## 🔐 Security & Key Pair Configuration

Here first we will cover , the setup of security groups and key pairs for deploying the vProfile application on AWS using the Lift and Shift strategy. All configurations follow AWS best practices to ensure least privilege access, security, and scalability.

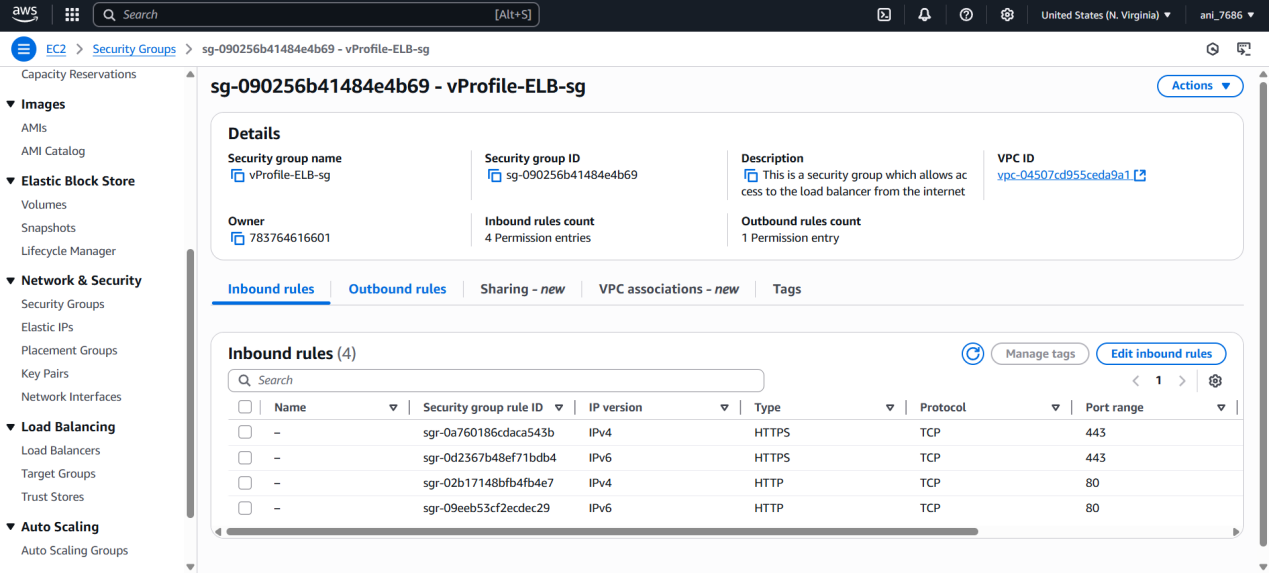
### 🔐 1. Security Group: vprofile-ELB-sg (Load Balancer)

**Purpose:** Allows public internet traffic to reach the Application Load Balancer.

**Inbound Rules:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Protocol** | **Port** | **Source** | **Description** |
| HTTP | 80 | Anywhere | Public access (IPv4 & IPv6) |
| HTTPS | 443 | Anywhere | Secure access (IPv4 & IPv6) |

⚠️ Do not modify outbound rules.



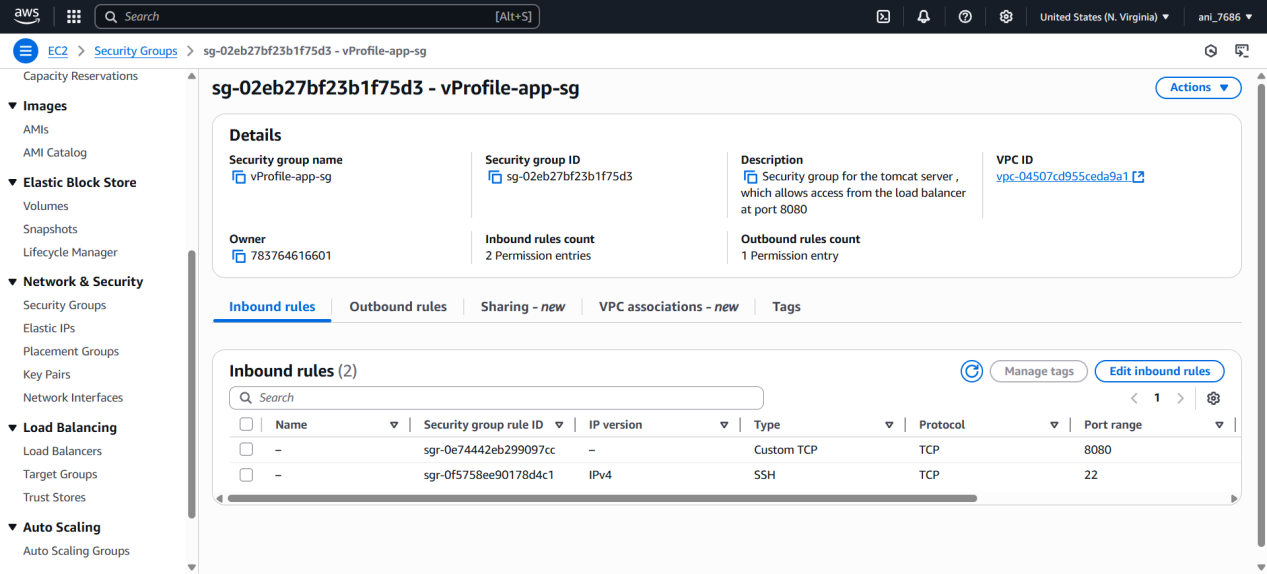
### 🔐 2. Security Group: vprofile-app-sg (Tomcat Instances)

**Purpose:** Allows traffic from the Load Balancer and SSH access for management.

**Inbound Rules:**

|  |  |  |  |
| --- | --- | --- | --- |
| Protocol | Port | Source | Description |
| TCP | 8080 | vprofile-ELB-SG | Application traffic from Load Balancer |
| SSH | 22 | My IP | Admin SSH access |

💡 Update the SSH rule when your IP changes.



