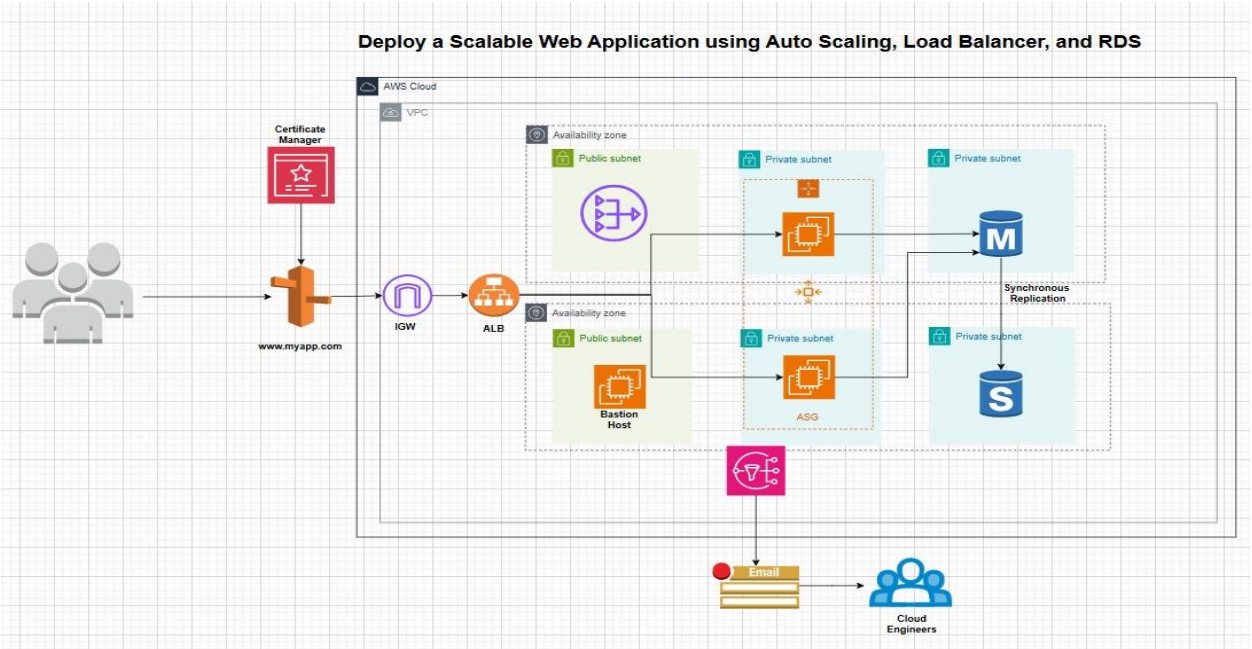
AWS 2-Tier Architecture Deployment – Professional Documentation
Author: *Aniket Talwekar*
Experience Level: 3+ Years | AWS Cloud & DevOps Engineer
Project Type: Hands-on Real-Time Cloud Deployment
Tech Stack: AWS (VPC, EC2, ALB, ASG, RDS, Route 53, ACM), Linux, PHP, MySQL

1. Project Objective
    1. The main objective of this project was to design, deploy, and manage a highly available and secure 2-tier web application architecture on AWS — a setup that mirrors how real-world production environments run enterprise web applications.
    2. I wanted to gain end-to-end cloud deployment experience, ensuring:
    3. Security: Application and database isolation
    4. Scalability: Auto scaling for unpredictable load
    5. Availability: Multi-AZ architecture
    6. Cost Optimization: Right-sized instance selection
    7. Professionalism: HTTPS access, domain integration, and monitoring
    8. This project helped me implement all the AWS building blocks that form the foundation of DevOps and Cloud Engineering.

2. Architecture Diagram



3. Architecture Overview
    1) A 2-Tier architecture separates the application into two logical layers:
        a. Web/Application Tier (Frontend Layer)
        b. Responsible for serving the web app and handling all client-side requests.
        c. Hosted on EC2 instances managed by an Auto Scaling Group (ASG) behind an Application Load Balancer (ALB).
    2) Database Tier (Backend Layer)
        a. Hosted on Amazon RDS (MySQL) for secure, managed, and fault-tolerant data storage.
        b. Placed inside private subnets for maximum security.

4. High-Level Flow
    1. End-user requests https://aniket123.shop

2. Route 53 resolves the domain to the Application Load Balancer (ALB).
3. ALB distributes traffic across EC2 instances located in private subnets.
4. Each EC2 instance (running Apache + PHP) processes the request and communicates with the RDS MySQL database.
5. All sensitive communication happens over private IPs within the VPC.
6. A Bastion Host is used for administrative access — providing a controlled entry point to private resources.
7. Auto Scaling Group (ASG) automatically increases or decreases EC2 instances based on load.
8. ACM (SSL) ensures HTTPS and encrypted traffic for end-user communication.

1. AWS Services Implementation (Detailed Explanation)

1. VPC & Networking

Creating a custom Virtual Private Cloud (VPC) was the foundation of the project.

a. VPC CIDR: 10.0.0.0/16

2. Subnets:

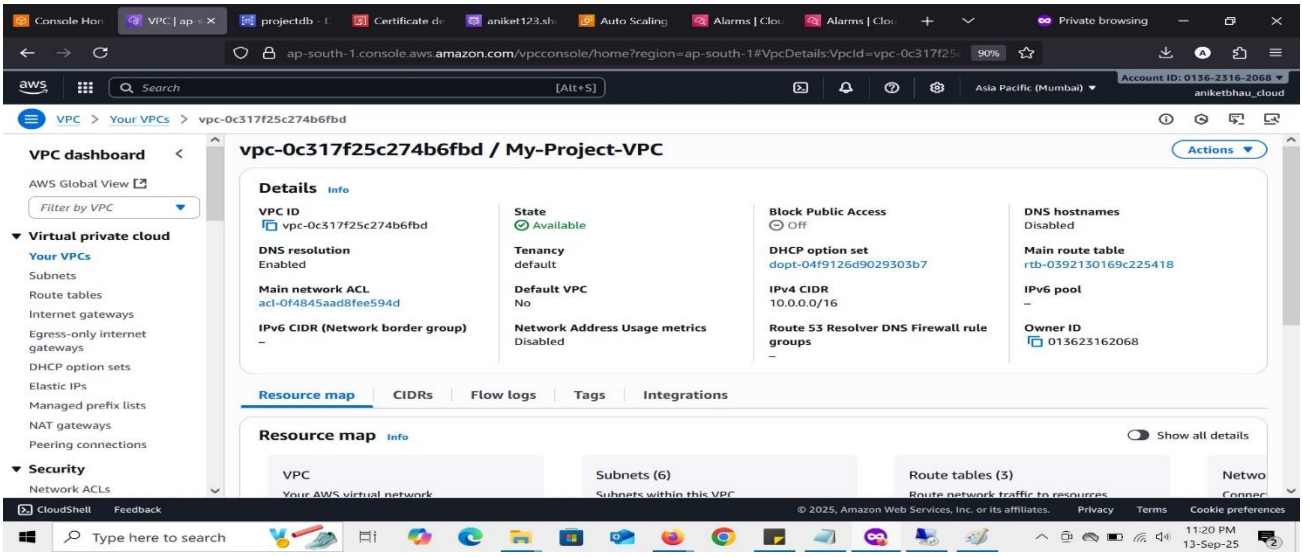
a. Public Subnets → for ALB and Bastion Host
b. Private Subnets → for App EC2 and RDS MySQL

