COS20019 – Assignment 2

Developing a highly available Photo Album Website

Bui Minh Thuan
104486358
Tutorial Time: Wednesday 10:00AM
Date of Submission: 14/07/2024

I. Introduction

This report documents my works for Assignment 2, Unit Cloud Computing Architecture (May 2024). In this assignment, it concentrates on developing a highly available photo album website that allow user upload and view images in their album. The website can be accessed via this link: http://webserver-elb-784619010.us-east-1.elb.amazonaws.com/photoalbum/album.php.

II. Infrastructure Deployment

In the first part, I start with building and configuring the overall AWS services used in this assignment. Compared to Assignment 1B, the infrastructure for this assignment shares some similarities, but requires some changes to extend functionality and enhance availability for final website.

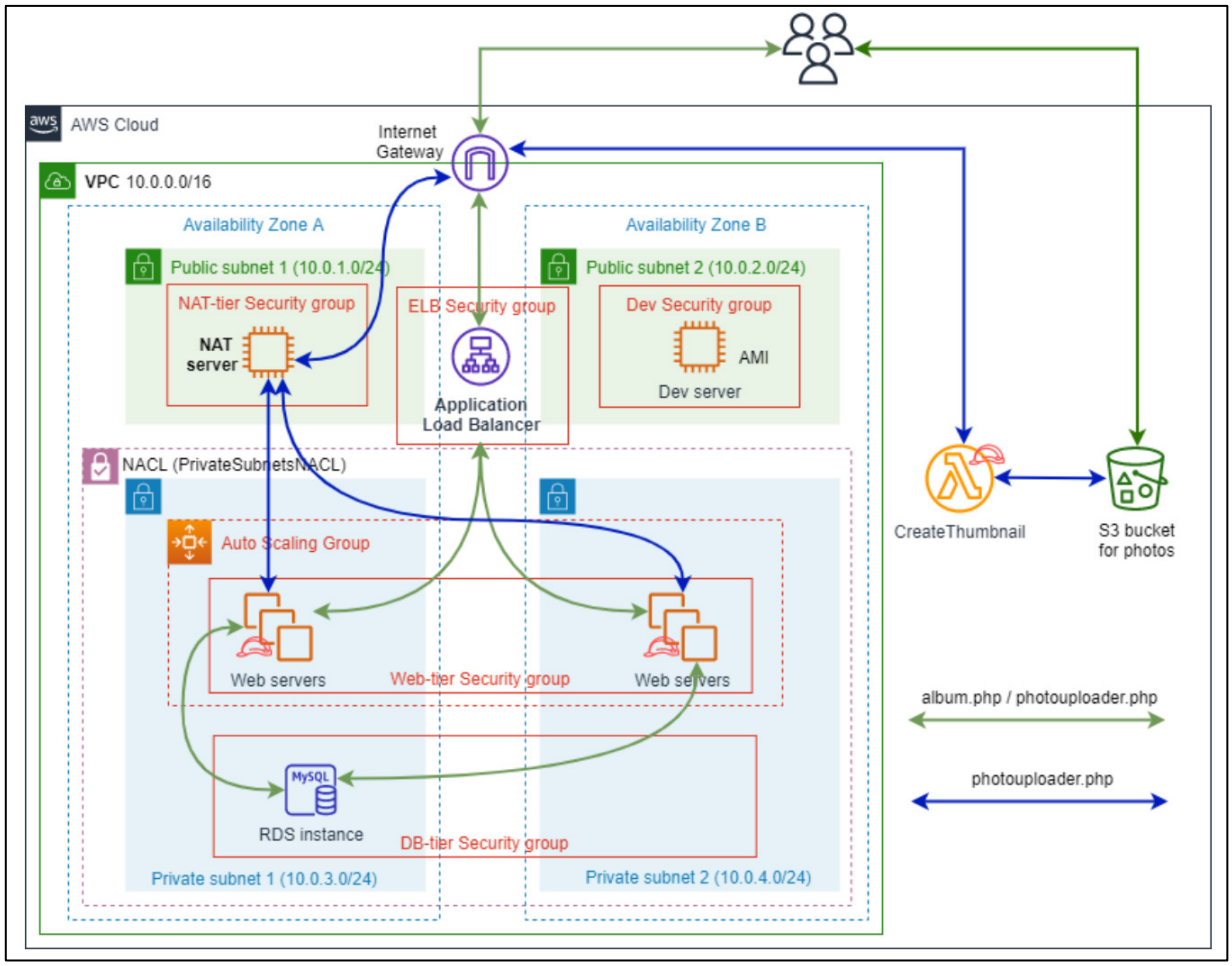

Figure 1: Overall infrastructure

A. VPC

For VPC, the configurations stay mostly the same with VPC for assignment 1B. It has "10.0.0.0/16" as CIDR, 2 Availability Zones with 4 subnets (1 public and 1 private for each AZ), 2 route tables (1 for public subnets and 1 for private subnets) and an Internet Gateway. However, because web servers in this assignment are deployed in private subnets, I added a NAT gateway which allows instances in private subnets can connect to the internet.