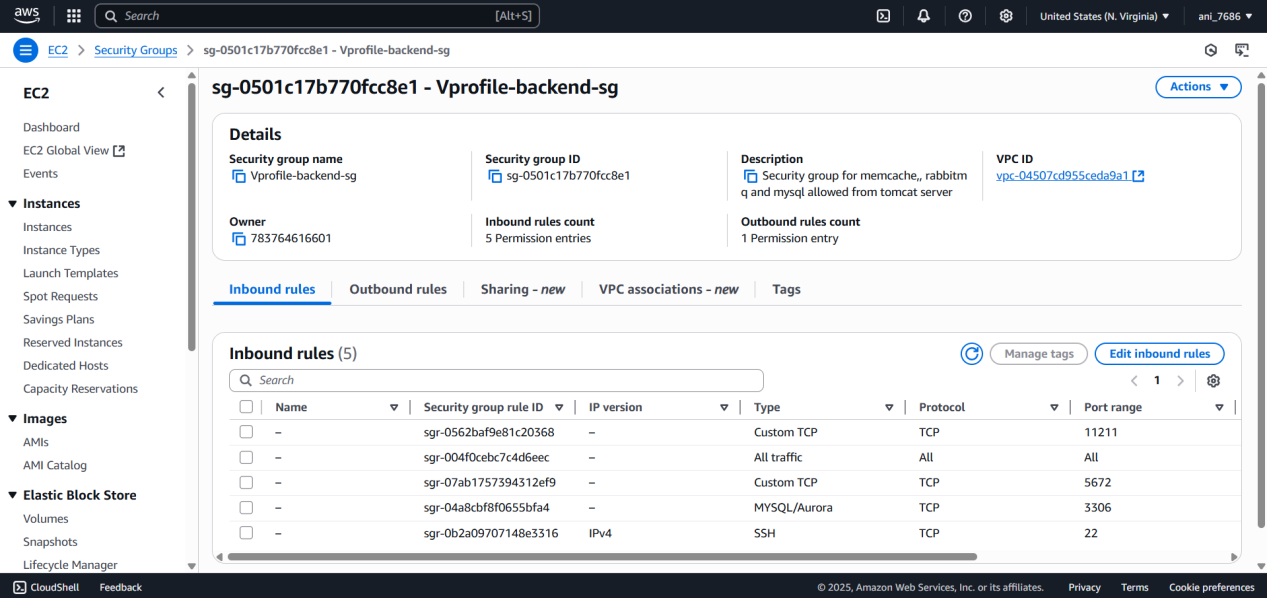
### 🔐 3. Security Group: vprofile-backend-sg (MySQL, Memcache, RabbitMQ)

**Purpose:** Controls backend service access from the application layer.

**Inbound Rules:**

|  |  |  |  |
| --- | --- | --- | --- |
| Protocol | Port | Source | Service |
| TCP | 3306 | vprofile-app-sg | MySQL |
| TCP | 11211 | vprofile-app-sg | Memcached |
| TCP | 5672 | vprofile-app-sg | RabbitMQ |
| SSH | 22 | My IP | SSH access |
| ALL | ALL | vprofile-backend-sg | Internal backend-to-backend communication |

✅ Internal traffic rule allows instances in this group to communicate freely.



### 🔑 Key Pair: vprofile-prod-key

Created in .pem format (for Git Bash / Terminal).

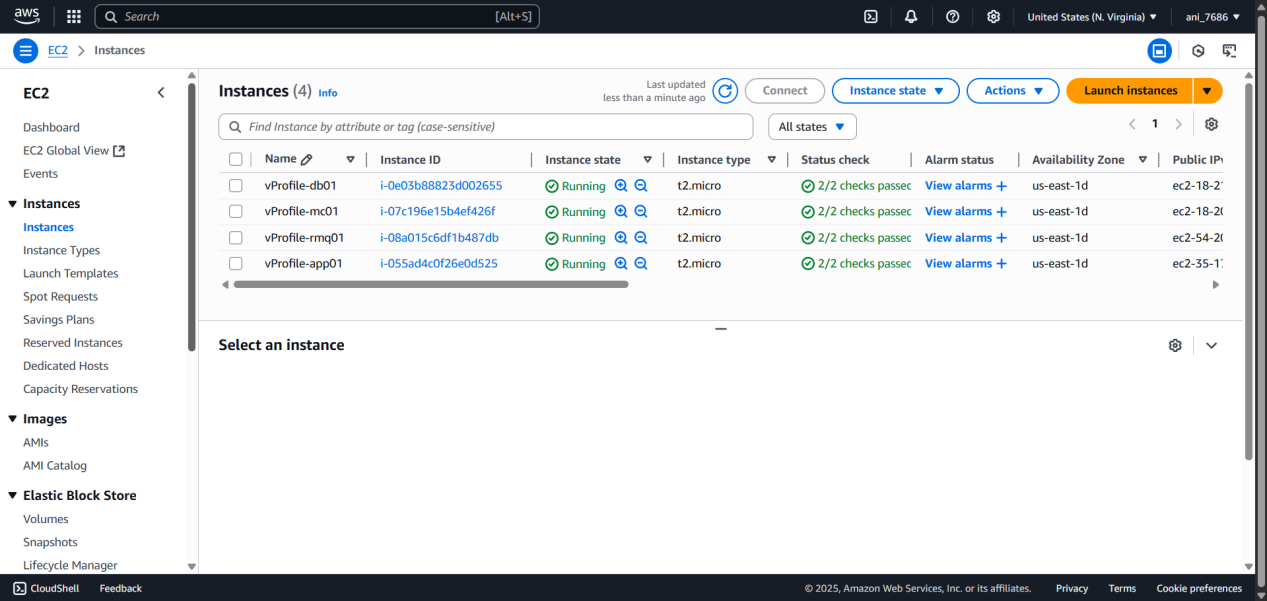
Used for secure SSH access to EC2 instances.

File should be kept **secure and private**.

## AWS EC2 Instance Setup and Service Verification

1.) We set up **four EC2 instances**, each with specific responsibilities and services using **user data scripts**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Instance Name** | **Purpose** | **AMI Used** | **Script Used** | **Security Group** |
| vprofile-db01 | MySQL DB | Amazon Linux 2023 | mysql.sh | vprofile-backend-sg |
| vprofile-mc01 | Memcached | Amazon Linux 2023 | memcache.sh | vprofile-backend-sg |
| vprofile-rmq01 | RabbitMQ | Amazon Linux 2023 | rabbitmq.sh | vprofile-backend-sg |
| vprofile-app01 | Tomcat Server | Ubuntu 24.04 LTS | tomcat\_ubuntu.sh | vprofile-app-sg |



### 🧾 Setup Steps Summary

**Repository Clone:**

Cloned from https://github.com/hkhcoder/vprofile-project into local machine.

Switched to aws-liftandshift branch.

