Photo-Sharing Social Platform Hosted on AWS

Yangyadi Jiang
The Chinese University of Hong Kong, Shenzhen
116010098

Yuchen Zeng
The Chinese University of Hong Kong, Shenzhen
116010281

Hongbo Zhang
The Chinese University of Hong Kong, Shenzhen
116010288

Laiyan Ding
The Chinese University of Hong Kong, Shenzhen
117010053

Yukun Lin
The Chinese University of Hong Kong, Shenzhen
117010158

Abstract

The cloud service is becoming more and more critical in applications deployment nowadays. In this paper, we design and implement a photo-sharing social platform "MyGram" based on the cloud services provided by AWS including the Elastic Beanstalk, S3, RDS, VPC, Route 53, CloudFront, etc.

Index Terms

Photo-sharing social platform, MyGram, AWS

I. Introduction

As the cloud is becoming more convenient and powerful to use with a relatively low cost and overheads for companies, especially the small ones, many companies are moving their services to the cloud. In other words, many applications and services are cloud-based.

As the beginner of cloud users, we want to explore the services provided by the cloud vendors by implementing a photo-sharing social platform using the cloud. This social platform focuses on the photo sharing between the users and their followers. Our main ideas for the platform are mainly inspired by the popular social platform "Instagram". Our platform is called "MyGram". According to our expectations, the target user group will be the students and the staffs of The Chinese University of Hong Kong, Shenzhen. This platform will obtain the basic functions including photo sharing, post management, account management, etc.

We have designed the entire social platform by ourselves using HTML as the frontend and Python Django to do the backend work. We implement the structures, database and functions for the social platform offline and then deploy the platform to the cloud using various services. After testing and the simulation of daily use, we invite the friends of our members to try our social platform and check the monitored data provided by the cloud services.

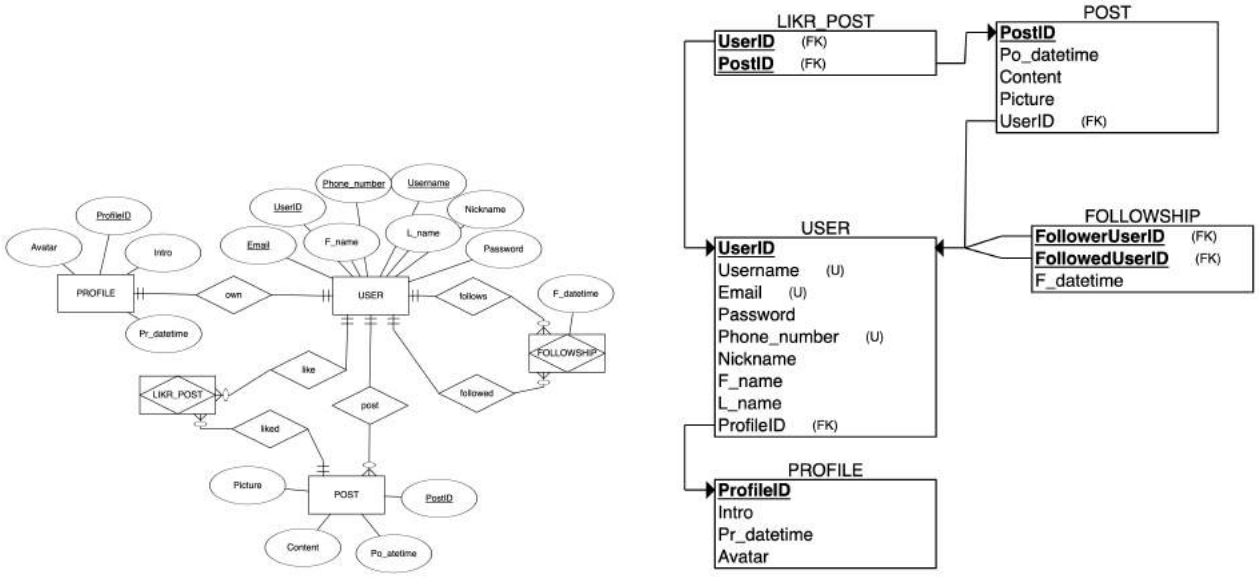
II. Project Design

We decided to create a photo-sharing social platform for the students and staffs in CUHK(SZ). This is very similar to the famous social media "instagram". Therefore, we set "instagram" as our final target and analyze its frontend layout and backend functions to realize its basic functions as a social platform. We have held a careful discussion on the platform and come out with some ideas. We first design the database structure, entities and instances needed and their relationship. Then we discuss the backend functions to form our general design of the project. The design has been slightly modified along with the development of the project.

