

**COS80001 – Cloud Engineering**

**Assignment 1B: Deploying a Photo Album Web Application on AWS on a custom VPC**

**Student Name:** Arun Ragavendhar Arunachalam Palaniyappan

**Student ID:** 104837257

**Tutorial Time:** Friday – 04:30 PM to 06:30 PM

**Assignment:** 1B - Photo Album Web Application Deployment

## 1. Infrastructure Setup

This section provides a detailed explanation of how the full cloud infrastructure was set up to deploy a secure and functional photo album web application using Amazon Web Services (AWS). The goal was to follow the assignment requirements closely while also applying best practices in cloud design. Every component—from the virtual private network to the way access was controlled for each part—was carefully created and configured. This included building a custom VPC to host all resources, creating separate subnets to keep public and private resources isolated, and using security tools like security groups and network ACLs to protect each layer. The setup ensured that all services could communicate properly while staying secure from external threats.

### 1.1 VPC and Subnets Configuration

A dedicated Virtual Private Cloud (VPC) named ARagavendharVPC was provisioned in the us-east-1 region with a base CIDR block of 10.0.0.0/16, offering broad IP address space for subnetting. Within this VPC, four subnets were created. Two public subnets (10.0.1.0/24 in AZ A and 10.0.2.0/24 in AZ B) were intended for outward-facing services, with Public Subnet 2 actively used to host the web server. The private layer was built with two subnets (10.0.3.0/24 for the RDS instance and 10.0.4.0/24 for the test instance), completely isolated from the internet for enhanced security. A dedicated public route table was linked to an Internet Gateway, ensuring only the public subnets had internet access. Private subnets were attached to a separate route table with no external routing, enforcing internal-only access.

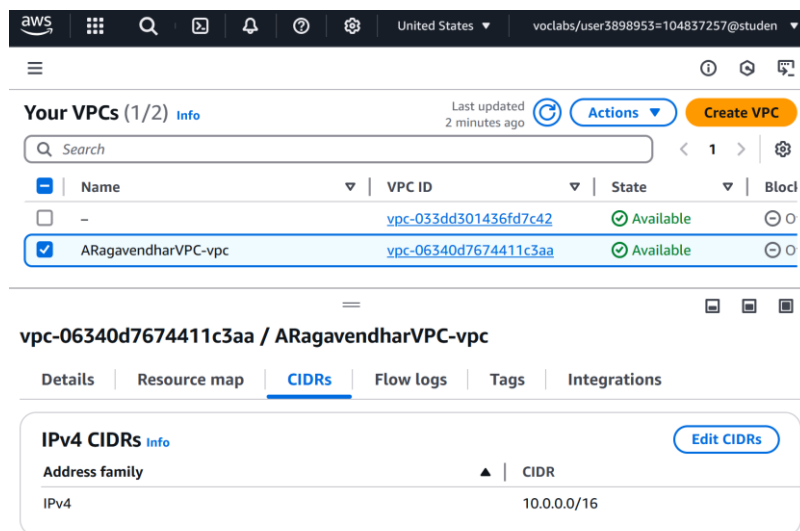


Figure 1 - Custom VPC with CIDR blocks

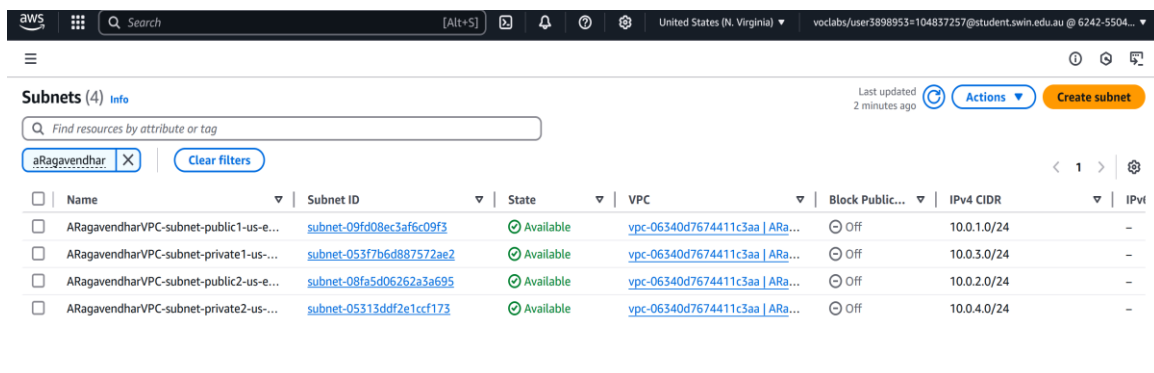


Figure 2 - Subnet configuration overview

The screenshot displays the AWS Management Console interface for Route Tables. The top navigation bar includes the AWS logo, a search bar, and user information. The main content area is titled "Route tables (1/4) Info" and shows a list of route tables. The selected route table, "rtb-0d28d385620e88418", is highlighted. Below the list, the "Routes" tab is active, showing two routes: a default route (0.0.0.0/0) pointing to an internet gateway and a local route (10.0.0.0/16).

Name	Route table ID	Explicit subnet associ...	Edge associations	Main	VPC	Own...
-	rtb-0ec94ebdc598a7818	-	-	Yes	vpc-0fbff943c75afcae4	624255...
AragavendharVPC-rtb-public	rtb-0d28d385620e88418	2 subnets	-	No	vpc-06340d7674411c3aa   ARa...	624255...
-	rtb-00773f78393804a70	-	-	Yes	vpc-06340d7674411c3aa   ARa...	624255...
AragavendharVPC-rtb-private	rtb-07d2df9f96bcd915d	2 subnets	-	No	vpc-06340d7674411c3aa   ARa...	624255...

Destination	Target	Status	Propagated
0.0.0.0/0	igw-0dda2e0b1e4d05e2d	Active	No
10.0.0.0/16	local	Active	No

Figure 3 - Public Route Table's route configurations

The screenshot shows the "Subnet associations" tab for the same public route table. It lists two explicit subnet associations, each linking a specific subnet to the route table. Below this, it indicates that there are no subnets without explicit associations.

