COS20019 – Assignment 1B

Creating and deploying Photo Album website onto a simple AWS infrastructure

Bui Minh Thuan
104486358
Tutorial Time: Wednesday 10:00AM
Date of Submission: 16/06/2024

I. Introduction

This report covers my works for Assignment 1B of Unit Cloud Computing Architecture (May 2024). The assignment focuses on using AWS services to host a website completely on cloud. All the tasks are performed on "Learner Lab" from AWS Cloud Academy. The website can be accessed via this link: http://ec2-44-221-204-226.compute-1.amazonaws.com/cos20019/photoalbum/album.php (44.221.204.226 is the Elastic IP)

II. Infrastructure Deployment

In the first part, the assignment focuses on setting up the structure and services that will be used in the website on AWS. These services including: VPC, EC2, RDS as infrastructure and Security Groups, Network ACL for security. Details and specifications of them are illustrated in the given instruction file. A diagram is also included to demonstrate the overall structure for the website deployment.

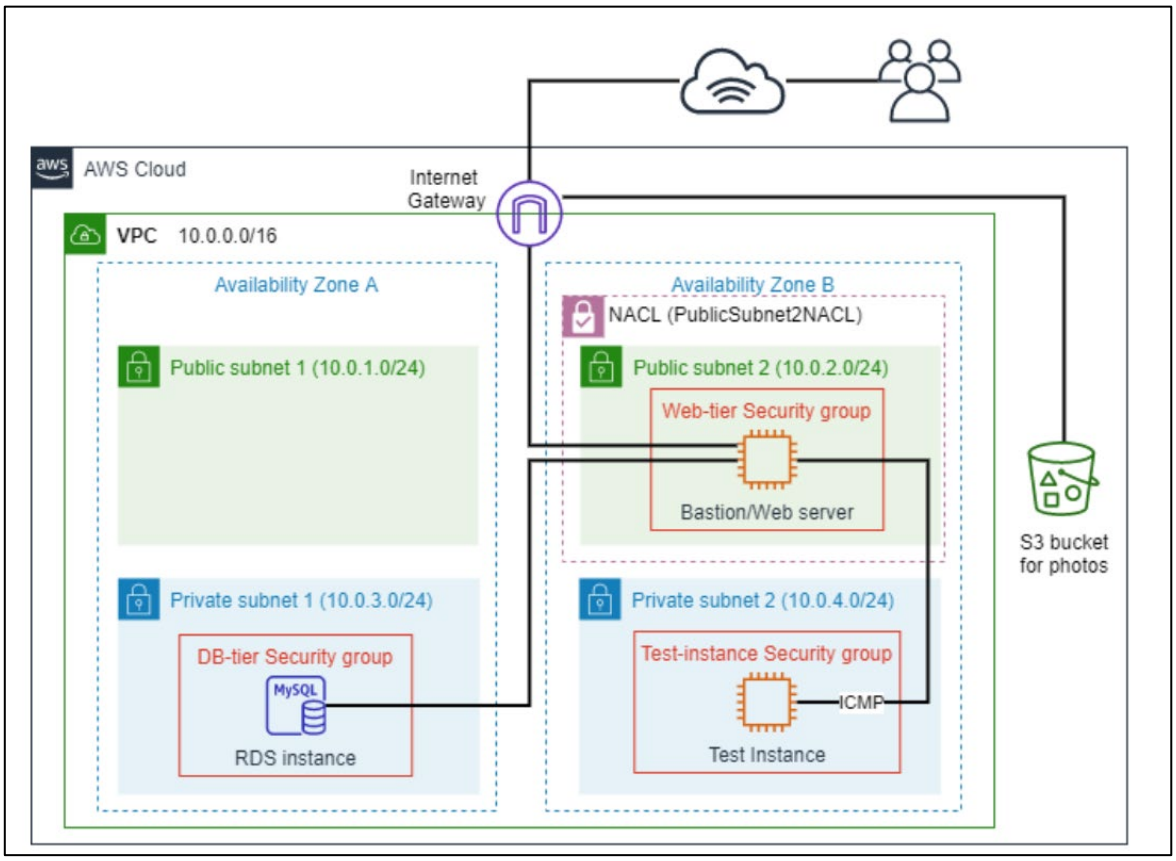

Figure 1: Infrastructure diagram

A. VPC

Starting from Assignment 1B, Default VPC is no longer used, instead, in each task, we need to manually create the own VPC. In this Assignment, the created VPC will have 2 Availability Zones and 4 Subnets (2 public, 2 private). For route tables, 2 new route tables created, 1 for public and 1 for private. For the gateway, only use Internet Gateway.

This step is not difficult, but we need to be careful when set up rules for the routable as well as assign IP to each subnet, ensuring each subnet gets the accurate range of IP as required in the architecture diagram, as well as subnet corresponds to the correct route tables.