**User Data Scripts Used:**

mysql.sh: Installs MariaDB, sets up DB, user, schema.

memcache.sh: Installs Memcached, enables remote access.

rabbitmq.sh: Downloads repo config, installs Erlang and RabbitMQ, configures admin user.

tomcat\_ubuntu.sh: Installs Tomcat10 on Ubuntu via APT.

**2.) Tags Added to Each Instance and Volume:**

Name: Based on role (e.g., vprofile-db01)

Project: vprofile

**3.) Security Group Mappings:**

App SG → for Tomcat (port 8080 + SSH from IP)

Backend SG → for MySQL (3306), Memcache (11211), RabbitMQ (5672) + SSH

Load Balancer SG → for HTTP/HTTPS

### 🧪 4.) Post-Launch Verification

✅ **vprofile-db01**:

Logged in using ssh -i vprofile-prod-key.pem ec2-user@<db01-public-ip>

Verified MariaDB: systemctl status mariadb

Logged into MySQL: mysql -u admin -p → DB: accounts → Tables verified

✅ **vprofile-mc01**:

Logged in using ssh

Verified Memcached: systemctl status memcached

✅ **vprofile-rmq01**:

Logged in using ssh

Verified RabbitMQ: systemctl status rabbitmq-server

✅ **vprofile-app01**:

Script installs Tomcat10 on Ubuntu 24.04

Will verify in next steps after deployment

### 🌐 Private DNS with Route 53

To ensure our **vprofile application** in vprofile-app01 (Tomcat instance) can reliably connect to backend services (**MySQL**, **Memcache**, and **RabbitMQ**), we use **private DNS** entries instead of hardcoding IP addresses.

#### 🧠 Why Private DNS?

IPs can change when instances are terminated and relaunched.

Using private DNS (like db01.vprofile.internal) provides **dynamic resolution**.

No need to modify application.properties or redeploy the app when IPs change.

#### \*\*\* application.properties (Excerpt)

db01=3306

mc01=11211

rmq01=5672

Here, db01, mc01, and rmq01 must resolve to the **private IPs** of the respective backend EC2 instances.

### \*\*\* Steps to Create Private Hosted Zone in Route 53

1. **Navigate to Route 53 → Hosted Zones → Create Hosted Zone**

**Domain Name**: vprofile.internal

**Type**: Private Hosted Zone for Amazon VPC

**VPC Selection**: Select the same VPC used for your EC2 instances

1. **Create DNS Records** for each backend service:

| **Record Name** | **Record Type** | **Value (Private IP of Instance)** | **Description** |
| --- | --- | --- | --- |
| db01 | A | Private IP of vprofile-db01 | MySQL Hostname |
| mc01 | A | Private IP of vprofile-mc01 | Memcache Hostname |
| rmq01 | A | Private IP of vprofile-rmq01 | RabbitMQ Hostname |

1. **Verify DNS Resolution (from app instance)**

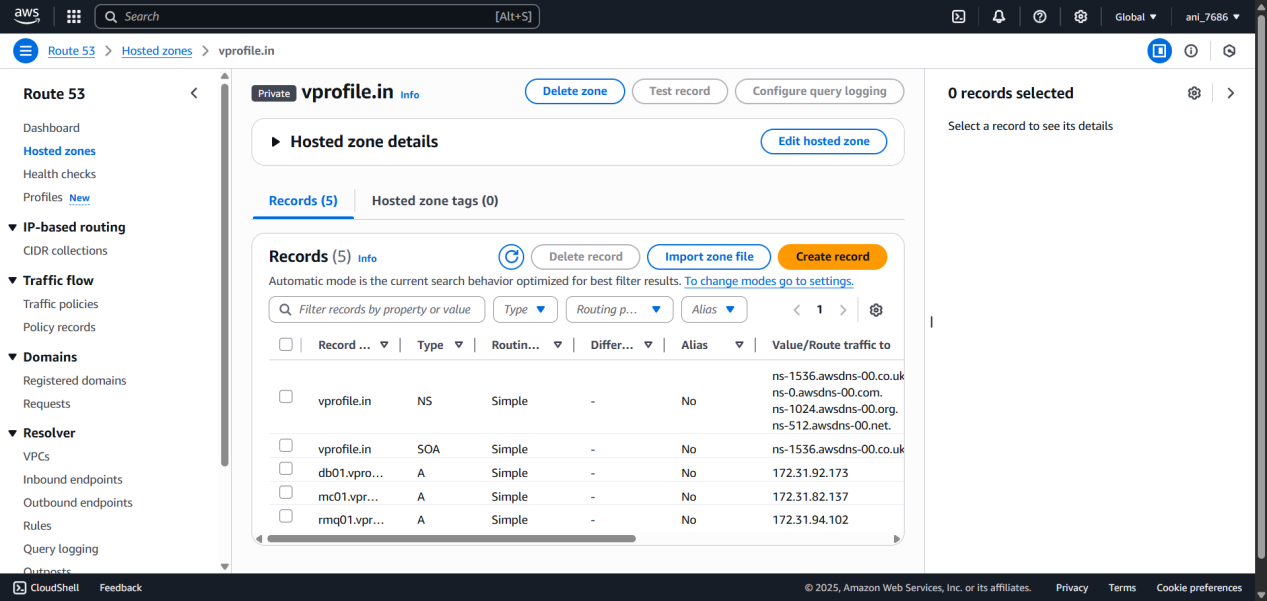
SSH into vprofile-app01 and test:

ping db01.vprofile.internal

ping mc01.vprofile.internal

ping rmq01.vprofile.internal

You should see the private IP resolving correctly for each.



✅ Benefits

Seamless failover and recreation of instances

No need to redeploy application on IP change

Easy environment separation (e.g., vprofile-dev.internal, vprofile-prod.internal)

**AWS Java Artifact Deployment using Maven, S3, and EC2 Tomcat**

To build a Java application (.war file) using Maven on your local computer, upload it to an AWS S3 bucket, and deploy it on a Tomcat EC2 instance using AWS CLI and IAM for secure communication.

## Tools & Technologies Required

|  |  |
| --- | --- |
|  |  |
| Maven | Build Java project into a .war file |
| JDK 17 | Required for Maven and Java build |
| AWS CLI | To interact with AWS services (S3, EC2) from the command line |
| S3 Bucket | Temporary storage for your artifact |
| IAM User | Authenticate your computer to access AWS |
| IAM Role | Authenticate EC2 instance to access AWS services |
| EC2 (Ubuntu) | Server where Tomcat is installed |
| Tomcat 10 | Application server to run the Java app |

## 📁 Structure of the Process

**AWS Setup**

Create S3 Bucket

Create IAM User (for your local machine)

Create IAM Role (for EC2)

**Local Computer Setup**

Install required tools

Build artifact with Maven

Upload .war to S3 using AWS CLI

**EC2 Server Deployment**

Install AWS CLI

Download artifact from S3

Deploy to Tomcat

**Security Concepts Explained**

## 🔐 IAM vs EC2 Root User: Why We Need IAM

### 🤔 Common Confusion:

"If I am root user on the EC2 instance, why do I need IAM roles?"

