Developer Documentation

1. File Architecture

1.1 UserUI

```
/Desktop/web-development-Cloud-Computing/webapp
|---      app
|                 |---static
|                 |---templates              # folder for storing all html files
|                         |---welcome.html     # welcome page of “Image Bay”
|                         |---signup.html        # where new users sign up
|                         |---signin.html        # old users sign in to personal page
|                 |---homepage.html   # personal page with upload function
|                         |---imagedetail.html   # where displays full-size image
|                 |---testupload.html    # assists TA for testing functions
|                 |---__init__.py              #Initialization
|                 |---config.py            # setting for connecting to the database
|                 |---Welcome.py          # Welcome page of “Image Bay”
|                 |---SignUp.py            # sign up for new users
|                 |---SingIn.py             # sign in for old users
|                 |---ImageProcess.py    # implement 3 image transformations
|                 |---ImgDetail.py        # show original and transformed images
|                 |---HomePage.py       # personal page for each user
|                 |---sql.py                # connect to database
|                 |---TestUpload.py       # realize test upload for TA
|---      run.py
|--- webapp.sql   # the schema of database
```

1.2 ManagerUI

```
/Desktop/Web-application_ImageBay/manager
|---      app
|                 |---static
                                |---flot              # file for java scrpit
|                 |---templates              # folder for storing all html files
|                         |---base.html            # base html
|                         |---manager.html       # main page for manager UI
|                         |---view.html             # view the details of a instance
|                 |---__init__.py              # Initialization
|                 |---config.py            # setting for connecting to the database
|                 |---AWS_config.py          # setting of AWS
|                 |---check_func.py          # to check CPU every few seconds
|                 |---ec2_Process.py         # to create or delete instances
|                 |---sql_del.py             # empty database and S3 bucket
|                 |---sql.py                    # connect to database
|---      run.py
|---webapp.sql                  # the schema of database
```

2. General Architecture

2.1 Browser

This web application is developed and tested using Mozilla Firefox web browser. The web browser in this architecture serves as the graphical user interface to general users and as the client to the Flask web development server. The main functionality of the web browser is to collect information from users in the format of forms and to display the transformed images as well as any additional information such as errors and warnings from the backend Flask web server to the users in a user-friendly environment. The HTTP post and put request methods will be issued by the web browsers to push collected information to the Flask web server and to retrieve information from it.

2.2 Flask Web Server

The web server used for this web application is named as Flask. Flask is a micro web framework written in Python which based on Werkzeug toolkit and Jinha2 template engine. The Flask web server in this architecture plays the backend web server role to process the HTTP request from the web browser and the client role to send requests to the MySQL database server for storing user specific data.

2.2.1 HTML

All html files are stored under the /app/templates directory.

2.2.2 Image

As each user successfully creates its user account in the database, the original image and its transformation will be temporarily stored in /app/static/. After the images are uploaded to S3 under the directory named by the username, the local images will be deleted.

2.2.3 Python files

All python files which define the backend logic and to initiate the web server will be stored under the /app/ directory.

2.2.4 MySQL connector

To save collected user specific information and to authenticate users once they finished creating user accounts, MySQL connector is imported to Flask in sql.py. Three functions are defined in this file. The connect_to_database() function is used to initiate the connection to MySQL database. The get_db() function is used to check the status of the mysql connection and will invoke the connect_to_database() function if there is no database exists. The close_db() function is used to close the connection to MySQL database. The parameter used to access the backend MySQL database is defined in config.py file

2.2.5 Cryptography

To store the hash of password concatenated with a per-user salt value, hashlib module is imported to Flask in both SignIn.py and SignUp.py files. Hashed password will be generated

and saved to the backend MySQL database once a user finished creating his or her user account through signup form. When user enters password through login page, a new hashed password will be generated by calling hashlib.sha256() together with salt value. Then this new generated password will be used to compare with the hashed password saved in the MySQL database. User is allowed to access his or her personal homepage only when the comparison result is true.

2.2.6 Imagemagic

To allow image transformation, a ctype-based simple ImageMagick binding for python named as Wand is imported to Flask in ImageProcessing.py file. Three types of image transformation (enhancing saturation, hue transfer and decreasing brightness) are defined in ImageTransSave() function. All the transformed images and the original images could be found in S3 bucket called 'ece1779a2john'.