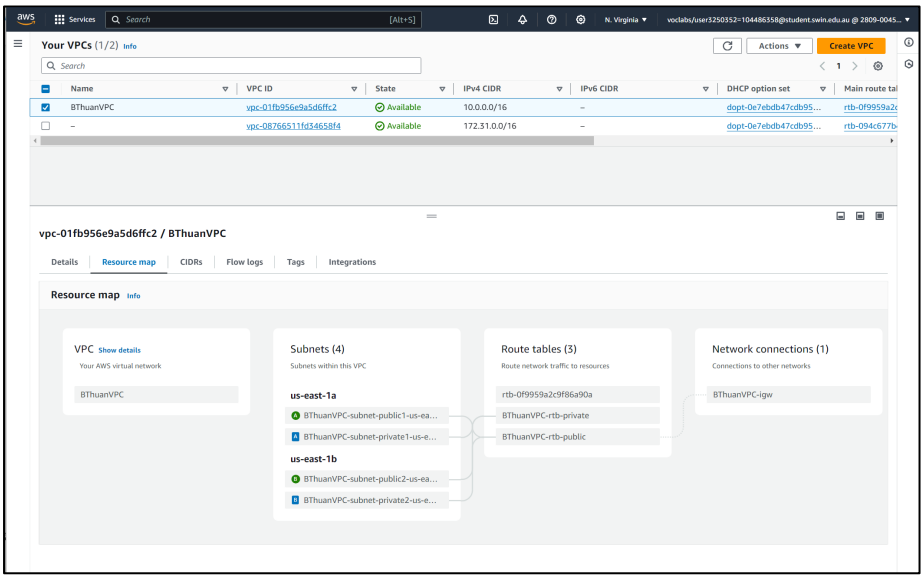

Figure 2: VPC with 2 public subnets and 2 private subnets

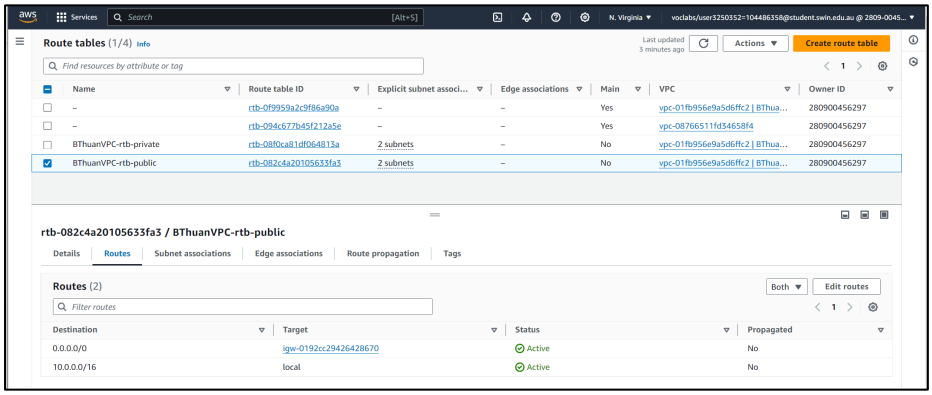

Figure 3: Public Route Table

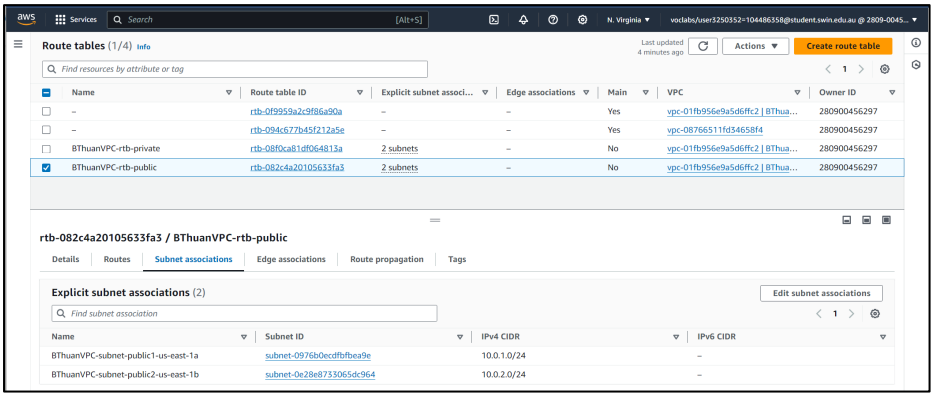

Figure 4: Public subnet 1,2 associates with Public Route Table

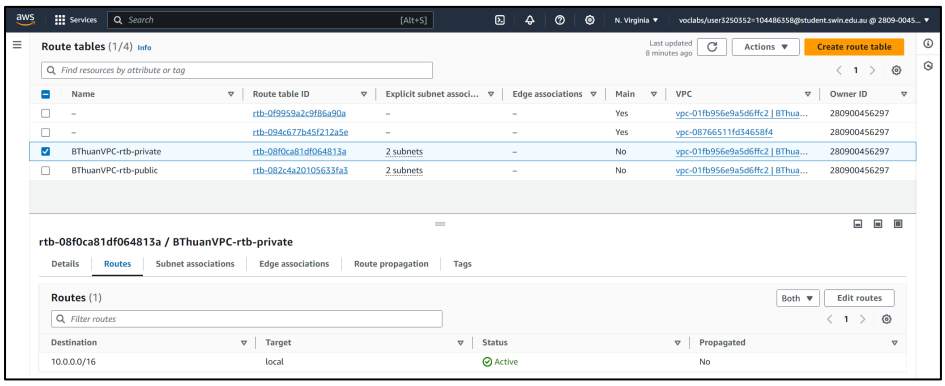

Figure 5: Private Route Table

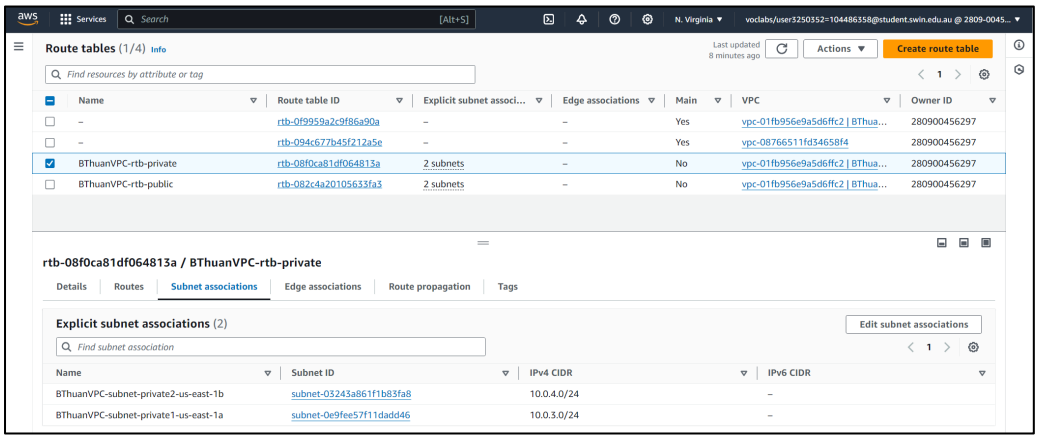

Figure 6: Private subnet 1,2 associates with Private Route Table

B. *Security Groups*

For instance level, 3 Security Groups are required, corresponding to 3 tiers of instance in the assignment's infrastructure. Their inbound rules are specified based on the usage of instance associate with each one of them:

School of Science, Computing and Engineering Technologies — Swinburne University of Technology

Security group name	Protocols	Source
TestInstanceSG	All traffic	Anywhere
WebServerSG	HTTP (80), SSH (22)	Anywhere
	ICMP	TestInstanceSG
DBServerSG	MySQL (3306)	WebServerSG

Figure 7: Security Groups Specification

1) TestInstanceSG: This security group is for Test-instance tier. The instance in this tier is in a private subnet (no public IP), and its inbound rule allows all traffic access from anywhere:

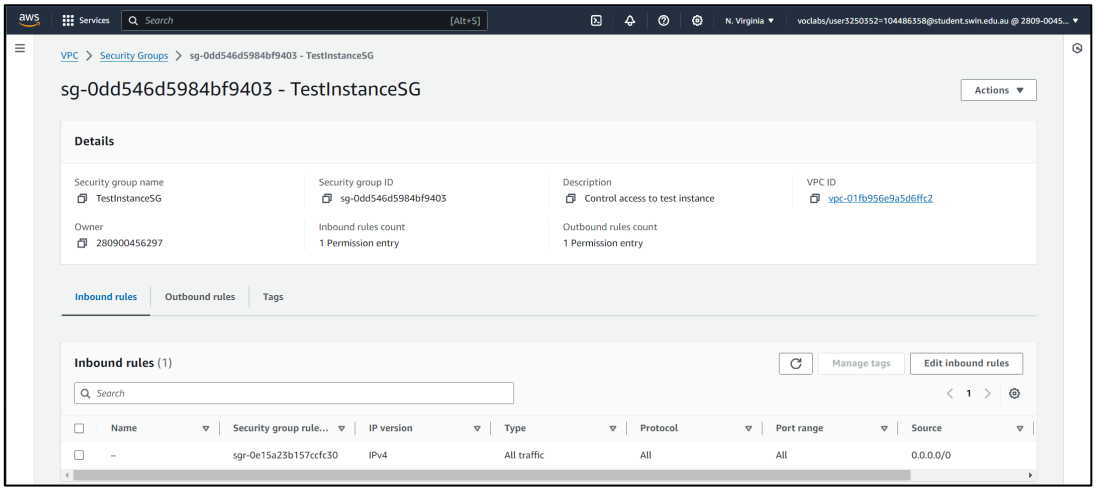

Figure 8: TestInstanceSG allows all traffic from everywhere

2) WebServerSG: The second SG is WebServerSG, which is used for Web Server tier instance. Instance in this tier is in public subnet, and it only permits HTTP, SSH for web request from everywhere, while allow ICMP from instance in TestInstanceSG

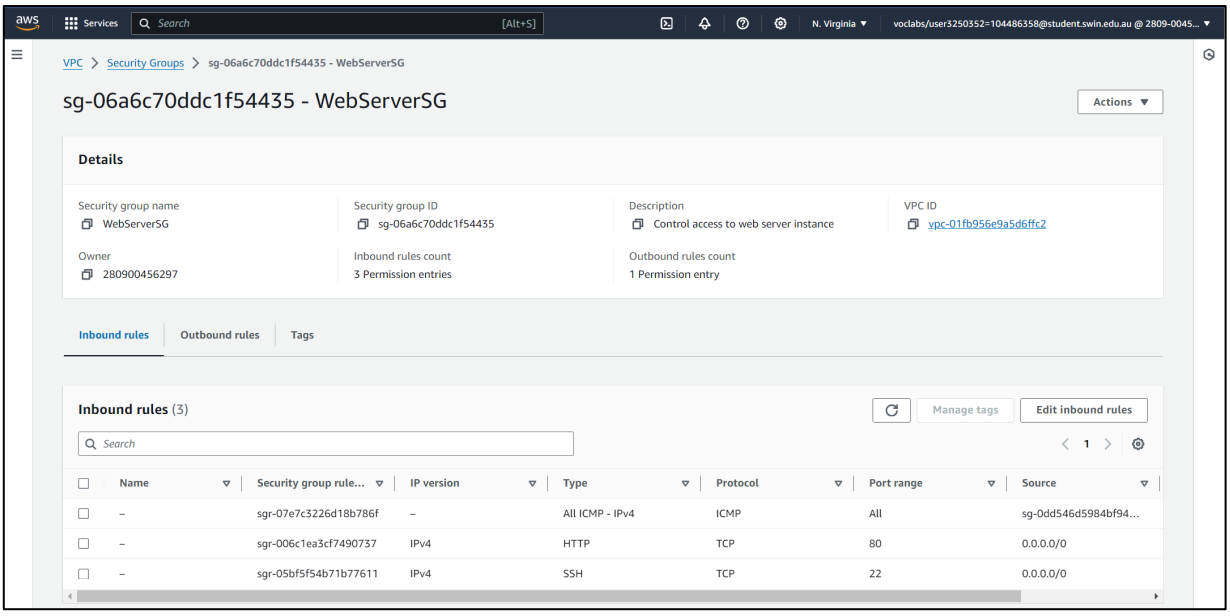

Figure 9: WebServerSG only allows ICMP from TestInstanceSG

3) DBServerSG: DBServer is the last SG in this architecture, used for database only. Database instance is put in a private subnet, and only the web server is allowed to access it via port 3306 (for MySQL).

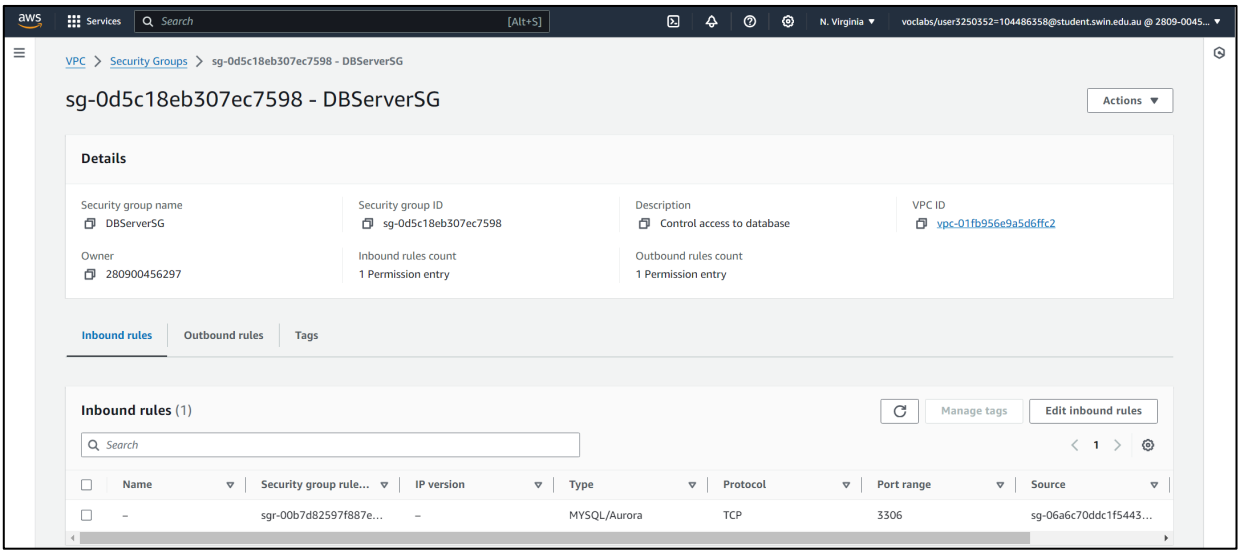

Figure 10: DBServerSG only allows connection via port 3306 from web server

C. EC2 Virtual Machine

The next step is setting up EC2 instances. In this architecture, 2 instances are required, 1 of them is for the main web server, another one is for testing only. All of them must be in the newly-created VPC and associate with the right SG created previously.

1) Bastion/Webserver Instance: This instance is used to host the "Photo Album" application. The configurations for this instance remains the same with Assignment 1A (same AMI, type, user data,…), but the public IP now needs to be reserved, so the website URL can stay consistent. In terms of network and security, this instance will be in "BThuanVPC-vpc", "public-subnet-2" and associate with "WebServerSG" SG.

You can see in the figures below: AMI is Amazon Linux 2, the type is t2.micro. The web server is in "BThuanVPC" VPC, the VPC created for this assignment. It also has an Elastic IP reserved, so the URL for web server will remains consistent overtime.

