Assignment 2: Image Gallery
ECE1779

2020.03.20

Group member:

- Ruixin Huang 1001781565
- Lichen Liu 1001721498
- Baiwu Zhang 1001127540

1 Introduction

In Assignment 2 for ECE1779: Cloud Computing, the team is asked to scale up the image gallery website built in Assignment 1. The website server needs to be re-architected in order to effectively handle a variety of loads of incoming requests. This requires the server to be scalable and dynamic in such a way that not only all computing resources can be fully utilized, but also in a cost-efficient and energy-efficient manner. The website will be hosted on more EC2 instances, using Elastic Load Balancer, and utilize Relational Database Service provided by AWS. In addition to the image gallery app, the team also needs to build a manager application to actively monitor the utilization status of all instances, and provide a friendly interface for scaling up or down.

This document provides a comprehensive review of both websites: user application and manager application. Section 2 will introduce the usage and navigation of both applications. Section 3 will illustrate how each element of the user application is adapted to be scalable. Section 4 will describe the design of the manager application, including a load balancer and an auto scaler. In section 5, the result obtained from testing the auto-scaling functionality will be explained.

To start the server, please launch an EC2 instance i-01247d748caf64ae6 (a2-manager), created under AWS Educate Account with "baiwu.zhang@mail.utoronto.ca" as the email and "ece1779ins!" as the password. Once the instance is started, the manager application and other instances will automatically start as well. There is no need to ssh into any of the instances. The manager application can be accessed at http://52.21.3.87:5000. The main entry point for the user application is at http://ece1779-user-app-load-balancer-679239310.us-east-1.elb.amazonaws.com.

2 Usage

2.1 User Application

The user application provides identical functionalities and user interfaces as in Assignment 1. All changes on the user side are hidden from users. Therefore the users will have a seamless experience during the upgrading process.

Detailed descriptions on how to use the user application are provided in Section 2 of the Assignment 1 report.

2.2 Manager Application

The manager application is hosted on a publicly accessible IP address. Only authorized users are allowed for accessing the manager application website. The authentication process consists of user identification and verification. Website administrators need to register their email addresses at the backend of the system (hard-coded in the manager application program code) first. Upon connecting to the manager application, the user needs to enter his email addresses. Then the system will send a one-time password to the user's email address. If the one-time password is successfully entered into the authentication page within a short expiry period, the user is identified and authenticated. For marking purposes, emails for TA (brianr@cs.toronto.edu and shawnliok@gmail.com) are authorized.

Once the manager webpage is launched, the server administrators will see a clear view of all actions and data on the entire user application service. On the left is a list of charts showing historical user application server status, which can be switched to display either CPU utilization percentage or incoming HTTP request rate for each EC2 instance. On the right is the configuration panel, including manual scaling control, user application-related control, and auto-scaling config control. The manager page is refreshed every 15 seconds, which can be disabled at the top of the website. The top right corner also displays a live status of the main DNS name of the user application website.

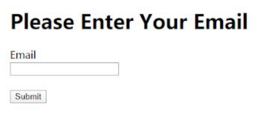
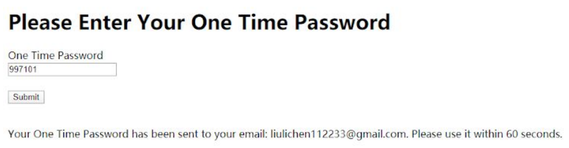


Figure 1. Screenshot of Email-address Login Authentication Page on Manager Application



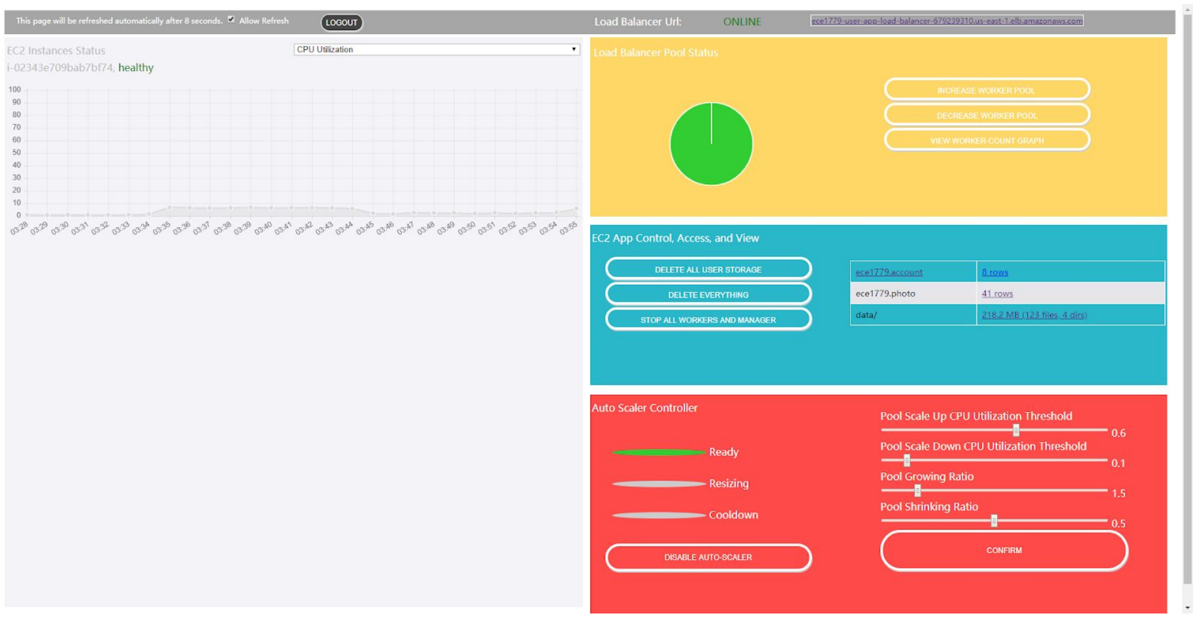
Please Enter Your One Time Password

One Time Password: 997101

Submit

Your One Time Password has been sent to your email: liulichen112233@gmail.com. Please use it within 60 seconds.

