COS20019 - Cloud Computing Architecture

Assignment 2

******Developing a highly Available Photo Album Website******

Student Name: Lau NgocQuyen
Student Identifiaction: 104198996
Grade Weighting: 15%

Dealine: 9 AM (AEST), Monday of Week 10
Late submission penalty: 10% of total available marks per day.

1. Introduction

- To create a reliable picture album website, using Amazon Web Services (AWS) services is my go-to choices. AWS provides a various services comes with superious functions, including storage, hosting, databases, caching, load balancing, monitoring, and so on . Using these services, we can build a strong and high durability picture album website with innovative features that outperform past jobs. AWS services provide the infrastructure and tools needed to assure high availability and optimal performance while storing, retrieving, and managing picture albums.

2. Foundation and Infrastructure

- To prepare for the deployment phase, we must first establish a solid foundation for our website. Configure VPC and subnets flawlessly based on the below Architecture structure. The VPC will provide the necessary networking infrastructure, ensuring isolation, security, and control over our resources. Properly configuring the subnets within the VPC ensures secure communication between website components. This step is crucial for building a reliable photo album website.

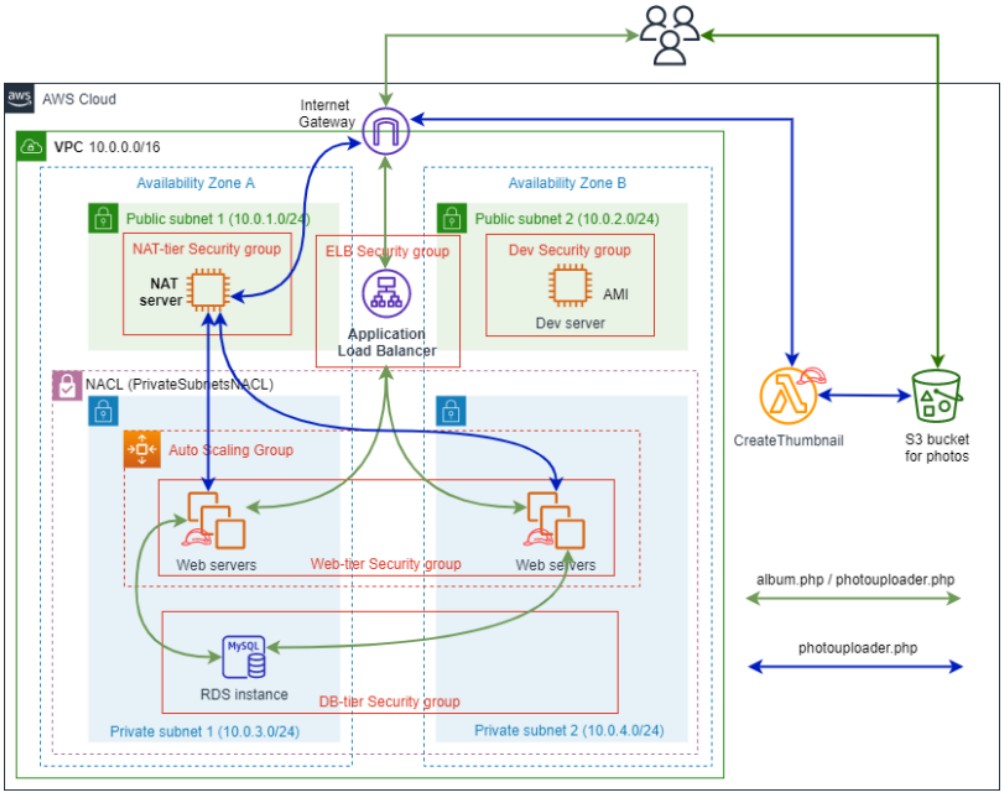

Image 1 – Architecture diagram

- Since the goal has previously been introduced in Assignments 1A and 1B, this work is not too tough. Following configuration, the following VPC road map for our VPC and the corresponding subnets may exist:

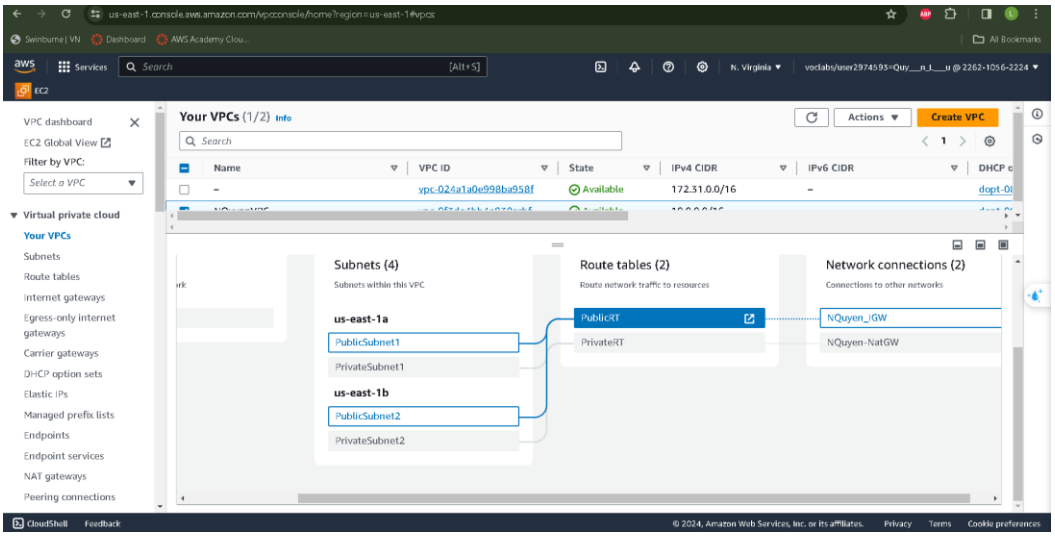

Image 2 – VPC resource linked road map

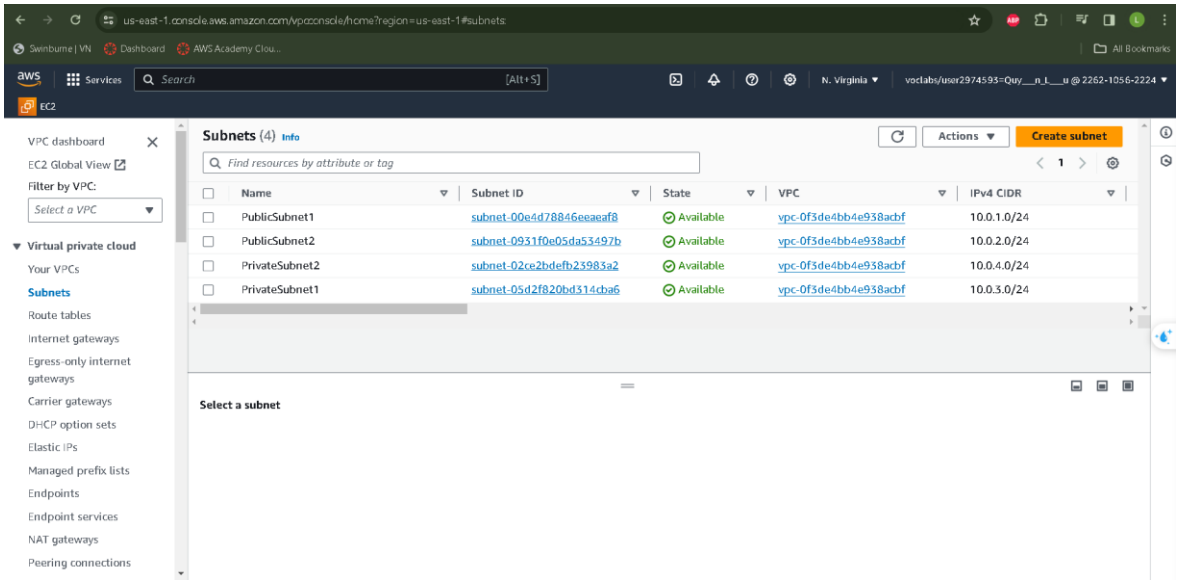

Image 3 – IPv4 CIDR for every subnet

- During the development phase, the PublicSubnet2 Subnet will be sent to the IGW to test website operation. If only our website is completely operational, an Amazon Machine Image (AMI) will be produced **for PublicSubnet2**, allowing for deployment in multiple locations as needed. **Private Subnet1** and **PrivateSubnet2** will be used to deploy an autoscaling group, which will be routed to the NAT Gateway. It's vital to note that this assignment will use a NAT Gateway rather than a NAT instance. The **PublicSubnet1** subnet will operate the NAT Gateway, which will be routed to the Internet Gateway. The graphic below demonstrates the configuartion of NAT gateway

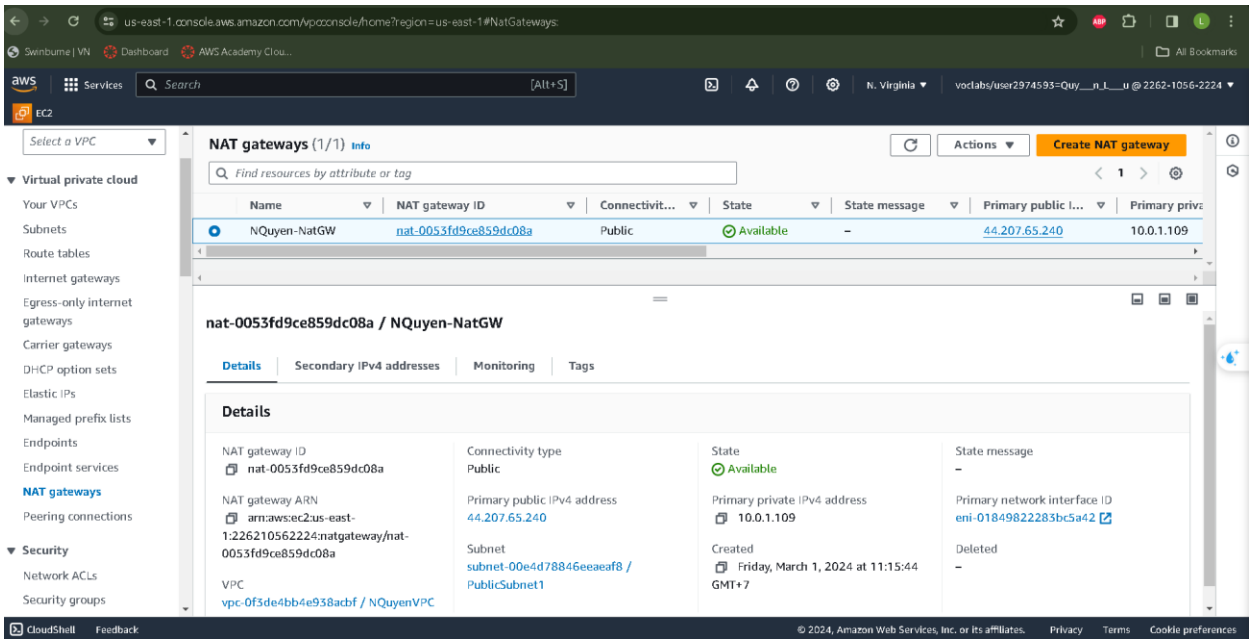

Image 4: NatGW in Nat_Server_Public1

- NAT will also bring this assignment's first phase to an end. After the VPC is built, we will establish Organizations that exist on these subnets.

3. Functionalities of the Website

- To make sure that our websites developed in subnet 2, S3 would be mandotaryly operated .

