

COS20019 – Cloud Computing Architecture

Assignment 2

Janaka Muthunayake

ID: 104315180

Tutorial: Wednesday 06:30pm

Submission: 08-10-2023

104315180@student.swin.edu.au

Abstract - In Assignment 2, I took the application and architecture developed in Assignment 1b to the next level. Several key components were added to enhance the system's functionality and reliability. These included the implementation of a Lambda function, the creation of a custom Amazon Machine Image (AMI), the establishment of a launch configuration based on our customized AMI, and the setup of an auto-scaling group spanning multiple Availability Zones. Additionally, I implemented policies for both scaling up and down based on system demands. To ensure efficient distribution of service requests, an Elastic Load Balancer was integrated into the architecture. I also incorporated robust access control mechanisms and implemented traffic throttling for optimal performance and resource utilization.

bucket while simultaneously recording essential metadata associated with the photo into the RDS Database.

II. CREATION

1. VPC

I have designed and implemented a Virtual Private Cloud (VPC) within the Amazon Web Services (AWS) environment. The VPC serves as the foundational network infrastructure to host our web application and its associated resources. The initial step in this process was the creation of the VPC, which was assigned the IPv4 CIDR (Classless Inter-Domain

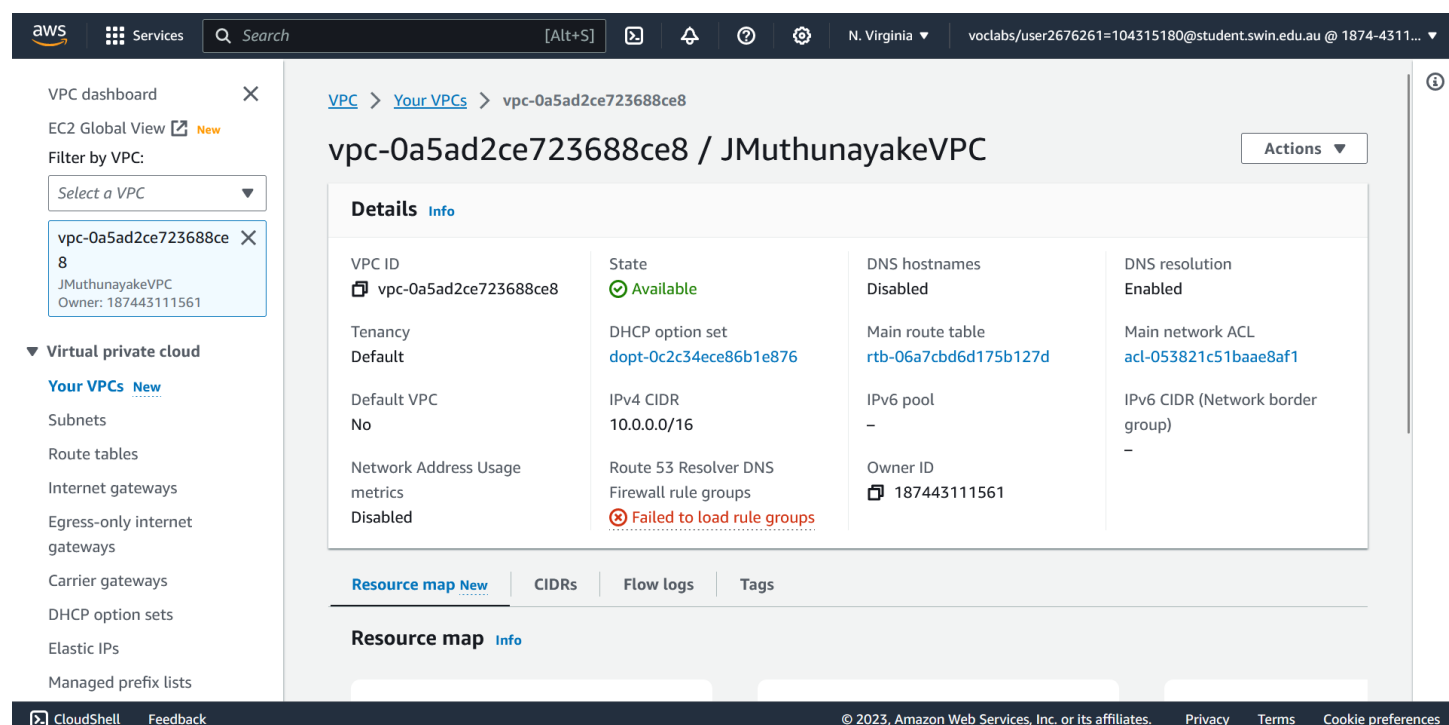


FIGURE 1 – CREATING VPC

Routing) block of 10.0.0.0/16, as illustrated in Figure 1.

I. INTRODUCTION

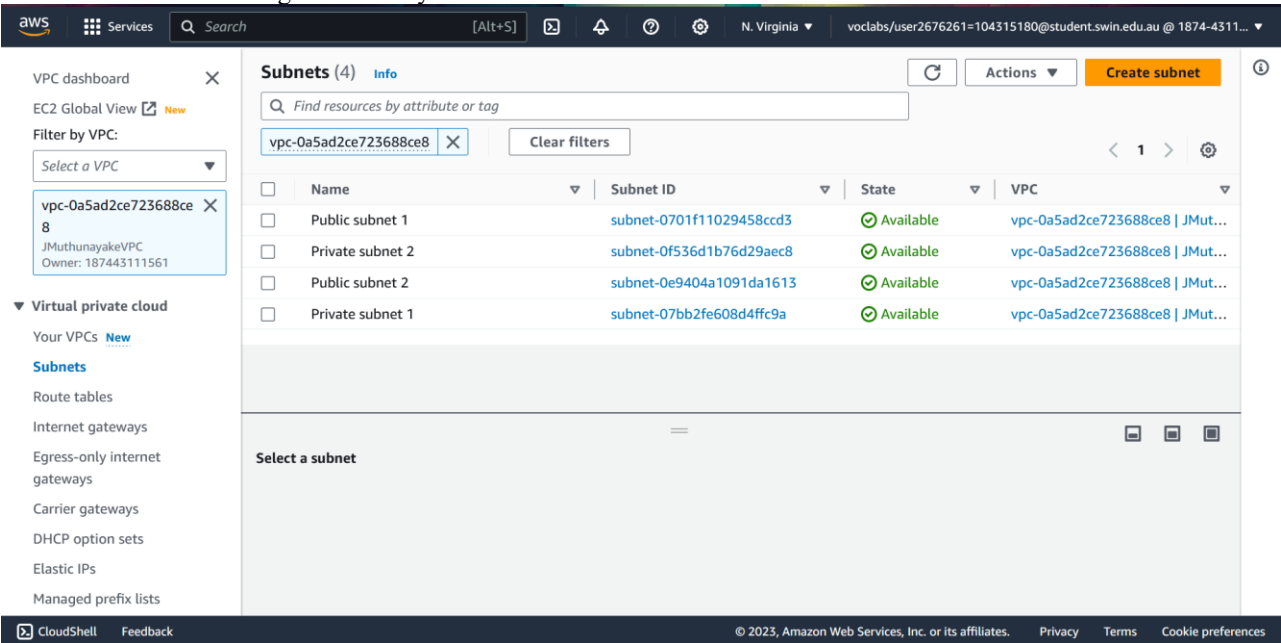
In this assignment, I developed a web application for managing photo albums. This application allows users to both upload their photos and view them seamlessly. When a user uploads a photo, the application stores it in an Amazon S3