A. Database Design

For every platform service, database is one of the most basic and critical component because every piece of data should be connected together without huge mistakes. Therefore, we have discussed carefully to design the ER Diagram for the database. The following Fig. 1(a) is the ER Diagram of our database and the Fig 1(b) is the Relational Schema of our database.

There are in total four entites that are profile, user, followship and post. The detailed instances of each entity is listed in Fig. 1(a). For the simplicity, we make every user have a unique UserID, Email and Phone Number. UserID is used as the key to do the query work. Both Email and Phone Number can be used by users to login. Each post will also has a unique PostID as the key to do the query work. Based on the initial ER Diagram, we have discussed the functions for the platform and modify the ER Diagram in some small details.



(a) ER Diagram

(b) Relational Schema

Fig. 1. Database

B. Backend Functions Design

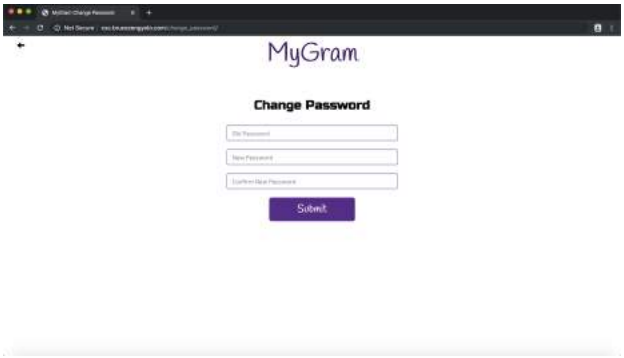
For the Backend, we use Python Django to implement the functions for our platform. The followings are the brief explanation of the main functions in our views.py file:

- sign_up: For user to register an account to use the web service.
- log_in: For user to login using the registered account.
- log_out: For user to logout the account.
- change_password: For user to change his/her password.
- edit_profile: For user to edit his/her profile including the nickname, introduction and avatar.
- new_post: For user to post a new post.
- del_post: For user to delete a post.
- toggle_like: For user to like or undo the like of a post.
- toggle_follow & unfollow: For user to unfollow a user.
- search_user: For user to search the users and return the results.
- show_homepage: Return the items that need to appear on the homepage. The items are the posts from the followings of the user and sorted in the chronological order.
- show_profile (show_my_profile): Return the items that need to appear on the personal profile page. The items are the information of the user including his/her username, nickname, the number of posts, followings and followers, and the post of that user sorted in the chronological order.
- show_post_detail: Return the items that need to appear on the detail page of a host. The items are the post's publisher, description and the number of likes for the post.
- show_my_following: Return the following list of the user.

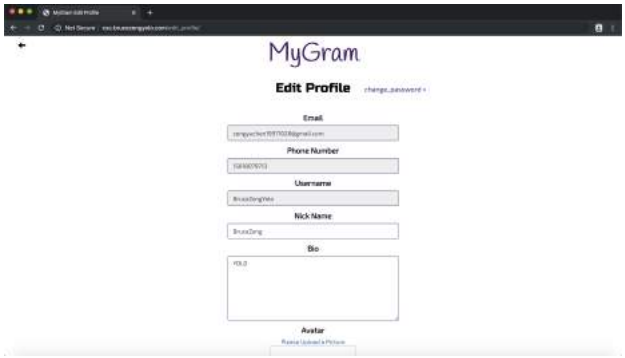
C. Frontend HTML Design

We use HTML, CSS and Javascript to implement our frontend design. The following are the brief explanations and screenshot of our websites:

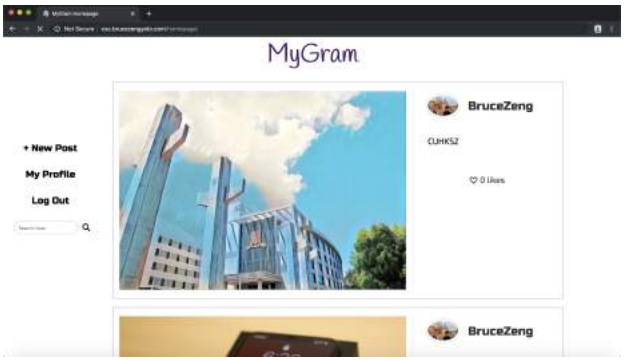
- Fig.2(a) change_password.html: For the purpose of changing the password.
- Fig.2(b) edit_profile.html: For the purpose of editting the profile.
- Fig.2(c) homepage.html: The homepage of our photo sharing social platform.
- Fig.2(d) login.html: For the purpose of login using the registered account. Our website only allows the registered user to use our service.
- Fig.2(e) manage_following.html: For the purpose of following management.
- Fig.2(f) new_post.html: For the purpose of posting a new post.