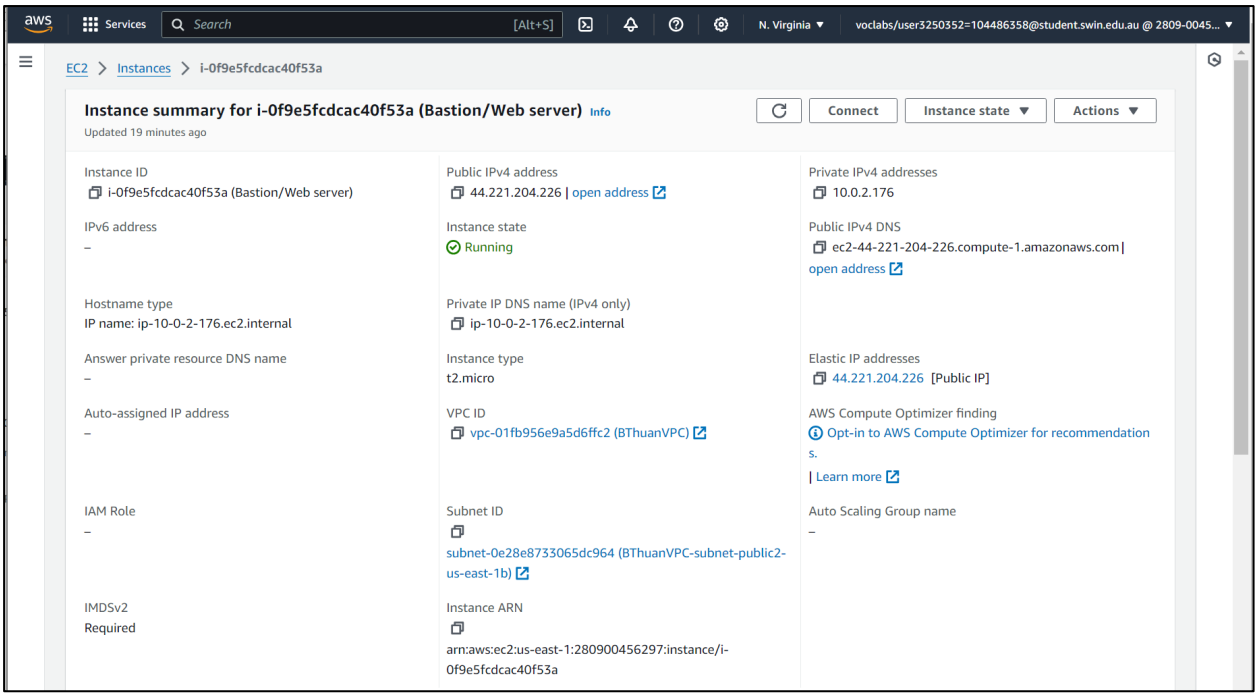

Figure 11: Instance configurations

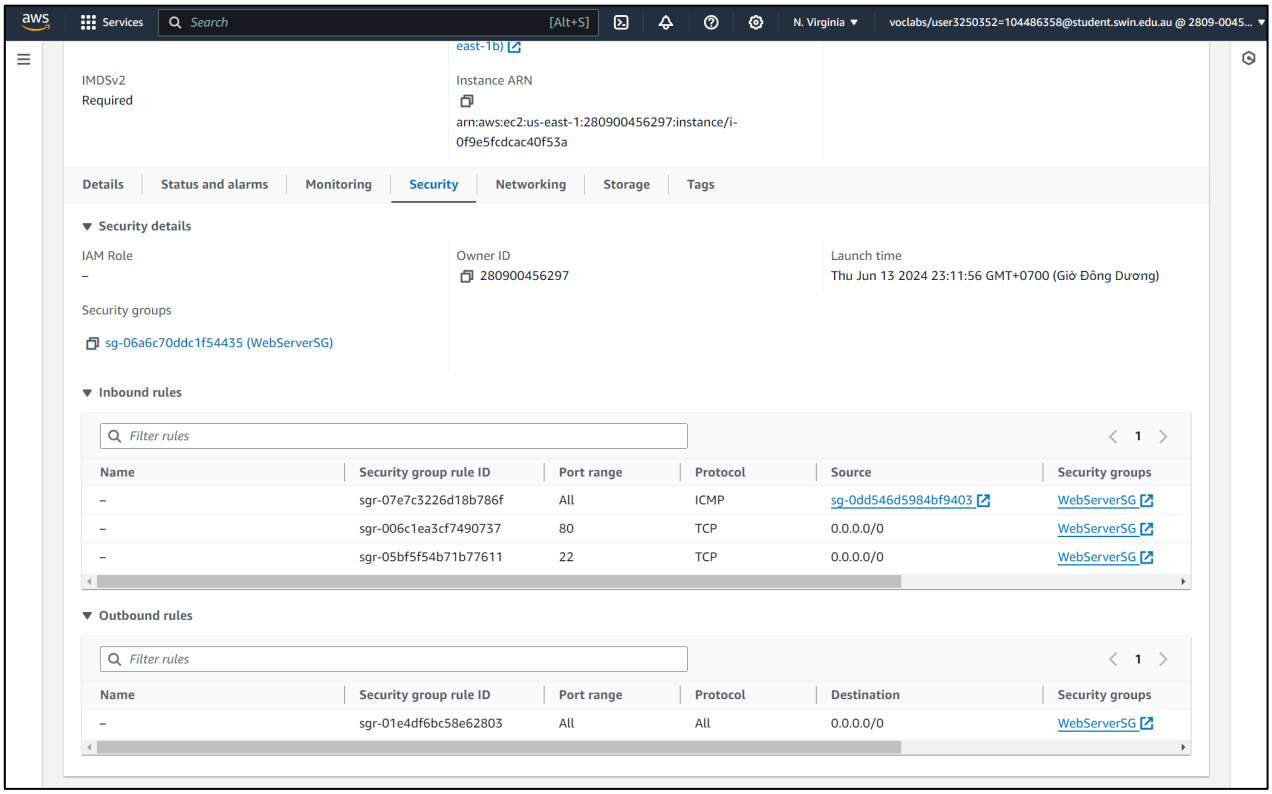

Figure 12: Instance is protected by WebServerSG

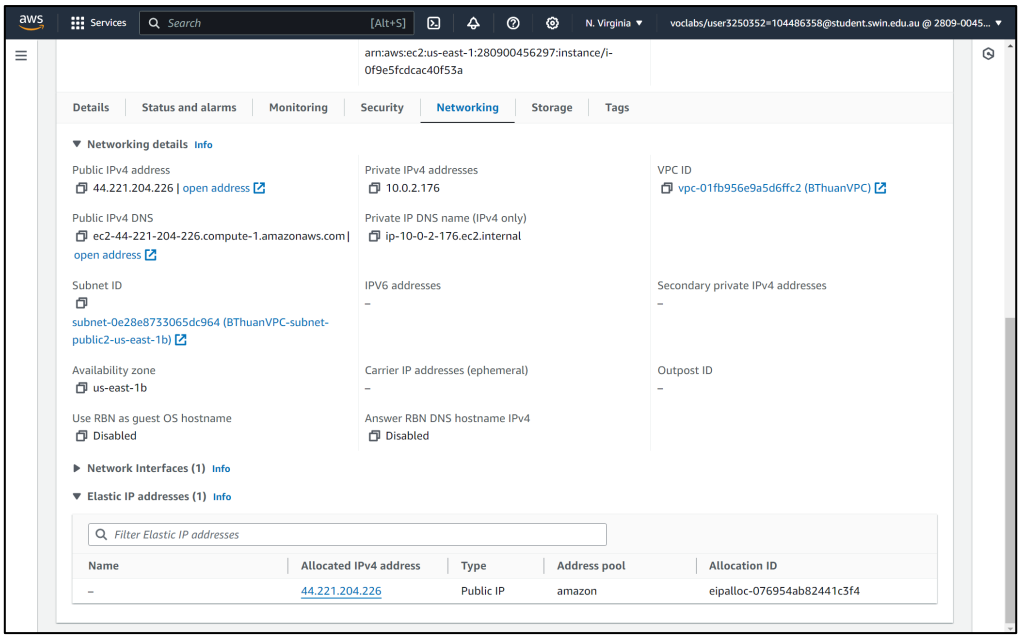

Figure 13: Instance networking settings (with Elastic IP reserved)

2) TestInstance: Beside the main instance, this architecture also have a Test Instance used for demonstration only. The configurations for this instance are not important, because there will be nothing deployed here, instead, this step focuses on connection between web server instance and the test instance. In this case, the web server will be used like a bastion host. I will access to the web server via SSH, then connect to the Test instance from there.

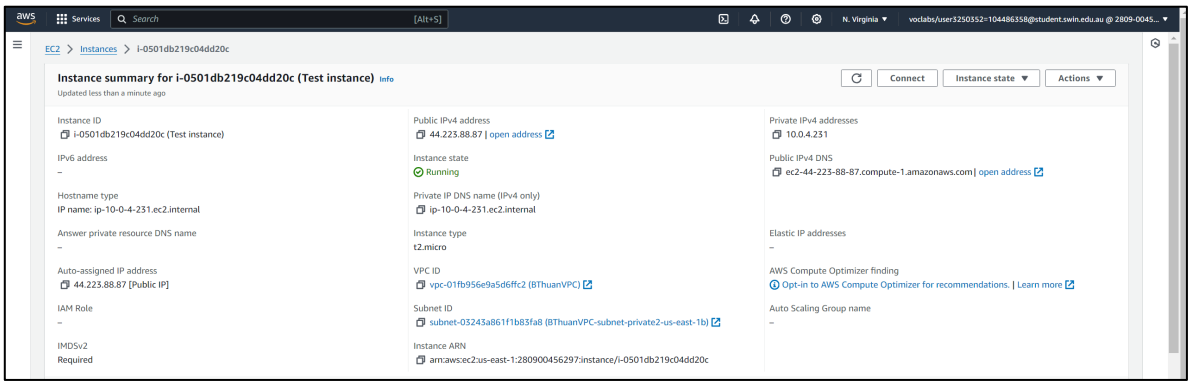

Figure 14: TestInstance is in private subnet 2 (us-east-1b)

