

# CLOUD COMPUTING ARCHITECTURE – ASSIGNMENT 2

Linh Dan Nguyen – 103488557

Lab Section: Week 10

Date: Saturday 25/3/2023

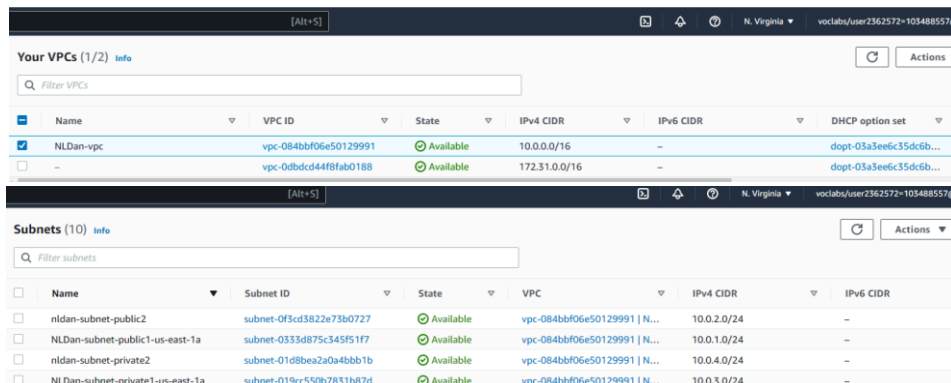
## I. INTRODUCTION

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 DEPLOYMENT

### 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.



The screenshot displays the AWS Management Console interface for VPC and Subnets. The top section, 'Your VPCs (1/2)', shows a table with columns: Name, VPC ID, State, IPv4 CIDR, IPv6 CIDR, and DHCP option set. The first entry is 'NLDan-vpc' with VPC ID 'vpc-084bbf06e50129991', State 'Available', IPv4 CIDR '10.0.0.0/16', and DHCP option set 'dopt-03a3ee6c35dc6b...'. The second entry is 'vpc-0dbdc4448fab0188' with VPC ID 'vpc-0dbdc4448fab0188', State 'Available', IPv4 CIDR '172.31.0.0/16', and DHCP option set 'dopt-03a3ee6c35dc6b...'. The bottom section, 'Subnets (10)', shows a table with columns: Name, Subnet ID, State, VPC, IPv4 CIDR, and IPv6 CIDR. The first entry is 'nldan-subnet-public2' with Subnet ID 'subnet-0f3cd3822e73b0727', State 'Available', VPC 'vpc-084bbf06e50129991 | N...', IPv4 CIDR '10.0.2.0/24', and IPv6 CIDR '-'. The second entry is 'NLDan-subnet-public1-us-east-1a' with Subnet ID 'subnet-0333d875c345f51f7', State 'Available', VPC 'vpc-084bbf06e50129991 | N...', IPv4 CIDR '10.0.1.0/24', and IPv6 CIDR '-'. The third entry is 'nldan-subnet-private2' with Subnet ID 'subnet-01d8bea2a0a4b0b1b', State 'Available', VPC 'vpc-084bbf06e50129991 | N...', IPv4 CIDR '10.0.4.0/24', and IPv6 CIDR '-'. The fourth entry is 'NLDan-subnet-private1-us-east-1a' with Subnet ID 'subnet-019ec550b78f31b87d', State 'Available', VPC 'vpc-084bbf06e50129991 | N...', IPv4 CIDR '10.0.3.0/24', and IPv6 CIDR '-'. The interface includes search filters and an 'Actions' button for each section.

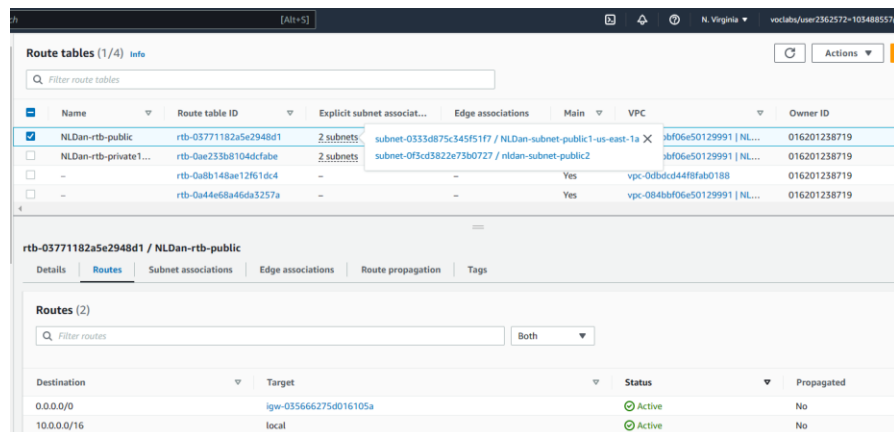
| Name      | VPC ID                | State     | IPv4 CIDR     | IPv6 CIDR | DHCP option set        |
|-----------|-----------------------|-----------|---------------|-----------|------------------------|
| NLDan-vpc | vpc-084bbf06e50129991 | Available | 10.0.0.0/16   | -         | dopt-03a3ee6c35dc6b... |
| -         | vpc-0dbdc4448fab0188  | Available | 172.31.0.0/16 | -         | dopt-03a3ee6c35dc6b... |

| Name                             | Subnet ID                 | State     | VPC                          | IPv4 CIDR   | IPv6 CIDR |
|----------------------------------|---------------------------|-----------|------------------------------|-------------|-----------|
| nldan-subnet-public2             | subnet-0f3cd3822e73b0727  | Available | vpc-084bbf06e50129991   N... | 10.0.2.0/24 | -         |
| NLDan-subnet-public1-us-east-1a  | subnet-0333d875c345f51f7  | Available | vpc-084bbf06e50129991   N... | 10.0.1.0/24 | -         |
| nldan-subnet-private2            | subnet-01d8bea2a0a4b0b1b  | Available | vpc-084bbf06e50129991   N... | 10.0.4.0/24 | -         |
| NLDan-subnet-private1-us-east-1a | subnet-019ec550b78f31b87d | Available | vpc-084bbf06e50129991   N... | 10.0.3.0/24 | -         |

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.



The screenshot displays the AWS Management Console interface for Route Tables. The top section, 'Route tables (1/4)', shows a table with columns: Name, Route table ID, Explicit subnet associ..., Edge associations, Main, VPC, and Owner ID. The first entry is 'NLDan-rtb-public' with Route table ID 'rtb-03771182a5e2948d1', Explicit subnet associ... '2 subnets', Edge associations 'subnet-0333d875c345f51f7 / NLDan-subnet-public1-us-east-1a', Main 'Yes', VPC 'vpc-084bbf06e50129991 | N...', and Owner ID '016201238719'. The second entry is 'NLDan-rtb-private1...' with Route table ID 'rtb-0ae253b8104dcabe', Explicit subnet associ... '2 subnets', Edge associations 'subnet-0f3cd3822e73b0727 / nldan-subnet-public2', Main 'Yes', VPC 'vpc-0dbdc4448fab0188', and Owner ID '016201238719'. The third entry is 'rtb-0a8b148ae12f61dc4' with Route table ID 'rtb-0a8b148ae12f61dc4', Explicit subnet associ... '-', Edge associations '-', Main 'Yes', VPC 'vpc-0dbdc4448fab0188', and Owner ID '016201238719'. The fourth entry is 'rtb-0a44e68a46da3257a' with Route table ID 'rtb-0a44e68a46da3257a', Explicit subnet associ... '-', Edge associations '-', Main 'Yes', VPC 'vpc-084bbf06e50129991 | N...', and Owner ID '016201238719'. The interface includes search filters and an 'Actions' button. The bottom section, 'Routes (2)', shows a table with columns: Destination, Target, Status, and Propagated. The first entry is '0.0.0.0/0' with Target 'igw-035666275d816105a', Status 'Active', and Propagated 'No'. The second entry is '10.0.0.0/16' with Target 'local', Status 'Active', and Propagated 'No'.