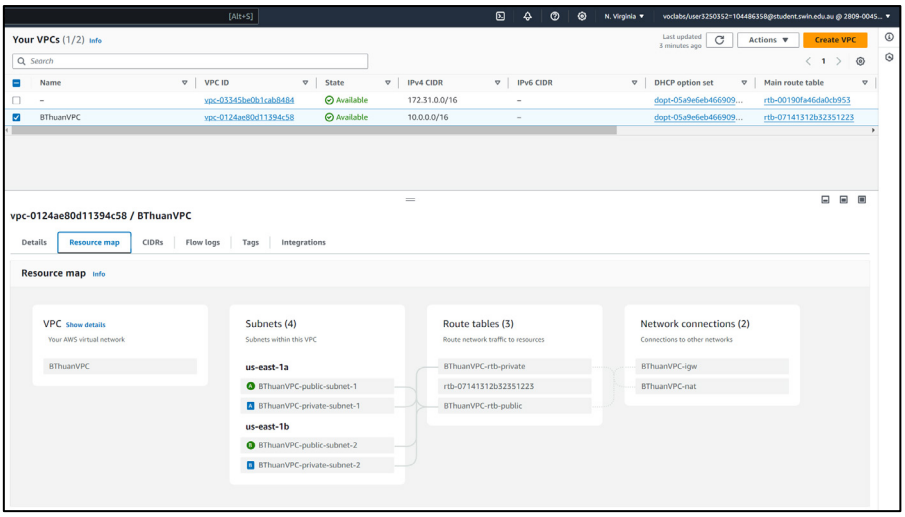

Figure 2: VPC Resources Map

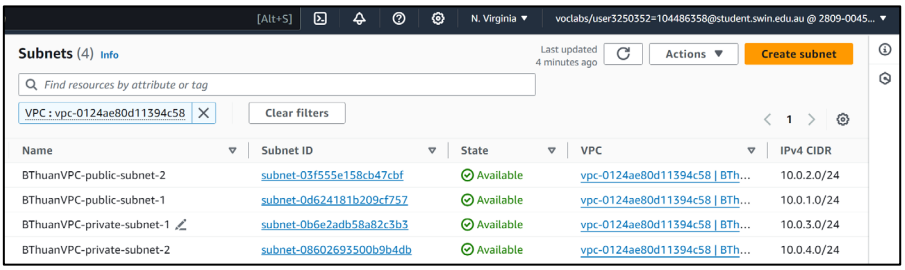

Figure 3: Subnets CIDR

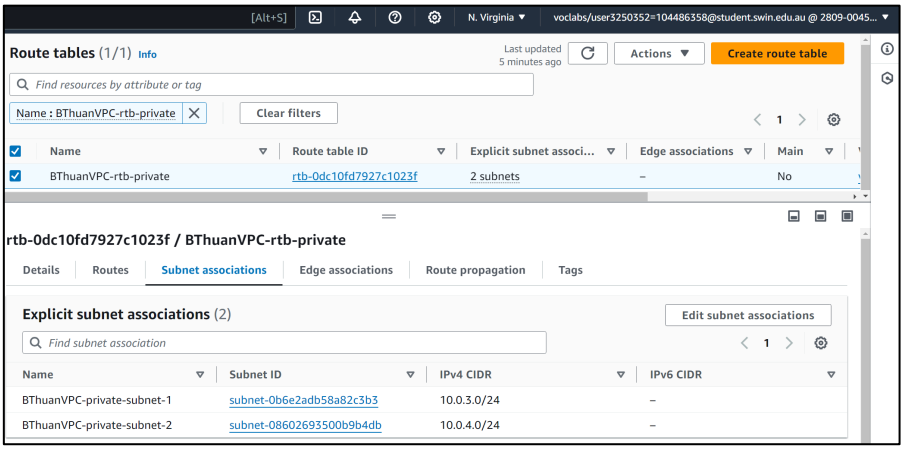

Figure 4: Private route tables associate with private subnets

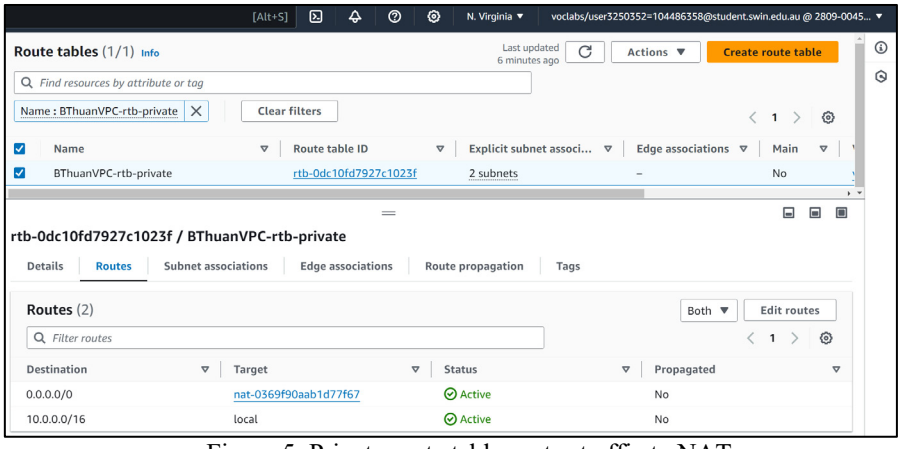

Figure 5: Private route table routes traffic to NAT

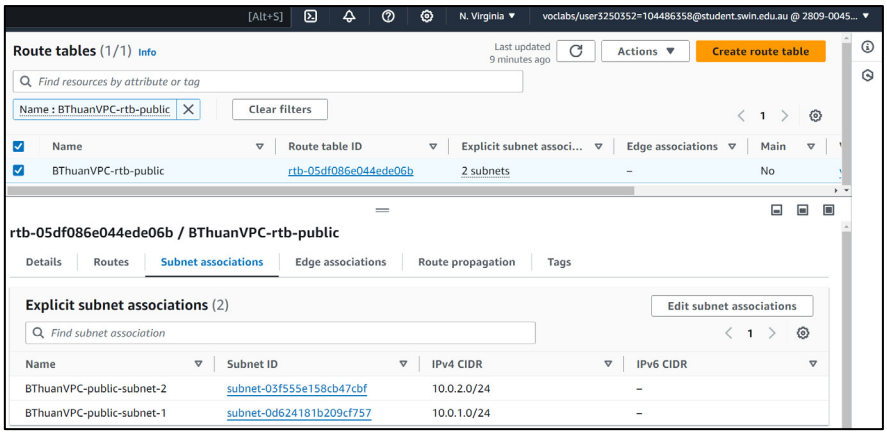

Figure 6: Public route tables associate with public subnets

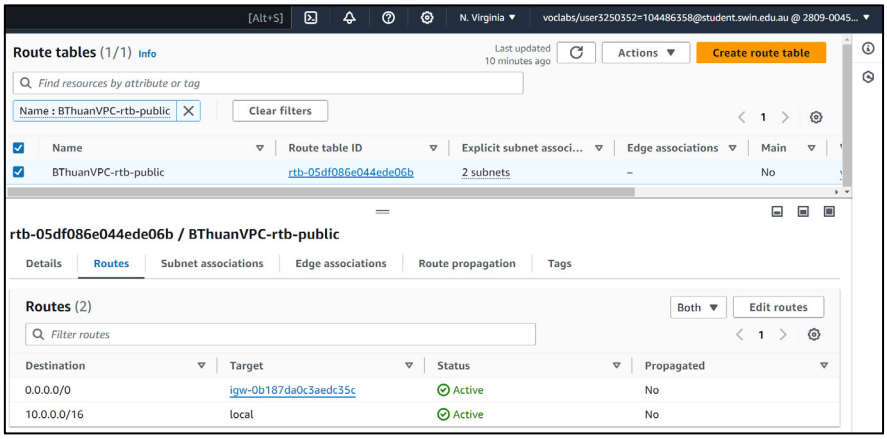

Figure 7: Public route table routes traffic to IGW

B. *Security Groups*

The next thing I need to configure after setting VPC is Security Groups. Unlike Assignment 1B, when all settings have already provided in the instruction file, in this assignment we need to analyze and decide Security Groups configurations. Below is my settings with explanations for each SG:

Security Group Name	Protocols	Sources
ELBSG	HTTP(80)	Anywhere
WebServerSG	HTTP (80)	ELBSG
	SSH (22)	DevServerSG
DBServerSG	MySQL (3306)	WebServerSG, DevServerSG
DevServerSG	All traffic	Anywhere

Figure 8: Public route table routes traffic to NAT

1) ELBSG: The first SG is for ELB (Elastic Load Balancer). Because this ELB will receive web request on behalf of web server, it needs to be able to accept HTTP traffic from anywhere.

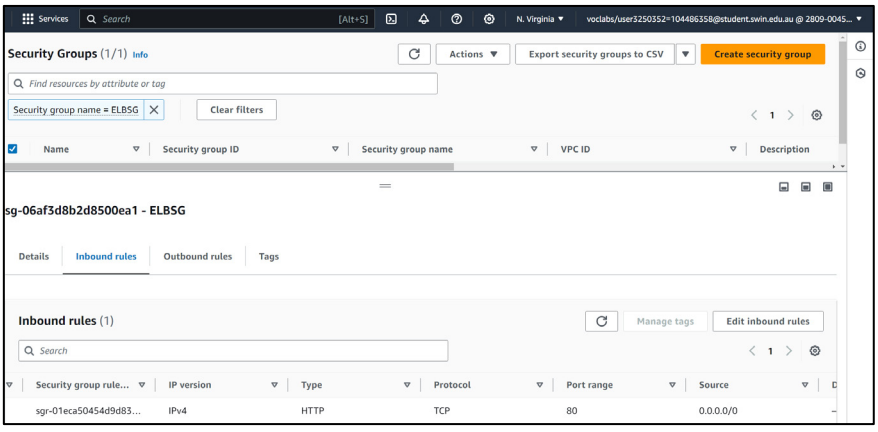

Figure 9: Inbound rules for ELBSG

2) WebServerSG: "WebServerSG" is for servers of this website. Here, I allow 2 types of traffic: HTTP from "ELBSG" and SSH from "DevServer". The first traffic is for web access, while the second one allow I remote access to web server from dev server and test ICMP between dev server and web server.

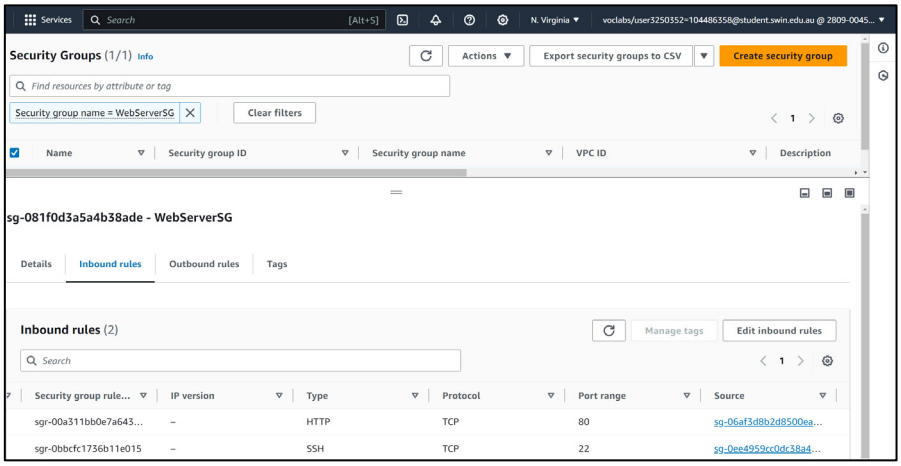

Figure 10: Inbound rules for WebServerSG

3) DBServerSG: Database for this website will be in the third Security Group – "DBServerSG". This SG will accept only MySQL via port 3306, but from 2 sources: "WebServerSG" and "DevServerSG". First source will be for web server to access its database, while second source allow us remote manage database with phpMyadmin (see details in RDS section).

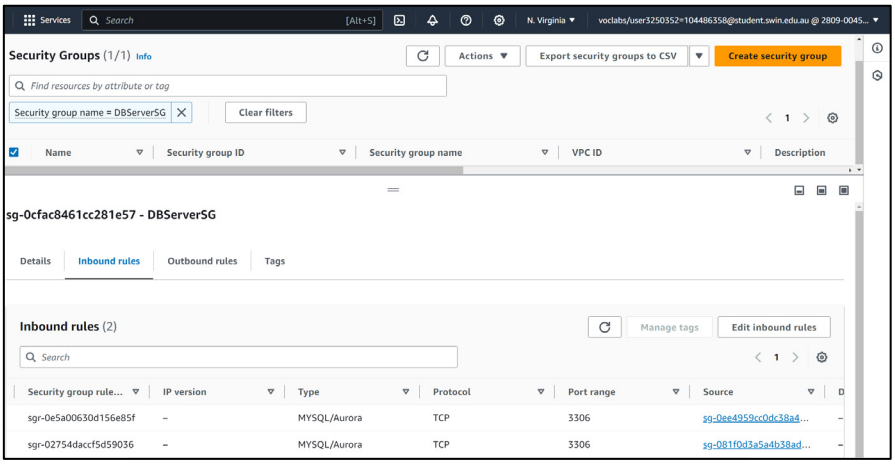

Figure 11: Inbound rules for DBServerSG

4) DevServerSG: Finally, the last needed SG is for Dev Server. As specified in instruction file, this SG doesn't need to follow least-privilege principles, therefore, allowing all traffic from anywhere is enough.

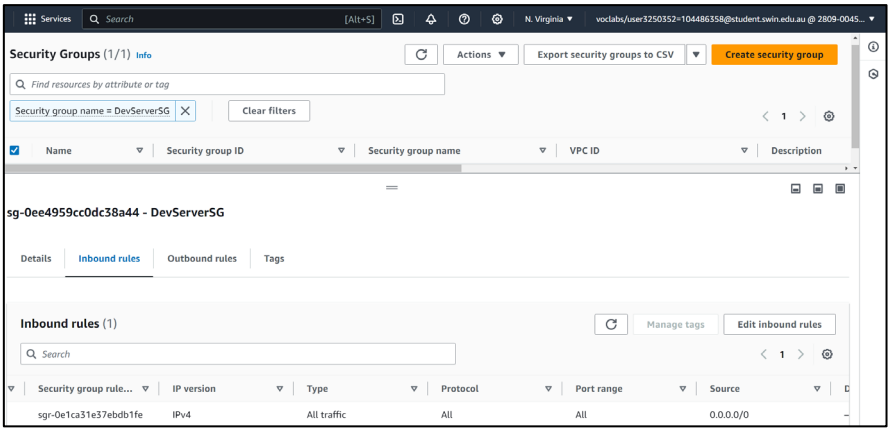

Figure 12: Inbound rules for DevServerSG

C. Network Access Control List (NACL)

Another layer of security for this website is NACL, which filters traffic to subnet. In this scenario, 1 NACL called "PrivateSubnetsNACL" is required for 2 private subnets that contains web servers and databases. This NACL will follow least-privileged principles, block all ICMP traffic from Dev Server to corresponding subnets but still allow the web-related requests so the website runs as expected.