Figure 2. Screenshot of One-Time-Password Login Authentication Page on Manager Application



This page will be refreshed automatically after 8 seconds. ✓ Allow Refresh

Load Balancer Url: ONLINE ece1779-user-app-load-balancer-679239310.us-east-1.elb.amazonaws.com

EC2 Instances Status

i-02343e709bab7bf74, healthy

CPU Utilization

Load Balancer Pool Status

EC2 App Control, Access, and View

ece1779.account	8 rows
ece1779.photo	41 rows
data/	218.2 MB (123 files, 4 dirs)

Auto Scaler Controller

Ready

Resizing

Cooldown

Pool Scale Up CPU Utilization Threshold: 0.6

Pool Scale Down CPU Utilization Threshold: 0.1

Pool Growing Ratio: 1.5

Pool Shrinking Ratio: 0.5

Figure 3. Screenshot of Manager Application.

2.2.1 Worker Charts

Worker charts display real-time information about instances in the worker pool. Each display chart corresponds to a running instance with the user application and registered with the Elastic Load Balancer. The charts can be switched to show either CPU utilization percentage or incoming HTTP request rate for each EC2 instance. CPU utilization information is obtained from AWS CloudWatch Application Programming Interface (API) with a granularity of 1 data point per minute. Data on incoming HTTP request rate, on the other hand, is obtained from a custom metric, published by each user application server. For each request, a user application EC2 instance server receives, it will publish to a CloudWatch metrics named “HttpRequestCount” with its own instance id. The total published number during each minute is aggregated and plotted in the chart.

For every chart displayed, the associated instance id and its health check status from Elastic Load Balancer are also shown.

2.2.2 Load Balancer Manual Control

Load Balancer manual control is in the top row of the right column of the manager page. It contains a pie chart and three buttons. The pie chart displays the number of instances registered with the load balancer at that moment, with colour representing their health check status. By hovering a mouse over each section of the pie chart, the exact number for that section will also be displayed.

The three buttons on the right will allow manually modifying the number of instances registered with the load balancer. Either increase, decrease or view the history count. With each click on the button, the manager application will perform such actions using AWS APIs.

2.2.3 Worker Count Graph

Clicking on the ‘View Worker Count Graph’ will direct the user to a graph that shows the total number of healthy targets in the worker pool for the past 30 minutes.

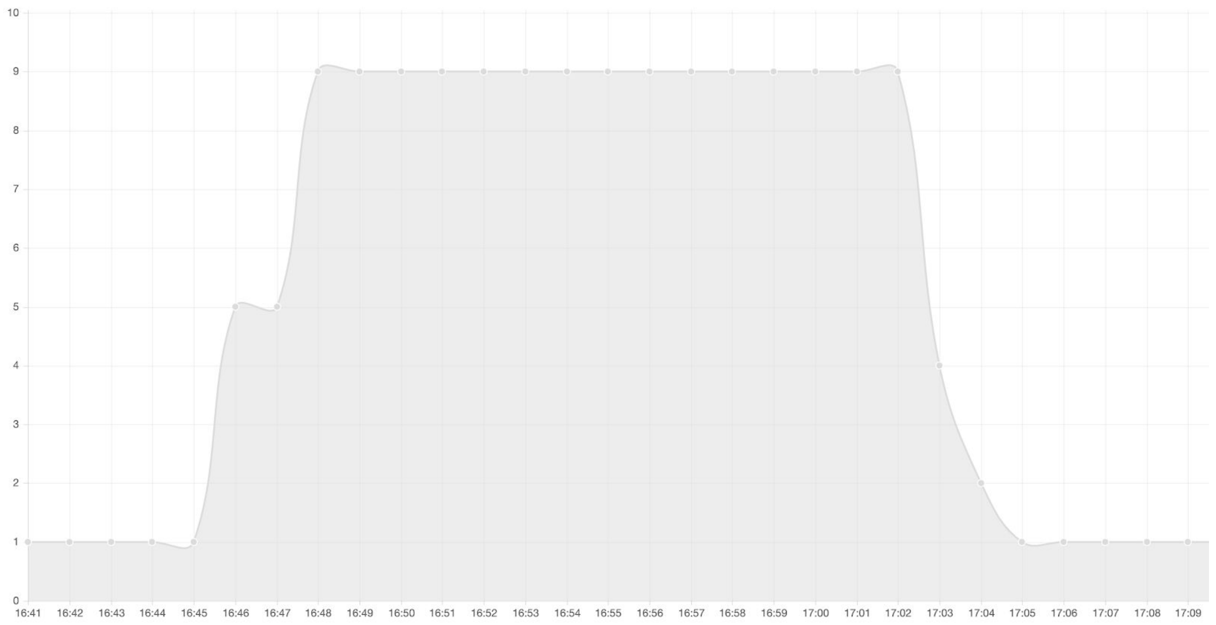


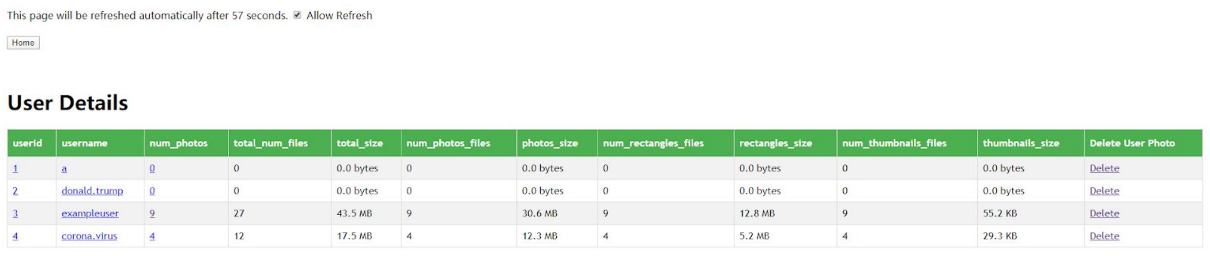
Figure 4. Screenshot of worker count graph.