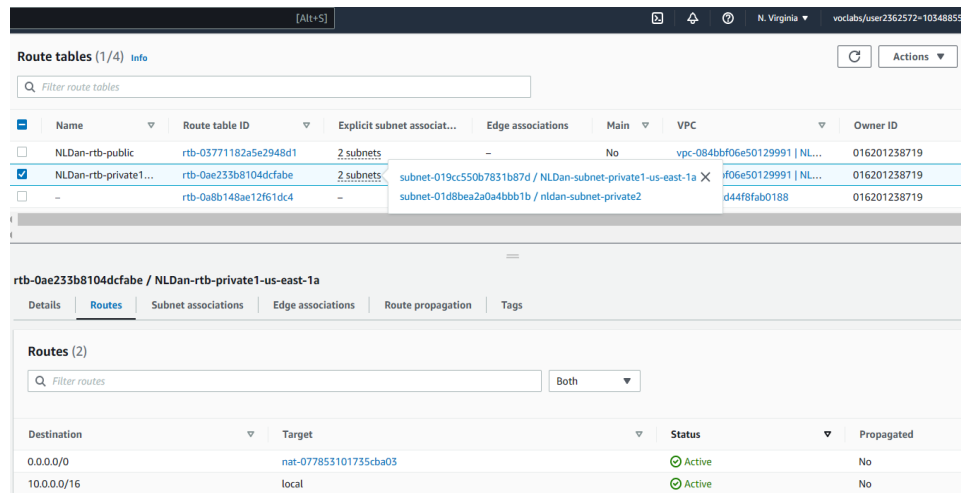
| Name                  | Route table ID        | Explicit subnet associ... | Edge associations  | Main | VPC   | Owner ID     |
|-----------------------|-----------------------|---------------------------|--|------|---|--------------|
| NLDan-rtb-public      | rtb-03771182a5e2948d1 | 2 subnets                 | subnet-0333d875c345f51f7 / NLDan-subnet-public1-us-east-1a | Yes  | vpc-084bbf06e50129991   NLDan-subnet-public1-us-east-1a | 016201238719 |
| NLDan-rtb-private1... | rtb-0ae253b8104dcabe  | 2 subnets                 | subnet-0f3cd3822e73b0727 / nldan-subnet-public2            | Yes  | vpc-0dbdc4448fab0188                                    | 016201238719 |
| -                     | rtb-0a8b148ae12f61dc4 | -                         | -  | Yes  | vpc-0dbdc4448fab0188                                    | 016201238719 |
| -                     | rtb-0a44e68a46da3257a | -                         | -  | Yes  | vpc-084bbf06e50129991   NLDan-subnet-public1-us-east-1a | 016201238719 |

| Destination | Target                | Status | Propagated |
|-------------|-----------------------|--------|------------|
| 0.0.0.0/0   | igw-035666275d816105a | Active | No         |
| 10.0.0.0/16 | local                 | Active | No         |

Figure 2. Route table (Public – igw)

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

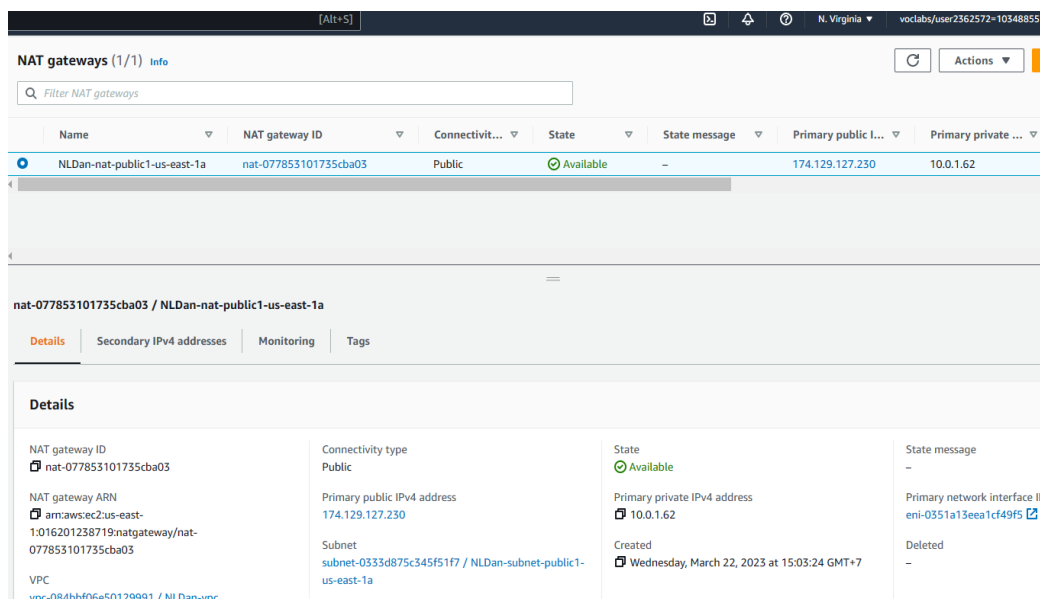


The screenshot shows the AWS Management Console interface for Route tables. The top section lists three route tables: 'NLDan-rtb-public', 'NLDan-rtb-private1...', and 'rtb-0a8b148ae12f61dc4'. The 'NLDan-rtb-private1...' table is selected, and its details are shown below. The 'Routes' tab is active, displaying two routes: '0.0.0.0/0' with target 'nat-077853101735cba03' and '10.0.0.0/16' with target 'local'. Both routes are in an 'Active' state.

| Destination | Target                | Status | Propagated |
|-------------|-----------------------|--------|------------|
| 0.0.0.0/0   | nat-077853101735cba03 | Active | No         |
| 10.0.0.0/16 | local                 | Active | No         |

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).



The screenshot shows the AWS Management Console interface for NAT gateways. The top section lists one NAT gateway: 'NLDan-nat-public1-us-east-1a'. The details for this gateway are shown below. The 'Details' tab is active, displaying information such as the NAT gateway ID, ARN, VPC, connectivity type, primary public and private IP addresses, and the state (Available).

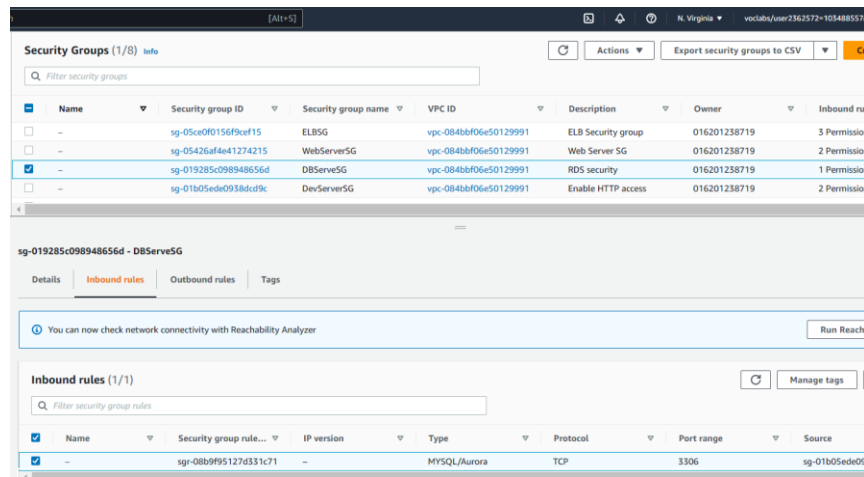
| Details   |
|---|
| <p>NAT gateway ID<br/>nat-077853101735cba03</p> <p>NAT gateway ARN<br/>arn:aws:ec2:us-east-1:016201238719:natgateway/nat-077853101735cba03</p> <p>VPC<br/>vpc-084bbf06e50129991 / NLDan-vpc</p> |

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.