For inbound rules, here is my rules:

1. Rule 100: Deny ICMP from Dev Server, so instances inside NACL can not receive ping from Dev Server instance
2. Rule 110: Allow MySQL from DevServerSG (10.0.2.0/24)
3. Rule 120, 130: Allow HTTP traffic from ELB, which can be in public subnet 1 (10.0.1.0/24) or public subnet 2 (10.0.2.0/24)
4. Rule 140: Allow SSH from Dev Server, so we can remote access from Dev Server and test ICMP ping
5. Rule 150: Allow MySQL from 10.0.4.0/24 because the RDS is deployed in 10.0.3.0/24, while ASG might create instance in 10.0.4.0/24, so we need to enable this traffic
6. Rule 160: Allow response from Lambda function. Even though Lambda doesn't have an associated IP but its response still uses ephemeral ports 1024 – 65535

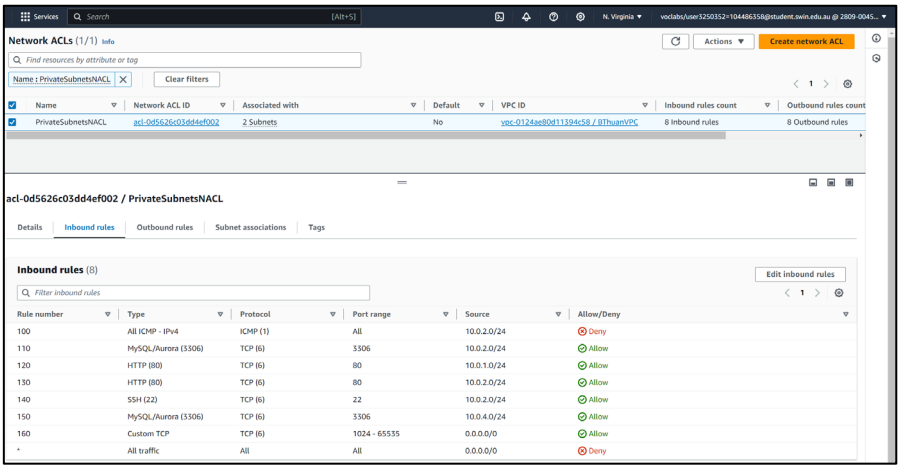

Figure 13: Inbound rules for "PrivateSubnetsNACL"

For outbound rules, here is my rules:

1. Rule 100: Deny ICMP to Dev Server, so instances inside NACL cannot send ping from Dev Server instance
2. Rule 110, 120: Enable TCP in ephemeral ports to 10.0.1.0/24 or 10.0.2.0/24 so web server can reply to web request from ELB
3. Rule 130: Allow send MySQL request to database from web server in 10.0.4.0/24 because sometimes a webserver can be deployed in subnet 10.0.4.0/24
4. Rule 140: Allow sending reply from database in 10.0.3.0/24 (port 3306) to web server in 10.0.4.0/24 (port 1024 – 65535), because webserver can be deployed in subnet 10.0.4.0/24 by ASG
5. Rule 150: Allow HTTPS, so the web server can upload image to S3 bucket and invoke Lambda function.

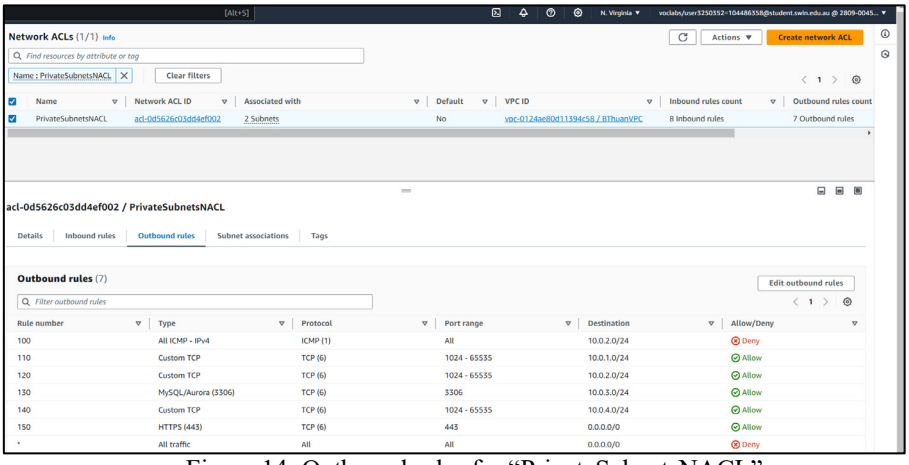

Figure 14: Outbound rules for "PrivateSubnetsNACL"

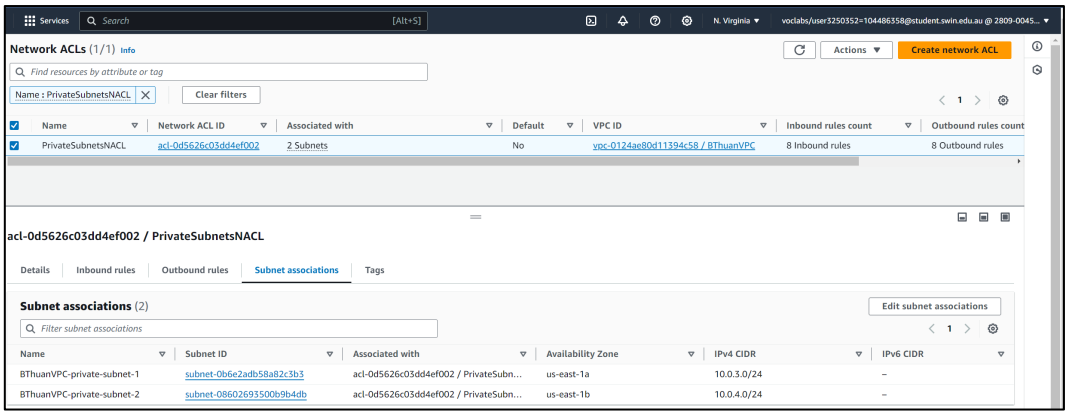

Figure 15: "PrivateSubnetsNACL" is associated with 2 private subnets

These rules ensure the web still function as expected, and block ICMP from/to Dev Server.

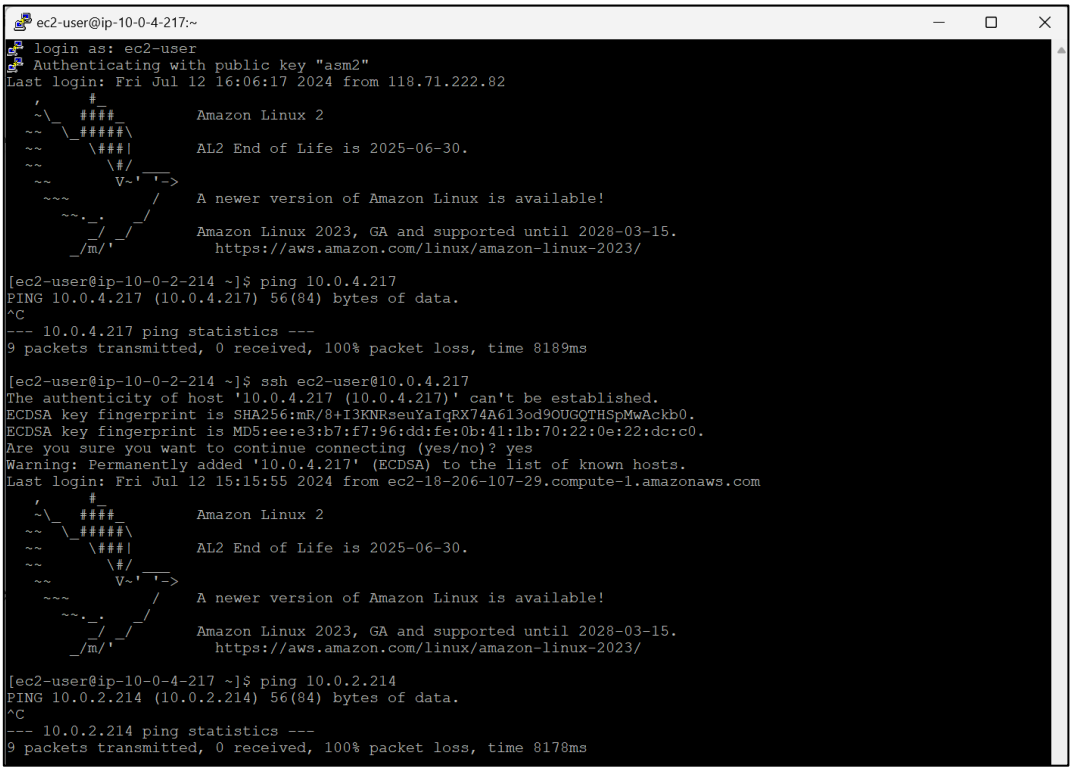

Figure 16: Cannot perform ping between Web Server and Dev Server

D. IAM Roles

In this assignment, we need EC2, Lambda and S3 being able to interact with each other, but still ensure security because we don't want all services in our account are publicly used. That's why we need IAM roles – which will restrict access of resources to only what we want and needed for the web to function.

Because this assignment is done in Learner Lab, we don't have any permissions to create new role, so I need to use a provided role called "LabRole". This role has all needed and no additional policy to resources in our account.

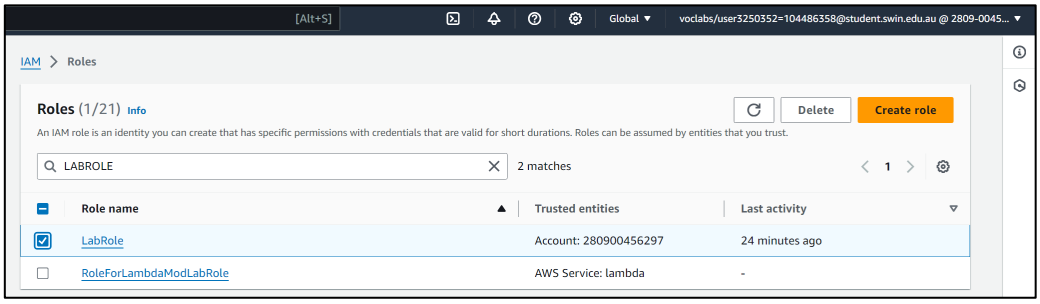

Figure 17: "LabRole" already created in Learner Lab account

This role will be used when I create the instance profile for the launch template, so it will allow the Web Server perform photo uploading with AWS PHP SDK, and when I set up execution role when create the Lambda function “CreateThumbnail”. Details about these 2 usages will be discussed in sections about “Launch Template” and “Lambda”.