Being root on EC2 only gives you **local machine power**. You can install software, delete files, run services — but AWS services like **S3, DynamoDB, etc. require AWS permissions**, not Linux permissions.

|  |  |
| --- | --- |
| Role | What it controls |
| **Root user (Linux)** | Superuser on the machine only |
| **IAM Role (AWS)** | Controls access to AWS services |

### 📦 Example:

You’re root on EC2 but cannot run:

aws s3 cp s3://my-bucket/myfile.txt .

Unless EC2 has an IAM Role attached with S3ReadAccess

### ✅ Local Machine Needs IAM User

To push .war to S3, use AWS CLI with IAM User keys

### ✅ EC2 Instance Needs IAM Role

To download .war from S3 securely without hardcoding keys

### 🛡️ Why IAM Roles Are Better:

No need to store access keys in EC2

Temporary credentials auto-rotated by AWS

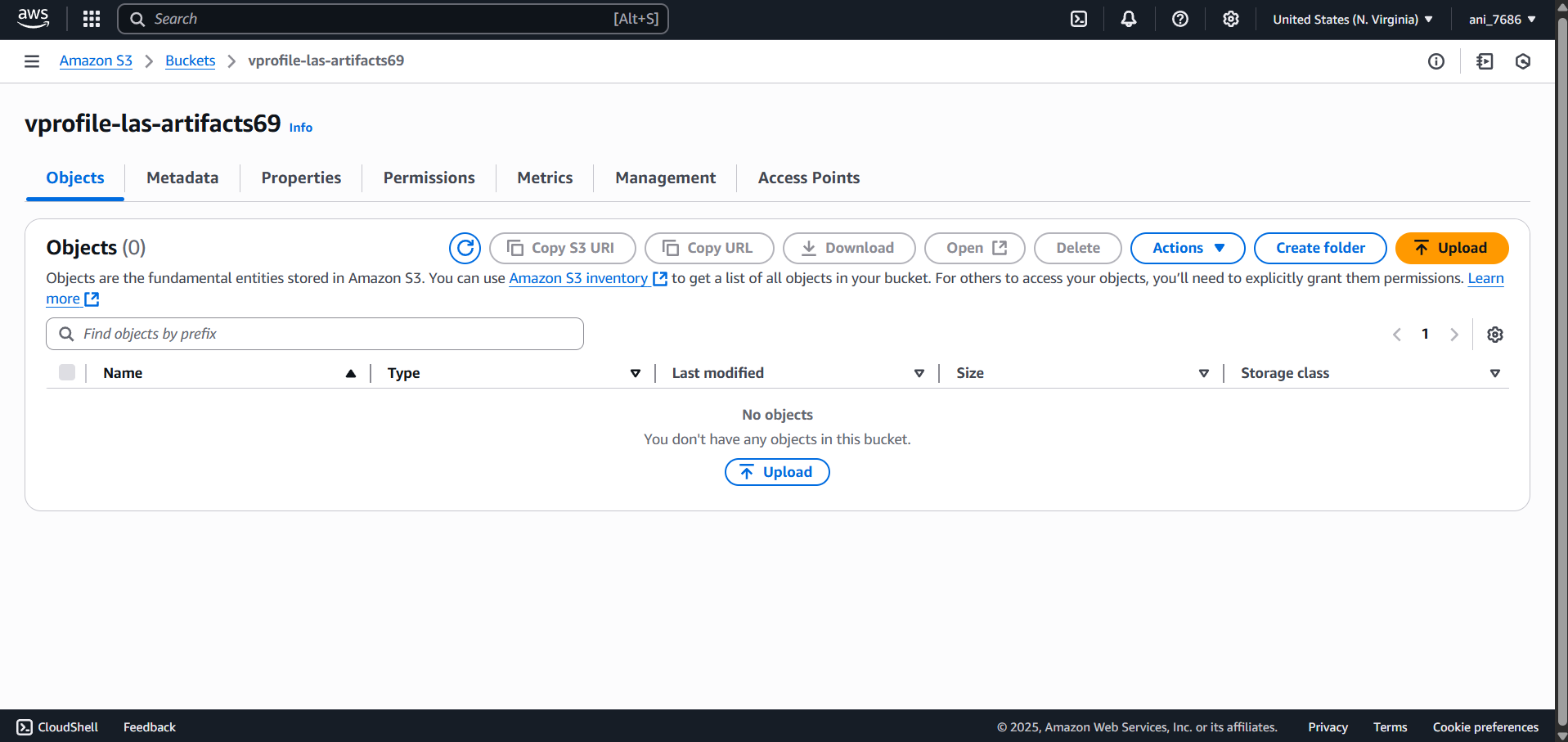