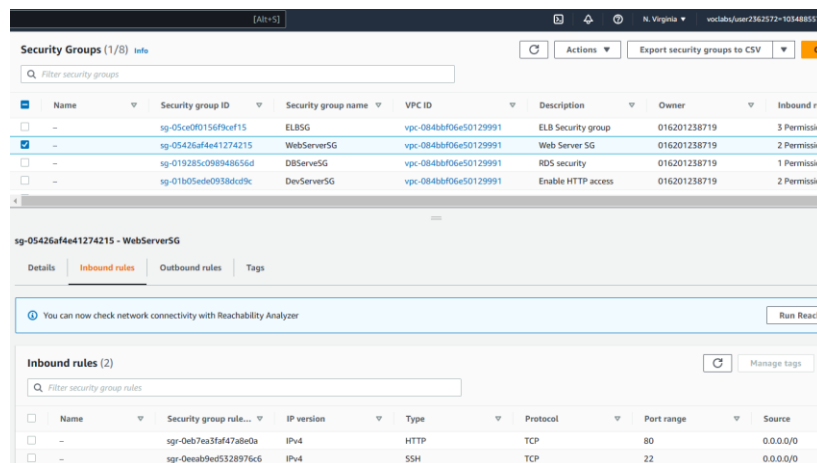
| Name | Security group ID    | Security group name | VPC ID                | Description        | Owner        | Inbound rule |
|------|----------------------|---------------------|-----------------------|--------------------|--------------|--------------|
| -    | sg-05ce0f0156f9cef15 | ELBSG               | vpc-084bbf06e50129991 | ELB Security group | 016201238719 | 3 Permission |
| -    | sg-05426af4e41274215 | WebServerSG         | vpc-084bbf06e50129991 | Web Server SG      | 016201238719 | 2 Permission |
| -    | sg-019285c098948656d | DBServerSG          | vpc-084bbf06e50129991 | RDS security       | 016201238719 | 1 Permission |
| -    | sg-01b05ede0938dcd9c | DevServerSG         | vpc-084bbf06e50129991 | Enable HTTP access | 016201238719 | 2 Permission |

| Name | Security group rule... | IP version | Type         | Protocol | Port range | Source               |
|------|------------------------|------------|--------------|----------|------------|----------------------|
| -    | sg-08b99f5127d331c71   | -          | MySQL/Aurora | TCP      | 3306       | sg-01b05ede0938dcd9c |

Figure 5. RDS security group

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



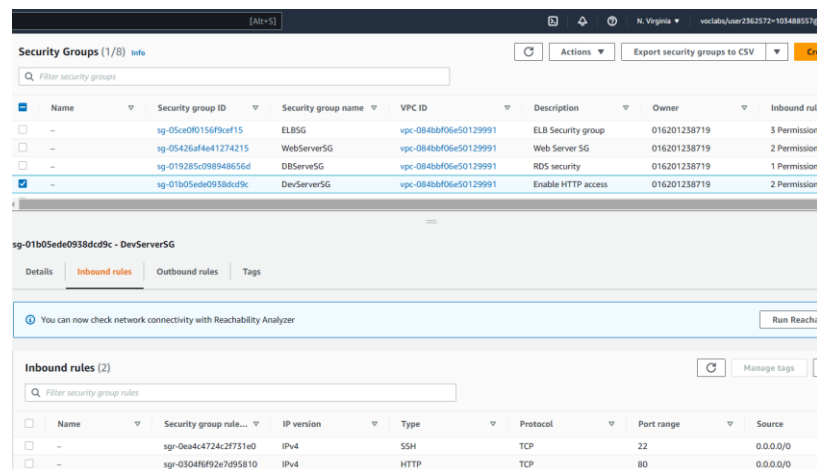
| Name | Security group ID    | Security group name | VPC ID                | Description        | Owner        | Inbound rule |
|------|----------------------|---------------------|-----------------------|--------------------|--------------|--------------|
| -    | sg-05ce0f0156f9cef15 | ELBSG               | vpc-084bbf06e50129991 | ELB Security group | 016201238719 | 3 Permission |
| -    | sg-05426af4e41274215 | WebServerSG         | vpc-084bbf06e50129991 | Web Server SG      | 016201238719 | 2 Permission |
| -    | sg-019285c098948656d | DBServerSG          | vpc-084bbf06e50129991 | RDS security       | 016201238719 | 1 Permission |
| -    | sg-01b05ede0938dcd9c | DevServerSG         | vpc-084bbf06e50129991 | Enable HTTP access | 016201238719 | 2 Permission |

| Name | Security group rule... | IP version | Type | Protocol | Port range | Source    |
|------|------------------------|------------|------|----------|------------|-----------|
| -    | sg-0eb7ea3f4728e0a     | IPv4       | HTTP | TCP      | 80         | 0.0.0.0/0 |
| -    | sg-0eeab9ed05328976c6  | IPv4       | SSH  | TCP      | 22         | 0.0.0.0/0 |

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.



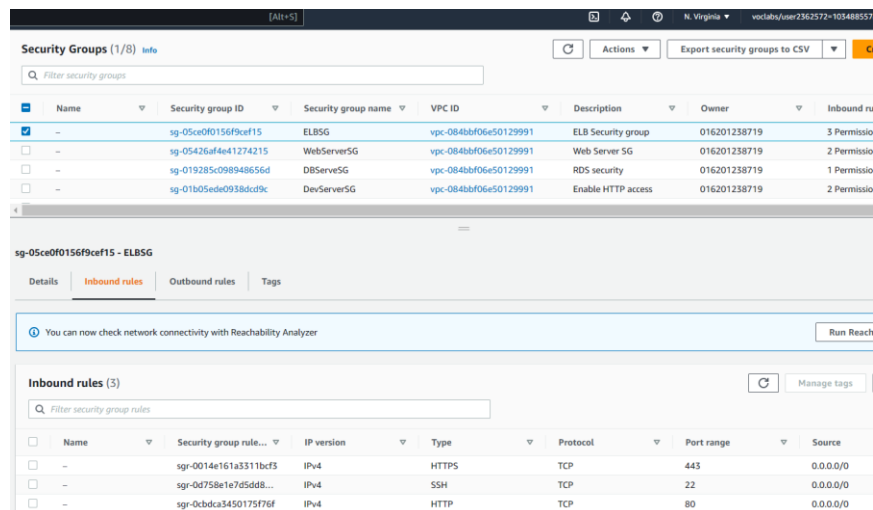
| Name | Security group ID    | Security group name | VPC ID                | Description        | Owner        | Inbound rule |
|------|----------------------|---------------------|-----------------------|--------------------|--------------|--------------|
| -    | sg-05ce0f0156f9cef15 | ELBSG               | vpc-084bbf06e50129991 | ELB Security group | 016201238719 | 3 Permission |
| -    | sg-05426af4e41274215 | WebServerSG         | vpc-084bbf06e50129991 | Web Server SG      | 016201238719 | 2 Permission |
| -    | sg-019285c098948656d | DBServerSG          | vpc-084bbf06e50129991 | RDS security       | 016201238719 | 1 Permission |
| -    | sg-01b05ede0938dcd9c | DevServerSG         | vpc-084bbf06e50129991 | Enable HTTP access | 016201238719 | 2 Permission |

| Name | Security group rule... | IP version | Type | Protocol | Port range | Source    |
|------|------------------------|------------|------|----------|------------|-----------|
| -    | sg-0ea4c4724c2f731e0   | IPv4       | SSH  | TCP      | 22         | 0.0.0.0/0 |
| -    | sg-0304f6f92e7095810   | IPv4       | HTTP | TCP      | 80         | 0.0.0.0/0 |

Figure 7. DevServer security group

## ELBSG – the Load Balancer security group



**Security Groups (1/8)**

| Name        | Security group ID    | Security group name | VPC ID                | Description        | Owner        | Inbound rules |
|-------------|----------------------|---------------------|-----------------------|--------------------|--------------|---------------|
| ELBSG       | sg-05ce0f0156f9cef15 | ELBSG               | vpc-084bbf06e50129991 | ELB Security group | 016201238719 | 3 Permissio   |
| WebServerSG | sg-05426af4e41274215 | WebServerSG         | vpc-084bbf06e50129991 | Web Server SG      | 016201238719 | 2 Permissio   |
| DBServerSG  | sg-019285c098948656d | DBServerSG          | vpc-084bbf06e50129991 | RDS security       | 016201238719 | 1 Permissio   |
| DevServerSG | sg-01b05ede0938dcdfc | DevServerSG         | vpc-084bbf06e50129991 | Enable HTTP access | 016201238719 | 2 Permissio   |