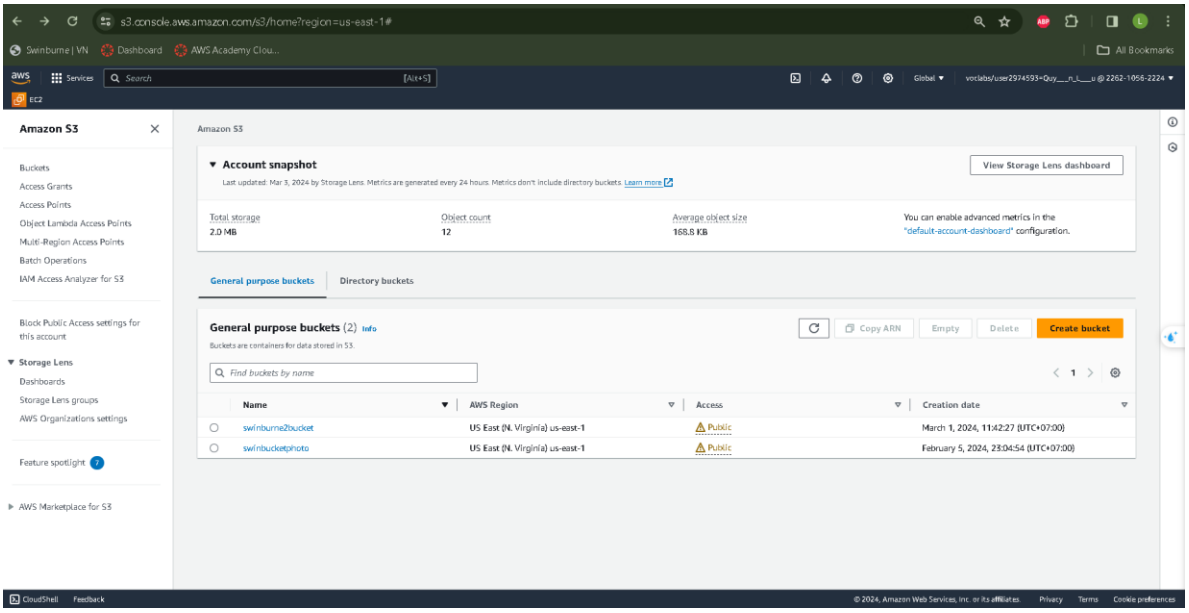

Image 5: Assignment 2's Amazon S3 bucket

- Based on the architecture structure, our Amazon S3 bucket will only allow entry from and by the **Load Balancer .** However, configure restriction permission for Amazon S3 bucket policy is required.

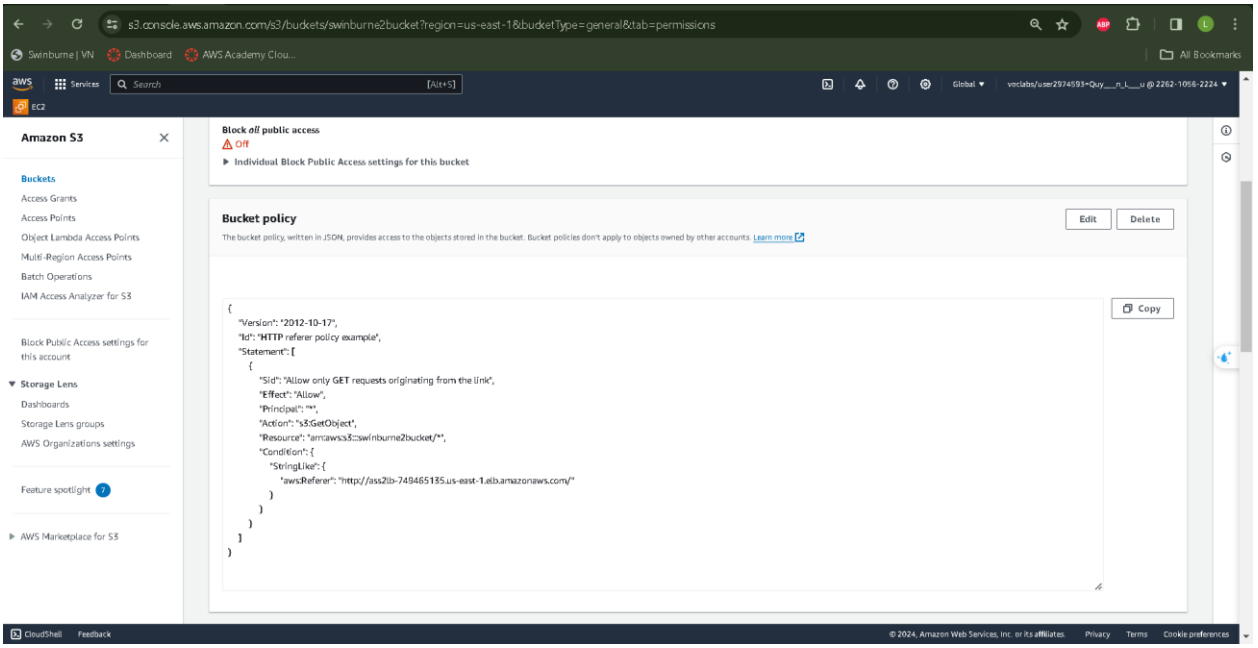

Image 6: Amazon Bucket's Policy

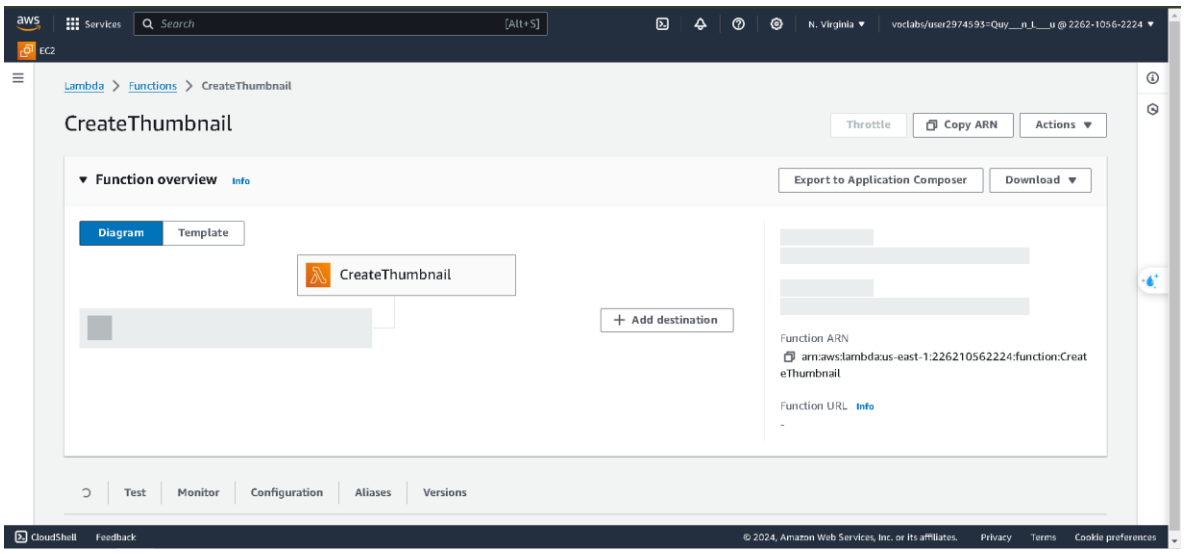

Image 7: Creating Lambda function by CreateThumbnail

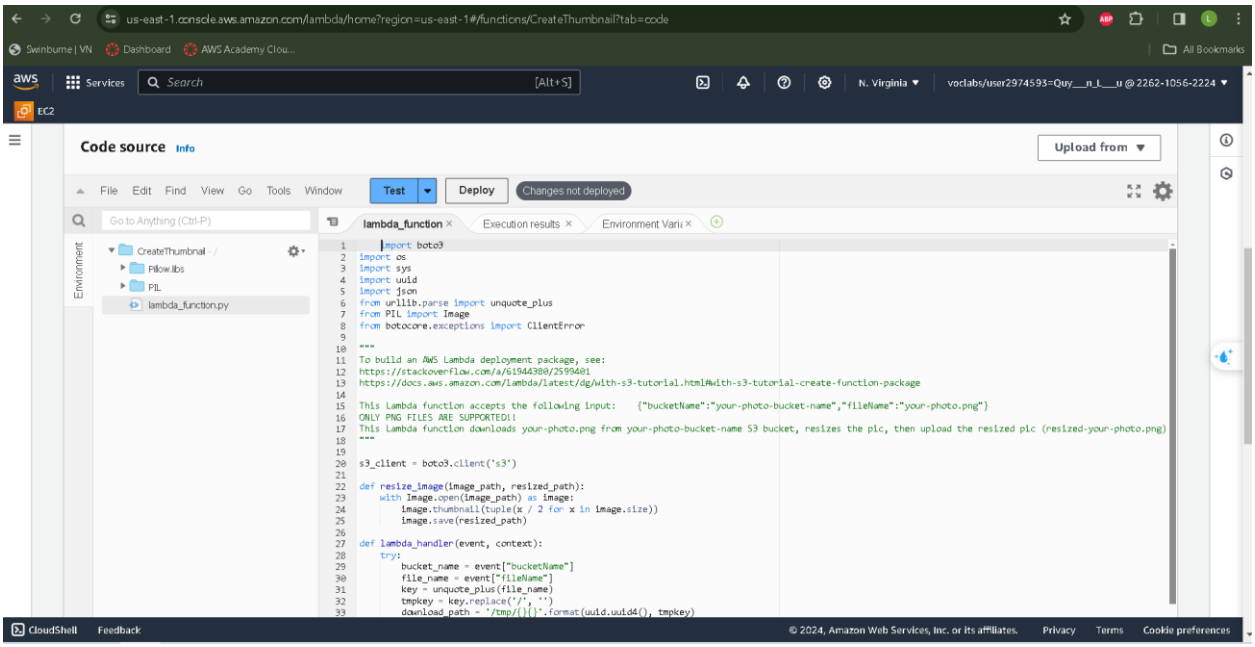

Image 8: Upload the provided lambda file to code source

- The function package is already given, therefore we simply need to test it to ensure accuracy.

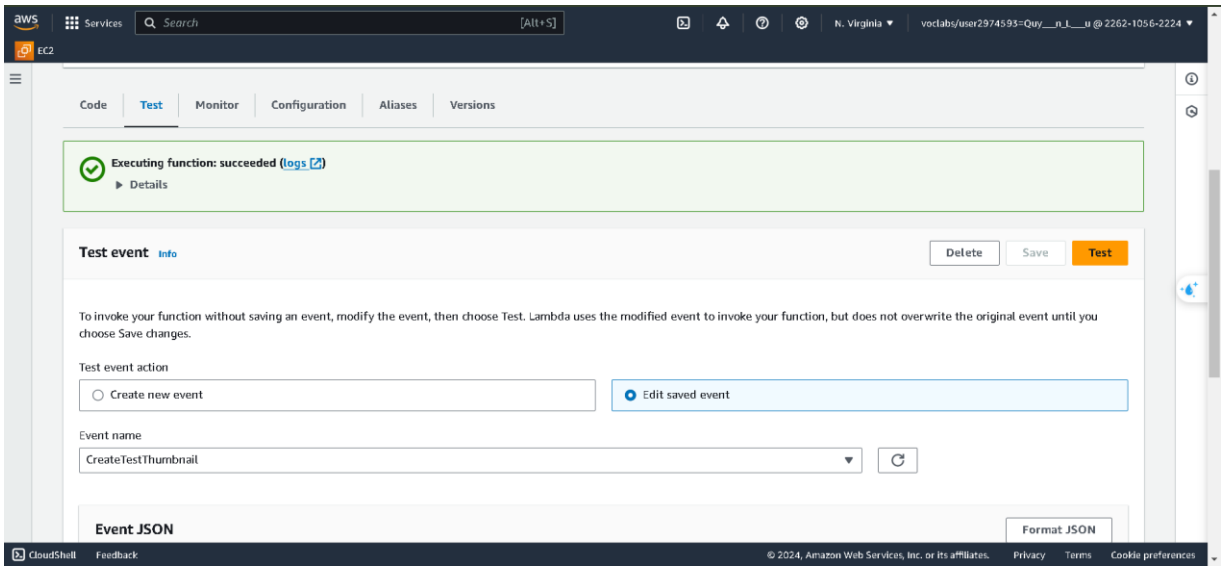

Image 9: Create test case for lamda provie

- After completing the test case, we will set up a Relational Database Service and create a database instance within a second.