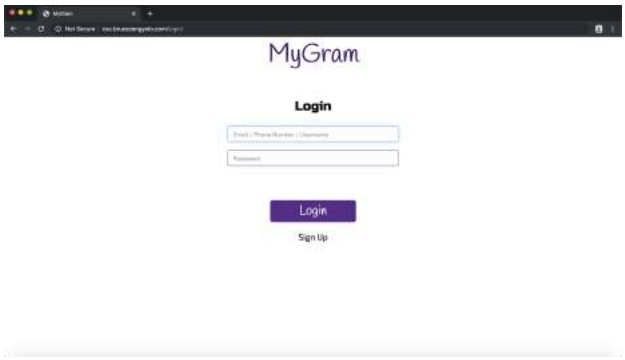
(a) Change Password



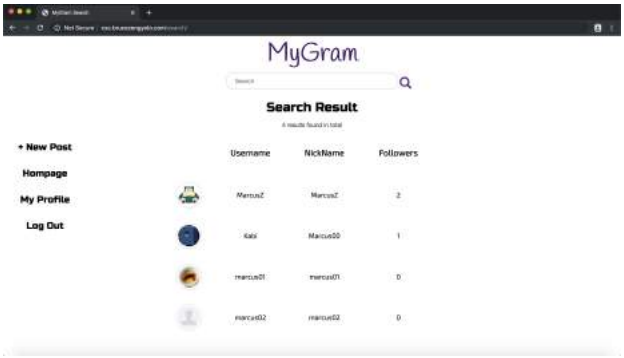
(b) Edit Profile



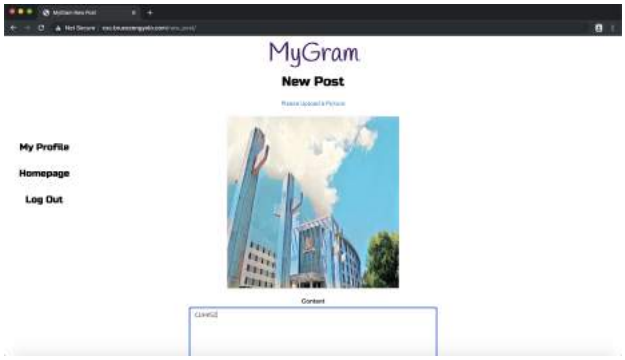
(c) Homepage



(d) Login



(e) Manage Following



(f) New Post

Fig. 2. Screenshots-1

- post_detail.html: For the purpose of showing the details of the post and post interaction and deletion.
- Fig.3(a) profile.html: For the purpose of showing the personal profile of a user.
- Fig.3(b) search.html: For the purpose of searching users using keywords and showing the search result.
- Fig.3(c) signup.html: For the purpose of registering an account.

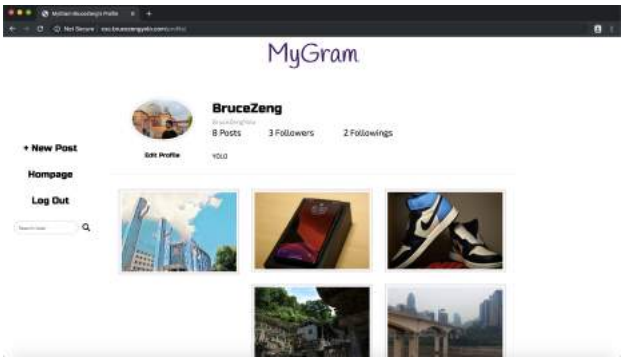
III. Platform Deployment on the AWS Cloud

We choose to use AWS as the cloud service to deploy our photo-sharing social platform.