E. S3 bucket

Same with Assignment 1B, photo uploaded are stored in an S3 bucket. However, this assignment required stricter access policy: the image should not be accessed publicly via internet, instead, it should be accessed by my website only.

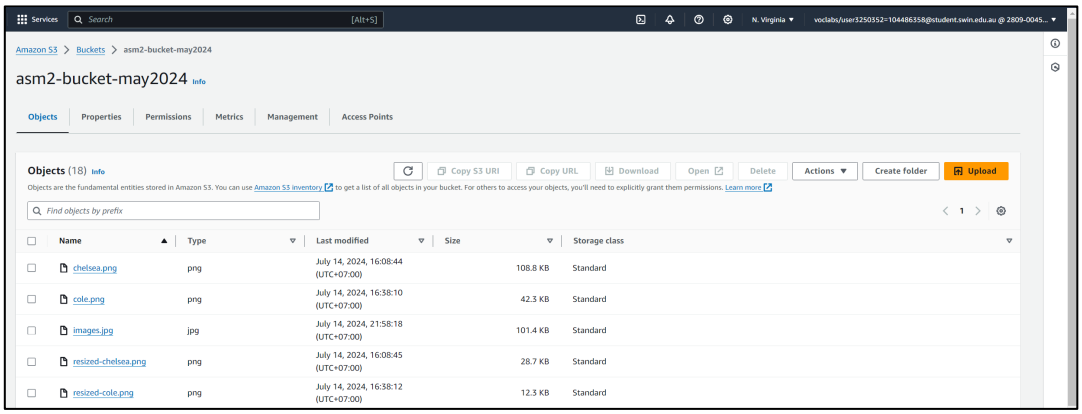

Figure 18: All images are stored in an S3 bucket

Because the image is only accessible through my website, I set up a bucket policy that restricts access to the bucket from a specific HTTP referrer [1]. In this case, only request from my website with DNS “webserver-elb-784619010.us-east-1.elb.amazonaws.com” or “www.webserver-elb-784619010.us-east-1.elb.amazonaws.com” is allowed to access my bucket.

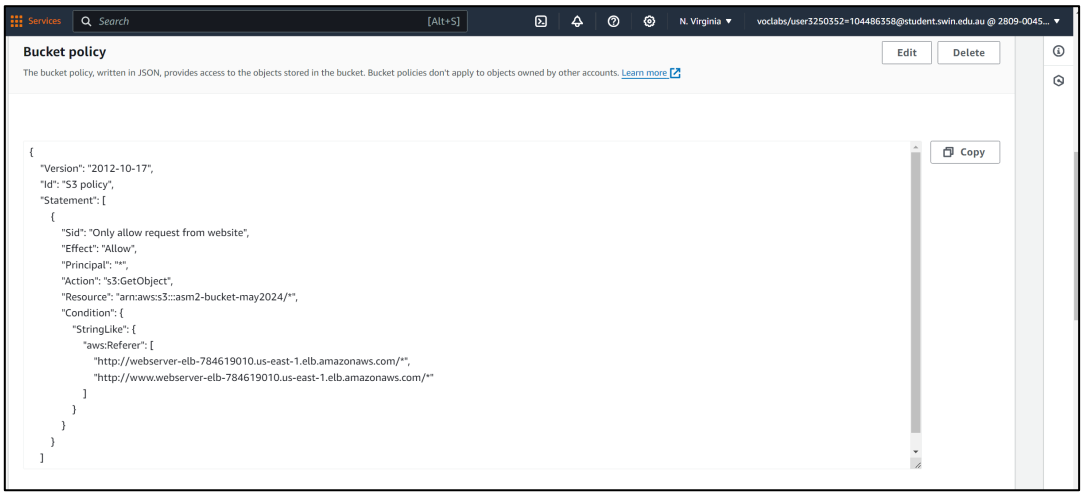

Figure 19: Updated bucket policy

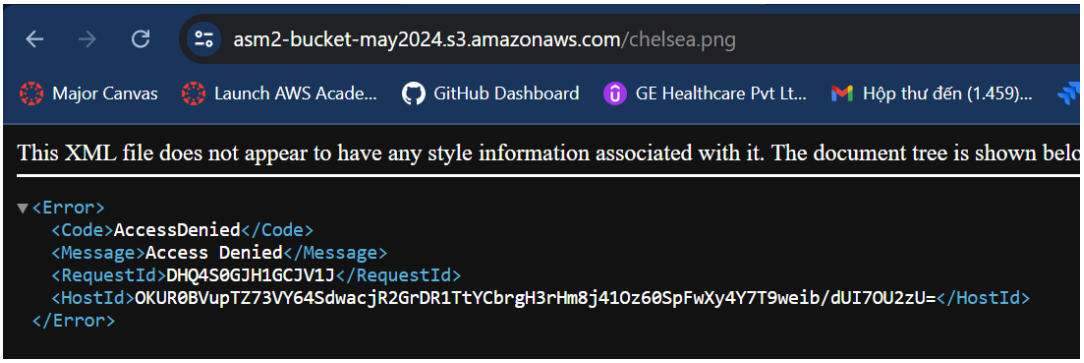

Figure 20: Image is not accessible publicly

F. RDS

Similar to Assignment 1B, in this assignment, we also need an RDS database to store meta-data for uploaded images. The settings for this database stays exactly the same with Assignment 1B, so there is no for changing in configurations. The difference is now instead of access to this database from web server, we need to access it from Dev Server. Therefore, I need to install phpMyAdmin in Dev Server and configure DBServerSG to accept MySQL from DevServerSG.

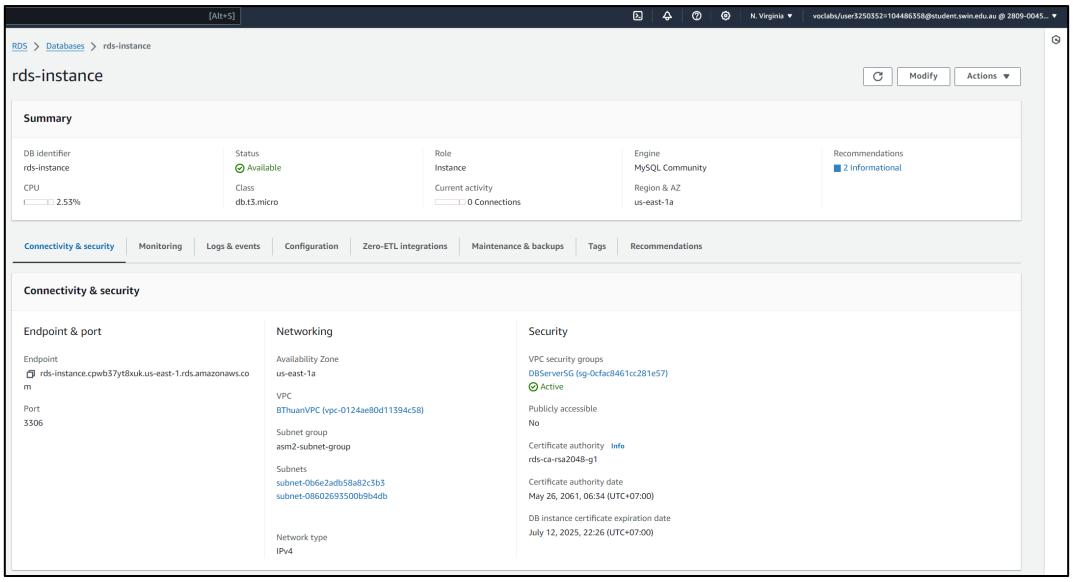

Figure 21: Database configurations similar to assignment 1B: MySQL, locates in “BThuanVPC” and associate with “DBServerSG”

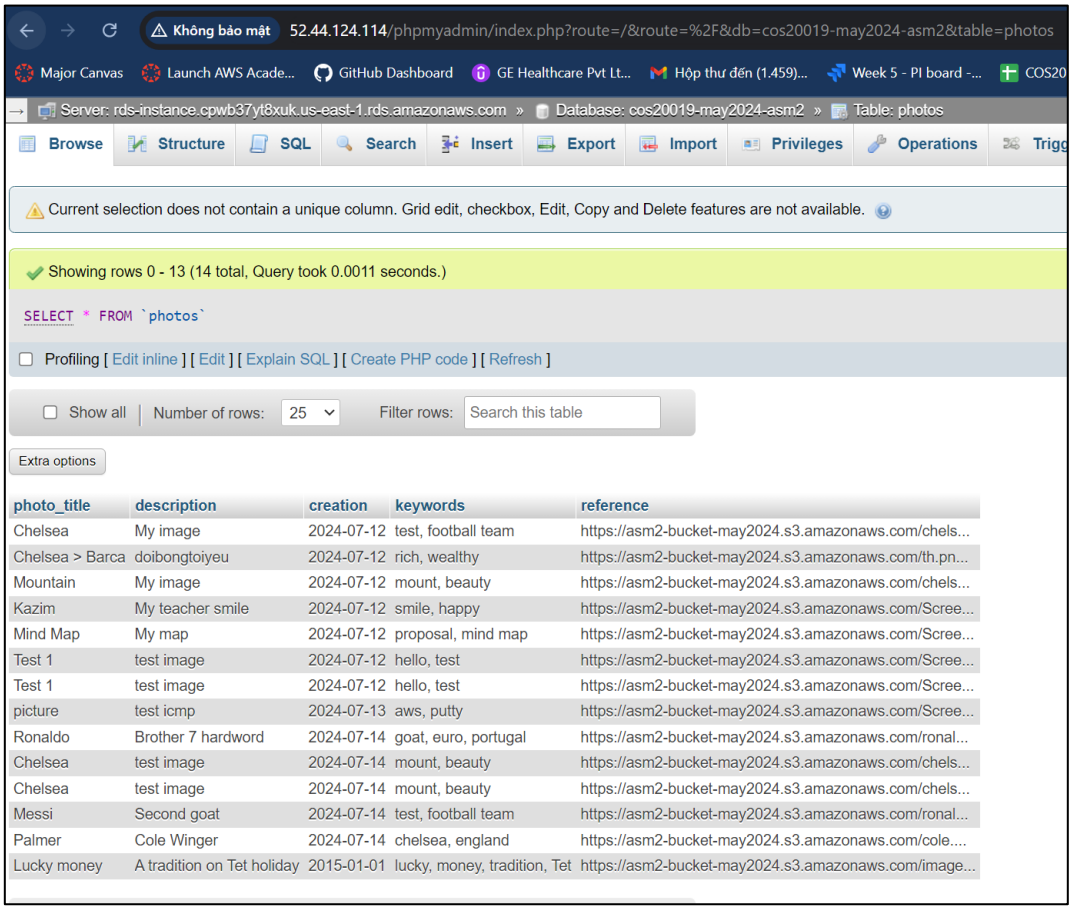

Figure 22: Meta-data are recorded correctly in table “photos”, similar to Assignment 1B

