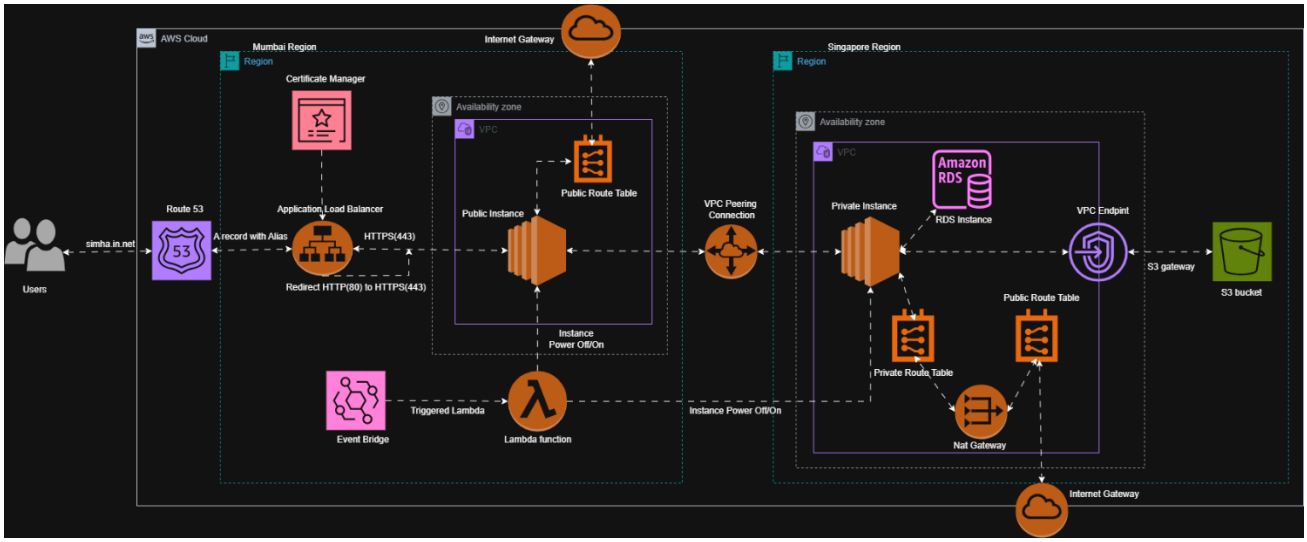
A Secure Multi-Tier Web Application Architecture on AWS

Table of Contents

1. Introduction:

This document outlines the steps to configure and establish a Secure Multi-Tier Web Application Architecture on AWS.



2. Pre-requisites

- Appropriate Access/Credentials to login to the AWS Console.
- AmazonEC2FullAccess
- AmazonVPCFullAccess
- AmazonRDSFullAccess
- AmazonS3FullAccess
- AWSCertificateManagerFullAccess
- AWSLambda_FullAccess
- CloudWatchEventsFullAccess
- AmazonRoute53FullAccess

3. Summary:

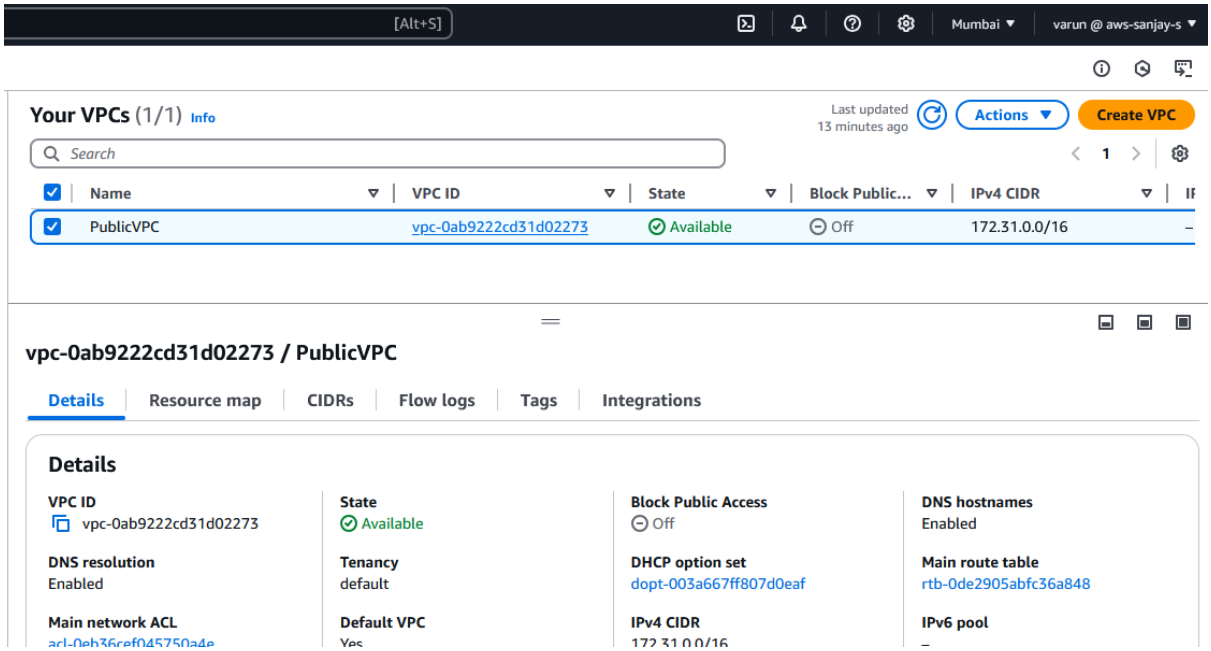
The architecture ensures secure and efficient operations by maintaining a strict separation between public and private resources. It leverages key AWS services such as Amazon RDS for reliable database management, S3 with VPC endpoints for secure data storage, and ACM for SSL certification to enhance security. Amazon Route 53 is ensuring reliable domain name resolution and traffic routing. VPC peering enables seamless and secure communication across AWS regions, while a NAT Gateway in the private subnet ensures controlled internet access for updates and external communications without exposing private instances. An Elastic Load Balancer (ELB) ensures efficient traffic distribution and availability. Automation is implemented using AWS Lambda and Event Bridge, enabling cost optimization by powering instances on and off during non-business hours.

This project represents a robust, scalable, and cost-effective solution for deploying secure web applications. The architecture is future-ready, with provisions for advanced features such as auto-scaling, ensuring it can adapt to growing business demands while maintaining optimal security and performance.

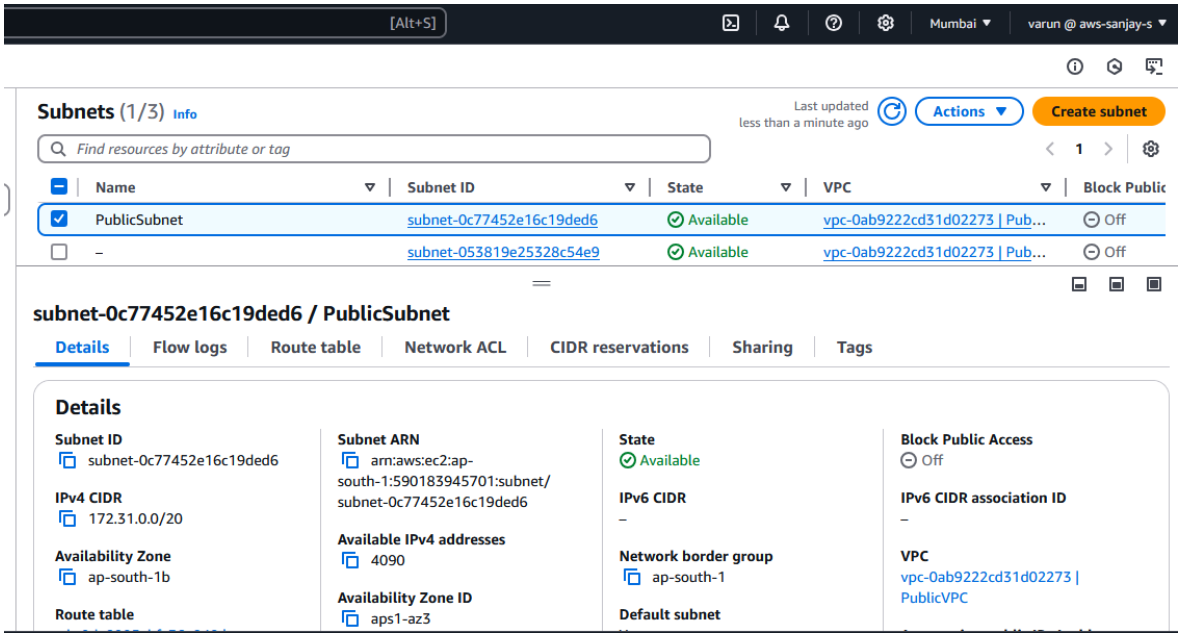
Module 1:

Set up a secure connection between the public instance and the private instance located in different regions.

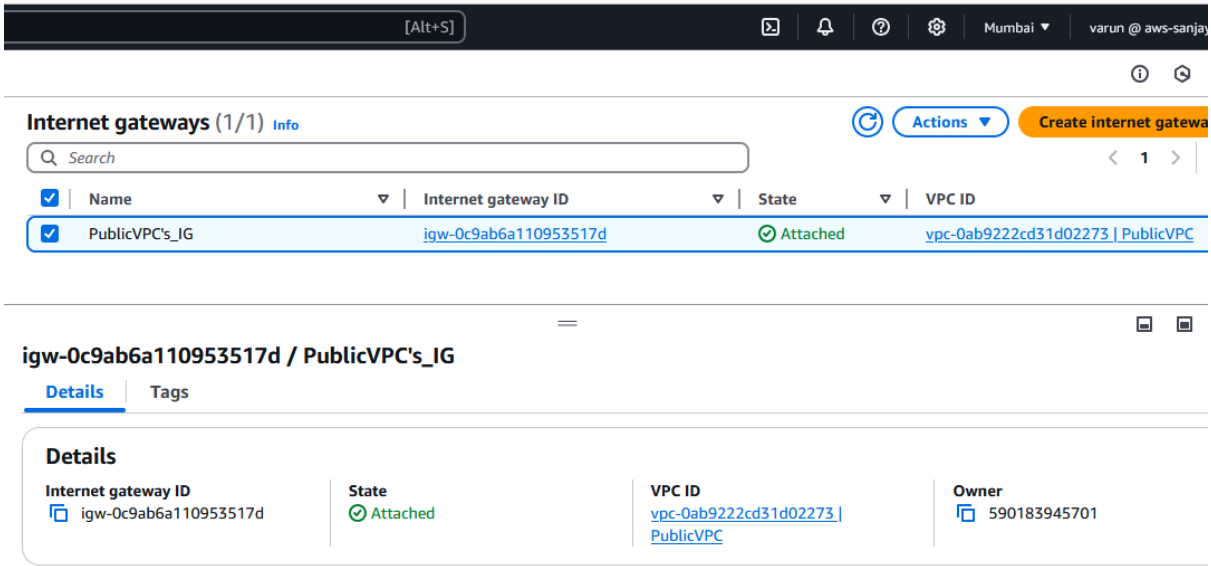
1. Create a VPC with a CIDR block of “172.31.0.0/16” in the Mumbai region.



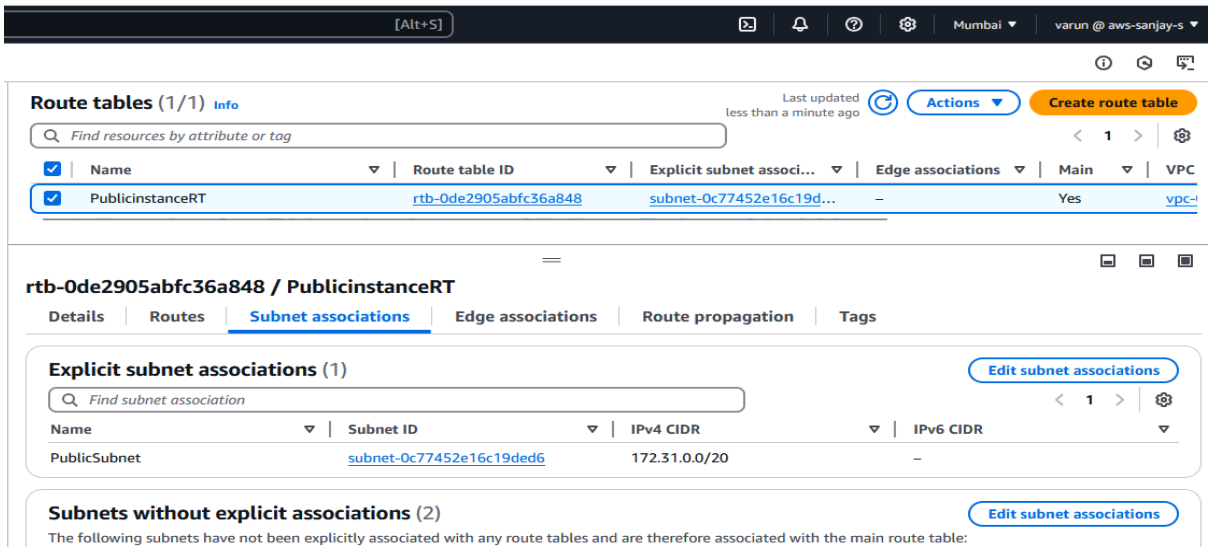
2. Create a subnet within the VPC and modify its settings to enable the assignment of public IP addresses.

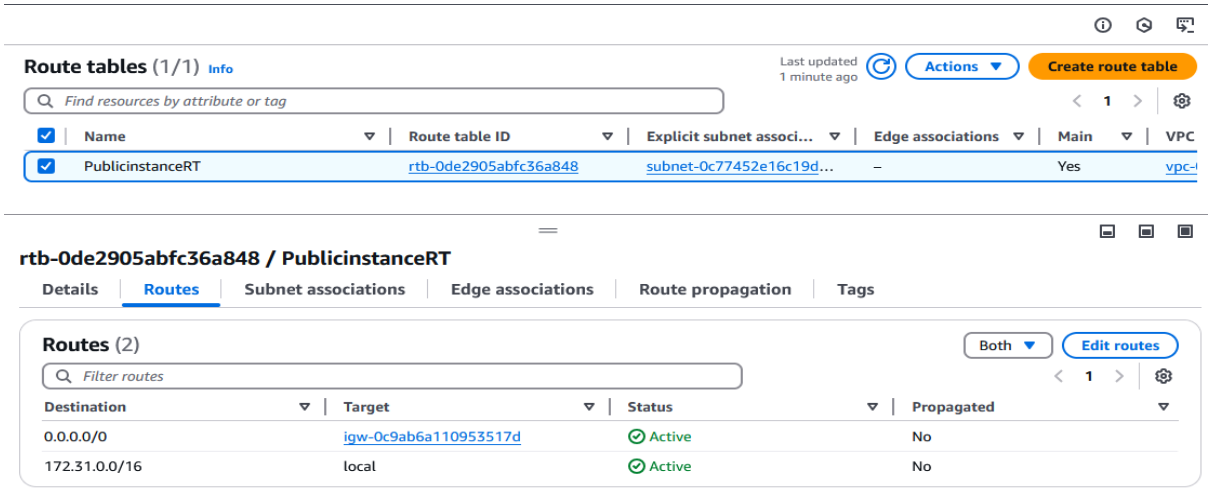


3. Create an Internet Gateway and attach it to the VPC.

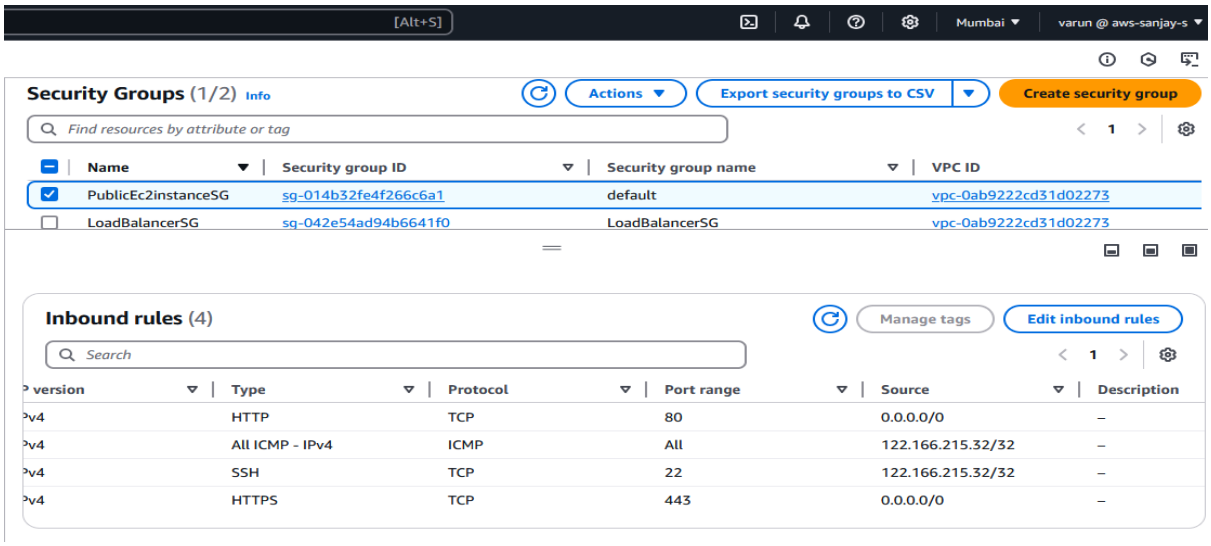


4. Update the VPC's route table to associate it with the subnet and add a route to the Internet Gateway linked to that VPC.

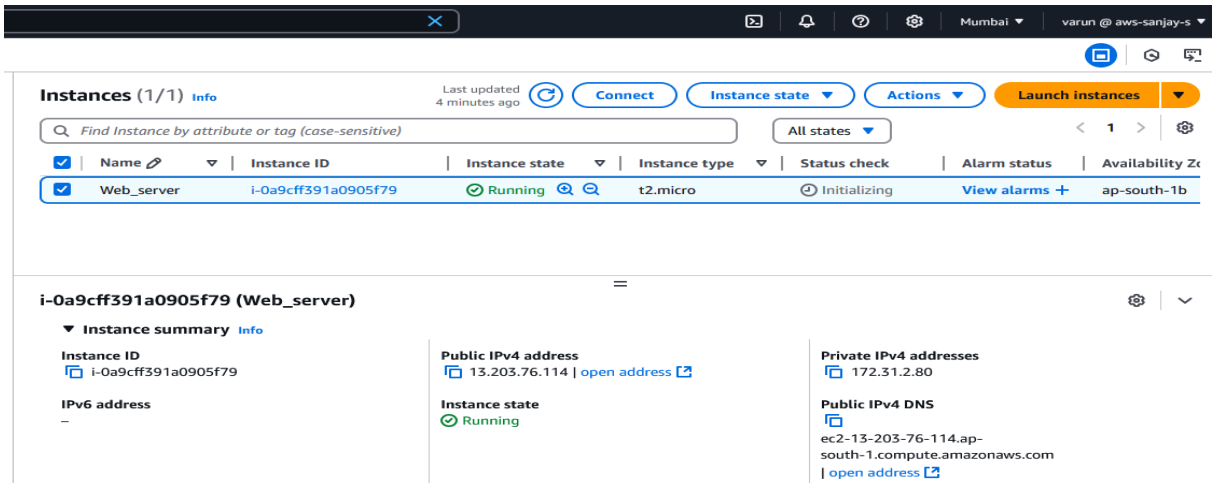




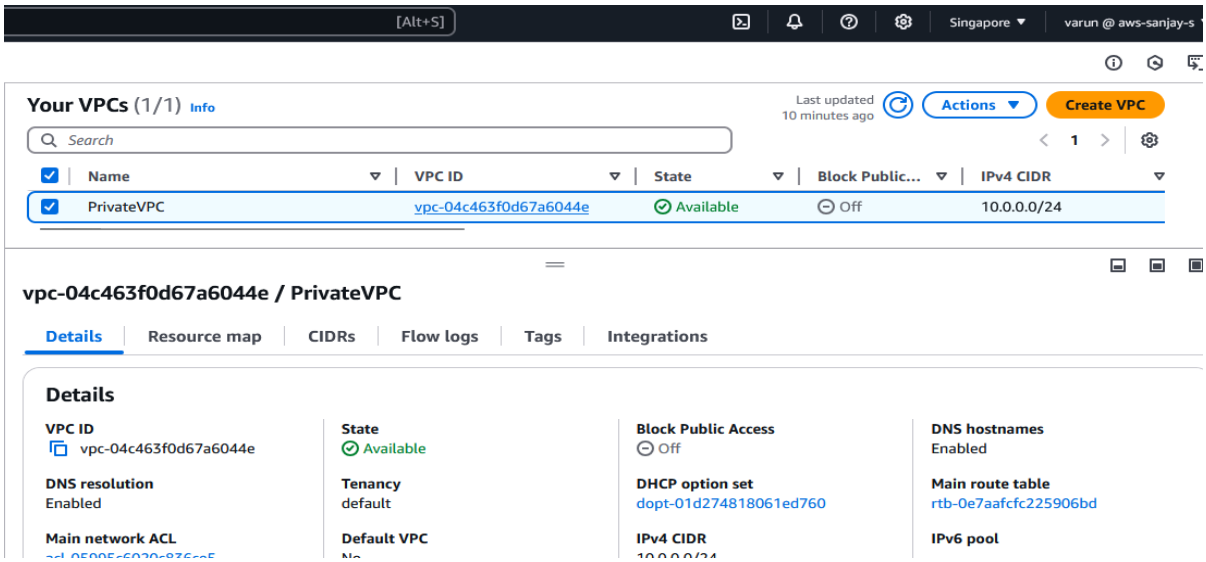
5. Create a security group for the VPC with inbound rules that allow HTTP and HTTPS traffic from anywhere, as well as SSH and ICMP (IPv4) for the specified IP range.

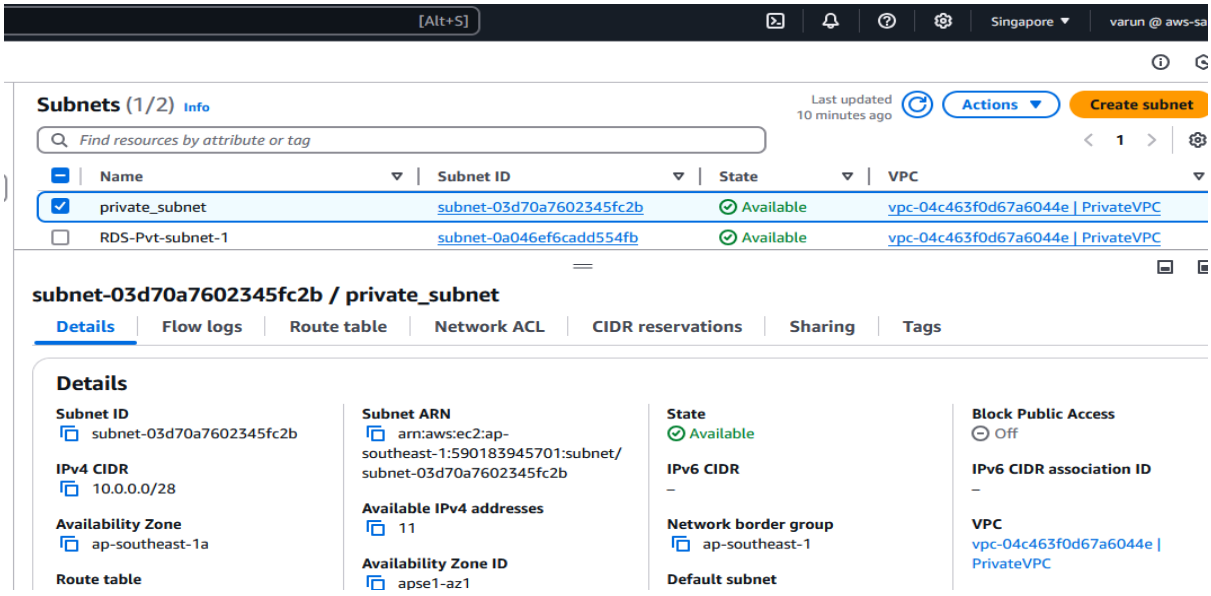


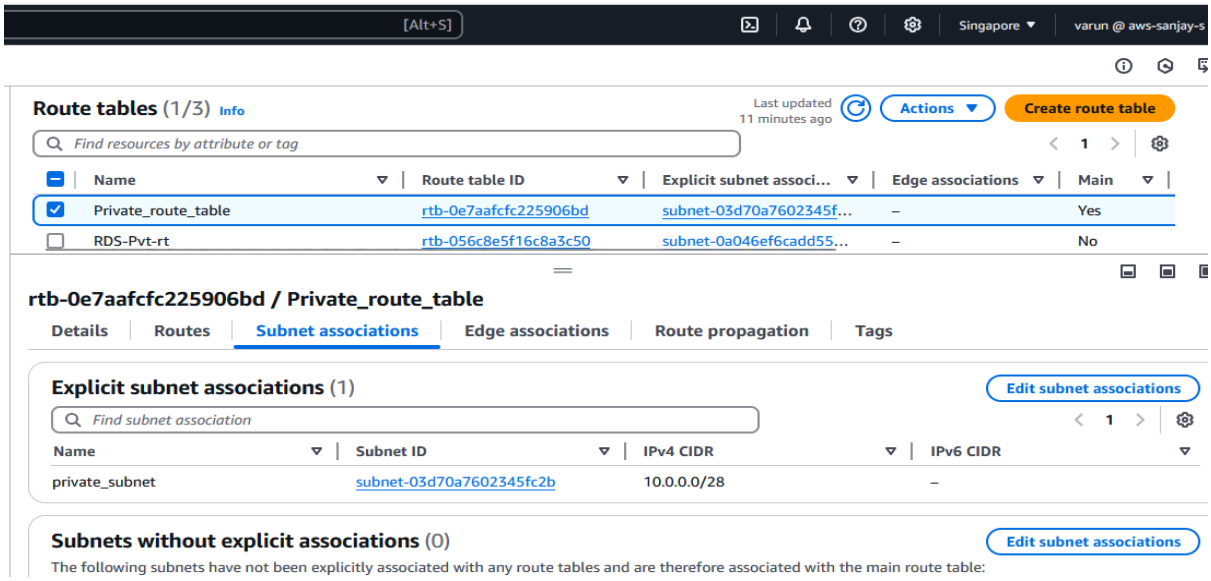
6. Launch a web server (public instance) within the public VPC to host the web application.