Safer and more manageable

## 🛠️ Step-by-Step Deployment Guide

### 1.) AWS Setup

#### a. Create S3 Bucket

Go to AWS Console → S3 → Create bucket → Name: vprofile-las-artifacts

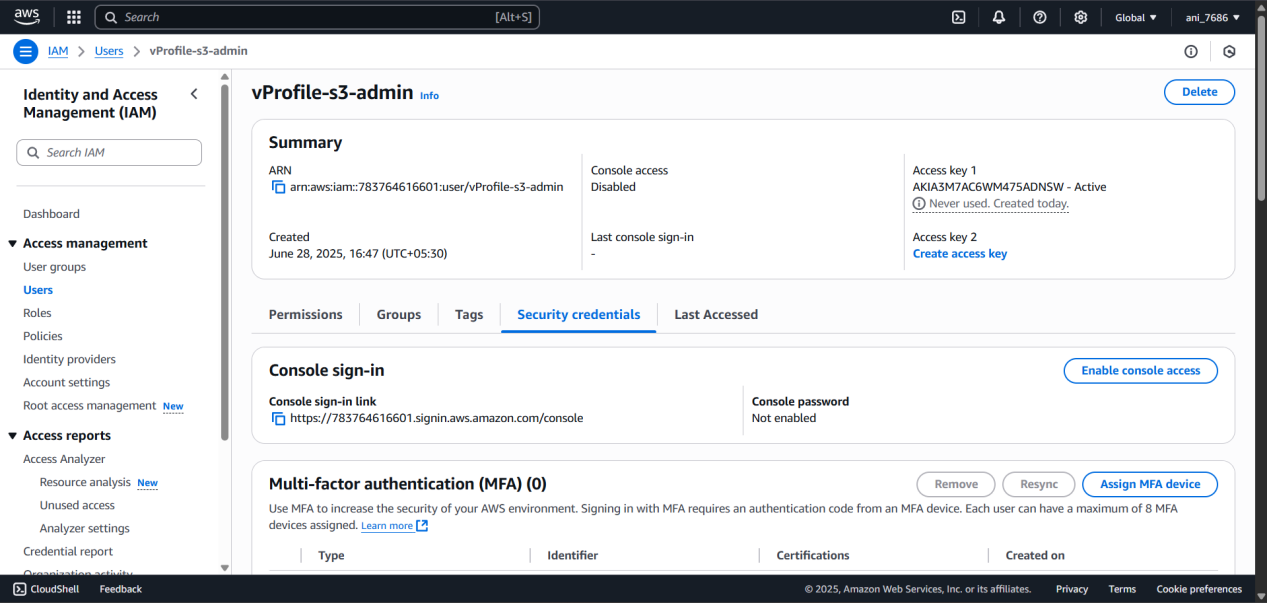


#### b. Create IAM User

IAM → Users → Create User → vprofile-s3-admin

Attach policy: AmazonS3FullAccess

Generate & download access keys



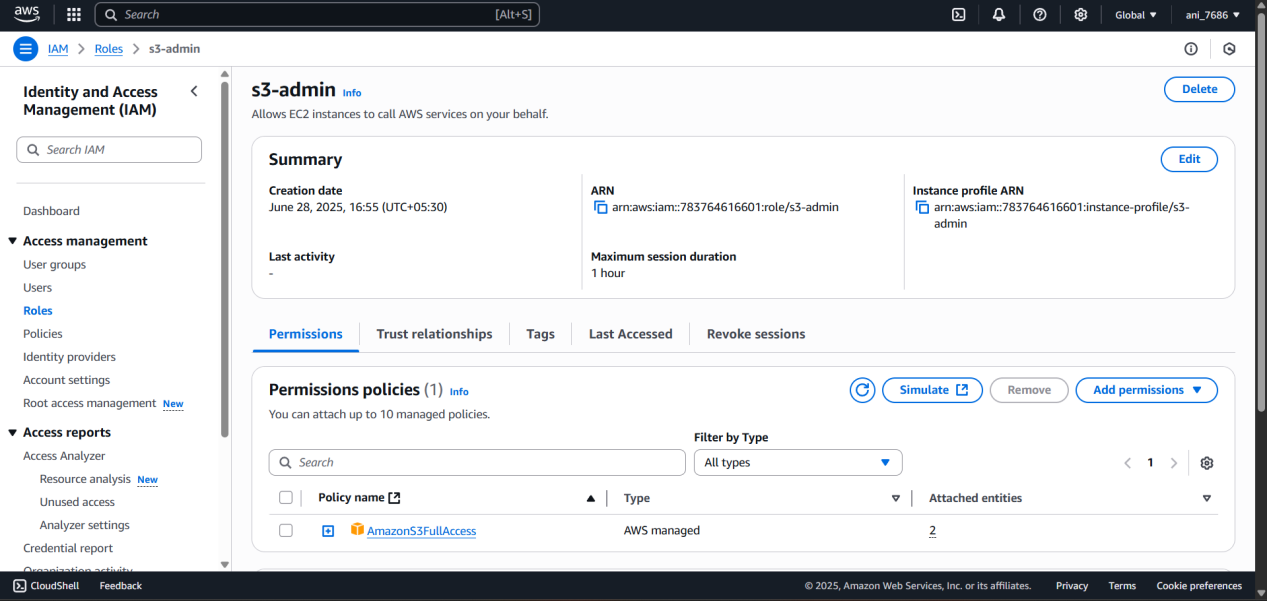
#### c. Create IAM Role for EC2

IAM → Roles → Create Role → Use Case: EC2

Attach policy: AmazonS3FullAccess

Name: S3-admin

Attach this role to your EC2 instance



### 2.) On Your Computer

#### a. Install Requirements

Install Maven, JDK 17, AWS CLI

#### b. Configure AWS CLI

aws configure

Enter your access key, secret key, region (e.g., us-east-1), format (e.g., json)

#### c. Update application.properties

Inside src/main/resources:

db.host=db01.vprofile.in

mc.host=mc01.vprofile.in

rmq.host=rmq01.vprofile.in

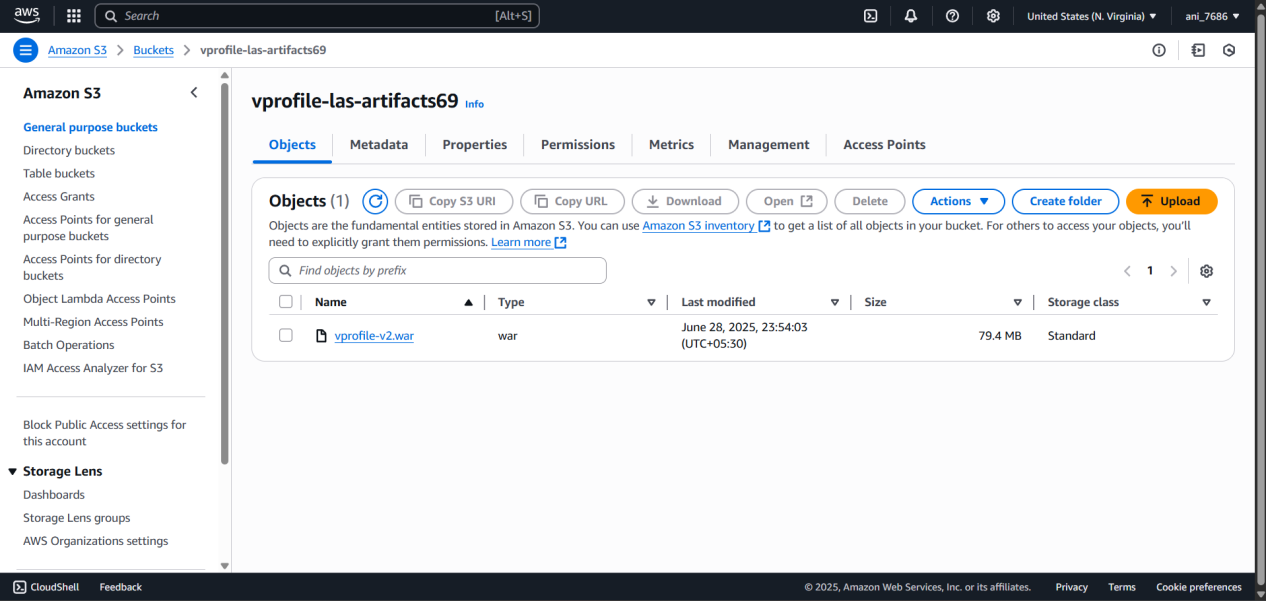
#### d. Build Artifact with Maven

mvn install

Output file: target/vprofile-v2.war

#### e. Upload Artifact to S3

aws s3 cp target/vprofile-v2.war s3://vprofile-las-artifacts69



### 3.) On EC2 Tomcat Server

#### a. SSH into EC2

ssh -i key.pem ubuntu@<your-public-ip>

#### b. Install AWS CLI

sudo snap install aws-cli --classic

#### c. Download Artifact from S3

aws s3 cp s3://vprofile-las-artifacts69/vprofile-v2.war /tmp/

#### d. Deploy to Tomcat

sudo systemctl stop tomcat10

sudo rm -rf /var/lib/tomcat10/webapps/ROOT

sudo cp /tmp/vprofile-v2.war /var/lib/tomcat10/webapps/ROOT.war

sudo systemctl start tomcat10

**AWS Load Balancer and Service Validation for VProfile App**

### Step 1: Validate Tomcat Application on EC2 (app01)

**Check Tomcat Application:**

Go to EC2 instance app01.

Retrieve **Public IP Address**.

**Security Group Configuration:**

Go to app security group (attached to app01).

Edit **Inbound Rules**:

Allow **port 8080** from **Load Balancer Security Group**.

Add another rule to allow **port 8080** from **your IP** (for testing).

**Test Application in Browser:**

Visit: http://<Public-IP>:8080

You should see the **login page** of the VProfile app.

This is a temporary check. Final setup will be accessed via Load Balancer.

### Step 2: Create Target Group

**Navigate to EC2 > Load Balancing > Target Groups**

**Create a new Target Group**:

Target type: **Instances**

Name: vprofile-las-tg

Protocol: **HTTP**

Port: **8080**

**Health Check Settings:**

Override default port with **8080**

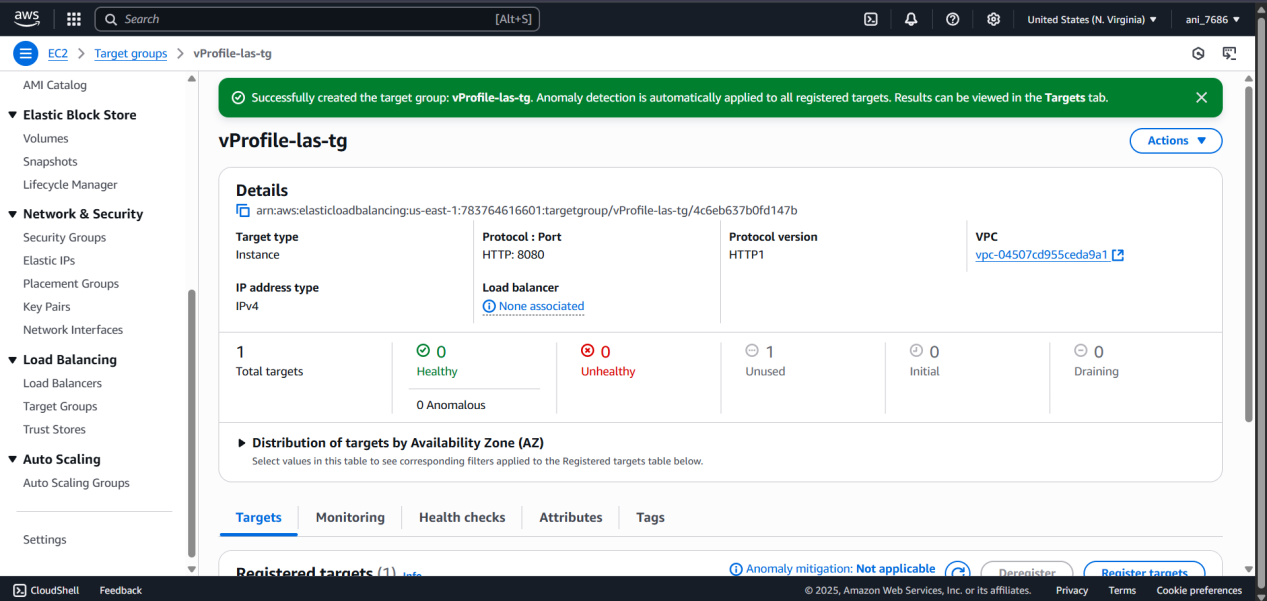
**Register Target Instance:**

Select instance app01

Confirm port: **8080**

Click: Include as pending below

Click: **Create Target Group**

****

### Step 3: Create Load Balancer (Application Load Balancer)

**Navigate to EC2 > Load Balancers > Create Load Balancer**

**Choose Application Load Balancer (ALB)**

**Basic Configuration:**

Name: vprofile-las-elb

Scheme: Internet-facing

IP Type: IPv4

**Network Mapping:**

Select your VPC

Select all **Availability Zones** (AZs)