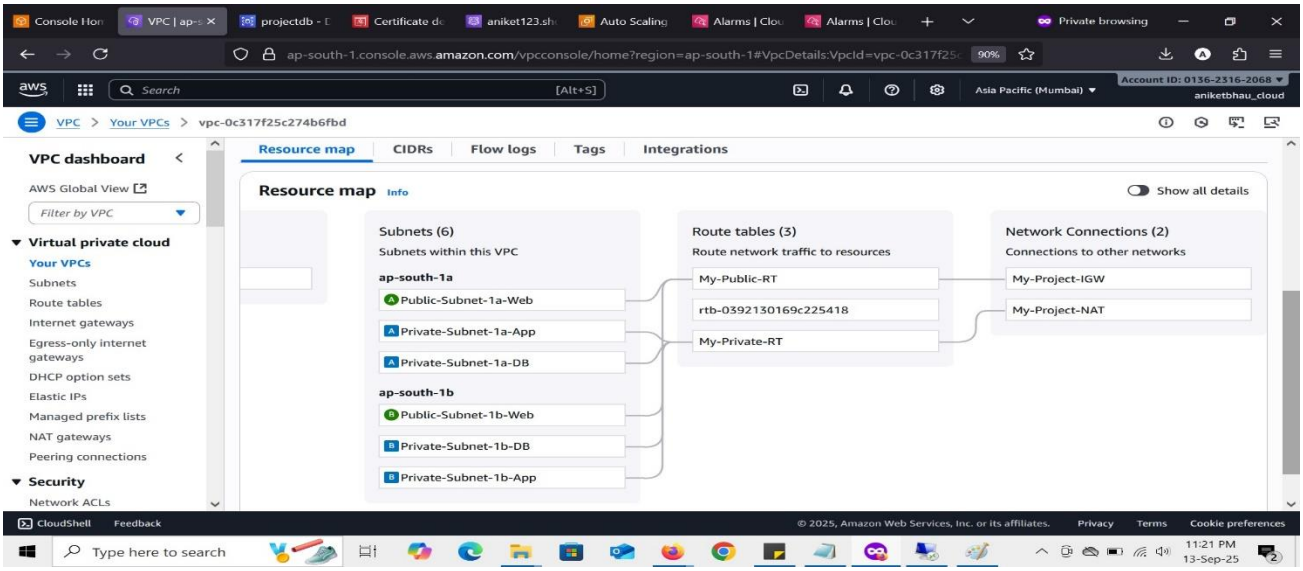
3. Internet Gateway (IGW): For public internet access
4. NAT Gateway: For private EC2 instances to access updates without direct exposure
5. Route Tables:

a. Public subnets route → IGW
b. Private subnets route → NAT Gateway

Learning:

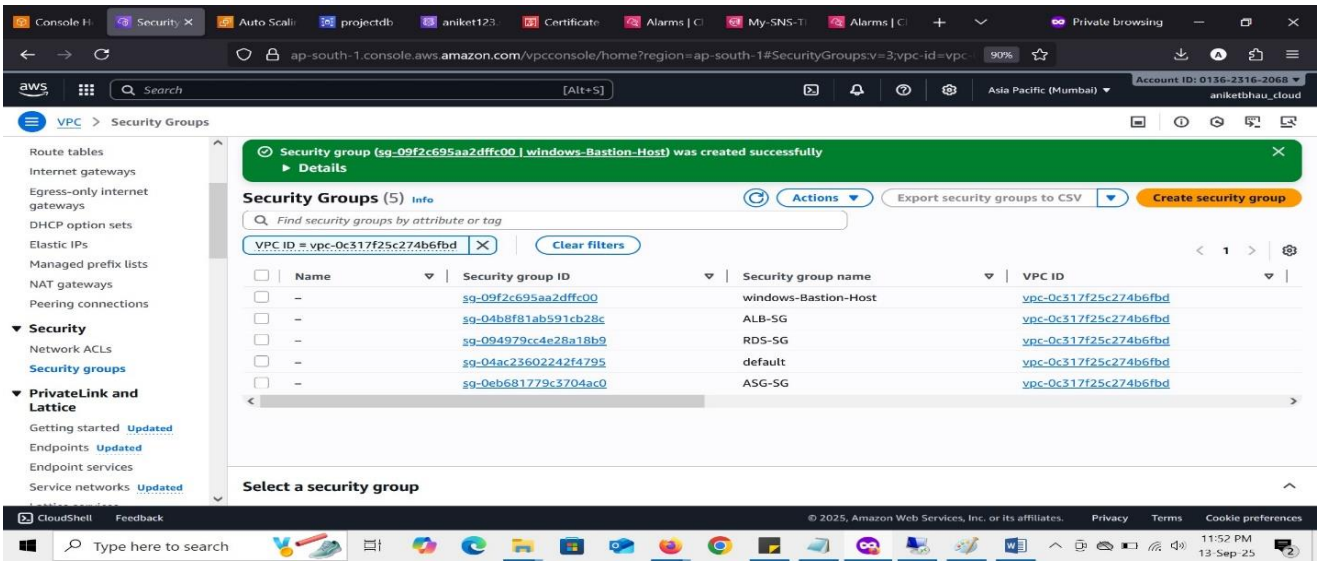
This helped me deeply understand how network segmentation improves security. the database and application tiers are never directly exposed to the internet, aligning with AWS best practices.





Security Groups

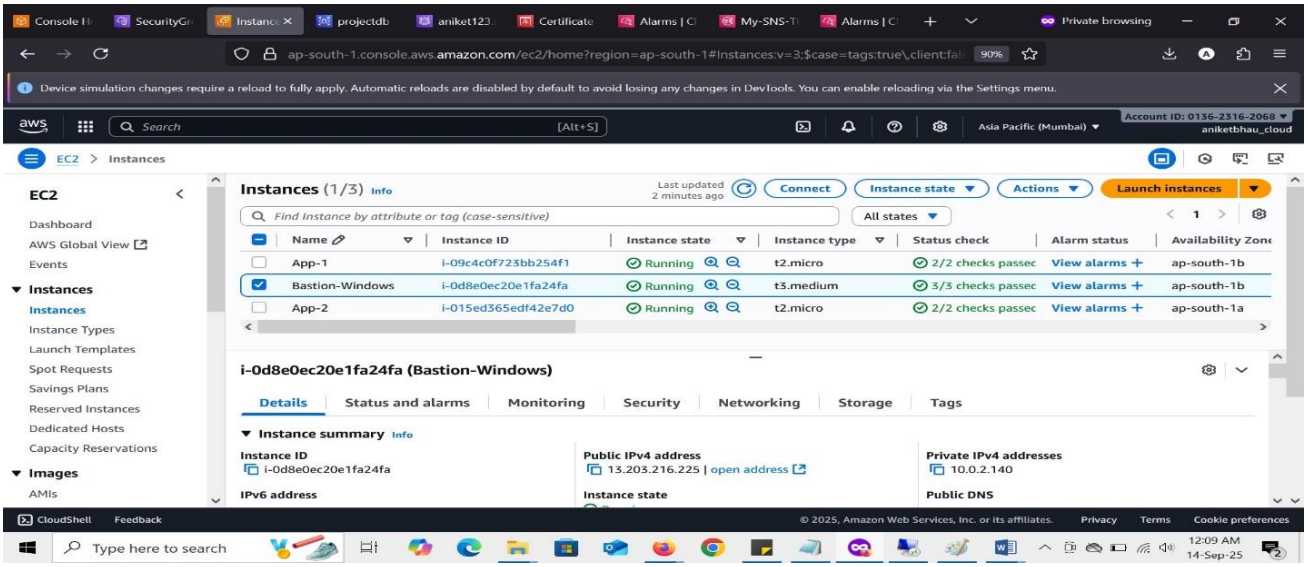
SG Name	Rule	Source
ALB-SG	Allow 80, 443	0.0.0.0/0
App-SG	Allow 80	From ALB-SG
RDS-SG	Allow 3306	From App-SG
Bastion-SG	Allow 3389 (RDP)	From My IP only

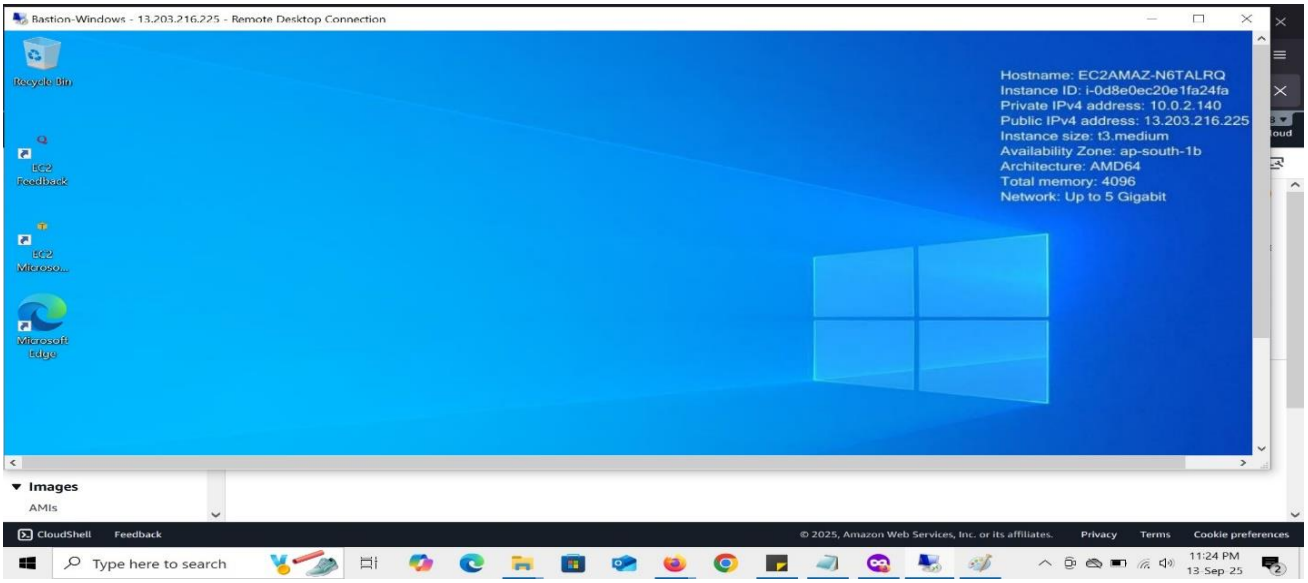


2. Bastion Host (Jump Server)

1. Purpose: To access resources in private subnets securely.
2. Instance Type: t3.medium (Windows Server 2022)
3. Access: RDP from my laptop (restricted via my IP).
4. Used to:
   a. SSH/RDP into private EC2s.
   b. Access MySQL database via EC2.
5. Perform maintenance without exposing internal resources publicly.