After we have finished testing our platform on local database and computer. We clean the local database and prepare to deploy the platform on the cloud. As a team, we deploy the platform on the AWS together using the team leader's computer. The followings are the steps we take to finish the deployment:

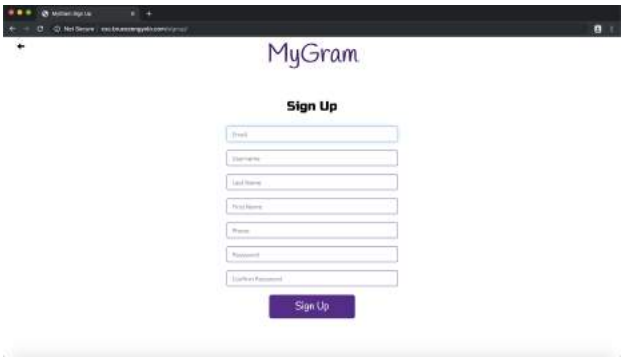
Step 1:

Create an empty database. Use Python Django to run manage.py to make migrations and then migrate to construct the structure of the database as Fig. 4. Then use "mysqldump" to dump the database for the later use of setting up the AWS RDS Aurora.



(a) Profile

(b) Search



(c) Sign Up

Fig. 3. Screenshots-2

```
jiangyangyadi@jiangyangyadideMacBook-Pro project4160 % python manage.py makemigrations
Migrations for 'MyGram':
  MyGram/migrations/0001_initial.py
    - Create model Followship
    - Create model LikePost
    - Create model Post
    - Create model Profile
    - Create model MyGramUser
    - Add field user to profile
    - Add field user to post
    - Add field postid to likepost
    - Add field user to likepost
    - Add field followed_user to followship
    - Alter unique_together for likepost (1 constraint(s))
    - Alter unique_together for followship (1 constraint(s))
jiangyangyadi@jiangyangyadideMacBook-Pro project4160 % python manage.py migrate
Operations to perform:
  Apply all migrations: MyGram, admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0001_initial... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying MyGram.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying sessions.0001_initial... OK
```

Fig. 4. Step1

Step 2:

Create a virtual environment to enter an isolated one for the purpose of easier configuration. The dependencies will be installed according to our requirements when putting onto the cloud. Here, we modify the requirement.txt by entering our requirement as followed:

- Django==2.1.7
- pytz==2019.3
- PyMySQL==0.9.3

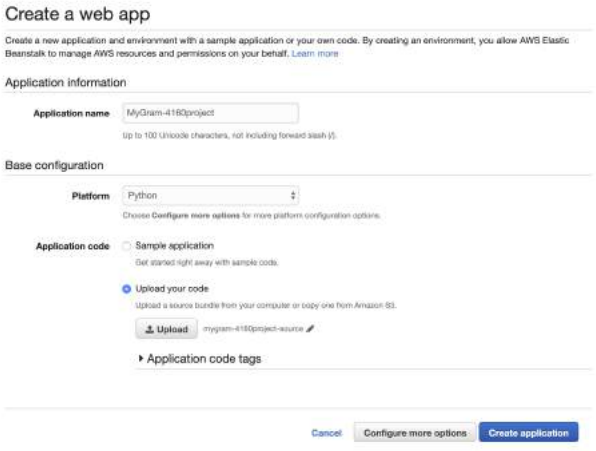
The setup of the prerequisites of our Django applications for Elastic Beanstalk steps are in the official guide from AWS.[1] .

Create .ebextensions directory in the project directory in which create a django.config file for project configuration. Write the following in the django.config:

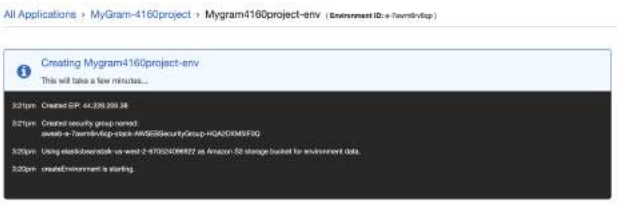
- option_settings:
- aws:elasticbeanstalk:container:python:
- WSGIPath: project4160/wsgi.py

Step 3:

Instead of using CLI to create the application, we go to the console to create Elastic Beanstalk environment and application. Then go to "Security Group" to add the inbound rule of SSH to allow access from everywhere. Then we upload the zip file of our project source code. The console will create a default environment when the application is created. After getting the url of the application, we add it to the "ALLOWED_HOST" in the "setting.py" file. The process is shown in Fig 5.