G. Lambda function

In this assignment, it is required to implement a serverless function (Lambda) to resize the newly-uploaded image. Whenever a new image is uploaded, this function will be invoked to resize this image and upload the resized version to the same S3 bucket. As the code for this function is already written in the instruction folder, what I need to do is just upload it to Lambda and configure function settings so that it works properly. This function is called “CreateThumbnail”, run in “Python 3.11”, “arm64” as architecture and has “LabRole” as execution role (see IAM Roles section for details of this role)

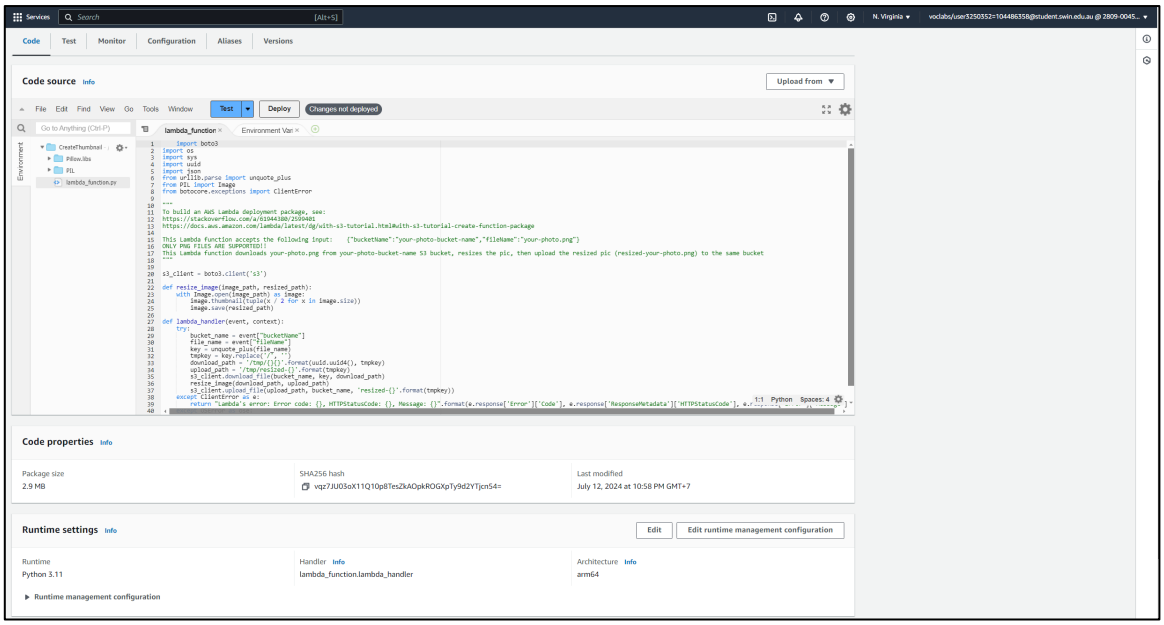

Figure 23: Lambda code and runtime settings

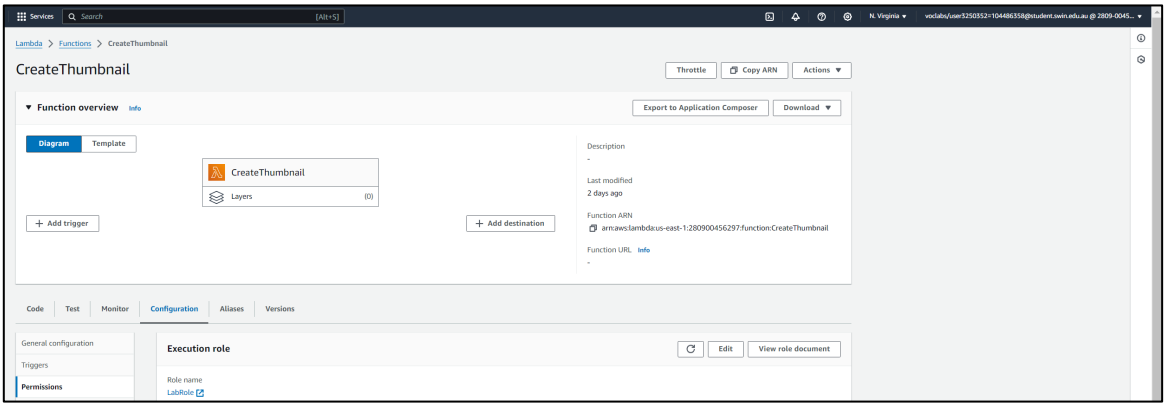

Figure 24: “LabRole” as execution role

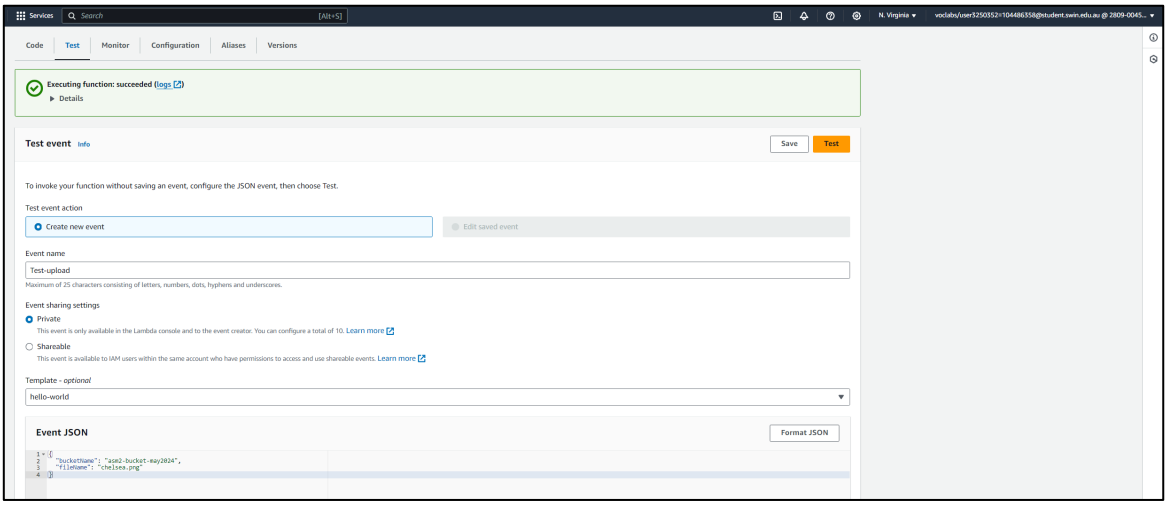

Figure 25: Function runs successfully when testing

H. Elastic Load Balancer (ELB)

To guarantee the high availability of this website, an Application Load Balancer is required to forward traffic to different EC2 instances. This ELB requires a target group contains a list of instances to distribute traffic into and perform health check.

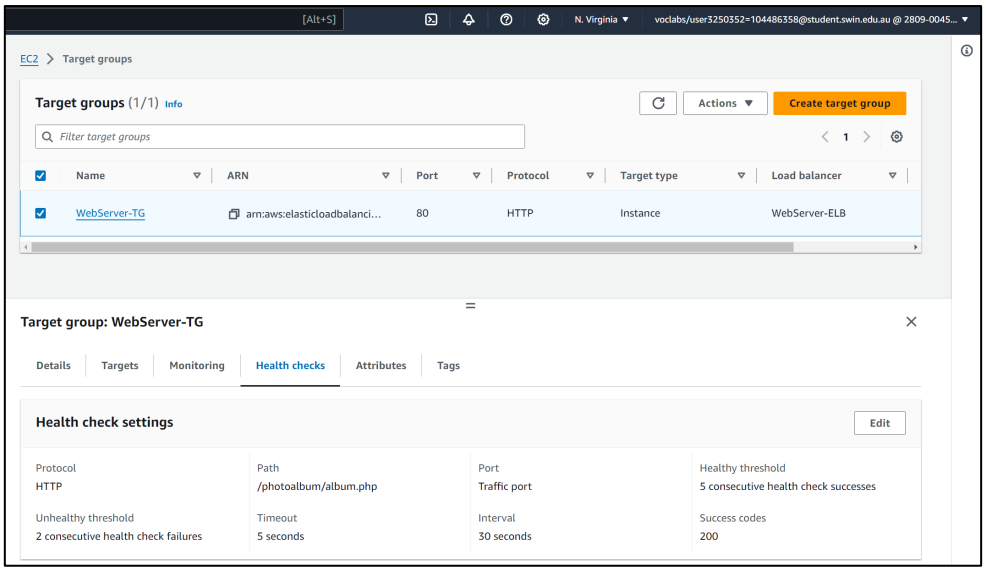

Figure 26: Target Group with health check protocol and path

With this target group, I define listener rule for the ELB. It will listen and perform health check to instances in this target group in port 80 (HTTP).