**sg-05ce0f0156f9cef15 - ELBSG**

Details | **Inbound rules** | Outbound rules | Tags

You can now check network connectivity with Reachability Analyzer Run Reach

**Inbound rules (3)**

| Name                 | Security group rule... | IP version | Type  | Protocol | Port range | Source    |
|----------------------|------------------------|------------|-------|----------|------------|-----------|
| sg-0014e161a3311bcf3 |                        | IPv4       | HTTPS | TCP      | 443        | 0.0.0.0/0 |
| sg-0d758e1e7d5dd8... |                        | IPv4       | SSH   | TCP      | 22         | 0.0.0.0/0 |
| sg-0cbdc3450175f76f  |                        | IPv4       | HTTP  | TCP      | 80         | 0.0.0.0/0 |

Figure 8. Load Balancer security group

## C. NACL

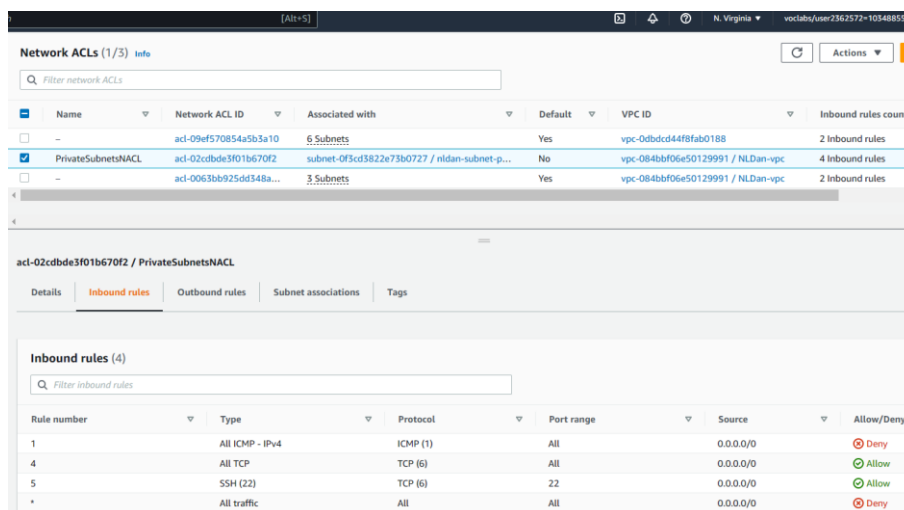
I configured a Network ACL to block **bidirectional traffic from/to the DevServer** instance. When the security rules are applied, you cannot use the DevServer to ping other instances, and other instances also cannot ping the DevServer.

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.



**Network ACLs (1/3)**

| Name               | Network ACL ID       | Associated with                              | Default | VPC ID                            | Inbound rules count |
|--------------------|----------------------|--|---------|-----------------------------------|---------------------|
| PrivateSubnetsNACL | acl-02cddb3f01b670f2 | subnet-0f3cd3822e73b0727 / nidan-subnet-p... | No      | vpc-084bbf06e50129991 / NIDan-vpc | 4 Inbound rules     |

**acl-02cddb3f01b670f2 / PrivateSubnetsNACL**

Details | **Inbound rules** | Outbound rules | Subnet associations | Tags

**Inbound rules (4)**

| Rule number | Type            | Protocol | Port range | Source    | Allow/Deny |
|-------------|-----------------|----------|------------|-----------|------------|
| 1           | All ICMP - IPv4 | ICMP (1) | All        | 0.0.0.0/0 | Deny       |
| 4           | All TCP         | TCP (6)  | All        | 0.0.0.0/0 | Allow      |
| 5           | SSH (22)        | TCP (6)  | 22         | 0.0.0.0/0 | Allow      |
| *           | All traffic     | All      | All        | 0.0.0.0/0 | Deny       |

Figure 9. Inbound rules of the NACL

The screenshot shows the AWS Network ACL console. The top section lists three Network ACLs. The second one, 'PrivateSubnetsNACL', is selected. Below, the 'Outbound rules' tab is active, showing four rules. Rule 1 allows ICMP (ping) traffic. Rule 2 allows all TCP traffic. Rule 3 allows SSH (port 22) traffic. Rule 4 denies all other traffic.

| Rule number | Type            | Protocol | Port range | Destination | Allow/Deny |
|-------------|-----------------|----------|------------|-------------|------------|
| 1           | All ICMP - IPv4 | ICMP (1) | All        | 0.0.0.0/0   | Deny       |
| 2           | All TCP         | TCP (6)  | All        | 0.0.0.0/0   | Allow      |
| 3           | SSH (22)        | TCP (6)  | 22         | 0.0.0.0/0   | Allow      |
| *           | All traffic     | All      | All        | 0.0.0.0/0   | Deny       |

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**).

The screenshot shows the AWS EC2 console with the 'Instances' page. Two instances are listed, both in a 'Running' state. The first instance, 'i-00e53f87e2226a8a1', is selected. The 'Details' tab is active, showing various instance attributes. The 'IAM Role' is highlighted with a red box and is 'LabRole'.

| Name      | Instance ID         | Instance state | Instance type | Status check      | Alarm status | Availability Zone | Public IPv4 DNS   |
|-----------|---------------------|----------------|---------------|-------------------|--------------|-------------------|-------------------|
| DevServer | i-00e53f87e2226a8a1 | Running        | t2.micro      | Initializing      | No alarms    | us-east-1b        | ec2-34-228-97-228 |
| DevServer | i-0cc72f7f5c0ee0d0b | Running        | t2.micro      | 2/2 checks passed | No alarms    | us-east-1b        | -                 |

| Instance: i-00e53f87e2226a8a1 (DevServer)  |  |  |
|--|--|--|
| <b>Instance summary</b><br>Instance ID: i-00e53f87e2226a8a1 (DevServer)<br>IPv6 address: -<br>Hostname type: IP name: ip-10-0-2-21.ec2.internal<br>Answer private resource DNS name: IPv4 (A)<br>Auto-assigned IP address: -<br><b>IAM Role: LabRole</b> | Public IPv4 address: 34.228.97.228   open address<br>Instance state: Running<br>Private IP DNS name (IPv4 only): ip-10-0-2-21.ec2.internal<br>Instance type: t2.micro<br>VPC ID: vpc-084bbf06e50129991 (NLDan-vpc)<br>Subnet ID: subnet-0f3cd3822e73b0727 (nldan-subnet-public2) | Private IPv4 addresses: 10.0.2.21<br>Public IPv4 DNS: ec2-34-228-97-228.compute-1.amazonaws.com<br>Elastic IP addresses: 34.228.97.228 [Public IP]<br>AWS Compute Optimizer finding: Opt-in to AWS Compute Optimizer for recommendations<br>Auto Scaling Group name: - |

| Instance: i-00e53f87e2226a8a1 (DevServer)   |                        |
|---|------------------------|
| <b>Security details</b><br>IAM Role: LabRole<br>Security groups: sg-01b05ede0938dcd9c (DevServerSG) | Owner ID: 016201238719 |

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.