2. Creation of Subnets (Figure 2)

I divided our VPC into four subnets, aligning them with the infrastructure diagram's specifications. Two

of these subnets were designated as public subnets, named Public Subnet 1 and Public Subnet 2, and they were placed in different availability zones (us-east-1a and us-east-1b, respectively). The other two subnets, named Private Subnet 1 and Private Subnet 2, were also placed in different availability zones. This distribution ensures high availability and fault

route tables as Figure 4 for the public subnets, which associate these subnets with the IGW. This allows traffic to flow in and out of the public subnets seamlessly. And private subnet route table as Figure 5.



tolerance for our resources.

FIGURE 2 – SUBNETS

3. Making Subnets Public

To enable public access for resources within the designated public subnets, I created an Internet Gateway (IGW) and associated it with our VPC as

4. Instance Deployment

I deployed four instances, each with its specific role within our infrastructure. These instances include a NAT Instance (NAT server) in Public Subnet 1, a Dev Server in Public Subnet 2, and two Bastion instances (Web Servers) in the private subnets. These instances are strategically placed to serve their

intended functions within our network.

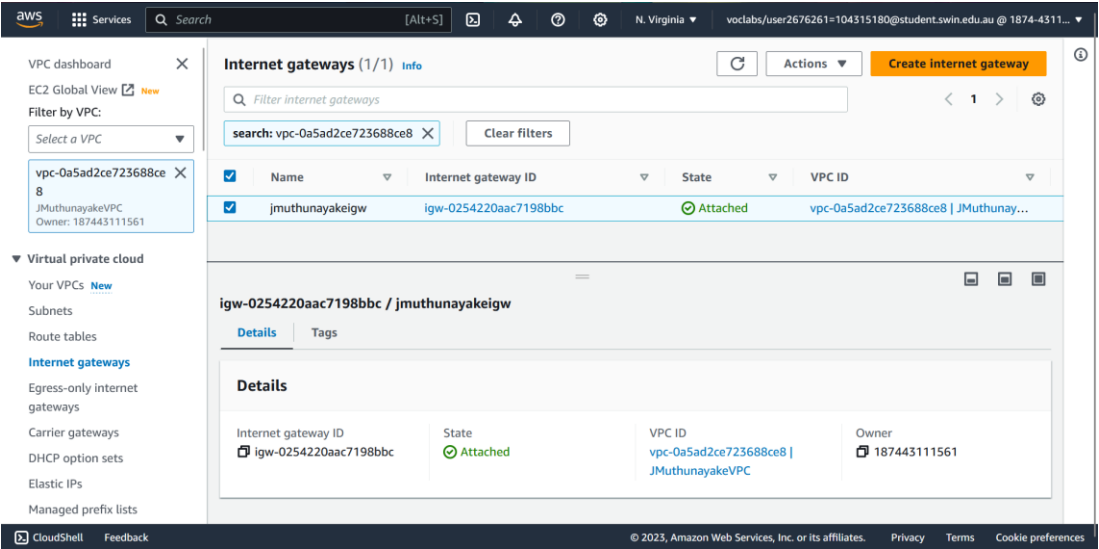


FIGURE 3 – INTERNET GATEWAY

facilitates communication between instances within our VPC and the public internet. I also configured

FIGURE 6 – PRIVATE ROUTE TABLE

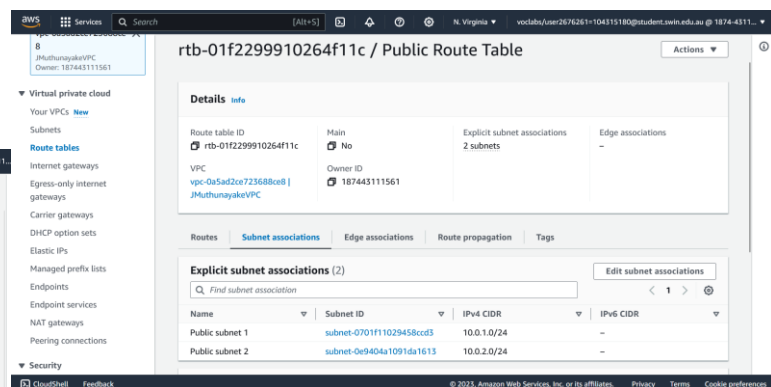
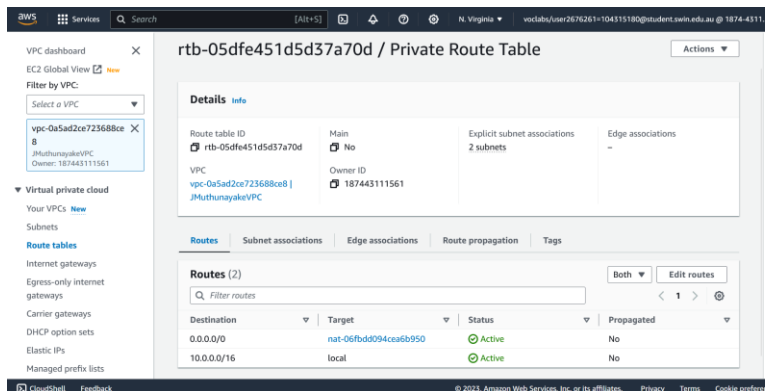


FIGURE 5 – PUBLIC ROUTE TABLE

5. Network ACL and Security Groups

To control traffic flow and enhance security, I created a Network ACL named "PrivateSubnetsNACL" as figure 7 to block bidirectional ICMP traffic to/from the Dev Server. Additionally, I established five security groups as figure 8, each tailored for specific tiers within our architecture: DevServerSG, ELBSG, WebServerSG, DBServerSG, and NATServerSG. These security groups enforce security policies, controlling inbound and outbound traffic for their respective instances.