Name	Subnet ID	IPv4 CIDR	IPv6 CIDR
AragavendharVPC-subnet-public1-us-east-1a	subnet-09fd08ec3af6c09f3	10.0.1.0/24	-
AragavendharVPC-subnet-public2-us-east-1b	subnet-08fa5d06262a3a695	10.0.2.0/24	-

Figure 4 - Public Route Table's subnet Associations

The screenshot displays the AWS Management Console interface for Route Tables, specifically for a private route table. The top navigation bar is consistent with the previous screenshots. The main content area is titled "Route tables (1/1) Info" and shows a list of route tables. The selected route table, "rtb-07d2df9f96bcd915d", is highlighted. Below the list, the "Routes" tab is active, showing two routes: a default route (0.0.0.0/0) pointing to a VPC endpoint and a local route (10.0.0.0/16).

Name	Route table ID	Explicit subnet associ...	Edge associations	Main	VPC	Own...
AragavendharVPC-rtb-private	rtb-07d2df9f96bcd915d	2 subnets	-	No	vpc-06340d7674411c3aa   ARa...	624255...

Destination	Target	Status	Propagated
0.0.0.0/0	vpc-0c4b730cbcca8518	Active	No
10.0.0.0/16	local	Active	No

Figure 5 - Private Route Table's route configurations

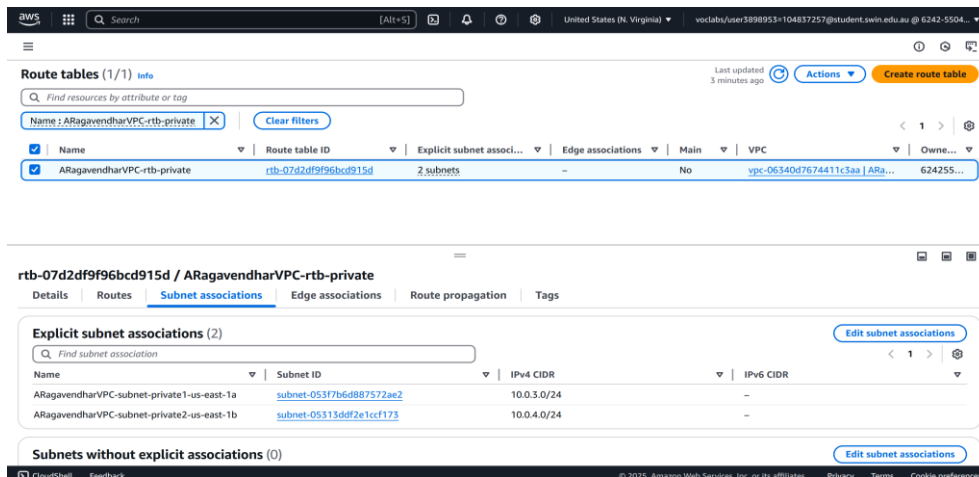


Figure 6 - Private Route Table's subnet Associations

## 1.2 Security Groups

To protect network boundaries at the instance level, three security groups were configured. WebServerSG allowed inbound HTTP (port 80) and SSH (port 22) access from any source to facilitate public interaction and remote administration. Additionally, it permitted ICMP (ping) requests strictly from the subnet of the test instance to verify internal connectivity. The DBServerSG was locked down to only allow MySQL (port 3306) traffic from the web server's security group, thus isolating the database from any direct access. The TestInstanceSG was left open to all traffic for flexibility during test phases, as it resided within a private subnet and was not exposed to the internet.