2.2.4 Instance/Database Control

The second row on the right contains controls related to instances and databases. The first button “DELETE ALL USER STORAGE” clears out all raw and processed photos uploaded by all users. The second button “DELETE EVERYTHING”, as the name suggests, deletes all photos and also user accounts. The third button “STOP ALL WORKERS AND MANAGER” stops all EC2 instances running user application server and manager application server.

To the right of the three buttons, there is a table showing the latest storage and database statistics. It prints out the number of currently registered users, uploaded photos, and the S3 Cloud Object Storage usages. Moreover, by clicking any of these figures, users can check user details, account table, photo table, and S3 storage in detail.

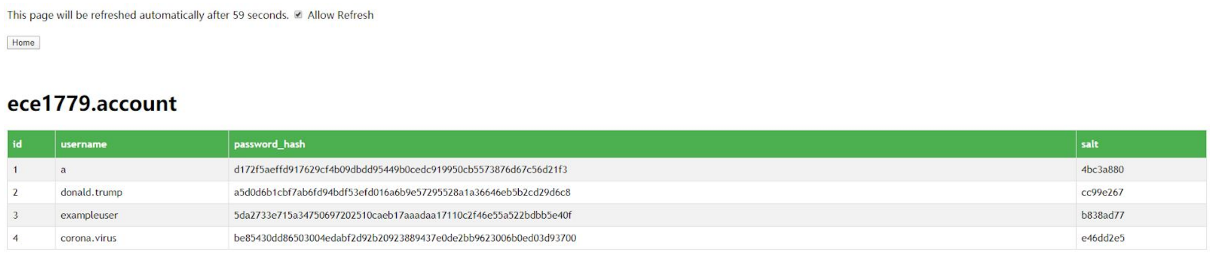
The account table and photo table print out the corresponding content stored inside the RDS MySQL database. The S3 storage page provides a more direct way to allow the administrator to go into the S3 storage space and check the content, for example, the number and size of files. The user details page is a combination of both database and S3 storage info: it prints out the number of photos, the number of files and file sizes for each user. Furthermore, each entry has clickable items that the administrator can click and check in depth. It also comes with an interface to delete all photos submitted by a particular user.


This page will be refreshed automatically after 57 seconds. ☑ Allow Refresh

User Details

userid	username	num_photos	total_num_files	total_size	num_photos_files	photos_size	num_rectangles_files	rectangles_size	num_thumbnails_files	thumbnails_size	Delete User Photo
1	a	0	0	0.0 bytes	0	0.0 bytes	0	0.0 bytes	0	0.0 bytes	Delete
2	donald.trump	0	0	0.0 bytes	0	0.0 bytes	0	0.0 bytes	0	0.0 bytes	Delete
3	exampleuser	9	27	43.5 MB	9	30.6 MB	9	12.8 MB	9	55.2 KB	Delete
4	corona.virus	4	12	17.5 MB	4	12.3 MB	4	5.2 MB	4	29.3 KB	Delete

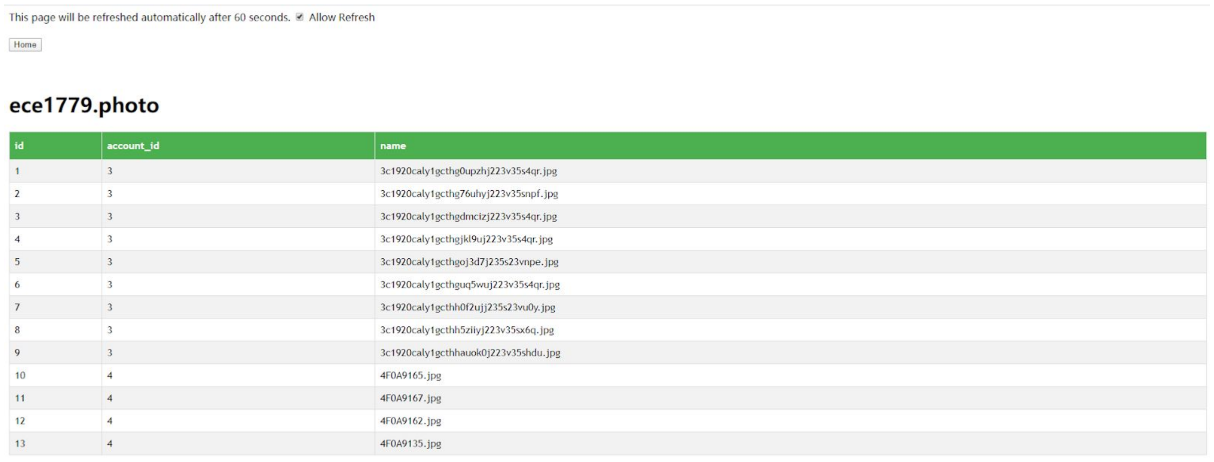
Figure 5. Screenshot of User Details Page on Manager Application


