CLOUD COMPUTING ARCHITECTURE – ASSIGNMENT 2

Linh Dan Nguyen – 103488557

Lab Section: Week 10

Date: Saturday 25/3/2023

I. INTRODUCTION

The target of Assignment 2 is to add more features to the Photo Album website deployed in the previous assignment. This assignment requires basic knowledge about VPC and, RDS, S3 Bucket, Security Groups, and new knowledge learned in recent weeks such as Lambda functions and Auto Scaling. In this report, I will summarize all configurations to deploy the website functionality.

II. INFRASTRUCTURE DEPLOYEMENT

A. VPC & ROUTE TABLES

The infrastructure used in this assignment will be a VPC with 4 subnets (2 public and 2 private subnets) divided into 2 Availability Zones. It means each Availability Zone will have 1 public subnet and 1 private subnet.

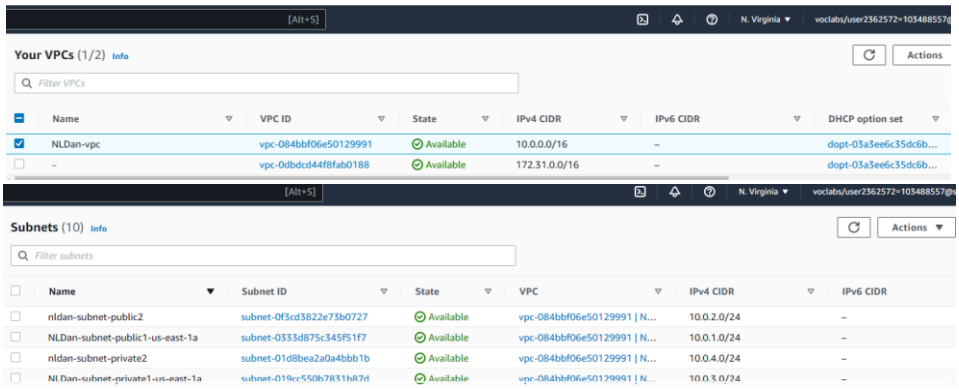

Figure 1. VPC set up & subnets

These subnets will be placed in their correct Route Tables, public subnets will route to the Internet gateway while private subnets will route to the NAT gateway.

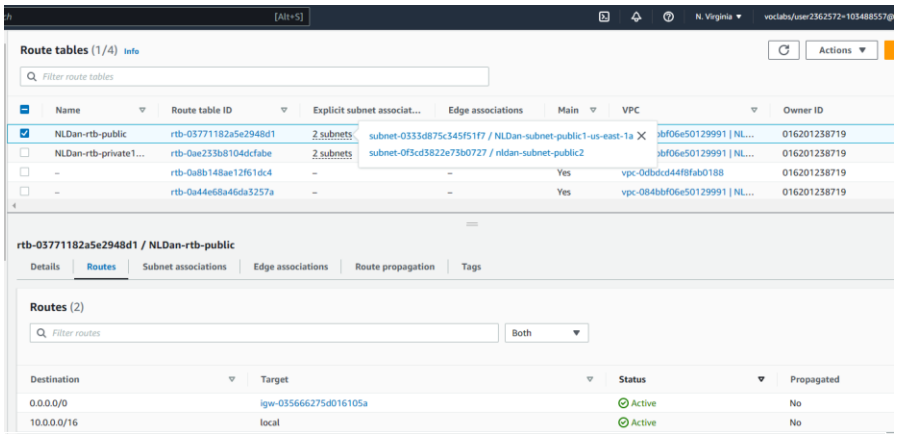

Figure 2. Route table (Public – igw)

Note: I will use the NAT gateway in this assignment instead of launching a NAT instance.

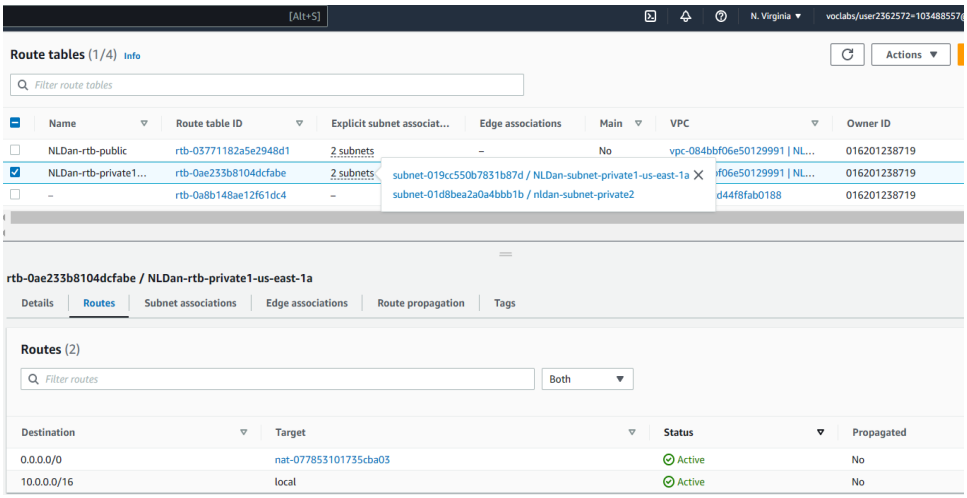

Figure 3. Route table (Private – nat)

I created a NAT gateway and use it to let the instances in private subnets connect to the Internet. Some additional EC2 instances would be added to the private subnets in the Auto Scaling step (see part **V – B** for details).

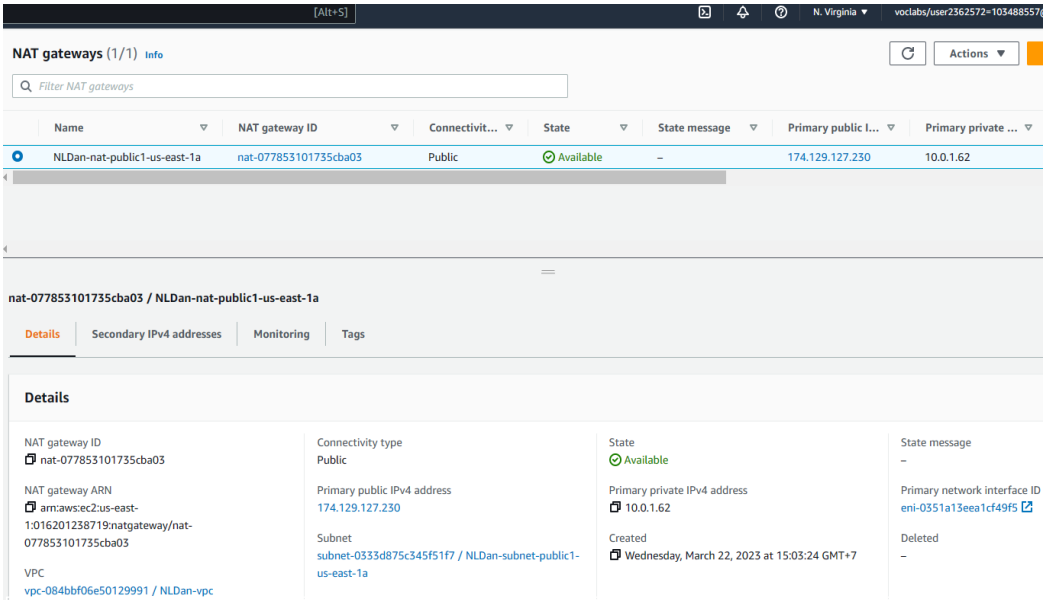

Figure 4. NAT gateway

B. SECURITY GROUPS

In this part, I created 4 different Security groups with rules following **the least-privilege principle** (I did not allow all traffic, I only allow necessary traffic). I did not create a security group for the NAT instance because I only used the NAT gateway (the instructions accept both ways). The detailed rules of these security groups will be attached below.