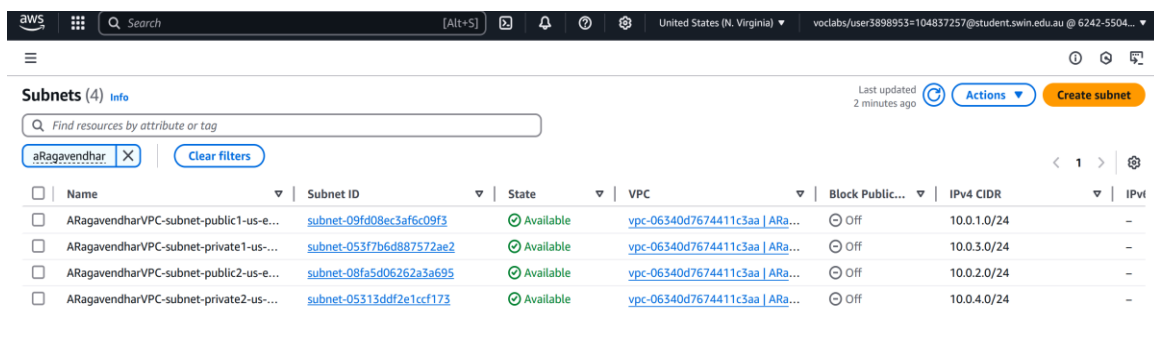


Figure 7 - WebServerSG Inbound Rules

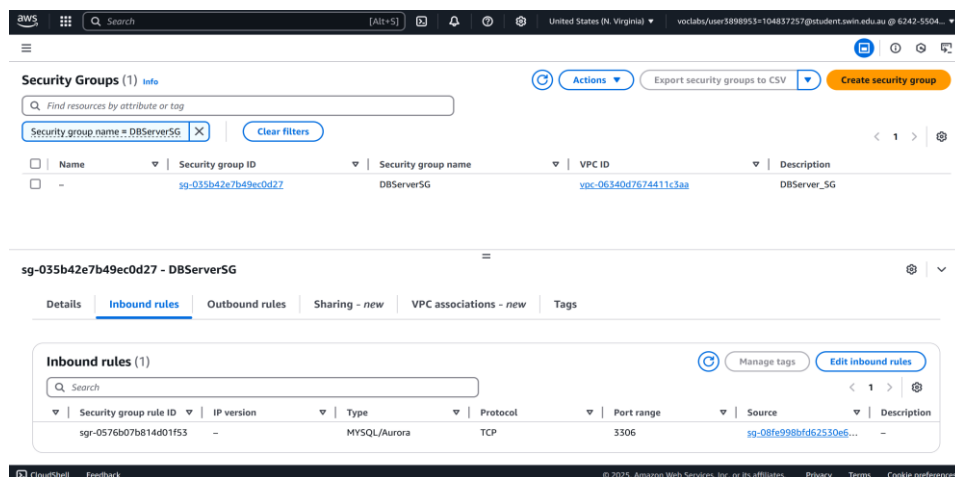


Figure 8 - DBServerSG Inbound Rules

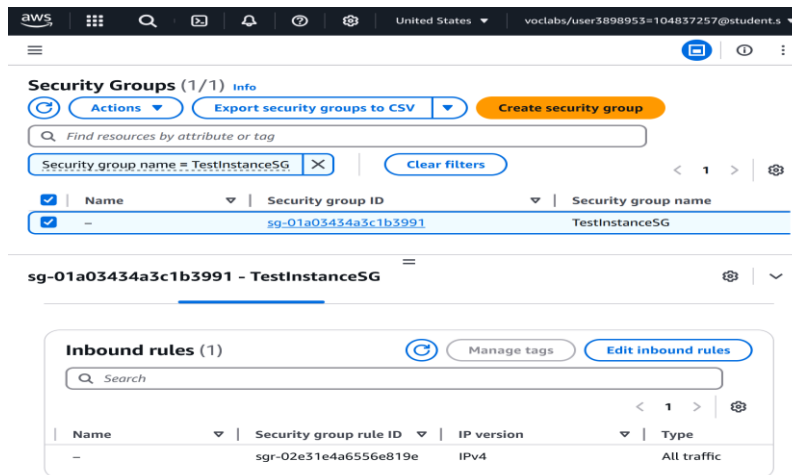


Figure 9 - TestInstanceSG Inbound Rules

### 1.3 EC2 Instances

The architecture included two EC2 virtual machines. The first, acting as both the web server and bastion host, was deployed in Public Subnet 2. It used the Amazon Linux 2023 AMI (t2.micro) and was set up via a startup script to install Apache, PHP, and phpMyAdmin. This instance was allocated an Elastic IP to guarantee a stable, externally accessible URL even after reboot. The second instance, used exclusively for testing, was created in Private Subnet 2 with SSH access routed through the web server.

To ensure persistent and reliable access to the web application, an Elastic IP address was created using the EC2 dashboard. This was necessary because public IP addresses assigned by AWS can change upon instance stop/start events. Associating an Elastic IP with the web server instance ensured that the application remained reachable through a consistent public address, which was also required for assignment submission and testing purposes.

To meet the internal connectivity requirement via ICMP, a successful ping test was conducted between the web server in the public subnet and the test instance in the private subnet. The test confirmed proper internal communication across subnets, with ICMP requests and responses functioning as expected. This validated that the web server could reach the internal resources securely, fulfilling the assignment requirement for subnet-level interaction.