(a) Step3-1



(b) Step3-2

```
ALLOWED_HOSTS = ['Mygram4160project-env.2pxxzwuhge.us-west-2.elasticbeanstalk.com',]
```

(c) Step3-3

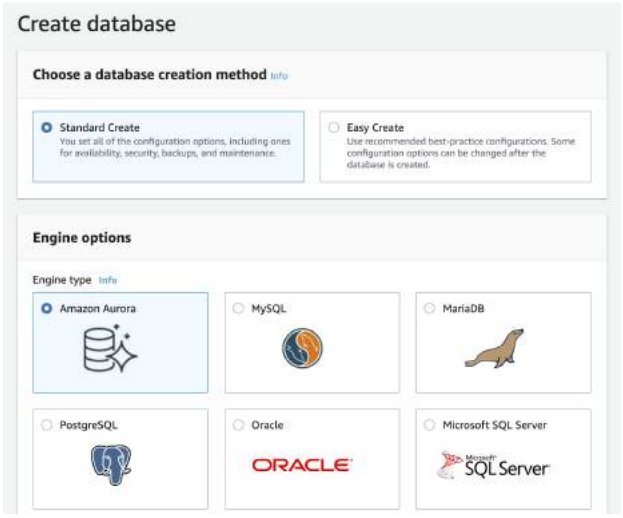
Fig. 5. Step3

Step 4:

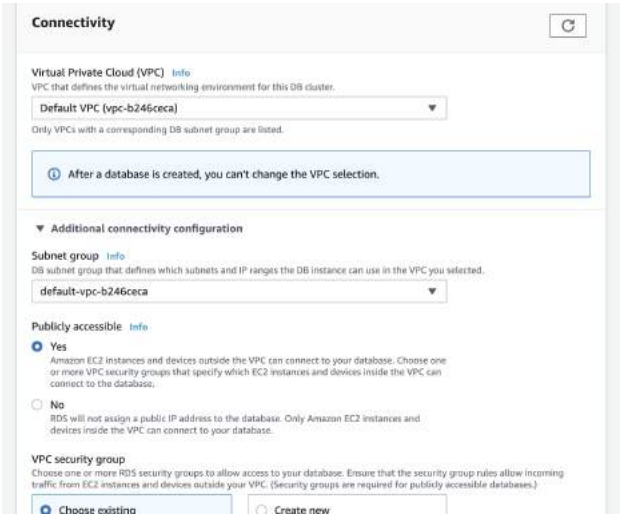
Create a database using Amazon Aurora as Fig. 6(a). It is critical to click "Yes" for "Publicly accessible" under "Connectivity" as Fig. 6(b), or the public will not be able to connect the database. Then go to "Security Group" to add the inbound rule of SSH to allow access from everywhere. Then import the previous dumped structured database to the Amazon Aurora. Then change the database setting in the "settings.py" including changing the "HOST" to the endpoint of the database cluster, the password and username. After updating the configure that is required to connect the cloud, upload the new version of the application in the zip form to the cloud through the console. Although we can upload the zip file from the console, in some situations, it is ok. However, in some cases, there will be an error called "Your WSGIPath refers to a file that does not exist." The safest way is to deploy the project folder using CLI. Therefore, in later update of the application, we choose to use the CLI with the instructions from the AWS website mentioned above. Fig. 6(c) is the screenshot of eb deployment using CLI in Step 4. Once the source code is deployed and the application is updated, we check whether the application can connect to the database. If the connection is established successfully, we go edit the inbound rule of the security group to disallow the public

[1]https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create-deploy-python-django.html

SSH access, so the database can only be accessed within the vpc security group of elastic beanstalk application and from the IP address of the administration device which is the group leader's laptop.



(a) Step4-1



(b) Step4-2

```
jiangyangyadi@jiangyangyadideMacBook-Pro project4160 % eb deploy
Creating application version archive "app-191130_185512".
Uploading MyGram-4160project/app-191130_185512.zip to S3. This may take a while.
Upload Complete.
2019-11-30 10:55:19    INFO    Environment update is starting.
2019-11-30 10:55:22    INFO    Deploying new version to instance(s).
2019-11-30 10:55:44    INFO    New application version was deployed to running EC2 instances.
2019-11-30 10:55:44    INFO    Environment update completed successfully.

Alert: An update to the EB CLI is available. Run "pip install --upgrade awsebcli" to get the latest version.
jiangyangyadi@jiangyangyadideMacBook-Pro project4160 % eb open
```

(c) Step4-3

Fig. 6. Step4

IV. Services Used For the Project

A. Elastic Beanstalk

We use AWS Elastic Beanstalk to run our application. EB can quickly deploy and manage applications in the AWS Cloud without having to learn about the infrastructure that runs those applications. We find this very convenient to deploy our application on AWS Cloud and it is much easier for us because we can update multiple versions by simply using "eb deploy". There are multiple ways for us to deploy our applications that are right through the console or using CLI.

When we are deploying, everything will be configured automatically for us including the EC2 instance, the python running platform, the dependency of packages and tools according to the "requirements.txt", the auto-scaling, etc. This reduces our management complexity and development cost. There is no additional charge for it and we pay only for the underlying AWS resources that our application consumes.