This page will be refreshed automatically after 59 seconds. ☑ Allow Refresh

ece1779.account

id	username	password_hash	salt
1	a	d172f5aeffd917629cf4b09dbdd95449b0cedc919950cb5573876d67c56d21f3	4bc3a880
2	donald.trump	a5d0d6b1cbf7ab6fd94bdf53efd016a6b9e57295528a1a36646eb5b2cd29d6c8	cc99e267
3	exampleuser	5da2733e715a34750697202510caeb17aaadaa17110c2f46e55a522bdbb5e40f	b838ad77
4	corona.virus	be85430dd86503004edabf2d92b20923889437e0de2bb9623006b0ed03d93700	e46dd2e5

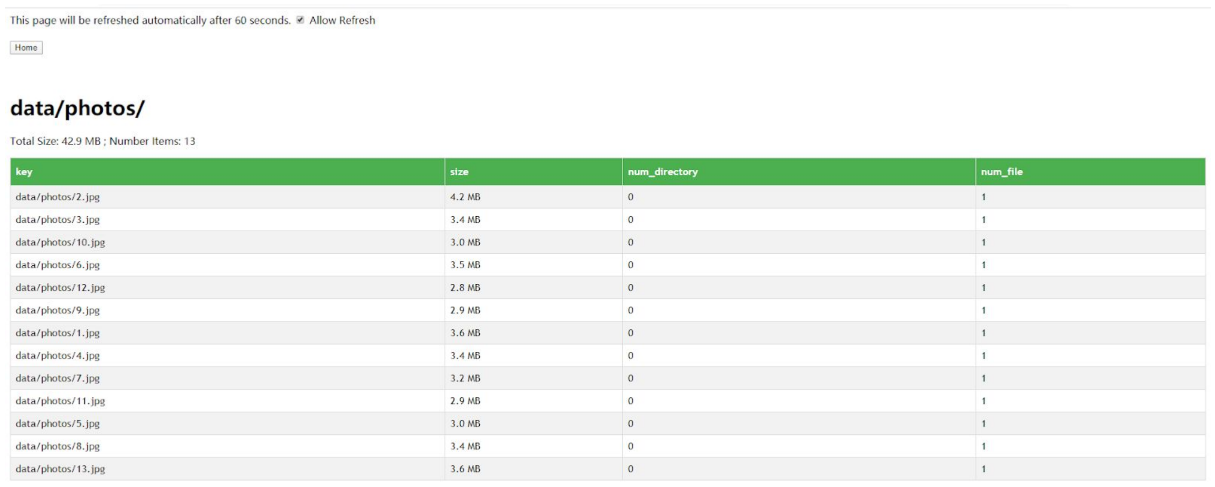
Figure 6. Screenshot of RDS MySQL Account Table Page on Manager Application


This page will be refreshed automatically after 60 seconds. ☑ Allow Refresh

ece1779.photo

id	account_id	name
1	3	3c1920caly1gcthg0upzhj223v35s4qr.jpg
2	3	3c1920caly1gcthg76uhyj223v35snpf.jpg
3	3	3c1920caly1gcthgdmcizj223v35s4qr.jpg
4	3	3c1920caly1gcthgjkl9uj223v35s4qr.jpg
5	3	3c1920caly1gcthgoj3d7j235s23vnpe.jpg
6	3	3c1920caly1gcthguq5wuj223v35s4qr.jpg
7	3	3c1920caly1gcthh0f2ujj235s23vu0y.jpg
8	3	3c1920caly1gcthh5ziiyj223v35sx6q.jpg
9	3	3c1920caly1gcthhauok0j223v35shdu.jpg
10	4	4F0A9165.jpg
11	4	4F0A9167.jpg
12	4	4F0A9162.jpg
13	4	4F0A9135.jpg

Figure 7. Screenshot of RDS MySQL Photo Table Page on Manager Application


This page will be refreshed automatically after 60 seconds. ☑ Allow Refresh

data/photos/

Total Size: 42.9 MB ; Number Items: 13

key	size	num_directory	num_file
data/photos/2.jpg	4.2 MB	0	1
data/photos/3.jpg	3.4 MB	0	1
data/photos/10.jpg	3.0 MB	0	1
data/photos/6.jpg	3.5 MB	0	1
data/photos/12.jpg	2.8 MB	0	1
data/photos/9.jpg	2.9 MB	0	1
data/photos/1.jpg	3.6 MB	0	1
data/photos/4.jpg	3.4 MB	0	1
data/photos/7.jpg	3.2 MB	0	1
data/photos/11.jpg	2.9 MB	0	1
data/photos/5.jpg	3.0 MB	0	1
data/photos/8.jpg	3.4 MB	0	1
data/photos/13.jpg	3.6 MB	0	1

Figure 8. Screenshot of S3 Directories Page on Manager Application

2.2.5 Auto Scaling Control

The last section on the right is associated with auto-scaler control and information. Users can use the four range bar to set auto-scaler policy. The three status bars on the left indicate the state of auto-scaler. Users can also use the button below the status bar to stop or start auto-scaler. The auto-scale is set to run in default when the manager application starts running.

3 User Application

3.1 Storage

User-uploaded photos and processed photos were all stored locally on the EC2 instance that was running the user application server. However, it is crucial to migrate the storage onto the cloud in order to scale up the user application server capacity. All static storages have been migrated onto Amazon Simple Storage Service (Amazon S3). S3 does not have notions of directory or folder, so in order to reuse the existing photo management logic and architecture, a simple abstraction layer around Boto3 S3 Application Programming Interface (API), has been created to mimic filesystem operations.

Once the photo is uploaded by the user and processed on the server, they are then uploaded to S3. The S3 bucket dedicated to using by the user application is set to private. Photos stored on S3 are retrieved by using pre-signed URLs, which give the application users temporary permissions to view the photos. The pre-signed URL expires after a short period of time.