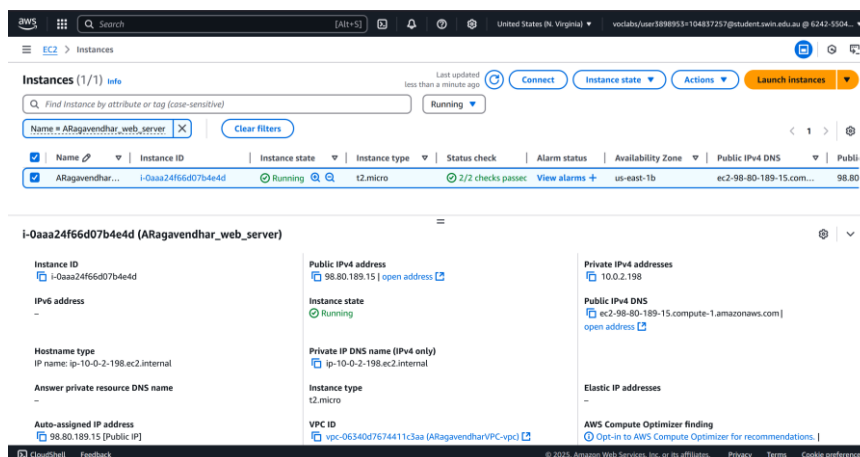


Figure 10 - Web Server EC2 Instance

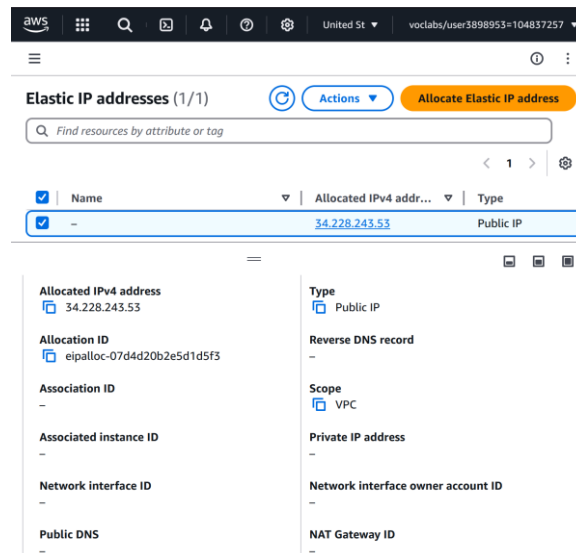


Figure 11 - Elastic IP creation

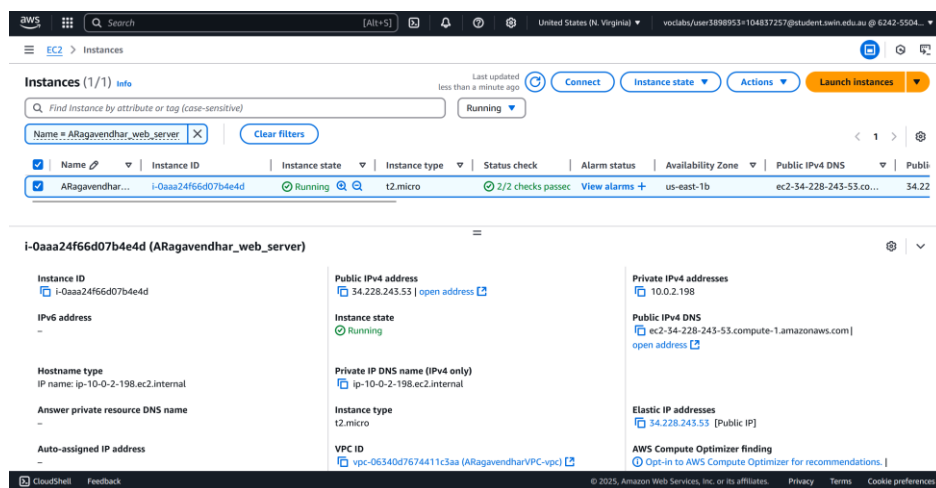


Figure 12 - Elastic IP address associated with the web server Instance

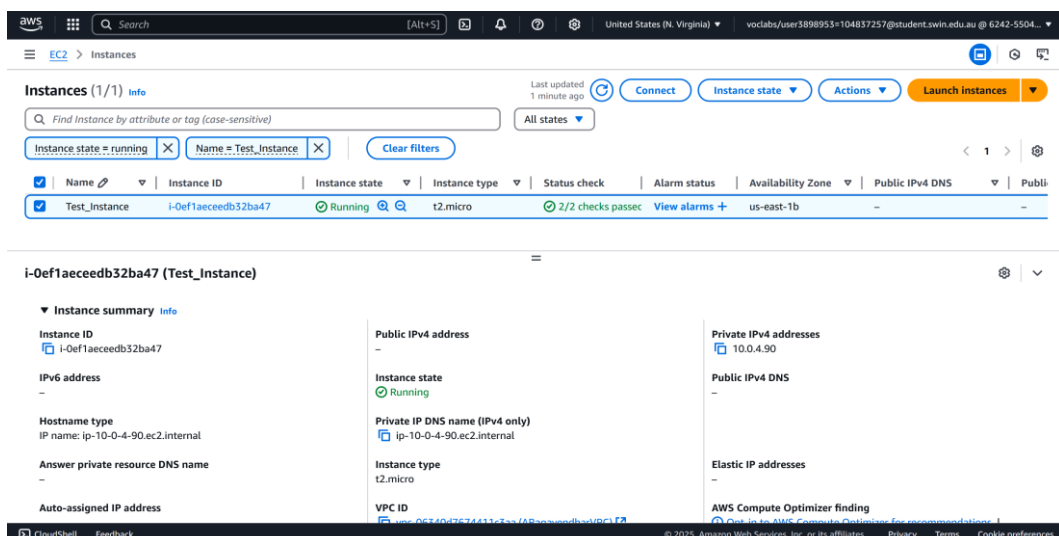


Figure 13 - Private Test Instance

```

[ec2-user@ip-10-0-2-198 ~]$ pwd
/home/ec2-user
[ec2-user@ip-10-0-2-198 ~]$ ping 10.0.4.90
PING 10.0.4.90 (10.0.4.90) 56(84) bytes of data.
64 bytes from 10.0.4.90: icmp_seq=1 ttl=127 time=1.11 ms
64 bytes from 10.0.4.90: icmp_seq=2 ttl=127 time=1.26 ms
64 bytes from 10.0.4.90: icmp_seq=3 ttl=127 time=0.303 ms
64 bytes from 10.0.4.90: icmp_seq=4 ttl=127 time=1.06 ms
64 bytes from 10.0.4.90: icmp_seq=5 ttl=127 time=0.932 ms
64 bytes from 10.0.4.90: icmp_seq=6 ttl=127 time=1.07 ms
64 bytes from 10.0.4.90: icmp_seq=7 ttl=127 time=0.936 ms
64 bytes from 10.0.4.90: icmp_seq=8 ttl=127 time=1.15 ms
64 bytes from 10.0.4.90: icmp_seq=9 ttl=127 time=1.93 ms
64 bytes from 10.0.4.90: icmp_seq=10 ttl=127 time=1.35 ms
64 bytes from 10.0.4.90: icmp_seq=11 ttl=127 time=1.07 ms
64 bytes from 10.0.4.90: icmp_seq=12 ttl=127 time=0.552 ms
^C
--- 10.0.4.90 ping statistics ---
12 packets transmitted, 12 received, 0% packet loss, time 11246ms
rtt min/avg/max/mdev = 0.303/1.060/1.934/0.383 ms
[ec2-user@ip-10-0-2-198 ~]$

```

Figure 14 - Ping result from Web Server to Test Instance

## 1.4 RDS Database

A MySQL 8.0.39 database was created using Amazon RDS and launched into Private Subnet 1 with public access explicitly disabled. It was protected by DBServerSG, which only permitted traffic originating from the web server. A table named photos was manually created using phpMyAdmin with the required fields: title, description, creation\_date, keywords, and s3\_reference. Each field was defined with suitable data types (VARCHAR and DATE), exactly matching the assignment requirements. phpMyAdmin was installed on the EC2 web server, providing a user-friendly interface to manage database entries securely without exposing the RDS instance externally.