Real-World Usage:
In production, teams use Bastion hosts as “security doors” to reach sensitive layers. this setup mimics that perfectly.



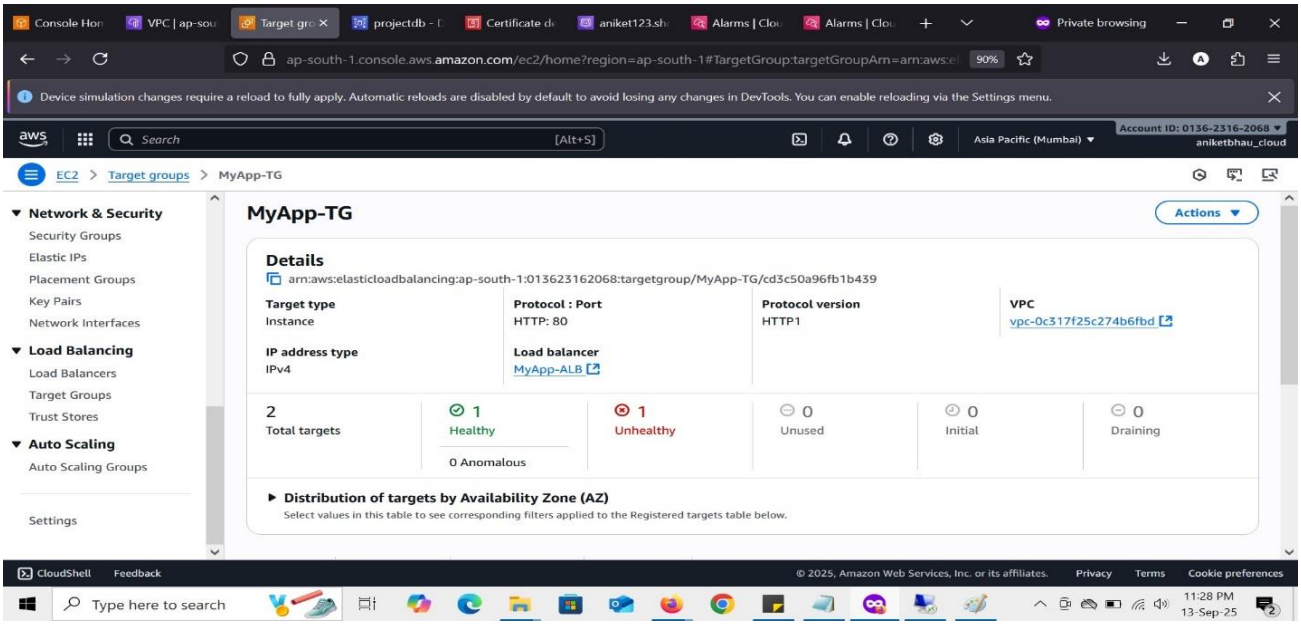


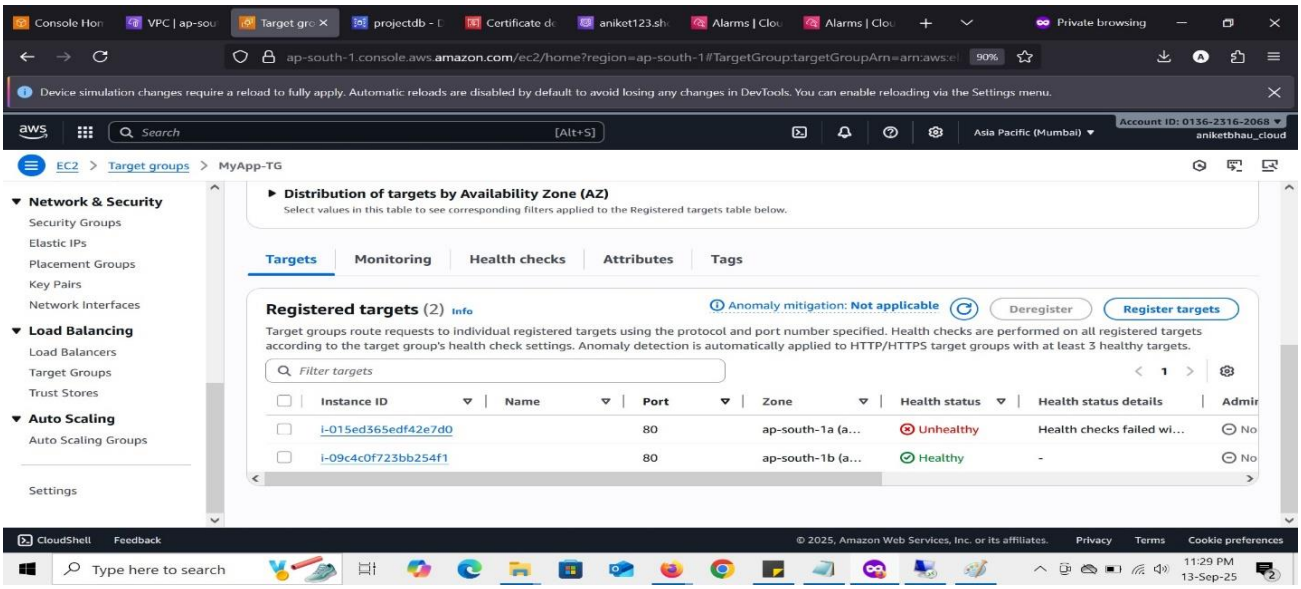
3. Target Group (TG)
   1. Name: MyApp-TG
   2. Target Type: Instances
   3. VPC: MyProject-VPC
   4. Health Checks:

a. Protocol: HTTP
b. Path: /login.php (or / if root page serves index)
c. Healthy threshold: 2
d. Unhealthy threshold: 2
e. Timeout: 5 seconds
f. Interval: 30 seconds

5. Registered Targets:
   a. Auto Scaling EC2 instances (added automatically via ASG).

Why: Ensures only healthy EC2s serve traffic.



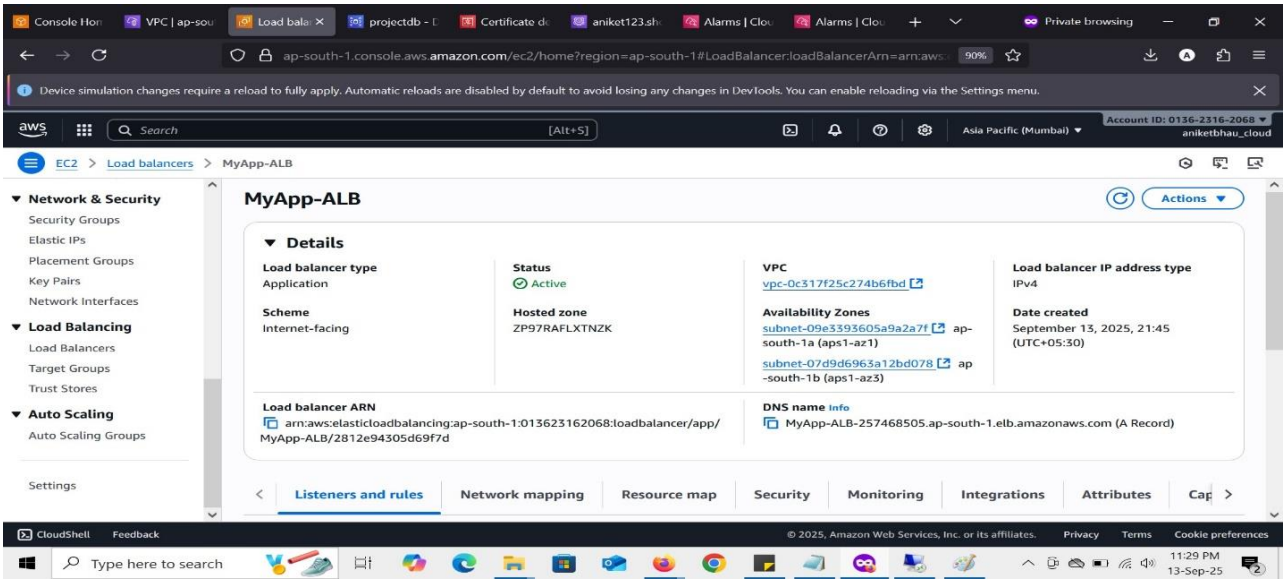


4. Application Load Balancer (ALB)
   1. Scheme: Internet-Facing
   2. Listeners: Port 80 (HTTP), Port 443 (HTTPS)
   3. Target Group: MyApp-TG (registered EC2 instances)
   4. Health Check: /login.php