DBServerSG – the RDS instance security group. It allows MySQL traffic to the database.

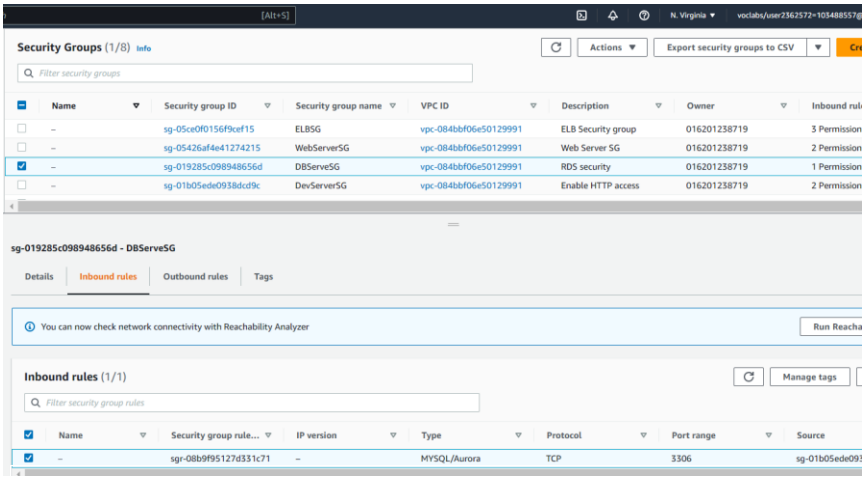

Figure 5. RDS security group

WebServerSG – the scaling instances security group. It will be used in the Auto Scaling stage.

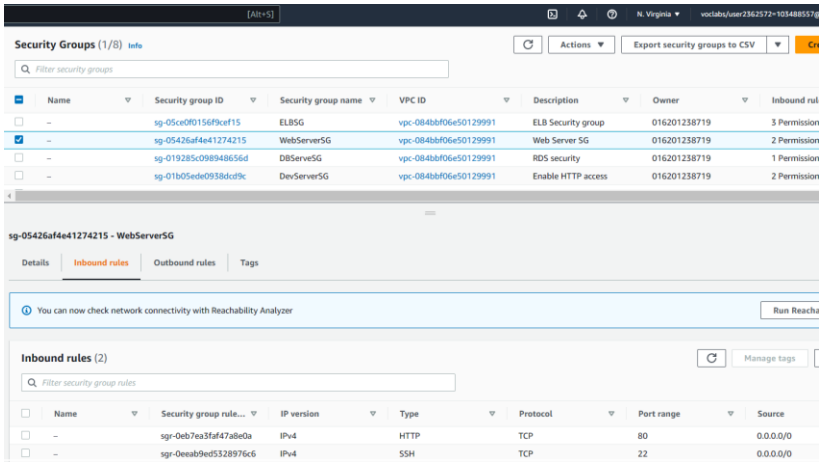

Figure 6. Security group for the scaling WebServer

DevServerSG – DevServer EC2 instance security group, the main instance of this assignment. It allows HTTP access, so the instance can access the Internet.

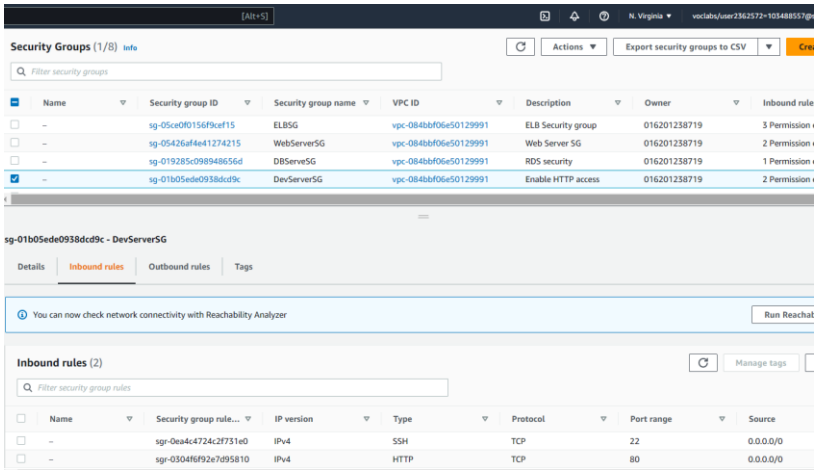

Figure 7. DevServer security group

ELBSG – the Load Balancer security group

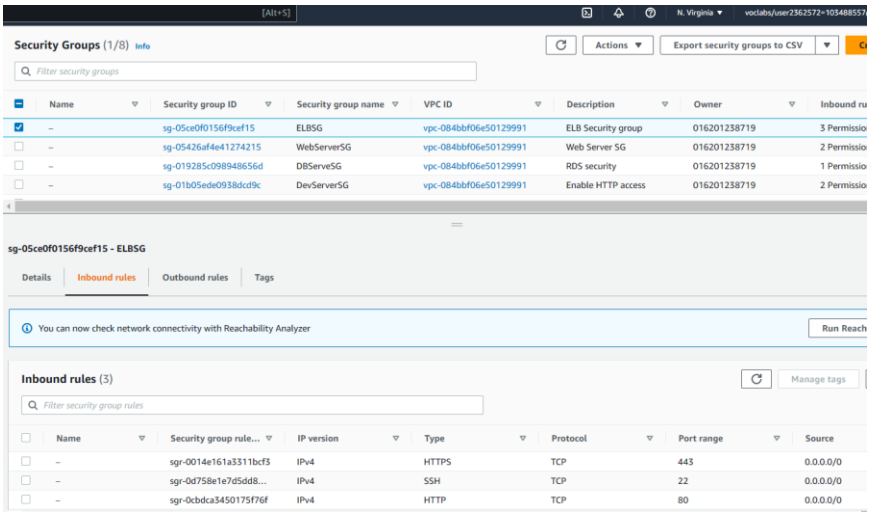

Figure 8. Load Balancer security group

C. NACL

I configured a Network ACL to block **bidirectional traffic from/to the DevServer** instance. When the security rules are applied, you cannot use the DevServer to ping other instances, and other instances also cannot ping the DevServer.

Overall, the rules of this NACL contain the **SSH and TCP rules** to allow SSH access through PuTTY. The last rule **ICMP** will block the bidirectional traffic, so we need to add the ICMP rule for both inbound and outbound rules.