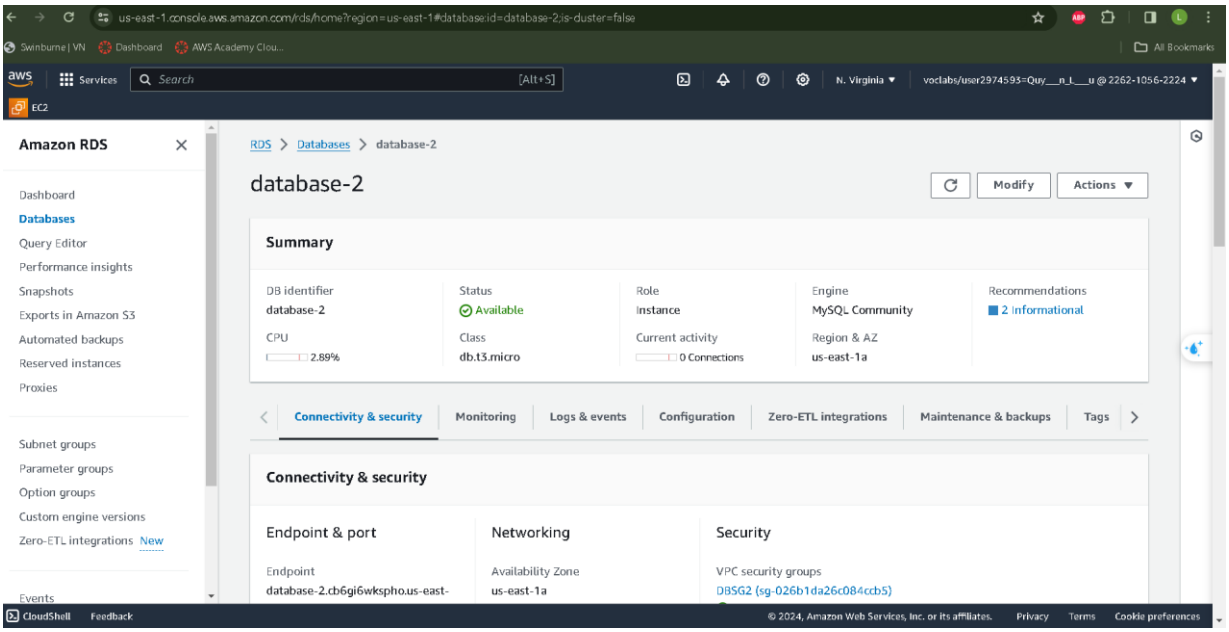

Image 10: database-2 in Amazon RDS for image's database

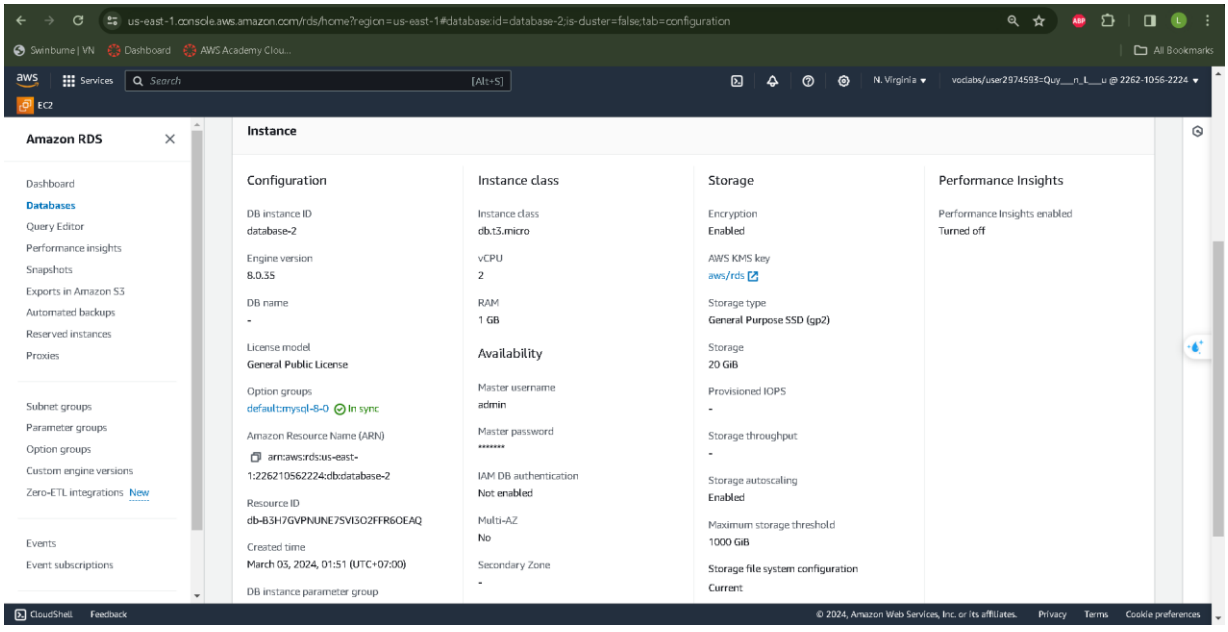

Image 11: RDS Database detailed

- Before uploading our website to the development server, we need to conImage all missing code.We will need to alter the constant.php and Config.inc.php files in the

given ZIP package.

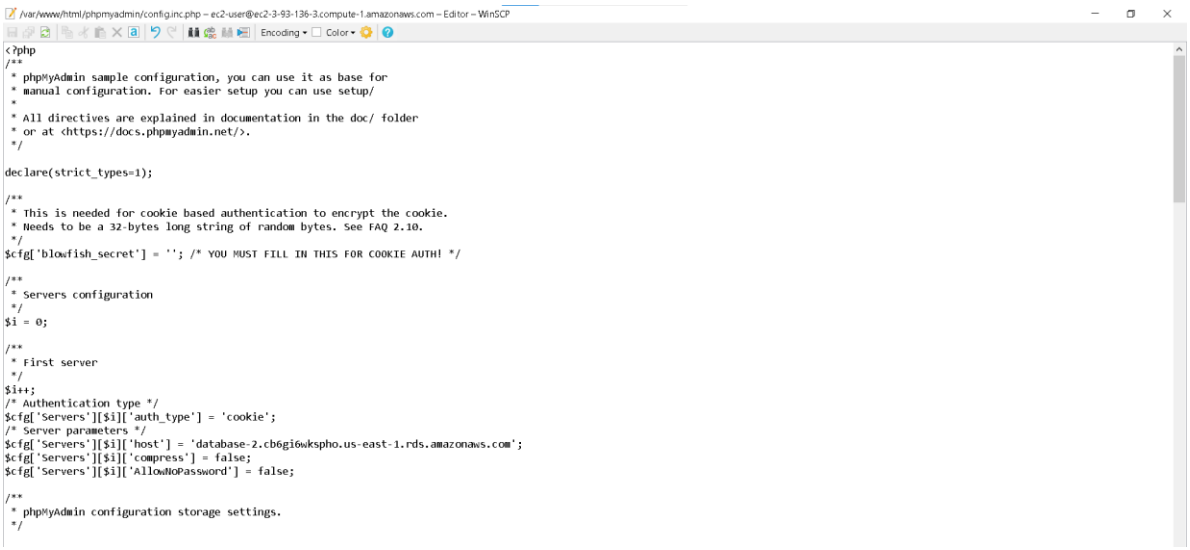

Image 12: RDS end point replacement for server

- Once all components are successfully conImaged, create an EC2 instance on the development server.

4. Built of the Website using EC2 and AMI

- EC2 instance function should be choose correctly in order to hosting the website

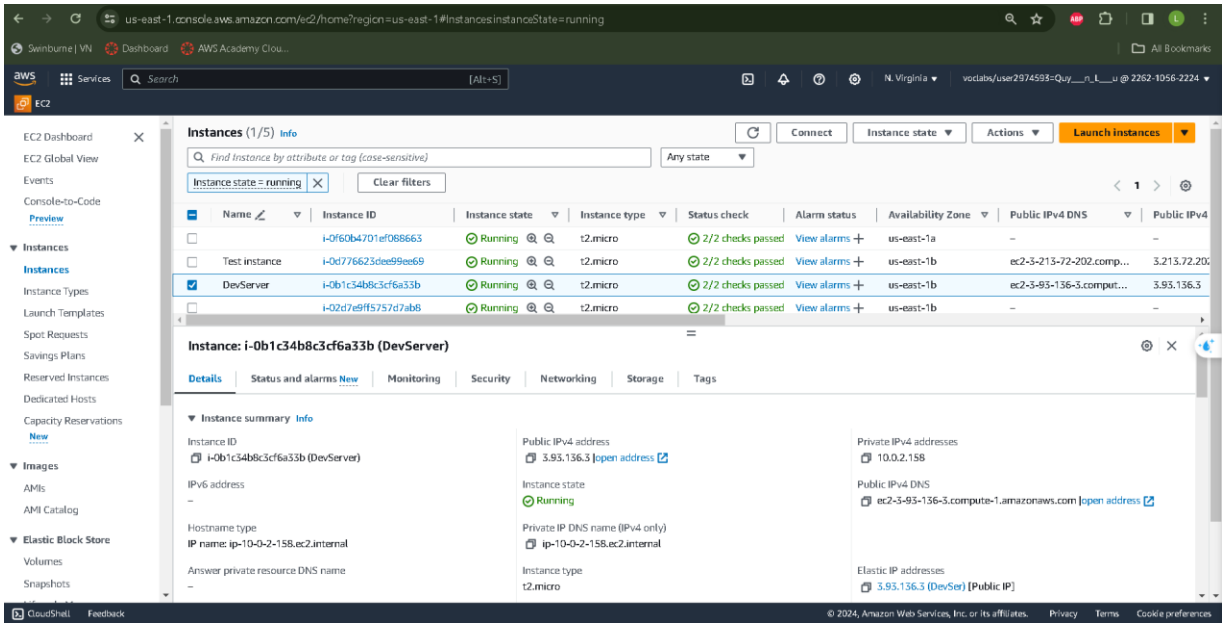

Image 13: DevServer Instacnce

- To use the CreateThumbnail Lambda function to upload items and re-size the image to an S3 bucket, an IAM role is required.

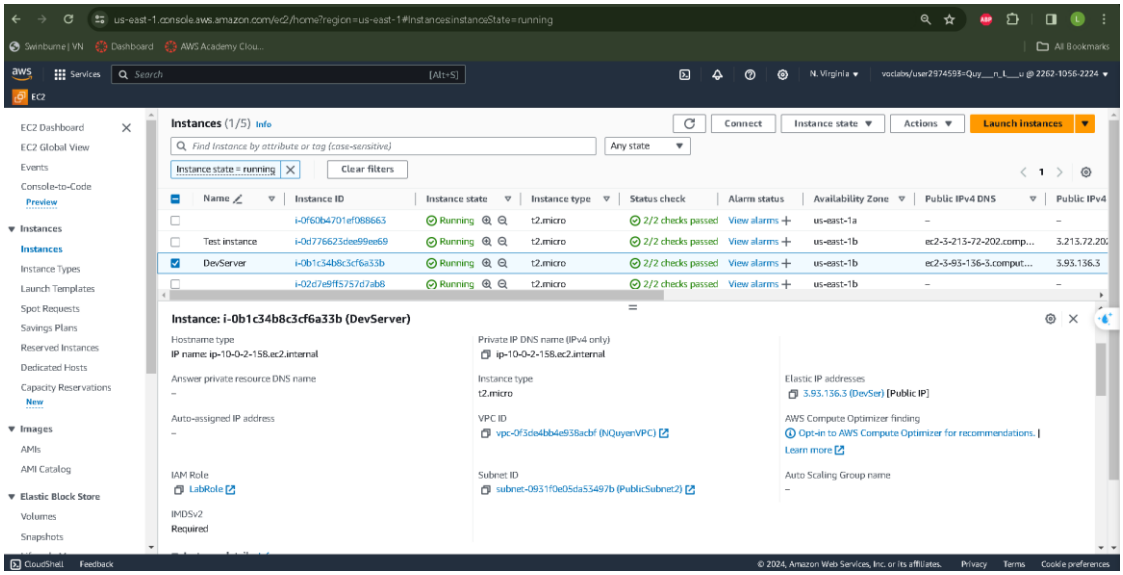

Image 14: Additional settings of WebDevServerInstance

- Verifying if an EC2 instance is accessible, associate Elastic IP address and see the website throughout public IPv4 DNS.