The S3 abstraction layer has been created as a library for use by both the user application and manager application. For manager application, its most important usage is probably to track down the total size of a set of files. Boto3 S3 API works by sending requests over the network, which is slow by its nature. Each single S3 query function requires at least sending a few network requests to the AWS server, of which the runtime just simply scales up as the number of queries increases. The team has identified this runtime bottleneck of Boto3 S3 API and has come up with a workaround to do batch processing whenever possible. Instead of sending 1000 requests to query for sizes of 1000 objects, the simple wrapper now collects them as a single query request to the Boto3 S3 API. The simple wrapper will then do its query using the result returned from the single query request sent to the Boto3 S3 API. This has sped all S3 queries compared to using the Boto3 S3 API in a naive way.

3.2 Database

The team has migrated the backend MySQL database from local MySQL instance on Amazon Relational Database Service (RDS) on the cloud, which is an essential process to scale up the user application server capacity. The cloud database service allows data synchronization across multiple user application servers running on multiple EC2 instances. The migration process is very straightforward and easy: the database interface and schema is not modified at all; only the host is changed from localhost to RDS host.

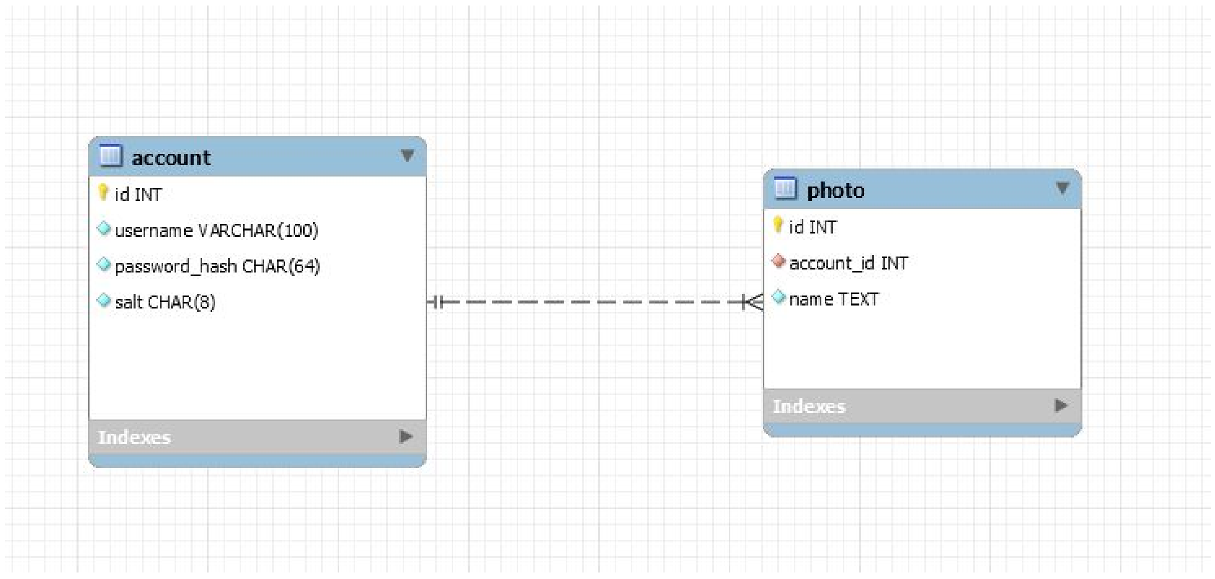


Figure 9. Database Schema

4 Manager Application

4.1 Load Balancer

The load balancing is achieved by using the load balancer provided by AWS provided load balancer, with the strategy of Round Robin.

4.2 Auto-scaler

Once the average CPU usage falls below or goes above the min and max threshold, the auto-scaler will try to resize the pool. The team has introduced a state pattern to better control the process of scaling and it avoids upsizing or downsizing too much toward the working pool.

The auto-scaler is designed to stabilize the CPU usage between the min and max threshold. After a resizing command is issued to Amazon AWS, the auto-scaler will check and wait for the worker pool to complete resizing, which is indicated by the number of running instances becoming equal to the projected worker number. If resizing takes too long, which can be caused by poor internet status, the auto-scaler handles it by setting a timeout period and will proceed to the next state.

If a resizing is completed or timeout, the auto-scaler enters a cooldown status. The granularity of CPU utilization stored in the AWS metric is one minute, however, a resizing process can take much shorter than one minute. Thus when the worker pool completes resizing, the latest CPU usage affected by such change is still not available. The auto-scaler should not be issuing another resizing command using the outdated CPU utilization. A cooldown period is introduced so that the auto-scaler can check if a new CPU utilization status is published so that the auto-scaler becomes ready for the next resizing

4.3 Worker Number Count Monitor

Another backend process running in the manager application is tracking the number of workers in the worker pool. It takes a record of the number of running EC2 instances in the pool once per minute and saves the sequence in memory. The frontend manager UI uses it to produce a worker number by time graph.

4.4 Authentication Process

As manager applications are powerful in managing all the AWS services that are keeping user applications running efficiently and smoothly, it also exposes a lot of security vulnerabilities of the entire user application server system if not treated carefully. The team has identified the most critical feature of the manager application being its security. To prevent creating a lot of overhead for a manager application that is only used by a few people, the team has brainstormed a lot of ideas and finally come up with a simple but effective solution, email-based verification.