Inbound ICMP deny: It will prevent all outside traffic from being sent to the DevServer.

Outbound ICMP deny: It will prevent the DevServer from sending any requests outside.

After configuring, the NACL will be associated with the **Public Subnet 2**, in which the DevServer instance resides.

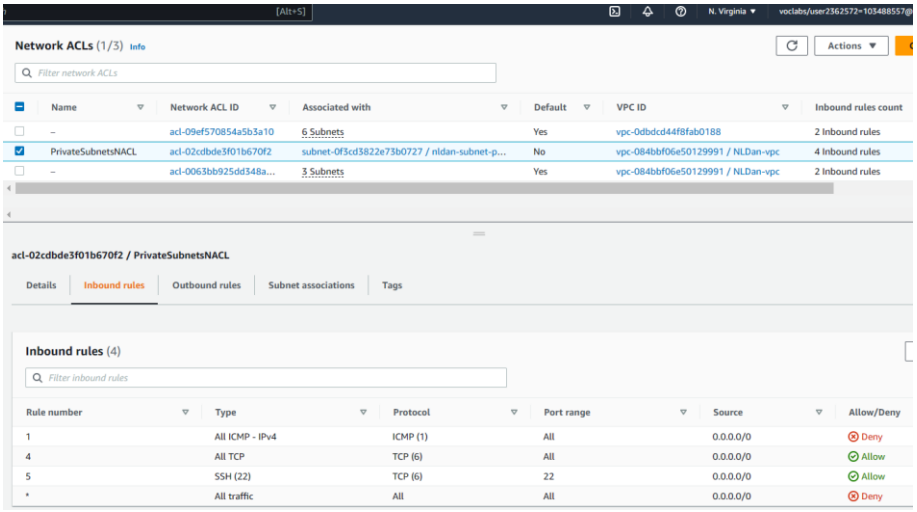

Figure 9. Inbound rules of the NACL

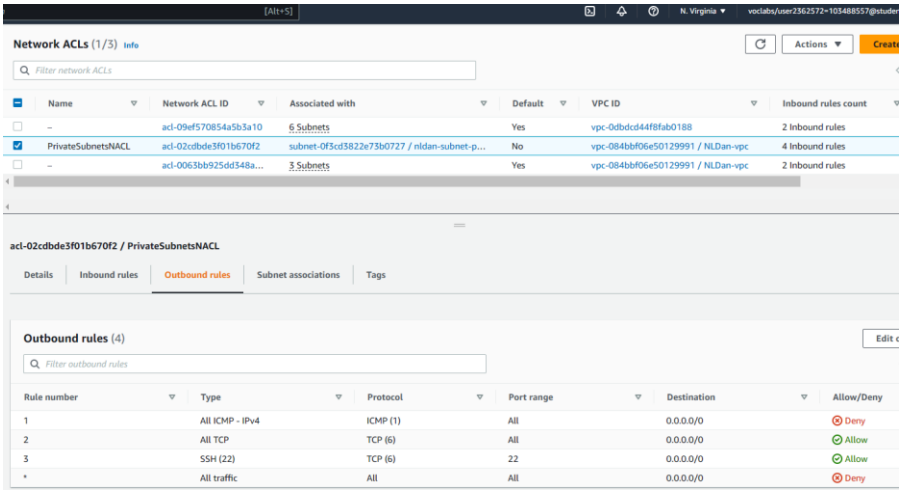

Figure 10. Outbound rules of the NACL

D. EC2 INSTANCES

Because I did not use the NAT instance, DevServer is the only EC2 instance that I needed to launch. The settings of this instance were simple and similar to all previous guided labs. There was a special point of this instance that the instance role was **LabRole** – an IAM role that allows all actions related to the EC2 (IAM roles will be discussed in part **IV**).

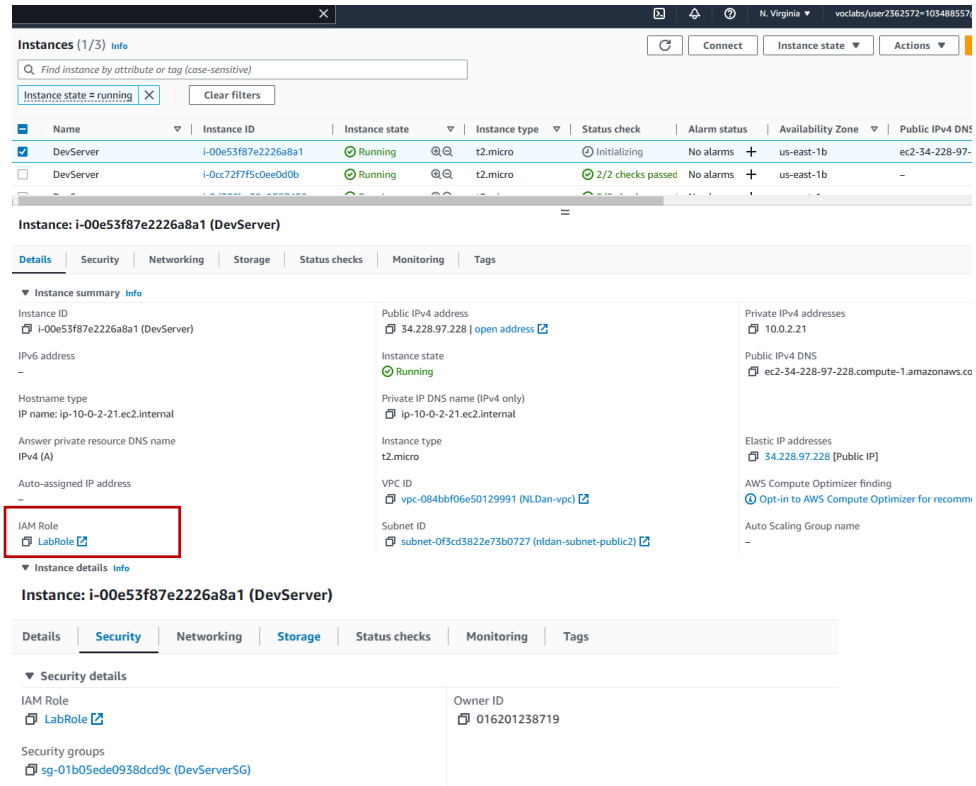

Figure 11. Details of DevServer instance

E. RDS DATABASE INSTANCE

In this stage, I created a MySQL database. This database will hold the responsibility of storing the metadata of all uploaded photos. I described the process of creating a new database in detail in the previous Assignment 1B, so I will not repeat the process here, I will only show the database summary after it is activated.