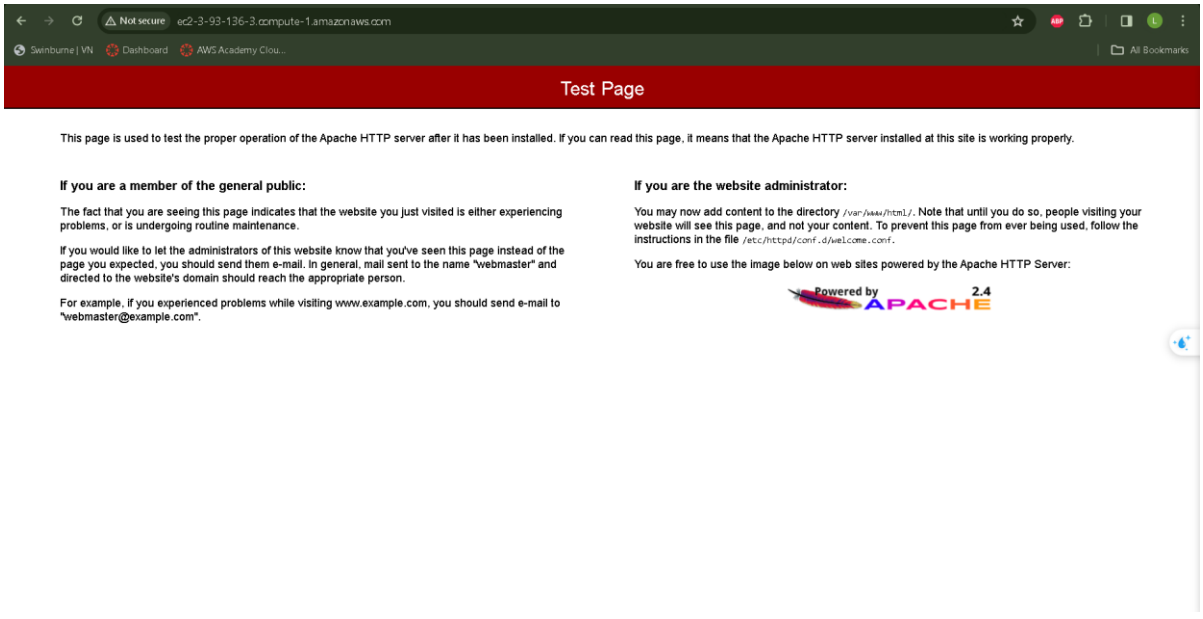

Image 15: Deploying EC2 and Page Testing Flawlessly

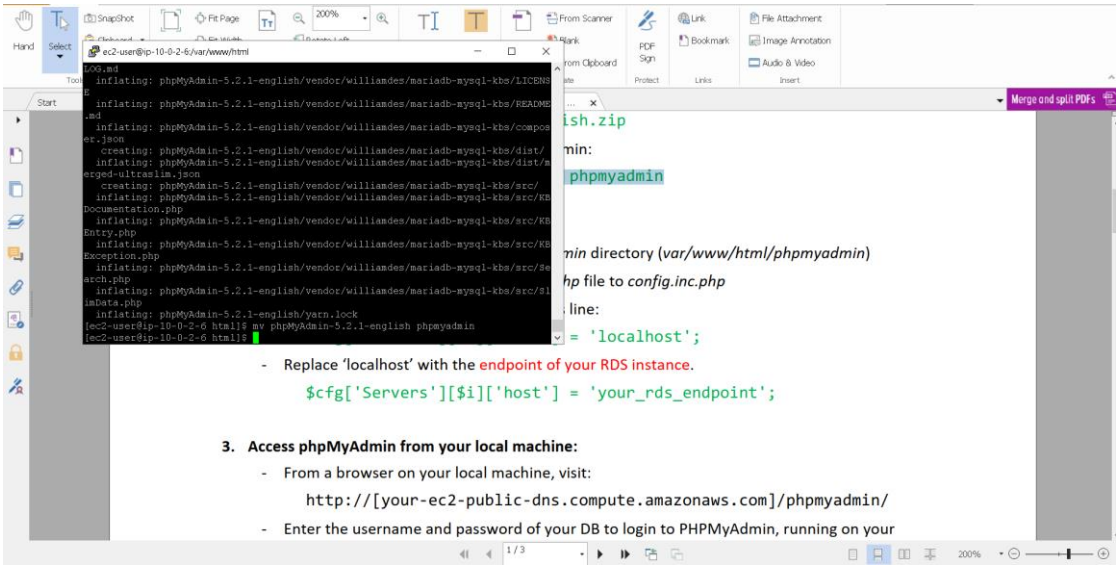

Image 16: WebDevServer SSH terminal using Putty

- Phpmyadmin creates meta-data for our database, monitors its operation. To connect our phpMyAdmin console to the RDS instance generated in stage 3, make a little change to config.inc.php.

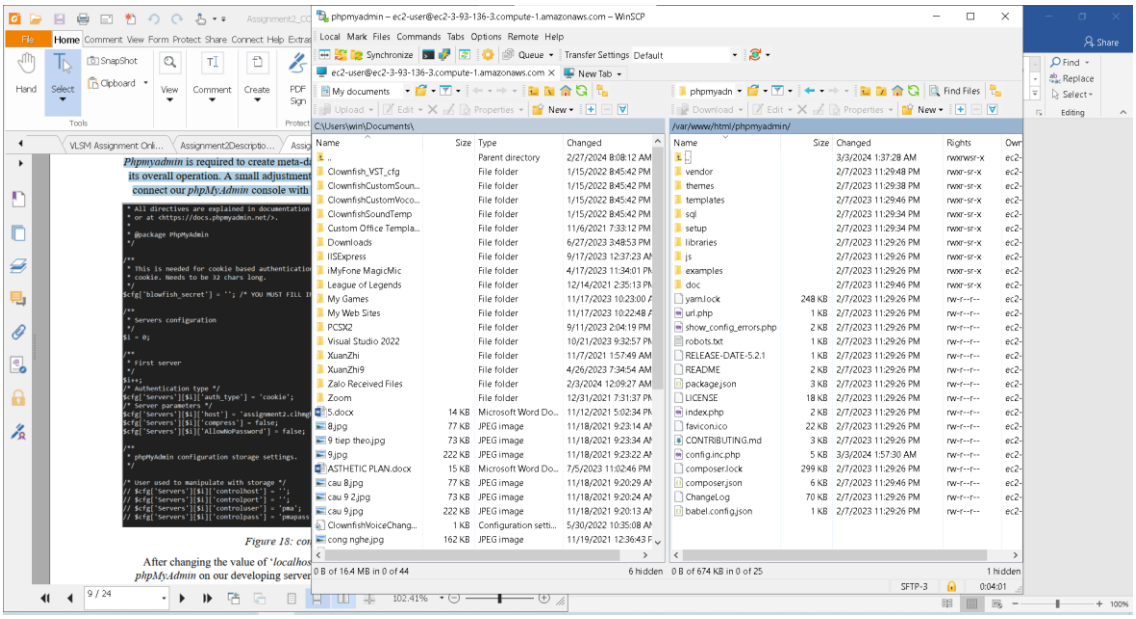

Image 17: Change the config.sample.inc.php into config.inc.php

- Modifying 'localhost' option in config.inc.php allows us to utilize phpMyAdmin on our development server website to generate database metadata.

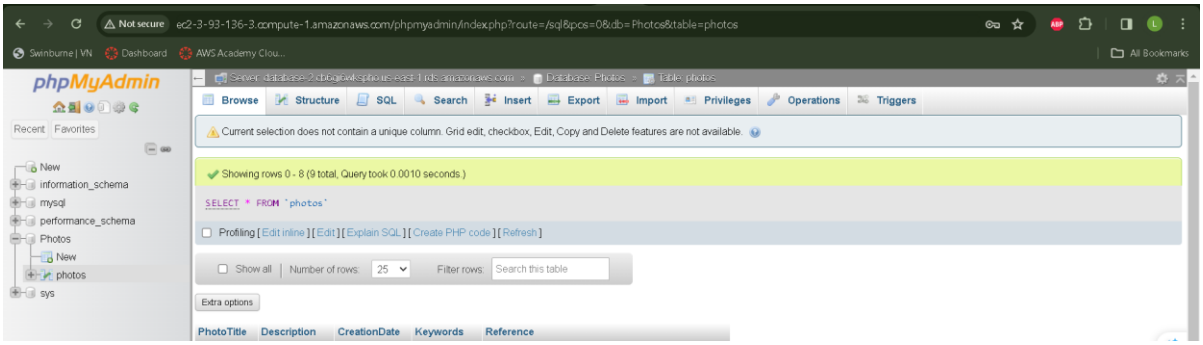

Image 18: Table photos in phpMyAdmin

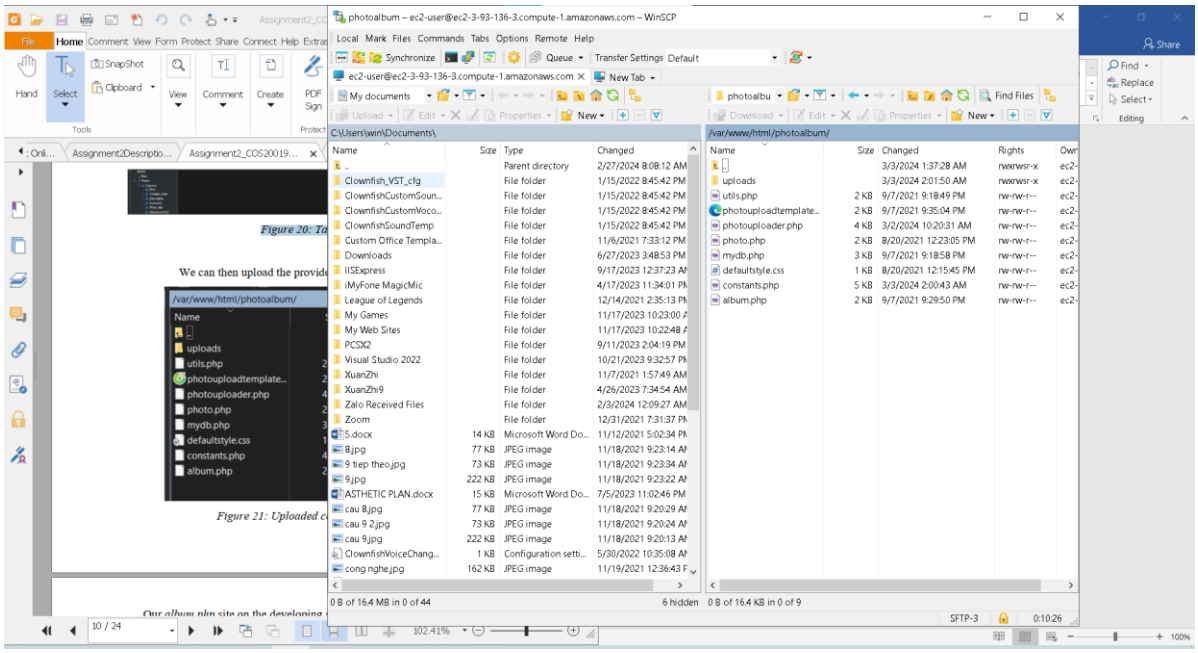

Image 19: Important php file for the website

- The ***album.php*** webpage on the development server will look like this.