An email address whitelist is hardcoded in the manager application server code. Only the email addresses that are inside the whitelist are allowed to proceed. Hardcoding the whitelist in the code does not compromise the overall security of the manager application authentication process since the manager application server is only allowed to run with AWS credentials set up properly, which makes modifying the server source code not an option for the attacker. Next, the authentication session generates a random one time password that expires after a very short period of time and is then sent to the input email address that is inside the whitelist. The user needs to retrieve the one time password from the mail inbox and copy them back to the authentication page in order to proceed. The one-time password is only valid once, within the expiry period, and only for the web session where the password is generated from. The one time password also expires immediately if the authentication web session receives an incorrect password. The level of security of the manager application server is tied to that of the email service providers. Lastly, the authenticator immediately deletes all sent emails that contain one time passwords.

The authentication module is written in a modular fashion and is very loosely coupled with the manager application core logic. The team can disable the current authentication process by modifying a single line of code when the manager application is not running under the public domain. The team is also able to switch to another authentication model very easily.

5 Infrastructure

The user application server capacity has scaled, so is the scale of the project. The project has had two server components and has been incorporating more and more AWS services, with user sensitive data stored on the cloud. However, the team is not willing to sacrifice development productivity and conveniences for security that is enforced with a single-rule policy. The team has actually configured a set of security rules to allow local development and testing without compromising the security of AWS services.

The first thing that the team has done is to control all accesses to the core AWS services that the manager application and user application are connected to. The RDS database is controlled to be only accessible from certain IP addresses; while the remaining AWS services are all accessed via Boto3 API, which requires AWS credentials or IAM roles. Only the machines owned by the team and certain EC2 instances have such accesses. Then the only potential channel an attacker can abuse is the manager application web interface. The team has implemented a whitelist email-address based one-time password verification approach for authentication purposes, which was explained in the previous section. In addition, the S3 URLs for user photos in the user application are also set with appropriate expiry periods. The team believes the above approaches towards the security of the entire system is sufficient given the scale and purpose of the project.

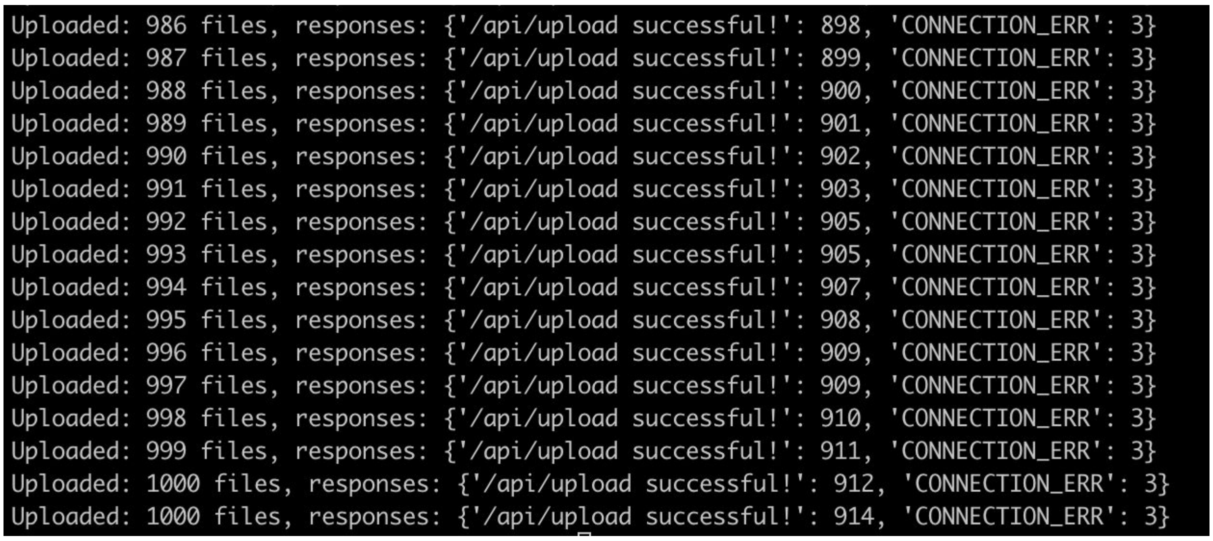
In addition, with more services and especially more EC2 instances brought in to host the servers, the team has also come up with a deployment plan that allows seamless upgrading. Bash scripts are created to automatically install all libraries, as well as launching the servers. In addition, it also fetches the GitHub repo for the latest version. The EC2 instances are configured to automatically run the above bash scripts during startup. In other words, the entire system can automatically update itself once the EC2 instances are rebooted. This has significantly increased the productivity and efficiency of the development, as the latest version of the servers can be synced to all EC2 instances automatically.

6 Results

We run two sets of experiments to verify our auto-scaling policy. For both experiments, we used the same policy as the following:

- Scale up threshold: 0.6
- Scale down threshold: 0.1
- Scale up ratio: 4
- Scale down ratio: 0.5