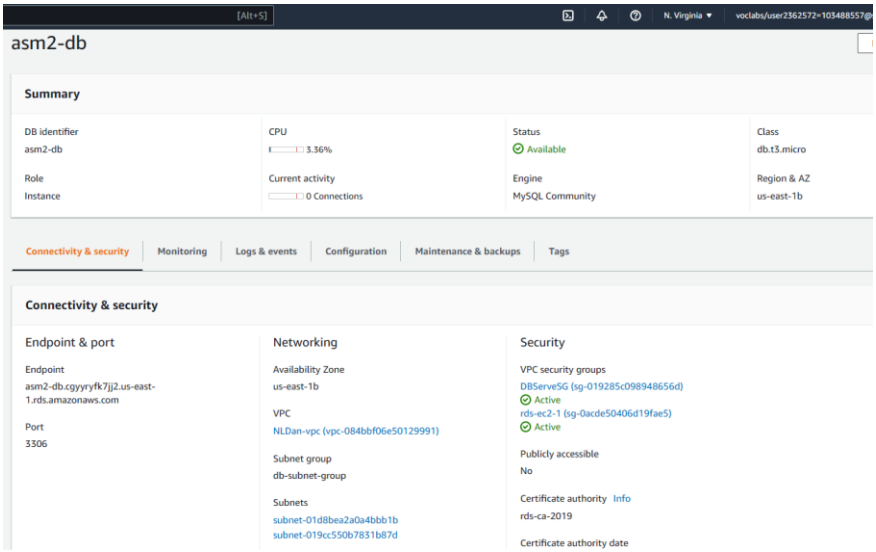

Figure 12. Database settings summary

S3 BUCKET

S3 is a storage service which was used to store all photos uploaded on the website. Then, the photo references will be copied and pasted into the metadata table in the database to display photos on the website. I modified the **Bucket Policy** of this Bucket to restrict access to a specific HTTP referrer. Before doing that, I changed the permission of the Bucket to Publicly accessible to allow people to view the photos on the Album website.

Below is the Bucket with **Public** access

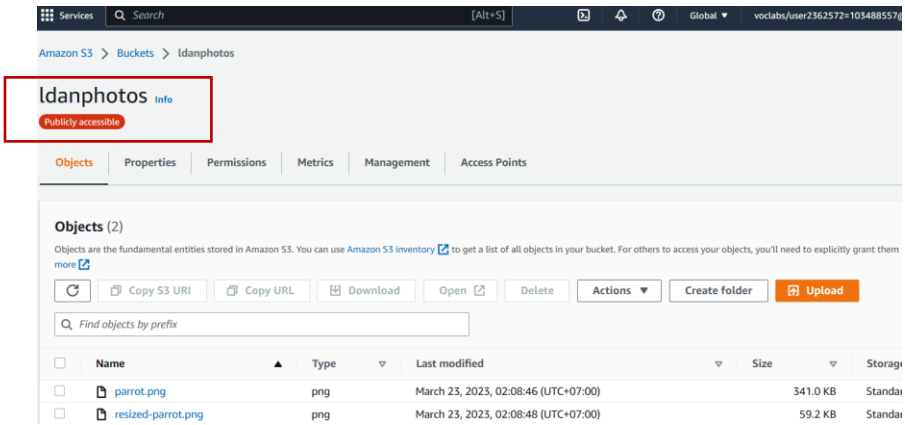

Figure 13: A public S3 Bucket

I wrote the **Bucket Policy** to restrict access (not everyone in the public can access the photos)

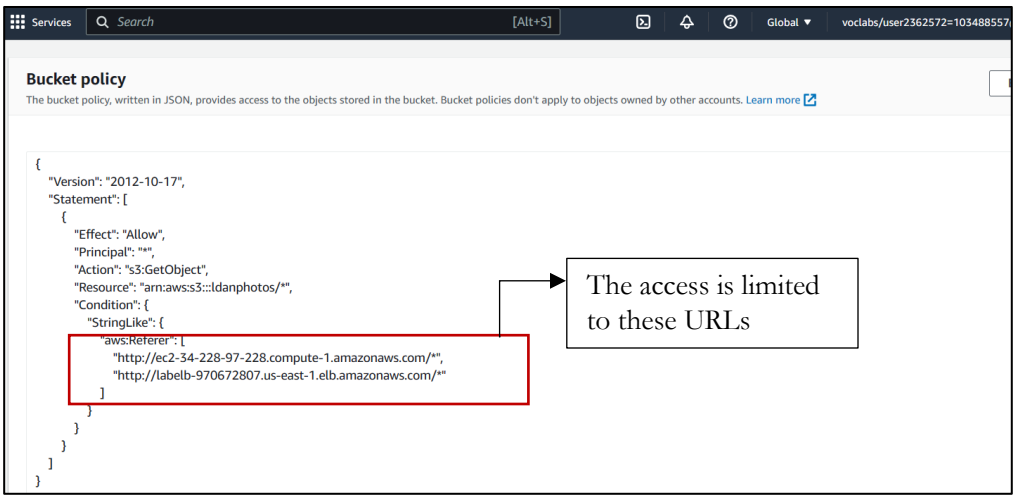

Figure 14: S3 Bucket Policy

Thanks to this policy, public access was restricted. Although this bucket was public, people cannot view the photos via the Object URL. They can only view them on the website specified in the **Referer** section of the policy.

Figure 15: The public access has been restricted by the policy

III. PHOTO ALBUM WEBSITE

A. INSERT SAMPLE DATA TO THE RDS

Before creating a Lambda function to automatically add new photos to the S3 Bucket when users upload their photos to the website, I inserted a sample data row into the database to ensure that everything (S3 bucket and RDS) worked fine.

First, I **SSH** into the **DevServer** instance and log in to my database. Then I inserted a row with the following values into the database.

```
MySQL [asm2]> insert into photos (title, description, creationdate, keywords, reference) values ('Parro
t', 'A bird', '20230203', 'colorful','https://ldanphotos.s3.amazonaws.com/parrot.png');
Query OK, 1 row affected (0.01 sec)
```

Figure 16: Insert into the database using MySQL command

Then, I checked the **phpmyadmin** to see if new metadata had been added.

+ Options

title	description	creationdate	keywords	reference
Parrot	A bird	2023-02-03	colorful	https://ldanphotos.s3.amazonaws.com/parrot.png

Figure 17: Metadata of the the new photo in phpmyadmin

Student name: Linh Dan Nguyen

Student ID: 103488557

Tutorial session: Sunday 09:15AM

Uploaded photos:

Upload more photos

Photo	Name	Description	Creation date	Keywords
	Parrot	A bird	2023-02-03	colorful

Figure 18: New photo displayed on the screen

B. LAMBDA FUNCTION

In this stage, we move to configure **Lambda** – a new service that we have never used in the previous assignment. A Lambda function is created to automatically download the uploaded photos and store them in a sub-folder **'uploads'** in the **photoalbum** folder. Then, the downloaded photos will be resized and uploaded to the S3 bucket with a pre-defined name.