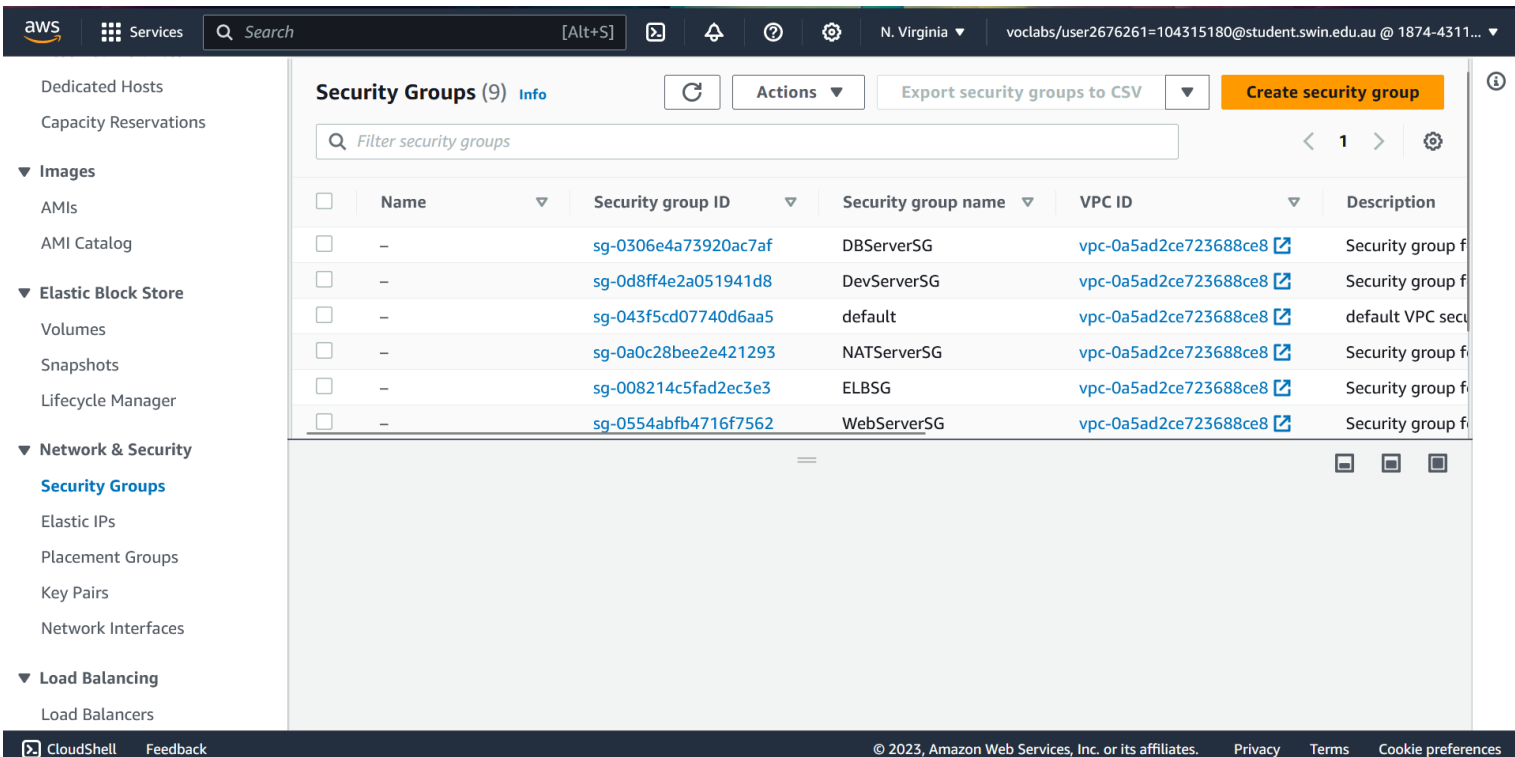
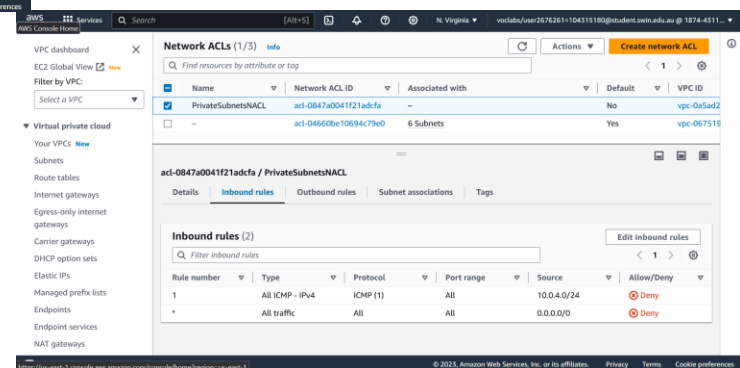


FIGURE 8 – SECURITY GROUPS

FIGURE 7 – NETWORK ACL

it resides within the private subnets.

FIGURE 10 – DATABASE CREATION

6. Source/Destination Check Disablement (Figure 9)

To enable NAT functionality for the NAT Instance (NAT server), I disabled the source/destination check. This step is crucial to allow private instances to communicate with the public internet through the NAT instance.