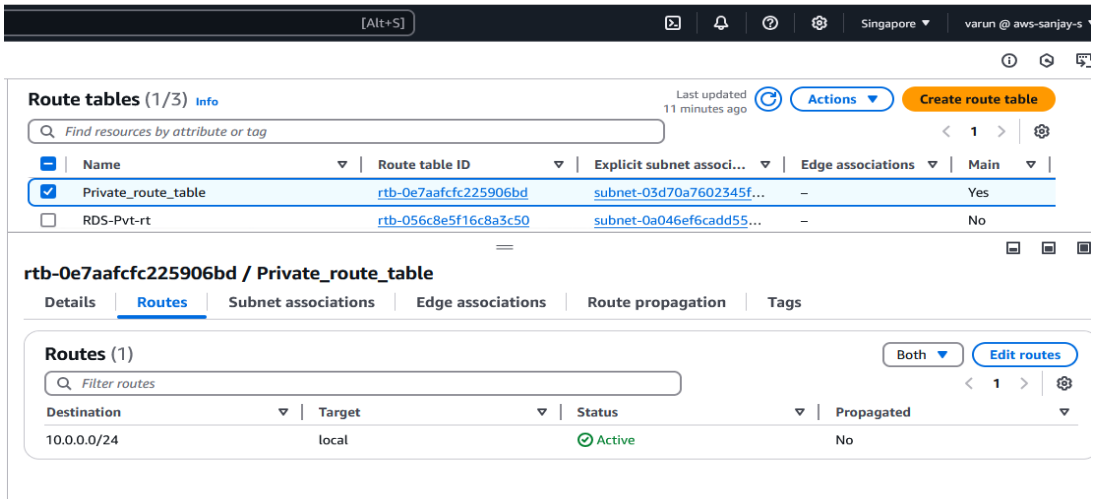


7. Similar, create another VPC for a private network, ensuring that the subnet does not have a public IP enabled and that no Internet Gateway is required.

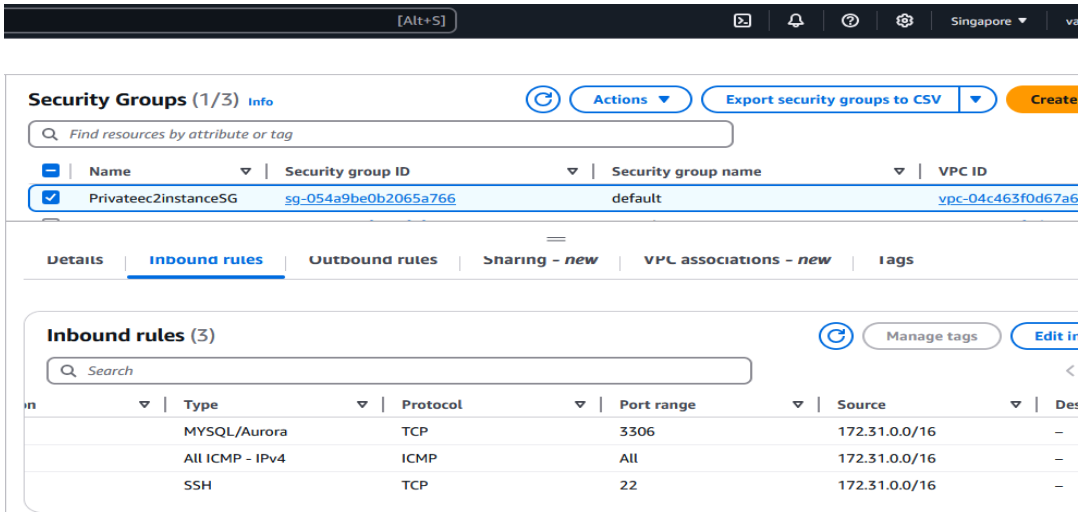




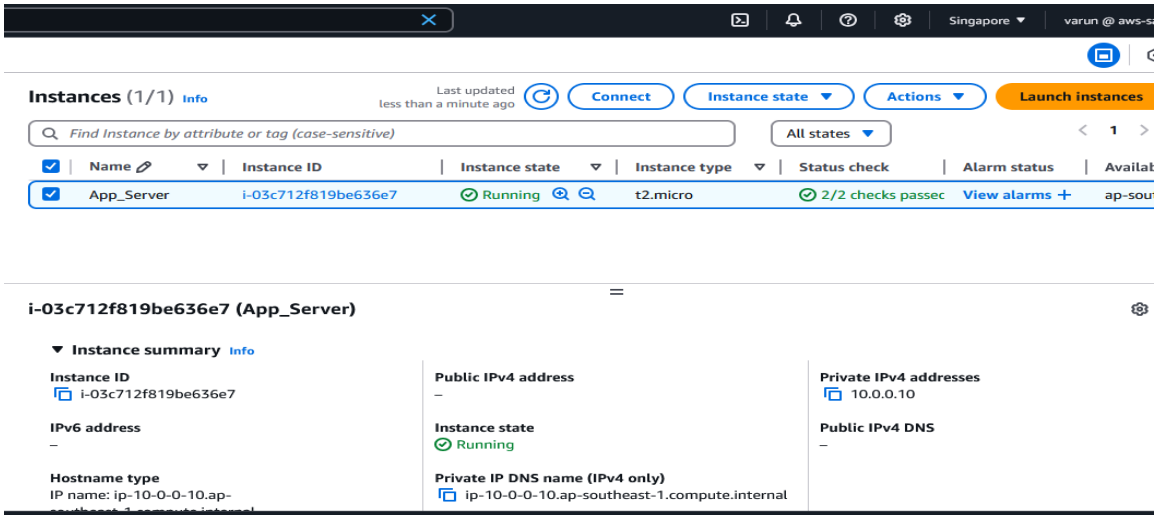




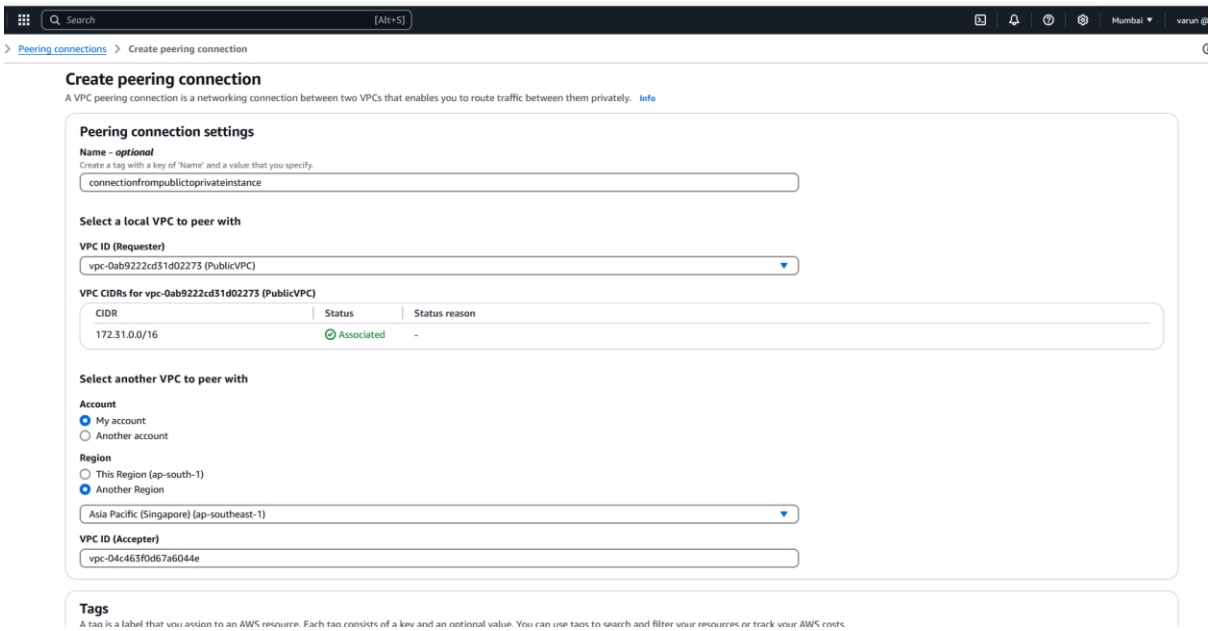
8. Create a security group for the private instance that allows traffic only from the public instance's CIDR block for MySQL, SSH, and ICMP ports.

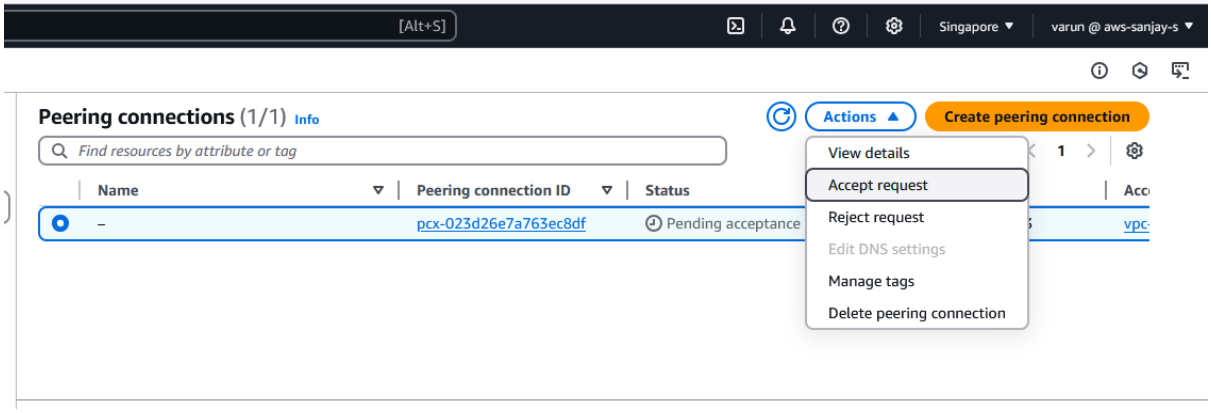


9. Launch an App Server (private instance) under the private VPC to configure RDS and S3 privately.

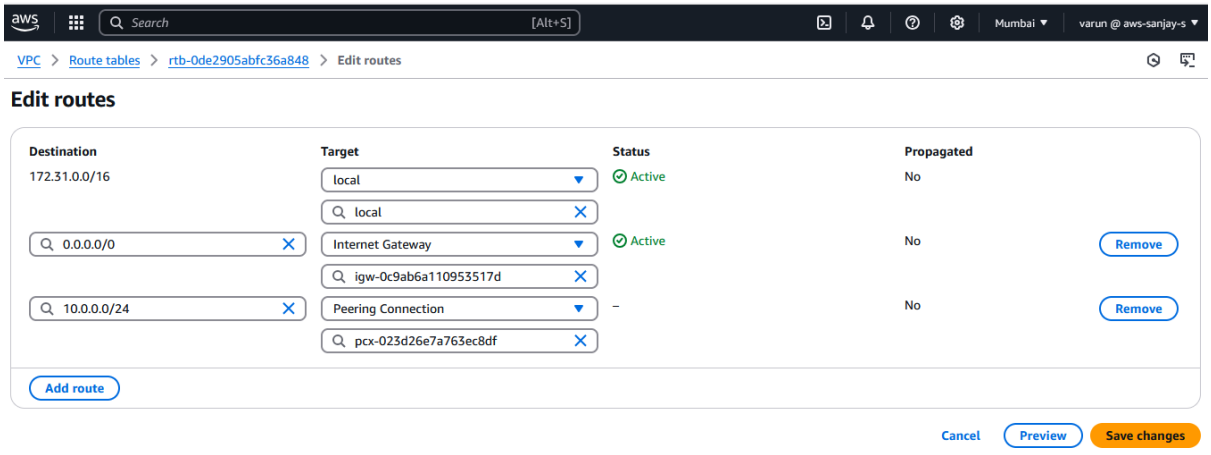


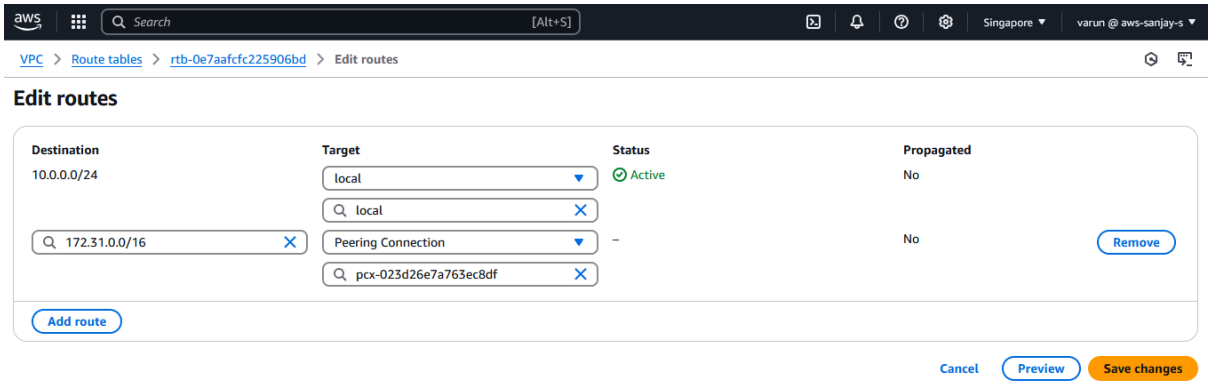
10. Now establish a VPC peering connection between the public instance and the private instance by providing the request and acceptor VPC ID:





11. Update the route table of both instances to allow the peering connection by adding routes with the opposite CIDR block as the destination.





12. After establishing the peering connection, test the connectivity between the public and private instances using ping and ssh with the .pem file.
    - Copy the PEM file of the private instance to the public instance for login access.
    - chmod 400 "singapore-key.pem"
    - ssh -i "singapore-key.pem" ubuntu@10.0.0.10

```
ubuntu@ip-172-31-2-80:~$ ping 10.0.0.10
PING 10.0.0.10 (10.0.0.10) 56(84) bytes of data.
64 bytes from 10.0.0.10: icmp_seq=1 ttl=64 time=62.3 ms
64 bytes from 10.0.0.10: icmp_seq=2 ttl=64 time=62.6 ms
64 bytes from 10.0.0.10: icmp_seq=3 ttl=64 time=62.5 ms
64 bytes from 10.0.0.10: icmp_seq=4 ttl=64 time=63.0 ms
^C
--- 10.0.0.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 62.348/62.638/63.032/0.250 ms
ubuntu@ip-172-31-2-80:~$
```

i-0a9cff391a0905f79 (Web_server)

PublicIPs: 13.203.76.114 PrivateIPs: 172.31.2.80

```
ubuntu@ip-172-31-2-80:~$ sudo -i
root@ip-172-31-2-80:~# vi kep.pem
root@ip-172-31-2-80:~# chmod 400 "kep.pem"
root@ip-172-31-2-80:~# ssh -i kep.pem ubuntu@10.0.0.10
The authenticity of host '10.0.0.10 (10.0.0.10)' can't be established.
ED25519 key fingerprint is SHA256:qbZSG8HVnCsXmswZvcCtktsIJx/dDa5am7Kpxze/NTw.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.0.0.10' (ED25519) to the list of known hosts.
Welcome to Ubuntu 24.04.1 LTS (GNU/Linux 6.8.0-1018-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Sat Dec 28 18:27:42 UTC 2024

  System load:  0.0               Processes:             102
  Usage of /:   24.6% of 6.71GB   Users logged in:       0
  Memory usage: 20%               IPv4 address for enX0: 10.0.0.10
  Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


The list of available updates is more than a week old.
To check for new updates run: sudo apt update


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-10-0-0-10:~$
```

i-0a9cff391a0905f79 (Web_server)

PublicIPs: 13.203.76.114 PrivateIPs: 172.31.2.80

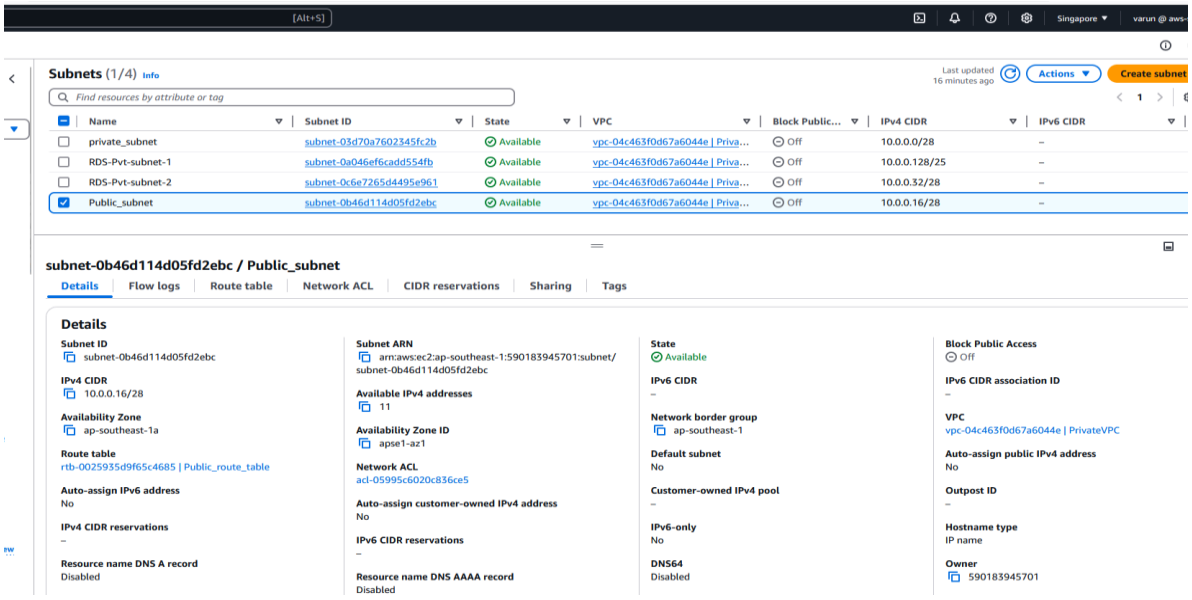
Conclusion on this module:

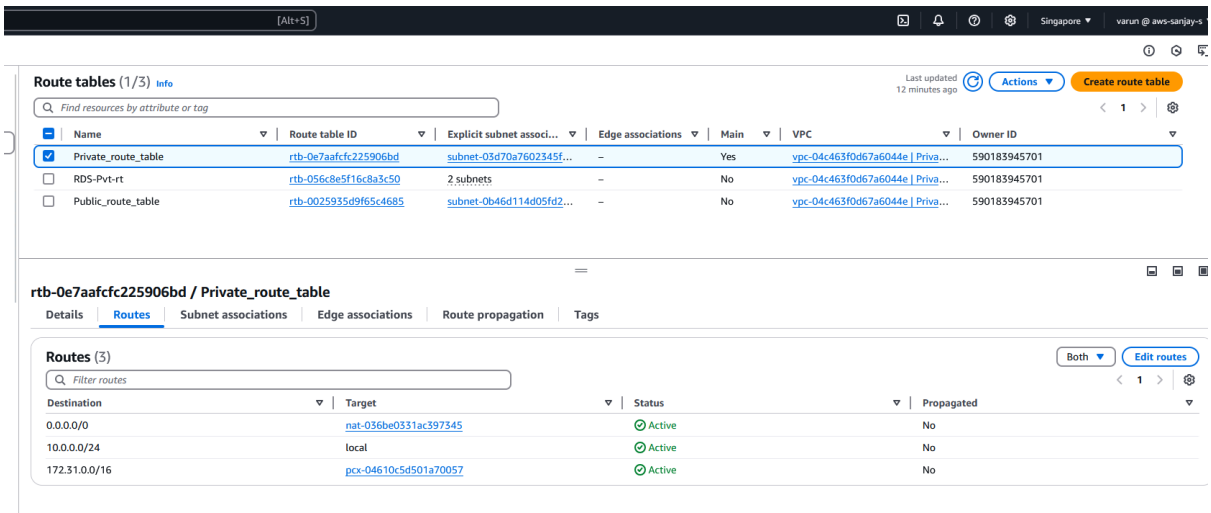
Now we have successfully established a secure connection from the public instance to the private instance in a different region using the VPC peering connection.

Module 2

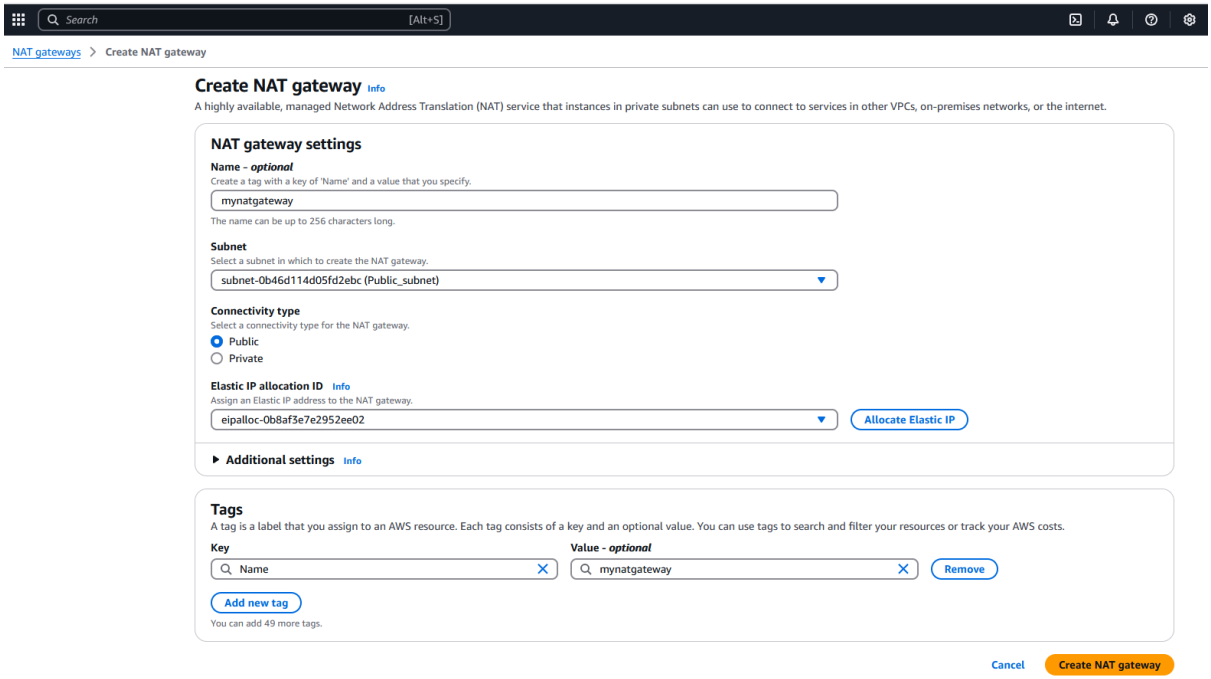
Configuring the NAT gateway to private instance and install the required packages and clone the code from GitHub repository

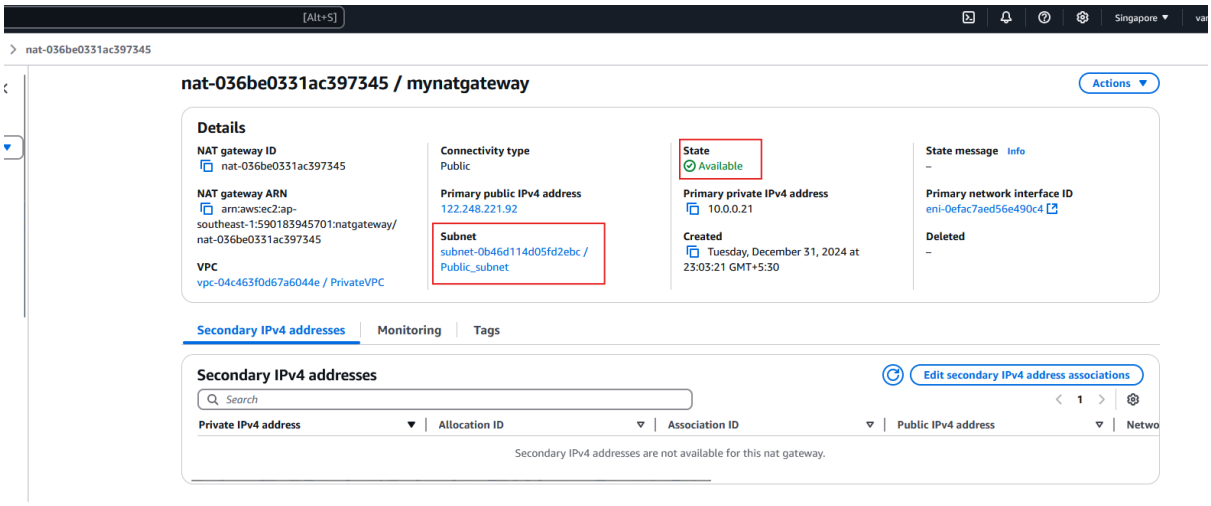
1. Create a public subnet within the VPC that hosts the private instance and update the route table to enable access to the target group through the internet gateway, allowing traffic from any source.