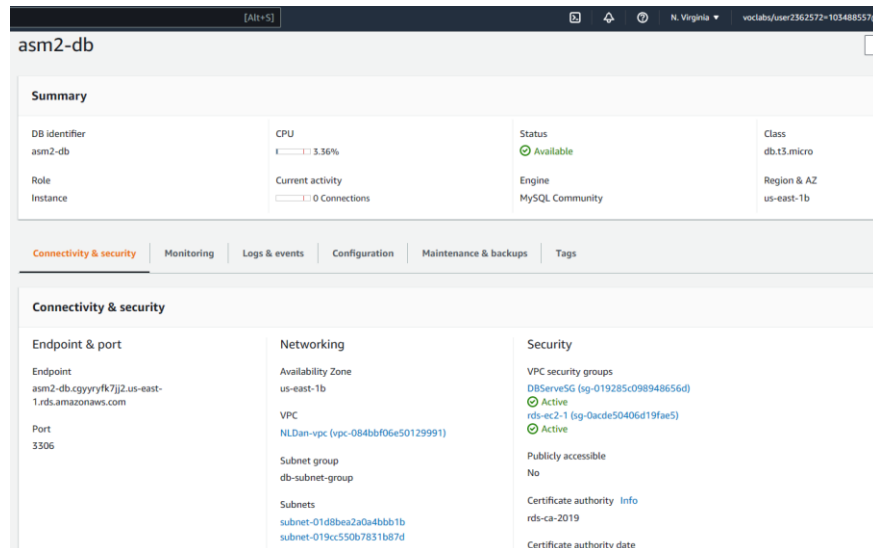


Figure 12. Database settings summary

## 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

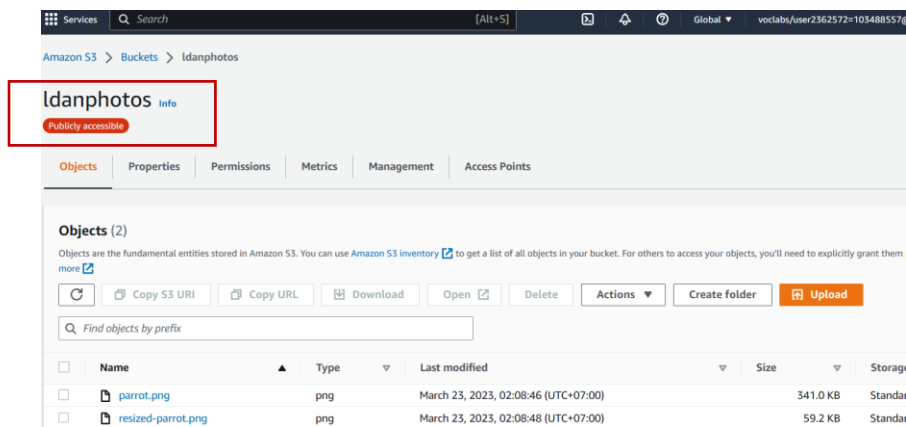


Figure 13: A public S3 Bucket

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

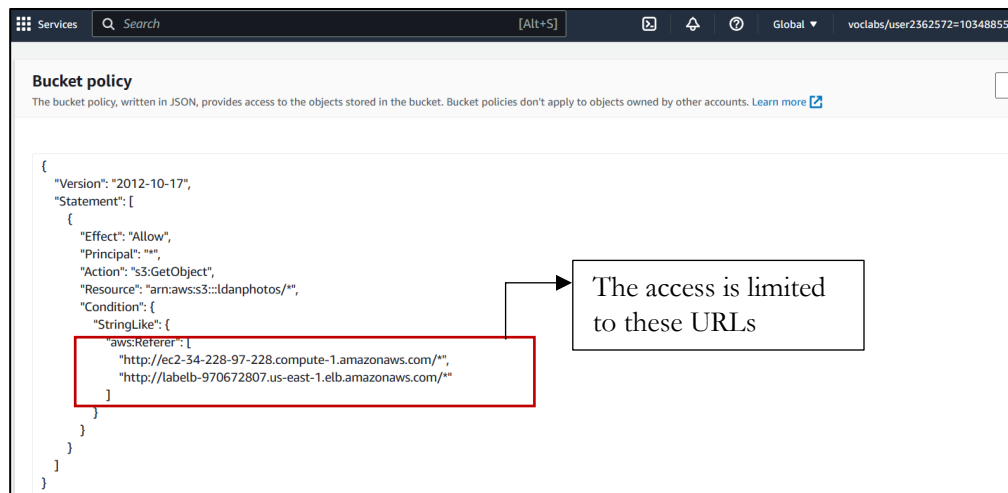


Figure 14: S3 Bucket Policy

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.

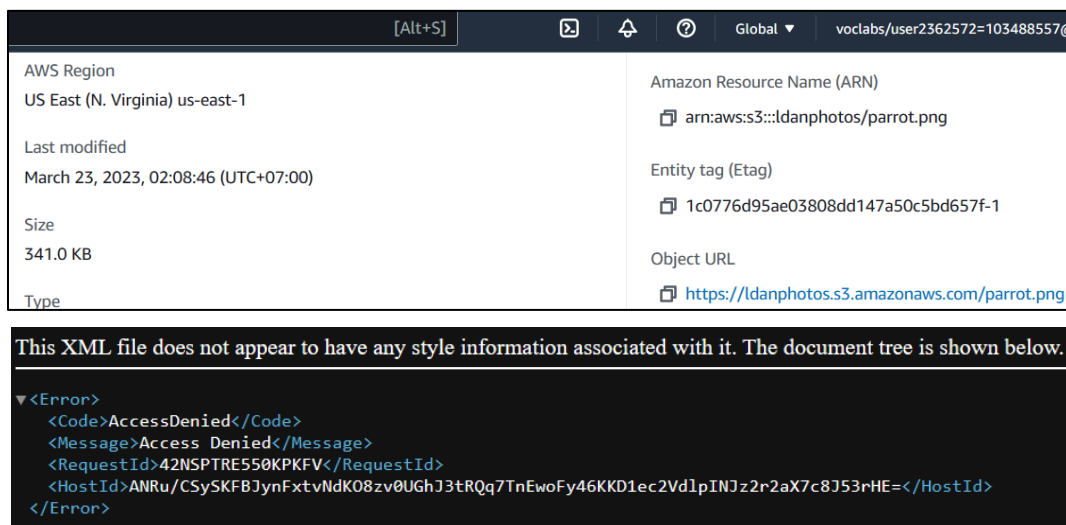


Figure 15: The public access has been restricted by the policy

### 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 ('Parrot', '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.

| <a href="#">+ Options</a> |             |              |          |  |
|---------------------------|-------------|--------------|----------|--|
| 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 |

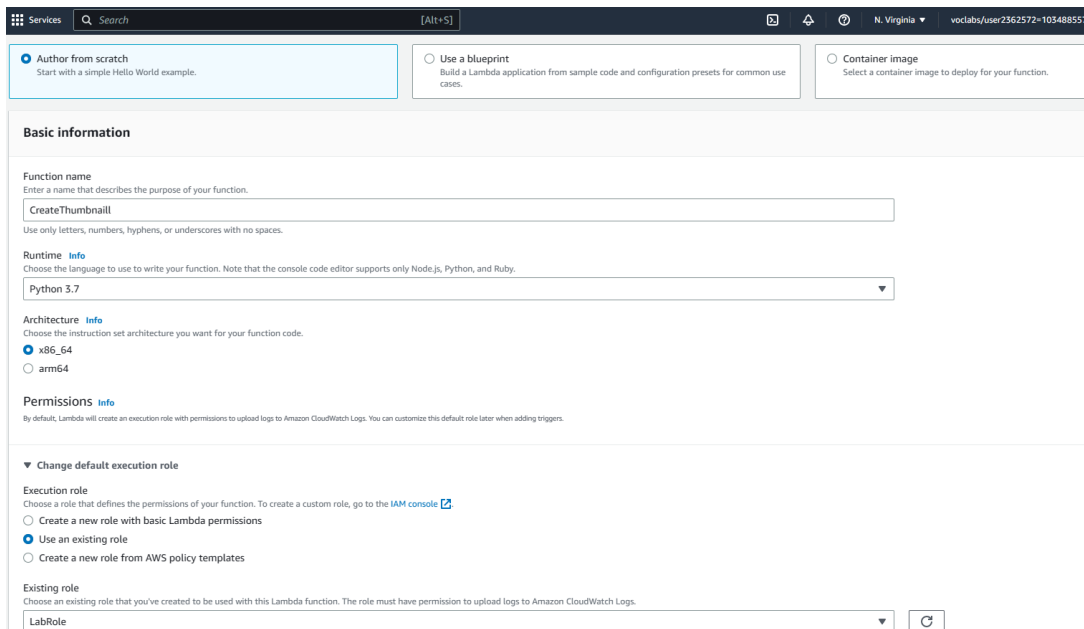
Figure 18: New photo displayed on the screen



## 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



The screenshot shows the AWS Lambda 'Create Function' page. At the top, there are three tabs: 'Author from scratch' (selected), 'Use a blueprint', and 'Container image'. Below these, the 'Basic information' section contains fields for 'Function name' (set to 'CreateThumbnai1l'), 'Runtime' (set to 'Python 3.7'), and 'Architecture' (set to 'x86\_64'). The 'Permissions' section shows 'Use an existing role' selected, with 'LabRole' chosen from the dropdown menu.