8. Application and phpmyadmin Configuration (figure 10)

To connect the application running on our Dev Server to the RDS database, I accessed the Dev Server via SSH using PuTTY. In this environment, I renamed the configuration file from "config.sample.inc.php" to "config.inc.php" and updated the "localhost" reference with the endpoint of the RDS database. This step ensured that the application interacts with the database securely. I configured phpMyAdmin to manage

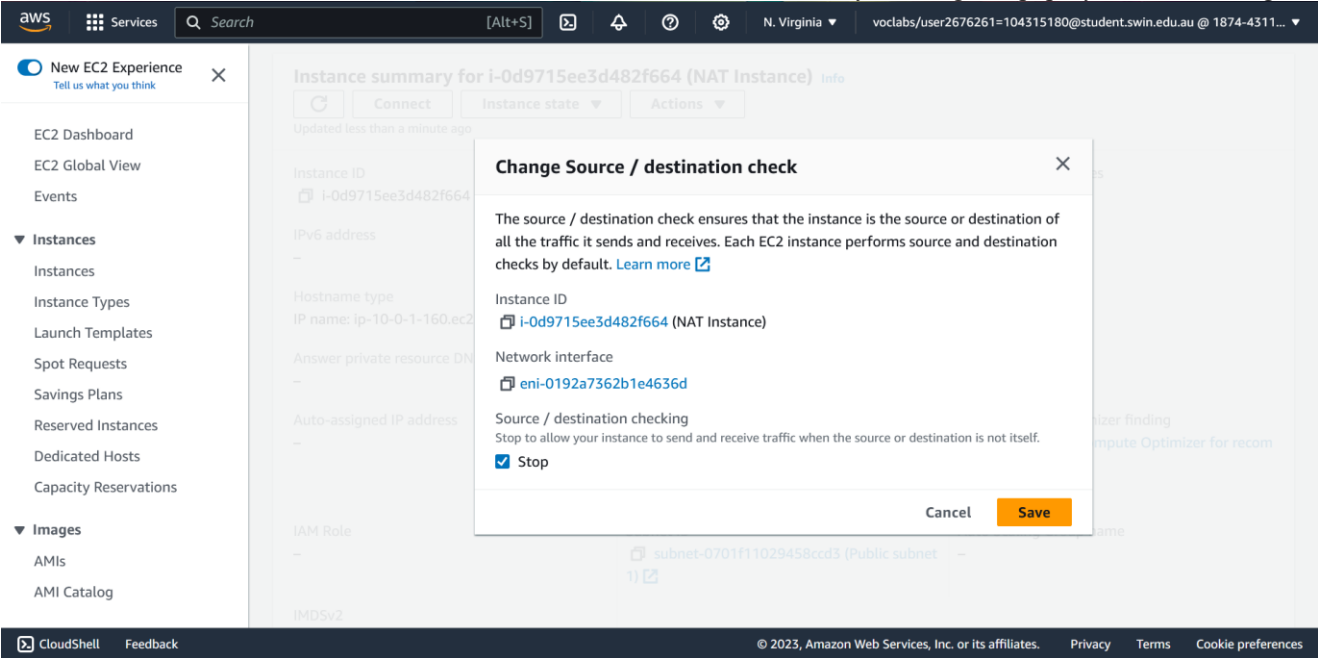


FIGURE 9 – SOURCE/DESTINATION CHECK DISABLEMENT

7. Database Creation and Configuration (figure 10)

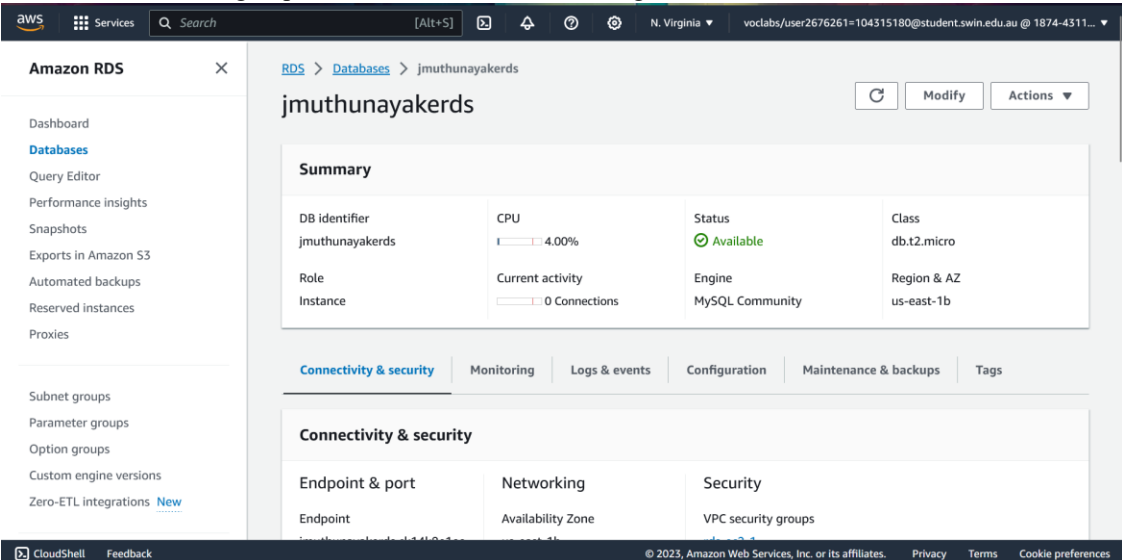
I began by creating an RDS database instance named "jmuthunayakerds." The database instance was configured with an admin as username and a secure password ("mypassword"). To enhance security, I created subnet groups for the RDS database, ensuring

the RDS database. Accessing phpMyAdmin via "<http://44.219.166.229/phpmyadmin>," I created the necessary tables to store photo data. This step prepared the database for storing metadata about the photos, which are stored in an S3 bucket.

