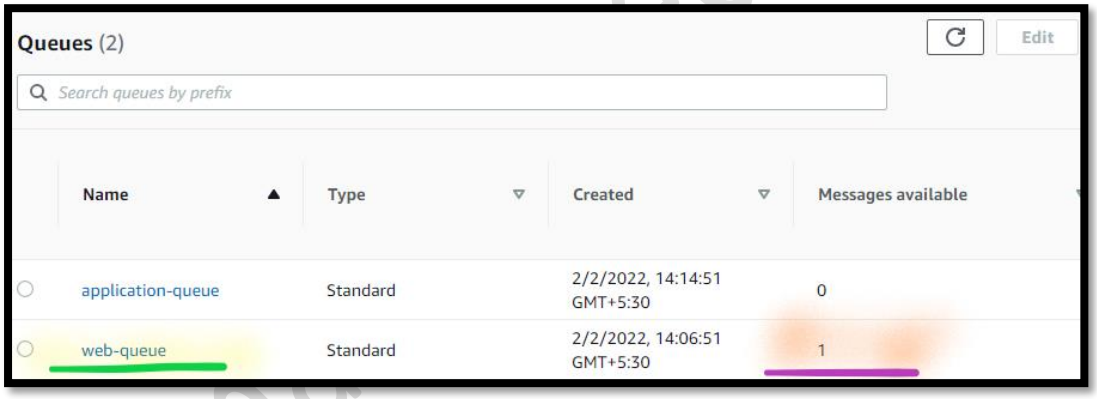
29. In the **AWS Management Console**, on the **Services** menu, click **Simple Queue Service**.

30. Choose the **YOUR ALLOCATED REGION** list to the right of your account information on the navigation bar.

31. Select the **Queues**.

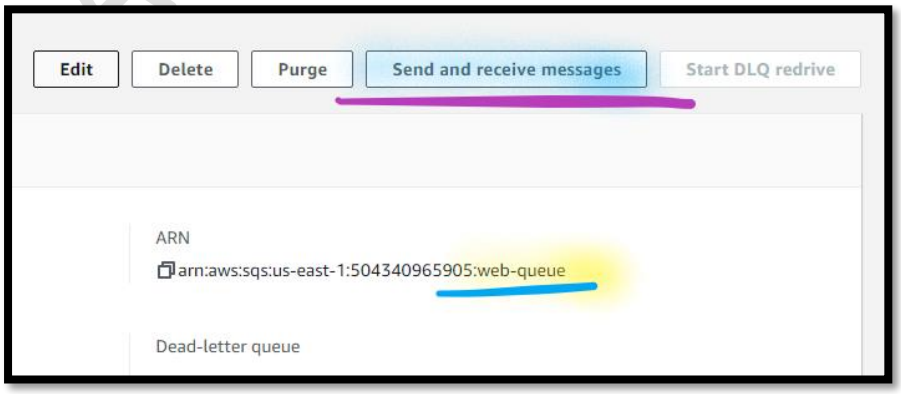
**Note**: You can see the **web-queue** and **application-queue** Queues.

**Note**: You can see the **Messages available** count as **1** under **web-queue** Queue.




a. **Open** the **web-queue**.

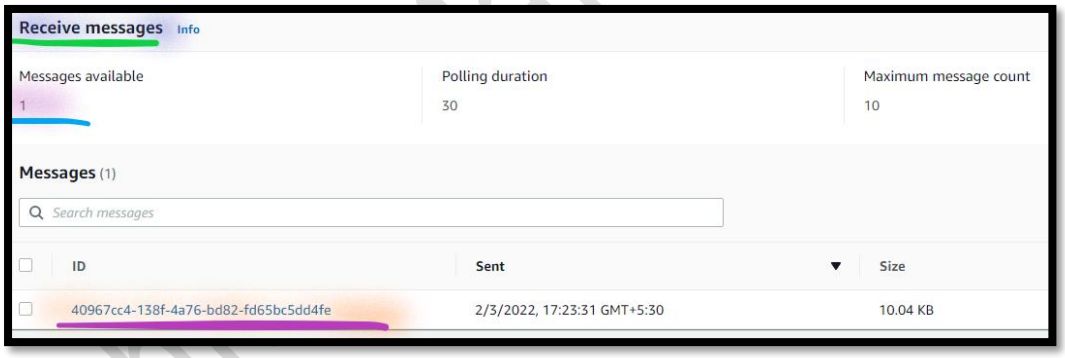
i. Select the **Send and receive messages**.



ii. Select the **Poll for messages**.


**Note**: **Wait**, unless you can see the **Polling progress** as **1 received**.


**Note**: You can see the **message Id**.




1) Open the **Message Id**.