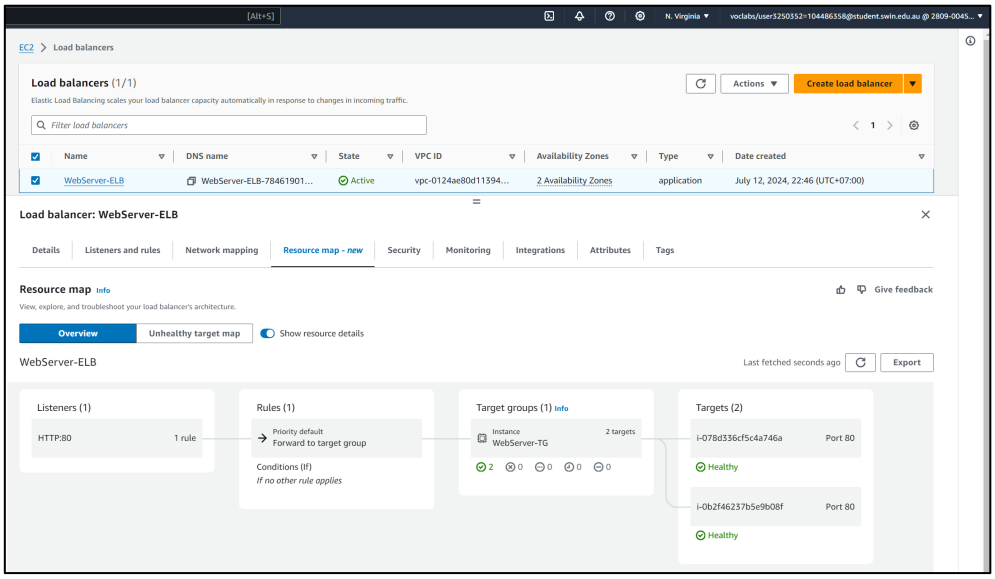

Figure 27: Load Balancer with correct rules and listener

The final ELB will be associated with an Elastic Public IP address and ELB SG as Security Group.

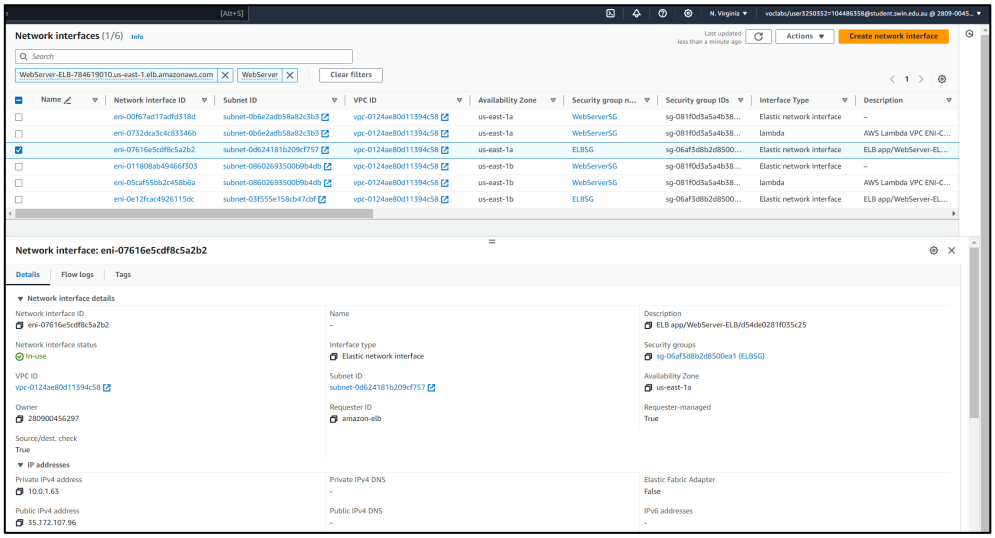

Figure 28: Load Balancer associate with elastic network interface

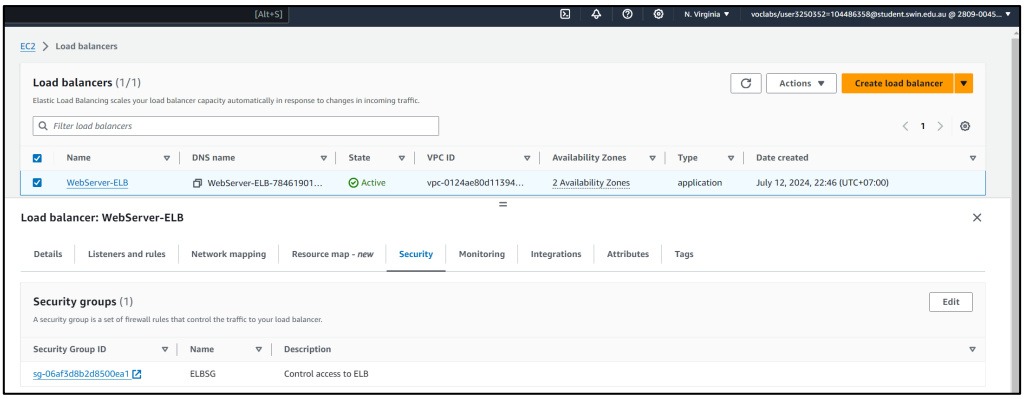

Figure 29: ELB is protected by ELBSG

I. *Auto Scaling Groups (ASG)*

After setting up the ELB, I will add an Auto Scaling Group to my infrastructure. The idea of using this ASG is to automatically deploy enough instances for the web server, so that my website is always available. In order to function properly, an ASG needs a Load Balancer (which is already set up) and a Launch Template.

Launch Template is the template containing configurations and settings, which the ASG will refer to when it set up new instances for my web servers. The Launch Template is created from a custom AMI, or instance image. Just imagine it as an image that capture all settings, software,.. of an instance. This means, the first thing we need to set up is an EC2 instance as sample instance, then create an AMI from it, which later will be used to set up the Launch Template. In this scenario, this EC2 instance is the Dev Server.

First, I deploy the Dev Server in my VPC, in subnet public 2. This instance has similar configuration for a web server instance. I will also install needed software for this website such as Apache, AWS PHP SDK, phpMyAdmin,… into it.

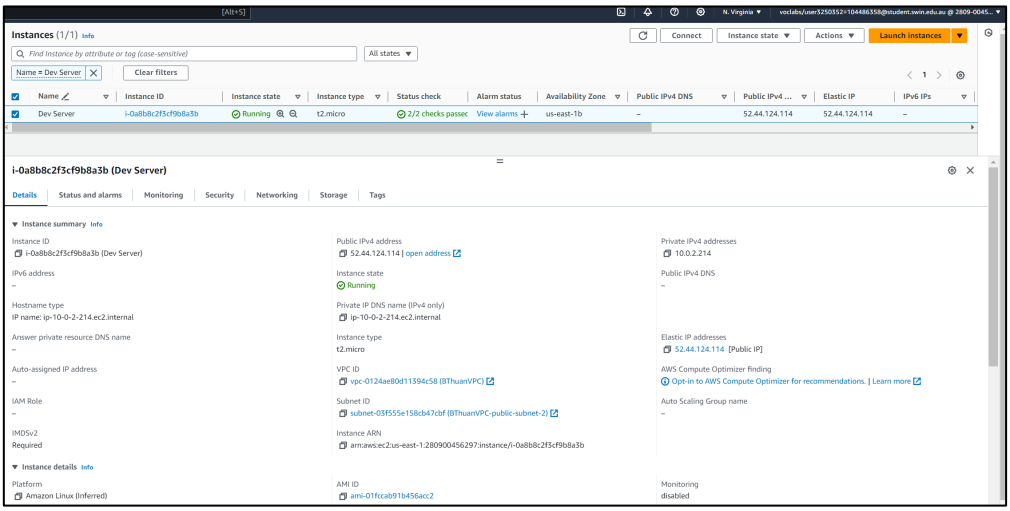

Figure 30: Instance configurations

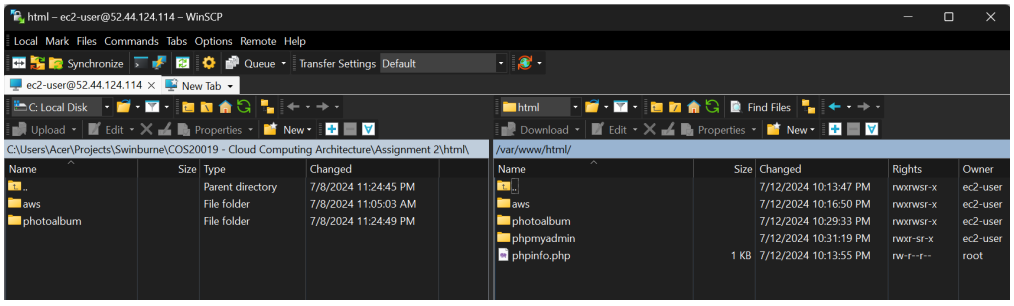

Figure 31: Need software installed on Dev Server

From this instance, I create an AMI, then used this AMI to set up the Launch Template for Web Servers. One thing to remember is setting instance role as “LabRole”, so it can function as expected.