2.3 MySQL Server

MySQL database in this architecture has one table "userInfo" which is used to store the information submitted by users during signup phase and to store the name and encrypted code each user for authentication. It also has the second table "user2Images" which is used to store the information of images uploaded by users for file tracking purpose. The details of the table are below:

userInfo (<u>userName</u>, userEmail, userPwd, userSalt)
The primary key of this table is userName. This table is used to save user name, user email and their encrypted password and per-user salt.

user2Images (<u>userName, Thumbnail</u>, original, trans_a, trans_b, trans_c)
The composite primary key of this table is (username, Thumbnail). This table is used to save addresses in S3 of each image, including original image, thumbnail and three transformations images. Each image record is be related to a user name.

Eg.

1. userInfo:

#	userName	userEmail	userPwd	userSalt
1	hanwen	hw@163.com	7684eaed6d7b576b59b0057fa…	Fv+jh4kGcvcSnuccq7wdY6A6fY…
2	zhangge	zg@qq.com	96bb1d87d562a1a9aec6888f2…	AQaXXctVKaLVHg3hMZetomNB…

Figure 1: Example of userInfo table in database

2. user2Images:

1	aaa	https://s3.amazonaws.com/ece1…	https://s3.amazonaws.com/ece1…	https://s3.amazona
2	aaa	https://s3.amazonaws.com/ece1…	https://s3.amazonaws.com/ece1…	https://s3.amazona
3	aaa	https://s3.amazonaws.com/ece1…	https://s3.amazonaws.com/ece1…	https://s3.amazona
4	aaa	https://s3.amazonaws.com/ece1…	https://s3.amazonaws.com/ece1…	https://s3.amazona

Figure 2: Example of user2Images table in database

2.4 Amazon S3

Amazon S3 in this architecture is used to store all the images and their transformations by users from UserUI. Within the "ece1779a2john" bucket, each registered user will have a directory named by userName and the images submitted by users will be saved to this user-specific directory, as well as the thumbnail and transformations of images. Once all the data is deleted from the manager UI, all the images stored in S3 as well as user-specific directories will be removed.

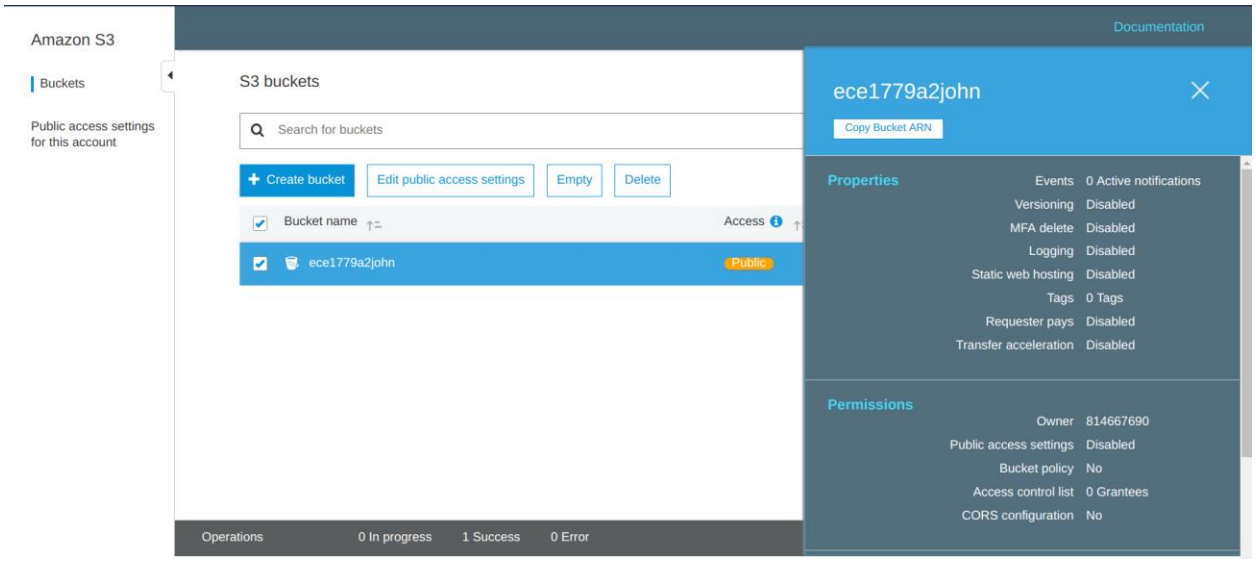

Figure 3: Amazon S3 bucket

2.4 Manager UI

Manager UI in this architecture is used to monitor the ec2 instances and the s3 instance in the backend. It lists all the ec2 instances running on Amazon AWS and give options to allow manager to set CPU threshold and to set ratio for expanding and shrinking the worker pool. It also allows manager to manually create and destroy ec2 instance. Manager is also given the option to view the details of each ec2 running instance such as its CPU and network IO usage. Manager can also choose to clear all the data stored in database and S3 right now.