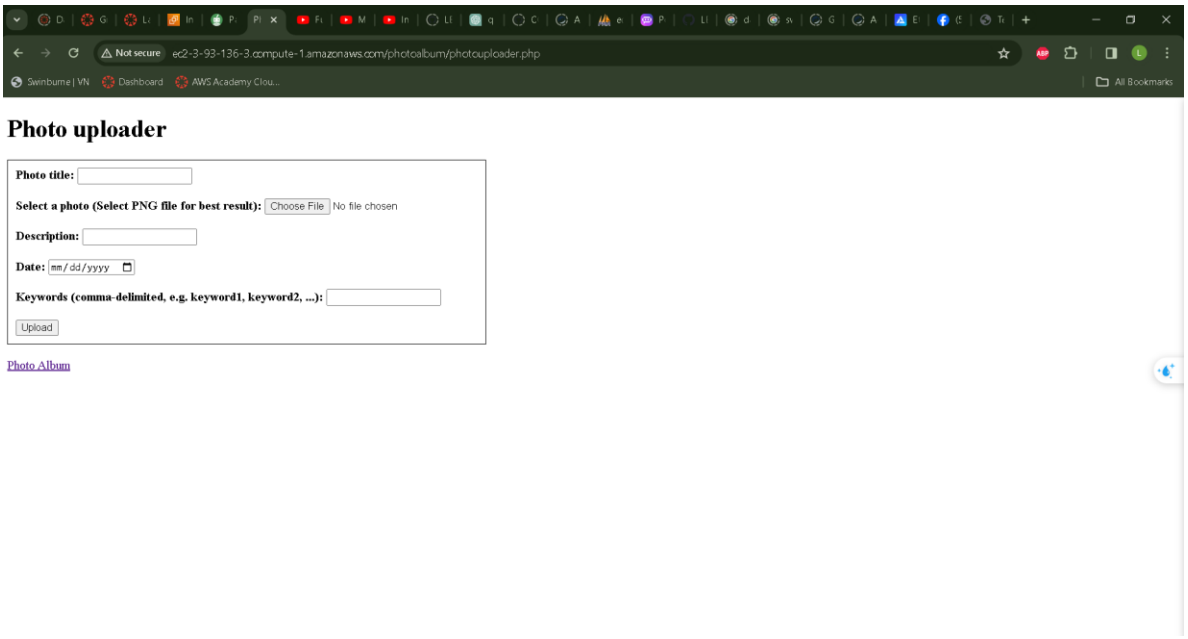

Image 20: photouploader page

- We may upload images to the website for testing.

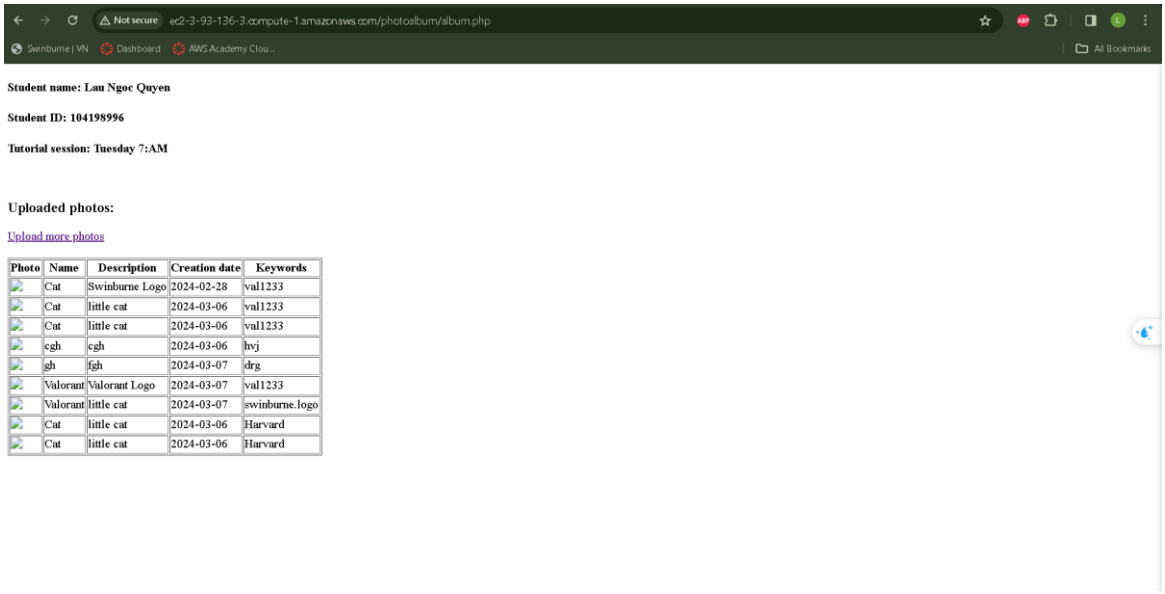

Image 21: photoalbum page

- After uploading, the image may be seen in both phpMyAdmin and S3. phpMyAdmin keeps the metadata, whereas S3 stores the resized picture. To enhance security, we have created a limitation that only displays photos on the album.php page when viewed using our Load Balancer. As a consequence, while accessing the

album.php page from the server, no photographs will be displayed. However, this behavior will change if we deploy our Web server from the Auto Scaling Group.

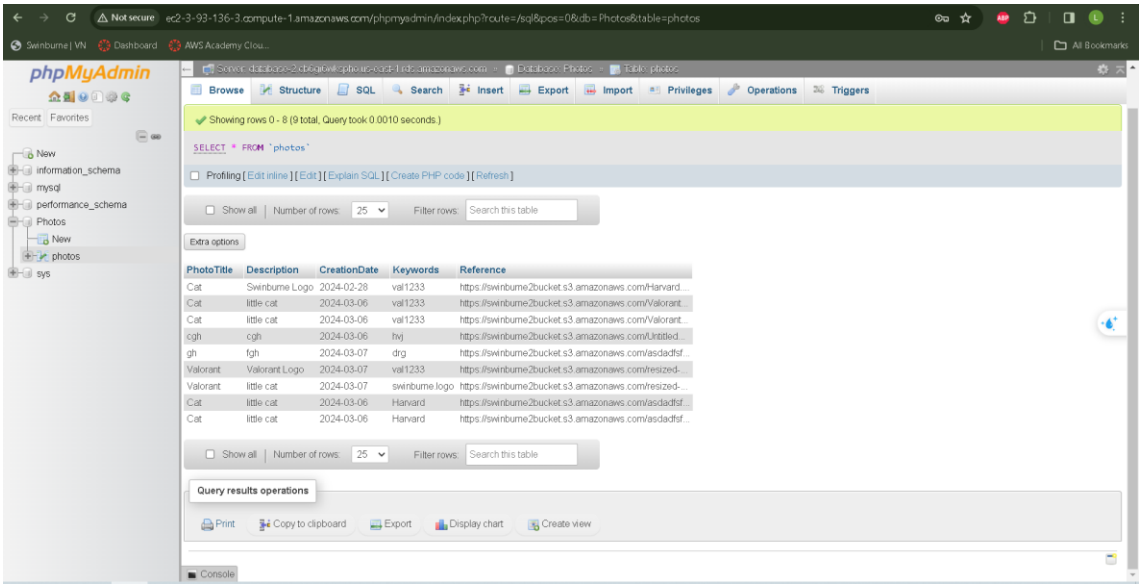

Image 22: Metadata in phpMyAdmin

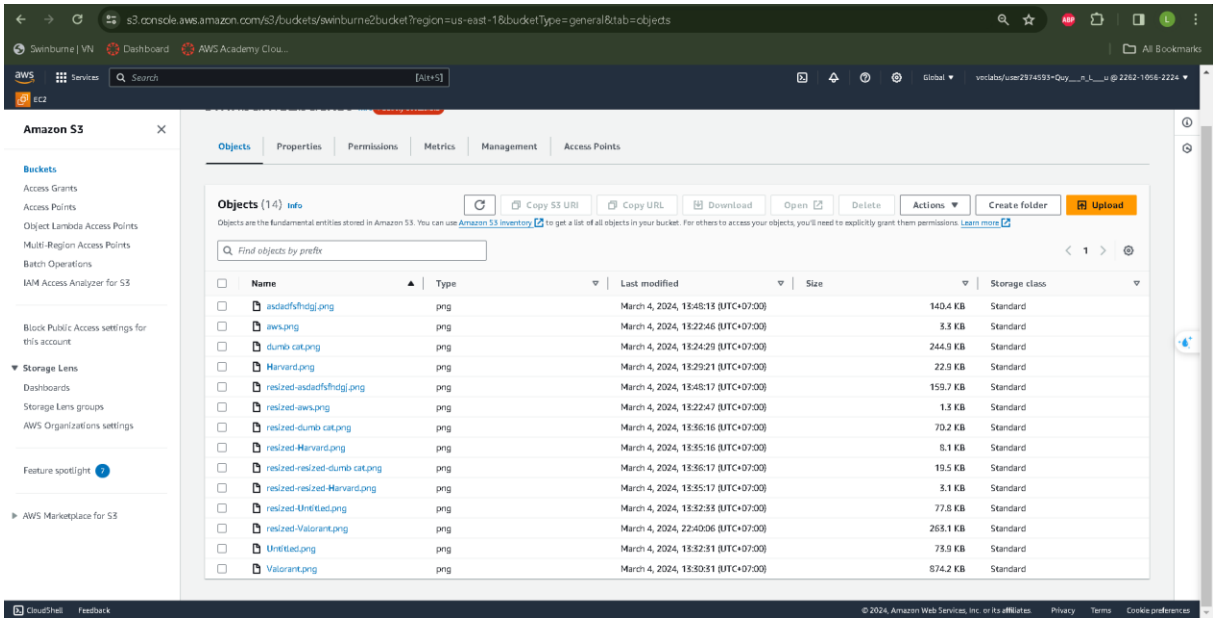

Image 23: All photos and image resizing objects are stored in an S3 bucket.

.

- With our website completely built and operational, the following step is to generate an Amazon Machine Image (AMI) for the WebDevServer. This AMI will provide a snapshot of our server's present state, including installed software, configurations, and

any changes done.

Creating an AMI allows us to capture the entire server setup, making it easier to replicate and deploy in different environments. By deploying the AMI to our Auto Scaling Group, we ensure that new instances launched by the group will have the same configuration as the WebDevServer, ensuring consistency and scalability.