Furthermore, EB supports various application development language including Go, Jave, Python, etc. We choose Python Django to implement our backend functions. The logs can be easily viewed on demand in the dashboard. This can help us find the problems more easily.

With the help of "Monitoring" function shown in Fig. 7, we can check the health condition of the application and its environment, the CPU utilization and network. This will help us see the traffic on our website and make some optimization to adapt these situations.

Fig. 8 shows the dashboard of our Elastic Beanstalk. With the help of EB, we can easily deploy our applications and modify our applications on our local computer in case of some bugs and simply update it by using one line in the CLI to deploy.

B. Simple Storage Service

AWS Simple Storage Service is the storage service we use to store our deployed project file and photos posted by users. S3 also stores all the versions of our applications deployed. S3 has the advantages of being reliable, salable, cheap, secure, etc.

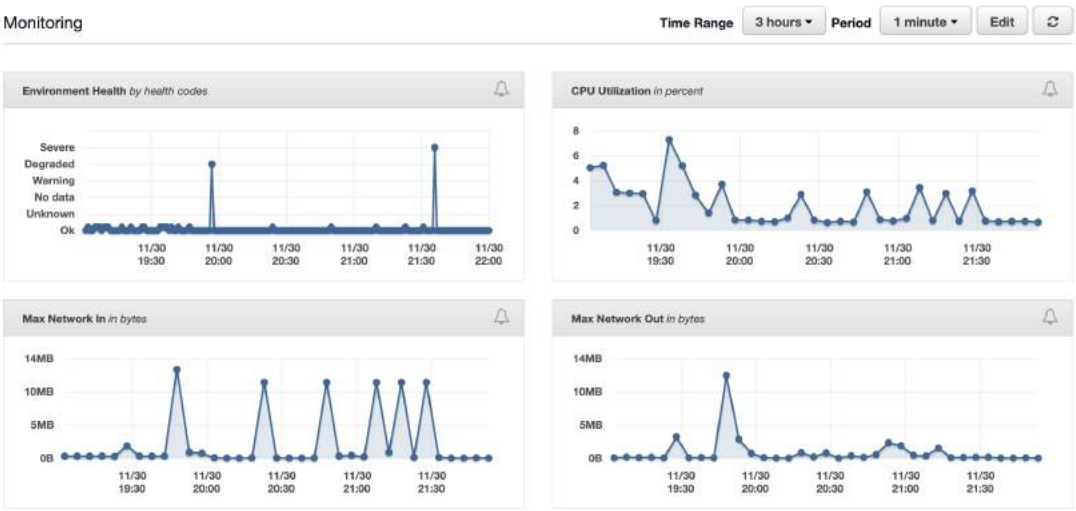


Fig. 7. EB-Monitoring

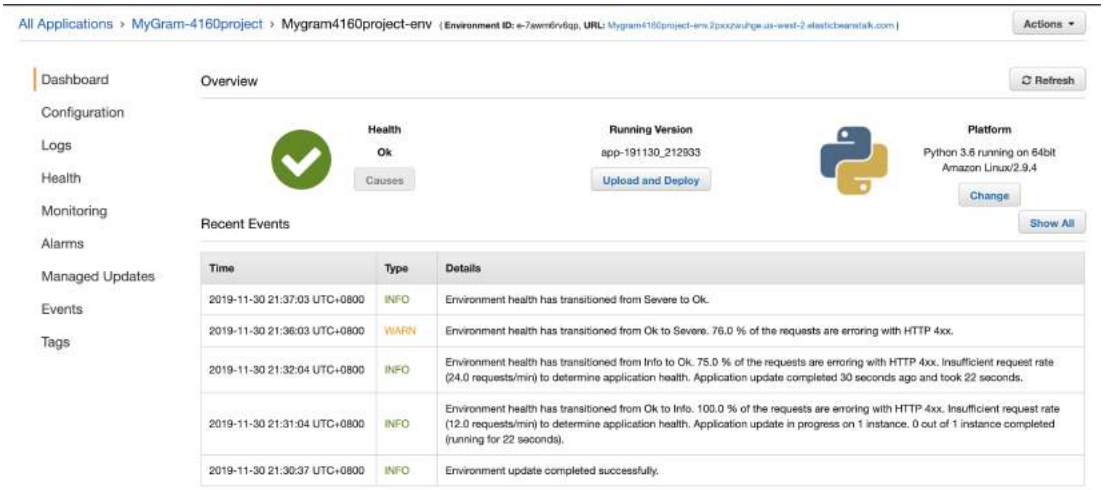


Fig. 8. EB-Dashboard

With the help of S3, it is much more convenient for us to track the versions of our applications. It will make our photo-sharing social platform much safer and robust to store the users' photos. Fig. 9 shows the snapshot of our S3 service.