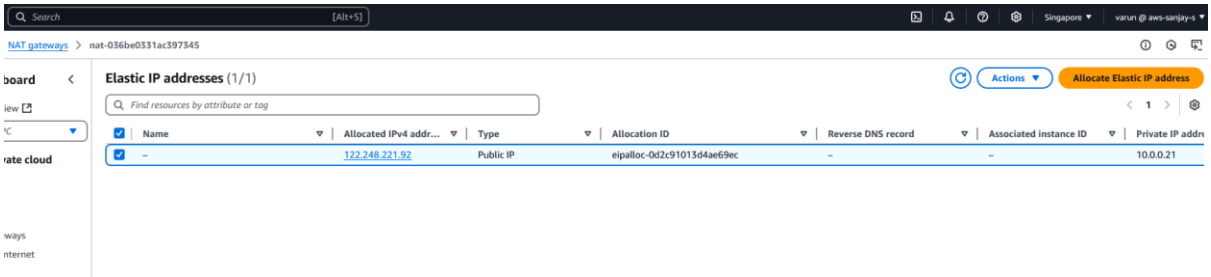




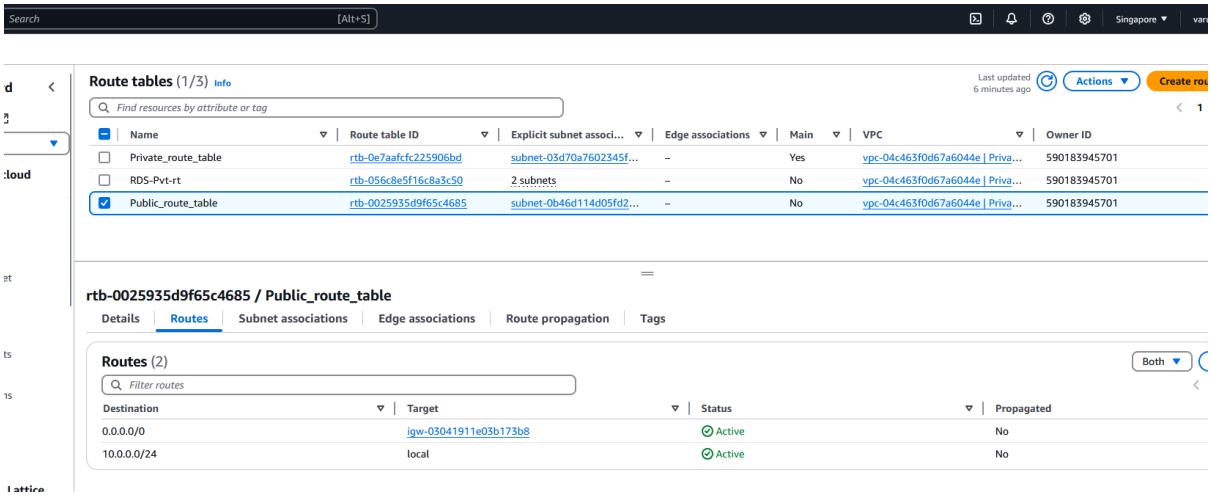
2. Set up a NAT Gateway in the public subnet and associate an Elastic IP with it and Wait for a few minutes until the status of the NAT Gateway changes to "Available."







3. Now, update the route table of the private of target NAT gateway by giving the destination anyway.



Note: For the private instance, configure the security group to allow all traffic (required port) from the public VPC CIDR block and outbound to anyway.

4. Now, on the public instance, install the required packages.
   - sudo apt install upgrade
   - sudo apt install apache2
   - sudo apt install php php-mysqlnd

5. Now, on the private instance, install the required packages.
   - sudo apt install upgrade
   - sudo apt install mariadb-server #For RDS instance configuration.
   - sudo apt install s3fs #For S3 Bucket configuration.

6. clone the code from the GitHub repository using the Git technology:

Git clone https://github.com/varunsimha-MP/Php_CRUD-Basic-.git

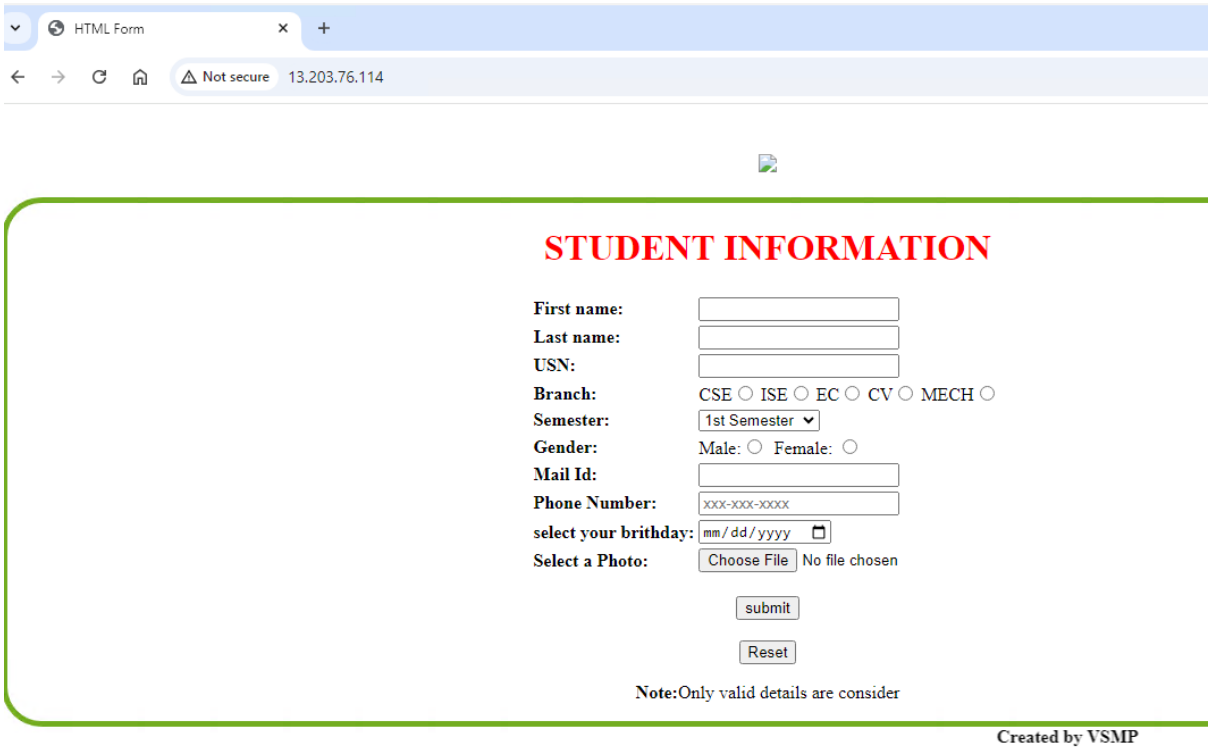
7. On the public instance, set up the web application using Apache2.
   - Transfer the previously cloned code to the /var/www/html directory as shown below.

```
root@ip-172-31-2-80:~# cp -r Php_CRUD-Basic-/php_crud/*.html /var/www//html/
root@ip-172-31-2-80:~# cp -r Php_CRUD-Basic-/php_crud/*.sql /var/www//html/
root@ip-172-31-2-80:~# cp -r Php_CRUD-Basic-/php_crud/*.php /var/www//html/
root@ip-172-31-2-80:~# cp -r Php_CRUD-Basic-/php_crud/*.js /var/www//html/
root@ip-172-31-2-80:~# cd /var/www/html
root@ip-172-31-2-80:/var/www/html# ls
basic.js  dbconnect.php  delete.php  delinfo.php  index.html  php_crud.sql  register.php  update.php  updateuser.php
root@ip-172-31-2-80:/var/www/html#
```

i-0a9cff391a0905f79 (Web_server)

PublicIPs: 13.203.76.114 PrivateIPs: 172.31.2.80

8. After transferring the code, check if the static/index file is working by accessing it through the public instance's public IP address as the URL.



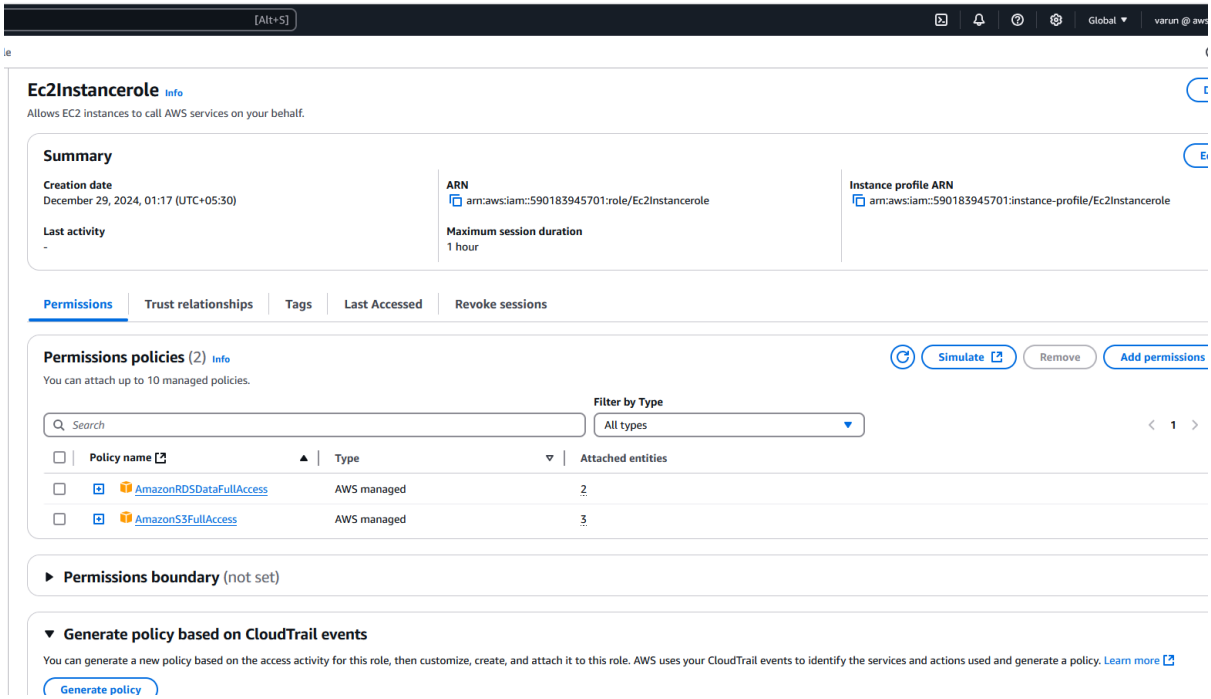
Conclusion of this module:

We have successfully configured the NAT gateway to private instance for secure internet and installed all necessary packages, including Apache2 for the web application, MariaDB for RDS, and the S3 package for mounting the S3 bucket on the respective instances. Additionally, we cloned the code from the GitHub repository using Git and tested the accessibility of the static web application on the public instance.

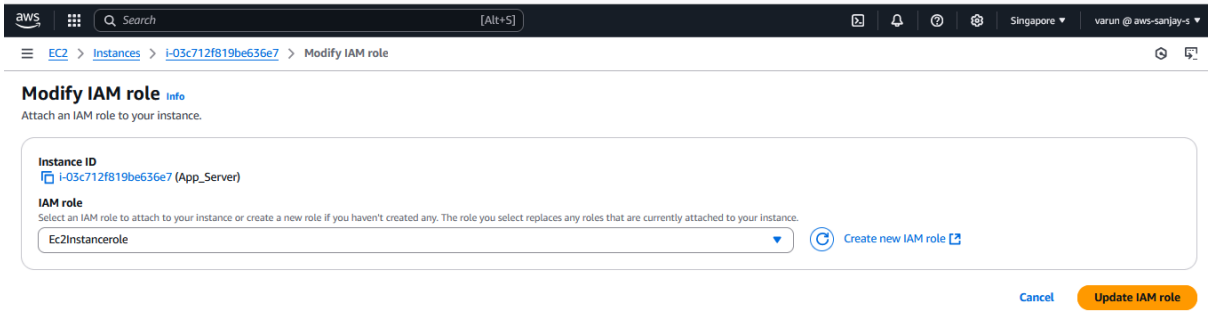
Module - 03

Create an RDS instance, set up the connection to the private instance and configuration accordingly

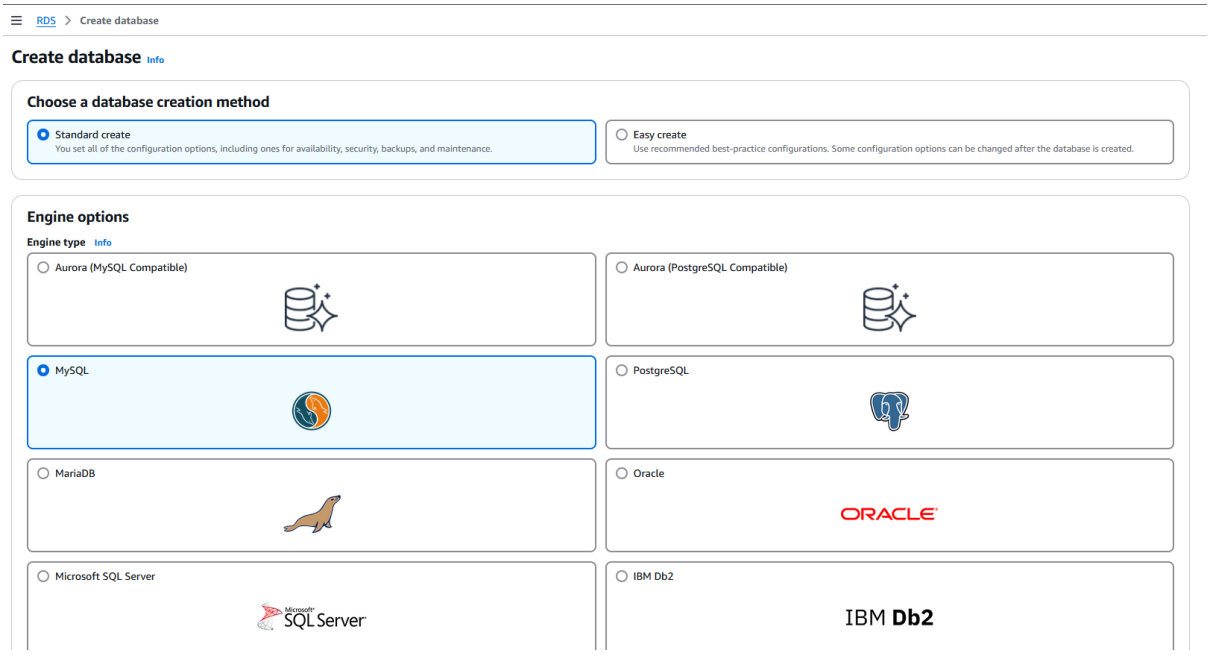
1. Create a role that provides access to the instance by attaching the RDS full access policy.



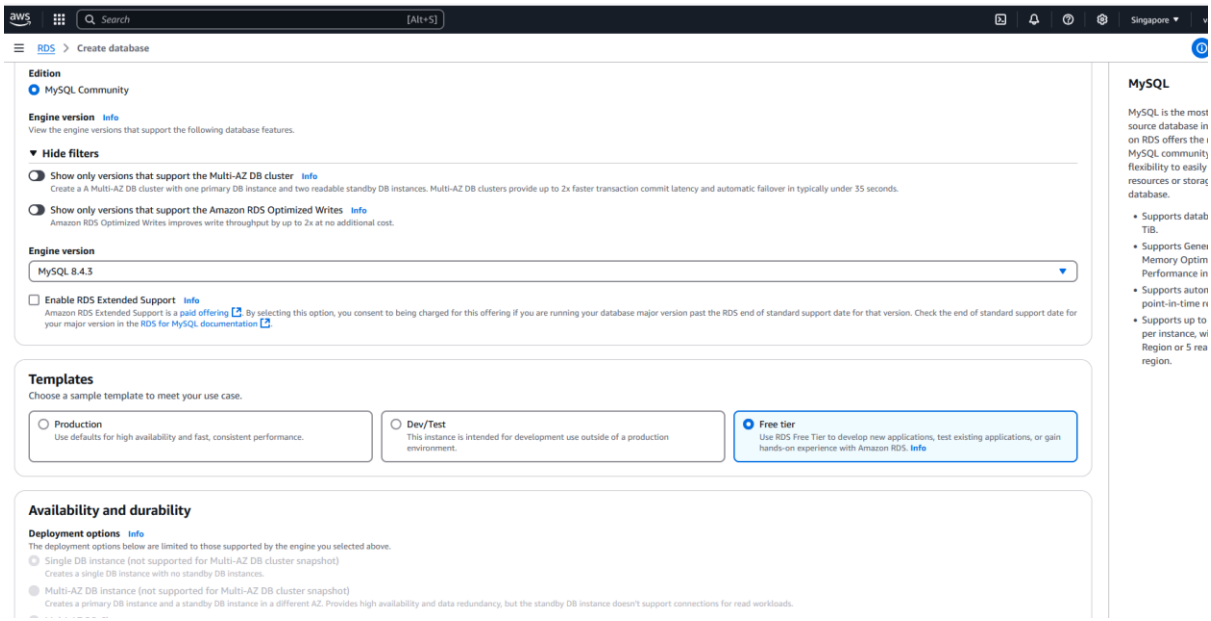
2. Now, attach the RDS full access role to the private instance.



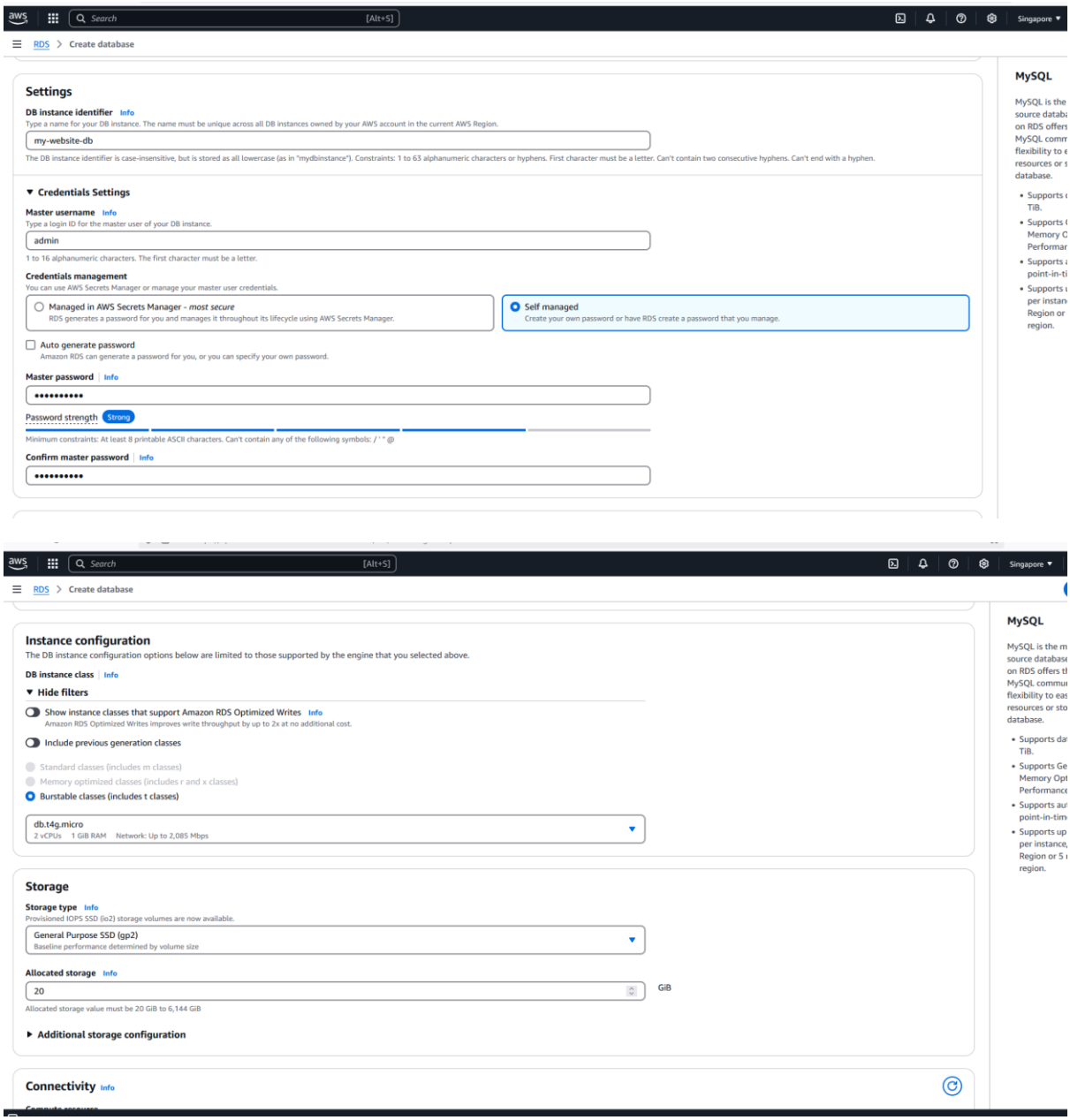
3. Now, create the **RDS database** in the same region as the private instance. Choose the standard method and select "MySQL" as the database engine according to the requirements.



4. Always choose the latest engine version or required for your and select the "Free Tier" template as we are doing for learning purposes.

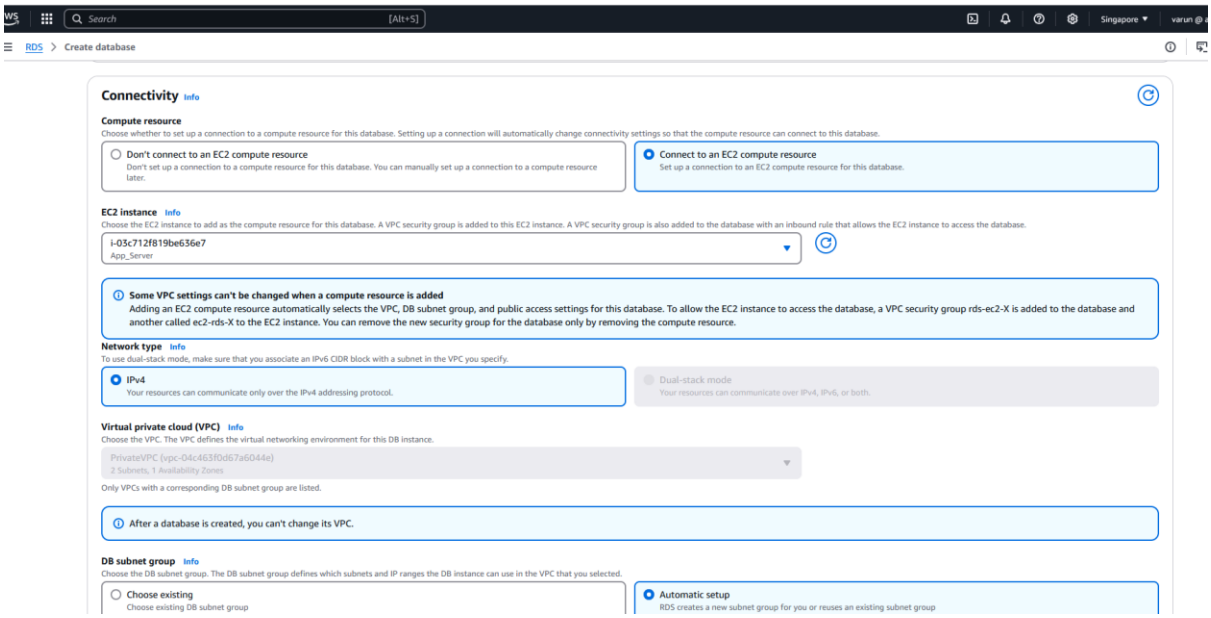


5. Provide the database name and the credentials for logging into the RDS instance. Additionally, configure the CPU, memory, and storage according to the requirements.