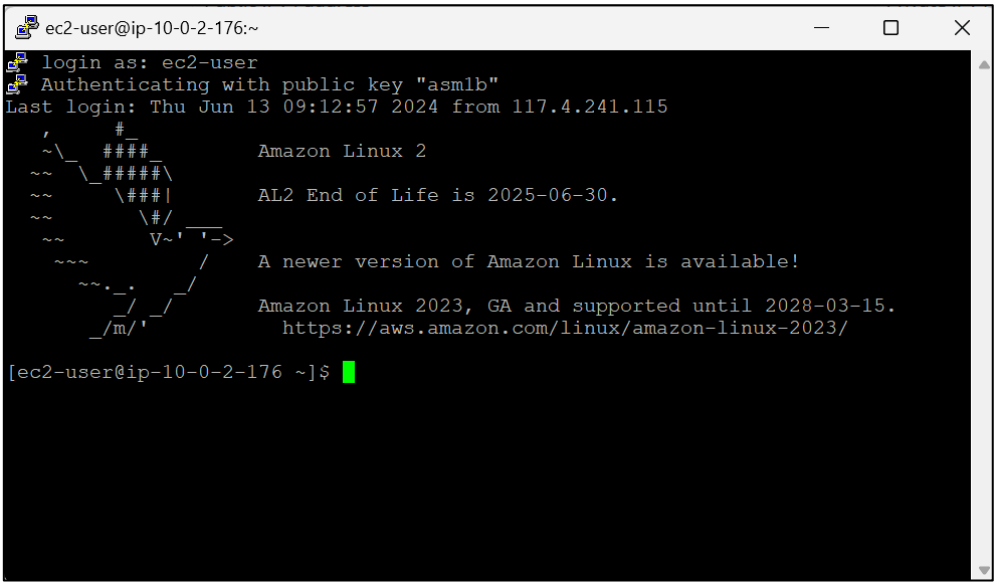

Figure 15: SSH access to Bastion Server (10.0.2.176)

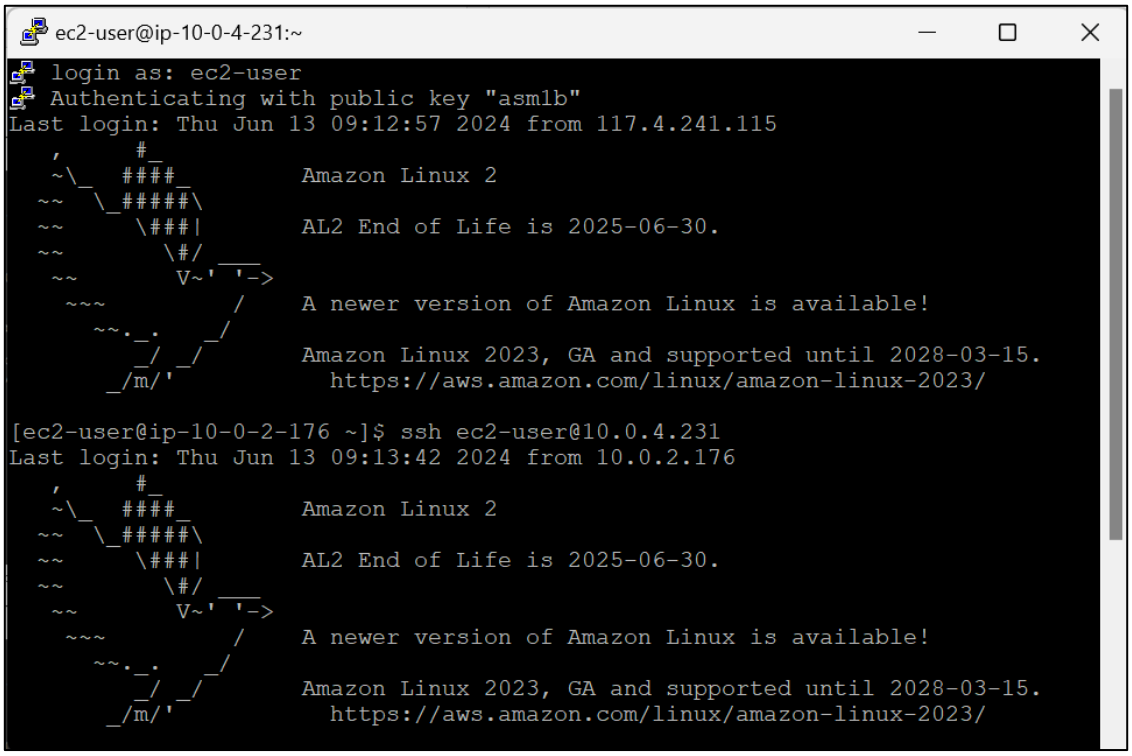

Figure 16: Access to TestInstance (10.0.4.231) from web server (bastion host)

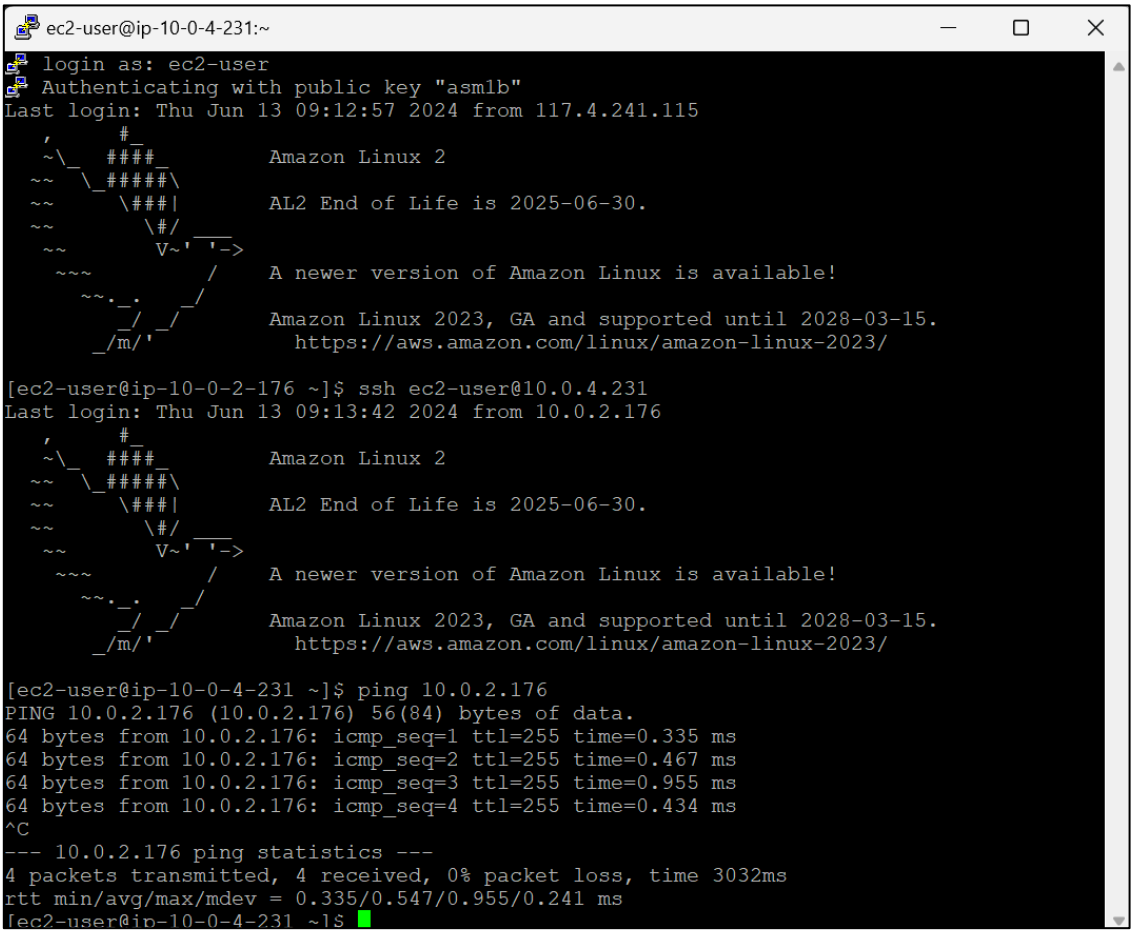

Figure 17: Ping to web server (10.0.2.176) from Test instance

The difficulty here is how the web server should access the Test instance. The simplest way to achieve this is storing the key pair on the web server instance, but this is not recommended due to security concerns. Instead, the better approach is “agent forwarding” with Pageant [1]. This is supported by Putty so the web server can get the key pair without the need to store it directly. Another thing to do is making sure the key pair file is “.ppk”. “.pem” is not supported by Putty.

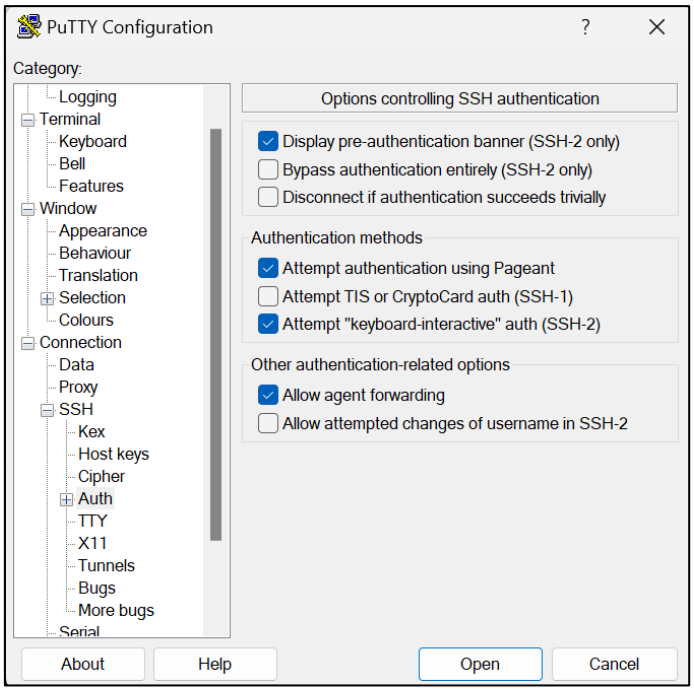

Figure 18: Enable agent forwarding

D. RDS Database

After setting up the web server, now continue with the database. In this case, RDS is chosen as the database service, with MySQL as the engine. The database should be secured, so it will be put in a private subnet, and only the web server can connect to it directly. I achieve this by setting up a subnet group in the VPC for this assignment.