The screenshot displays the AWS Management Console interface for an Amazon RDS instance. The top navigation bar shows the AWS logo, a search bar, and the current region (United States (N. Virginia)). The breadcrumb trail indicates the path: Aurora and RDS > Databases > aragavendhar-db. The instance name 'aragavendhar-db' is prominently displayed at the top of the main content area, along with 'Modify' and 'Actions' buttons. Below this, a 'Summary' section provides key details: DB identifier (aragavendhar-db), Status (Available), Role (Instance), Engine (MySQL Community), CPU (4.26%), Class (db.t4g.micro), Current activity (0 Connections), Region & AZ (us-east-1a), and a link to '2 Recommendations'. A horizontal tab bar below the summary includes 'Connectivity & security' (selected), 'Monitoring', 'Logs & events', 'Configuration', 'Zero-ETL integrations', 'Maintenance & backups', 'Data migrations - new', and 'Tag'. The 'Connectivity & security' section is expanded, showing three sub-sections: 'Endpoint & port' (Endpoint: aragavendhar-db.cc7i6xj0ixzq.us-east-1.rds.amazonaws.com, Port: 3306), 'Networking' (Availability Zone: us-east-1a, VPC: ARagavendhar/VPC-vpc (vpc-06340d7674411c3aa), Subnet group: aragavendhar-db-subnet-group-private-1), and 'Security' (VPC security groups: DBServerSG (sg-035b42e7b49ec0d27) - Active, Publicly accessible: No, Certificate authority: rds-ca-rsa2048-g1).

Figure 15 - RDS Instance Configuration

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	Photo_title	varchar(255)	utf8mb4_0900_ai_ci		No	None		
2	Description	varchar(255)	utf8mb4_0900_ai_ci		No	None		
3	Creation_date	date			No	None		
4	Keywords	varchar(255)	utf8mb4_0900_ai_ci		No	None		
5	Reference_to_photo_object_in_S3	varchar(255)	utf8mb4_0900_ai_ci		No	None		

Figure 16 - photos table schema in phpMyAdmin

## 1.5 Network ACLs

A Network ACL named PublicSubnet2NACL was configured and attached to Public Subnet 2 to apply an additional layer of security beyond security groups. Initially, issues were encountered where the web server could not be accessed over HTTP even though the rules appeared correct. Through research, it was discovered that unlike security groups, NACLs are stateless, meaning that both inbound and outbound rules must be explicitly defined. I also learned that HTTP and ICMP responses use ephemeral ports (typically ranging from 1024 to 65535), which were initially not accounted for. After reading AWS documentation and reviewing community forums [1,2], the outbound rules were updated to allow traffic over the ephemeral range, enabling bidirectional communication. The ACL rules were configured to allow SSH (22) and HTTP (80) from any source, ICMP from the test subnet, and the necessary ephemeral ports for outbound responses, strictly applying the least privilege principle.

Rule number	Type	Protocol	Port range	Source	Allow/Deny
100	HTTP (80)	TCP (6)	80	0.0.0.0/0	Allow
110	All ICMP - IPv4	ICMP (1)	All	10.0.4.0/24	Allow
120	SSH (22)	TCP (6)	22	0.0.0.0/0	Allow
130	Custom TCP	TCP (6)	1024 - 65535	10.0.3.0/24	Allow
*	All traffic	All	All	0.0.0.0/0	Deny

Figure 17 - Inbound Rules for PublicSubnet2NACL



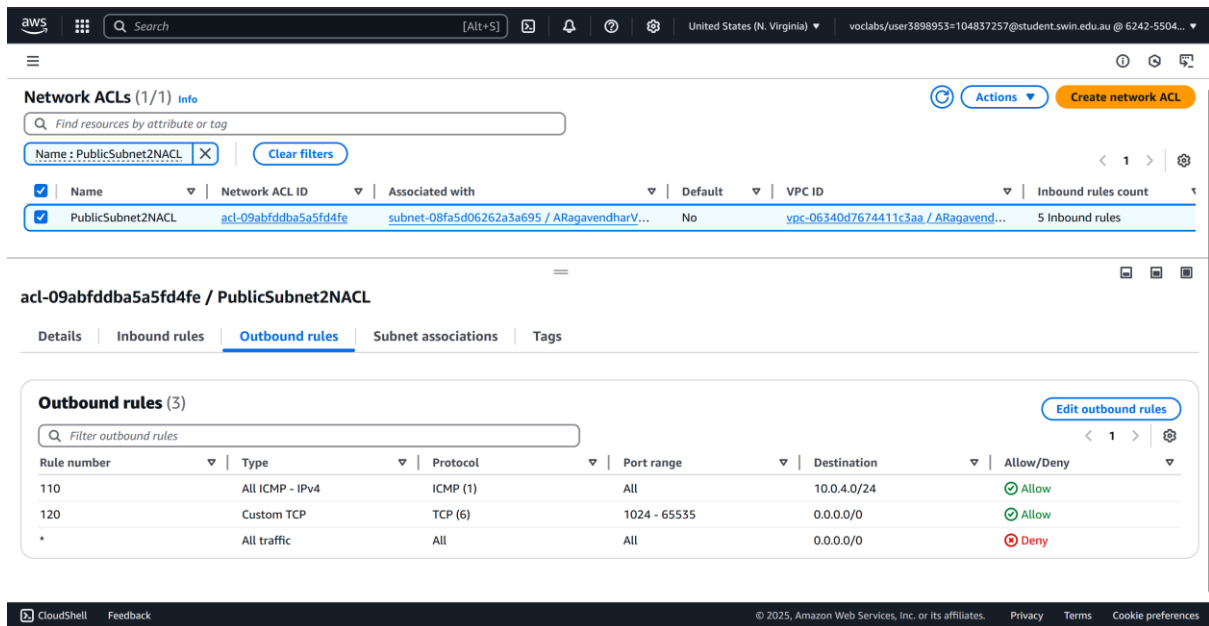


Figure 18 - Outbound Rules for PublicSubnet2NACL

## 2. Photo Album Application Deployment

### 2.1 S3 Bucket Configuration

An S3 bucket named aragavendhar-photo-album was created to store photo files for the web application. Images were uploaded manually through the AWS Console. Before applying public access permissions, the **Block all public access** setting in the bucket's **Permissions** tab was modified—all four checkboxes were unchecked, and the acknowledgement prompt was confirmed. This step was Important because, without disabling these settings, the bucket policy would not take effect. To comply with assignment requirements, a bucket-wide policy was then applied to enable public read-only access using the s3:GetObject permission for all objects in the bucket. This approach avoided modifying each object's permission individually.