9. Lambda Function Creation (figure 11)

I created a Lambda function named "CreateThumbnail" and associated it with the IAM

role "LabRole" to grant the necessary permissions for accessing the S3 bucket and performing image processing tasks. The deployment package "lambda_deploymen t_package.zip" was uploaded to this



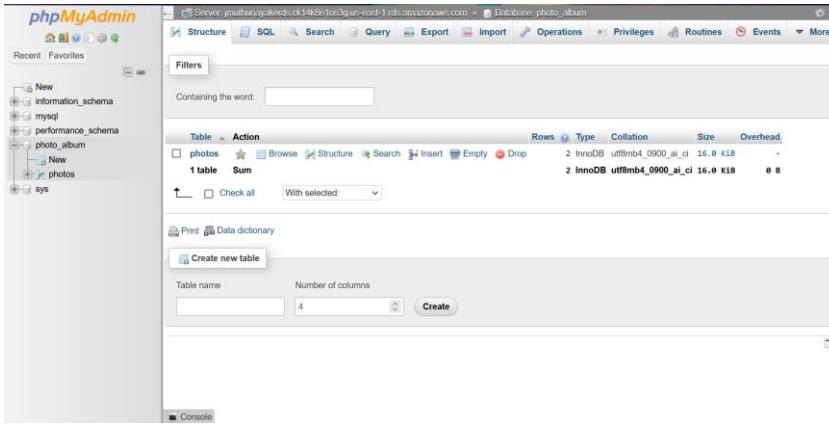


FIGURE 10 – MYPHPADMIN CONFIGURATION

10. S3 Bucket Configuration (figure 12)

I created an S3 bucket to store photos and configured an event destination that triggers the "CreateThumbnail" Lambda function when new photos are uploaded to the bucket. A bucket policy was established to allow the Lambda function to access the bucket and process photos.

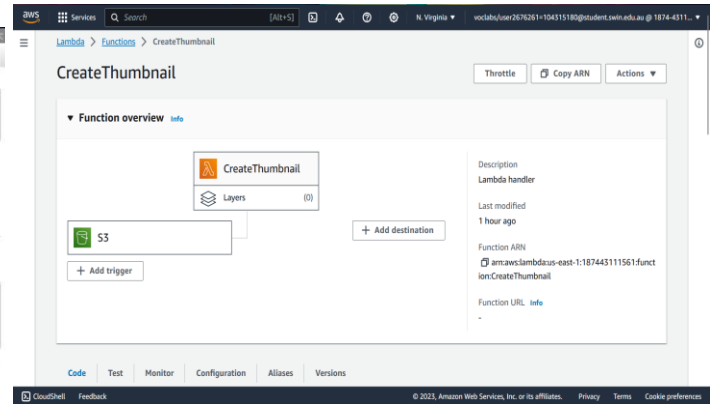


FIGURE 12 – LAMBDA FUNCTION

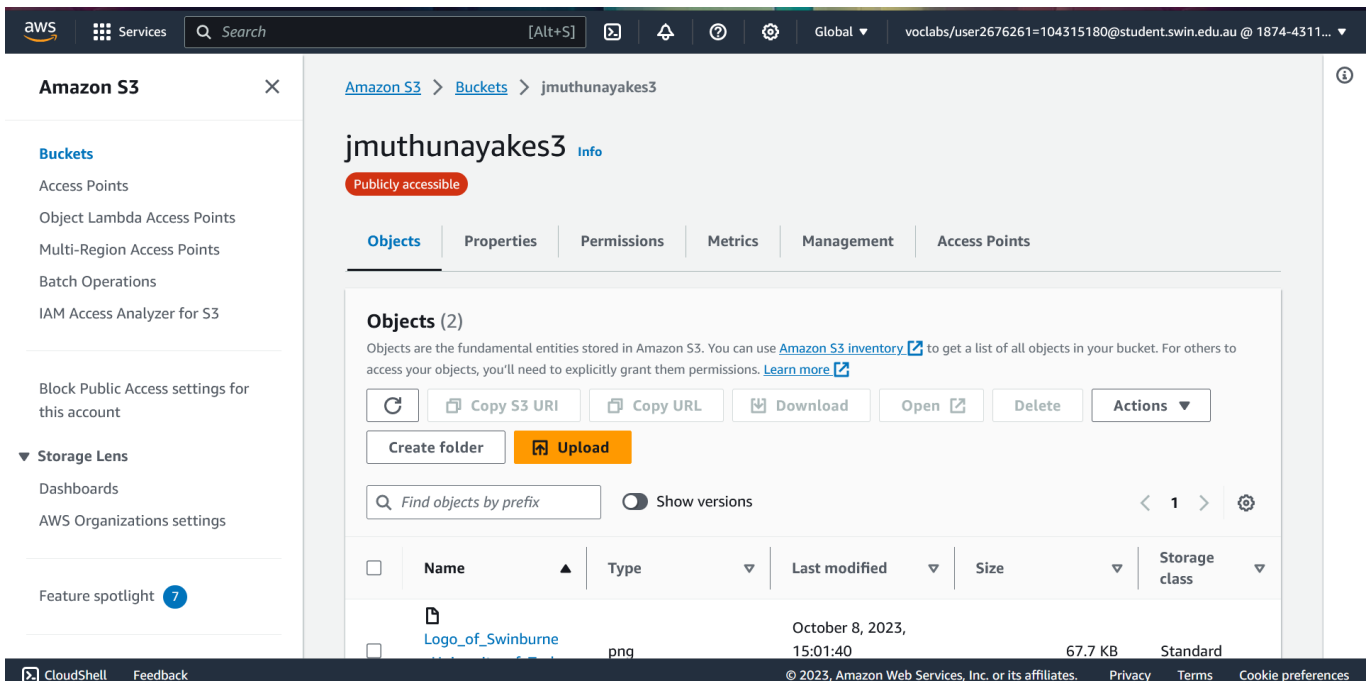
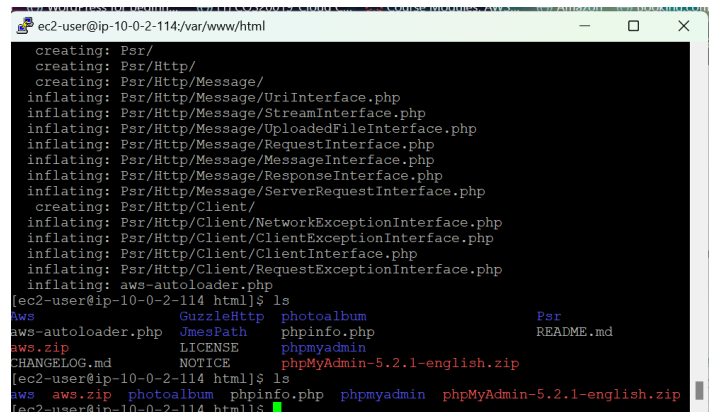


FIGURE 11 – MYPHPADMIN CONFIGURATION

11. Application Deployment (figure 12)

To make our website accessible to users, I edited the "constants.php" file and installed the AWS SDK, following the instructions provided within the "constants.php" file. The "photoalbum" website files were then uploaded to the Dev Server using WinSCP.