This process involves creating the AMI from the running WebDevServer instance and then updating the Launch Configuration or Launch Template used by the Auto Scaling Group to use the

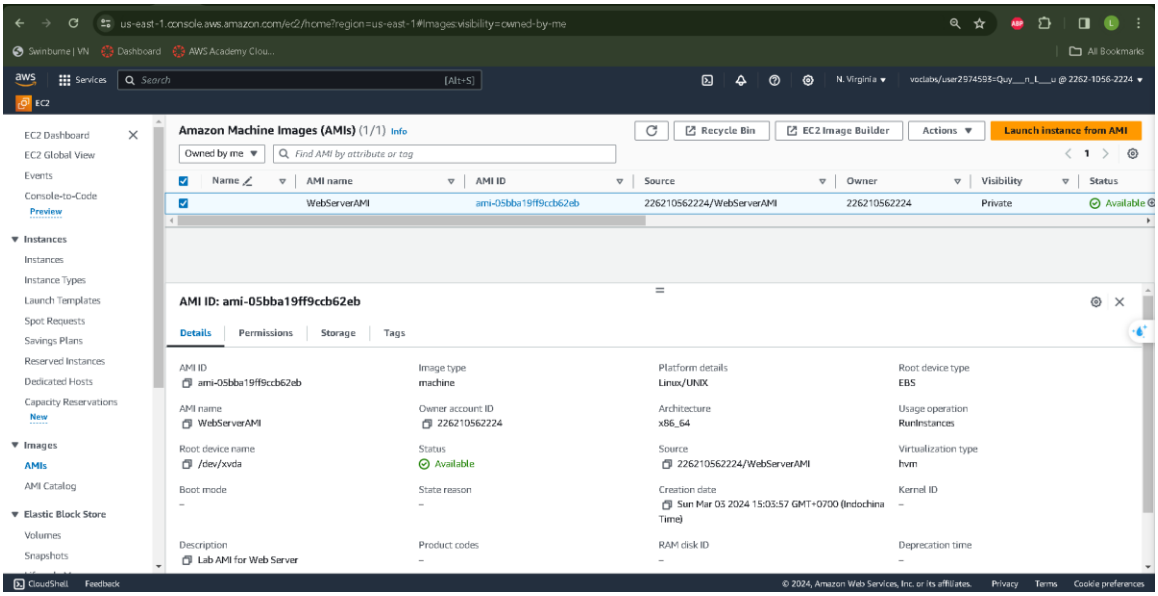

Image 24: Our WebDevServer's AMI.

5. Load Balancer and Auto Scaling Group

- To improve website availability and scalability after development, we may use Elastic Load Balancer (ELB) and Auto Scaling Group (ASG). Setting up an ASG

allows us to automatically grow the number of EC2 instances based on predetermined circumstances. Integrating it with an ELB allows for effective load balancing and enhances fault tolerance by spreading traffic over numerous instances. This combination ensures that our website can withstand variable traffic levels while maintaining excellent availability.

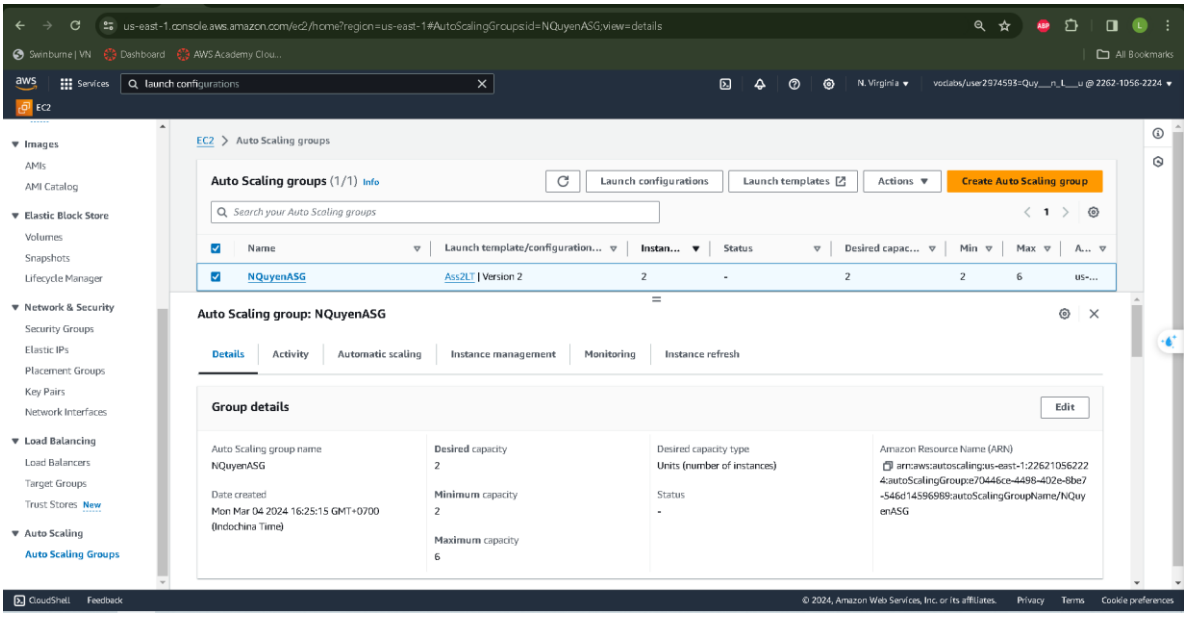

Image 25: Auto Scaling Group's Detail

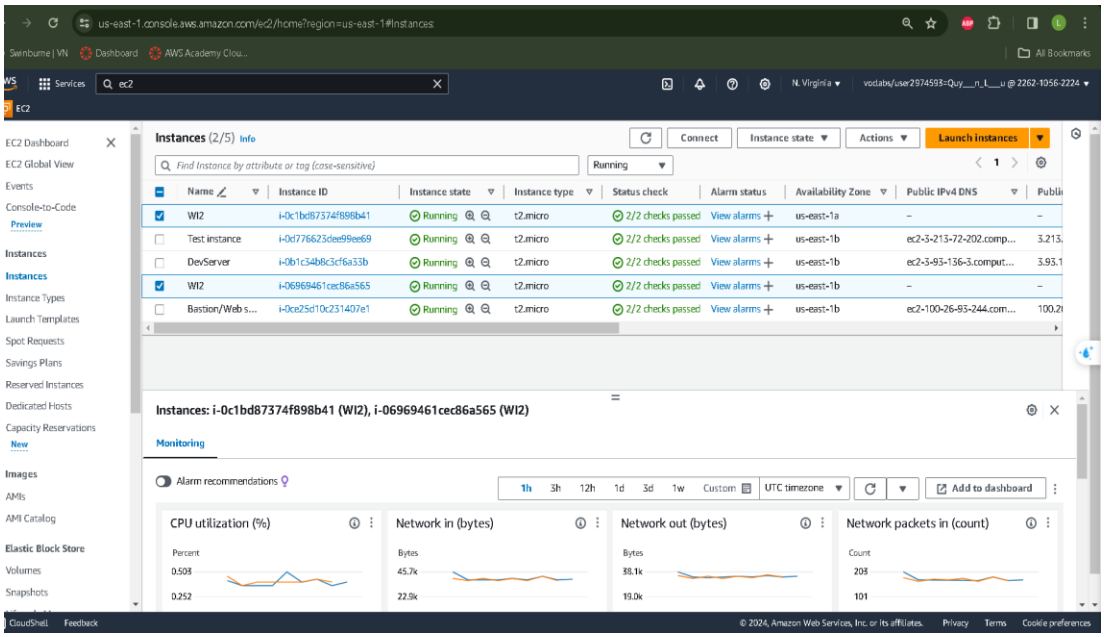

Image 26: 2 WI2 creating from Auto Scaling Group

- Launching the load balancer requires a target group.

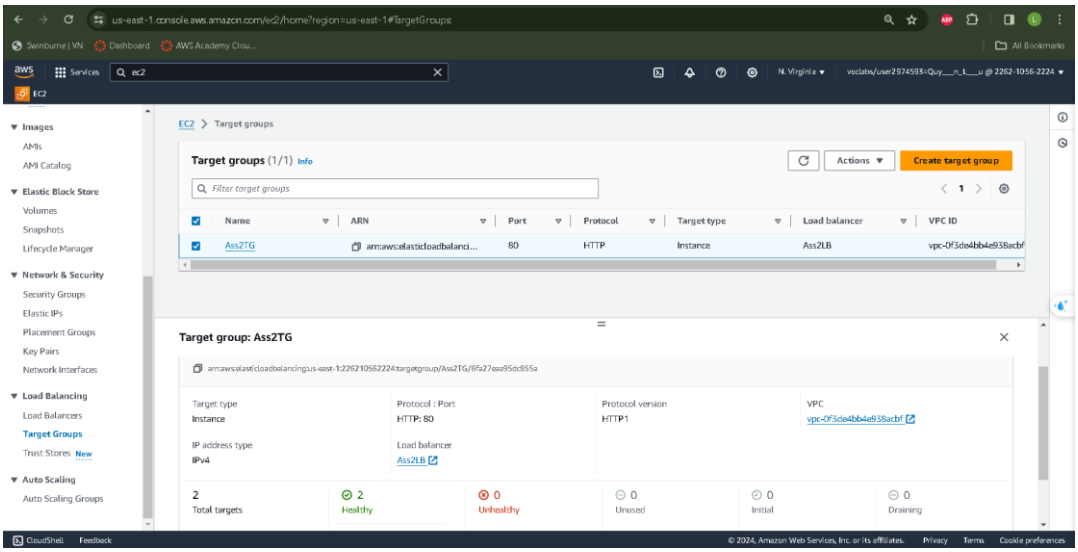

Image 27: Loadbalancers attached by Target group

- Our Load Balancer after attach Target Group will demonstrate like this:

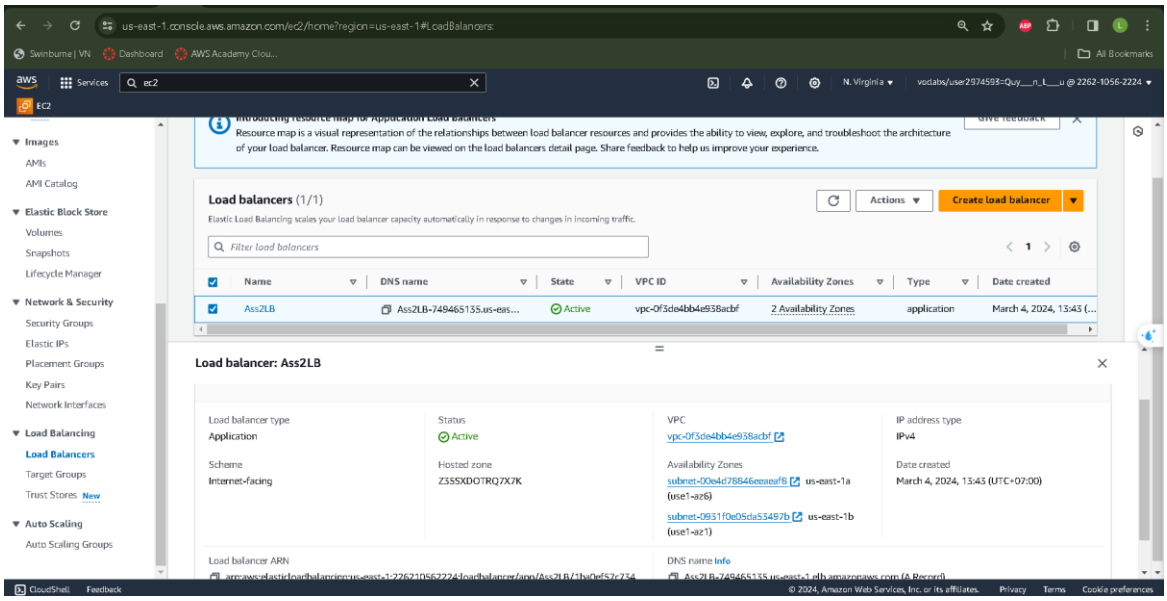

Image 28: Ass2LB is creating

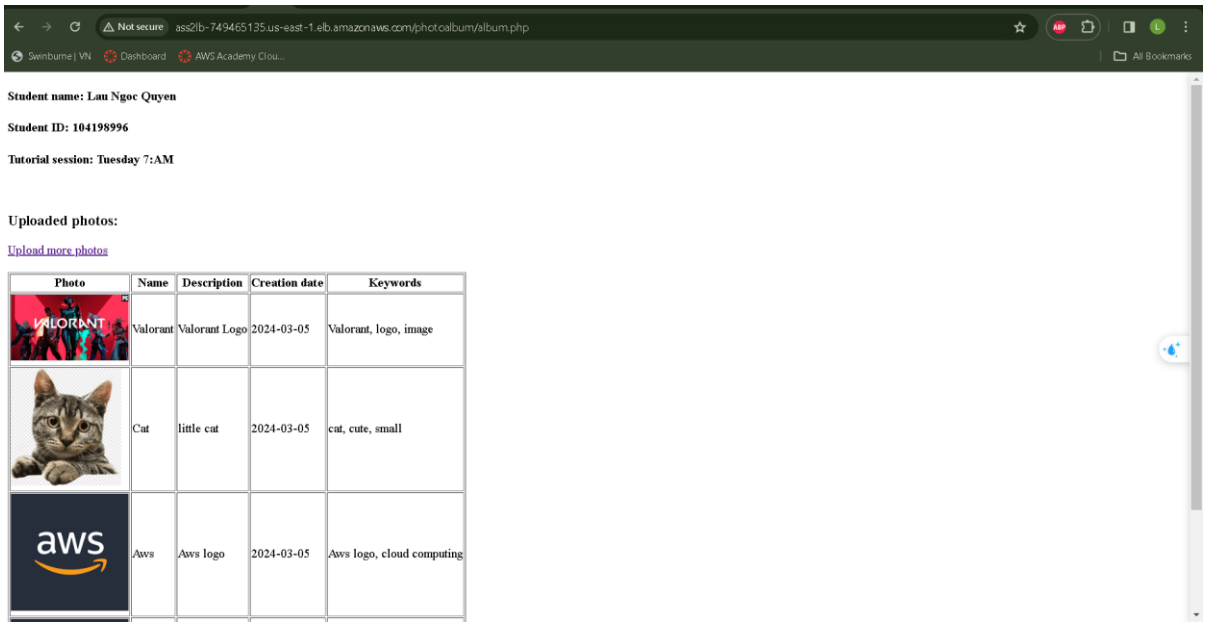

Image 29: Album.php may be accessed via the load balancer.