#### Edit Block public access (bucket settings) Info

**Block public access (bucket settings)**

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to all your S3 buckets and objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to your buckets or objects within, you can customize the individual settings below to suit your specific storage use cases. [Learn more](#)

☐ **Block all public access**  
Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

- ☐ **Block public access to buckets and objects granted through new access control lists (ACLs)**  
S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.
- ☐ **Block public access to buckets and objects granted through any access control lists (ACLs)**  
S3 will ignore all ACLs that grant public access to buckets and objects.
- ☐ **Block public access to buckets and objects granted through new public bucket or access point policies**  
S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.
- ☐ **Block public and cross-account access to buckets and objects through any public bucket or access point policies**  
S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

[Cancel](#) [Save changes](#)

Figure 19 – Unchecking all boxes in Block public access section under bucket permissions

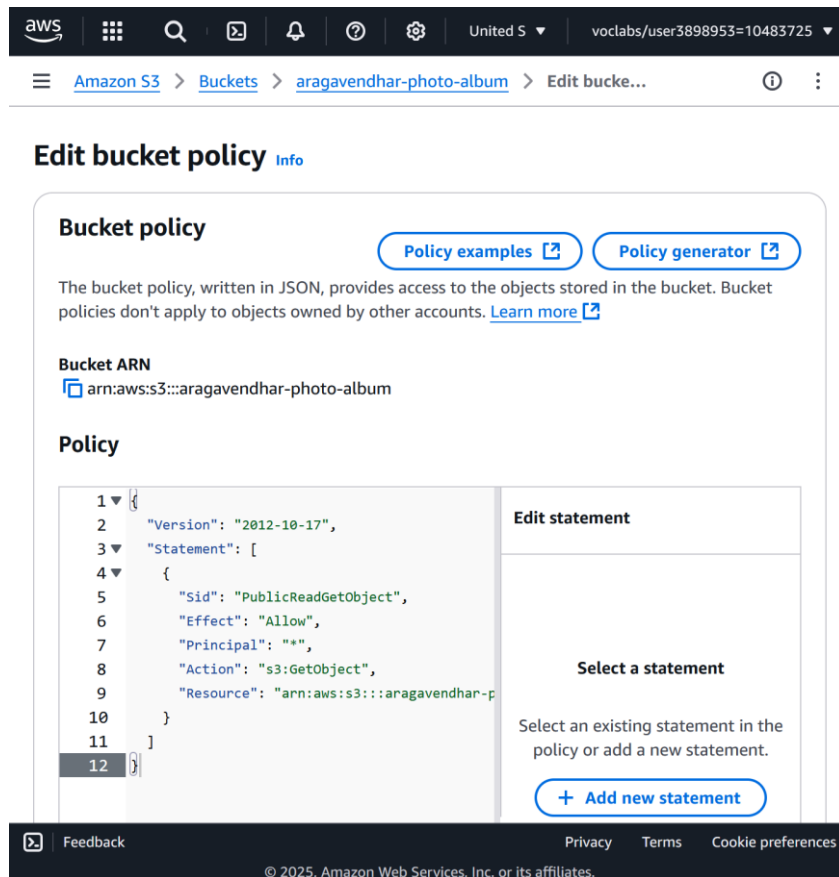


Figure 20 - S3 Bucket Policy Settings

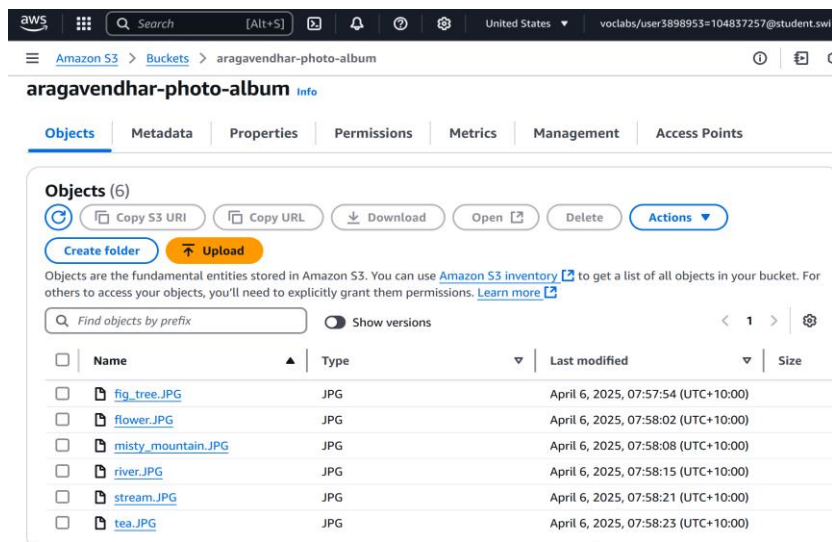


Figure 21 – 6 objects added to S3 bucket

The public URLs of the images were tested successfully in a private browser session to confirm that global access was enabled as intended.

URL: [https://aragavendhar-photo-album.s3.us-east-1.amazonaws.com/misty\\_mountain.JPG](https://aragavendhar-photo-album.s3.us-east-1.amazonaws.com/misty_mountain.JPG)