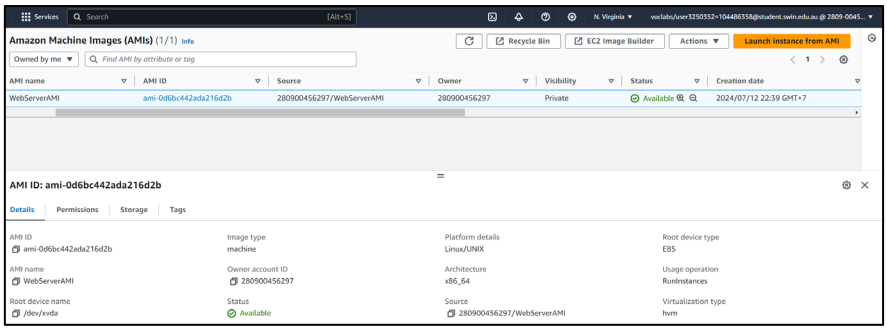

Figure 32: WebServerAMI is created from Dev Server

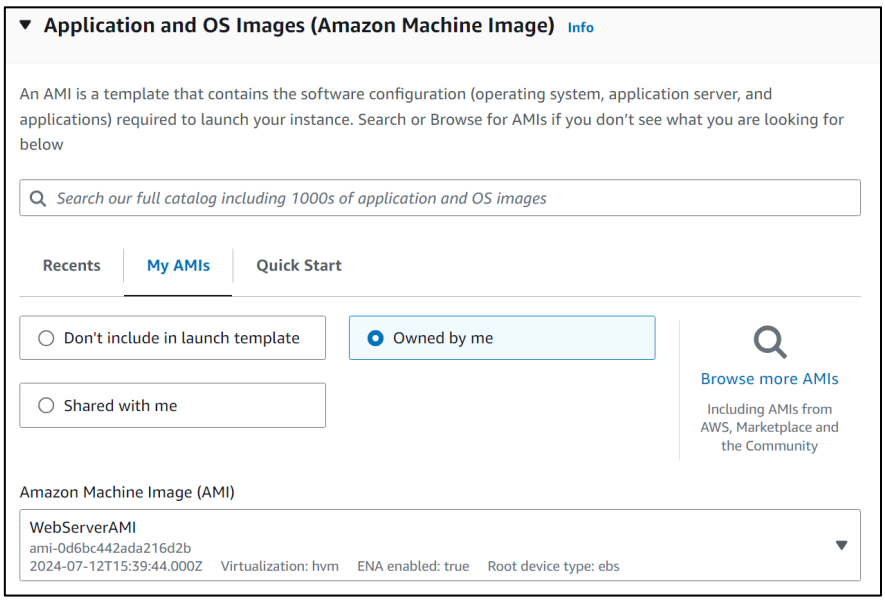

Figure 33: Specify WebServerAMI as AMI when create Launch Template

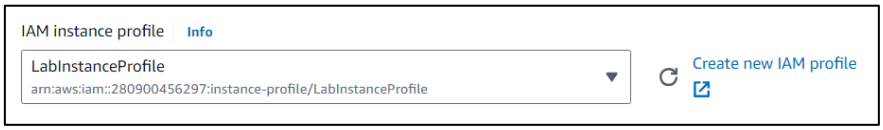

Figure 34: Specify role for this Launch Template as "LabRole"

After setting up the Launch Template, we can use this to create the Auto Scaling Group. This Auto Scaling Group will take the newly-created Launch Template as template to create new instances when any instances become unhealthy. It ensures that there are always at least 2 and maximum 3 instances running at the same time, so that the ELB can distribute traffic properly.

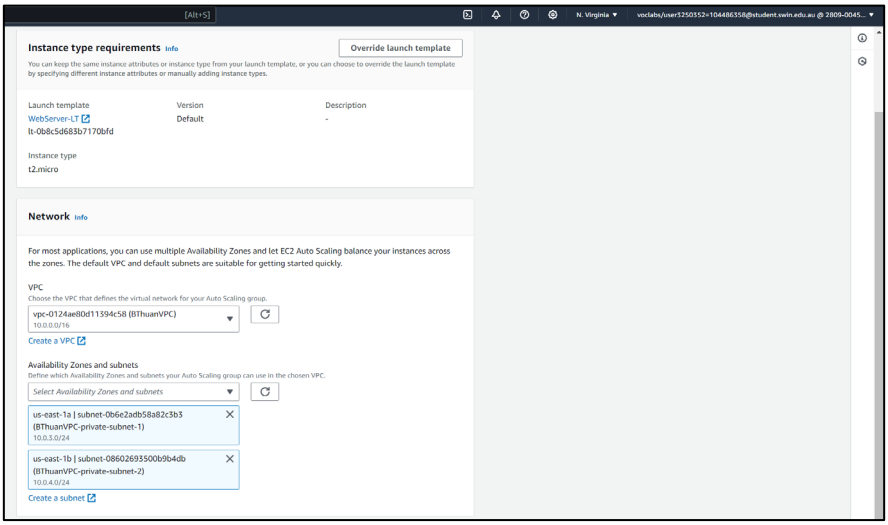

Figure 35: Deploy ASG to private subnets in my VPC

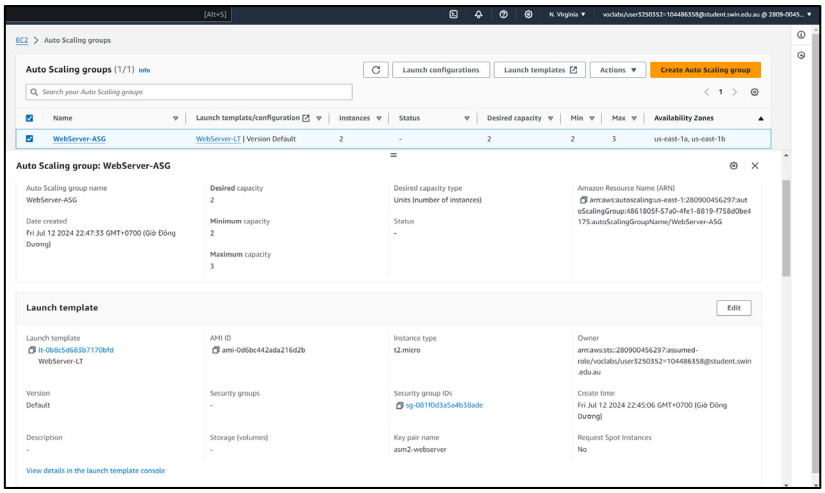

Figure 36: WebServer-ASG is created

Another thing we need to concern about is target tracking policy. In this assignment, it is required to track the request count part target to 30 for this Auto Scaling Group. I achieve that by configuring the target tracking policy when setting up the ASG.

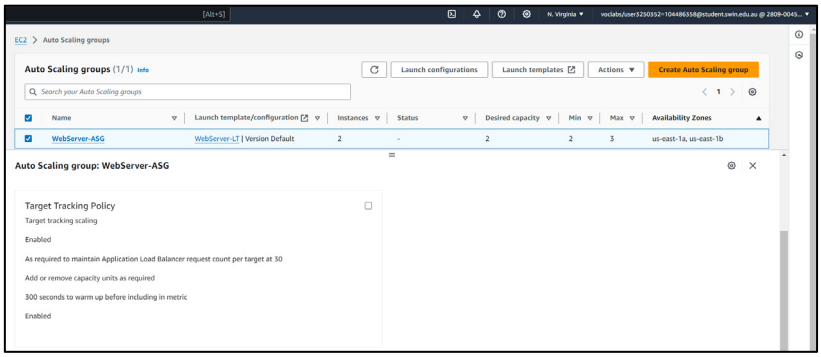

Figure 37: Target tracking policy is configured

After the ASG is set up successfully, the main web servers used for this website will be automatically deployed. These servers have the correct settings created based on the Launch Template, having all needed software and functionalities with “LabRole” as instance role. It is ready to serve the Photo Album users right after launched without any further configurations.

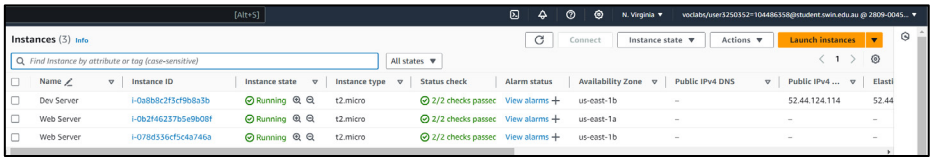

Figure 38: 2 Web Server instances are deployed

III. Functional Requirements

Based on the infrastructure deployed above, the website is set up. The code file has already provided in the assignment instruction file, so all I need to do is updating the constants in “constants.php” to match with the newly-configured infrastructure.

```
// [ACTION REQUIRED] your full name
define('STUDENT_NAME', 'Bui Minh Thuan');
// [ACTION REQUIRED] your Student ID
define('STUDENT_ID', '104486358');
// [ACTION REQUIRED] your tutorial session
define('TUTORIAL_SESSION', 'Wednesday 10:00AM');

// [ACTION REQUIRED] name of the S3 bucket that stores images
define('BUCKET_NAME', 'asm2-bucket');
// [ACTION REQUIRED] region of the above bucket
define('REGION', 'us-east-1');
define('S3_BASE_URL','https://'.BUCKET_NAME.'.s3.amazonaws.com/');

// [ACTION REQUIRED] name of the database that stores photo meta-data (note that this is not the DB identifier of the RDS instance)
define('DB_NAME', 'cos20019-may2024-asm2');
// [ACTION REQUIRED] endpoint of RDS instance
define('DB_ENDPOINT', 'rds-instance.cpwb37yt8xuk.us-east-1.rds.amazonaws.com');
// [ACTION REQUIRED] username of your RDS instance
define('DB_USERNAME', 'main');
// [ACTION REQUIRED] password of your RDS instance
define('DB_PWD', 'buiminhthuan12345');

// [ACTION REQUIRED] name of the DB table that stores photo's meta-data
define('DB_PHOTO_TABLE_NAME', 'photos');
// The table above has 5 columns:
// [ACTION REQUIRED] name of the column in the above table that stores photo's titles
define('DB_PHOTO_TITLE_COL_NAME', 'photo_title');
// [ACTION REQUIRED] name of the column in the above table that stores photo's descriptions
define('DB_PHOTO_DESCRIPTION_COL_NAME', 'description');
// [ACTION REQUIRED] name of the column in the above table that stores photo's creation dates
define('DB_PHOTO_CREATIONDATE_COL_NAME', 'creation');
// [ACTION REQUIRED] name of the column in the above table that stores photo's keywords
define('DB_PHOTO_KEYWORDS_COL_NAME', 'keywords');
// [ACTION REQUIRED] name of the column in the above table that stores photo's links in S3
define('DB_PHOTO_S3REFERENCE_COL_NAME', 'reference');

// [ACTION REQUIRED] name (ARN can also be used) of the Lambda function that is used to create thumbnails
define('LAMBDA_FUNC_THUMBNAILS_NAME', 'CreateThumbnail');
```

Figure 39: Updated constants for website

A. Website accessible via ELB and photos, metadata are displayed correctly.

My website is now accessible via ELB's DNS with the link: http://webserver-elb-784619010.us-east-1.elb.amazonaws.com/photoalbum/album.php. Because ELB has its own Elastic IP, the DNS stays the same, so this link is always consistent. With accurate ASG settings, the website is always available and shows consistent content every time we use it. In my website, we can see all uploaded photos and its meta-data displayed in a table.