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.

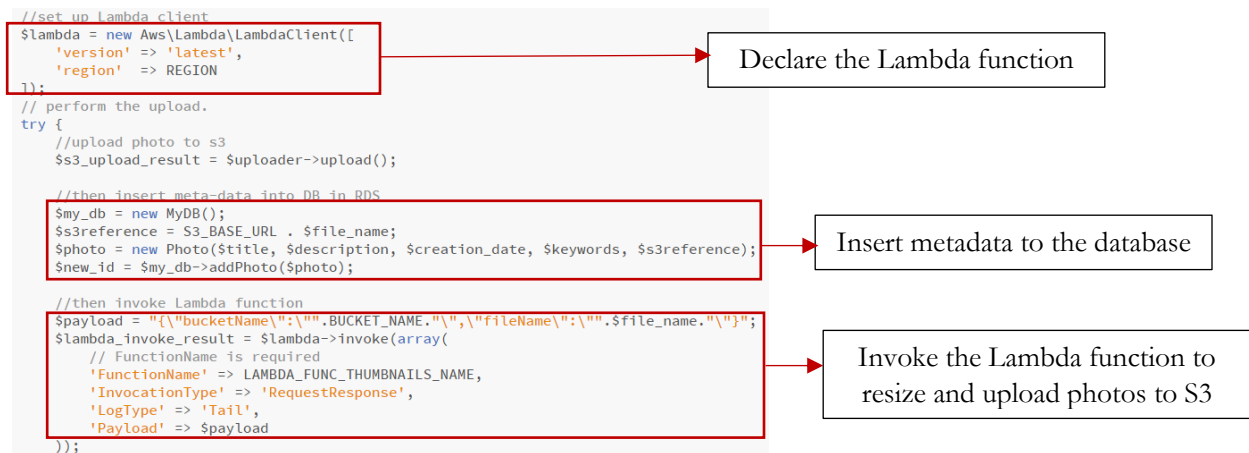


Figure 21: photouploader.php file

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.

## Photo uploader

Photo title:

Select a photo (Select PNG file for best result):  seal.png

Description:

Date:

Keywords (comma-delimited, e.g. keyword1, keyword2, ...):

[Photo Album](#)




Figure 22: Photo Uploader UI

## Result

Find objects by prefix

<

1

| <input type="checkbox"/> | Name   | Type | Last modified                        | Size     | Storage class |
|--------------------------|--|------|--------------------------------------|----------|---------------|
| <input type="checkbox"/> |  <a href="#">parrot.png</a>       | png  | March 24, 2023, 02:13:47 (UTC+07:00) | 341.0 KB | Standard      |
| <input type="checkbox"/> |  <a href="#">resized-seal.png</a> | png  | March 24, 2023, 02:53:32 (UTC+07:00) | 82.5 KB  | Standard      |
| <input type="checkbox"/> |  <a href="#">seal.png</a>         | png  | March 24, 2023, 02:53:30 (UTC+07:00) | 385.2 KB | Standard      |

+ Options

| title  | description | creationdate | keywords | reference   |
|--------|-------------|--------------|----------|---|
| Parrot | A bird      | 2023-02-03   | colorful | <a href="https://ldanphotos.s3.amazonaws.com/parrot.png">https://ldanphotos.s3.amazonaws.com/parrot.png</a> |
| Seal   | Sea animal  | 2023-03-22   | grey     | <a href="https://ldanphotos.s3.amazonaws.com/seal.png">https://ldanphotos.s3.amazonaws.com/seal.png</a>     |

A resized photo is created

Metadata is updated to the database

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.

[Alt+S]
Global
voclabs/user2362572-103488557

### LabRole

**Summary**

|   |  |   |
|---|--|---|
| <b>Creation date</b><br>March 22, 2023, 14:56 (UTC+07:00)                 | <b>ARN</b><br>arn:aws:iam::016201238719:role/LabRole | <b>Instance profile ARN</b><br>arn:aws:iam::016201238719:instance-profile/Lab |
| <b>Last activity</b><br><span style="color: green;">18 minutes ago</span> | <b>Maximum session duration</b><br>1 hour            |   |

Permissions
Trust relationships
Tags (1)
Access Advisor
Revoke sessions

**Trusted entities**

Entities that can assume this role under specified conditions.

```

5  Effect: ALLOW,
6  "Principal": {
7    "Service": [
8      "kendra.amazonaws.com",
9      "cloudfront.amazonaws.com",
10     "secretsmanager.amazonaws.com",
11     "transcribe.amazonaws.com",
12     "cloud9.amazonaws.com",
13     "personalize.amazonaws.com",
14     "cloudtrail.amazonaws.com",
15     "ec2.amazonaws.com",
16     "glue.amazonaws.com",
17     "ssm.amazonaws.com",
18     "credentials-iam.amazonaws.com",
19     "lex.amazonaws.com",
20     "cloudformation.amazonaws.com",
21     "sns.amazonaws.com",
22     "iot.amazonaws.com",
23     "kms.amazonaws.com",
24     "application-autoscaling.amazonaws.com",
25     "elasticloadbalancing.amazonaws.com",
26     "athena.amazonaws.com",
27     "lambda.amazonaws.com",
28     "iotanalytics.amazonaws.com",

```

The **LabRole** allows access to necessary services in this assignment such as Lambda, EC2, RDS, and S3