6. Security Group and Network ACL

- Once our Web Server Instances are operational, we can focus on ensuring their security and accessibility.

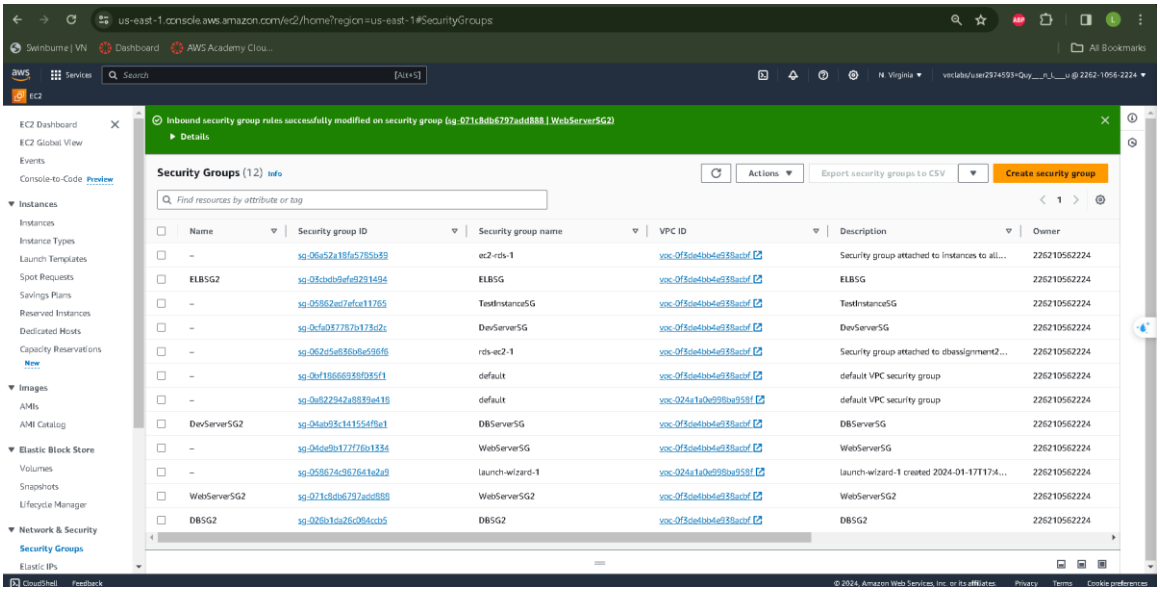

Image 30: Security Groups

- **Web Server Security Group** should only allow access for inbound traffic from Elastic Load Balancer and outbound traffic to the NAT gateway.
- Inbound and Outbound from All Traffics may accpected by **DevServer Security Group**
- The **Database Server Security Group** may receive from both inbound and outbound traffic from the Webserver and Devserver.
- The **Elastic Load Balancer Security Group** controls all inbound and outgoing connections from the Internet gateway.
- In the final setup step, we will construct an ACL to prevent DevServer from transmitting ICMP packets to the WebServer.
  The configured inbound and outbound rules of our ACL will look like this.

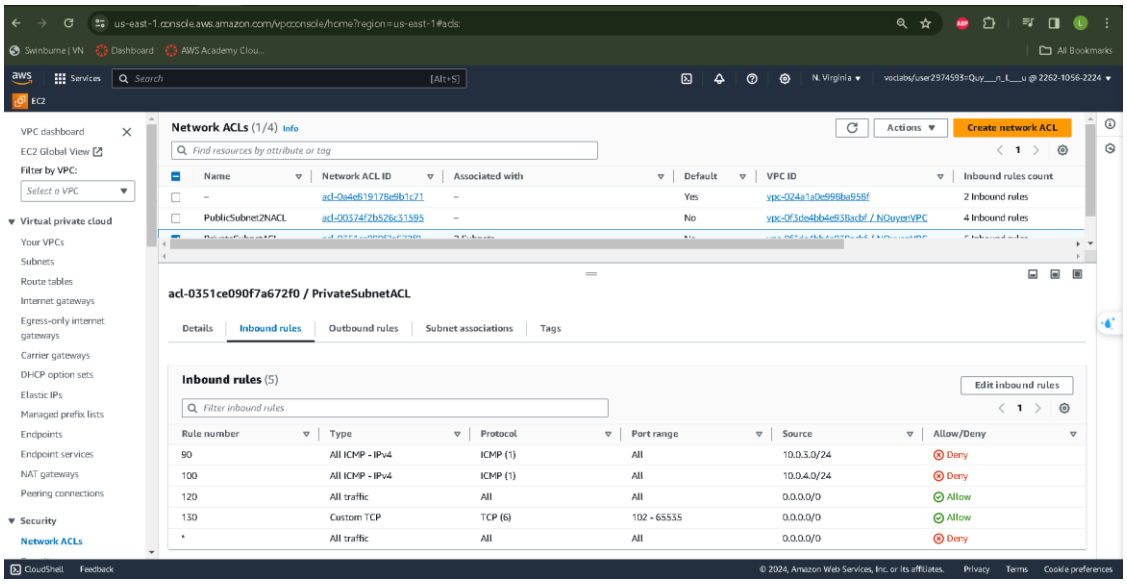

Image 31: Network ACLs

- Using Putty to SSH into our DevServer to ensure created ACL is working as we expected.

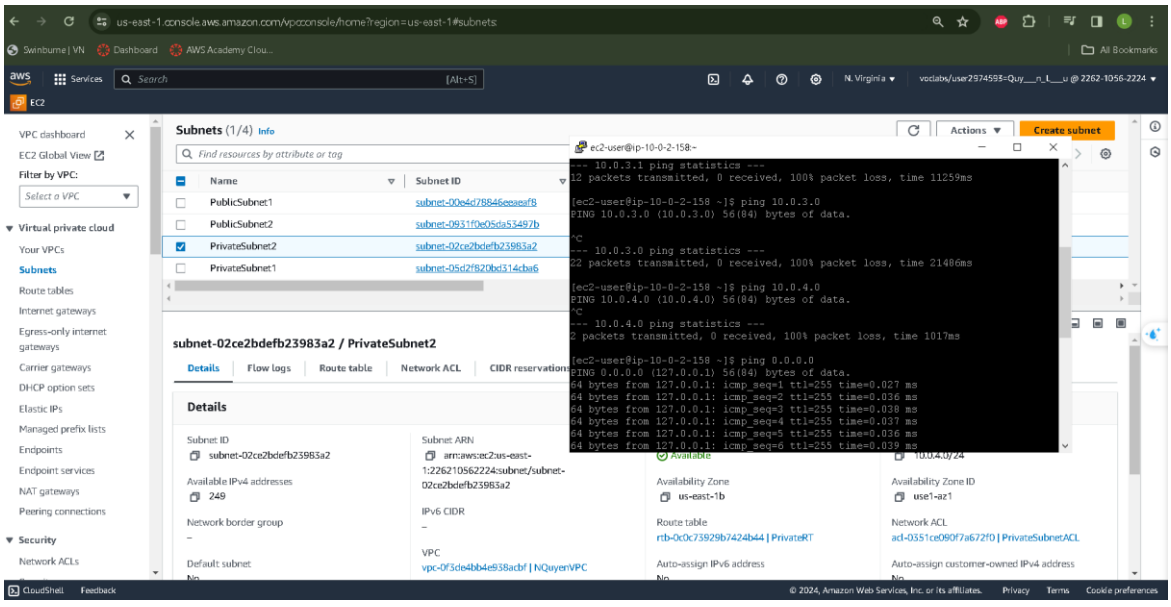

Image 32: ICMP testing

- We can ping the NAT gate way at 0.0.0.0 gate, but ICMP messages sent to private subnets 1 (10.0.3.0) and 2 (10.0.4.0) are unreachable. As a result, our Network ACL is operating perfectly.

7. Testing

- We will test our website's functionality to verify proper setting. This section will focus on the remaining exams, as many have already been completed.

Test 1: Termination(Stop) of EC2 and ASG

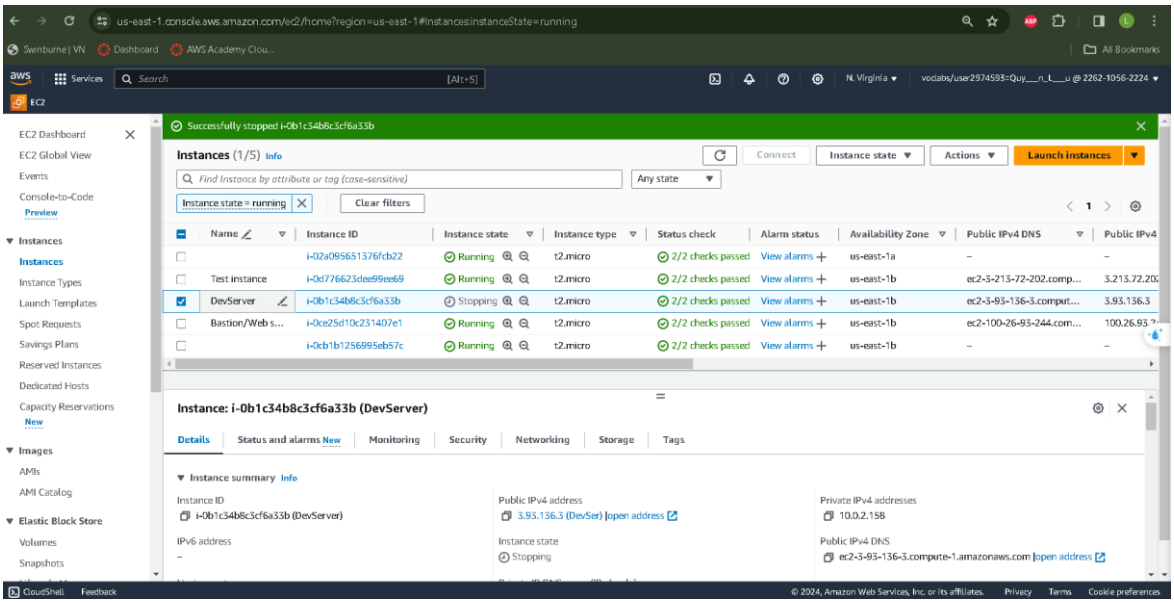

Image 33: A DevServer instance in the Auto Scaling Group is terminated.