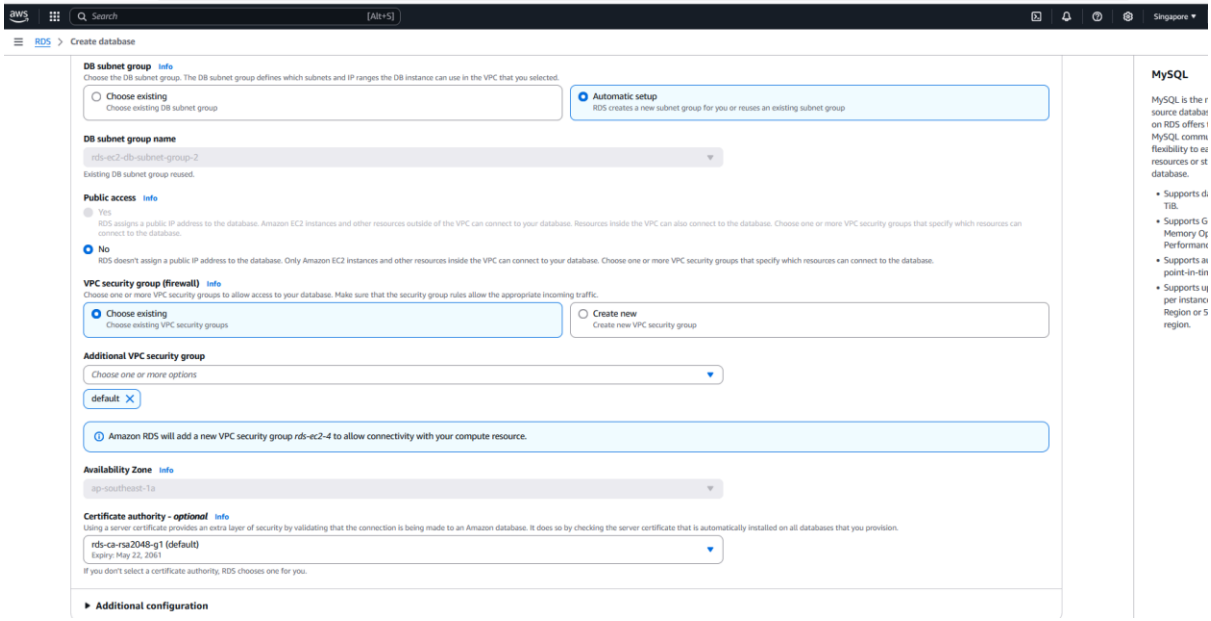


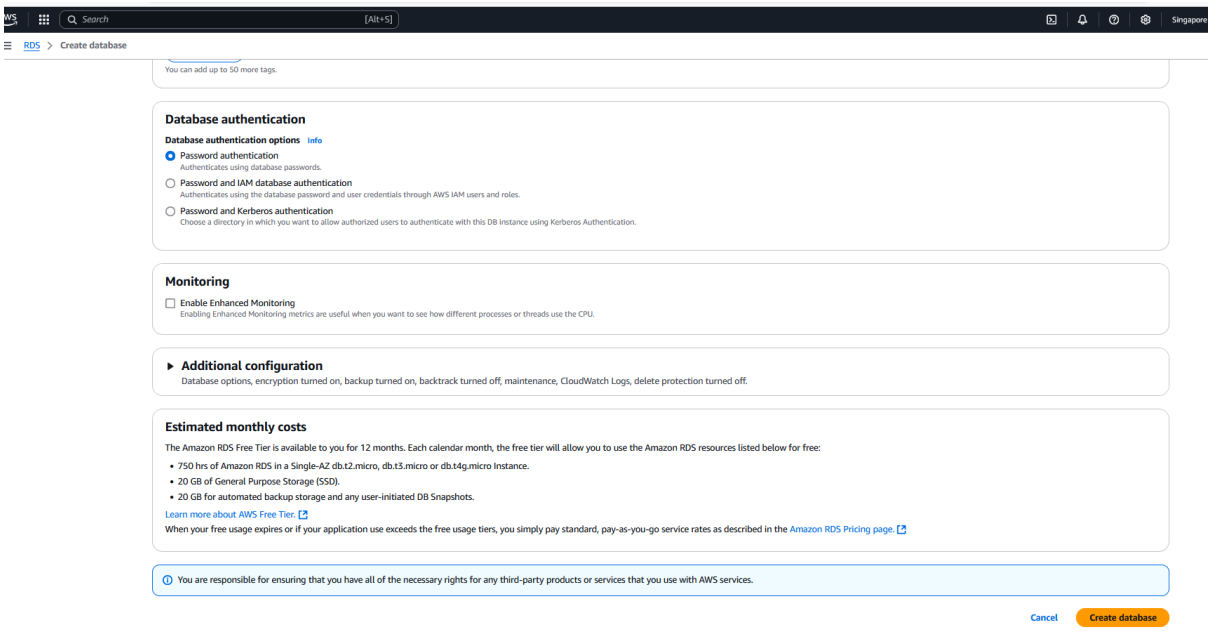
6. For connectivity, choose to set up the connection now and select the private instance.

Note: The connection to the instance can be established after the RDS creation is complete.

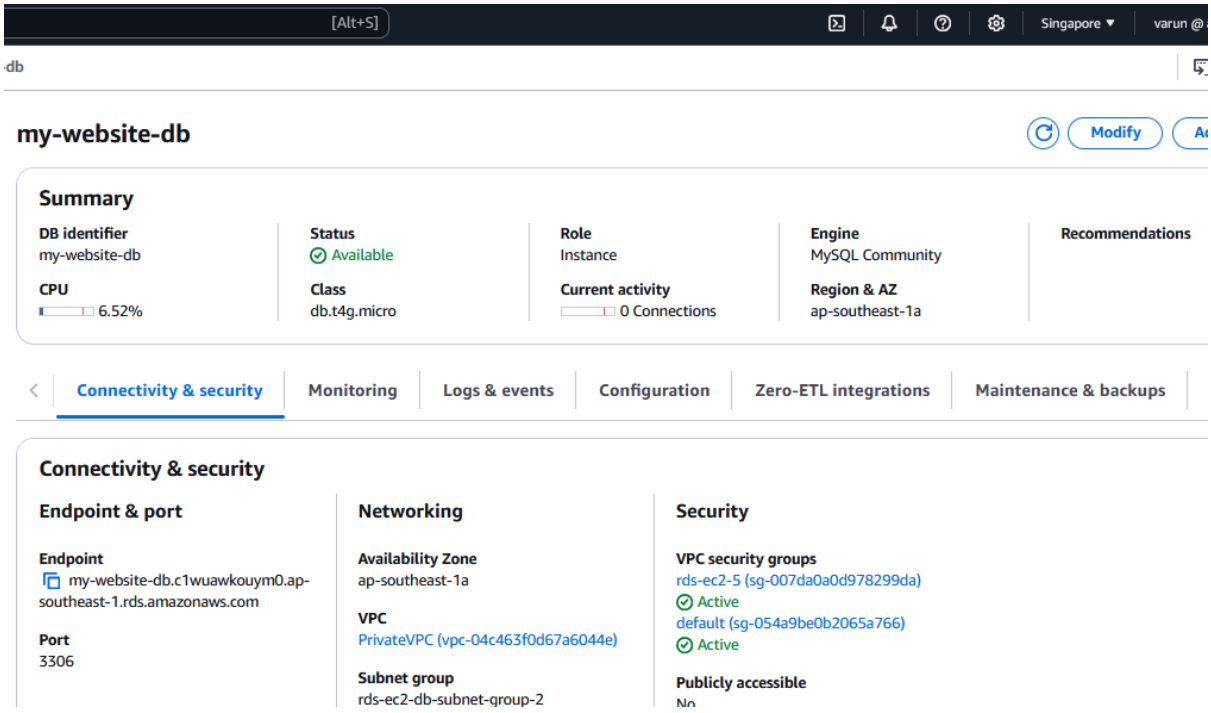


7. Now, select the automatic options and leave the settings as default, then click on "Create."





8. Wait for a few minutes until the RDS status changes to "Available."



9. Now, from the private instance configure MariaDB on the Private Instance:

- Log in to the RDS instance using the RDS endpoint:

  Syntax: mysql -h <RDS_ENDPOINT> -u admin -p

  mysql -h my-website-db.c1wuawkouym0.ap-southeast-1.rds.amazonaws.com-u admin -p

- Enter the password when prompted to access the RDS instance.

- After logging in, create a database and grant full access to the admin user for the newly created database.

  CREATE DATABASE my_db;

  GRANT ALL PRIVILEGES ON my_db.* TO 'admin'@'%';

  FLUSH PRIVILEGES;

  Exit;

```
Last login: Sat Dec 28 18:55:45 2024 from 172.31.2.80
ubuntu@ip-10-0-0-10:~$ sudo -i
root@ip-10-0-0-10:~# mysql -h my-website-db.c1wuawkouym0.ap-southeast-1.rds.amazonaws.com -u admin -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 23
Server version: 8.4.3 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]> create database my_db;
Query OK, 1 row affected (0.038 sec)

MySQL [(none)]> GRANT ALL PRIVILEGES ON my_db.* TO 'admin'@'%';
Query OK, 0 rows affected (0.029 sec)

MySQL [(none)]> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.006 sec)

MySQL [(none)]>
```

i-0a9cff391a0905f79 (Web_server)

PublicIPs: 13.203.76.114 PrivateIPs: 172.31.2.80

10. Use the code transferred earlier from the public to the private instance, now import the SQL file into the RDS database:

    mysql -h <RDS_ENDPOINT> -u admin -p my_db < /tmp/php_crud.sql

```
MySQL [(none)]> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.006 sec)

MySQL [(none)]> exit;
Bye
root@ip-10-0-0-10:~# mysql -h my-website-db.c1wuawkouym0.ap-southeast-1.rds.amazonaws.com -u admin -p my_db < /tmp/php_crud.sql
-bash: /tmp/php_crud.sql: No such file or directory
root@ip-10-0-0-10:~# mysql -h my-website-db.c1wuawkouym0.ap-southeast-1.rds.amazonaws.com -u admin -p my_db < /tmp/Php_CRUD-Basic-/php_crud/php_crud.sql
root@ip-10-0-0-10:~# mysql -h my-website-db.c1wuawkouym0.ap-southeast-1.rds.amazonaws.com -u admin -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 26
Server version: 8.4.3 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]> use my_db;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MySQL [my_db]> show tables;
+------------------+
| Tables_in_my_db  |
+------------------+
| personalinfo     |
+------------------+
1 row in set (0.001 sec)

MySQL [my_db]>
```

i-0a9cff391a0905f79 (Web_server)

PublicIPs: 13.203.76.114 PrivateIPs: 172.31.2.80

11. Now, go to the public instance and update the database connection file.

    {

```
    "db_host": "<RDS_ENDPOINT>",
    "db_user": "admin",
    "db_password": "<Your_Password>",
    "db_name": "my_db"
}
```

12. After updating the database connection file, test the database connectivity on the public instance by running:

    php dbconnect.php

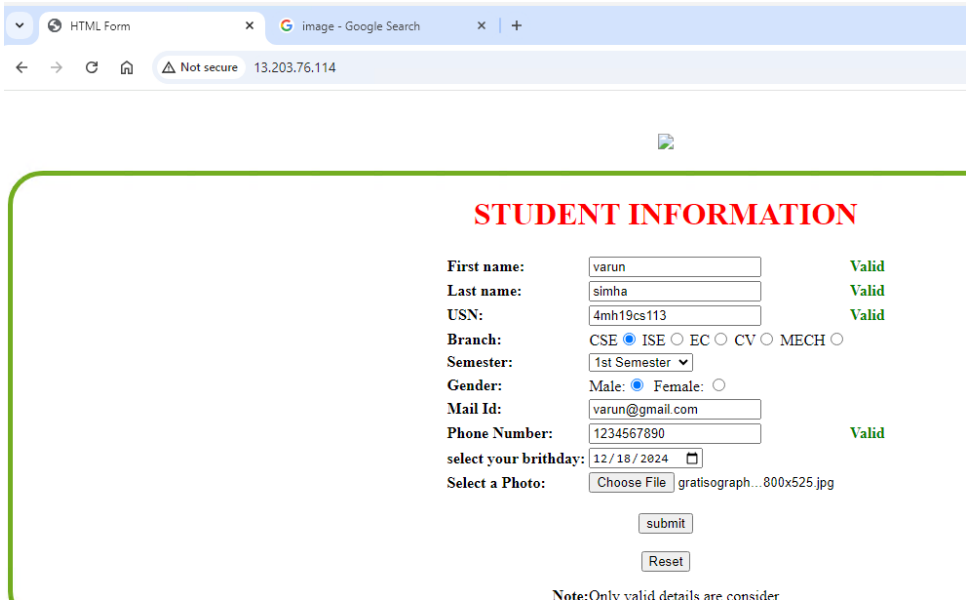
Note: If no error occurs, the connection is working fine.

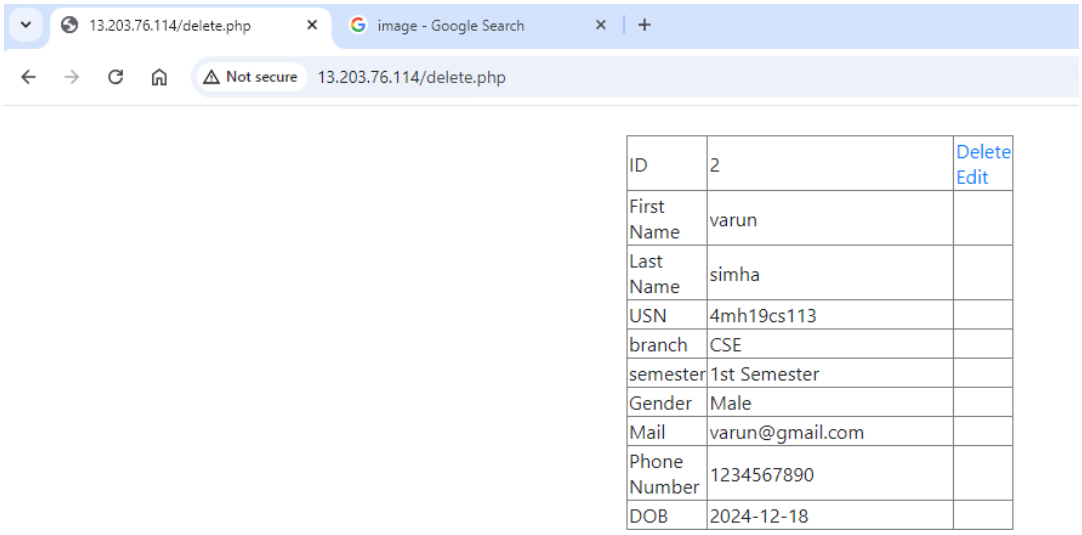
```
root@ip-172-31-2-80:~# cd /var/www/html/
root@ip-172-31-2-80:/var/www/html# ls
basic.js  dbconnect.php  delete.php  delinfo.php  index.html  php_crud.sql  register.php  update.php  updateuser.php
root@ip-172-31-2-80:/var/www/html# vi dbconnect.php
root@ip-172-31-2-80:/var/www/html# cat dbconnect.php
<?php
$servername = "my-website-db.c1wuawkouym0.ap-southeast-1.rds.amazonaws.com";
$username = "admin";
$password = "SimhaVarun";
$dbname = "my_db";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
?>
root@ip-172-31-2-80:/var/www/html# php dbconnect.php
root@ip-172-31-2-80:/var/www/html#
```

i-0a9cff391a0905f79 (Web_server)

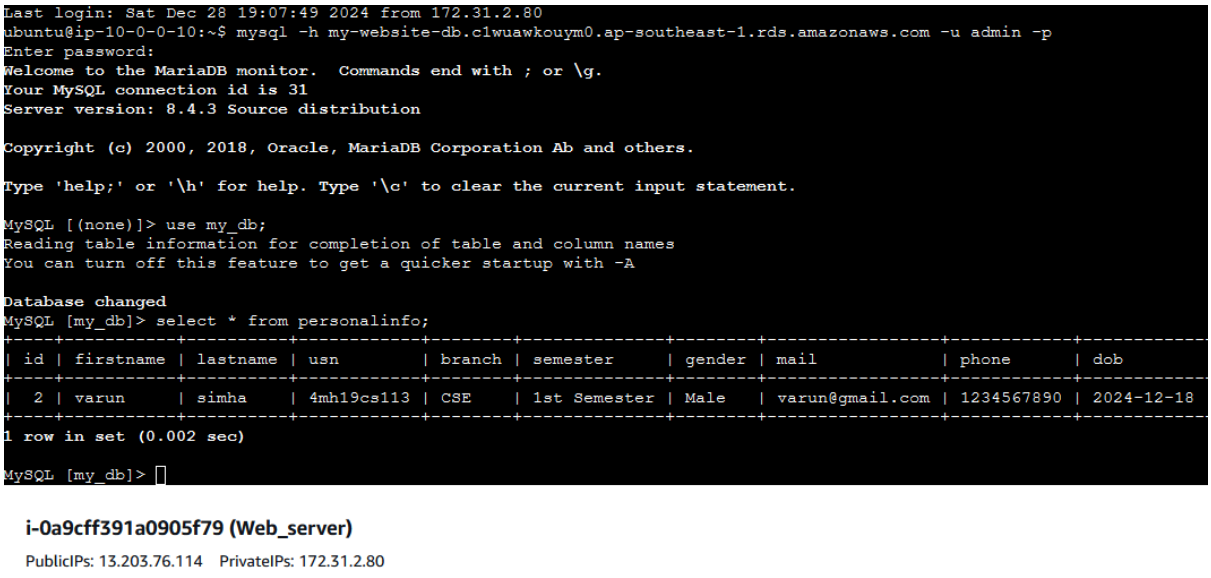
PublicIPs: 13.203.76.114 PrivateIPs: 172.31.2.80

13. To test if the database web application is working, use the public instance's IP address as the URL. Fill out the form and submit it to verify the functionality.





14. Let's check from the instance's end: the changes are reflecting, and everything is working as expected.



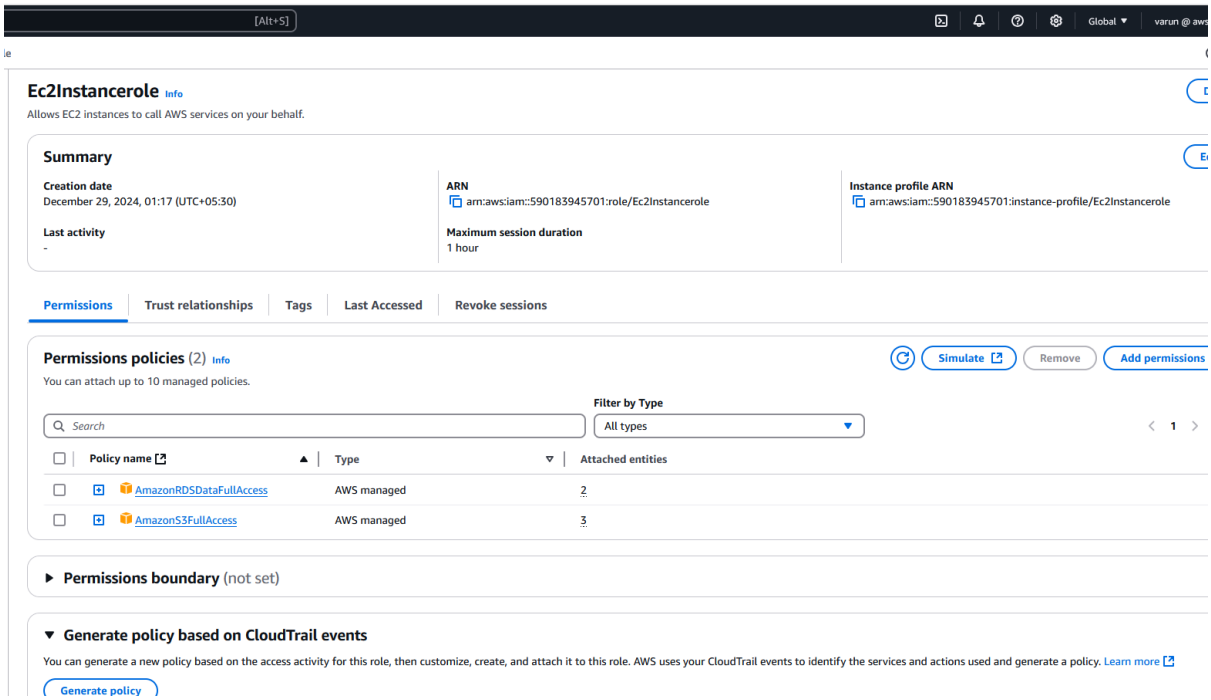
Conclusion for this module:

We have successfully created an RDS instance and configured it with the private instance according to our code requirements. The SQL file was imported into the RDS, and the connection was established from the public instance by updating the dbconnect.php file. After testing, we confirmed that the public instance's IP address is working as expected, and the changes are reflected in the RDS.

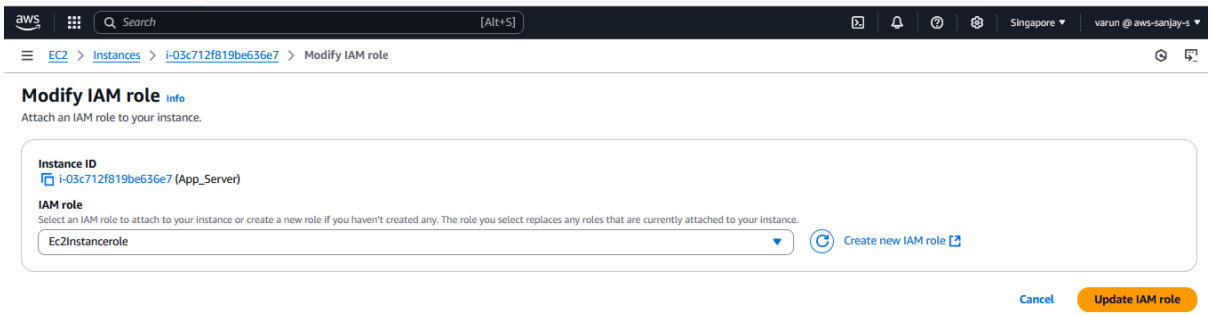
Module - 04

Configure the S3 bucket for the private instance using a VPC endpoint

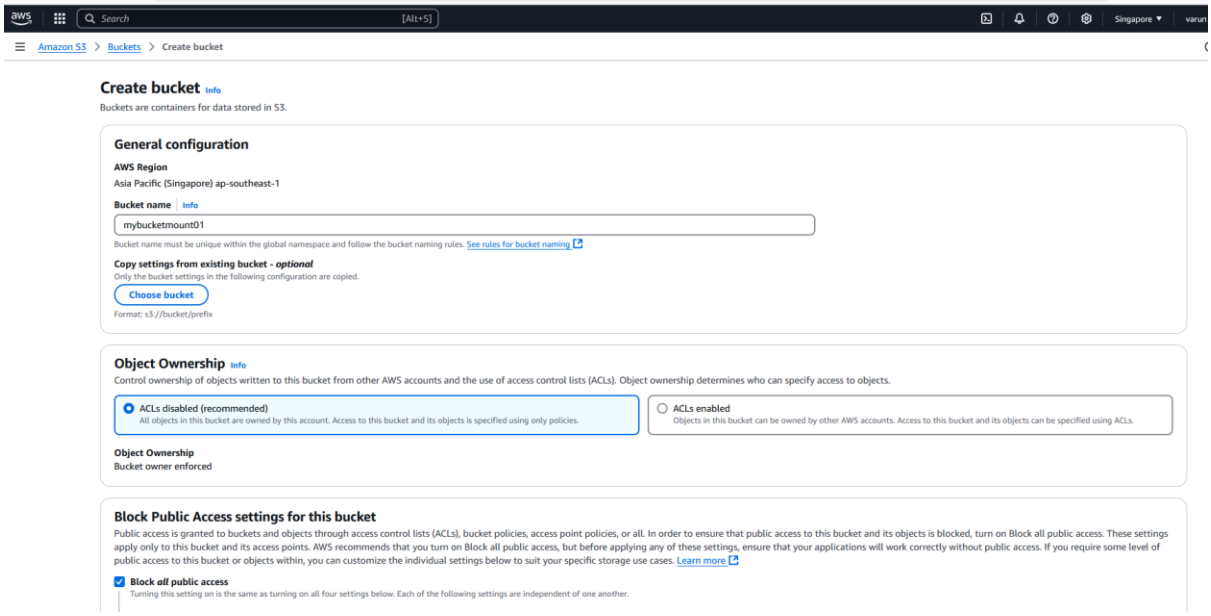
1. Create a role that grants access to the instance by attaching the "AmazonS3FullAccess" policy.