The first experiment utilizes the generator to upload 1 image per second, and up to 1000 images. The following two screenshots show the successful results as we can see that only 3 requests were lost due to the help of the auto-scaler. The reason for the 3 lost packages is due to the slow refresh rate of monitoring the CPU utilization, which is set at once every minute and is out of our control. If we can monitor the CPU with finer granularity, this problem can be easier solved. The auto scaler successfully scaled to 9 instances in a very short time and scaled down to 1 after the load generator stopped.



```
Uploaded: 986 files, responses: {'/api/upload successful!': 898, 'CONNECTION_ERR': 3}
Uploaded: 987 files, responses: {'/api/upload successful!': 899, 'CONNECTION_ERR': 3}
Uploaded: 988 files, responses: {'/api/upload successful!': 900, 'CONNECTION_ERR': 3}
Uploaded: 989 files, responses: {'/api/upload successful!': 901, 'CONNECTION_ERR': 3}
Uploaded: 990 files, responses: {'/api/upload successful!': 902, 'CONNECTION_ERR': 3}
Uploaded: 991 files, responses: {'/api/upload successful!': 903, 'CONNECTION_ERR': 3}
Uploaded: 992 files, responses: {'/api/upload successful!': 905, 'CONNECTION_ERR': 3}
Uploaded: 993 files, responses: {'/api/upload successful!': 905, 'CONNECTION_ERR': 3}
Uploaded: 994 files, responses: {'/api/upload successful!': 907, 'CONNECTION_ERR': 3}
Uploaded: 995 files, responses: {'/api/upload successful!': 908, 'CONNECTION_ERR': 3}
Uploaded: 996 files, responses: {'/api/upload successful!': 909, 'CONNECTION_ERR': 3}
Uploaded: 997 files, responses: {'/api/upload successful!': 909, 'CONNECTION_ERR': 3}
Uploaded: 998 files, responses: {'/api/upload successful!': 910, 'CONNECTION_ERR': 3}
Uploaded: 999 files, responses: {'/api/upload successful!': 911, 'CONNECTION_ERR': 3}
Uploaded: 1000 files, responses: {'/api/upload successful!': 912, 'CONNECTION_ERR': 3}
Uploaded: 1000 files, responses: {'/api/upload successful!': 914, 'CONNECTION_ERR': 3}
```

Figure 10. Screenshot of the results for uploading 1000 images with a speed of one image per second

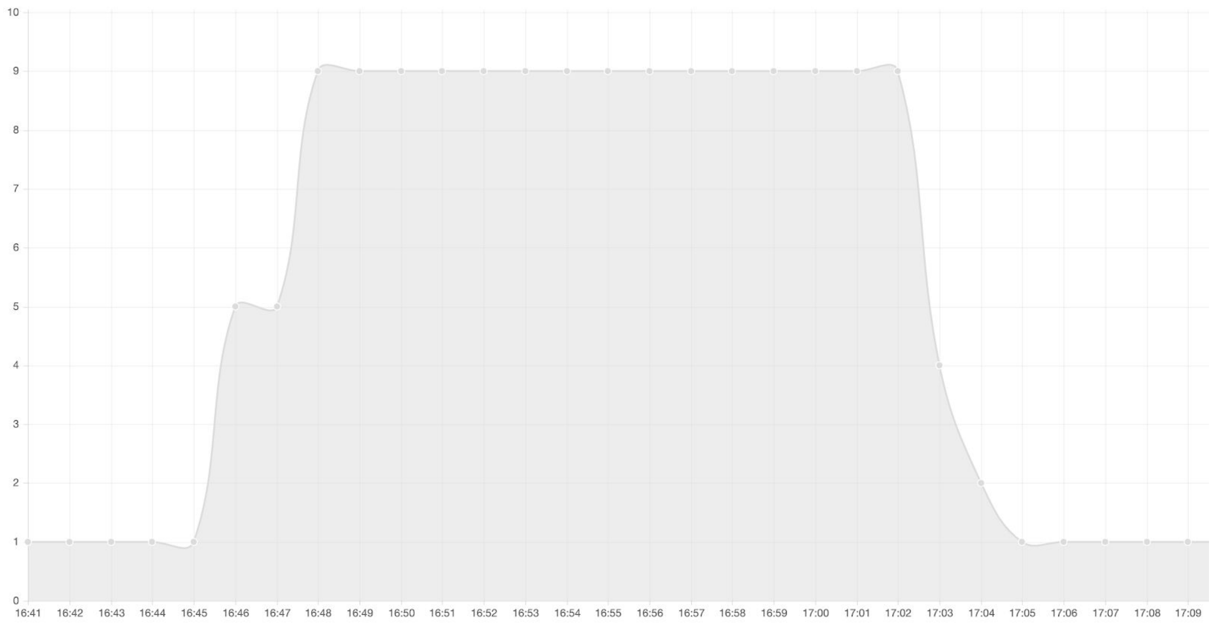
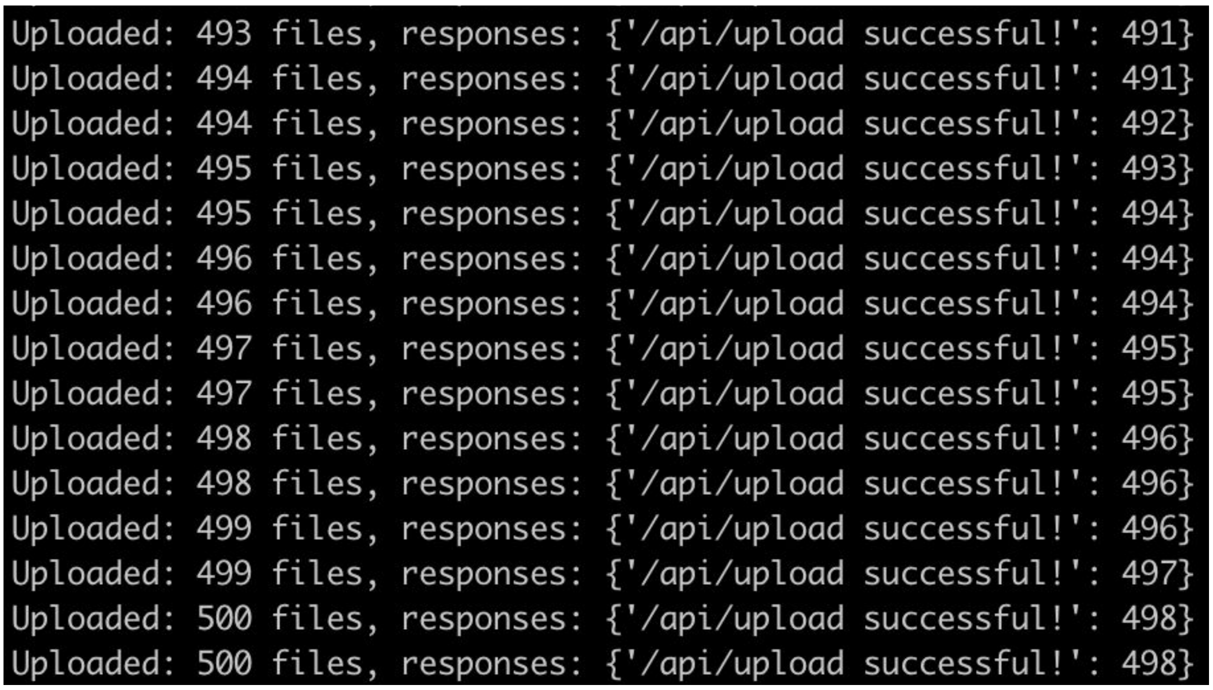