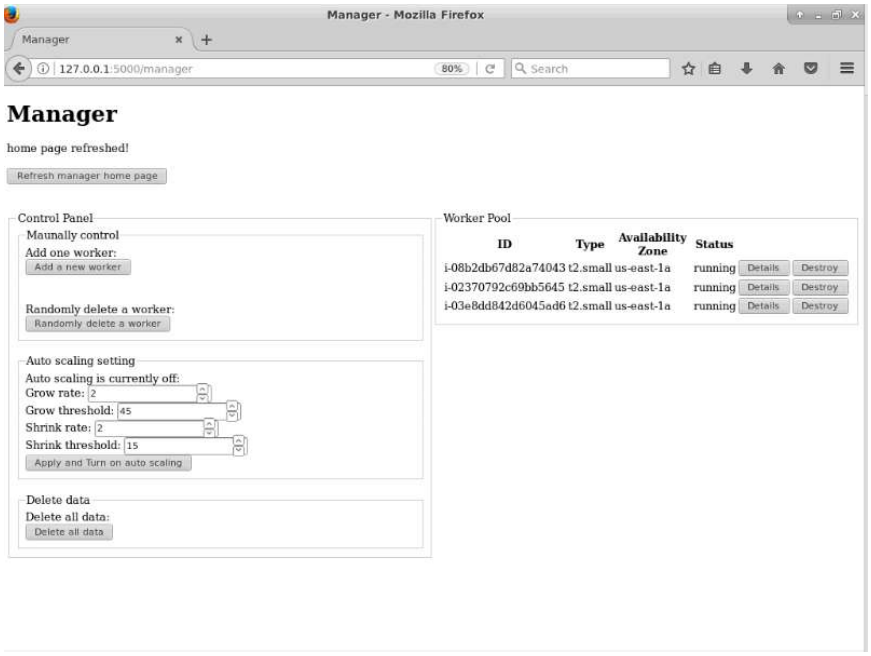

Figure 4: Manager UI interface

2.5 Elastic Load Balancer

The elastic load balancer configured here provides basic load balancing across multiple Amazon EC2 instances and operate at both the request level and connection level. The ELB load balancer will distribute all Amazon EC2 instances in the region of us-east-1. When a new instance created by Manager UI, this new instance will be automatically subscribed to this elastic load balancer.

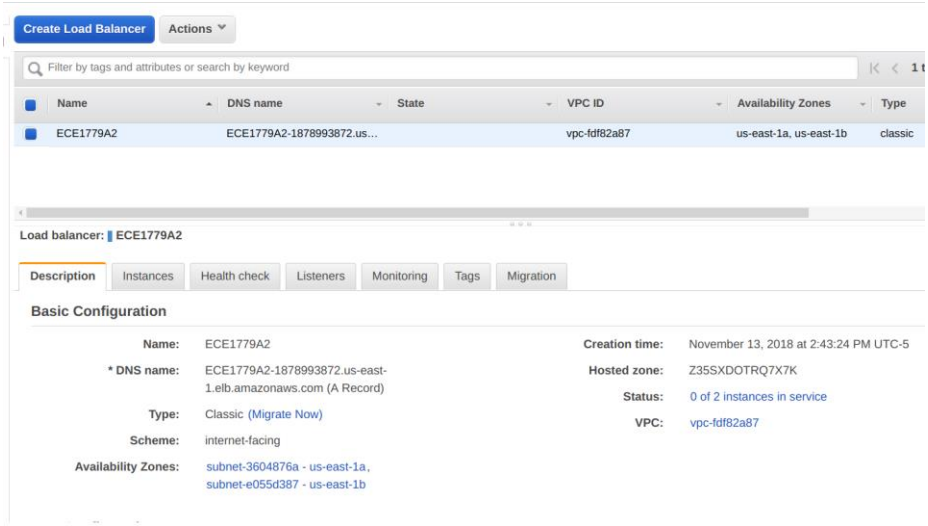

Figure 5: Elastic Load Balancer configuration