5. Certificate: SSL (ACM) for *.aniket123.shop
6. ALB terminates SSL and forwards traffic internally over HTTP to EC2.

Why ALB?

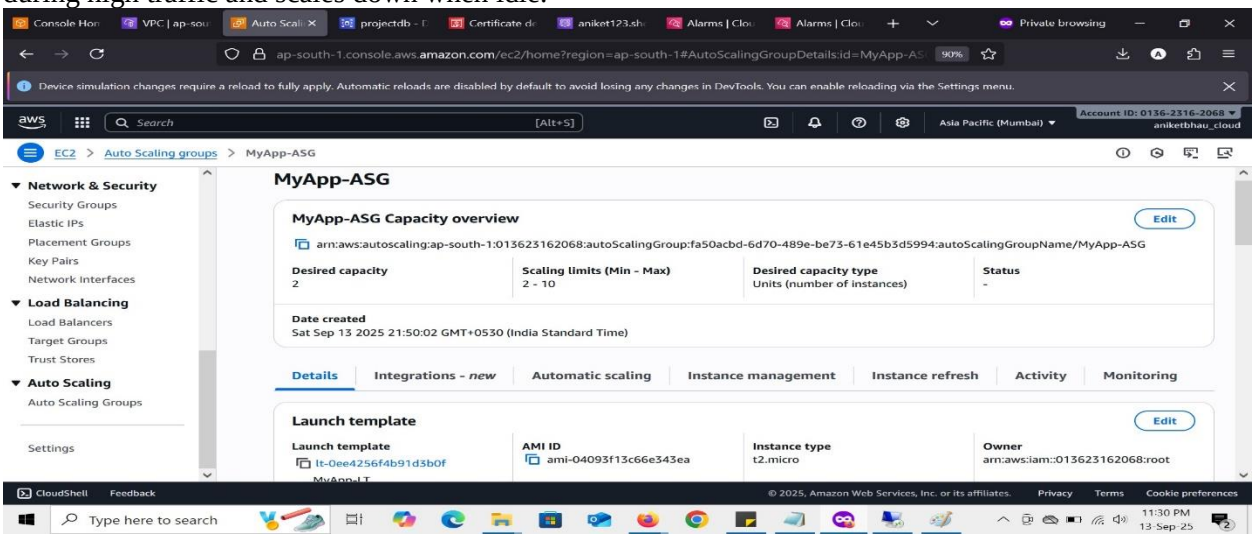
ALB provides load balancing, path-based routing, and SSL offloading, which increases performance and implifies certificate management.

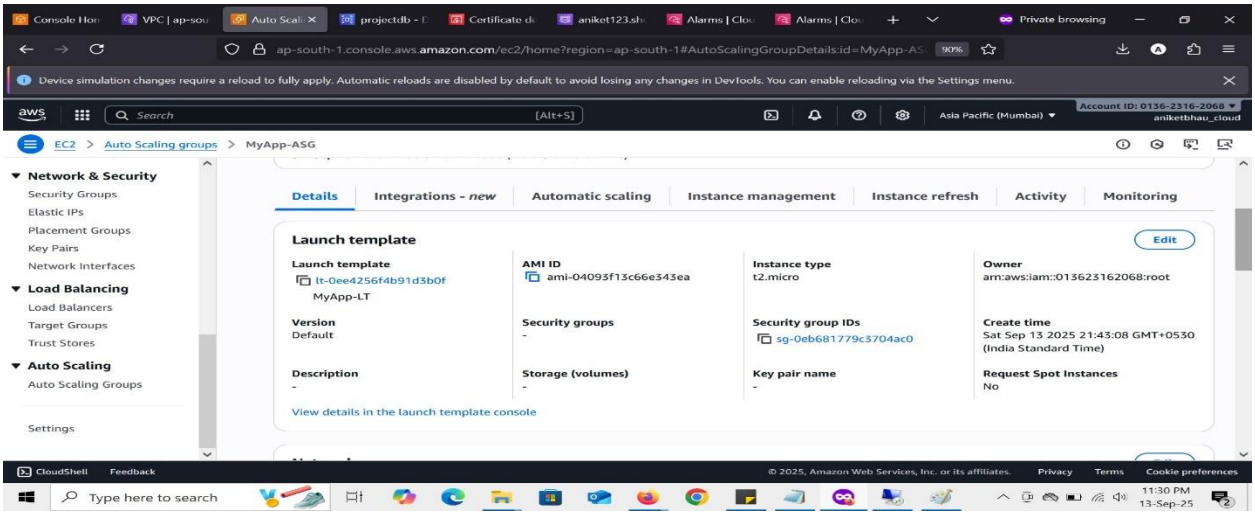


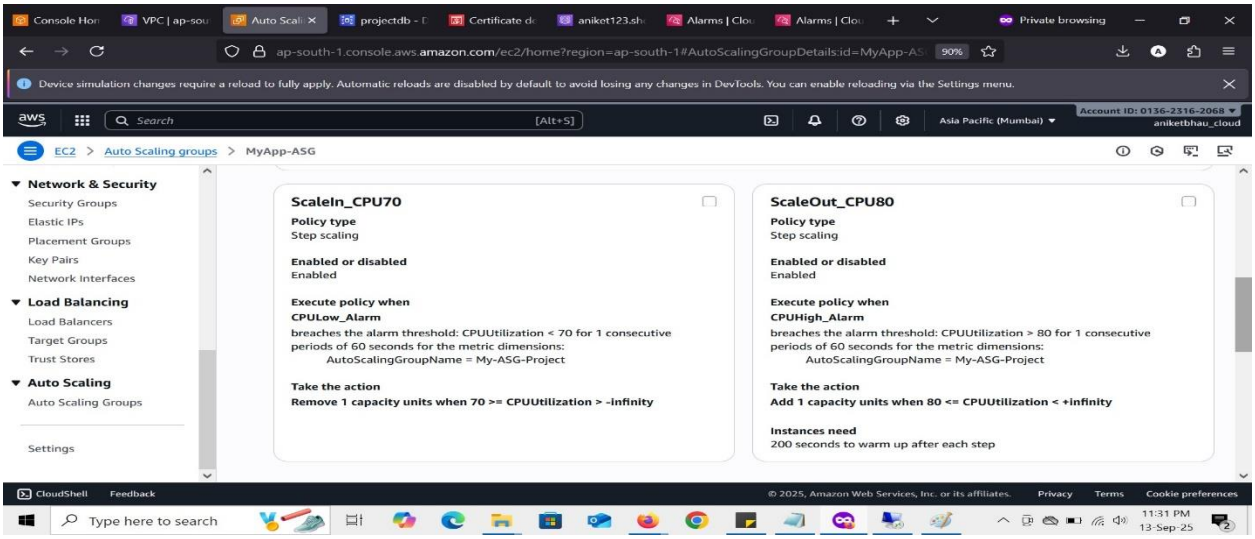
5. Auto Scaling Group (ASG)
   1. Launch Template: Preconfigured AMI with Apache, PHP, and application code.
   2. ASG Configuration:
      a. Desired = 2
      b. Min = 2
      c. Max = 10
      d. Policy: CPU > 80% → Scale Out | CPU < 60% → Scale In
   3. Availability Zones: Multi-AZ setup for redundancy.