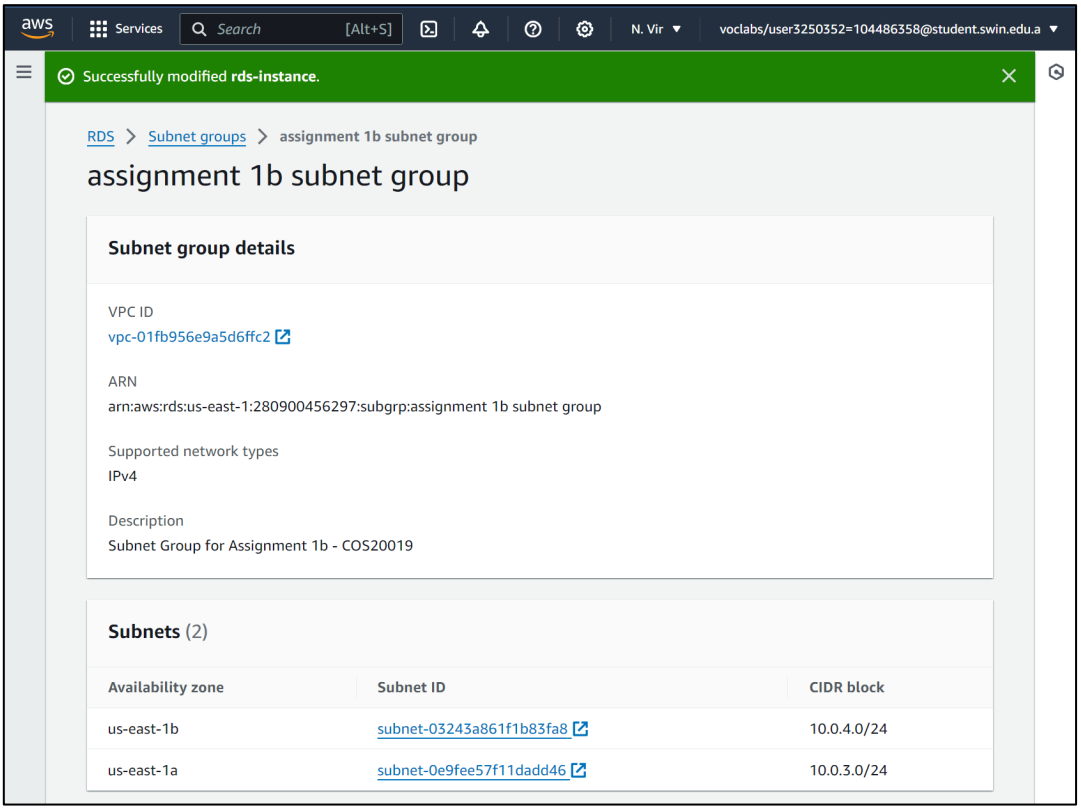

Figure 19: Setting up subnet group

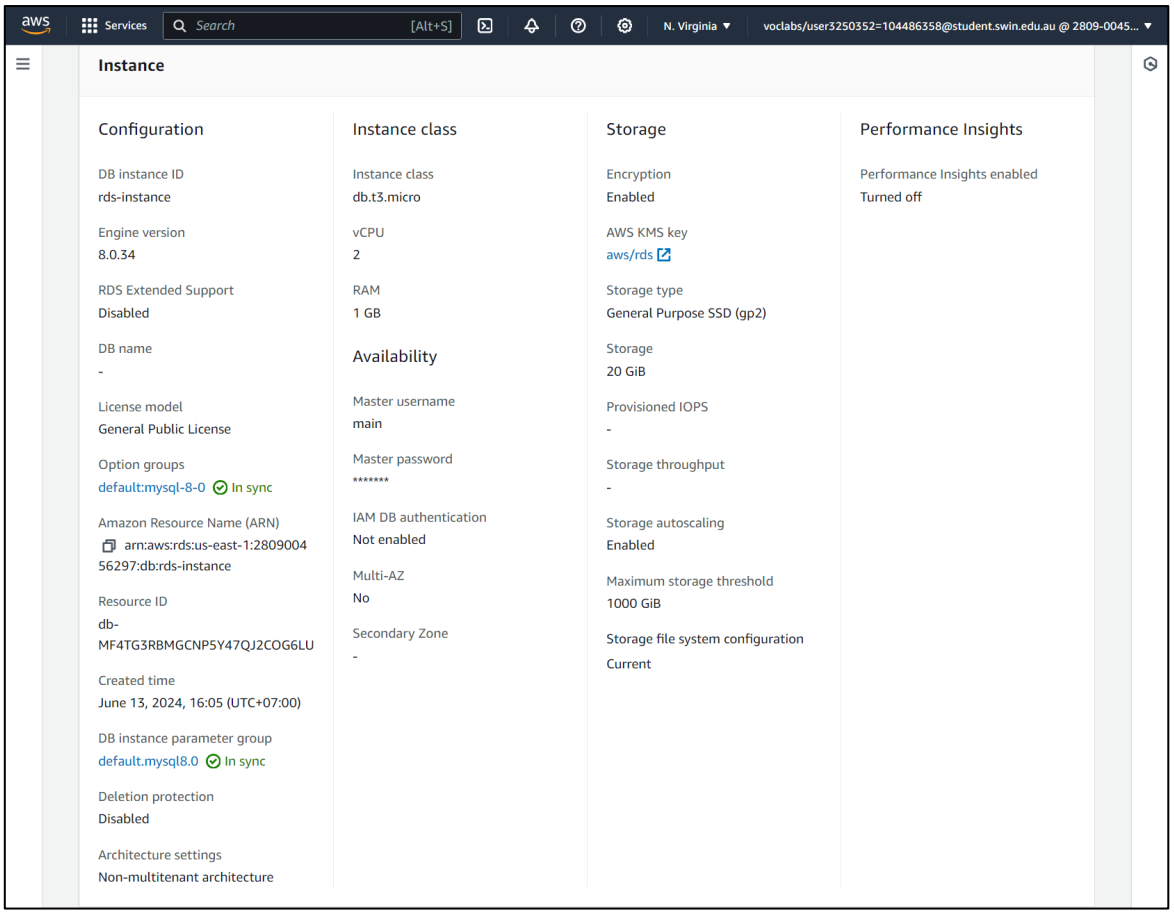

Figure 20: Database configurations

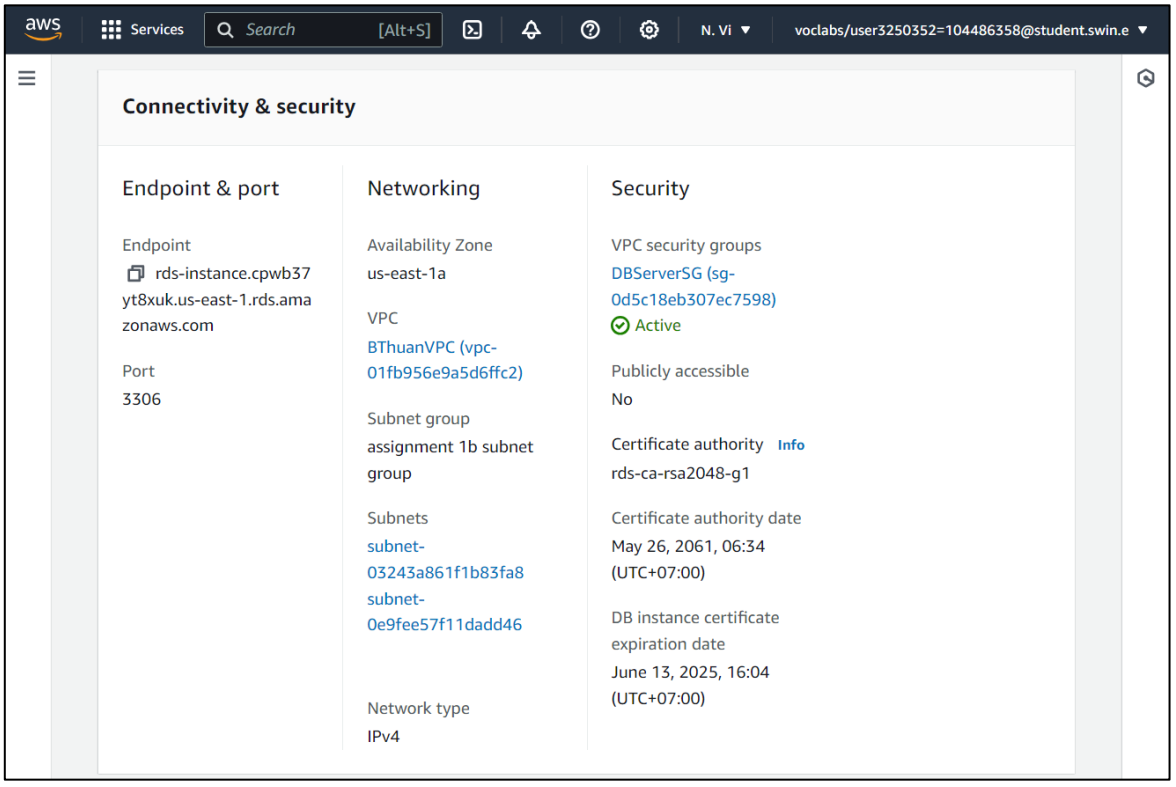

Figure 21: Database networking configurations

The database needs to be secured, but we still needs access to use it. To do that, I will install phpMyAdmin on the web server instance, then use it to access to the database. In this assignment, it is required to create a table called “photos” storing information relating to photos in S3 bucket, which will be set up later.