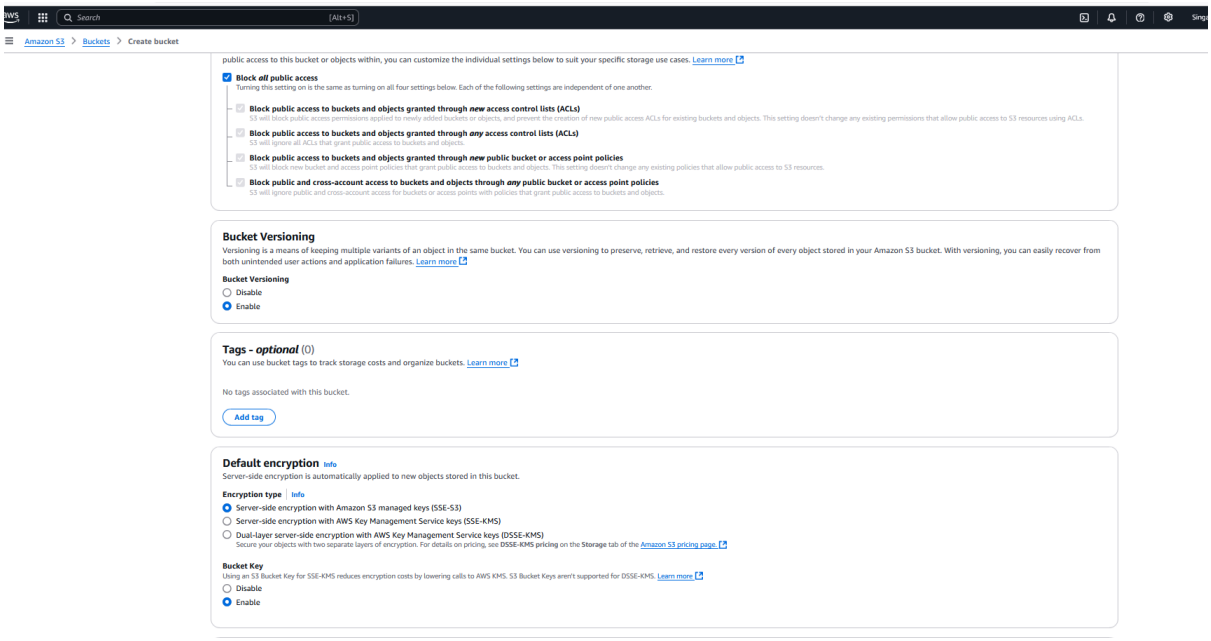


2. Now, attach the "S3 Full Access" role to the private instance.

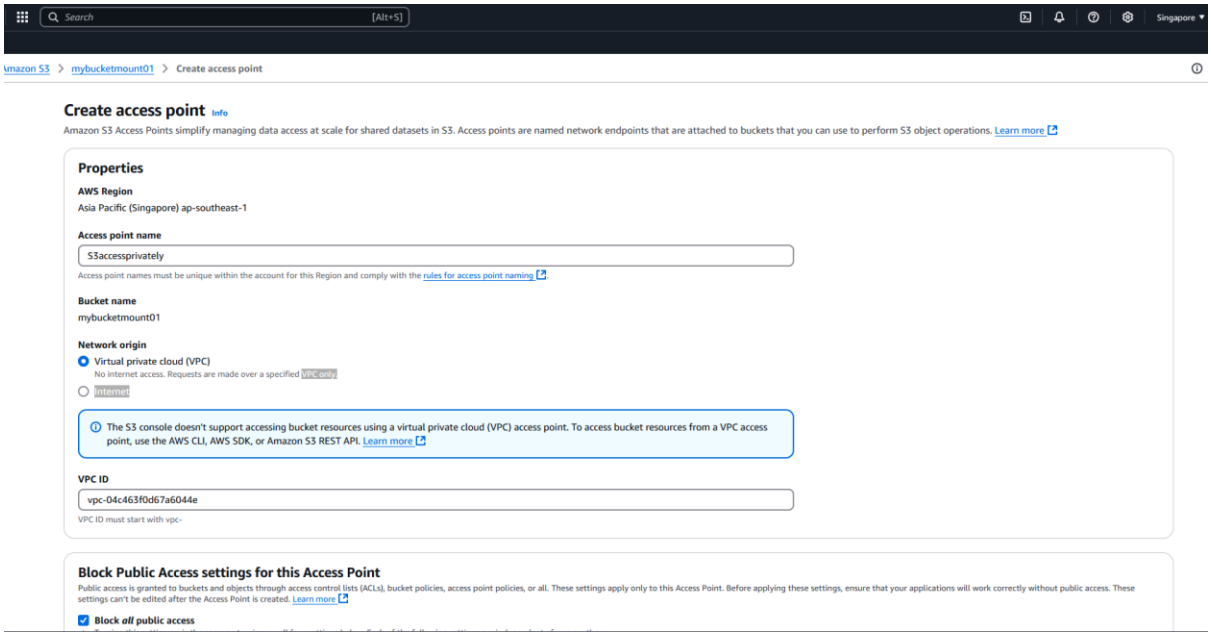


3. Now, create an S3 bucket and block all public access to it.

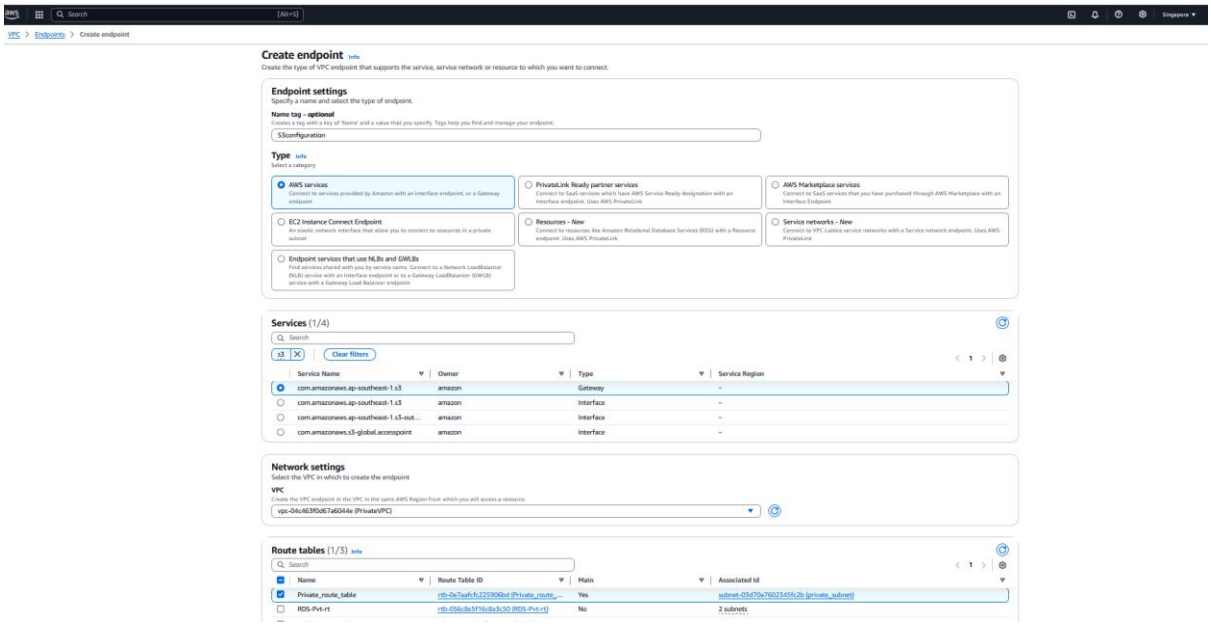


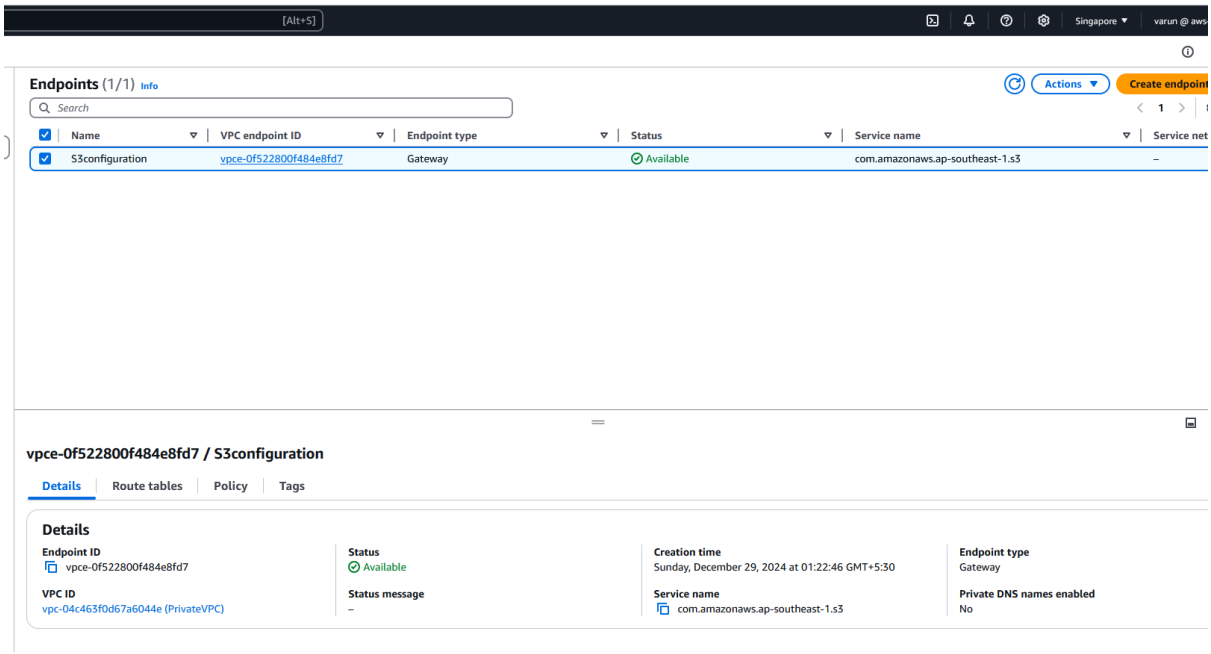


4. Navigate to the bucket's access points, create a new access point by providing a name, selecting the private VPC, and ensuring that public access is blocked.

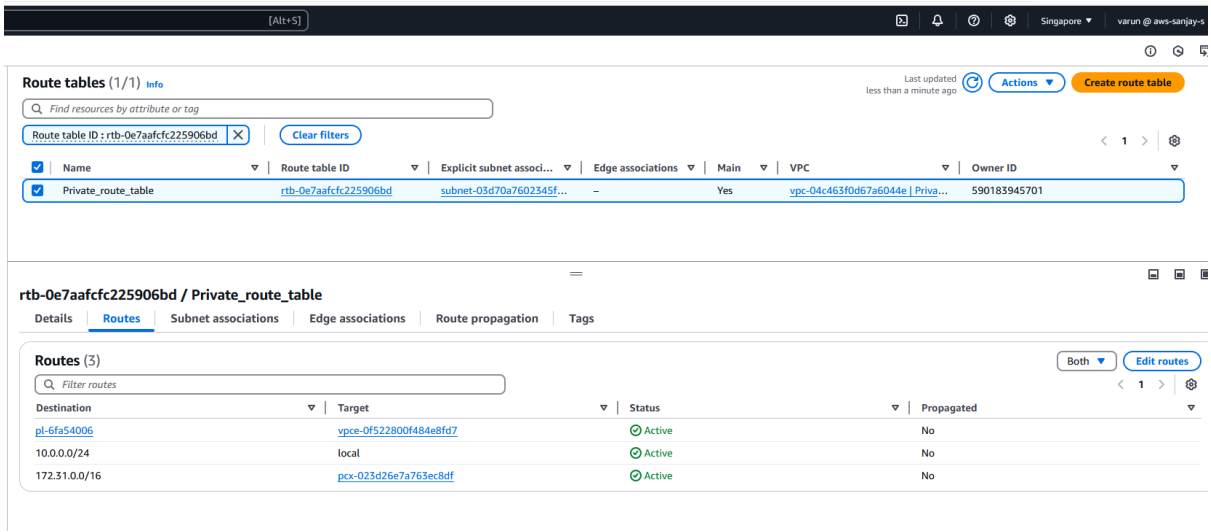


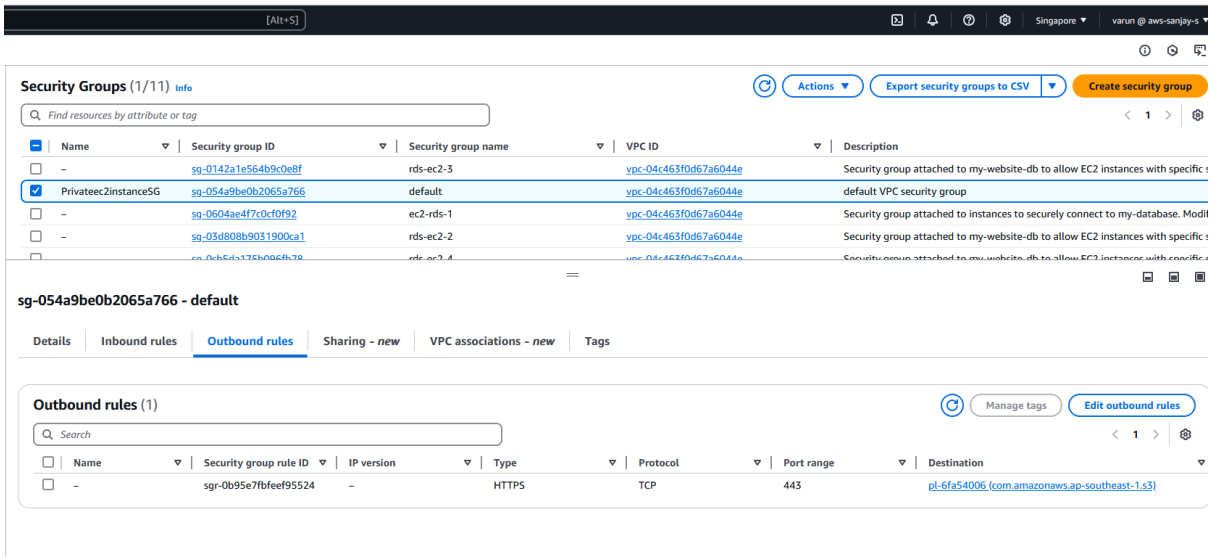
5. Create a VPC endpoint with an S3 gateway and attach it to the private instance's route table.





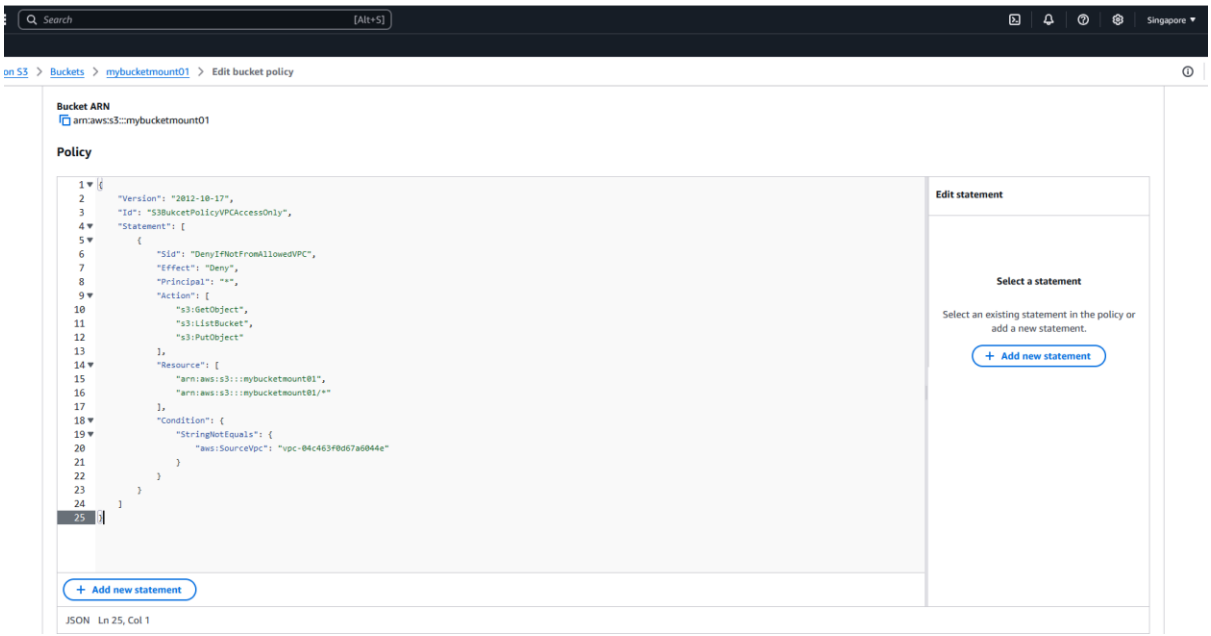
6. Wait for a few minutes until the VPC endpoint is automatically updated in the attached route table and the outbound rule of the private instance's security group.





7. Goto Edit bucket policy give the below code which will Locking down Amazon S3 bucket to VPC access only.

Note: Outside VPC even from the console we can't access it.



8. Now, from the private instance, create a mount point directory:

- sudo mkdir -p /mnt/s3bucket

9. Mounting the S3 bucket Using IAM Role:

Note: Earlier, we have installed the s3fs package on the private instance.

- Mount the S3 bucket using `s3fs`:

Syntax: s3fs <bucket-name> /mnt/s3bucket -o iam_role=auto -o url=https://s3.<region>.amazonaws.com -o use_path_request_style

s3fs mybucketmount01 /mnt/s3bucket -o iam_role=auto -o url=https://s3.ap-southeast-1.amazonaws.com -o use_path_request_style

Replace:

- `<bucket-name>` with your S3 bucket name.

- `<region>` with your AWS region (e.g., `us-east-1`).

Options Explained:

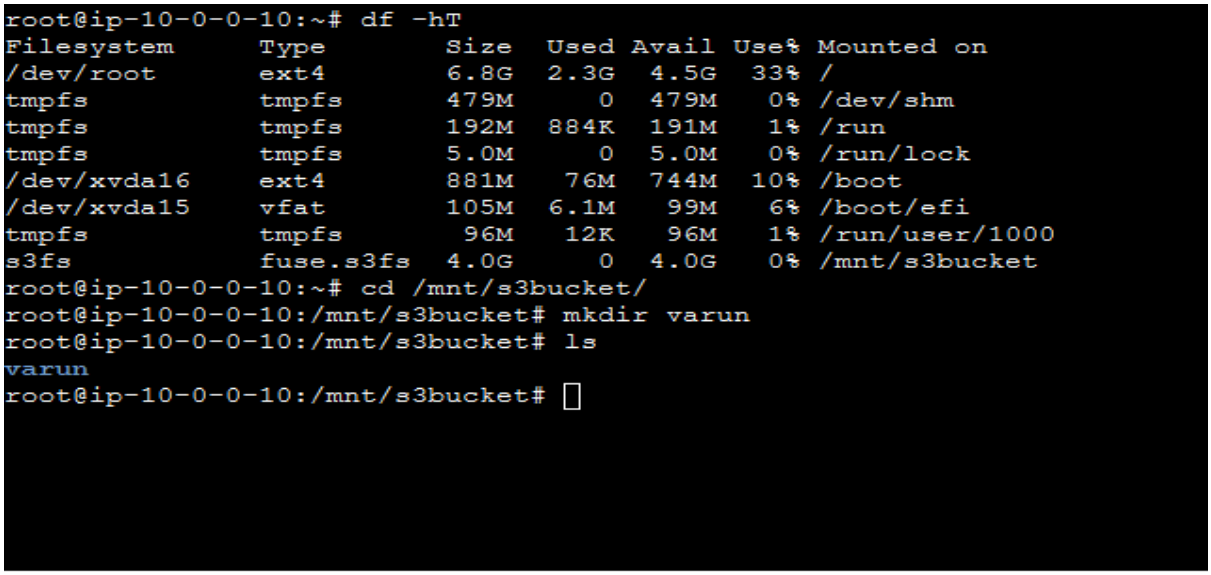
- `-o iam_role=auto`: Automatically retrieves credentials from the attached IAM role.

- `-o url`: Directs requests to the S3 service in the specified region.

- `-o use_path_request_style`: Enables path-style access, necessary for certain configurations.

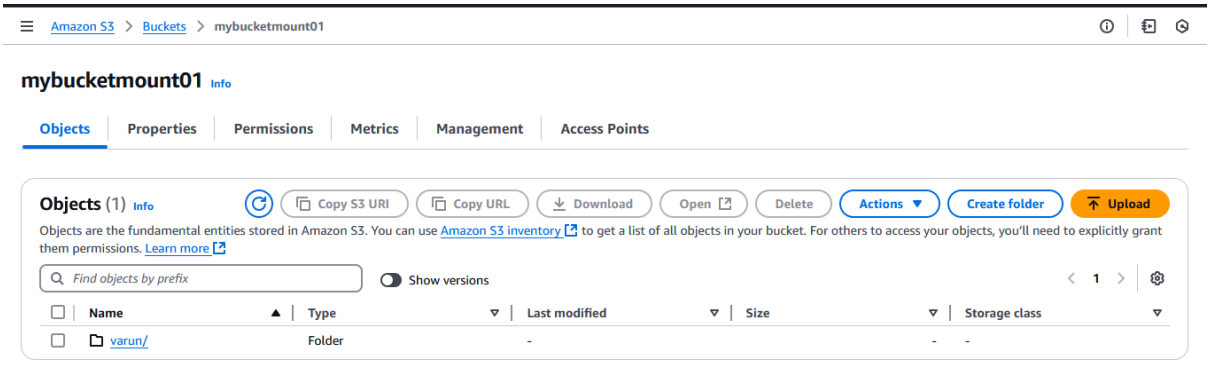
10. Test the mount point before making it persistent.

- Check the file system - df -hT
- Create a file on the instance and check if it is reflected in the S3 bucket.



i-0a9cff391a0905f79 (Web_server)

PublicIPs: 13.203.76.114 PrivateIPs: 172.31.2.80



11. Persist the mount in the “/etc/fstat” file:
    - Edit the file: sudo vi /etc/fstab
    - Add the following entry:

      s3fs#<bucket-name> /mnt/s3bucket fuse _netdev,iam_role=auto,allow_other,use_path_request_style,url=https://s3.<region>.amazonaws.com 0 0

Options Explained:

- `_netdev`: Ensures the mount is dependent on network services.

- `allow_other`: Allows access to users other than the owner.

12. Save the file and test:
    - sudo mount -a

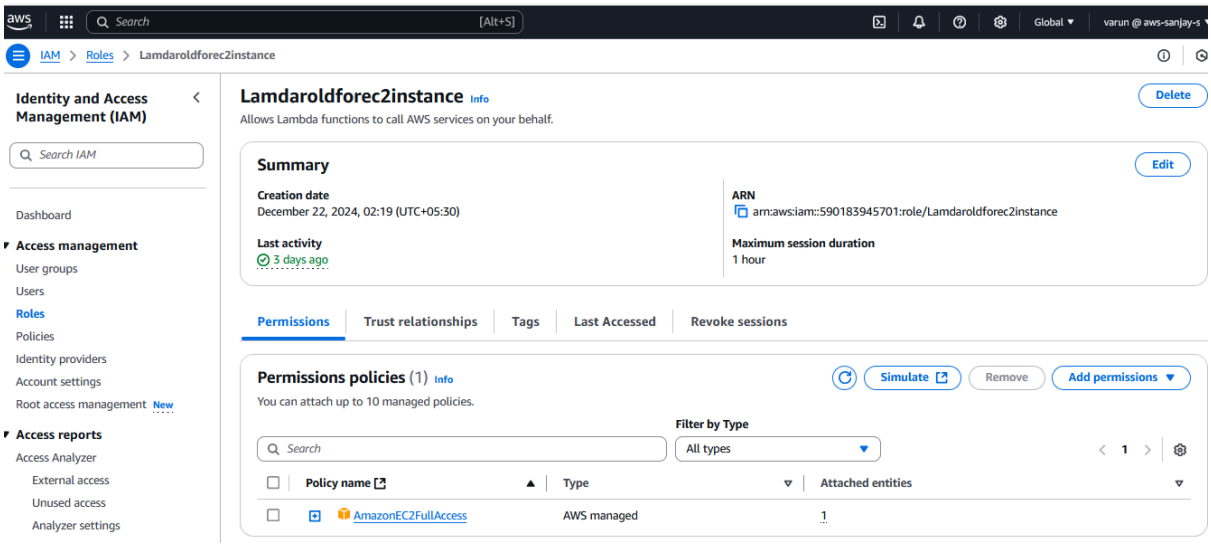
Conclusion of this module:

We have successfully configured the S3 bucket on the private instance using the VPC endpoint, ensuring that data is transferred securely without exposure to the public internet.

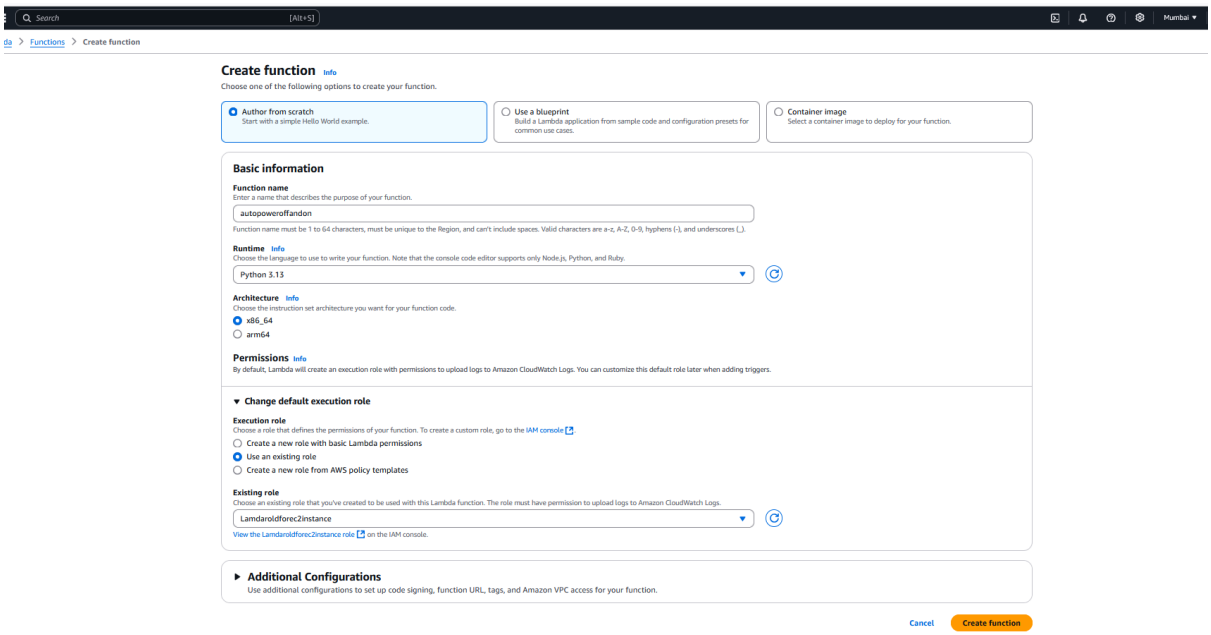
Module – 05

Configure automatic power off and on for an EC2 instance using AWS Lambda and Event Bridge

1. Create a role that grants access to the Lambda service by attaching the "EC2 Full Access" policy.



2. Now, create a Lambda function by selecting the runtime (type of language) and architecture, and then assign the "EC2 Full Access" role to it.



3. After creation, provide the following Python code to execute:

Note: The code below will toggle the instance power off/on based on the event-driven time.