Figure 25: LabRole permissions

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

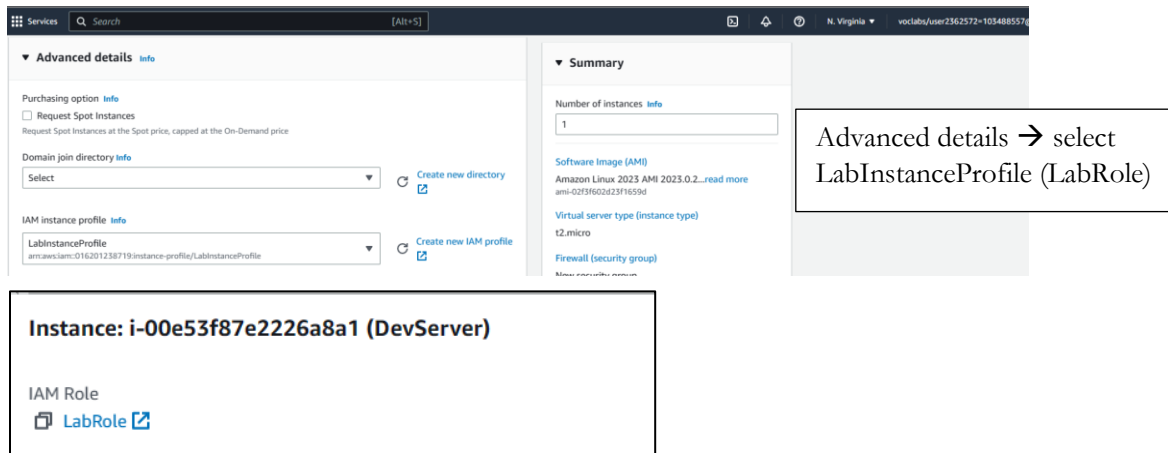


Figure 26: Attach the IAM role to EC2 instance

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

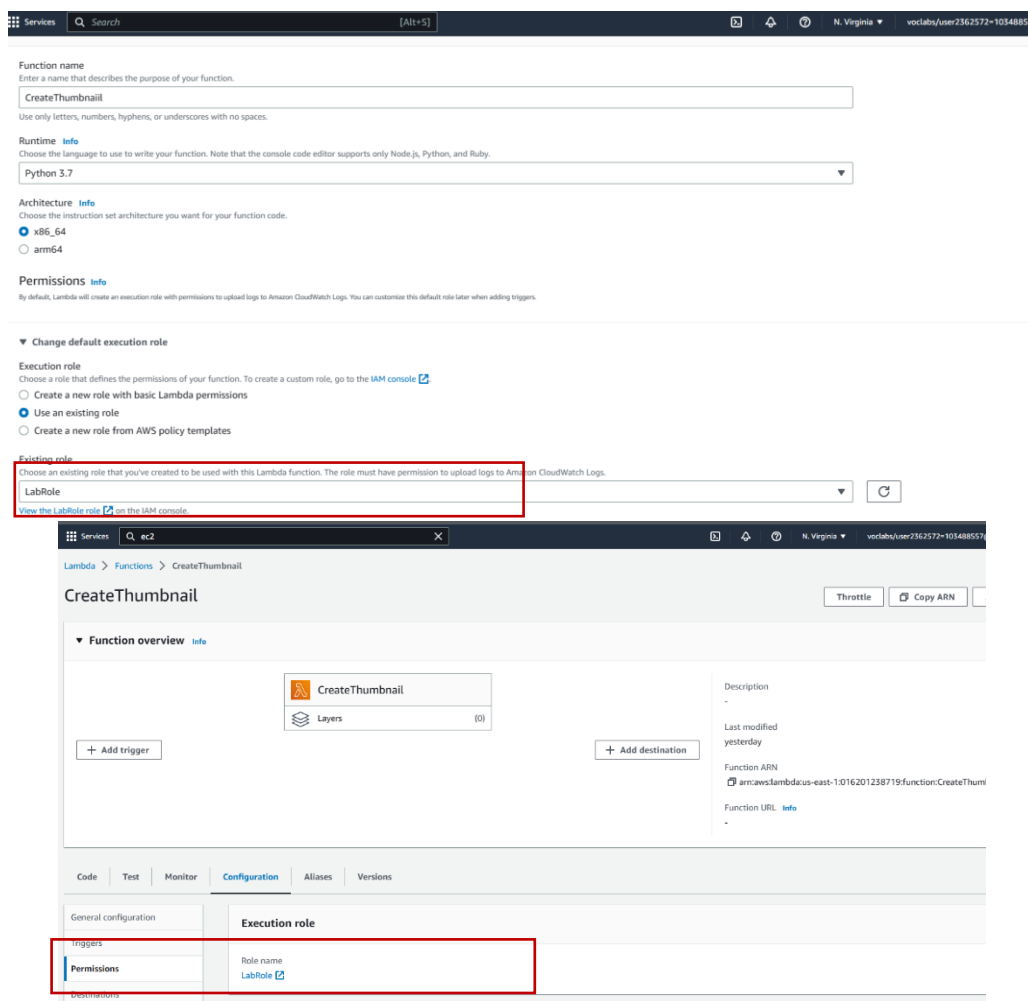
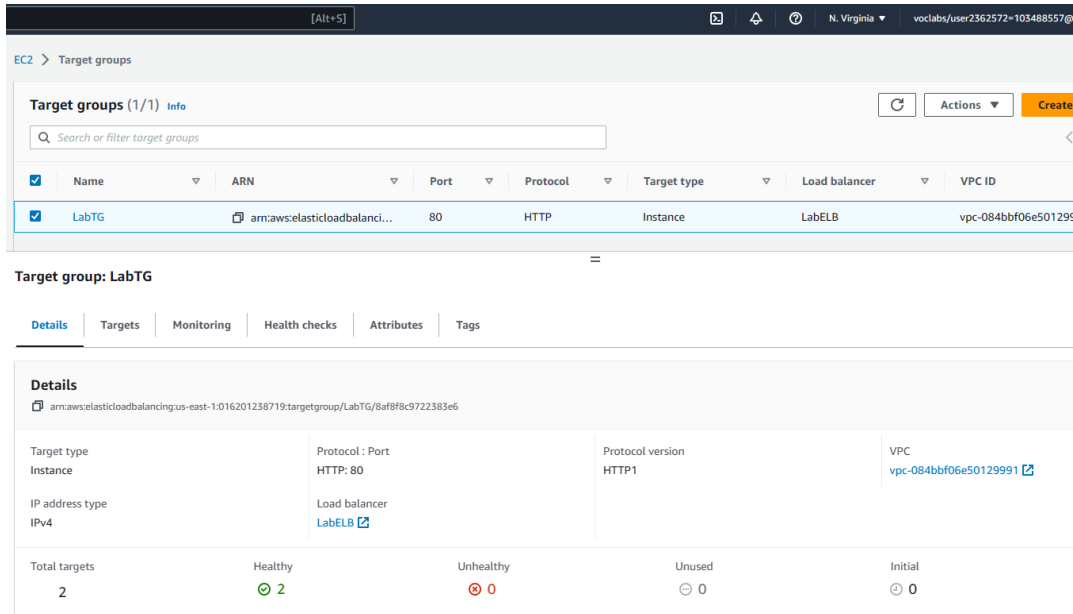


Figure 27: Attach the IAM role to Lambda

## 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.



The screenshot shows the AWS Management Console interface for 'Target groups'. A table lists the target groups, with 'LabTG' selected. Below the table, the 'Details' tab for 'LabTG' is active, showing various configuration parameters.

| Name  | ARN                             | Port | Protocol | Target type | Load balancer | VPC ID               |
|-------|---------------------------------|------|----------|-------------|---------------|----------------------|
| LabTG | arn:aws:elasticloadbalancing... | 80   | HTTP     | Instance    | LabELB        | vpc-084bbf06e5012991 |

**Target group: LabTG**

**Details**

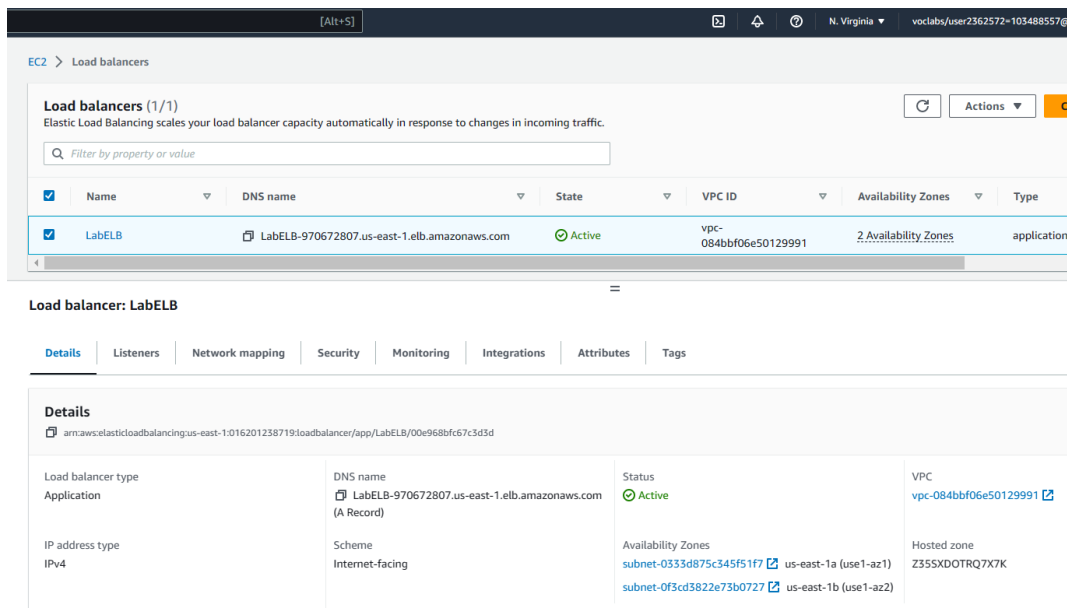
arn:aws:elasticloadbalancing:us-east-1:016201238719:targetgroup/LabTG/8af8f8c9722383e6

|                         |                             |                           |                             |
|-------------------------|-----------------------------|---------------------------|-----------------------------|
| Target type<br>Instance | Protocol : Port<br>HTTP: 80 | Protocol version<br>HTTP1 | VPC<br>vpc-084bbf06e5012991 |
| IP address type<br>IPv4 | Load balancer<br>LabELB     |                           |                             |

Total targets: 2, Healthy: 2, Unhealthy: 0, Unused: 0, Initial: 0

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.