**Security Group:**

Remove default

Attach **Load Balancer Security Group**

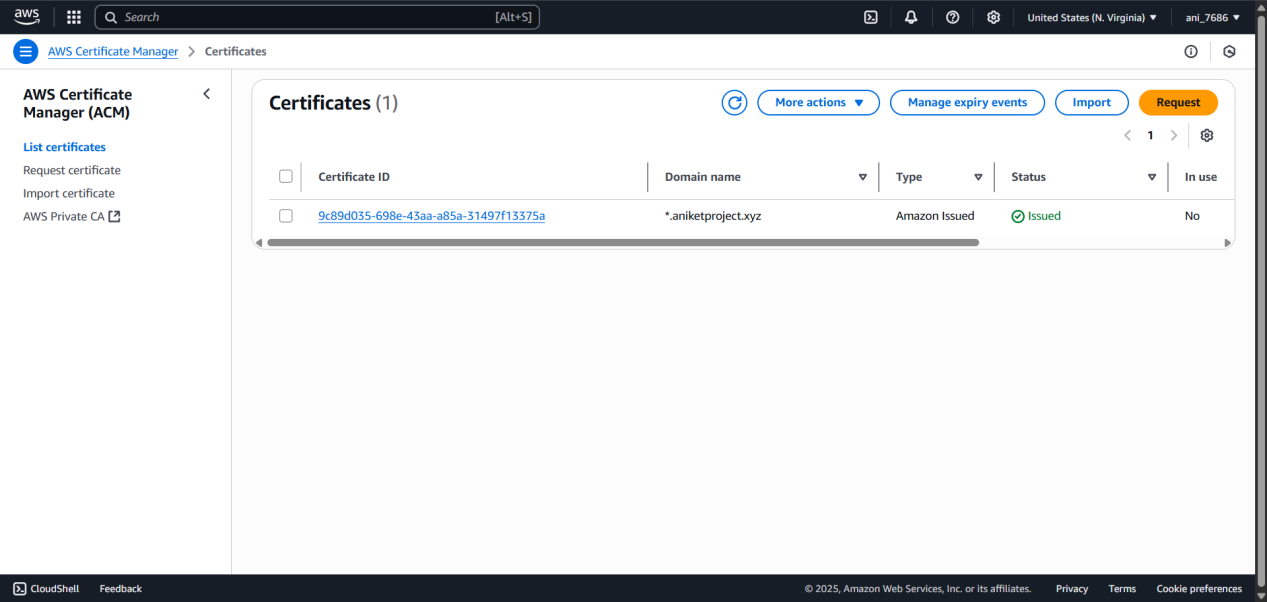
**Listeners and Routing:**

Listener 1: HTTP (port 80) ➔ Forward to vprofile-las-tg

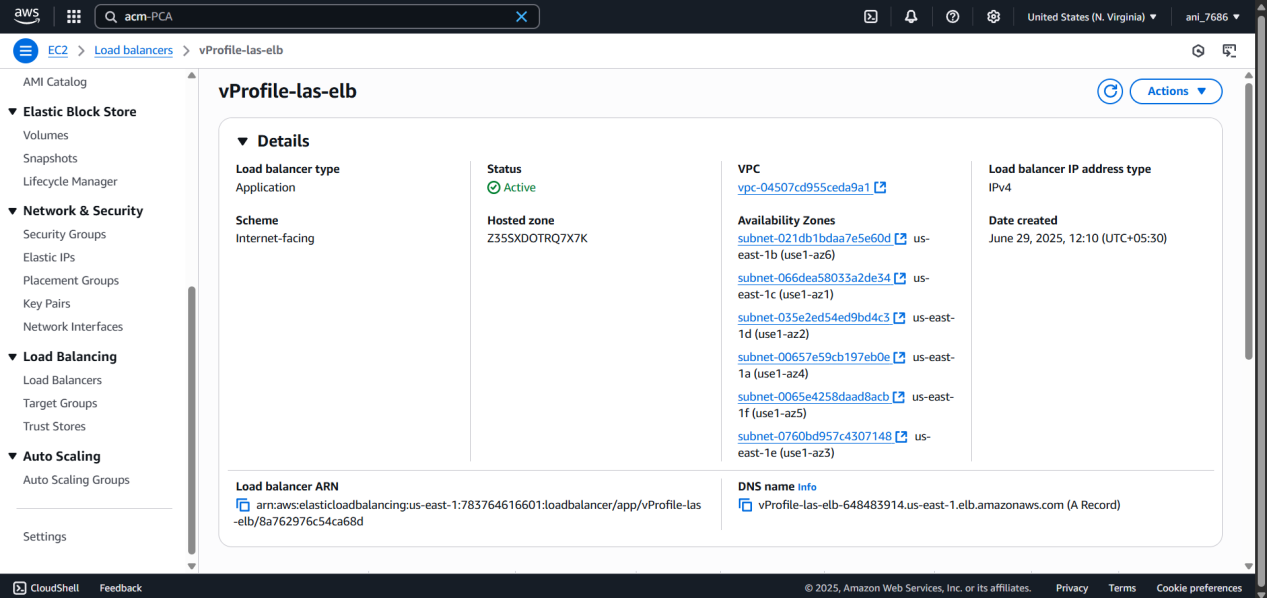
Optional: Add HTTPS (port 443) if ACM Certificate and Domain are configured

Add Listener: **HTTPS (443)**

Select **Certificate from ACM** (e.g. vprofileapp.aniketproject.xyz)



**Create Load Balancer**

****

### What’s happening in your setup:

**Tomcat** (your application server) is **running on port 8080** inside your EC2 instance.

Why? Because Tomcat by default runs on **8080** to avoid conflict with Apache or Nginx, which usually run on port 80.

Also, running on port 80 requires root privileges on Linux.

**Load Balancer** listens on:

**Port 80** for **HTTP traffic**

**Port 443** for **HTTPS traffic**

**Load Balancer forwards the request to target group (EC2 instances)** on **port 8080**

This means:

Client → sends request to LB on port 80/443

LB → forwards request to EC2 (Tomcat) on port 8080

### Step 4: Domain Configuration (Optional - Only If You Own a Domain)

For domain users (e.g., GoDaddy):

Go to your **Domain Provider** (e.g., GoDaddy).