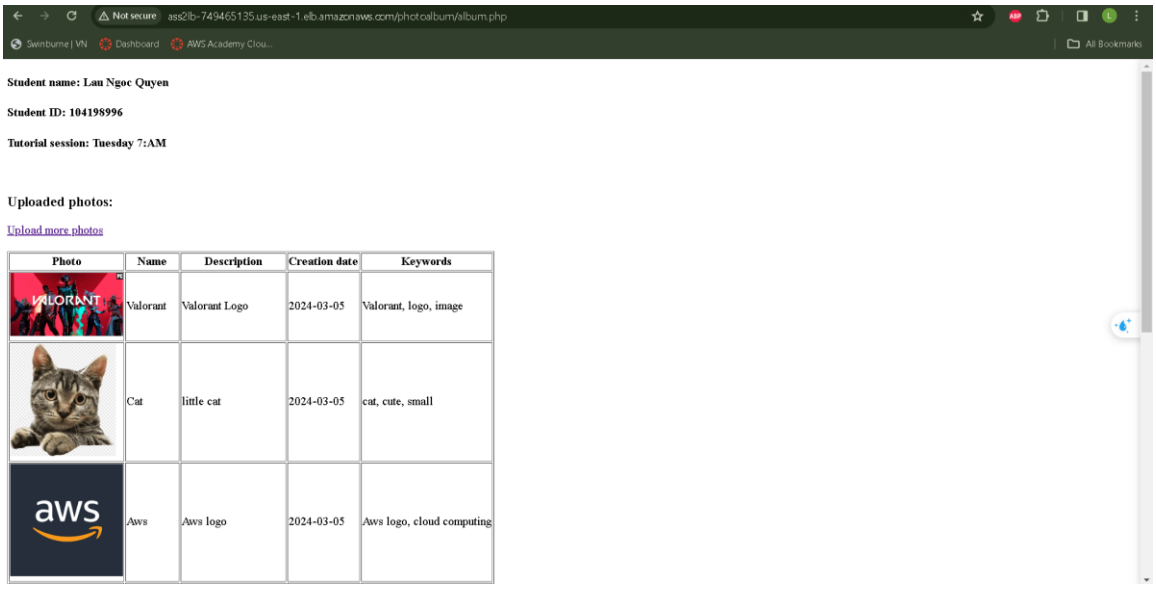

Image 34: The Load Balancer DNS remains available even after the Web Instance is terminated and initialized.

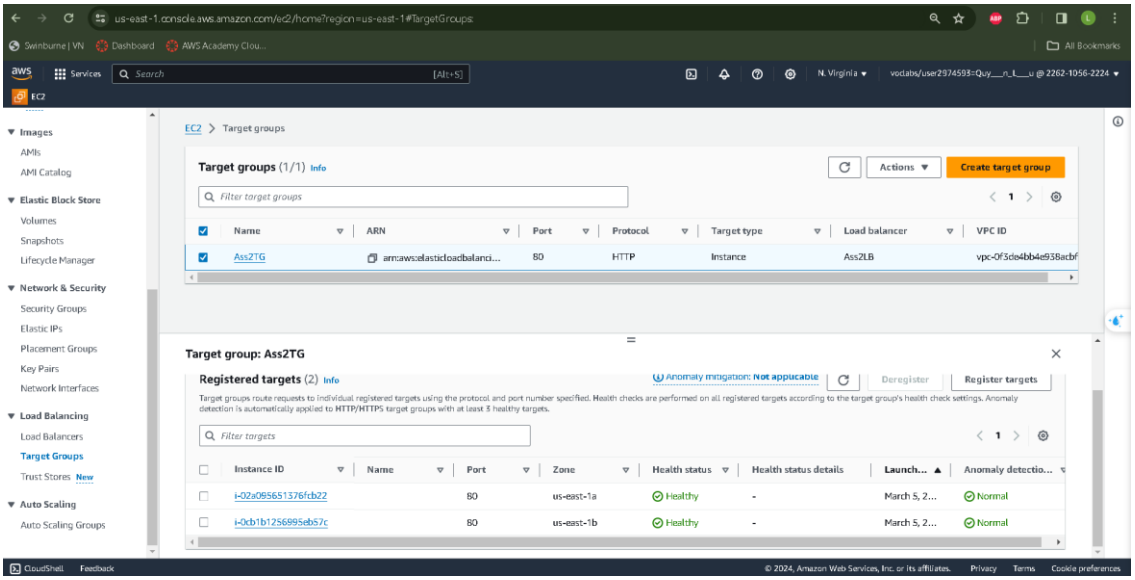

Image 35: Two DevServer Instances are healthy

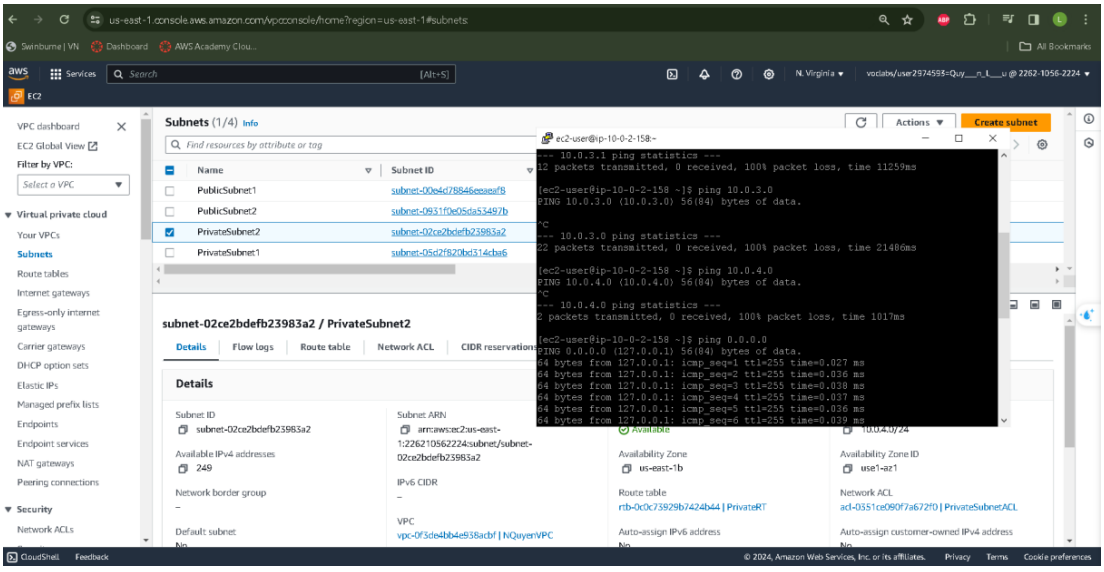

Image 36: Successfully transmit ICMP traffic between the web servers and the development server.

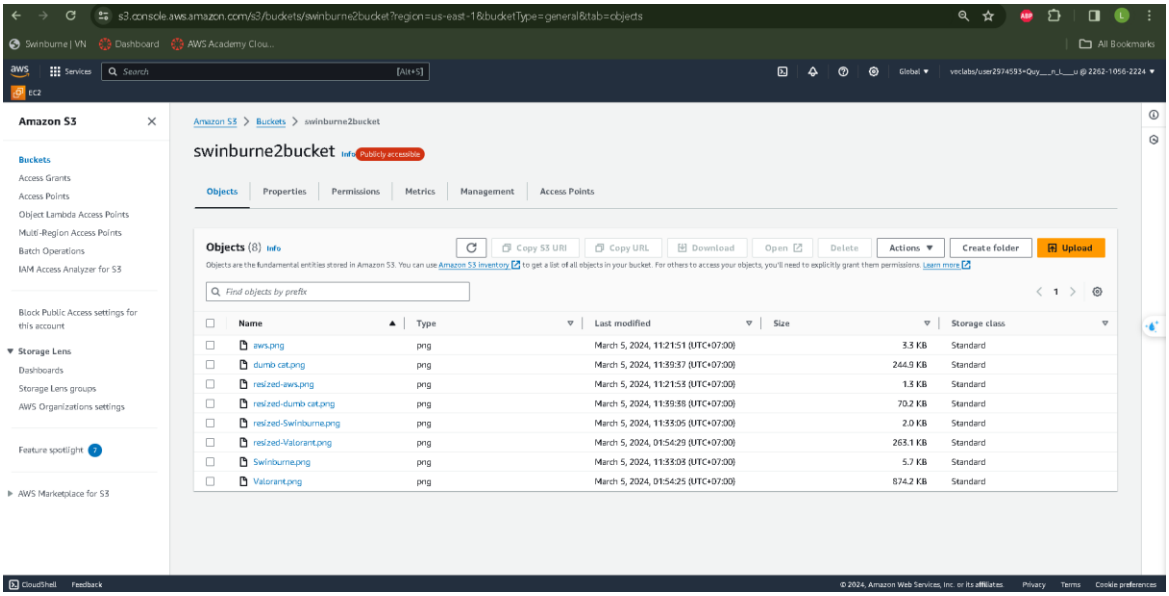

Image 37: Check the S3 bucket to ensure photographs are existed and resized perfectly.

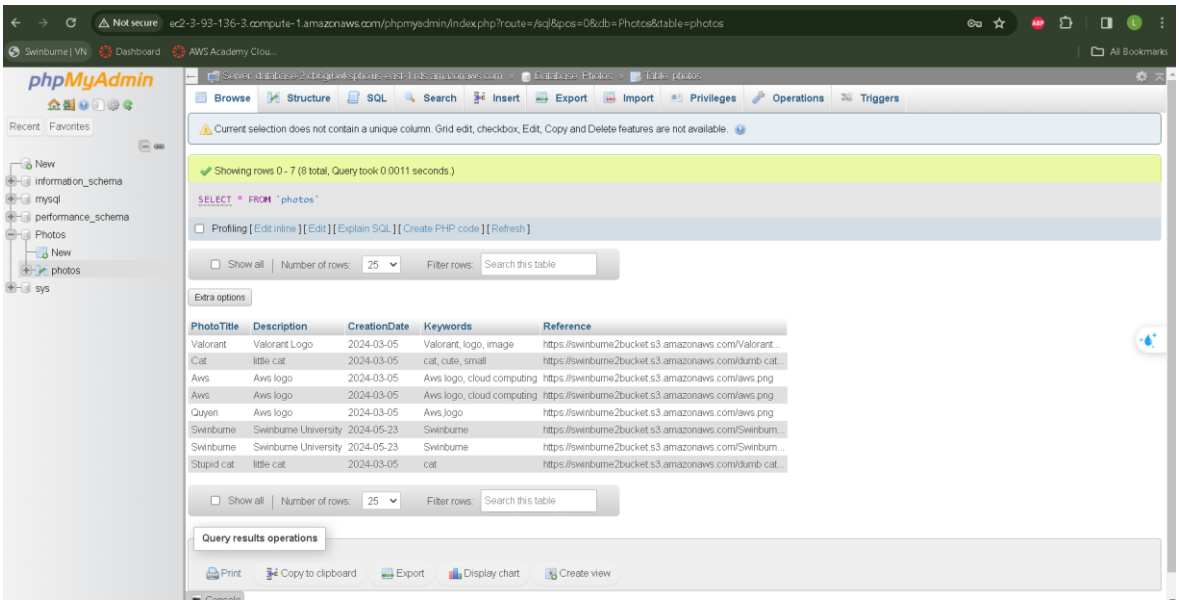

Image 38: Metadata in phpMyAdmin

Link of album.php:

http://ass2lb-749465135.us-east-1.elb.amazonaws.com/photoalbum/album.php

Link of ELB photouploader.php:

http://ass2lb-749465135.us-east-1.elb.amazonaws.com/photoalbum/photouploader.php

Link of DevServer Instance:
http://ec2-3-93-136-3.compute-1.amazonaws.com/

----------*The End*----------