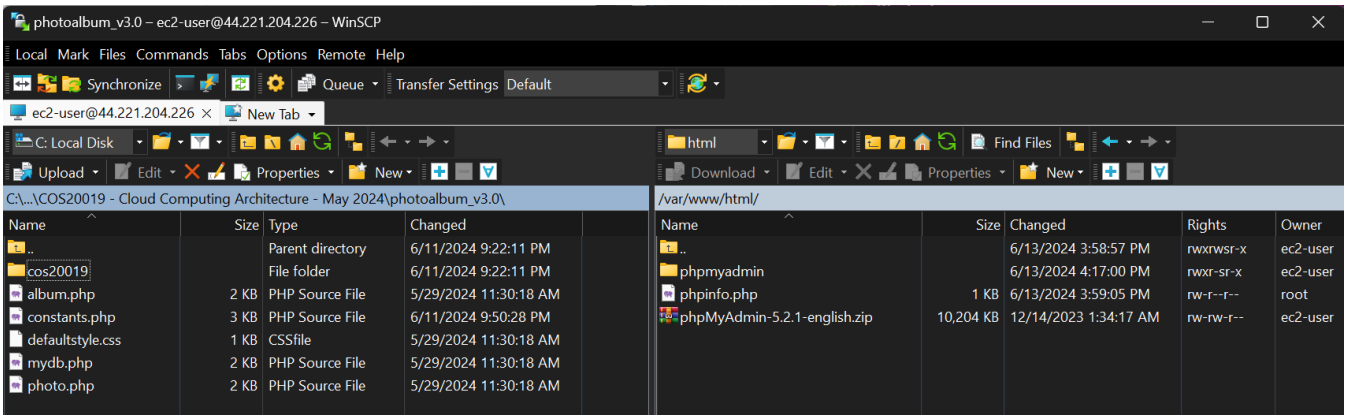

Figure 22: phpMyAdmin is installed in the web server instance

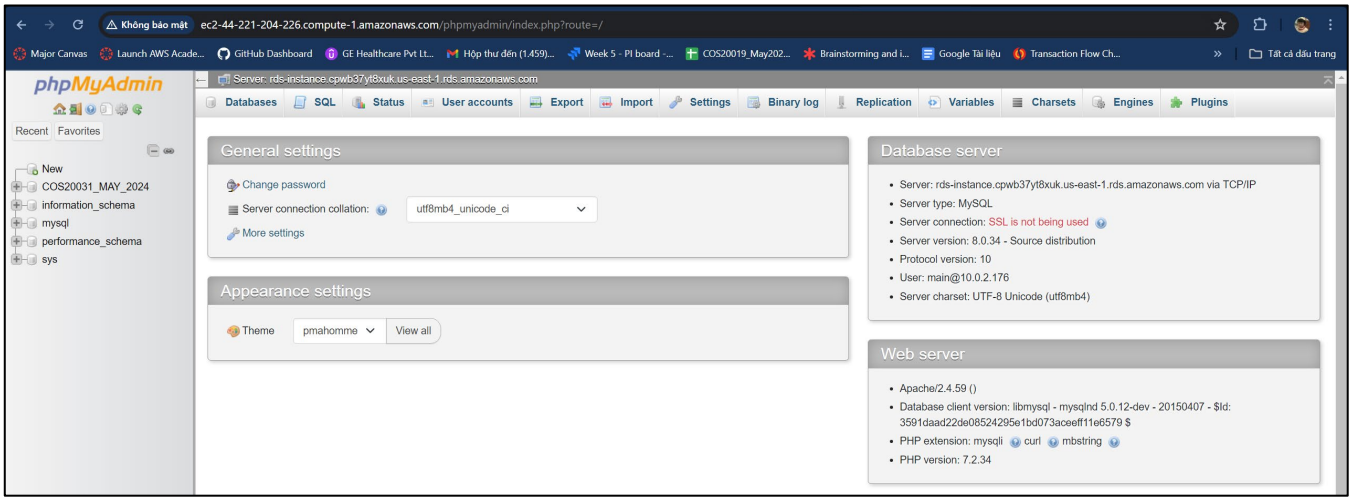

Figure 23: Database is accessible via browser

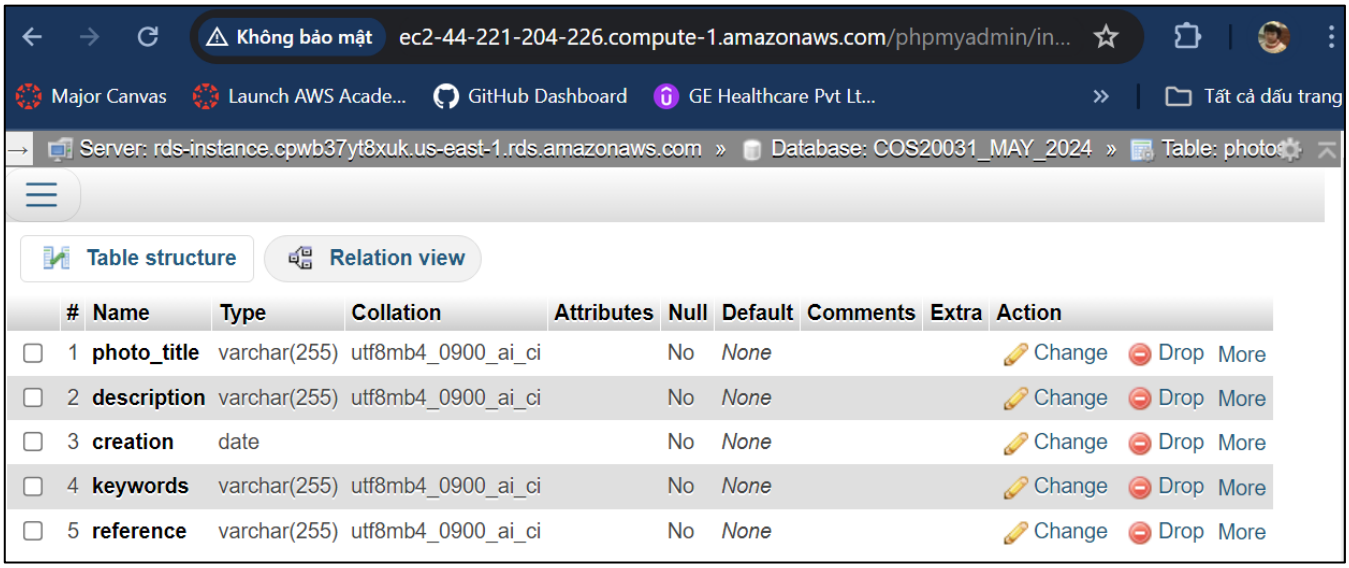

Figure 24: “photos” schema is set up

E. Network ACL

Finally, for subnet level, I add an additional security layer with Network Access Control List. This layer will control traffic into Public Subnet 2, the subnet contains web server instance. It should only allow necessary connection like SSH, HTTP for server access and ICMP for connecting with the test instance

While doing this task, I encounter a problem: Even though I have allowed inbound rules for SSH, HTTP, the web server is blocked, and I can’t access it (previously I can, but the problem occur when I attach the NACL to the subnet). The reason is unlike security group, NACL is stateless, which means a successful traffic needs to match the rule in both inbound and outbound rules [2].

Let's clarify this problem with an example. When a client establish a connection, for example request for a webpage via HTTP (port 80), the destination port will be 80, but the source port is unknown. In fact, it is randomly chosen between 1024 and 63535, so there is no fixed value in every situation. Let's take the source port as 10000 to demonstrate. In this case, when the web server reply, the source port in the reply traffic is 80, but the destination port will be 10000, not 80 because 10000 is the source of the sender. Therefore, only allowing outbound rules for 80 will not enough, we need to make sure outbound rules can cover return traffic from ephemeral ports also.

So, in the context of this assignment, below is my configuration for PublicSubnet2NACL:

For web application traffic, allow HTTP (port 80) for web server request and custom TCP with ephemeral ports (1024-63535) from database subnet(10.0.3.0/24) so the instance can get data from database.

For secure management, allow SSH (port 22) from anywhere (this is not best practice, but acceptable in this assignment)

For connection with test instance, allow ICMP and custom TCP with ephemeral ports (1024-63535) from test instance subnet (10.0.4.0/24) to receive ping and reply when remote access the test instance with SSH.

For outbound rules, allow all to make sure all connections are successful.

The settings above will ensure all necessary traffic are handled following the least privilege but still secure important information.

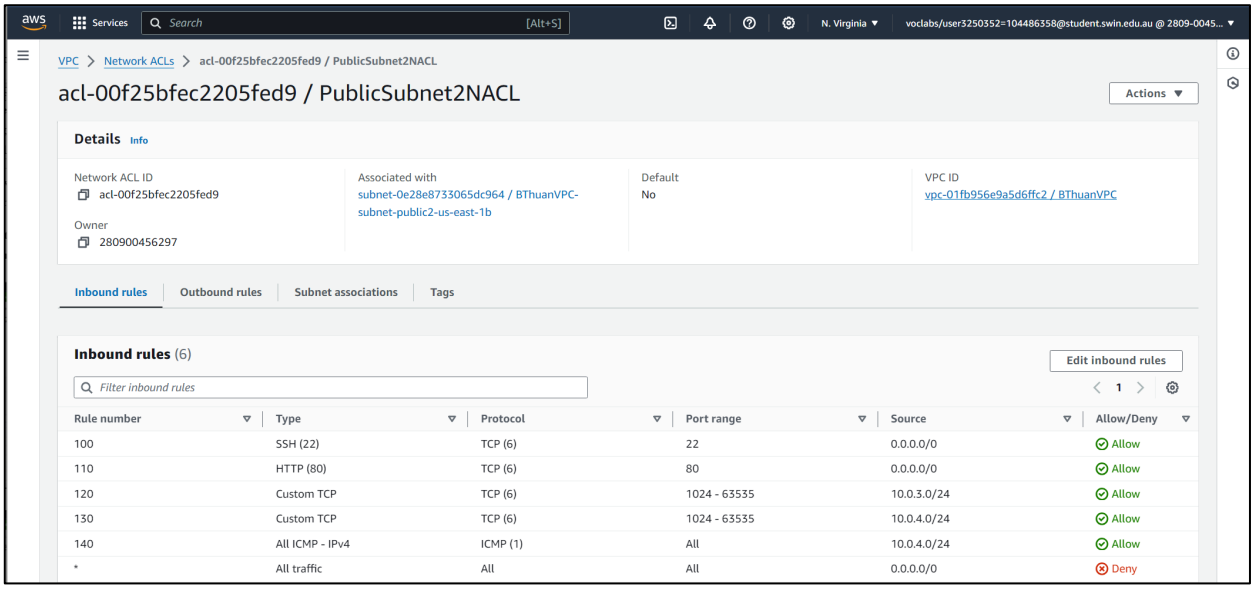

Figure 25: Inbound Rules

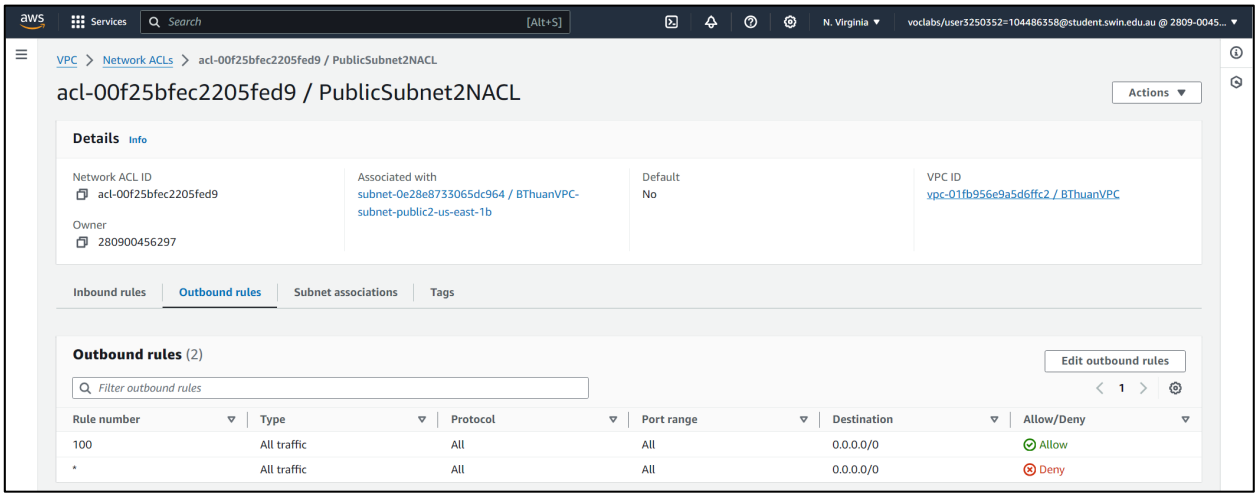

Figure 26: Outbound Rules

III. Functional Requirements

After setting up the AWS architecture, the second part of the assignment focuses on deploying the website in this architecture. In this assignment, the website is a photo album, in which images are stored on AWS, and will be displayed whenever user access the web page. Most of the code for the website is already provided in the instruction folder, so my main tasks will be modifying the code to match the architecture, deploying it to the web server, and testing it functionality.