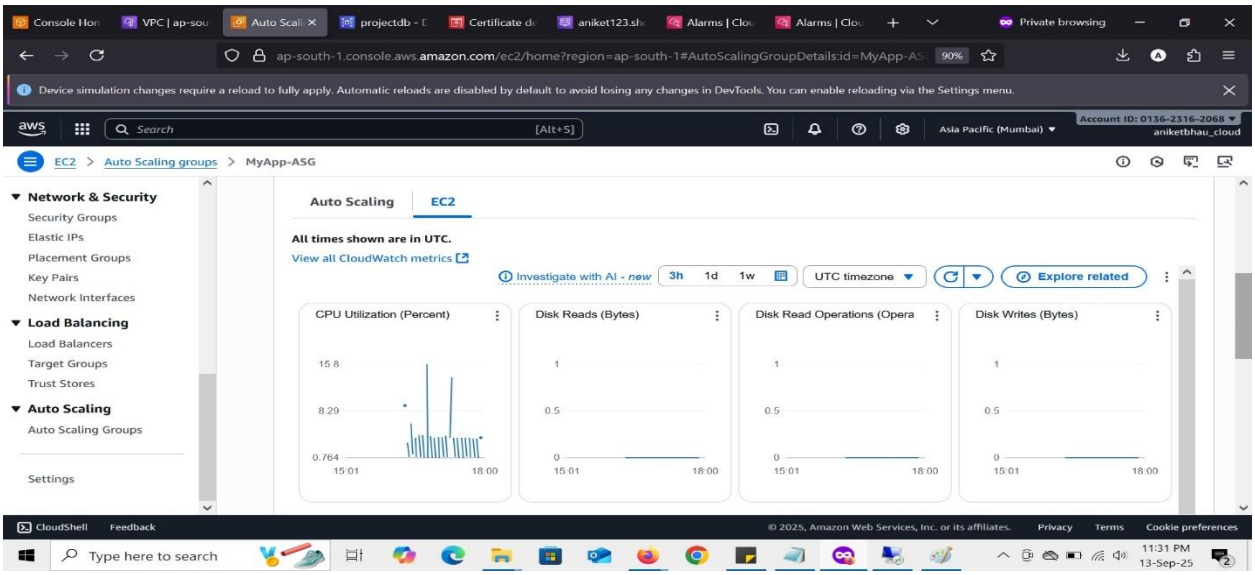
Learning:

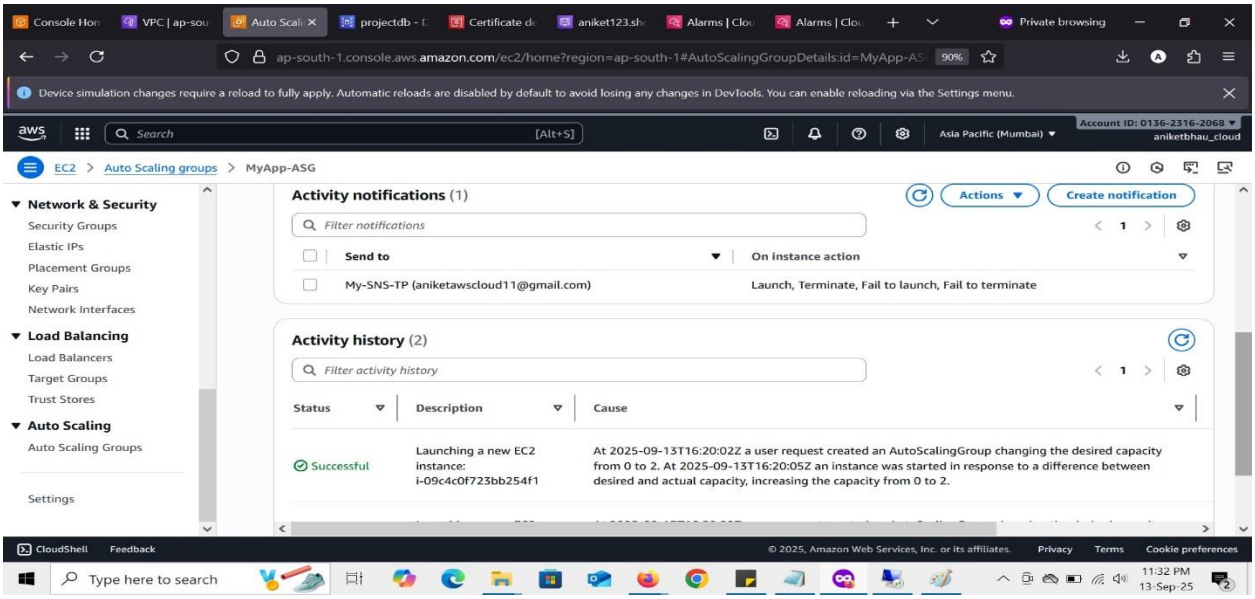
This gave me hands-on experience in elastic scaling — a key DevOps concept ensuring the app scales automatically during high traffic and scales down when idle.




Device simulation changes require a reload to fully apply. Automatic reloads are disabled by default to avoid losing any changes in DevTools. You can enable reloading via the Settings menu.
EC2 > Auto Scaling groups > MyApp-ASG
Details
Integrations - new
Automatic scaling
Instance management
Instance refresh
Activity
Monitoring
Launch template
Edit
Launch template
lt-0ee4256f4b91d3b0f
MyApp-LT
AMI ID
ami-04093f13c66e343ea
Instance type
t2.micro
Owner
arn:aws:iam::013623162068:root
Version
Default
Security groups
-
Security group IDs
sg-0eb681779c3704ac0
Create time
Sat Sep 13 2025 21:43:08 GMT+0530 (India Standard Time)
Description
-
Storage (volumes)
-
Key pair name
-
Request Spot Instances
No
View details in the launch template console


Device simulation changes require a reload to fully apply. Automatic reloads are disabled by default to avoid losing any changes in DevTools. You can enable reloading via the Settings menu.
EC2 > Auto Scaling groups > MyApp-ASG
ScaleIn_CPU70
Policy type
Step scaling
Enabled or disabled
Enabled
Execute policy when
CPULow_Alarm
breaches the alarm threshold: CPUUtilization < 70 for 1 consecutive periods of 60 seconds for the metric dimensions:
AutoScalingGroupName = My-ASG-Project
Take the action
Remove 1 capacity units when 70 >= CPUUtilization > -infinity
ScaleOut_CPU80
Policy type
Step scaling
Enabled or disabled
Enabled
Execute policy when
CPUHigh_Alarm
breaches the alarm threshold: CPUUtilization > 80 for 1 consecutive periods of 60 seconds for the metric dimensions:
AutoScalingGroupName = My-ASG-Project
Take the action
Add 1 capacity units when 80 <= CPUUtilization < +infinity
Instances need
200 seconds to warm up after each step

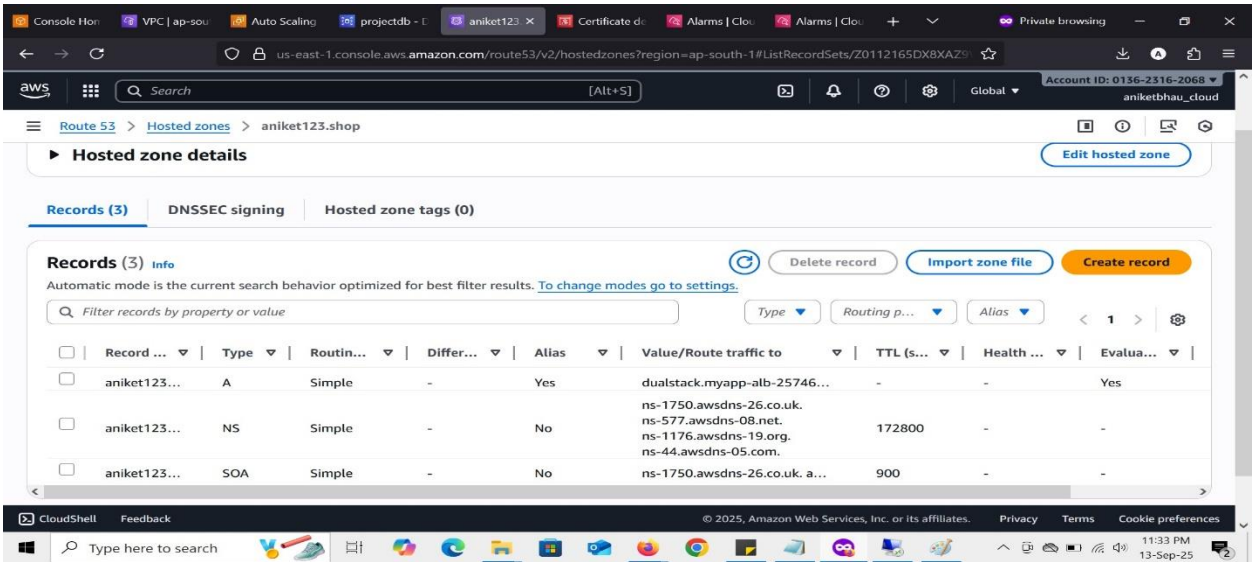

Device simulation changes require a reload to fully apply. Automatic reloads are disabled by default to avoid losing any changes in DevTools. You can enable reloading via the Settings menu.
EC2 > Auto Scaling groups > MyApp-ASG
Auto Scaling
EC2
All times shown are in UTC.
View all CloudWatch metrics
Investigate with AI - new
3h
1d
1w
UTC timezone
Explore related
CPU Utilization (Percent)
Disk Reads (Bytes)
Disk Read Operations (Opera
Disk Writes (Bytes)
15.8
8.29
0.764
15:01
18:00
1
0.5
0