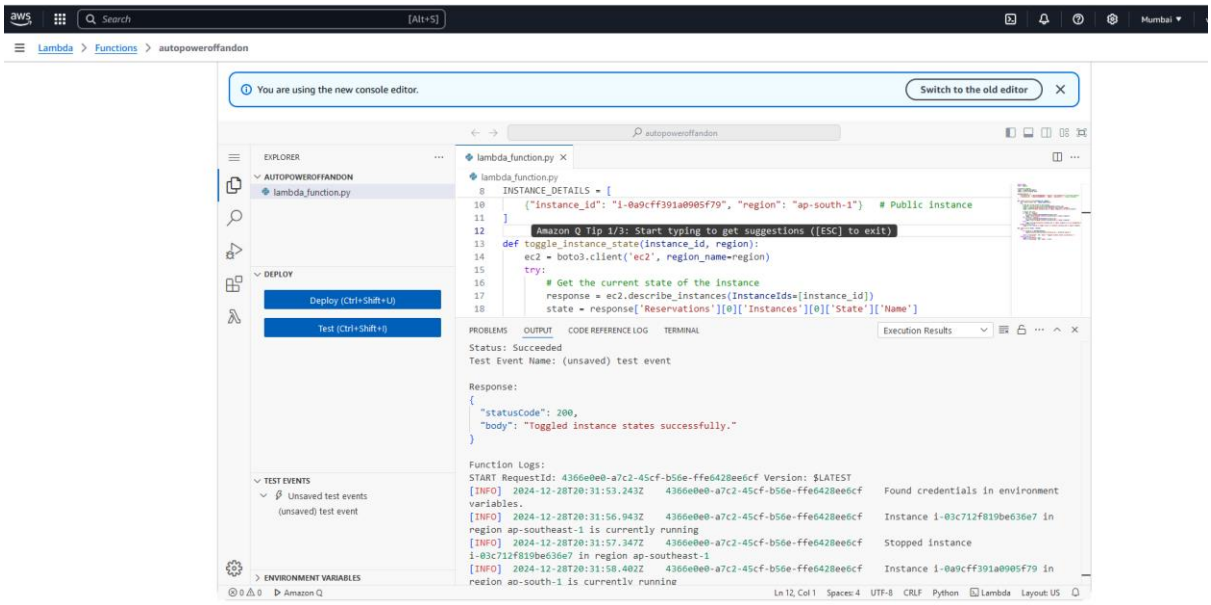
```
import boto3
import logging

# Configure logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

INSTANCE_DETAILS = [
  {"instance_id": "i-0ddde86ee005cbab5", "region": "ap-southeast-1"}, # Private instance
  {"instance_id": "i-0e82be1a040318011", "region": "ap-south-1"}  # Public instance
]

def toggle_instance_state(instance_id, region):
  ec2 = boto3.client('ec2', region_name=region)
  try:
    # Get the current state of the instance
    response = ec2.describe_instances(InstanceIds=[instance_id])
    state = response['Reservations'][0]['Instances'][0]['State']['Name']
    logger.info(f"Instance {instance_id} in region {region} is currently {state}")

    # Toggle the state
    if state == "stopped":
      ec2.start_instances(InstanceIds=[instance_id])
      logger.info(f"Started instance {instance_id} in region {region}")
```

```
        elif state == "running":
            ec2.stop_instances(InstanceIds=[instance_id])
            logger.info(f"Stopped instance {instance_id} in region {region}")
        else:
            logger.warning(f"Instance {instance_id} in region {region} is in an unsupported state: {state}")
    except Exception as e:
        logger.error(f"Failed to toggle state for instance {instance_id} in region {region}: {e}")

def lambda_handler(event, context):
    try:
        for instance in INSTANCE_DETAILS:
            toggle_instance_state(instance["instance_id"], instance["region"])

        return {"statusCode": 200, "body": "Toggled instance states successfully."}
    except Exception as e:
        logger.error(f"Error: {e}")
        return {"statusCode": 500, "body": str(e)}
```

Note: Edit the region and instance ID in the code according to the specific requirements.

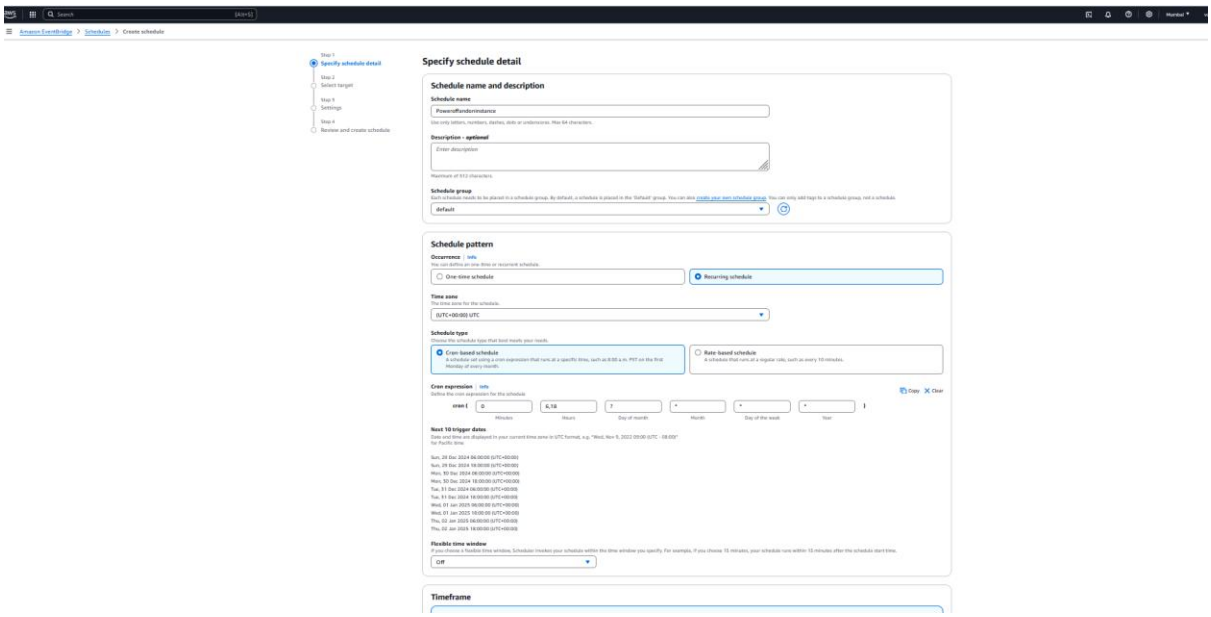
---

4. Now, click on **Deploy** to deploy the code. Then, change the configuration time to 10 seconds for the Lambda function and test the code.

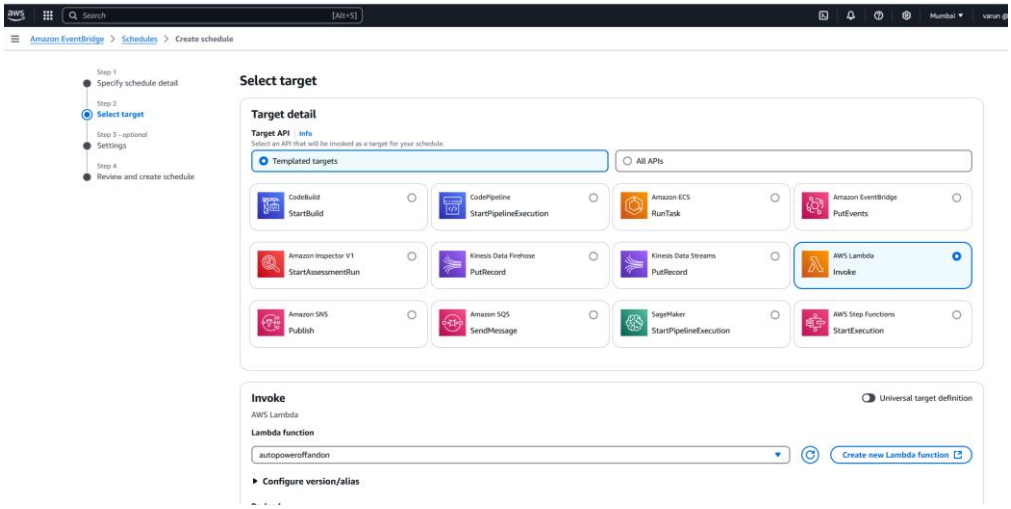


5. Create an Event Bridge Scheduler with the following configuration:
   - Set the schedule to recur daily.
   - Set the time zone.
   - Use the cron expression: (0 6,18 ? * * *).
   - Turn off the flexible time window.

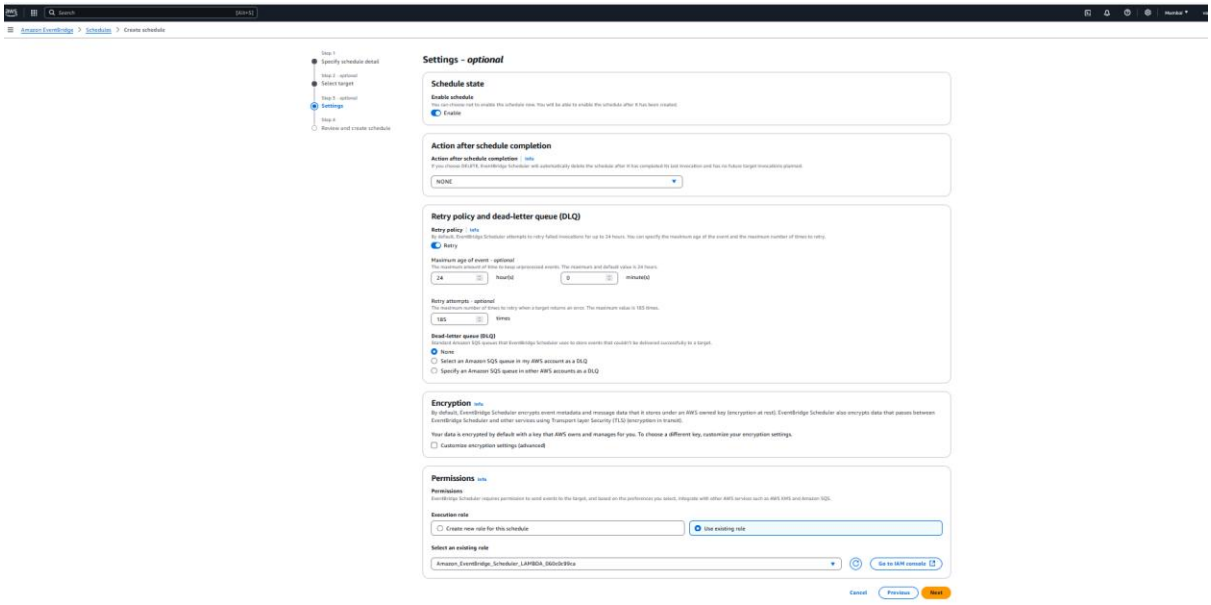
Note: The cron expression will trigger daily at 6 AM and 6 PM.

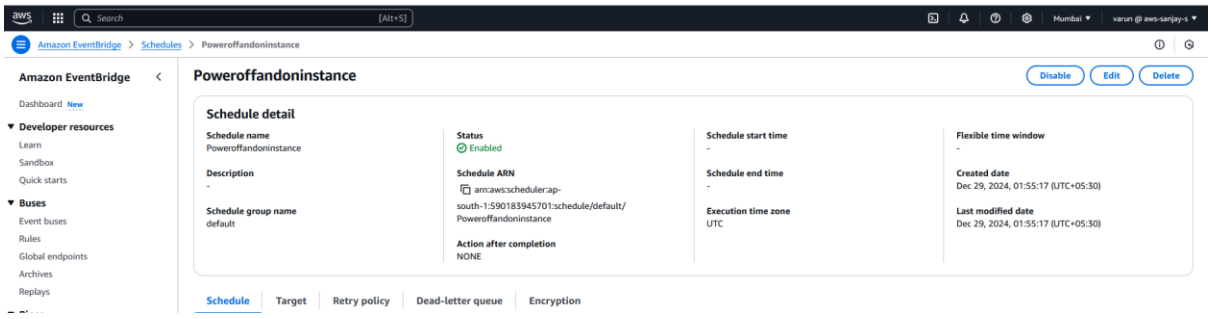


6. After specifying the schedule, select the target as **Lambda Function**.



7. Then, configure the basic settings and click on **Create**.





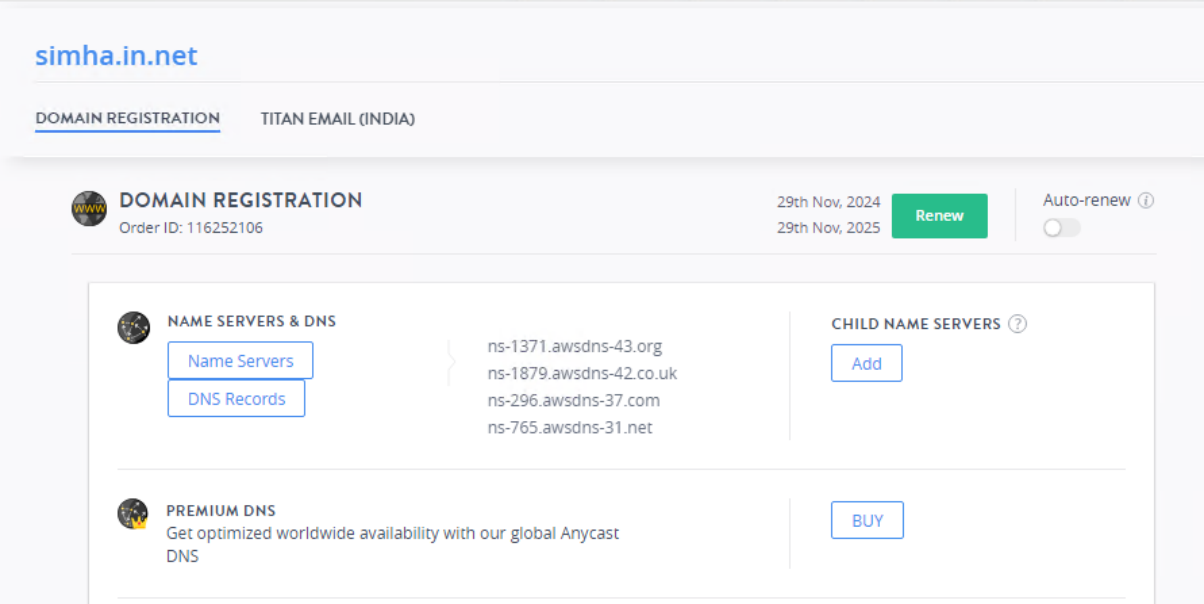
Conclusion of this module:

We have successfully set up automation that will power off and power on the instance at 6 AM and 6 PM UTC daily, triggered by the Lambda function through the Event Bridge scheduler, helping to reduce resource costs.

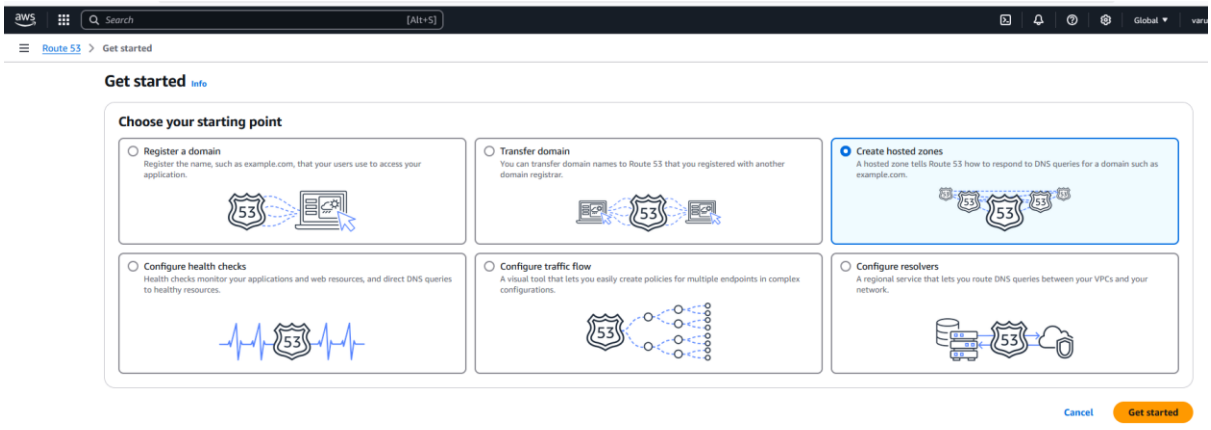
Module – 06

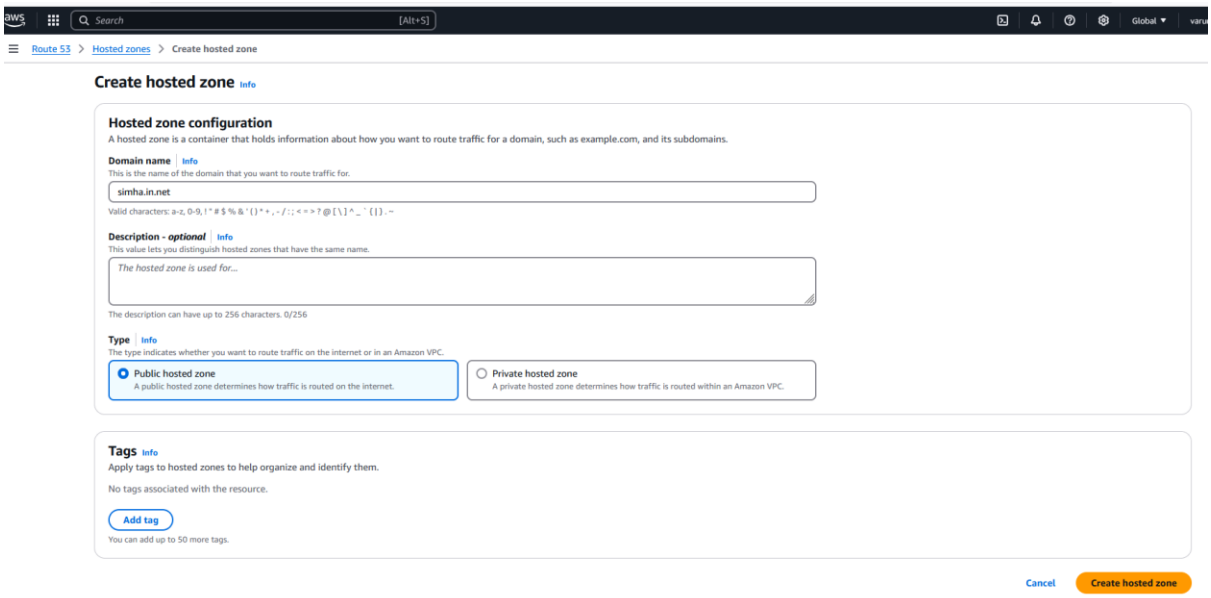
Setting up a secure web application using Route 53, a Load Balancer, and ACM

1. Purchased or registered a domain (in this case, it was purchased through BigRock, a third-party provider).

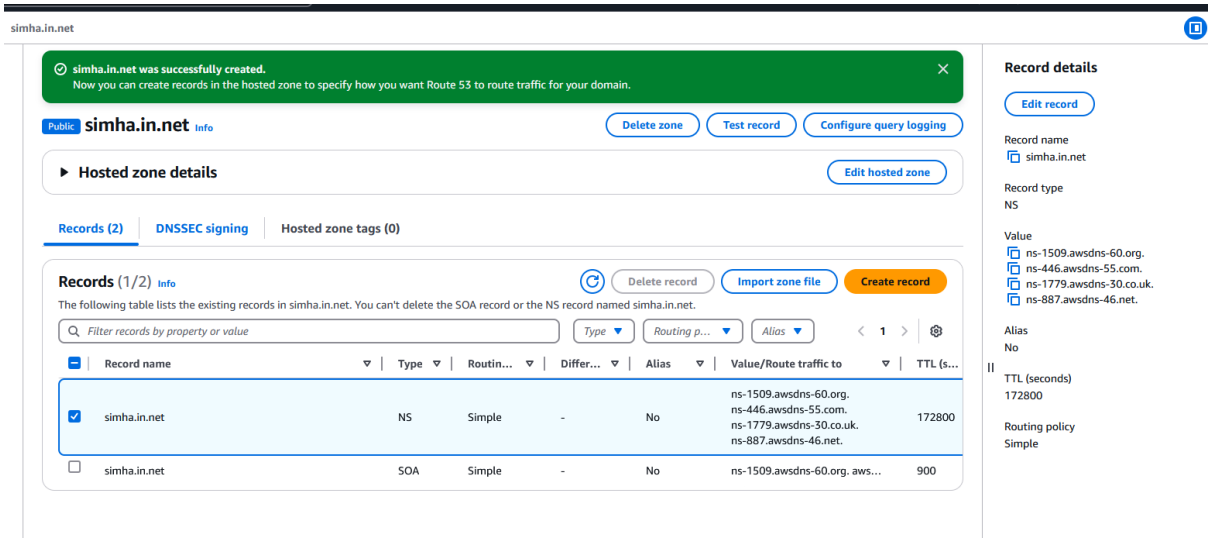


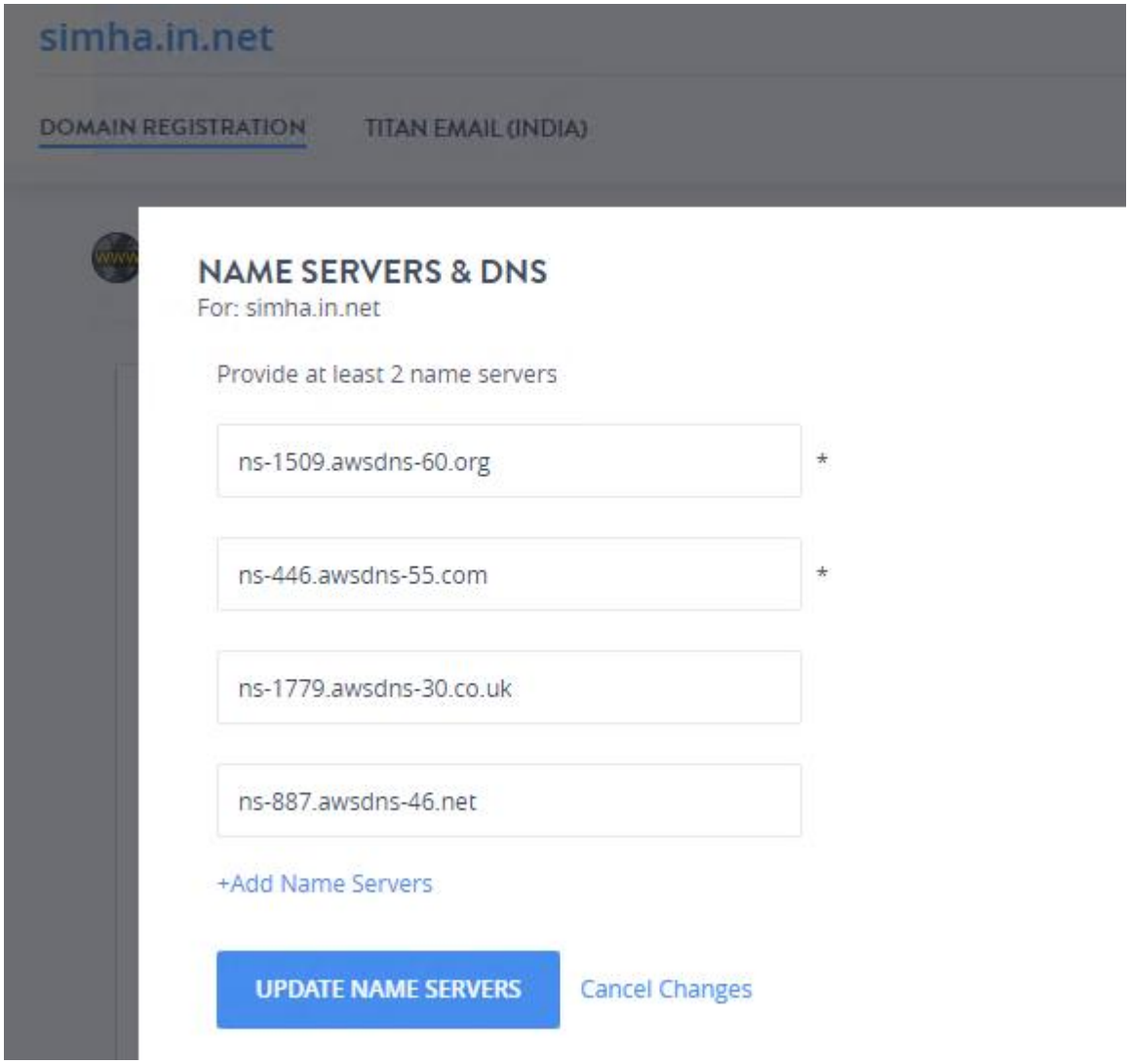
2. Create a hosted zone on Route 53 by providing the domain name you purchased and selecting the type as **Public Hosted Zone**.