FIGURE 12 – INSTALLING AWS SDK

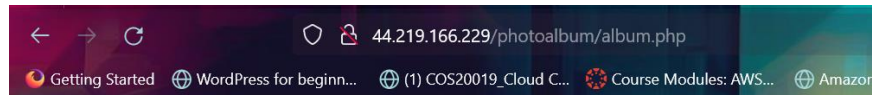


12. Website Access (figure 13)

I assigned an Elastic IP address to the Dev Server, and users can now access the website by navigating to "http://44.219.166.229/photoalbum/album.php." This configuration ensures that our website functions correctly and that it can interact with the RDS database and S3 bucket as needed.

14. Configuring the Load Balancer

To enable load balancing, I commenced by creating a Target Group. I navigated to the "Target Groups" page and selected "Create Target Group." The target type chosen was "Instances," and I named it "assignment2tg." The VPC was selected, and the target group was created successfully, as depicted in Fig. 15.



Student name: Janaka Muthunayake

Student ID: 104315180

Tutorial session: Wednesday 06:30 PM

Uploaded photos:

[Upload more photos](#)



Photo	Name	Description	Creation date	Keywords
	Swinburne Logo	Logo of Swinburne uni	2023-08-10	logo, university
	SLIIT Logo	Logo of SLIIT University	2023-08-10	university, logo

FIGURE 13 – IMPLEMENTED WEBSITE

13. Creating an Amazon Machine Image (AMI)

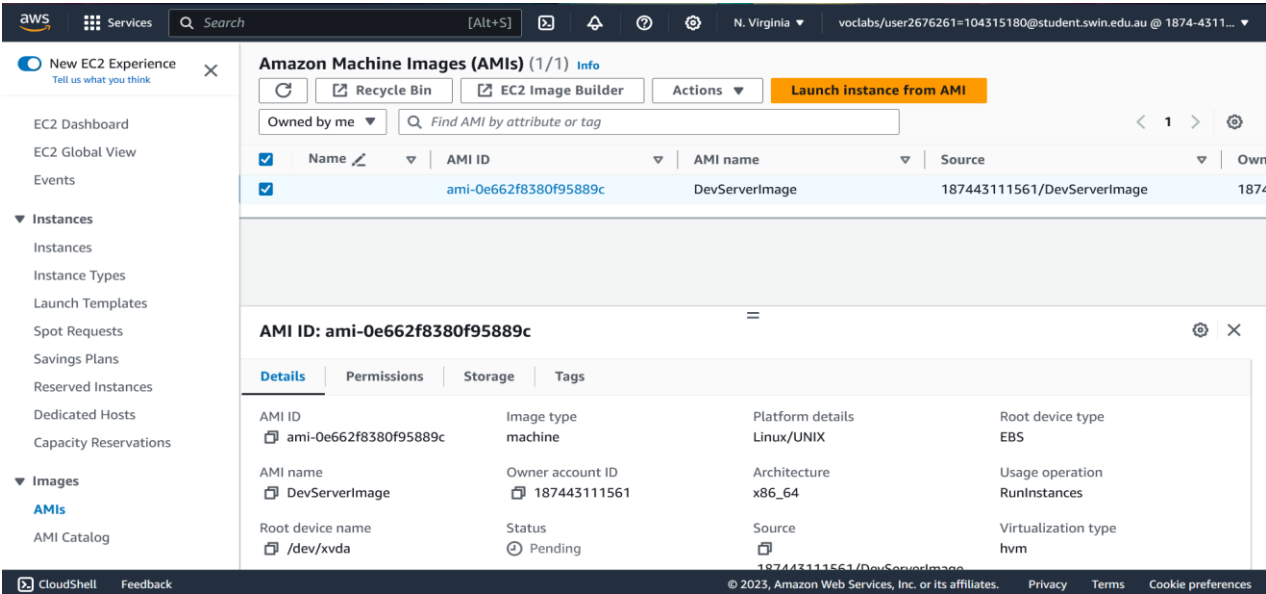
I initiated the process by creating an Amazon Machine Image (AMI) based on the Dev Server instance. This was accomplished by selecting the Dev Server instance, accessing the "Actions" menu, and choosing "Image and templates." After providing a suitable name, "DevServerimage," I selected "Create Image," resulting in the successful creation of the AMI as shown in Fig. 14.

Subsequently, I proceeded to set up the Application Load Balancer. On the Load Balancers page, I clicked "Create Load Balancer" and selected "Application Load Balancer." The Load Balancer was named "jmuthunayakealb." Within the Network mapping section, I assigned our VPC and associated the Load Balancer with public subnet 1 and public subnet 2. I ensured the Load Balancer used "ELBSG" as the Security Group and removed the default security group. Additionally, I configured the listener to route incoming traffic on HTTP (port 80) to the "assignment2tg" target group. The Load Balancer was successfully created, as seen in Fig. 16.

15. Creating a Launch Template and Auto Scaling Group

I proceeded to create a Launch Template to standardize the configuration of our instances. I accessed the Launch template page and clicked "Create launch configuration." I named it "jmuthunayakelt" and selected the "DevServerImage" as the AMI. I specified the instance type as "t2.micro" and assigned the "LabInstanceProfile" IAM instance profile. Monitoring was enabled, and user data was configured as required. The "WebServerSG" security group was applied, and the key pair was chosen. Upon completing these steps, the Launch Configuration was created successfully.

FIGURE 14 – AMIs



Subsequently, I created an Auto Scaling group utilizing the "jmuthunayakelt" Launch Configuration. I named it "jmuthunayakeasg" and selected the desired VPC, "private subnet 1," and "private subnet 2" for subnets. In the Load balancing section, I attached it to the "jmuthunayakealb" Load Balancer. I enabled group metrics collection within CloudWatch and defined the desired, minimum, and maximum capacity as 2, 2, and 3, respectively. A target tracking scaling policy named "Target Tracking policy" was configured, which monitored the "Application Load Balancer request count per target" and aimed to maintain a target value of "30." These configurations led to the creation of the Auto Scaling group, as seen in Fig. 17.