6. Route 53 — Domain Configuration (DNS Management)

1. Domain Purchase:
   a. I purchased the custom domain aniket123.shop from GoDaddy.
   b. Then, I connected it to AWS Route 53 by updating the GoDaddy Name Servers to Route 53's name servers.
2. Hosted Zone:
   a. Created a Public Hosted Zone in Route 53 for aniket123.shop.
   b. This Hosted Zone acts as the DNS manager for my entire domain.
3. Records Configuration:
   a. A Record (Alias):
      1. Name: aniket123.shop
      2. Type: A (Alias)
      3. Value: Application Load Balancer DNS name (e.g., myapp-alb-123456.ap-south-1.elb.amazonaws.com)
      4. Purpose: Routes all incoming web traffic from the domain directly to the ALB.
   a. CNAME Record (Optional):
      1. Name: www.aniket123.shop
      2. Value: ALB DNS name
      3. Purpose: Allows both aniket123.shop and www.aniket123.shop to resolve to the same ALB endpoint.

Explanation (Mentor-level tone):
Route 53 is responsible for domain name resolution — it translates the human-readable name (aniket123.shop) into the ALB's IP address, so end-users can reach the application through a clean and professional domain name.



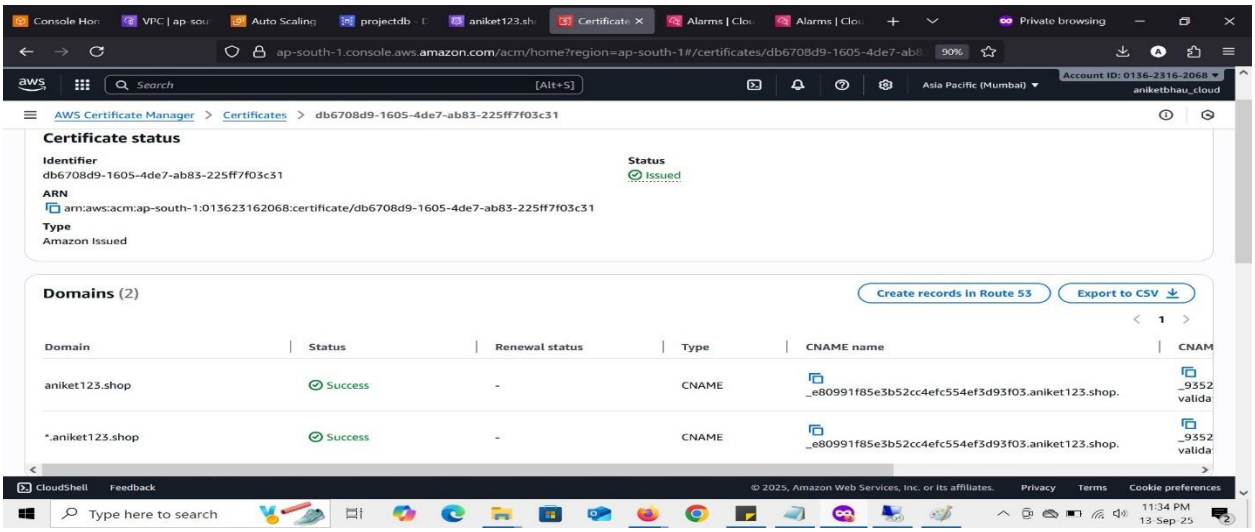
7. ACM — SSL Certificate Configuration (HTTPS Security)

Once the domain was working through Route 53, I used AWS Certificate Manager (ACM) to issue an SSL certificate, ensuring that all traffic between clients and the ALB is encrypted and secure.

1. Service: AWS Certificate Manager (ACM)
2. Certificate Request Details:
3. Domain Names:
4. aniket123.shop
5. *.aniket123.shop (wildcard certificate for all subdomains)
6. Validation Method: DNS Validation
7. ACM automatically created validation records in Route 53.
8. After verification, the certificate status changed to “Issued”
9. Attached To: Application Load Balancer (HTTPS listener, port 443)

Explanation (Friend-level tone):

ACM works hand-in-hand with Route 53. When you request a certificate, AWS automatically adds special CNAME validation records inside Route 53. Once validated, your site gets the green padlock (HTTPS) in the browser — meaning all traffic to your site is fully encrypted.

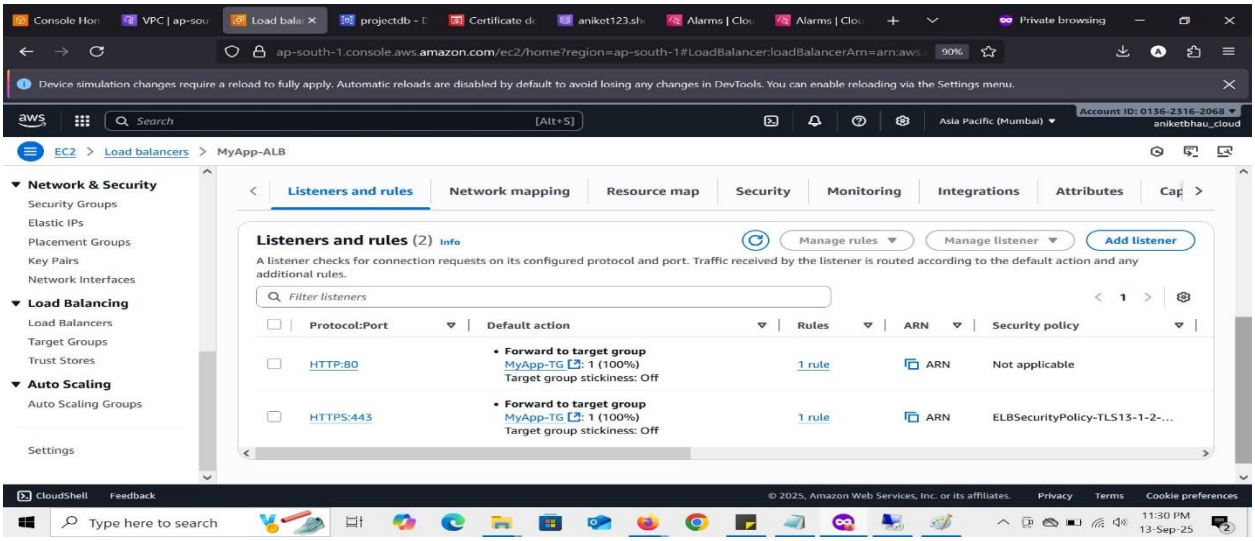


8. Update ALB Listeners — Enable HTTPS Traffic

Once the ACM certificate was issued, I configured my Application Load Balancer (ALB) to use HTTPS:
Steps:

1. Open the ALB → Navigate to Listeners
2. Add a new listener for HTTPS (Port 443)
3. Attach ACM Certificate: aniket123.shop
4. Action: Forward all HTTPS traffic to Target Group (MyApp-TG)
5. Optionally:
   a. Remove HTTP (port 80) listener
   b. Or keep it and create a redirect rule from HTTP → HTTPS