Figure 11. Screenshot of the number of instances we use during 1000 images testing

The second experiment we perform is to upload 500 images with a speed of 1 image every 2 seconds. Since the upload speed is reduced to half for this experiment compared to the previous one, no request is lost during as shown in Figure 12. The auto scaler successfully scaled to 5 instances and converged. After the testing is finished, the number of instances is automatically reduced.



```
Uploaded: 493 files, responses: {'/api/upload successful!': 491}
Uploaded: 494 files, responses: {'/api/upload successful!': 491}
Uploaded: 494 files, responses: {'/api/upload successful!': 492}
Uploaded: 495 files, responses: {'/api/upload successful!': 493}
Uploaded: 495 files, responses: {'/api/upload successful!': 494}
Uploaded: 496 files, responses: {'/api/upload successful!': 494}
Uploaded: 496 files, responses: {'/api/upload successful!': 494}
Uploaded: 497 files, responses: {'/api/upload successful!': 495}
Uploaded: 497 files, responses: {'/api/upload successful!': 495}
Uploaded: 498 files, responses: {'/api/upload successful!': 496}
Uploaded: 498 files, responses: {'/api/upload successful!': 496}
Uploaded: 499 files, responses: {'/api/upload successful!': 496}
Uploaded: 499 files, responses: {'/api/upload successful!': 497}
Uploaded: 500 files, responses: {'/api/upload successful!': 498}
Uploaded: 500 files, responses: {'/api/upload successful!': 498}
```

Figure 12. Screenshot of the results for uploading 500 images with a speed of one image every two seconds

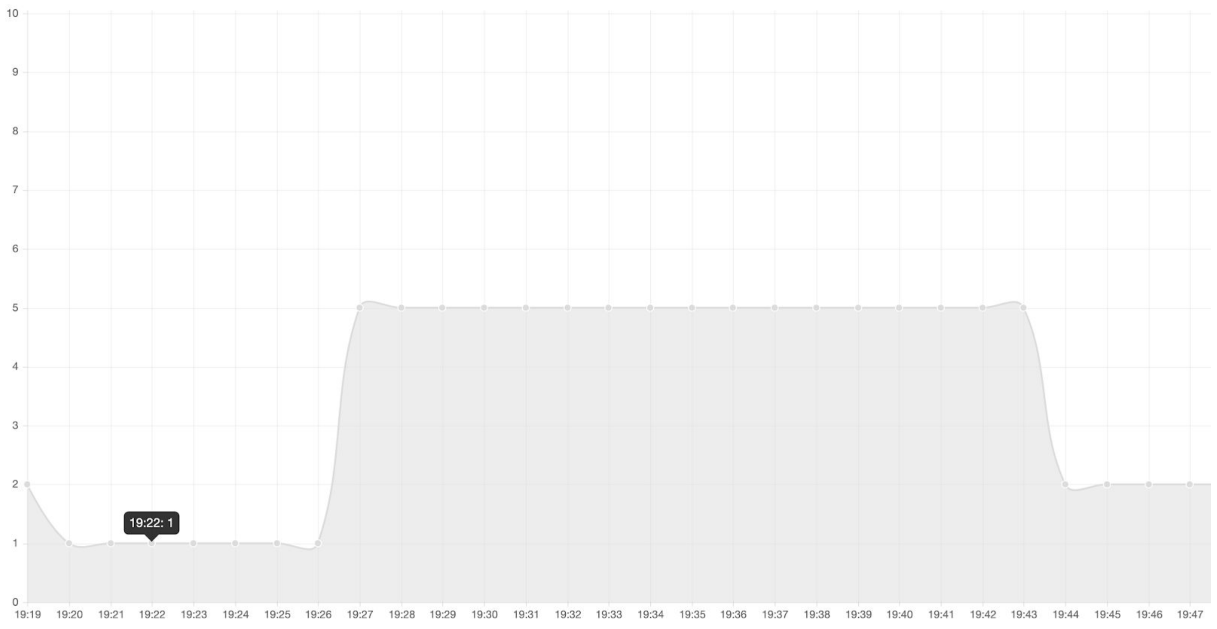


Figure 13. Screenshot of the number of instances we use during 500 images testing

7 Division of Tasks

Ruixin:

- Manager Application UI
- Manager Application backend
- Auto-scaler

Lichen:

- Implementation of a wrapper S3 interface for use by the user application and manager application
- Migration to RDS MySQL from the local MySQL instance
- Migration to S3 from the local filesystem storage
- Monitoring and controlling of RDS MySQL and S3 on the manager application
- Implementation of the manager application authentication process

Baiwu:

- Configure AWS with health check and load balancing
- Develop https request custom metrics and display results
- Design the auto-scaling policy
- Perform testing for the auto-scaling and load balancer