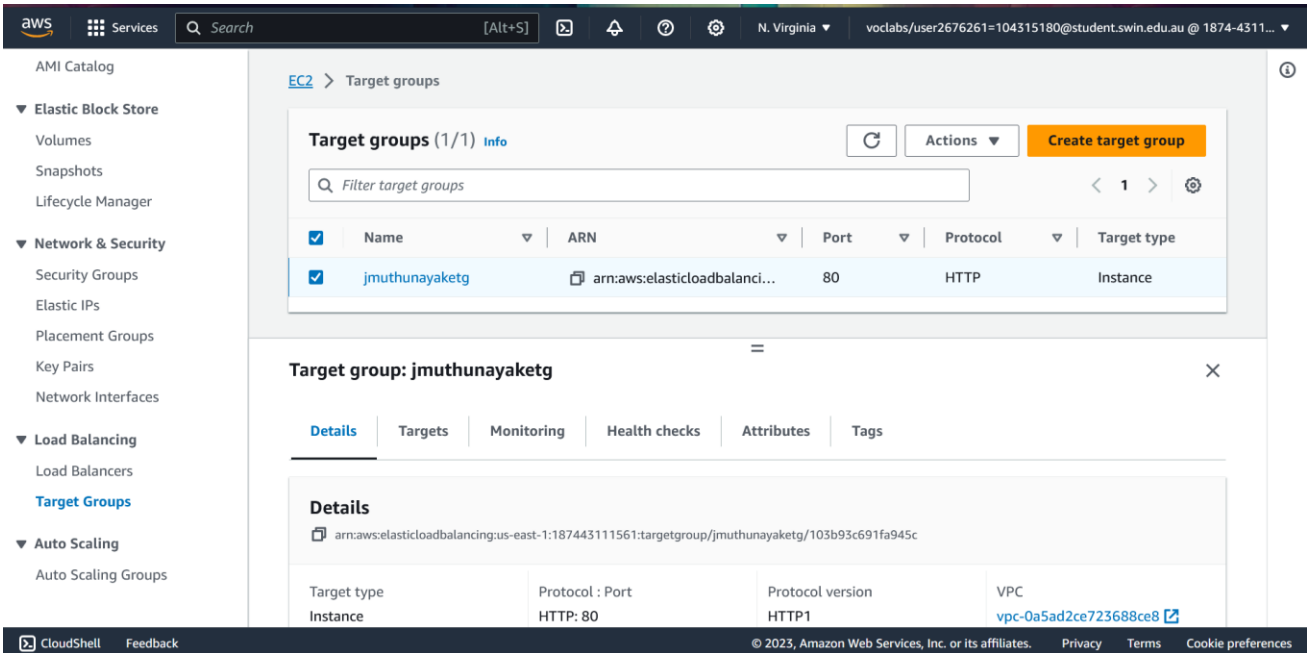


FIGURE 15 – TARGET GROUPS

FIGURE 16 – LOAD BALANCERS

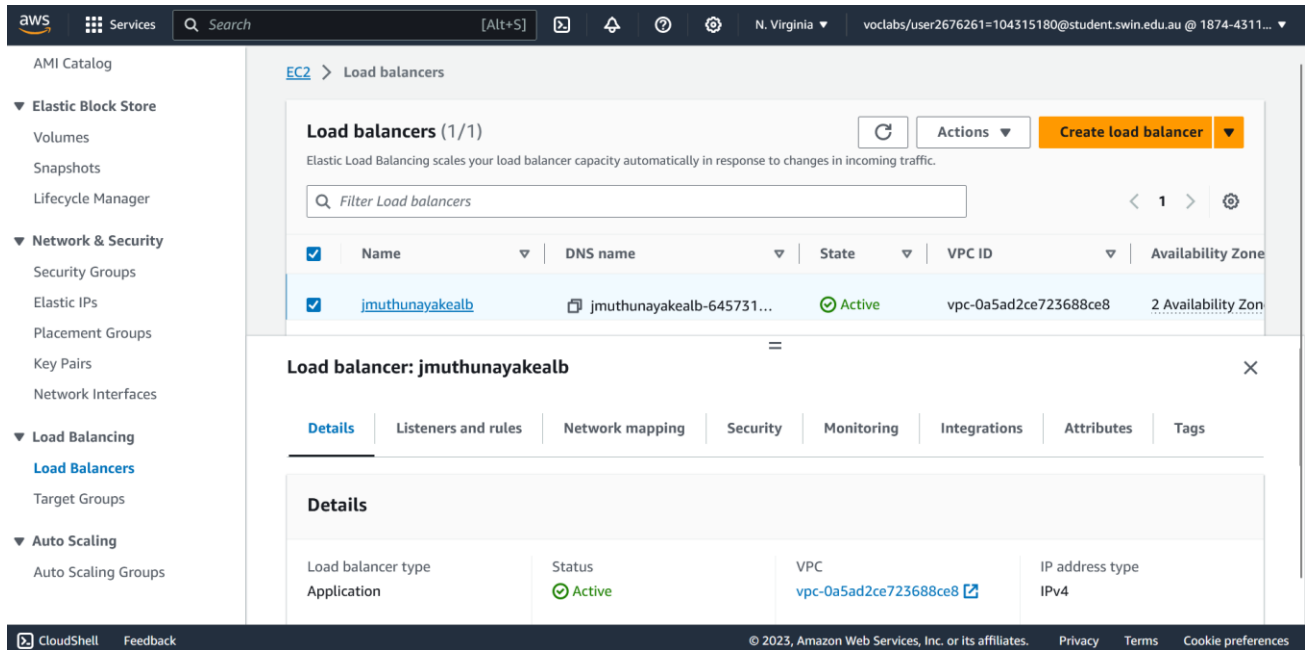
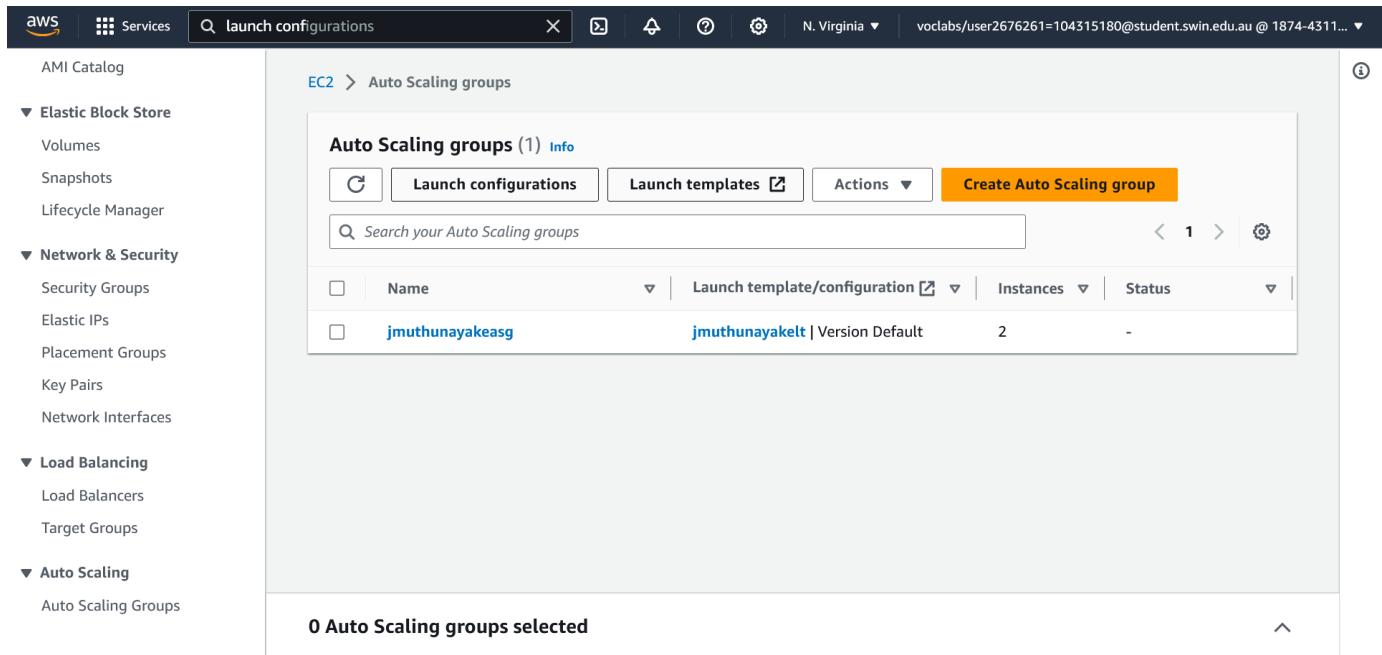