I. **From** the **message**:

A. Select the **Details** to view the *message details*.


II. Select **Done**.

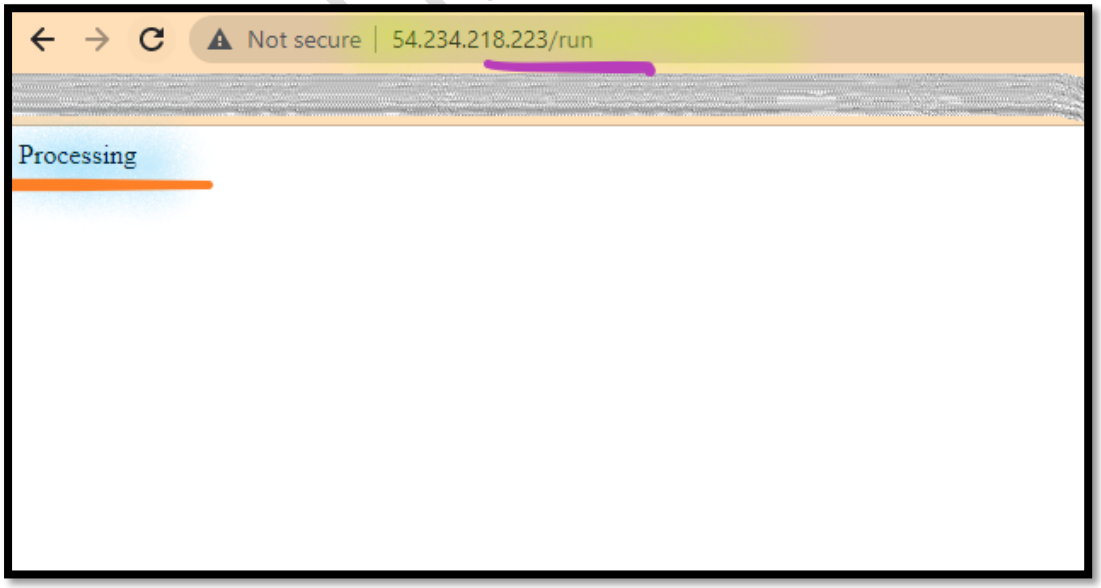
## Step 4: Processed the Raw Image

32. **Return** to the **Application Server web page**.

**Note:** Currently **Application** is **Stopped**.

a. **Update** the **URL** with **/run** to **Start** the **services**.

**Note:** Currently **Application** is **Started**.



33. **Return** to the **Application Server** **terminal**.

**Note:** You can see the **Message Id** send to **Output Queue** after processing.

```
72.21.217.143 - - [03/Feb/2022 12:05:06] "GET /__aes/proxy_health HTTP/1.1" 404
-
72.21.217.143 - - [03/Feb/2022 12:05:06] "GET /__aes/proxy_health HTTP/1.1" 404
-
183.82.153.227 - - [03/Feb/2022 12:05:16] "GET /run HTTP/1.1" 200 -
ed66eafe-7386-4acd-b9c4-f5b1f56ddf52
running...
running...
running...
```

## Step 5: View the Messages in application-queue

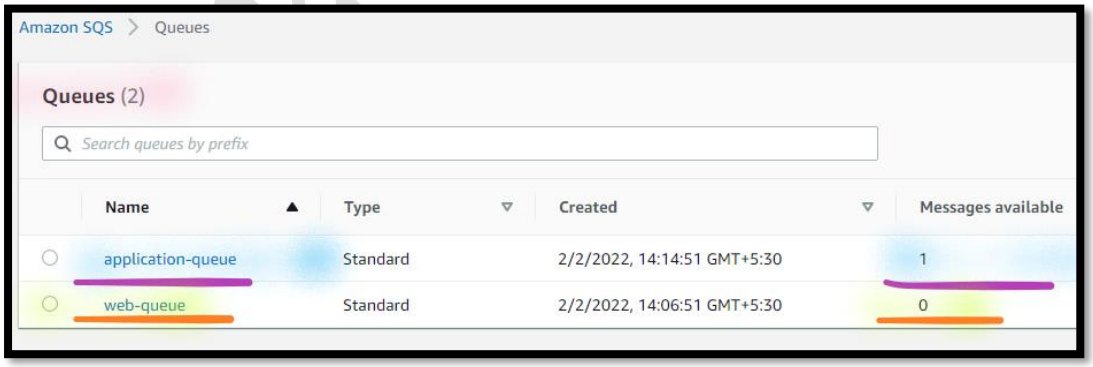
34. **Return** to the **Simple Queue Service**.

35. Select the **Queues**.

**Note**: You can see the **web-queue** and **application-queue** Queues.

**Note**: You can see the **Messages available** count as **0** under **web-queue** Queue, as **message** is processed and **deleted**.

**Note**: You can see the **Messages available** count as **1** under **application-queue** Queue.




a. **Open** the **application-queue**.

i. Select the **Send and receive messages**.

a) Select the **Poll for messages**.