Open the **Lambda** service → **Create Function**. The function configurations are shown below

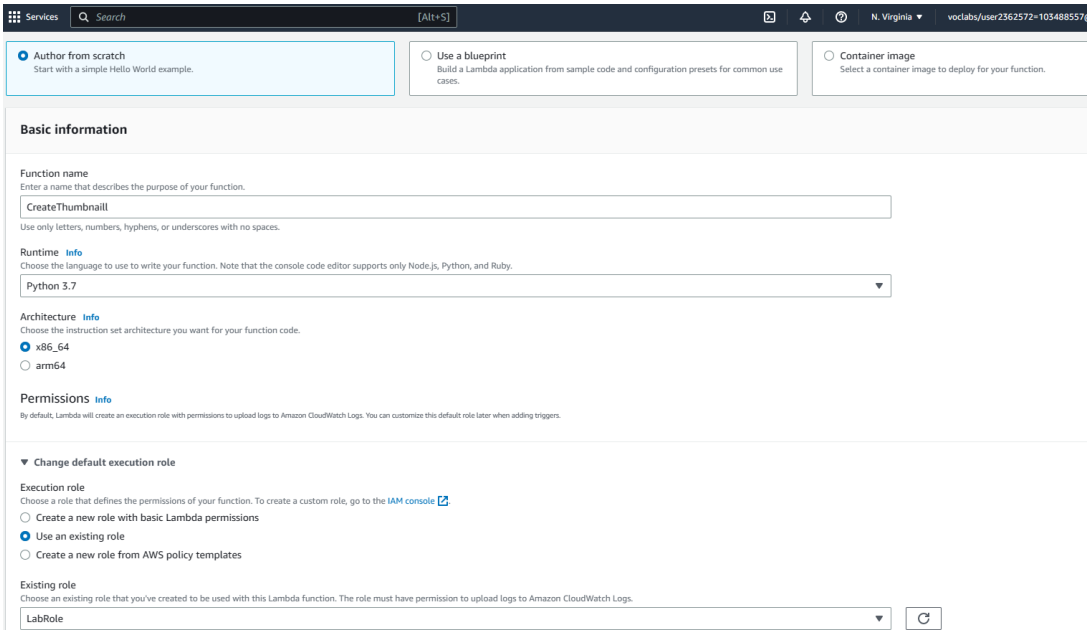

Figure 19: The Lambda function configurations

After creating the function, I found the **Code** tab and uploaded the **lambda-deployment-package.zip** – this package will define what the function can do. The code in the package will:

1. Download the uploaded photo
2. Resize the photo
3. Upload the resized photo to S3 with the new name using the format **'resized-filename'**

```
def lambda_handler(event, context):
    try:
        bucket_name = event["bucketName"]
        file_name = event["fileName"]
        key = unquote_plus(file_name)
        tmpkey = key.replace('/', '')
        download_path = '/tmp/{}{}'.format(uuid.uuid4(), tmpkey)
        upload_path = '/tmp/resized-{}'.format(tmpkey)
        s3_client.download_file(bucket_name, key, download_path)
        resize_image(download_path, upload_path)
        s3_client.upload_file(upload_path, bucket_name, 'resized-{}'.format(tmpkey))
    except ClientError as e:
        return "Lambda's error: Error code: {}, HTTPStatusCode: {}, Message: {}".format(e.response['Error']['Code'],
        e.response['ResponseMetadata']['HTTPStatusCode'], e.response['Error']['Message'])
    except OSError as ose:
        return "Lambda's error: Message: {}".format(ose)
```

Figure 20: Code used for Lambda function deployment

In the **photoalbum** folder, there is a PHP file that controls the photo uploader. It will add new metadata to the database and invoke the above Lambda function to upload a new photo to the S3.

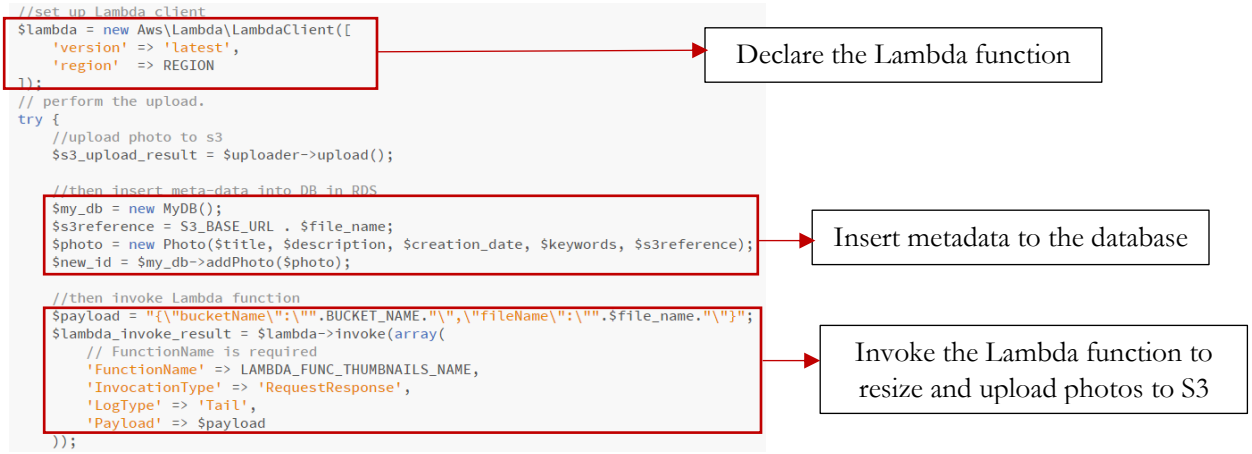

Figure 21: photouploader.php file

Now I will try to upload a photo from my computer to see whether the Lambda function and the code in PHP files work correctly.

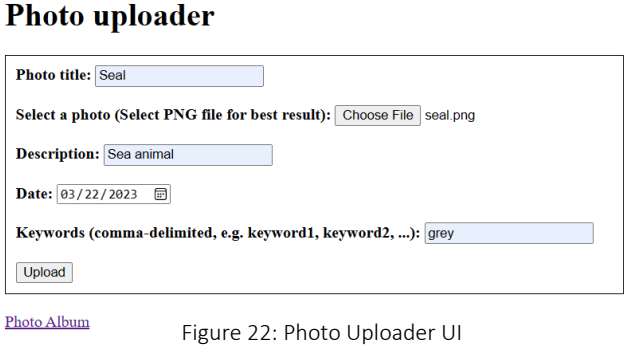

Figure 22: Photo Uploader UI

Result

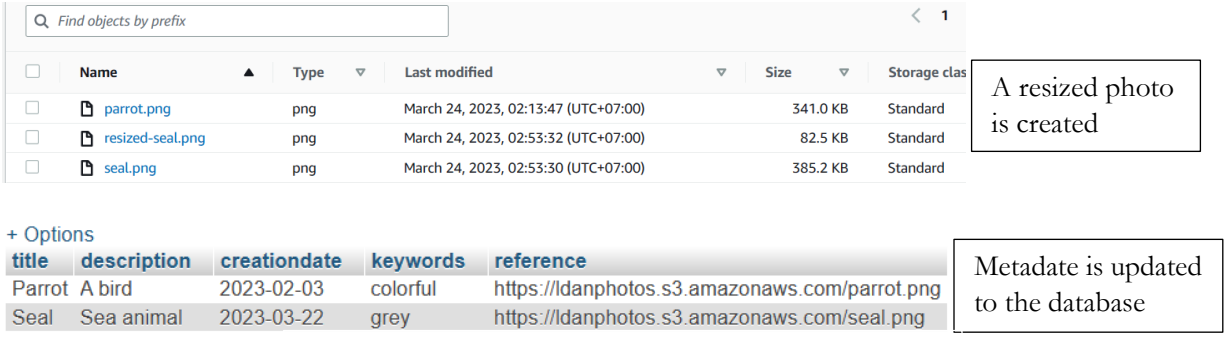

Figure 23: New data is updated

Student name: Linh Dan Nguyen

Student ID: 103488557

Tutorial session: Sunday 09:15AM

Uploaded photos:

Upload more photos

Photo	Name	Description	Creation date	Keywords
	Parrot	A bird	2023-02-03	colorful
	Seal	Sea animal	2023-03-22	grey

Figure 24: The new photo is displayed correctly

IV. IAM ROLES

The IAM roles will be used to control the permissions of different user groups to access different services. There is a pre-defined role which we can use to restrict access to some specific services. In this assignment, IAM roles are assigned to the EC2 and the Lambda function.