C. Amazon Relational Database Service-Amazon Aurora

Amazon Relational Database Service provides us with a solution to store and use our designed database. RDS can provide cost-efficient and resizable capacity while automating time-consuming administration task such as hardware provisioning, database setup, patching and backups will benefit us with much convenience to implement our project.

We chose Amazon Aurora as our database engines for its compatability of MySQL and PostgreSQL. Aurora is up to five times faster than standard MySQL databases and three times faster than standard PostgreSQL databases which will make our application run at a faster time. It also provides the security, availability, and reliability to the database with a relatively lower cost which will relief our burden as we have a limited budget to establish a campus-wide photo-sharing social platform.

D. Amazon Virtual Private Cloud

Amazon Virtual Private Cloud provides us with an isolated section of the AWS Cloud where we can fully control our virtual networking environment, including selection of our IP address range, subnets management, etc.

With the help of Amazon VPC, we can fully control the access to our website. We can hide our database and the backend functions of our application in the private subnet which will make our application safer by restricting public users to access them, while the public users can still access the frontend website and services.

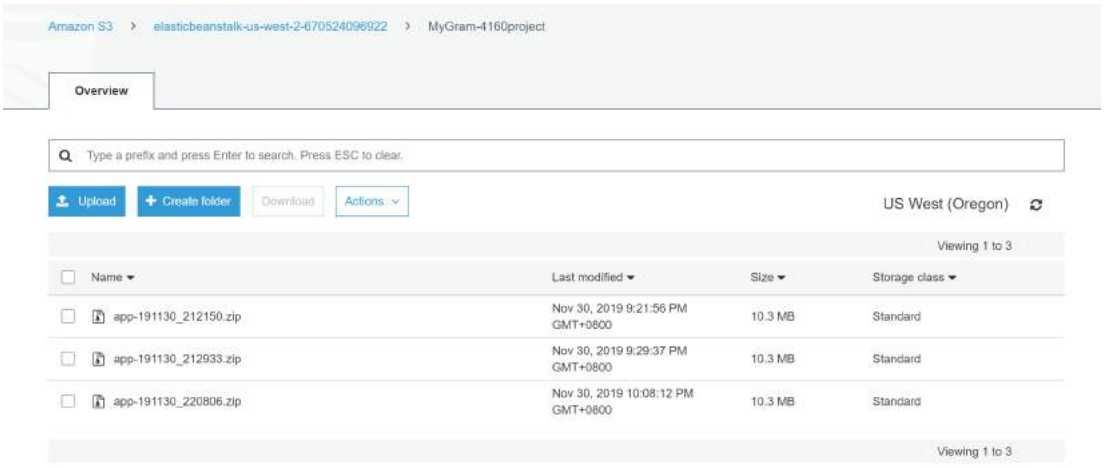


Fig. 9. S3

E. Amazon CloudWatch

Amazon CloudWatch is a monitoring and observability service for us, the managers and developers of MyGram, to check the status of our website by detecting anomalous behavior, viewing visualize logs and metrics. This will help us to have a better knowledge of the condition of our websites and make adjustment in case there is an emergency. Fig. 10 shows the CloudWatch of our database.