**Note**: Wait, unless you can see the **Polling progress** as **1 received**.

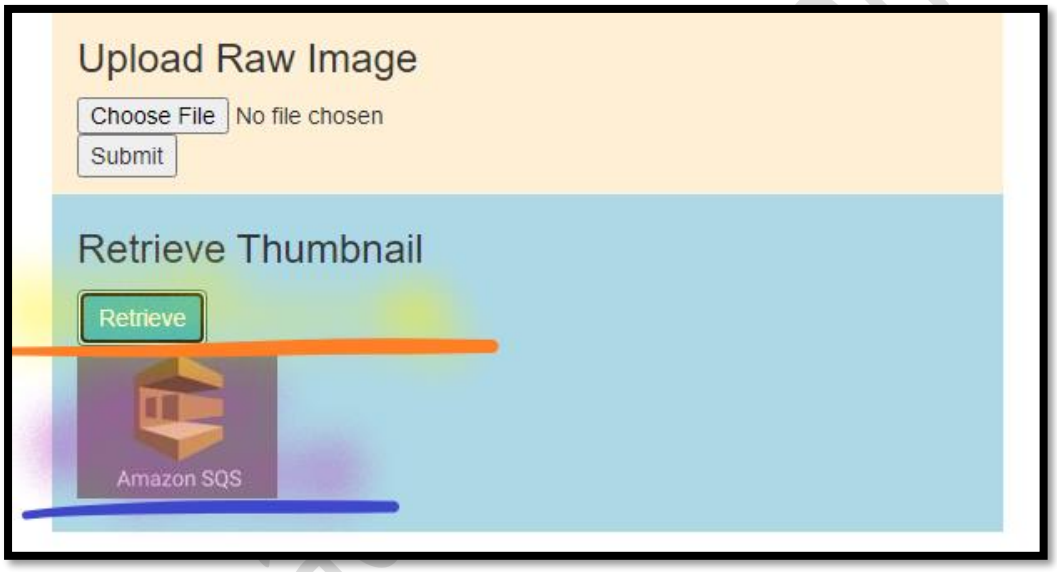
**Note**: You can see the **message Id**.

### Step 6: View the Converted Image

36. **Return** to the **Web Server** **web page**.

    a. Select the **Retrieve**.

**Note**: You can view the **Converted** (thumbnail) **Images**.

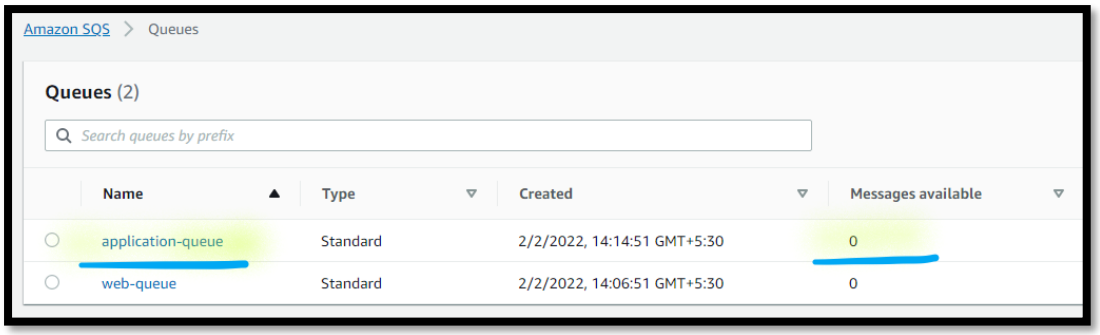



### Step 7: View the Messages in application-queue

37. **Return** to the **Simple Queue Service**.

**Note**: You can see the **web-queue** and **application-queue** Queues.

**Note**: You can see the **Messages available** count as **0** under **application-queue** Queue, as **message** is processed and **deleted**.




## Task 6: Clean up the Environment

### Step 1: Delete the SQS Queues

38. In the **AWS Management Console**, on the **Services** menu, search and select **Simple Queue Service**.

39. Choose the **YOUR ALLOCATED REGION** list to the right of your account information on the navigation bar.

40. Select the **Queues**.

    a. Select the **web-queue**.

        i. Select **Delete**.

            a) When you ***get prompt***, type **confirm**.

            b) Select **Delete**.

    b. Select the **application-queue**.

        i. Select **Delete**.

            a) When you ***get prompt***, type **confirm**.

            b) Select **Delete**.

### Step 2: Delete the Stack

41. In the **AWS Management Console**, on the **Services** menu, search and select **CloudFormation**.

42. Choose the **YOUR ALLOCATED REGION** list to the right of your account information on the navigation bar.

43. Select **Stack**.

a. Select **LAB-SQS**.

i. Select **Delete**.

a) Select **Delete**.

**Note**: If **delete stack gets failed**. Select **Retry delete** and select **Force delete this entire stack**.

### Step 3: Delete S3 Buckets

44. In the **AWS Management Console**, on the **Services** menu, search and select **S3**.

45. Choose the **YOUR ALLOCATED REGION** list to the right of your account information on the navigation bar.

46. Select **Buckets**.

a. Select the **cf-templates.....** bucket.

i. Select **Empty**.

ii. Once **prompted**, type **permanently delete**.

a) Select **Empty**.

b) Select **Exit**.

47. Select **cf-templates.....** bucket.

i. Select **Delete**.

ii. Once **prompted**, type **bucket name**.

a) Select **Delete bucket**.