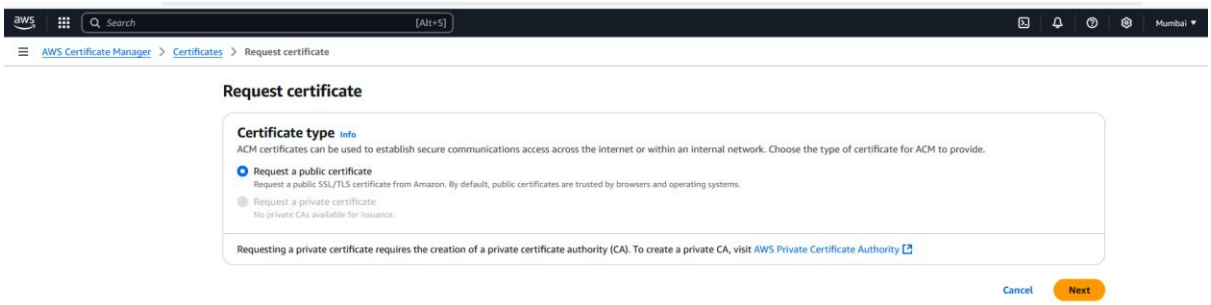


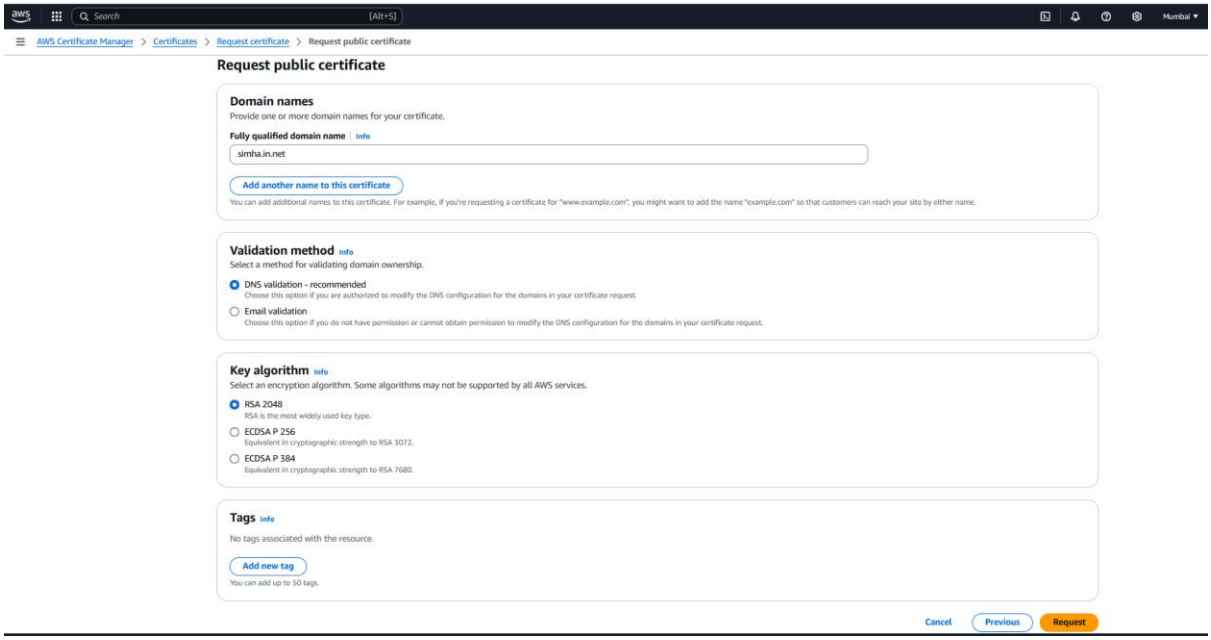
3. After creating the hosted zone, copy the **NS** (Name Server) record values and paste them into the **Name Server** settings where you purchased the domain.



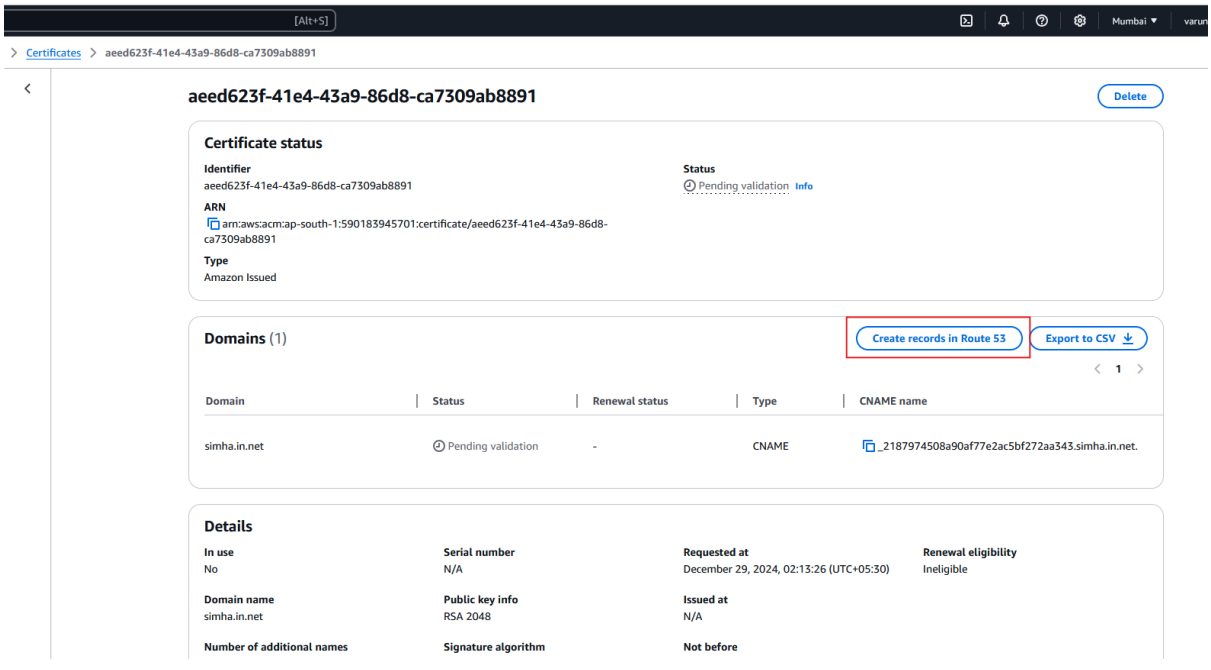


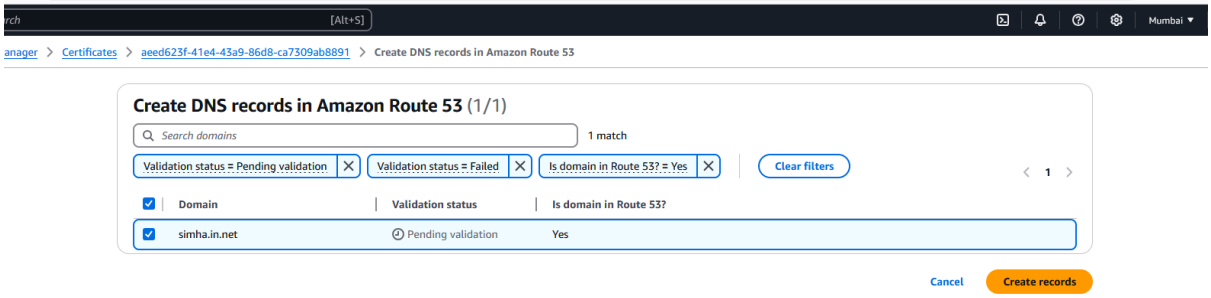
4. Now, create a certificate in ACM for the region where the public instance is located. Choose the certificate type as **Public** and leave the remaining settings as default.

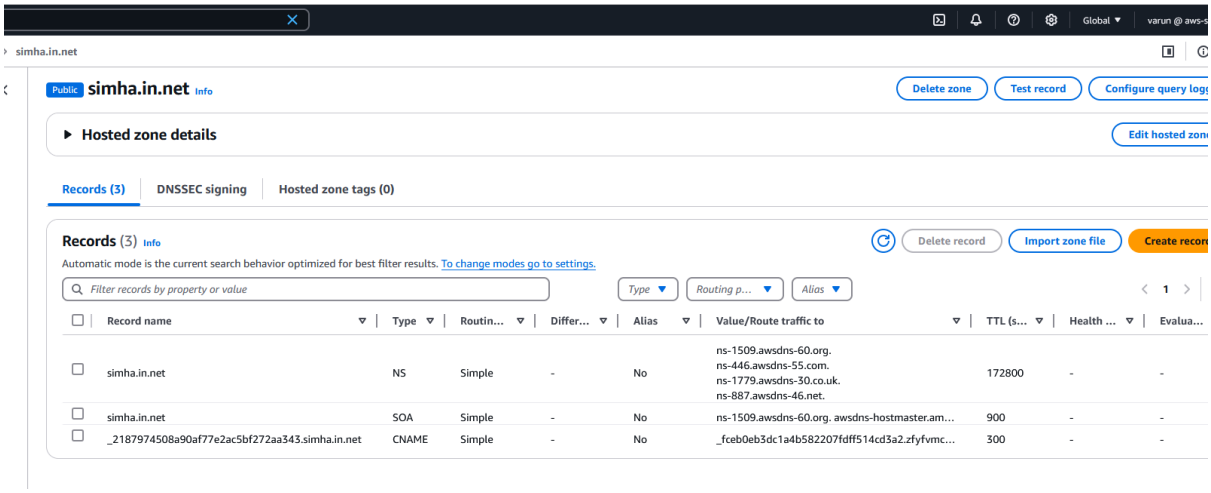




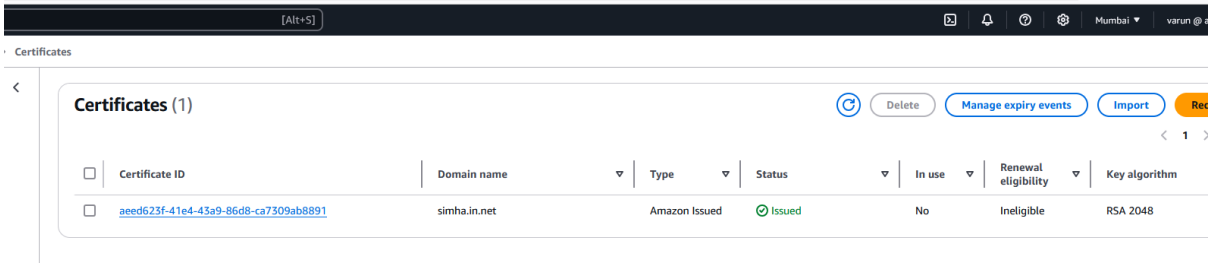
5. After the certificate is created, copy the **CNAME** record and then add it to Route 53.



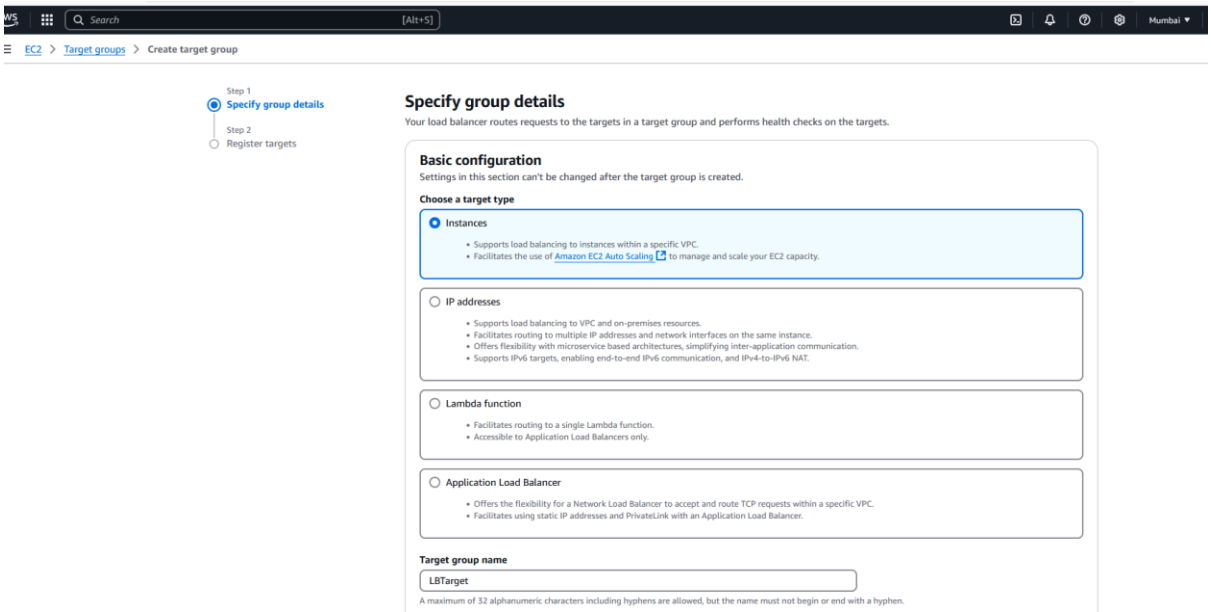


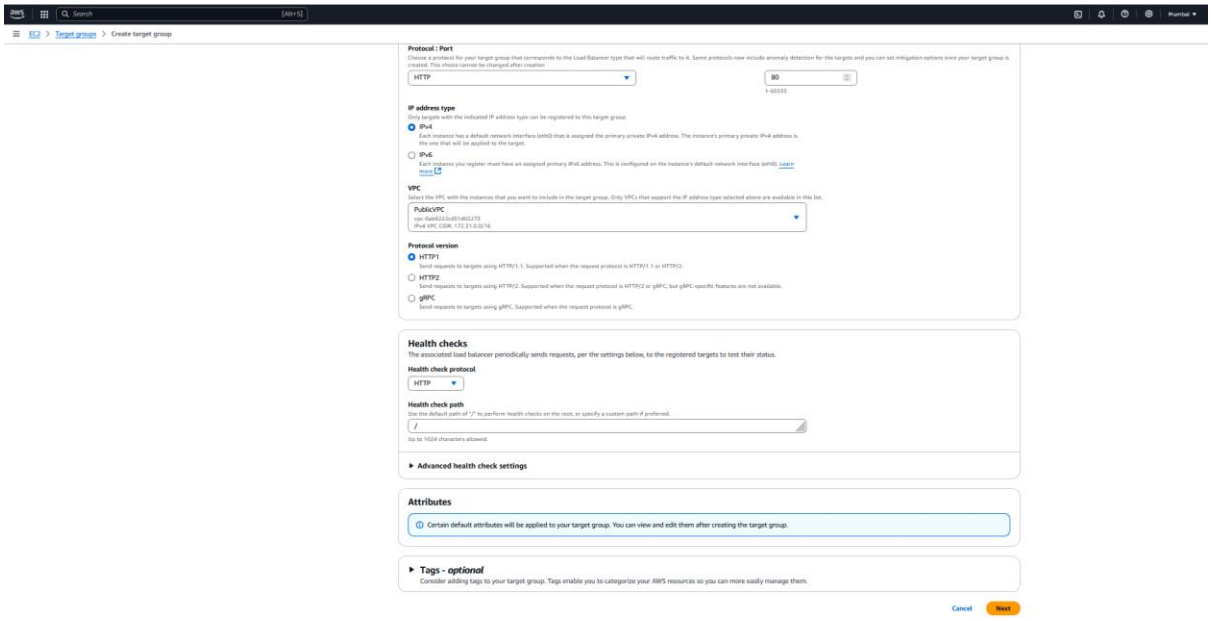


6. Wait for a few minutes for the validation to complete, and the status will change to **Issued** in the ACM service.

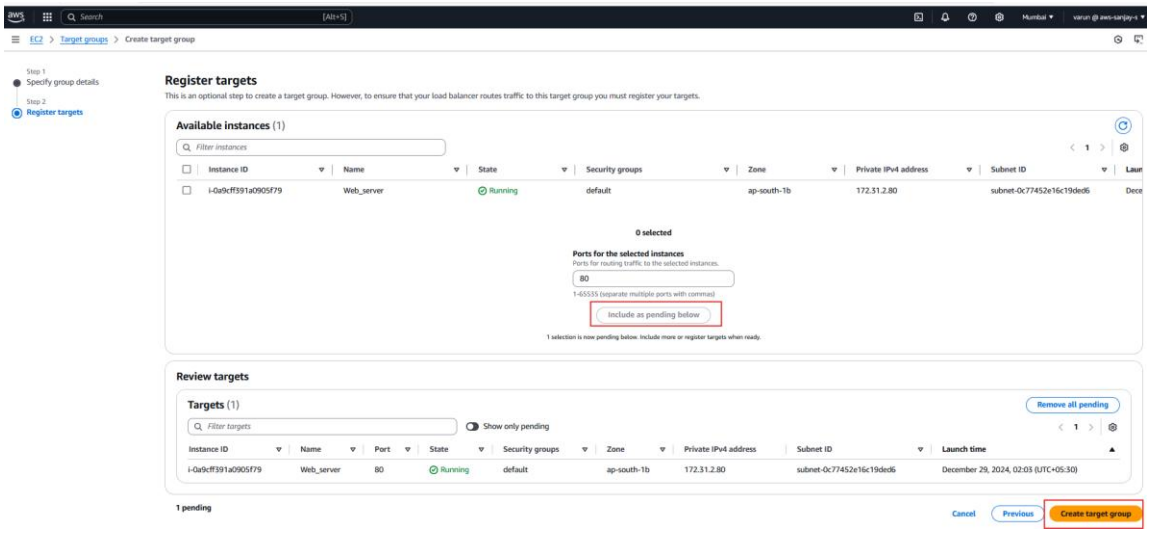


7. Now, create the **Target Group** for the Load Balancer, selecting the target type as **Instance**, and leave the remaining settings at their default values.

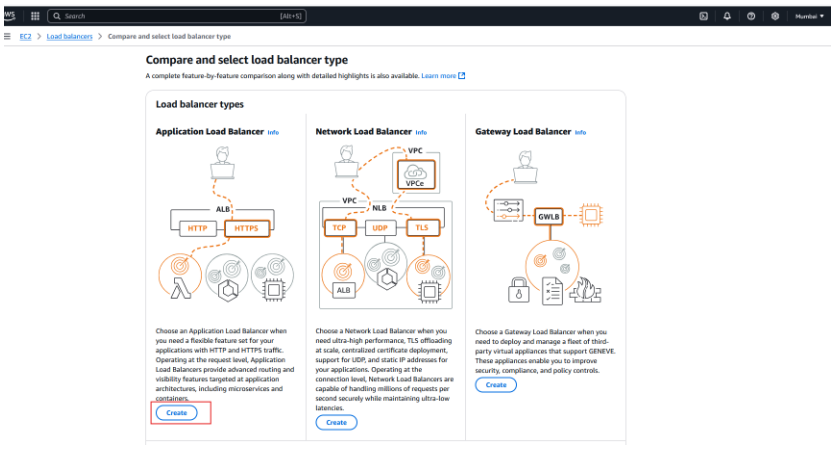




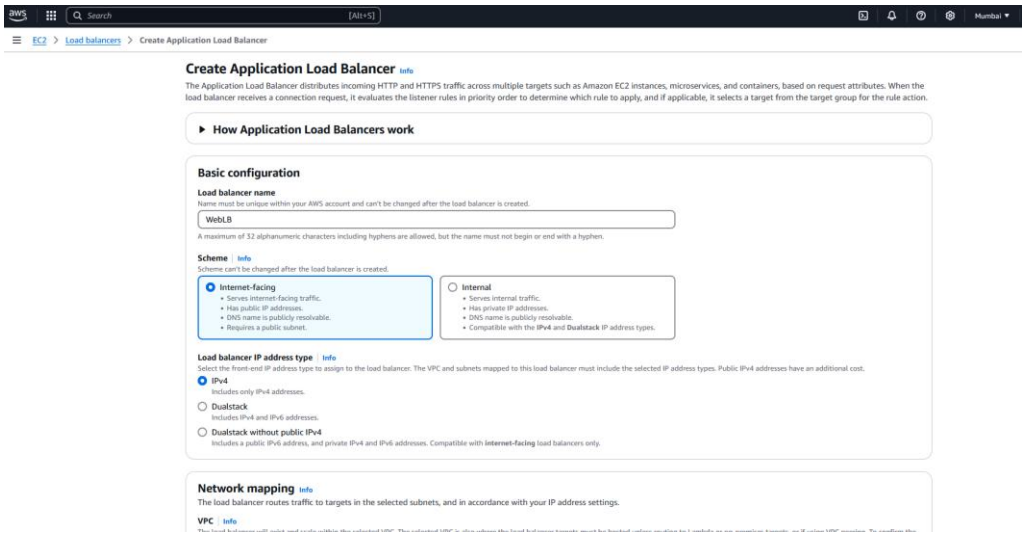
8. Here, selects the instance that needs to be registered as the target for the target group.

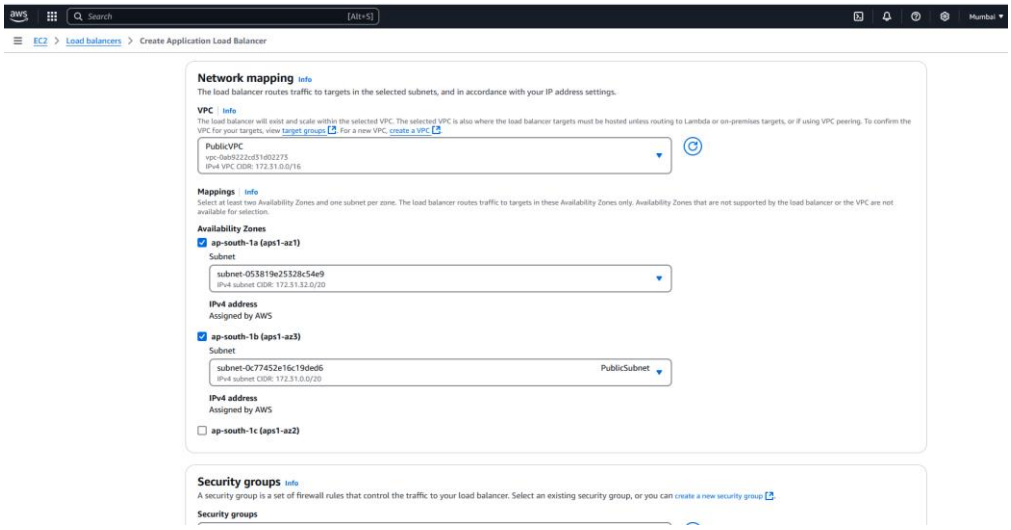


9. After creating the target group, create the **Application Load Balancer** to allow secure communication to the instance.

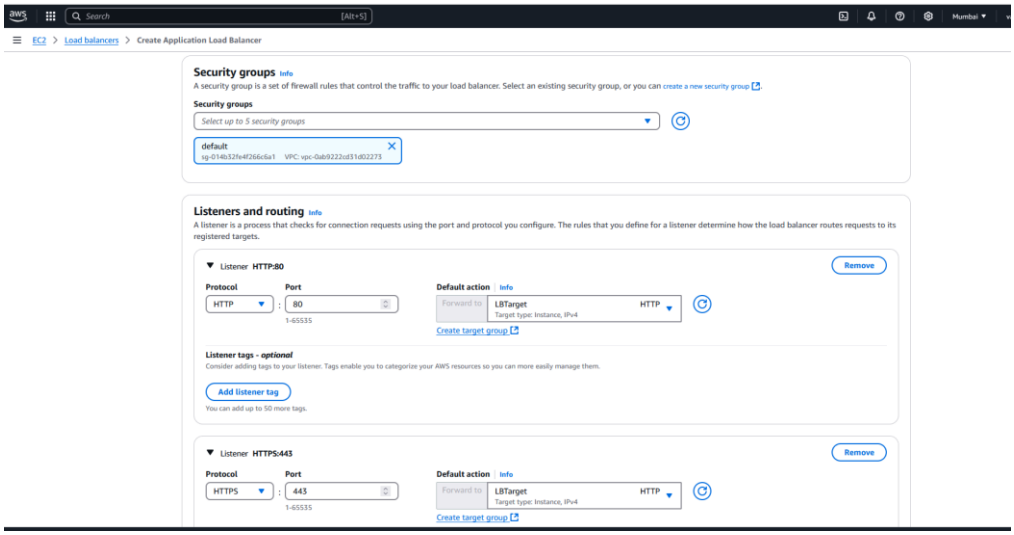


10. Configure the internet-facing and choose the VPC and its minimum two availability zone where instance lies.

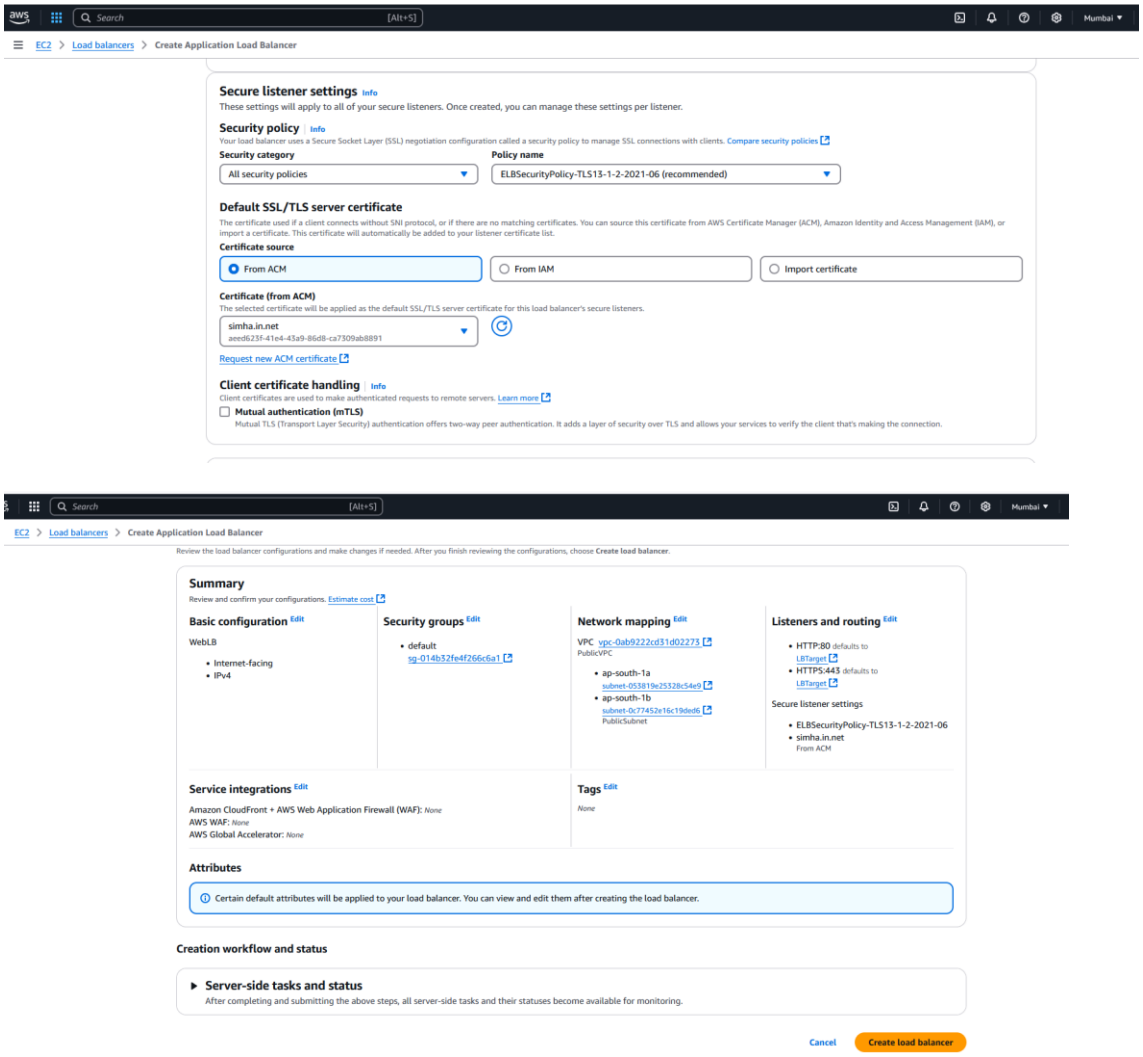




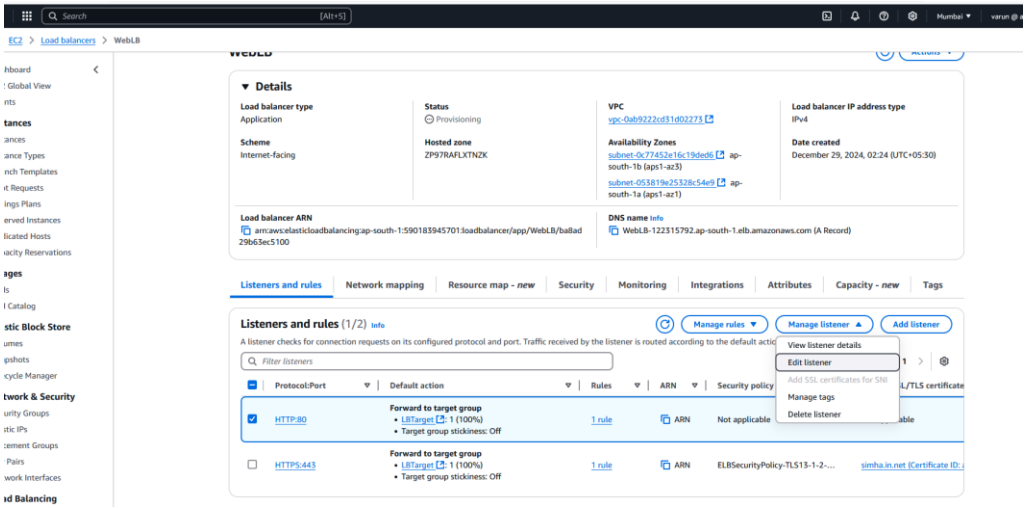
11. On the **Listeners and Routing** section, add both **HTTP** and **HTTPS** listeners. For both listeners, set the default action to **"forward to target group"**.