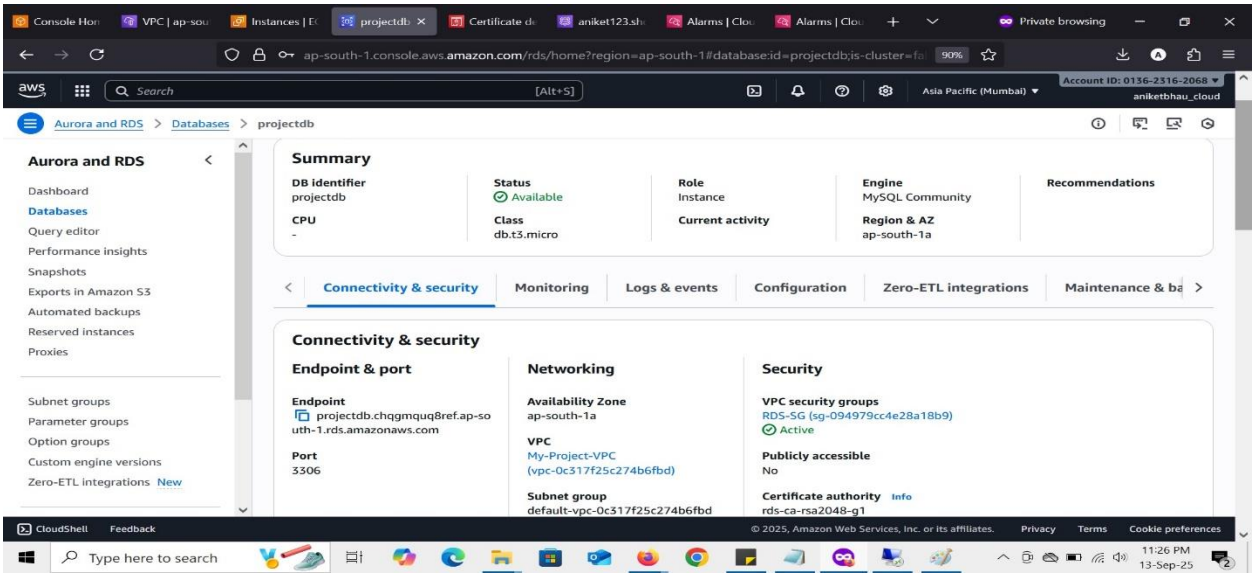
Result:
My application became securely accessible at:
🔗 https://aniket123.shop



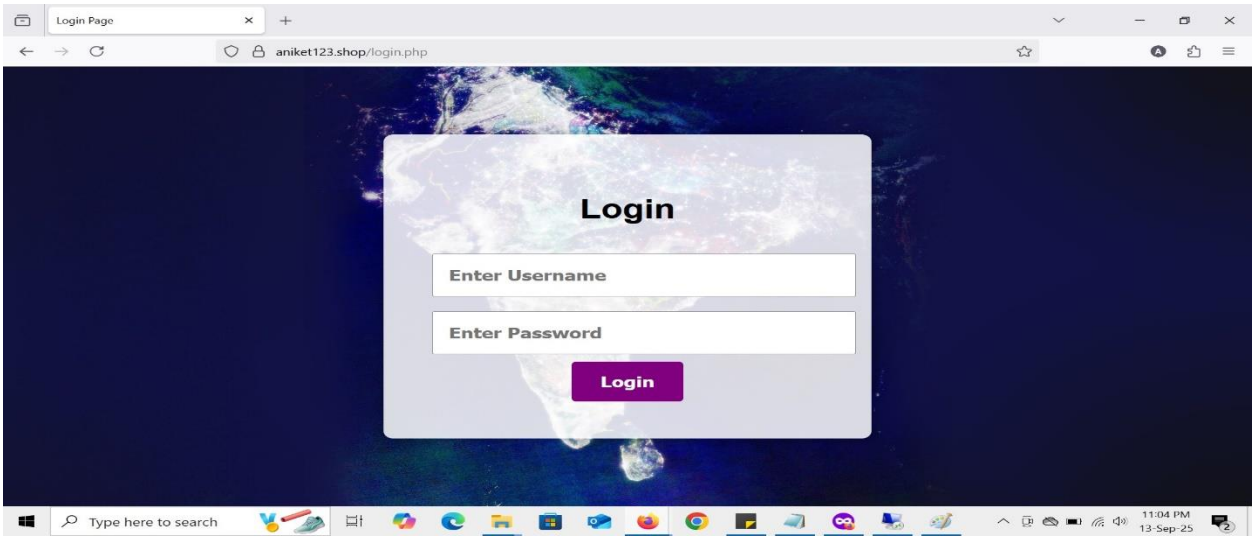
9. Amazon RDS (MySQL)

1. Engine: MySQL 8.x
2. Deployment: Multi-AZ enabled
3. Instance Type: db.t3.micro
4. Subnet Group: Private DB subnets
5. Backup Retention: 7 days
6. Encryption: Enabled
7. Access: Only from App-SG (no public access)

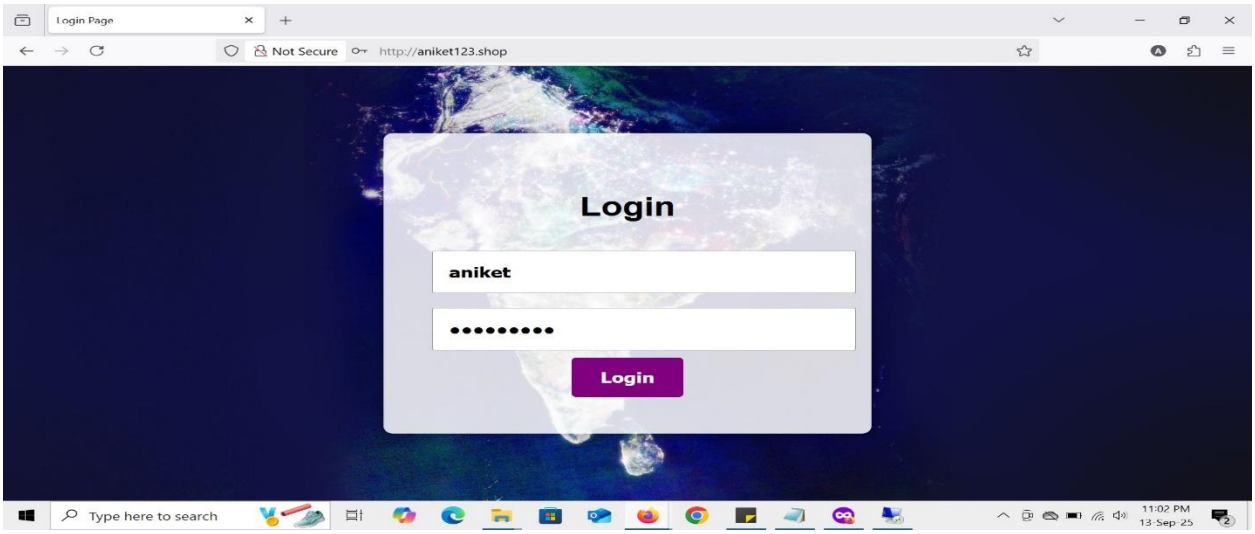
Why RDS?
It eliminates manual database maintenance — backups, failover, patching are managed by AWS.
This improves reliability while reducing admin overhead.