The screenshot shows the AWS Management Console interface for 'Load balancers'. A table lists the load balancers, with 'LabELB' selected. Below the table, the 'Details' tab for 'LabELB' is active, showing various configuration parameters.

| Name   | DNS name                                     | State  | VPC ID               | Availability Zones   | Type        |
|--------|--|--------|----------------------|----------------------|-------------|
| LabELB | LabELB-970672807.us-east-1.elb.amazonaws.com | Active | vpc-084bbf06e5012991 | 2 Availability Zones | application |

**Load balancer: LabELB**

**Details**

arn:aws:elasticloadbalancing:us-east-1:016201238719:loadbalancer/app/LabELB/00e968bfc67c3d3d

|                                   |   |  |                               |
|-----------------------------------|---|--|-------------------------------|
| Load balancer type<br>Application | DNS name<br>LabELB-970672807.us-east-1.elb.amazonaws.com (A Record) | Status<br>Active   | VPC<br>vpc-084bbf06e5012991   |
| IP address type<br>IPv4           | Scheme<br>Internet-facing   | Availability Zones<br>subnet-0333d875c345f51f7 us-east-1a (use1-az1)<br>subnet-0f3cd3822e73b0727 us-east-1b (use1-az2) | Hosted zone<br>Z35SXDOTRQ7X7K |

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**

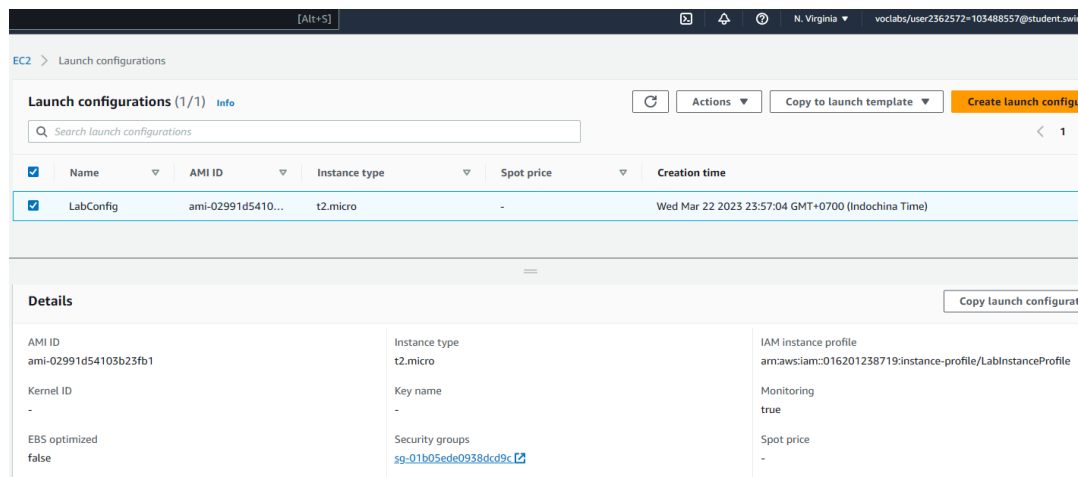


Figure 30: Launch Configurations

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

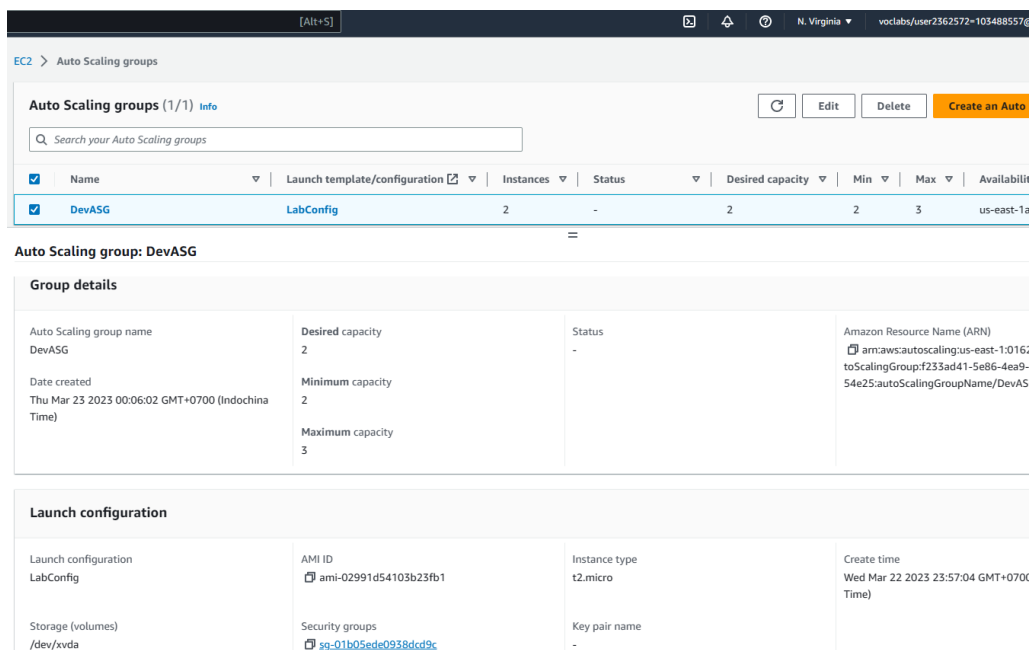


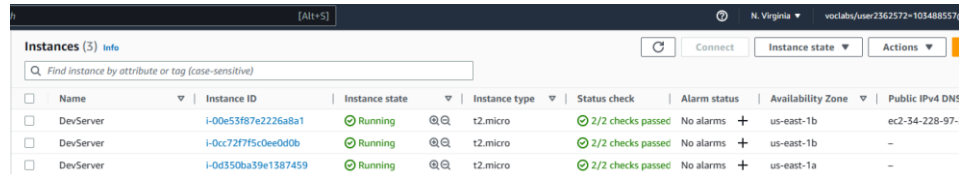
Figure 31: Auto Scaling Group

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**.



|                          | Name      | Instance ID         | Instance state | Instance type | Status check      | Alarm status | Availability Zone | Public IPv4 DNS |
|--------------------------|-----------|---------------------|----------------|---------------|-------------------|--------------|-------------------|-----------------|
| <input type="checkbox"/> | DevServer | i-00e53f87e2226a8a1 | Running        | t2.micro      | 2/2 checks passed | No alarms    | us-east-1b        | ec2-34-228-97-  |
| <input type="checkbox"/> | DevServer | i-0cc72f7f5c0ee0d0b | Running        | t2.micro      | 2/2 checks passed | No alarms    | us-east-1b        | -               |
| <input type="checkbox"/> | DevServer | i-0d350ba39e1387459 | Running        | t2.micro      | 2/2 checks passed | No alarms    | us-east-1a        | -               |

Figure 32: Auto Scaling Group added 2 new instances

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

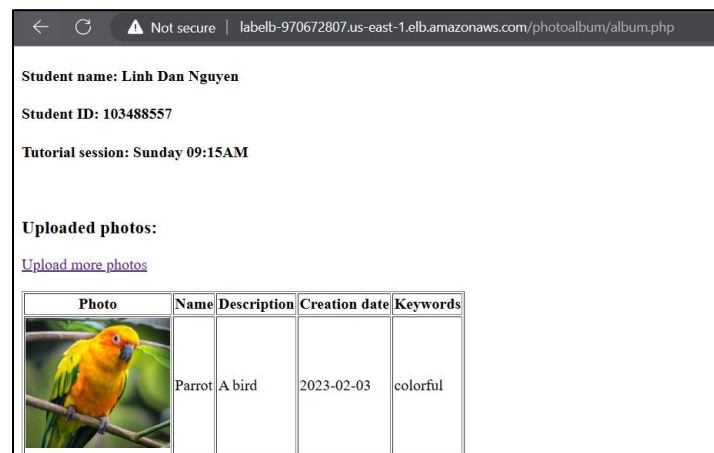


Figure 33: Access the web via Load Balancer DNS

## 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>