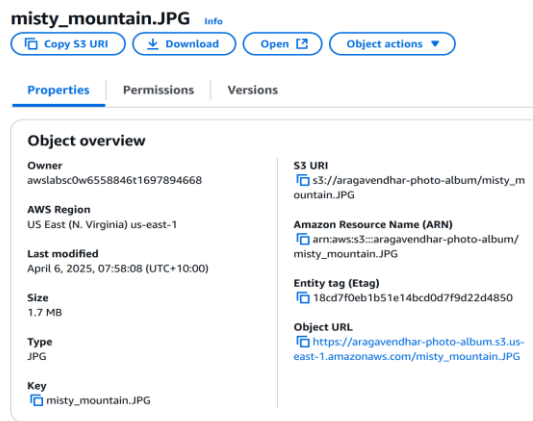


Figure 22 – public access URL of misty\_mountain object from the S3 bucket

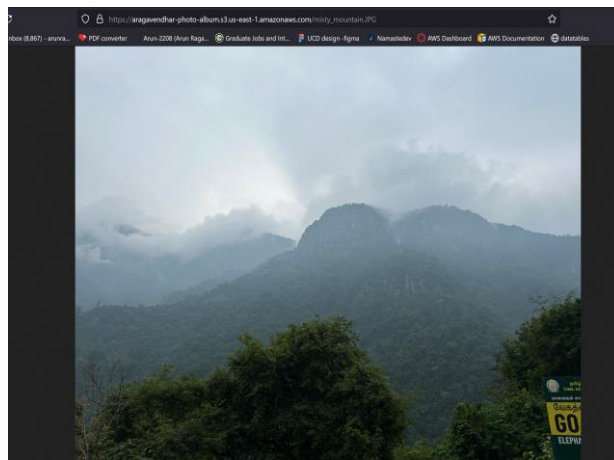


Figure 23 - Public Access Confirmation via Object URL

## 2.2 Metadata Storage in RDS

Each image uploaded to S3 had its respective metadata manually entered into the photos table through phpMyAdmin. This metadata included a descriptive title, brief description, creation date, a set of tags (keywords), and the full S3 object reference URL. These records were accurately reflected on the webpage.

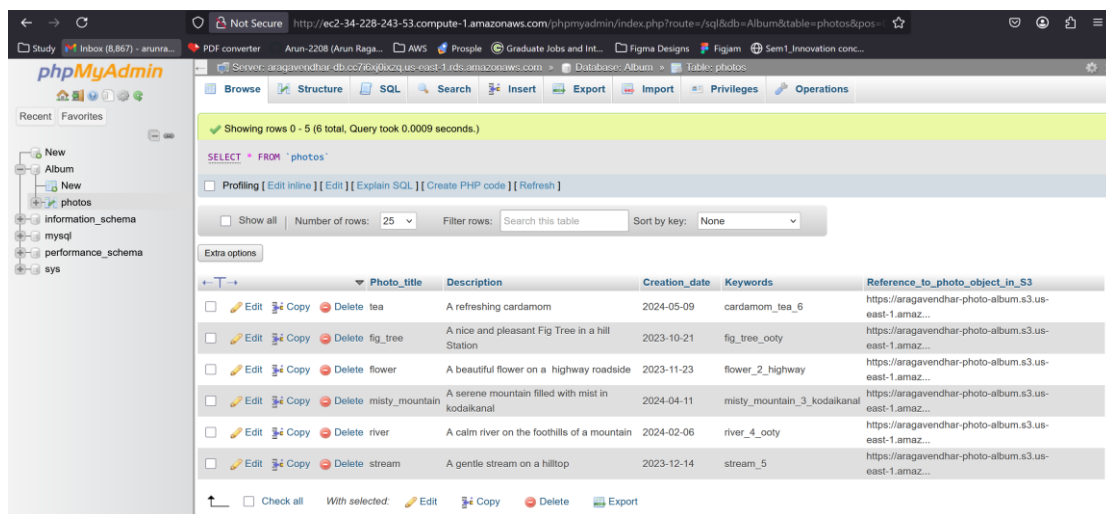
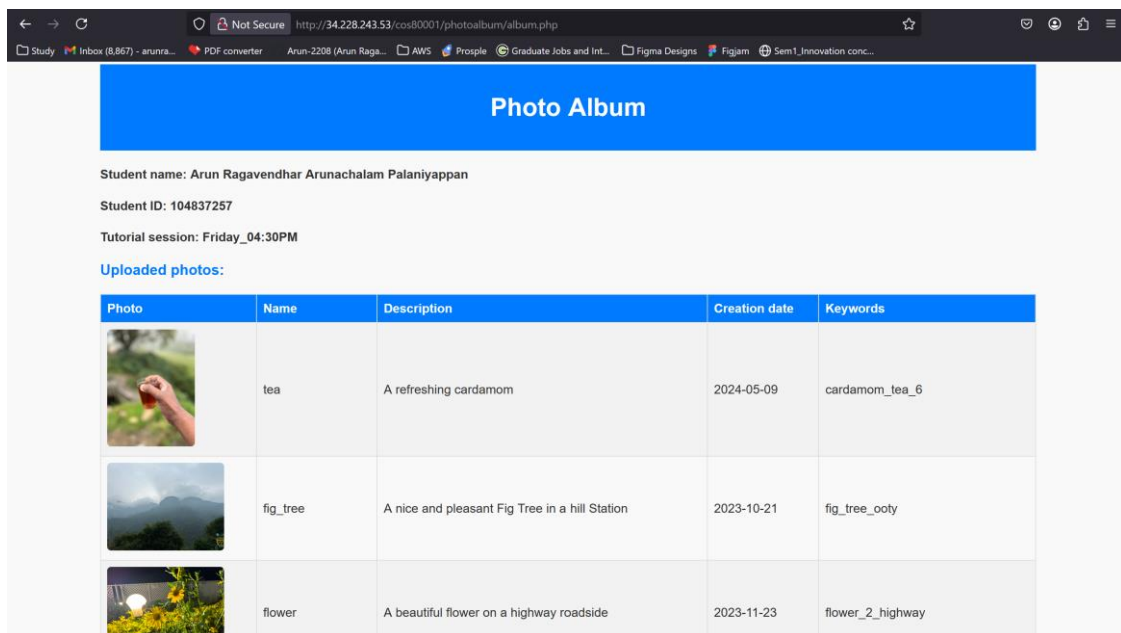


Figure 24 - metadata records in phpMyAdmin




## 2.3 Web Application Setup

The provided photoalbum\_v3.0 code was deployed to the /var/www/html/cos80001/photoalbum/ directory. The configuration file constants.php was edited to include the MySQL connection parameters and column mappings. The application became accessible. Upon testing, the webpage successfully rendered uploaded images alongside metadata. Features like keyword search and filtering were functional.