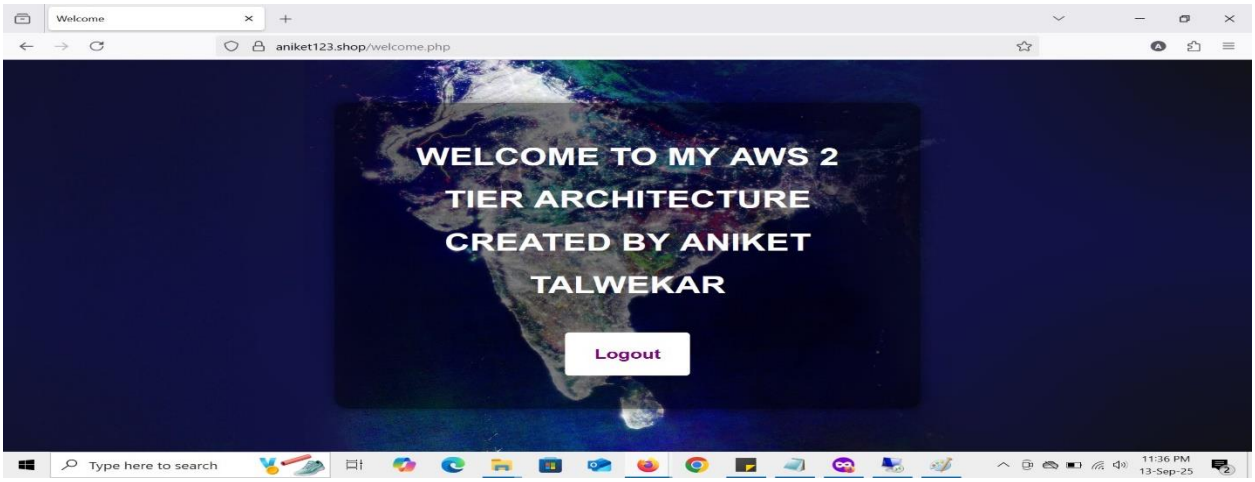


10. Domain Test: ✅
1. https://aniket123.shop → Opens Login Page directly

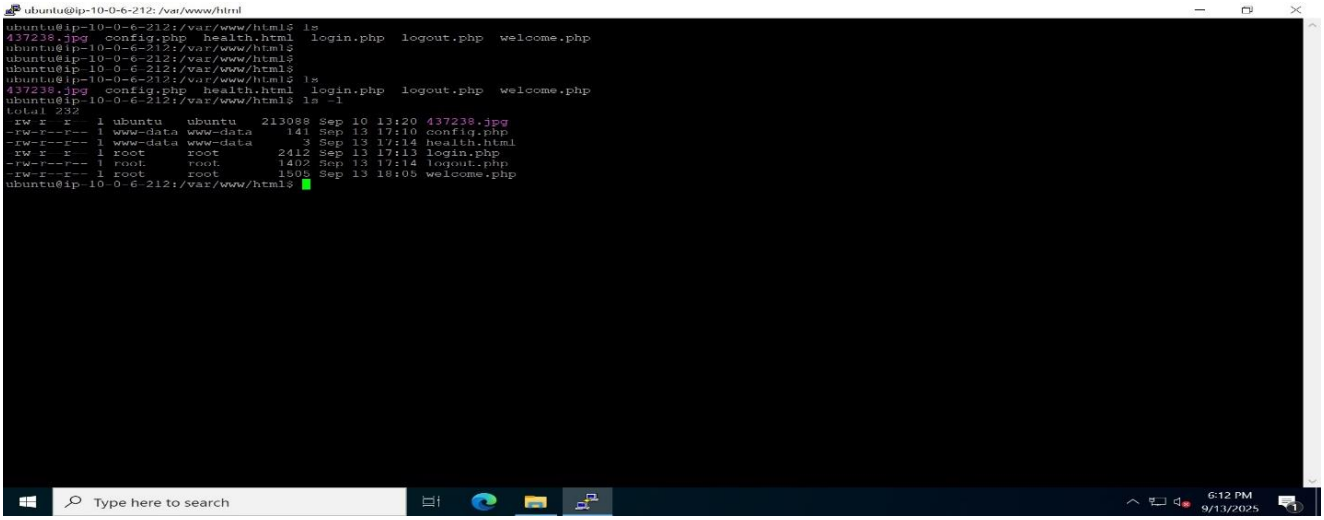


11. Login Test: ✅
    1. Enter valid user (from RDS users table) → Redirects to welcome.php



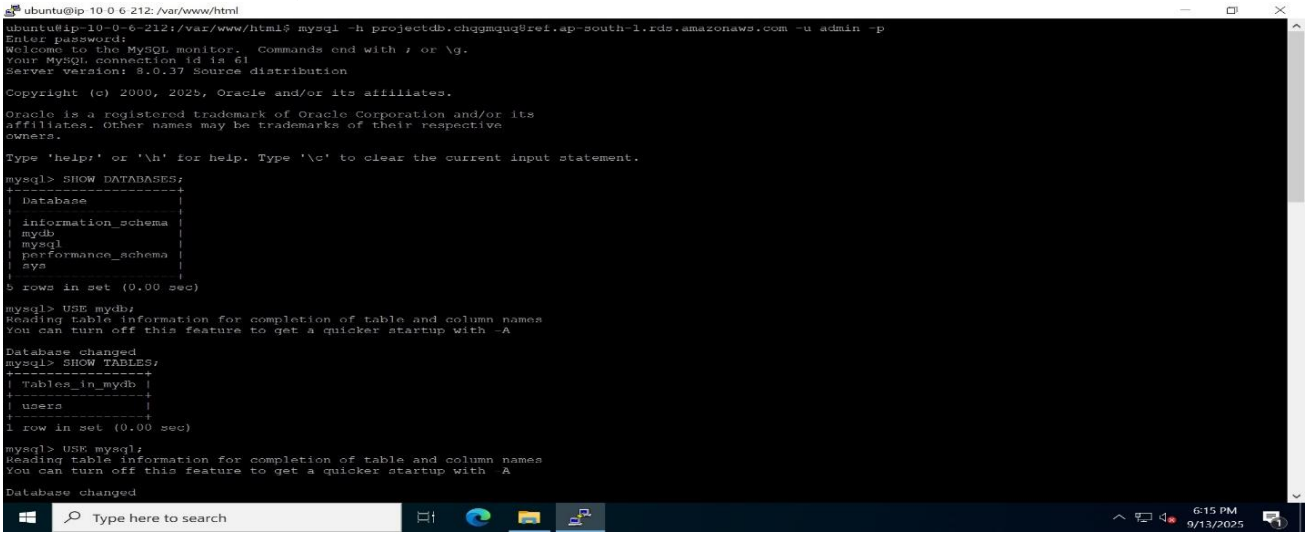


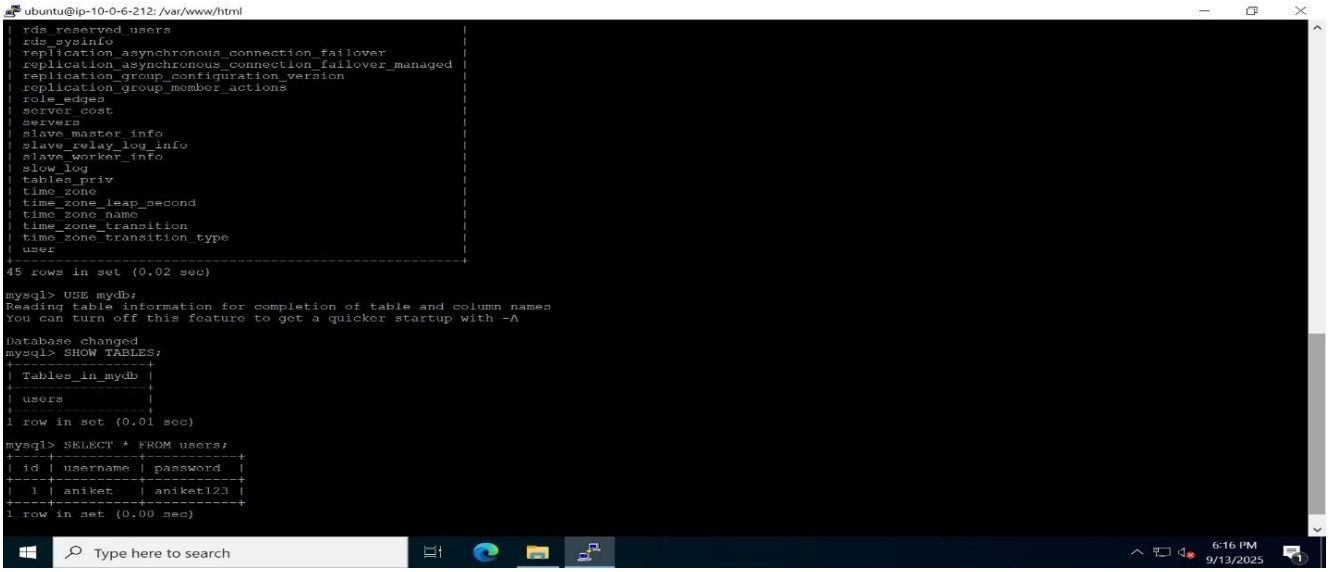
12. Web Application (PHP)
    1. Add list of PHP files (login.php, welcome.php, logout.php, config.php)



13. Database (RDS)

1. screenshot of RDS creation summary
2. screenshot of MySQL CLI:
3. show databases;
4. use mydb;
5. select * from users;




ubuntu@ip-10-0-6-212: /var/www/html
| rds_reserved_users |
| rds_sysinfo |
| replication_asynchronous_connection_failover |
| replication_asynchronous_connection_failover_managed |
| replication_group_configuration_version |
| replication_group_member_actions |
| role_edges |
| server_cost |
| servers |
| slave_master_info |
| slave_relay_log_info |
| slave_worker_info |
| slow_log |
| tables_priv |
| time_zone |
| time_zone_leap_second |
| time_zone_name |
| time_zone_transition |
| time_zone_transition_type |
| user |
45 rows in set (0.02 sec)
mysql> USE mydb;
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> SHOW TABLES;
| Tables_in_mydb |
| users |
1 row in set (0.01 sec)
mysql> SELECT * FROM users;
| id | username | password |
| 1 | aniket | aniket123 |
1 row in set (0.00 sec)
Type here to search
6:16 PM
9/13/2025