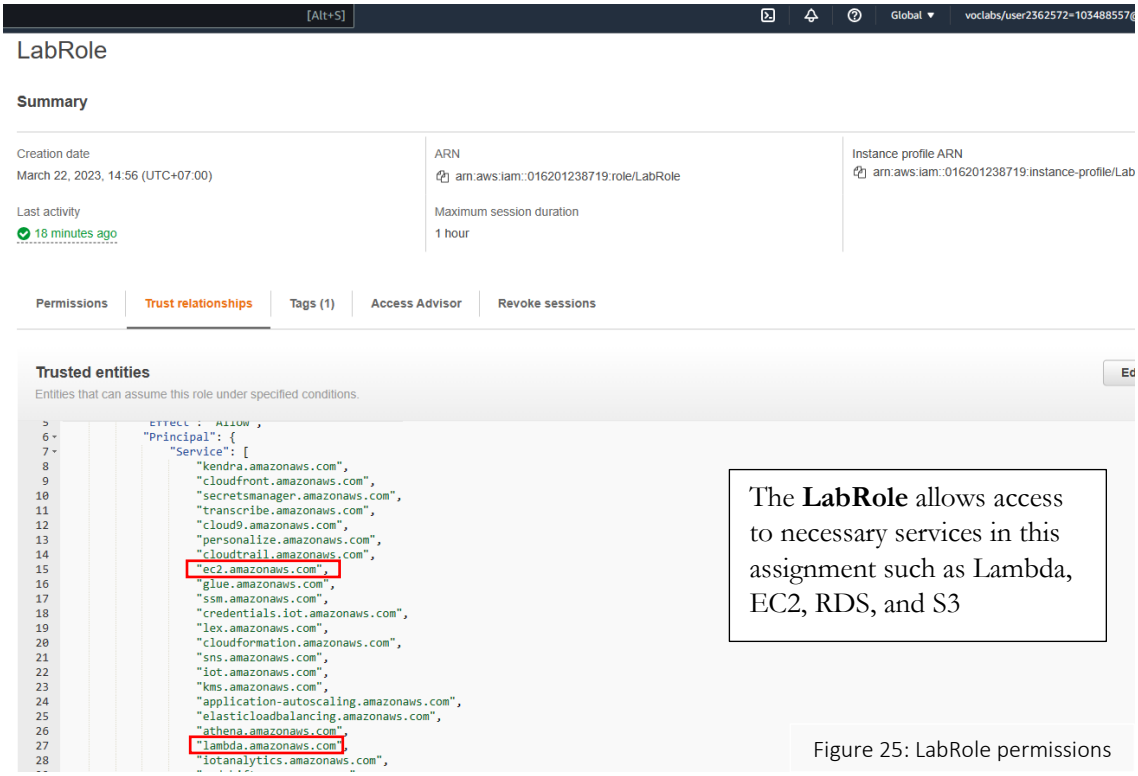

Figure 25: LabRole permissions

The **LabRole** can be attached to the **EC2** when we launch the instance

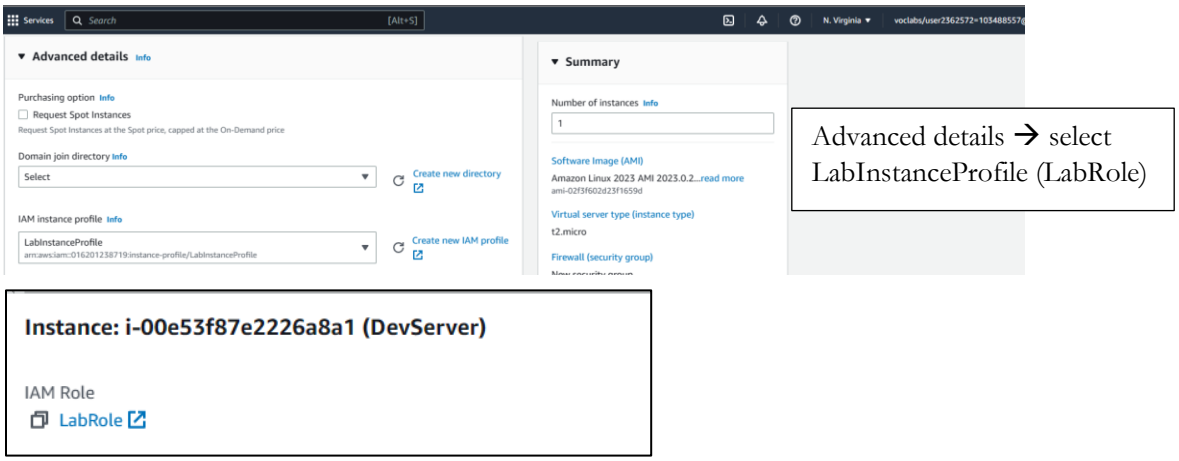

Figure 26: Attach the IAM role to EC2 instance

Similarly, the **LabRole** can also be attached to the **Lambda** function in the creating process.