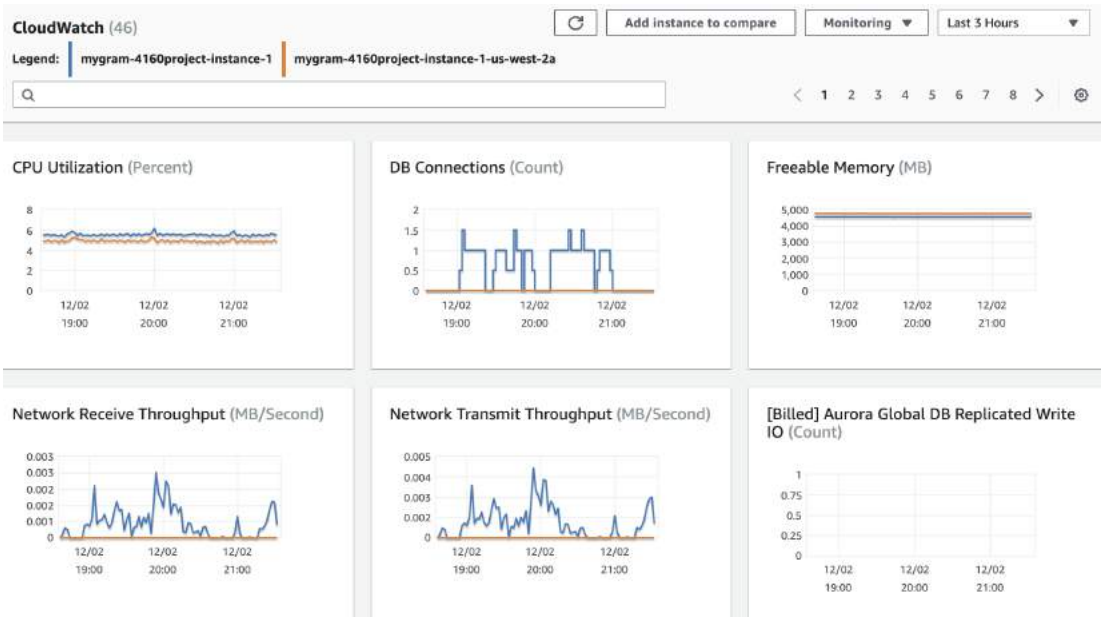


Fig. 10. CloudWatch

F. Route 53

In order to make our website service unique and easy to access. We use Route 53 to route end users to Internet applications by translating names csc.brucezengyolo.com to our EC2's public IPv4 public IP address 44.228.200.38 to access our website.

One of our team member Mr. Yuchen Zeng has his unique domain. Therefore, we try to use this AWS service. We first create a public hosted zone in the Route 53 as Fig. 11 and copy the four Nameservers to the domain provider's dashboard. Here, Mr. Zeng use Namecheap service and copy the four Nameservers to the "Custom DNS" field. Then add "44.228.200.38" and "csc.brucezengyolo.com" to the "settings.py" and deploy the new version of the application.

With the help of Route 53 and our unique domain, our photo-sharing social platform website can be easily accessed by enter "csc.brucezengyolo.com" instead of our long url provided by AWS.

Name	Type	Value	Evaluate Target Health
brucezengyolo.com.	A	ALIAS csc.brucezengyolo.com. (z3adspgvrk83y4)	No
brucezengyolo.com.	NS	ns-1349.awsdns-40.org. ns-1749.awsdns-26.co.uk. ns-1018.awsdns-63.net. ns-445.awsdns-55.com.	-
brucezengyolo.com.	SOA	ns-1749.awsdns-26.co.uk. awsdns-hostmaster.amaz	-
csc.brucezengyolo.com.	A	44.228.200.38	-

Fig. 11. ROUTE53

G. Amazon CloudFront

CloudFront is a large scale, global, and feature-rich CDN that provides secure application delivery. The CDN is Content Delivery caching. It uses cache to store the frequently used data and especially photo in our case. This can speed up delivery of static content. Furthermore, CloudFront can be used with SLL certificate to ensure security. And the network can also provide dynamic content that is unique to requester that is not cacheable. Because our social platform mainly shares photo contents which will have a large size. In order to improve the loading time and user experience we decide to use Amazon CloudFront to do CDN speed up. Fig. 12 shows the screenshot of the CloudFront service in AWS. After applying CDN speed up by using Cloudfront, the browsing of photos becomes smoother and the user experience of our application is improved by a large extend that the picture loading time and the web page response waiting time become more than 2 times shorter than before.

Fig. 12. CloudFront

V. Conclusion

By finishing this project, we have explored AWS services as much as we can and learn the real practice of using cloud services to implement a project. In the process of project development, we have witnessed the power of cloud that can really make application development and deployment easier for developers.

Acknowledgment

Our team started our project by designing the basic database. Then we discussed the functions needed for the platform and distributed the tasks to every group member. Yangyadi Jiang, Yukun Lin and Laiyan Ding are responsible for the functions implementation using Python Django. Yuchen Zeng and Hongbo Zhang are responsible for the frondend design and implementation. All of the members worked hard together to connect the frontend with the backend and finished the integration work. At the end of the project development, all the members were working together in the discussion room to explore the cloud service and find the solutions to the problems occur in the process of project deployment to the AWS cloud. During the cloud deployment and application testing period, we successively deployed more than 20 versions fixing errors, applying better solutions, adjusting user interfaces and supplementing more functions.The report is the fruition of unremitting efforts of all members. Everyone has a even contribution to the project.