A. Photo Storage

The website is about uploading and storing images, so the first thing we need is a place to store all uploaded images. In this assignment, Amazon S3 is used, and setting it up is the first task in the second part. I will create an S3 bucket, manually upload some images to test it, and configure the access policy to allow bucket objects are publicly available.

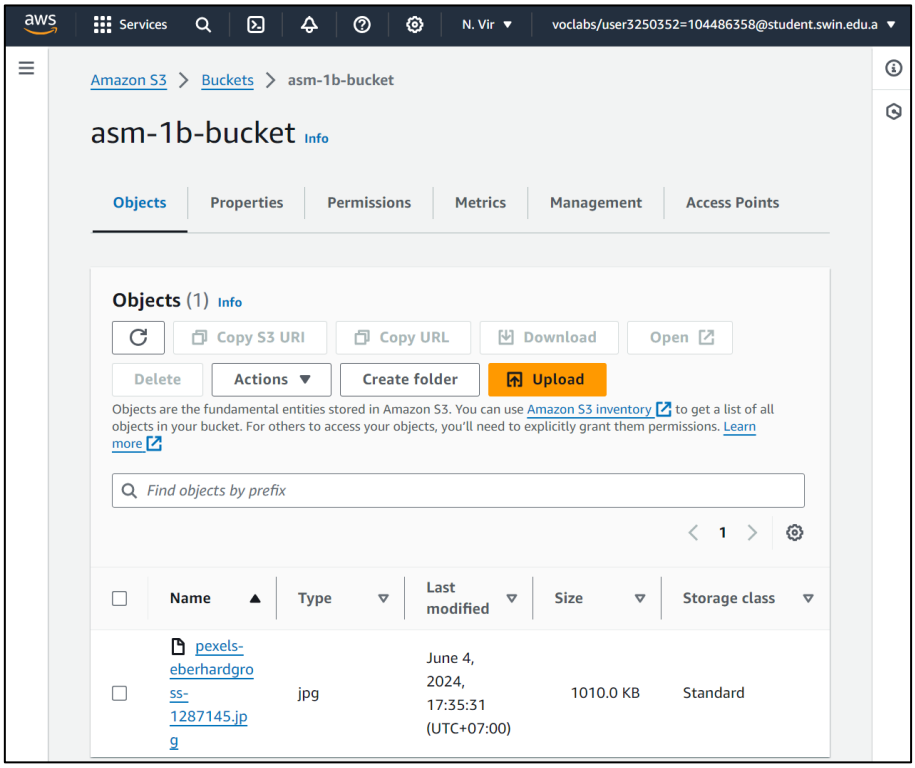

Figure 27: S3 bucket is created with a sample image

Because the web server needs to access to all objects in this bucker to display them, they needs to be publicly available. However, by default, all objects in a S3 bucket is blocked to all connections outside. To solve this problem, I need to do 2 things: disable “block public access” and update new “bucket policy”. Ideally, the bucket should not be blocked, and all every sources can have a read-only access to all objects inside bucket.

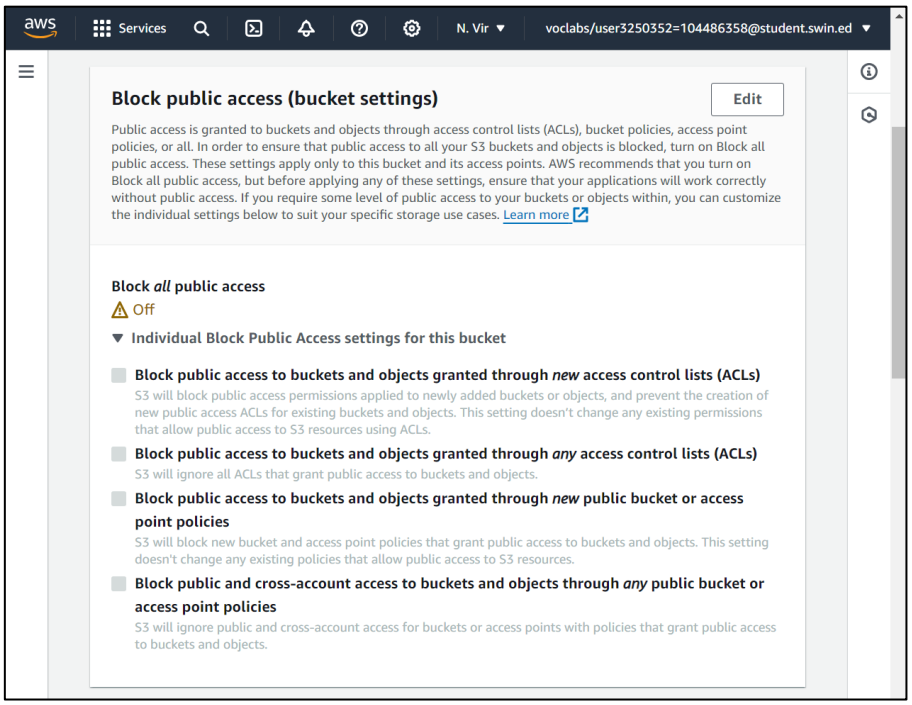

Figure 28: Turn off “Block all public access”