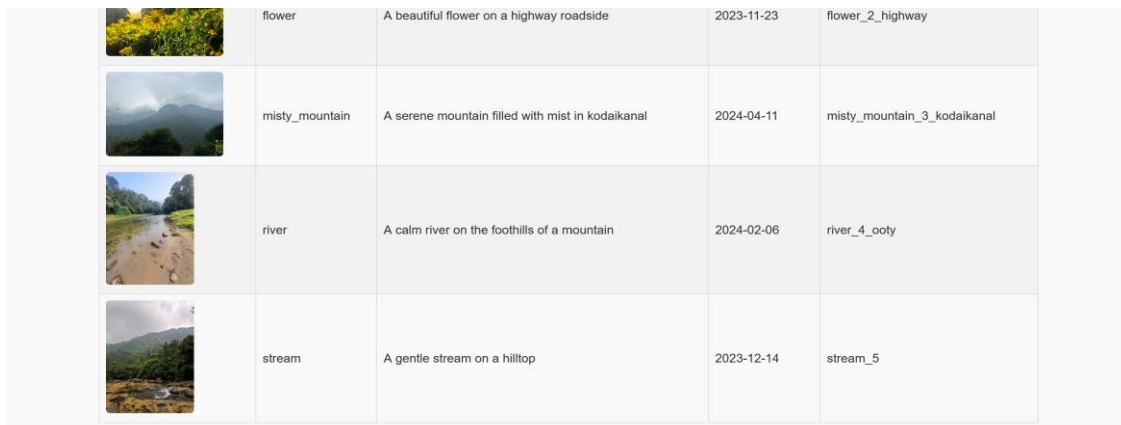
**URL:** <http://ec2-34-228-243-53.compute-1.amazonaws.com/cos80001/photoalbum/album.php>







The screenshot shows a web browser displaying the 'Photo Album' application. At the top, a blue header contains the title 'Photo Album'. Below the header, the student's name 'Arun Ragavendhar Arunachalam Palaniyappan' and ID '104837257' are listed, along with the tutorial session 'Friday\_04:30PM'. A link 'Uploaded photos:' leads to a table of uploaded photos. The table has five columns: Photo, Name, Description, Creation date, and Keywords. Three photos are listed: 'tea', 'fig\_tree', and 'flower'.

Photo	Name	Description	Creation date	Keywords
	tea	A refreshing cardamom	2024-05-09	cardamom_tea_6
	fig_tree	A nice and pleasant Fig Tree in a hill Station	2023-10-21	fig_tree_ooty
	flower	A beautiful flower on a highway roadside	2023-11-23	flower_2_highway

*Figure 25 - album.php web page displaying the photos and their metadata*



The screenshot shows a continuation of the 'Photo Album' application, displaying a table of uploaded photos. The table has five columns: Photo, Name, Description, Creation date, and Keywords. Four photos are listed: 'flower', 'misty\_mountain', 'river', and 'stream'.

	flower	A beautiful flower on a highway roadside	2023-11-23	flower_2_highway
	misty_mountain	A serene mountain filled with mist in kodaikanal	2024-04-11	misty_mountain_3_kodaikanal
	river	A calm river on the foothills of a mountain	2024-02-06	river_4_ooty
	stream	A gentle stream on a hilltop	2023-12-14	stream_5

*Figure 26 - album.php web page displaying the photos and their metadata*

## 3. Testing and Validation

Internal connectivity was verified by executing a successful ping from the web server in the public subnet to the test instance in the private subnet. Screenshots were captured for all major configurations, such as security groups, NACLs, and routing tables. The album page was verified to be functional, and the use of an Elastic IP ensured consistent access across instance reboots. All configurations were validated against assignment requirements.

## 4. Challenges and Learnings

A major challenge during deployment was the unexpected failure of the web server to serve HTTP requests despite correct-looking NACL rules. Initially, this was confusing as the security groups allowed the necessary ports. It was then realised that NACLs do not behave like security groups—they are stateless and require symmetrical rules for request and response traffic. I had to explicitly allow outbound ephemeral ports (1024–65535) to accommodate return traffic for HTTP and SSH connections. This was an important learning as the AWS documentation and community blogs were studied and researched to understand how traffic flow behaves at the subnet level [3]. Another helpful article from CloudAffaire document [2] helped clarify the concept of stateless firewalls.

Initially, there was an issue with getting the phpMyAdmin working due to a missing PHP-MySQL module, which was not included by default. This led to exploration of package installations using yum and verifying Apache module dependencies. After installing php-mysqld and restarting Apache, phpMyAdmin worked correctly.

Finally, interpreting the ICMP test requirement from a different perspective allowed me to simplify the task. Rather than SSHing into the private test instance, I initiated the ping from the public web server. Since ICMP is bidirectional, this verified internal connectivity effectively and required fewer steps, which also minimised SSH key management and connection hops.

These experiences provided hands-on knowledge about traffic flow design, the relationship between NACLs and security groups, and how real-time debugging is essential in cloud environments.

## 5. Conclusion

Through this assignment, a secure and functional AWS infrastructure was developed to support a fully operational dynamic web application. The design involved creating a custom VPC with well-structured public and private subnets, deploying EC2 instances, configuring RDS for data storage, and applying network-level security measures. Each task reinforced a practical understanding of cloud networking principles. During the process, several real-world challenges were encountered and resolved, particularly related to ACL misconfigurations, software package dependencies, and instance communication across subnet boundaries. These issues provided valuable opportunities to apply technical knowledge in practical scenarios.

Overall, the experience contributed to a deeper comprehension of cloud engineering concepts and demonstrated how AWS services can be combined to create secure, scalable, and reliable solutions in line with industry best practices.

## 6. References

- [1] Amazon Web Services. (2024). Control traffic to subnets using Network ACLs. AWS Documentation. <https://docs.aws.amazon.com/vpc/latest/userguide/vpc-network-acls.html>
- [2] CloudAffaire. (2023). AWS NACL vs Security Group. <https://cloudaffaire.com/aws-nacl-vs-security-group/>
- [3] Amazon Web Services. (2024). Ephemeral ports and firewall configurations. AWS Knowledge Center. <https://aws.amazon.com/premiumsupport/knowledge-center/ephemeral-ports/>