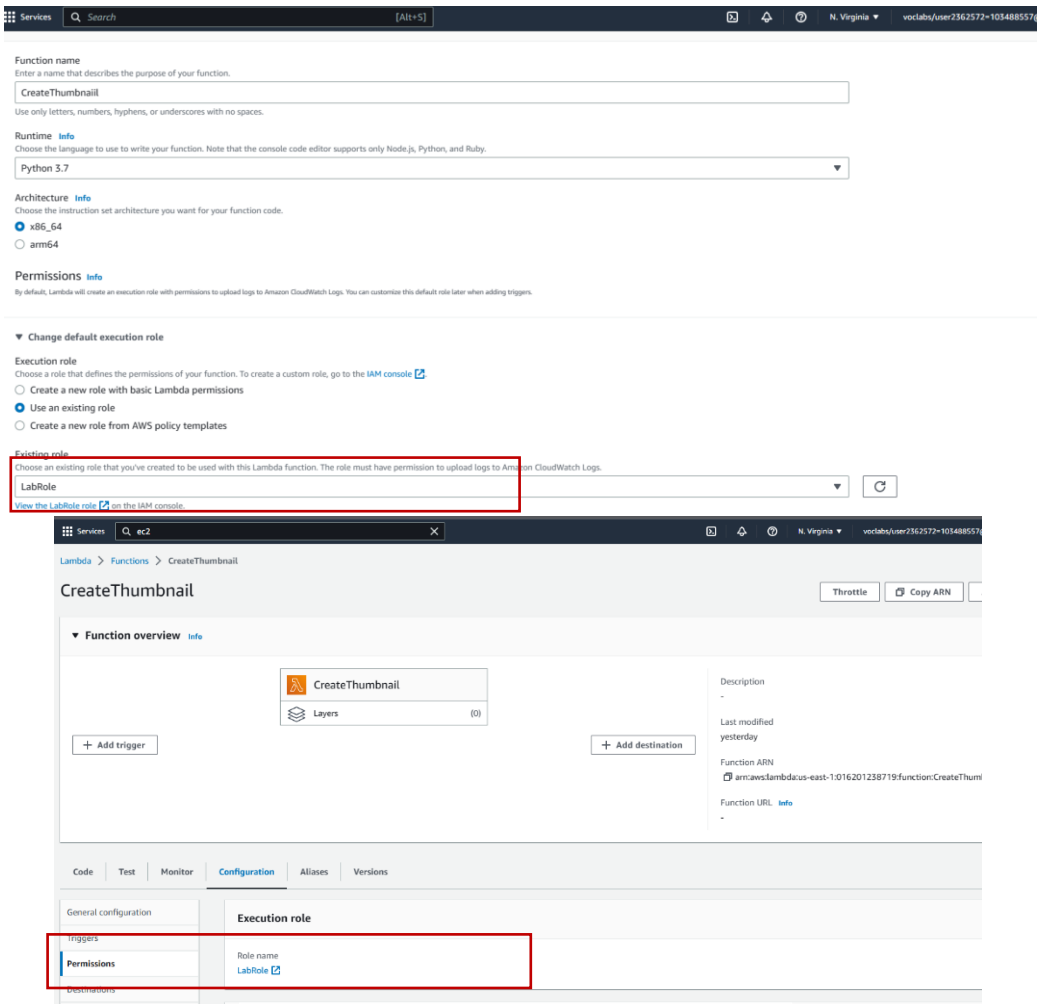

Figure 27: Attach the IAM role to Lambda

V. HIGH AVAILABLE ENVIRONMENT

A. LOAD BALANCER

Before creating a Load Balancer, I need a **Target Group** to determine where traffic will be sent to. I only filled in its name, VPC, and other required information. I did not register any targets. The targets will be added later by the Auto Scaling Group.

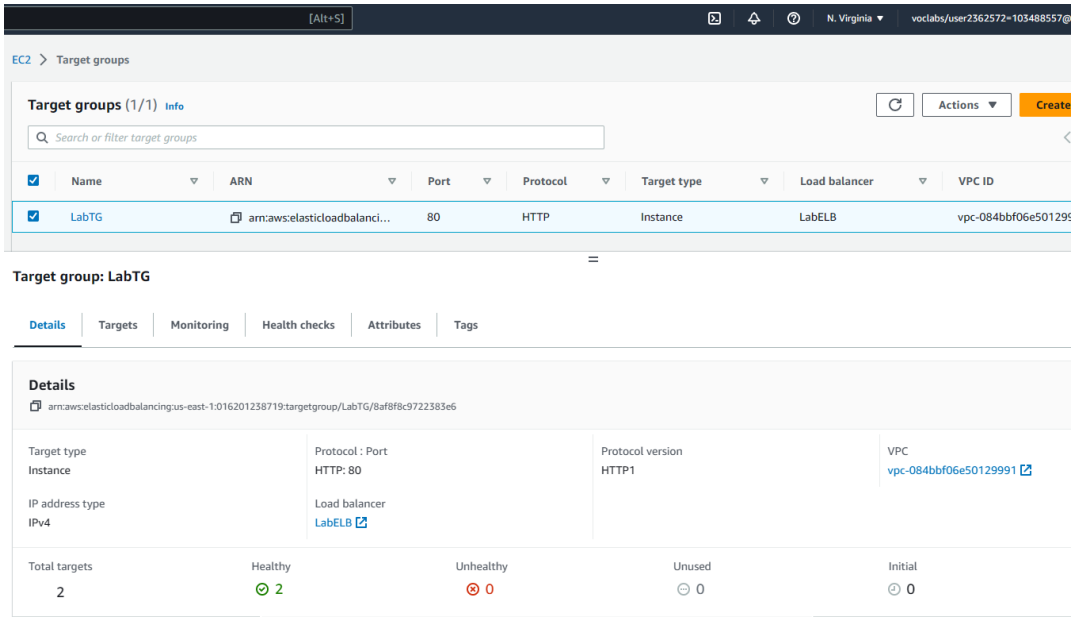

Figure 28: Target Group

Next, I created an **Application Load Balancer** in two **Public subnets** of my VPC. This load balancer will use the **ELBSG** security group defined in part 1 and the **target groups** created above.

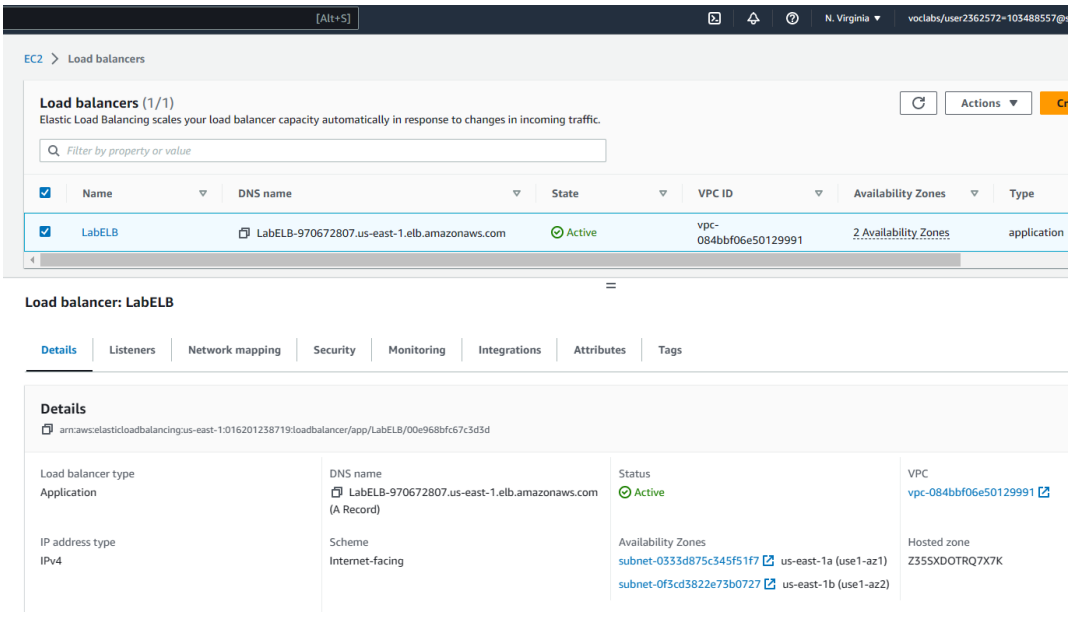

Figure 29: Load Balancer

B. AUTO SCALING GROUP

If the Target group defines where the traffic will be sent to, the **Launch Configuration** will determine how new instances will be created to handle the increased traffic. Below is the Launch Configuration created to prepare for the Auto Scaling Group. There are some key points that we should notice about this Launch Configuration:

- It was created by the **AMI of the DevServer**
- It used the **WebServerSG security group**
- Its IAM role is **LabRole**

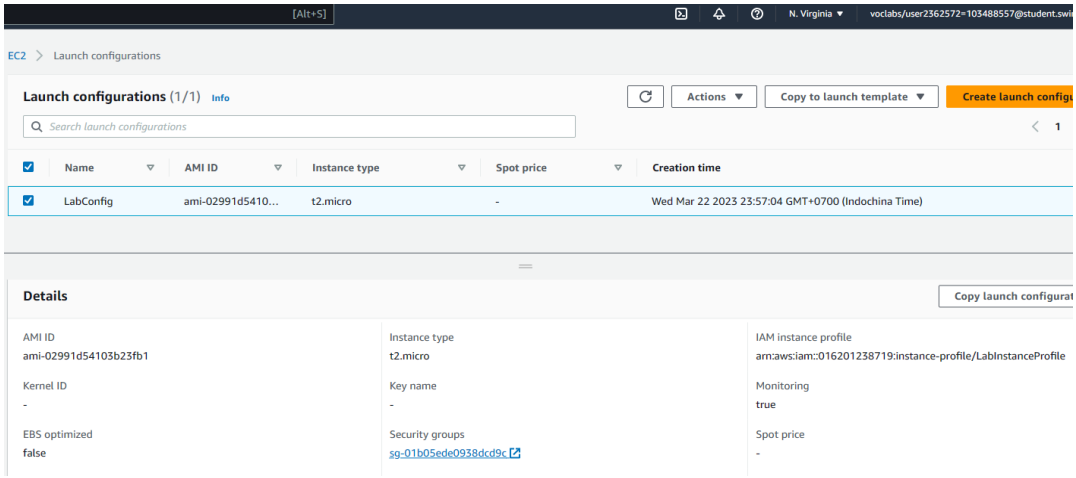

Figure 30: Launch Configurations

Finally, I created the **Auto Scaling Group** to automatically scale in/out to satisfy the demand of the DevServer instance.

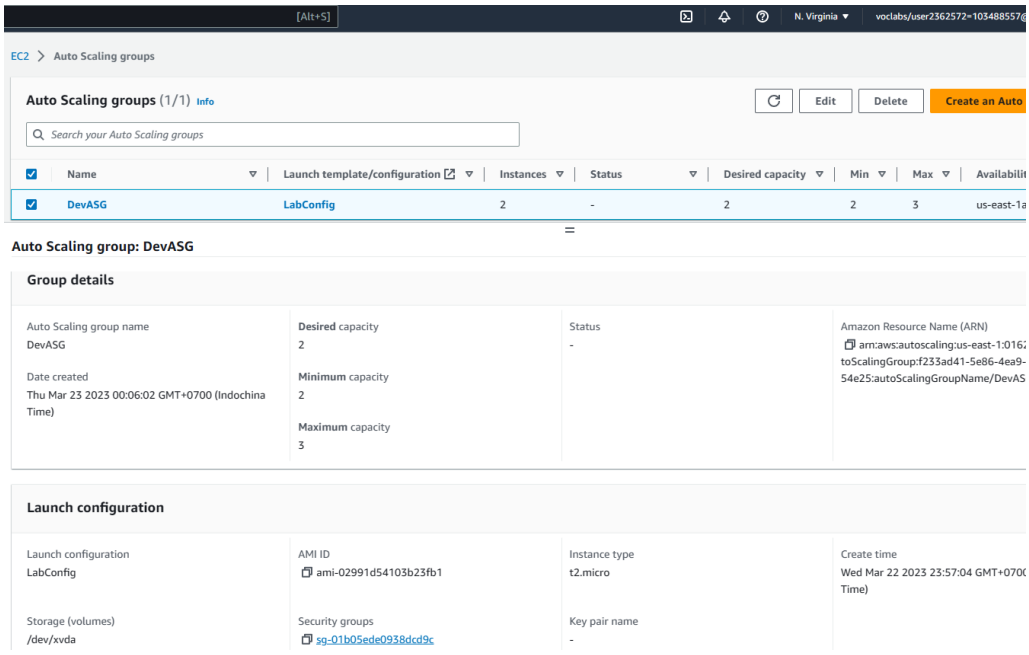

Figure 31: Auto Scaling Group

There are some important points about this **Auto Scaling Group**:

- The **Desired** and **Minimum capacity = 2**, the **Maximum capacity = 3**
- It used the **WebServerSG** security group
- It used the **LabConfig** Launch Configurations (created above)
- It used **2 Private Subnets** of my VPC

Result

Two scaling instances are added, and they are **healthy**.

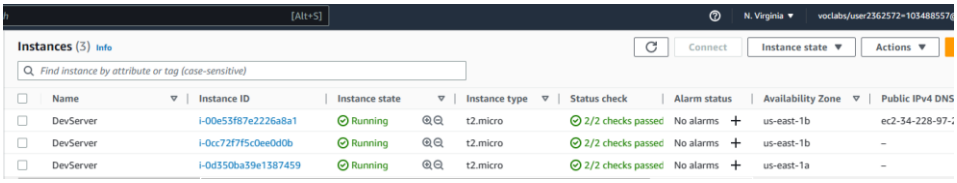

Figure 32: Auto Scaling Group added 2 new instances

I can access the DevServer using the **DNS name** of the **Load Balancer**

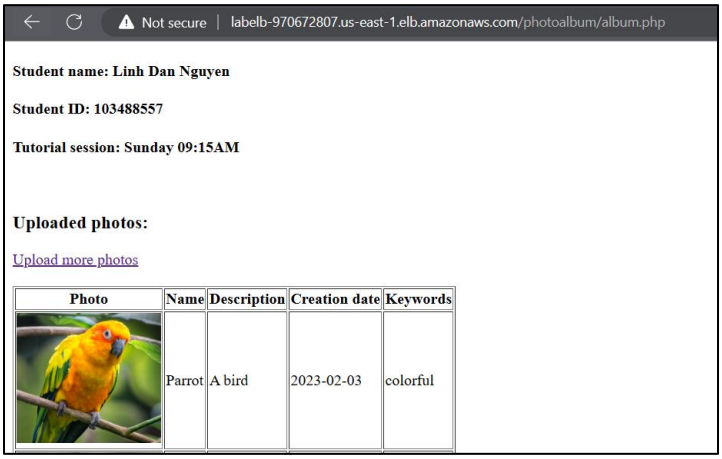

Figure 33: Access the web via Load Balancer DNS

VI. REFERENCES

URL of the website

http://labelb-970672807.us-east-1.elb.amazonaws.com/photoalbum/album.php

http://ec2-34-228-97-228.compute-1.amazonaws.com/photoalbum/album.php