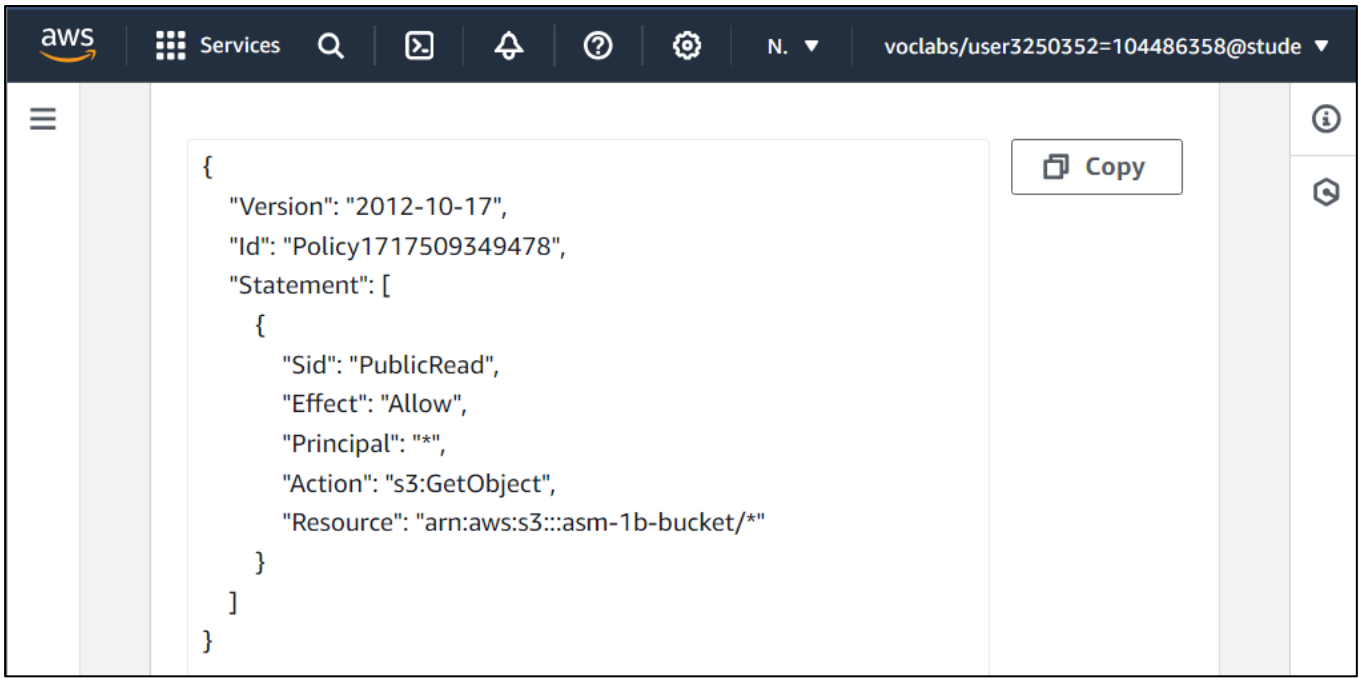

Figure 29: policy allows read-only permission

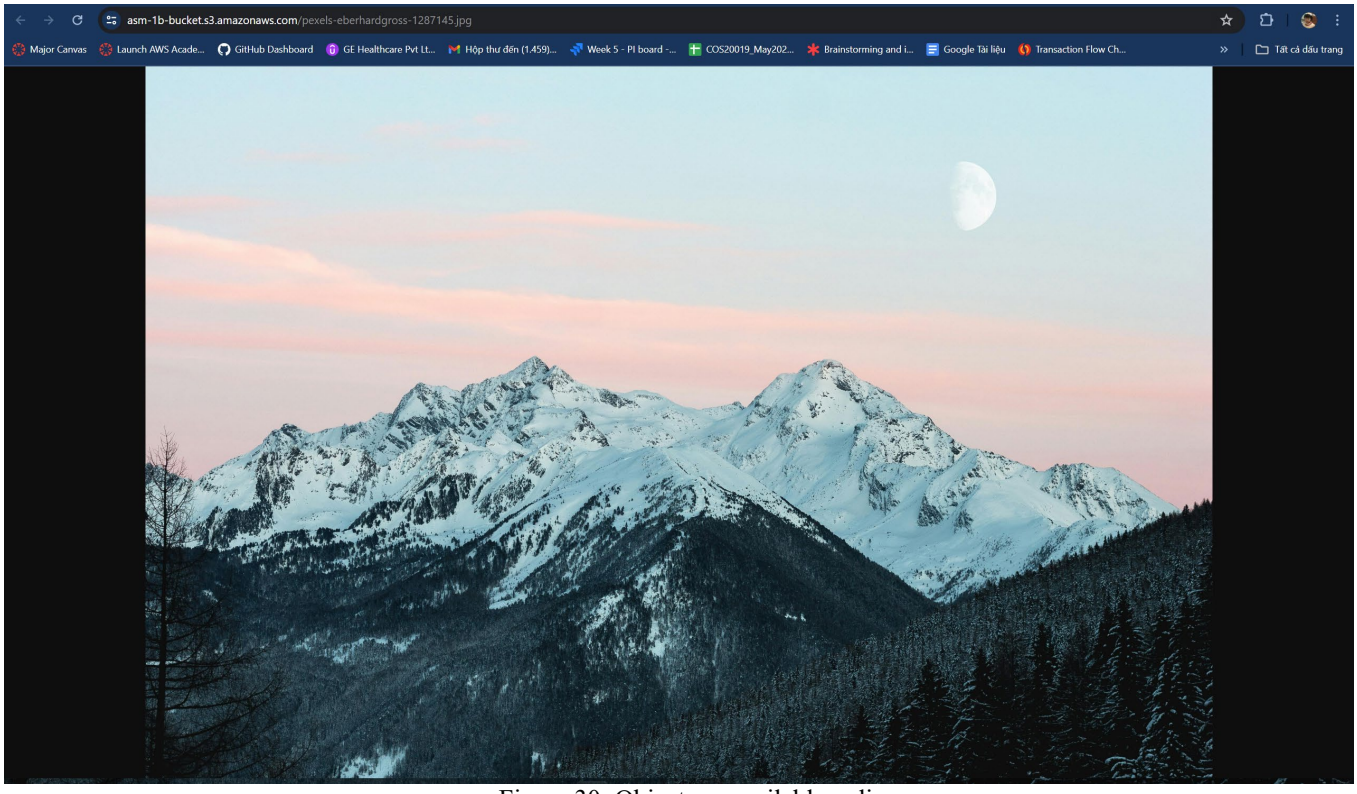

Figure 30: Object are available online

B. Photo meta-data in RDS Database

Now, images have a place to store, web server is ready, the next task is populating some meta-data records to the RDS database, so later the website can use these data to display images and their information. Because in this assignment we don't have an uploading functionality, so I will manually insert some records to the database through the SQL Editor in phpMyAdmin. Each record will store a link to an object in S3 bucket, along with some information like title, description, keywords,…

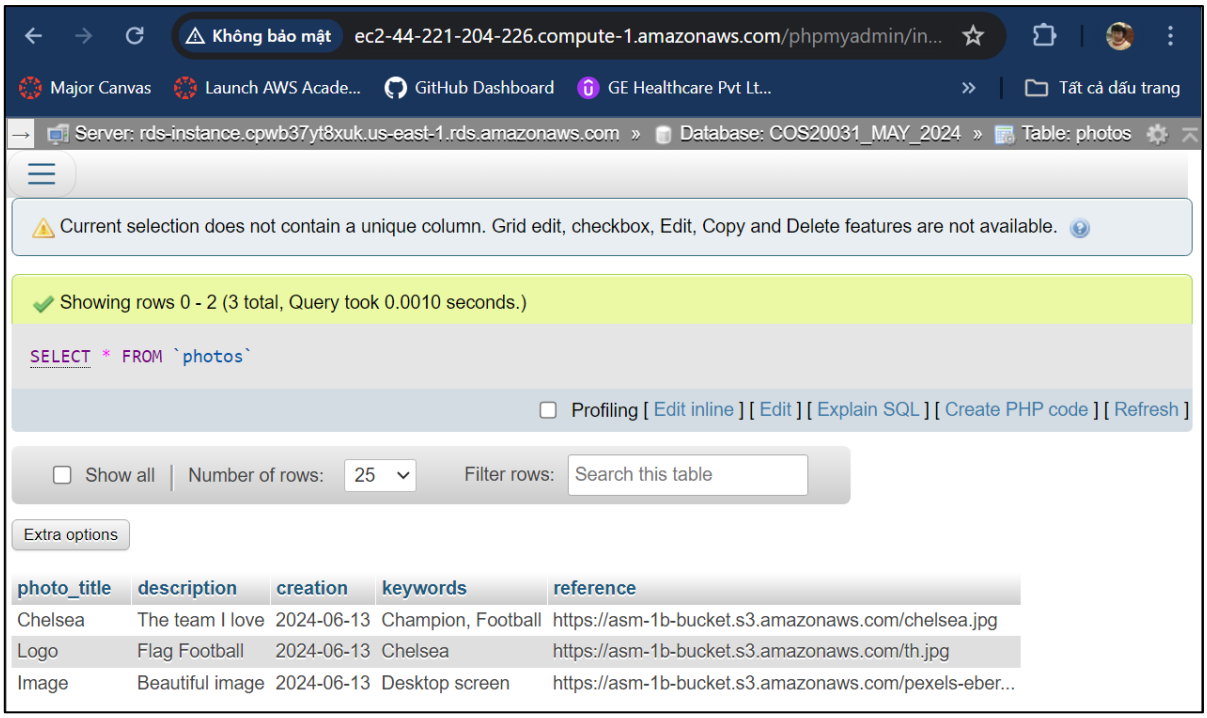

Figure 31: 2 records are added to the database

C. Photo Album website functionality

The final step involves writing the code to display images and their meta-data when users open the website. Because the code has been provided in the instruction folder, I don't need to write everything from scratch. What I need to do is adjust some constants, which includes student's information (in this case is me, used to identify assignment) and database constants (used to connect to the database for this assignment).

```php
// [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', 'Sunday 09:15AM');

// [ACTION REQUIRED] name of the S3 bucket that stores images
define('BUCKET_NAME', 'asm-1b-bucket');
// [ACTION REQUIRED] region of the above bucket
define('REGION', 'us-east-1');
// no need to update this const
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', 'COS20031_MAY_2024');
// [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');
```

Figure 32: Updated constant variables

After finish the code, I will upload it the web server, using WinSCP. In terms of folder, it is also provided in the instruction file. The website now can be access via this link: http://ec2-44-221-204-226.compute-1.amazonaws.com/cos20019/photoalbum/album.php

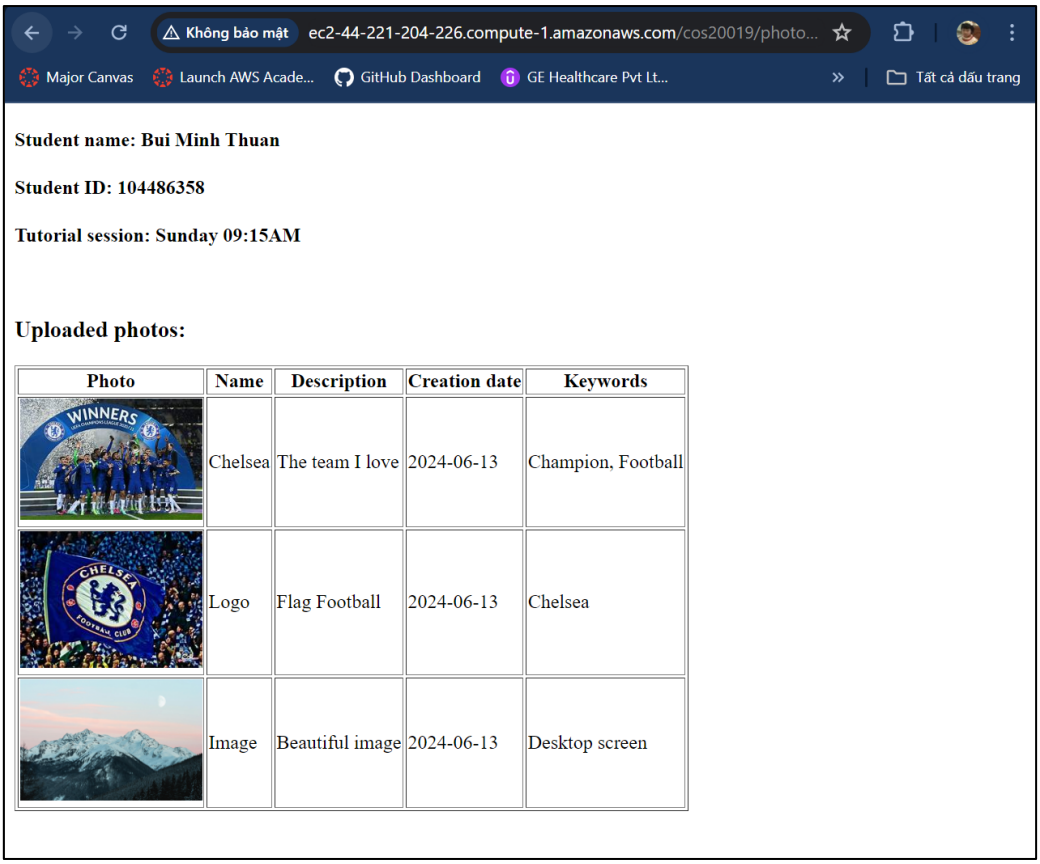

Figure 33: The website works as expected.

You can see the output above: my personal information on top, and a table contains all images in the album. In this table, the meta-data is fetched directly from the RDS database, while the image is taken from the link stored in it.

IV. Conclusion

Throughout this report, I summarized my work for all tasks in Assignment 1B – Unit COS20019 Cloud Computing Architecture. This report covered specifications details (VPC, EC2,.. settings), testing actions (web server access, ICMP, SSH remote connection,..) as well as problems I encounter during doing the assignment.

References

[1] Pope, M. (2014, May 21). *Securely Connect to Linux Instances Running in a Private Amazon VPC | Amazon Web Services*. Amazon AWS. Retrieved June 16, 2024, from https://aws.amazon.com/vi/blogs/security/securely-connect-to-linux-instances-running-in-a-private-amazon-vpc/

[2] Amazon Web Services. (2021). Network ACLs. In Amazon Virtual Private Cloud User Guide. Retrieved June 16, 2024, from https://docs.aws.amazon.com/vpc/latest/userguide/vpc-network-acls.html#nacl-ephemeral-ports.