FIGURE 17 – AUTO SCALING GROUP



III. CHALLENGES

Despite successful instance launches and the meticulous configuration of essential AWS components, I encountered some challenges in the final phases of this deployment. The 504 Gateway Error I encountered when attempting to access the web application through the Load Balancer's IP address proved to be elusive in its resolution. Extensive checks on the Target Groups, Load Balancer, Launch Template, and Auto Scaling Group configurations yielded no obvious issues, leaving me in need of further investigation. Additionally, while trying to upload photos to the application, I encountered a persistent error, as depicted in Figure 18. This error has prevented the seamless functioning of the photo uploader feature, and its root cause remains under scrutiny. These challenges underline the complexities of AWS environments and the importance of meticulous troubleshooting to ensure smooth operation.

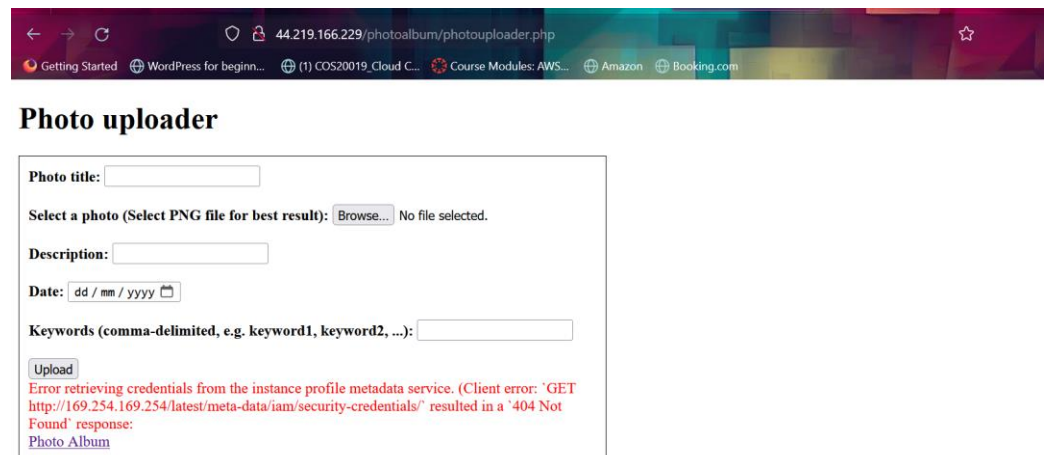


FIGURE 18 – ERROR IN PHOTOUPLoader.PHP

ACKNOWLEDGMENT

I would like to extend our sincere gratitude to our instructors and the entire support team for their guidance and assistance throughout the course and in completing this assignment. Their expertise and dedication have been invaluable in helping me navigate the intricacies of AWS services and infrastructure setup. Additionally, I want to express our appreciation to our fellow learners for fostering a collaborative learning environment. This experience has provided me with valuable

insights and skills in cloud computing and AWS, which will undoubtedly prove beneficial in our future endeavors.

REFERENCES

- [1] <https://aws.amazon.com/documentation-overview/>

Assignment 2

Checklist

Make sure all the following are completed.

Submission Checklist

Student Name: Janaka Muthunayake

Student Id: 104315180

Tutorial time: Wednesday 06:30pm

Date of submission: 09/10/2023

Submit to Canvas:

☒ A PDF document file as specified in the Submission section of the assignment specification.

Marking Scheme

☒ Infrastructure Requirements (10 marks)

VPC configured with 2AZs both with public and private subnets. Public and private route tables route to IGW and NAT, respectively. Security groups created and properly configured.

☒ NACL correctly configured.

☒ IAM roles properly configured

☒ ASG configured and working correctly.

☒ ELB configured and working correctly with associated Elastic Public IP address.

☒ Photos stored in S3 are correctly accessible.

☒ S3 bucket policy is correct.

☒ Lambda configured and working correctly.

☒ RDS configured and working correctly.

Functional Requirements (5 marks)

☒ Website accessible via ELB.

☒ Photos and their meta-data displayed on album.php page

☒ Photos and their meta-data can be uploaded to the S3 bucket and RDS database, respectively.

☒ Photos are resized by the Lambda function.