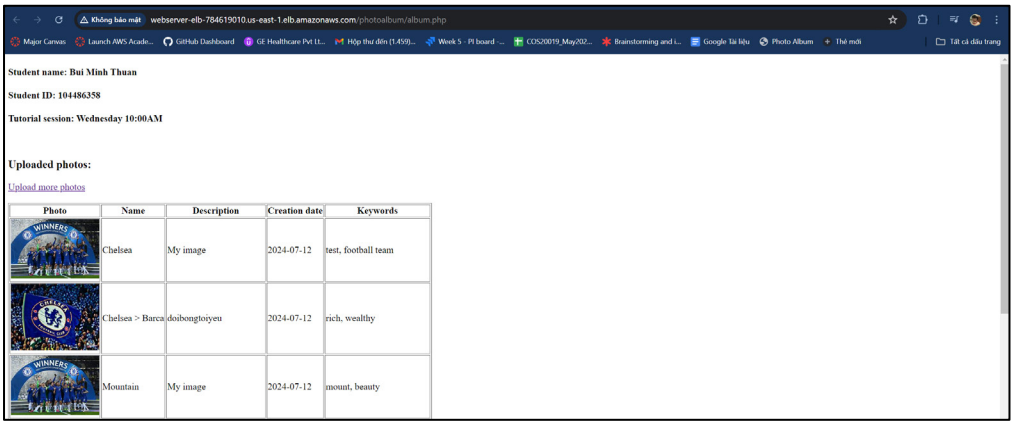

Figure 40: Website accessible with correct information displayed

B. Photo are uploaded properly to S3 and RDS, and also resized by RDS

A significant features of this website is being able to upload images user want to its database. The images will be stored in S3 bucket, its meta-date, include image link are stored in RDS, while resized version of this photo created by Lambda will be in the same S3 bucket. Details about S3, RDS, and Lambda have already mentioned in above sections.

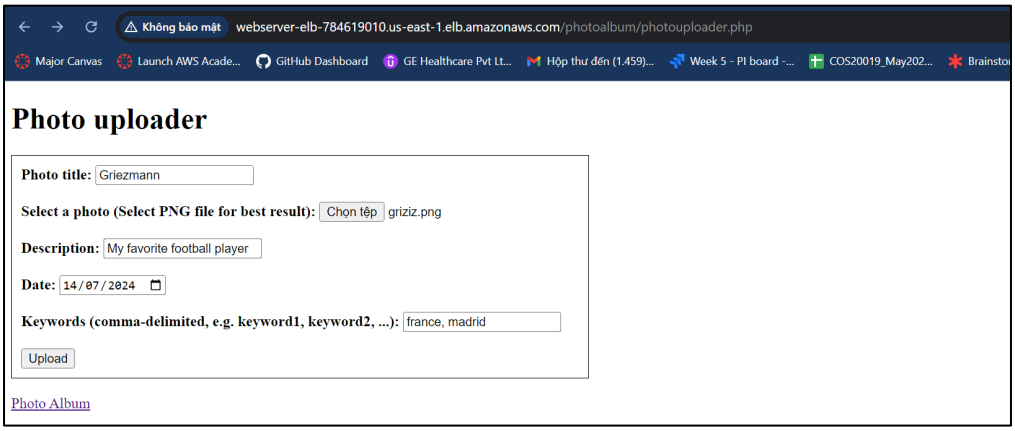

Figure 41: Upload images with a form in "photouploader.php"

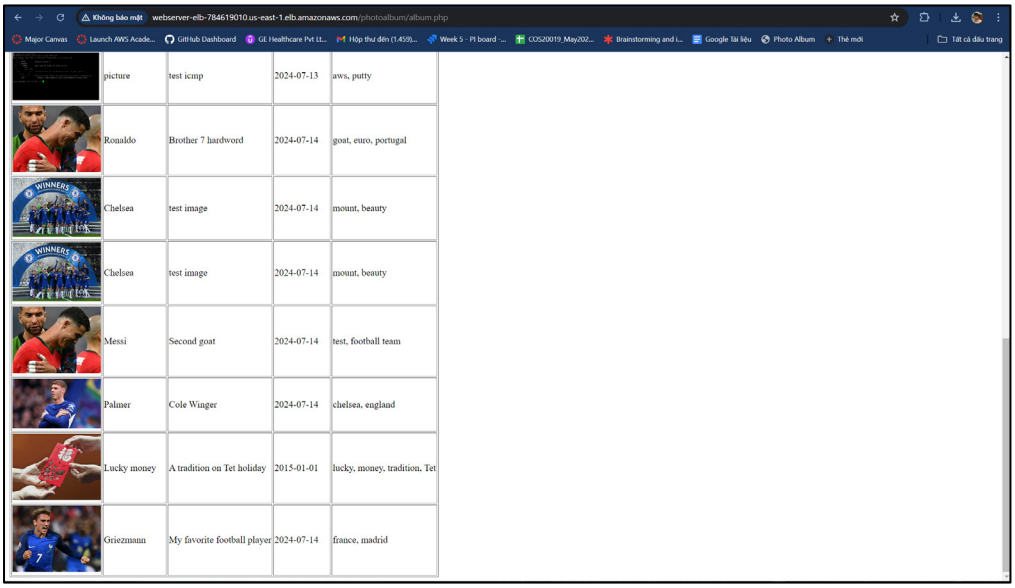

Figure 42: Newly added image is displayed in the bottom

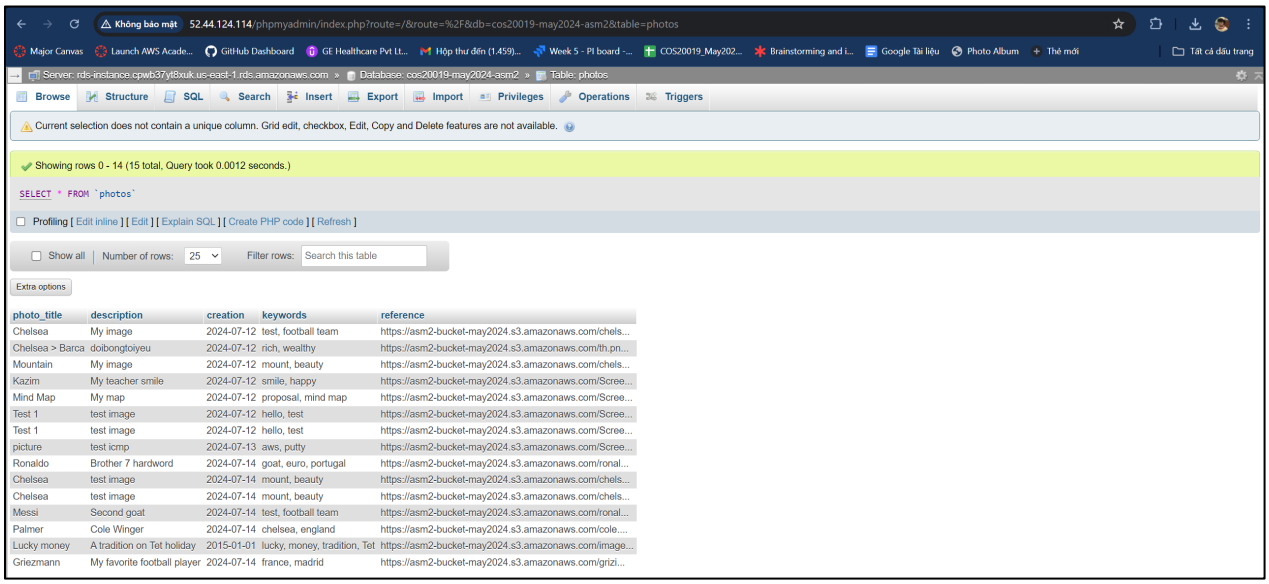

Figure 43: Meta-data is recorded in RDS database

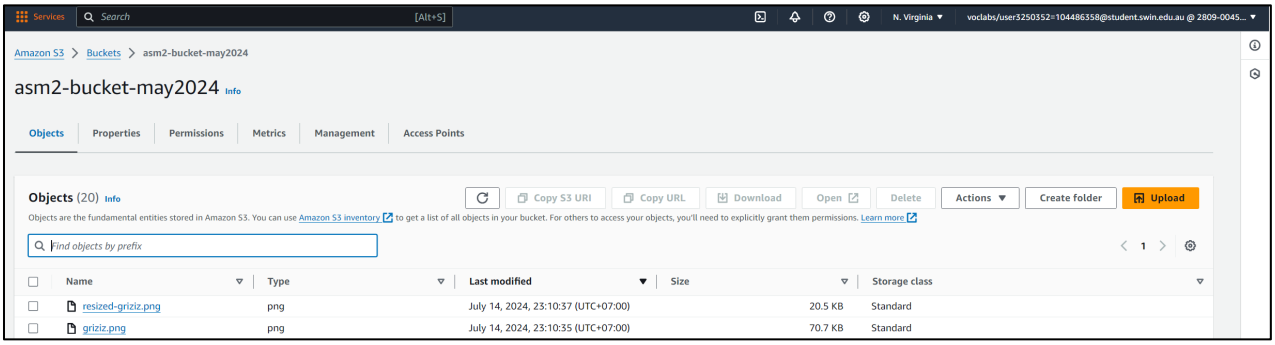

Figure 44: Images are stored in S3.

The photo I just uploaded, “griziz.png”, is automatically taken to create a resized version called “resized-cole.png” and uploaded to the same bucket. You can see the image above the verify that both 2 images exist, and resized version is significantly lighter (12.3 KB compared to 42.3KB)

IV. Conclusion

Throughout this report, all my works in Assignment 2 – Unit COS20019 Cloud Computing Architecture are reported. Based on the given requirements of the website, I have deployed everything from scratch, from setting up infrastructure to website functionalities deployment.

References

[1] Amazon Web Services. (2012, Oct 17). “Examples of Amazon S3 bucket policies”. Amazon Simple Storage Service Documentation. https://docs.aws.amazon.com/AmazonS3/latest/userguide/example-bucket-policies.html (accessed July. 14, 2024).