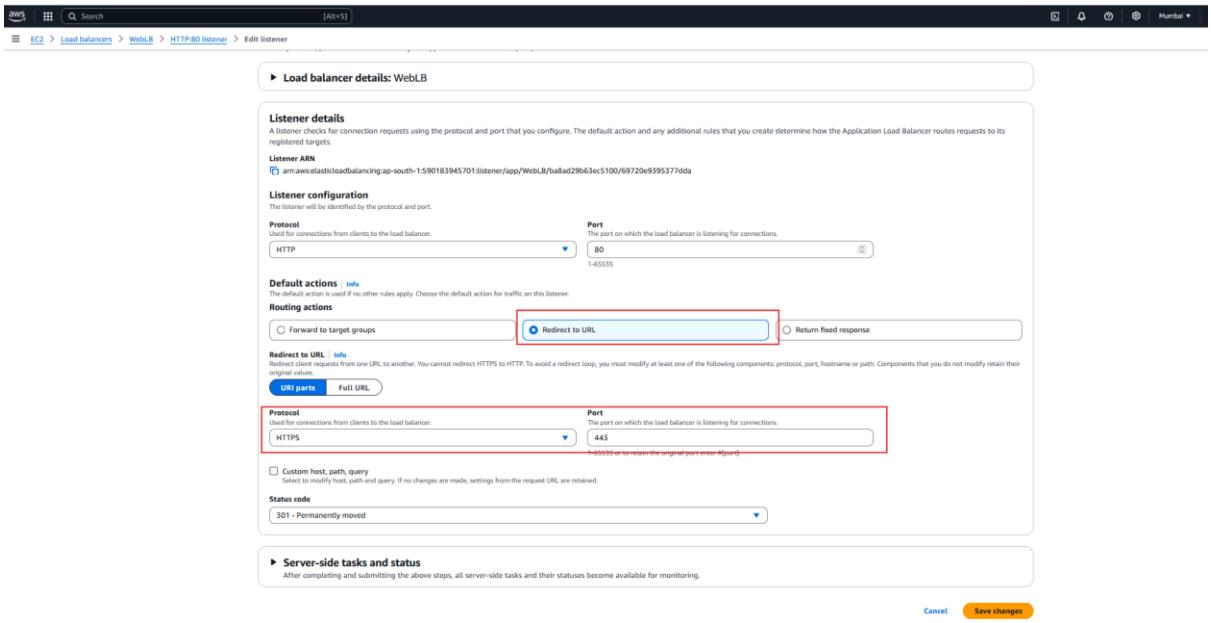
12. Since we have added the **HTTPS listener**, choose the **default security policy**. For the certificate, select **from ACM** and choose the certificate that was created earlier. Then, click on Create.



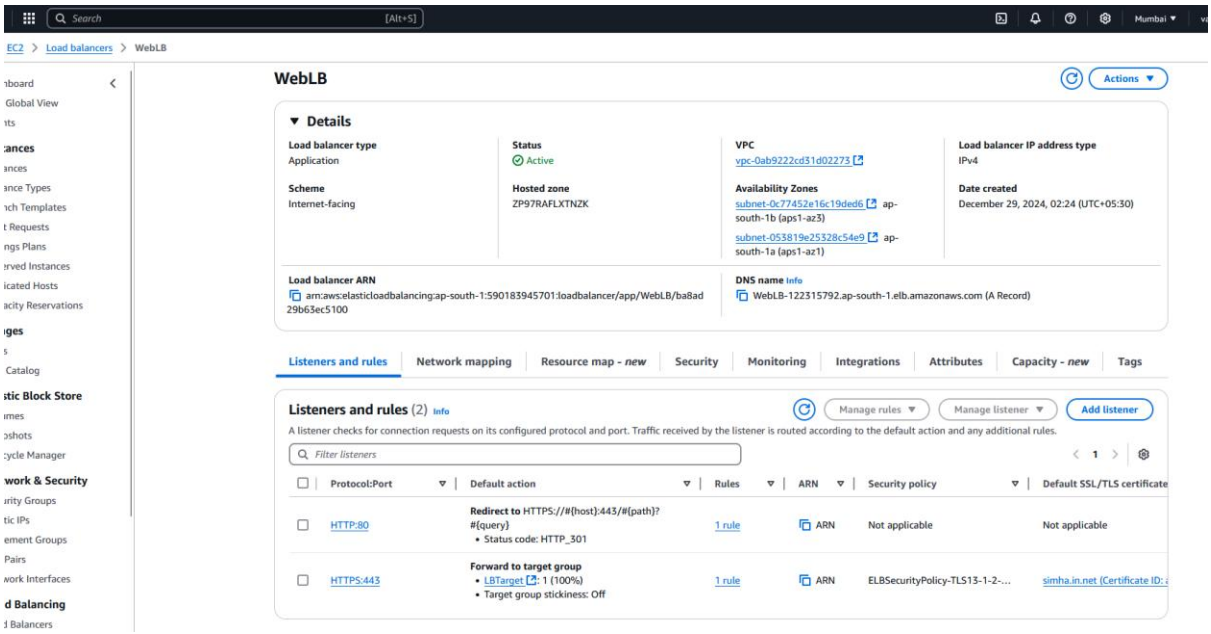
13. This will take a few minutes to set up. Once the process is complete, you can edit the **HTTP listener** settings.

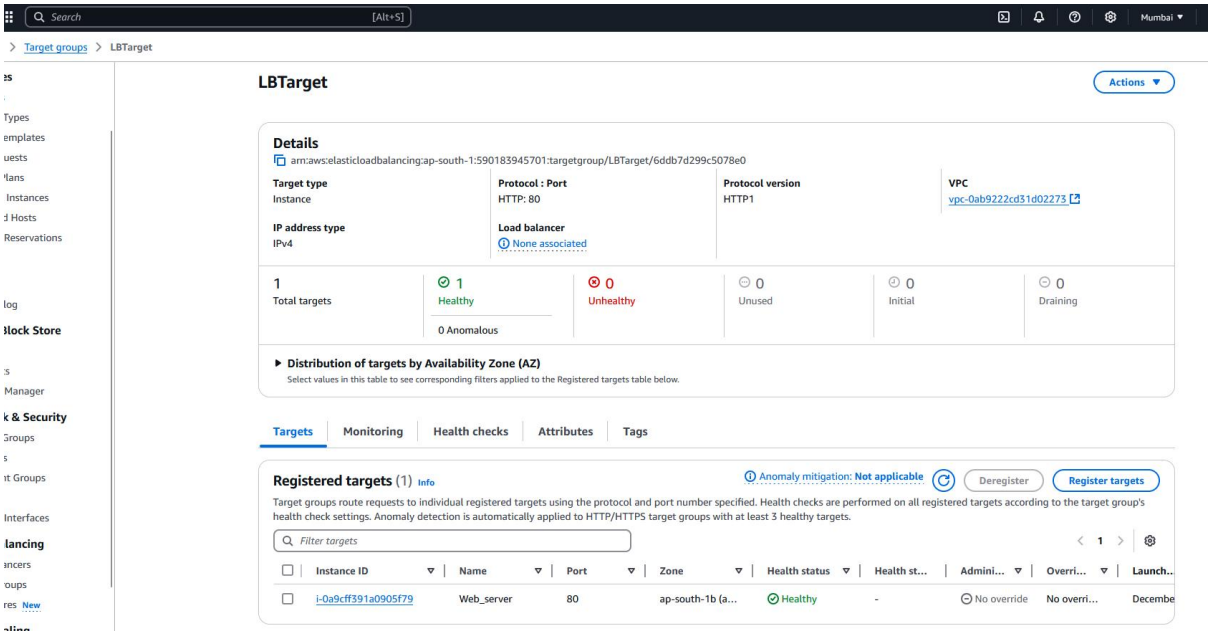


14. While editing the HTTP listener, change the default action to **"Redirect to URL"** and set the URL port to **HTTPS – 443**. This ensures that whenever a user tries to access the application using HTTP, they will be automatically redirected to HTTPS for a secure connection.

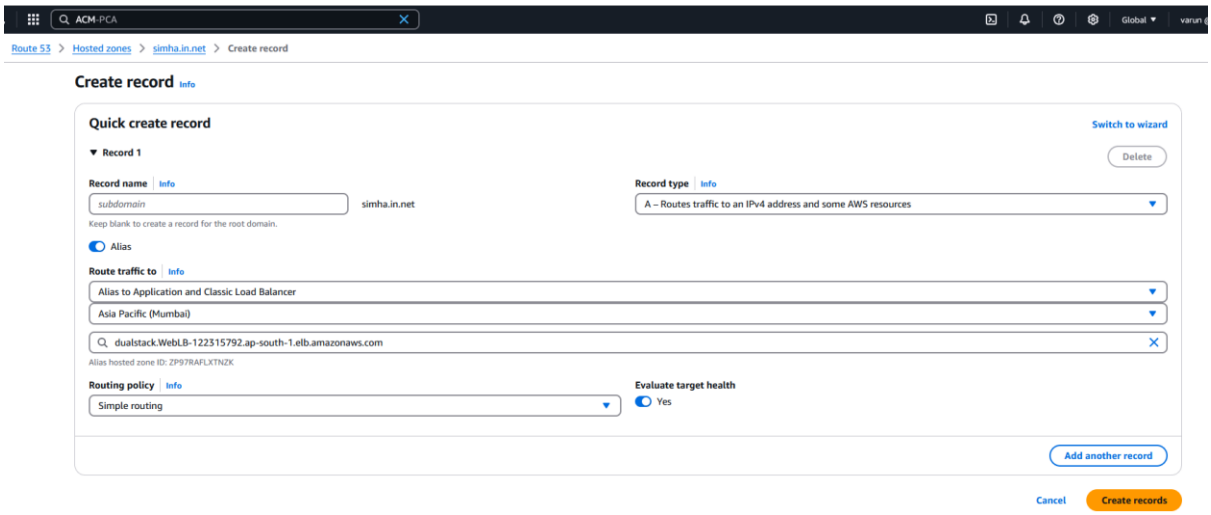


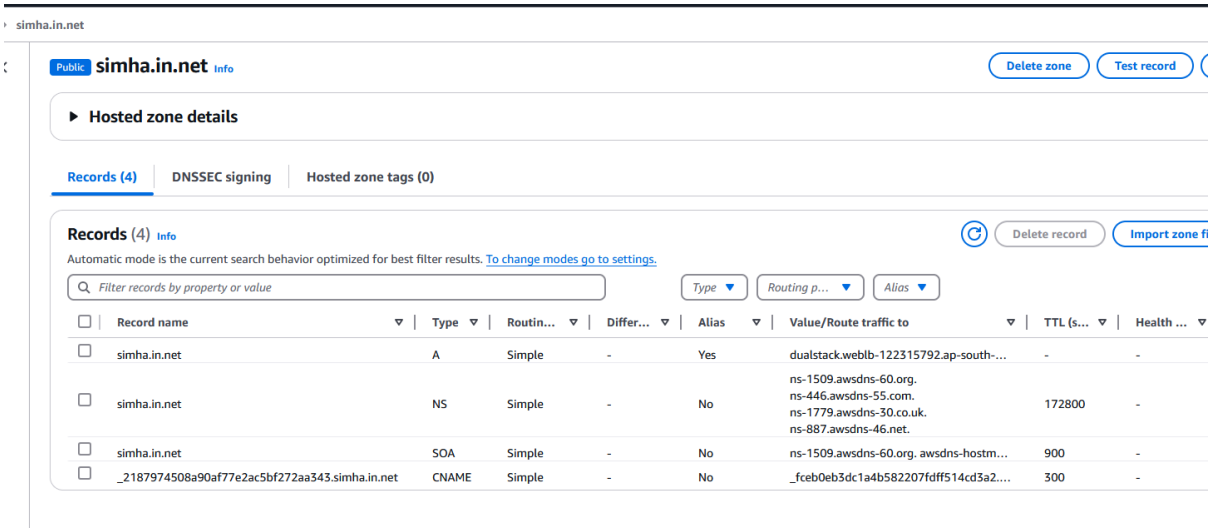
15. After the Load Balancer is active, the target group will begin to register the instances, and the status of the instances should show as **healthy**. This indicates that the instances are properly receiving traffic and responding as expected through the Load Balancer.



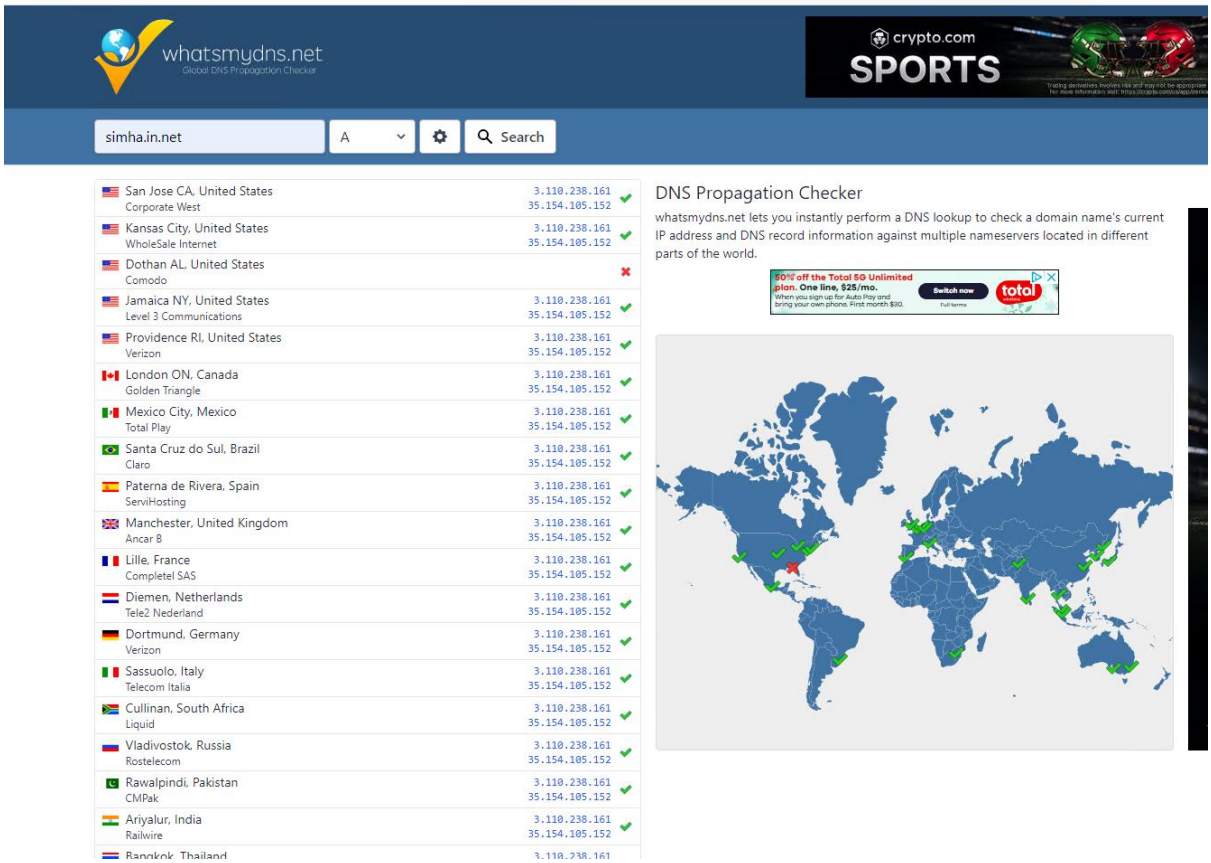


16. Now, in Route 53, create a record for HTTPS by setting the traffic routing value to the application Load Balancer and specifying its region, along with the Load Balancer's DNS name.



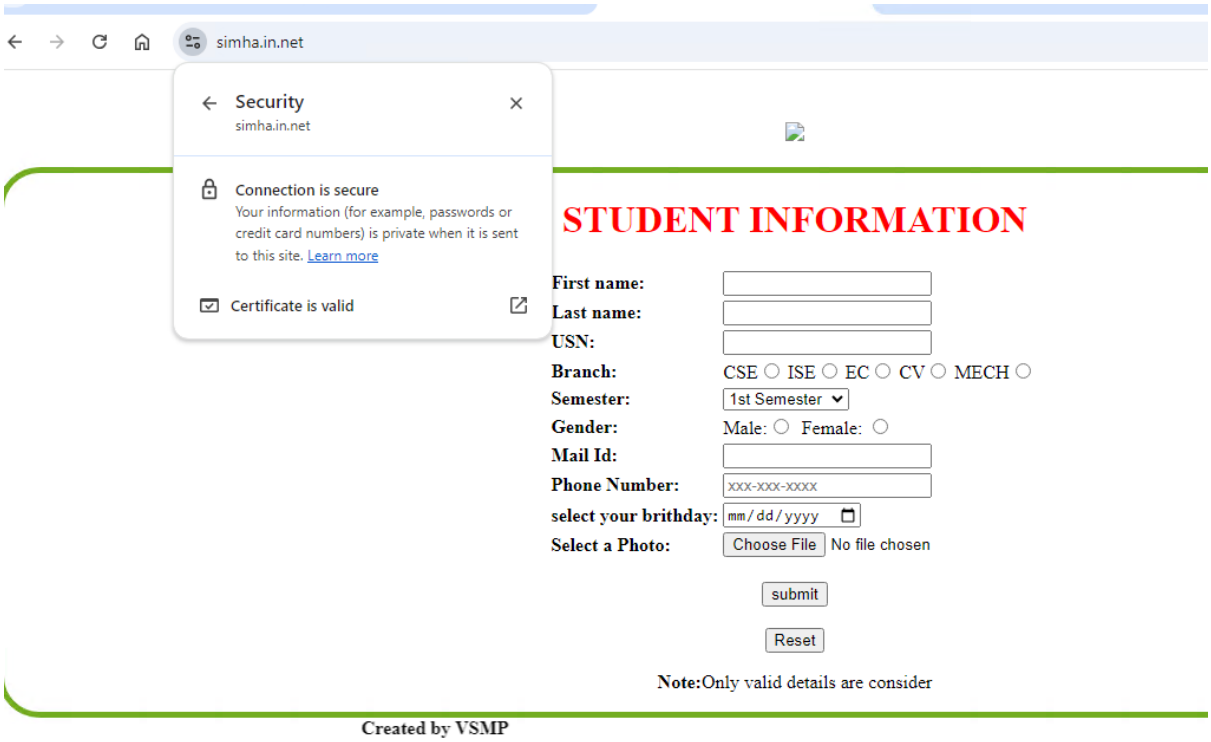


Note: We can check the DNS from all the location where it is accessible using the “whatsmydns” website:



17. Wait for a few minutes for the secure connection to be established. After that, search for "simha.in.net" and ensure it appears as secure.

Note: The time for this process may vary; it could take only a few minutes, or sometimes longer.



Conclusion on this module:

We have successfully configured a secure HTTPS web application using the Load Balancer, Route 53 for DNS, and ACM for the SSL certificate.

Overall, of this Project:

Here I like to conclude that, I have designed and implemented a **secure multi-tier cloud architecture** leveraging a range of AWS services. This project showcases a deep focus on **security**, **cost optimization**, and **effective resource utilization** while remaining adaptable for future scalability.

Core Components and Configuration

Public Instance (Web Layer):

- Deployed in a **public subnet**, serving as the entry point for users.
- A dynamic website is hosted on this instance using **Git technology** to pull the codebase directly from a **GitHub repository**.
- Configured with HTTPS using **AWS Certificate Manager (ACM)** to ensure secure communication.

- This instance connects securely to a private database hosted in the private subnet.

Private Instance (Application and Data Layer):

- Configured in a **private subnet** in a different AWS region for enhanced security.
- **NAT Gateway** is deployed in the public subnet to provide **controlled internet access** to the private instances.
- **Amazon RDS (Relational Database Service)** hosts the backend database, providing automated backups, high availability, and reliability.
- An **S3 bucket with a VPC endpoint** is integrated into this instance for secure file storage, eliminating the need for internet exposure.

Fey Features

✅ **VPC Peering Connection:**

- Established between the public and private VPCs to enable seamless, secure communication.
- Eliminates the need for direct internet access, ensuring a robust, private, and secure data flow.

✅ **AWS Lambda and Event Bridge:**

- Automated **powering on/off of instances** during non-business hours, significantly reducing operational costs.
- Event-driven architecture ensures efficient management of resources without manual intervention.

✅ **Domain and SSL Configuration:**

- The domain name was purchased through **Big Rock** and seamlessly configured in **Amazon Route 53** for DNS management.
- **AWS ACM (Certificate Manager)** provides SSL certificates, ensuring secure HTTPS communication for the website.

✅ **Elastic Load Balancer (ELB):**

- Used to distribute incoming traffic securely and efficiently.

- Enables high availability and prepares the architecture for potential future scalability, such as auto-scaling.

Why This Architecture Stands Out

🔒 **Security-First Approach:**

- Private resources in the private subnet have **no direct internet access**, ensuring data and resource isolation.
- SSL encryption safeguards all communication between the website and users.

💡 **Cost Optimization:**

- **Lambda-driven automation** for powering resources on/off during non-peak hours reduces operational costs.
- Efficient use of VPC endpoints and multi-region VPC peering minimizes data transfer costs.

🛠️ **Future-Ready Design:**

- The architecture is built to scale with growing workloads by enabling auto-scaling and other cloud-native features like **WAF (Web Application Firewall)** for enhanced security.

💼 **Business Applications and Use Case**

This multi-tier architecture is ideal for businesses that need secure, reliable, and cost-effective web application hosting in the cloud. Benefits include:

- **High Security**: Private and public layers ensure data isolation and controlled communication.
- **Cost Efficiency**: Automated management minimizes wasteful spending.
- **Scalability**: Built to scale with business growth.

🎯 **Personal Insights and Future Enhancements**

This project demonstrates how AWS services can be creatively combined to deliver secure, efficient, and scalable solutions. Looking ahead, I plan to:

- Enable **Auto Scaling Groups** to handle fluctuating workloads.

- Integrate **AWS WAF** to improve application security against evolving threats.
- Incorporate **CloudWatch monitoring** for better insight into system performance.

🌐 **What are your thoughts on this architecture?**
Have you worked on similar projects? I'd love to hear your feedback and ideas on how we can enhance this design even further!

#AWS #CloudArchitecture #VPC #RDS #Lambda #S3 #Route53 #Automation #SSL #Scalability