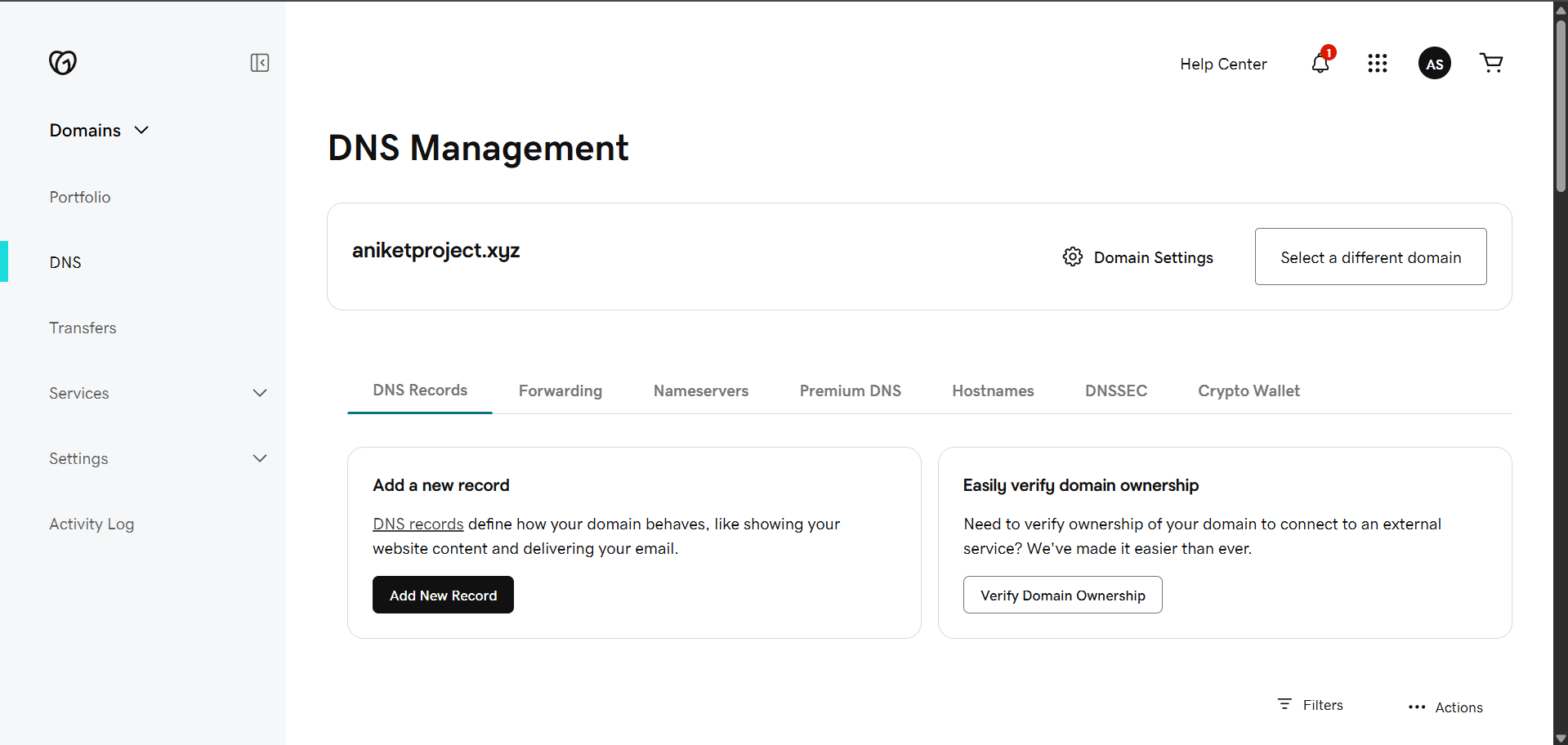
Navigate to **DNS Management**.

Create a **CNAME Record**:

Name: vprofileapp

Value: **Load Balancer DNS Name**

Full URL: vprofileapp.yourdomain.com



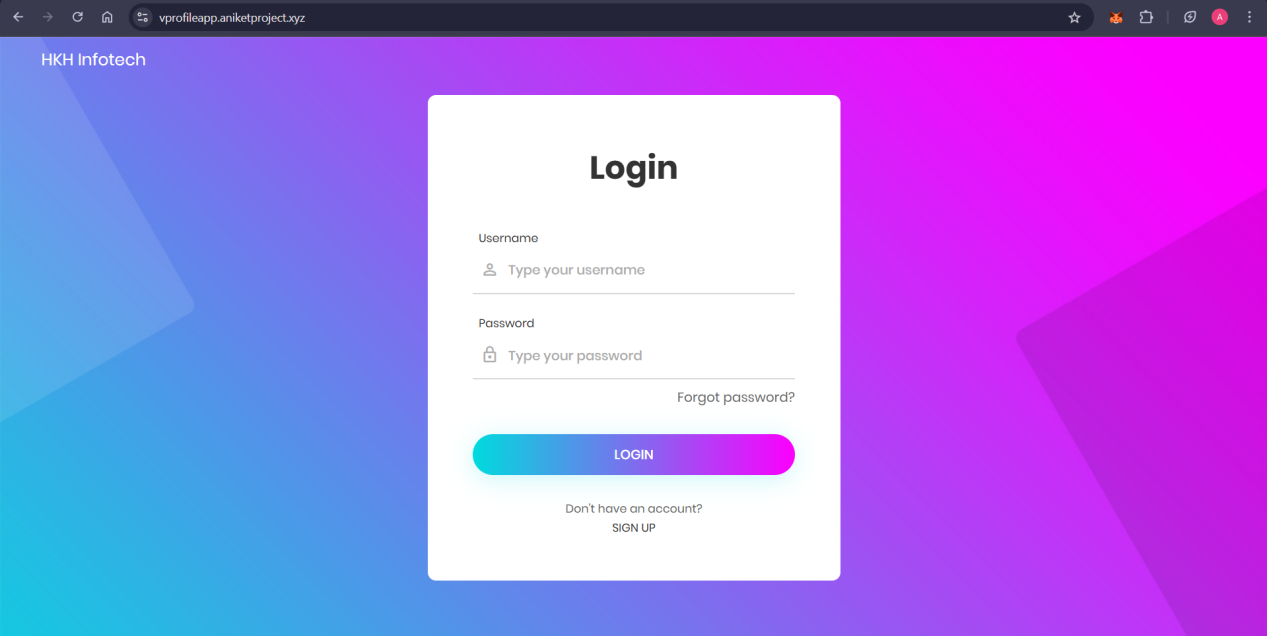
### Step 5: Validate Load Balancer

Wait until Load Balancer state = **Active**

Visit:

HTTP: http://<Load-Balancer-DNS>

HTTPS (if configured): <https://vprofileapp.aniketproject.xyz>



If HTTPS is active:

Check the padlock symbol

View SSL certificate details (issued by Amazon)

### Step 6: Check Target Group Health

Navigate to your **Target Group**

Go to **Targets** tab

Refresh to check health status:

Status = healthy

Troubleshooting Unhealthy Targets:

\* Ensure Tomcat is running on app01

\* Ensure Security Group allows port 8080 from Load Balancer SG

\* Ensure Health Check is set to port **8080**

### Step 7: Application Functional Validation

Visit application through **Load Balancer URL**

Login with credentials:

Username: admin\_vp

Password: admin\_vp

Verify:

**DB connection:** Successful login implies DB is connected

**RabbitMQ:** Queue should be visible after login

**Memcache:**

Click a user ➔ shows "data from DB"

Click again ➔ shows "data from cache"

# Auto Scaling Setup for vProfile Application

Enable automatic scaling of application servers (Tomcat instances) based on load to ensure high availability and performance.

## Step 1: Create AMI from EC2 Instance

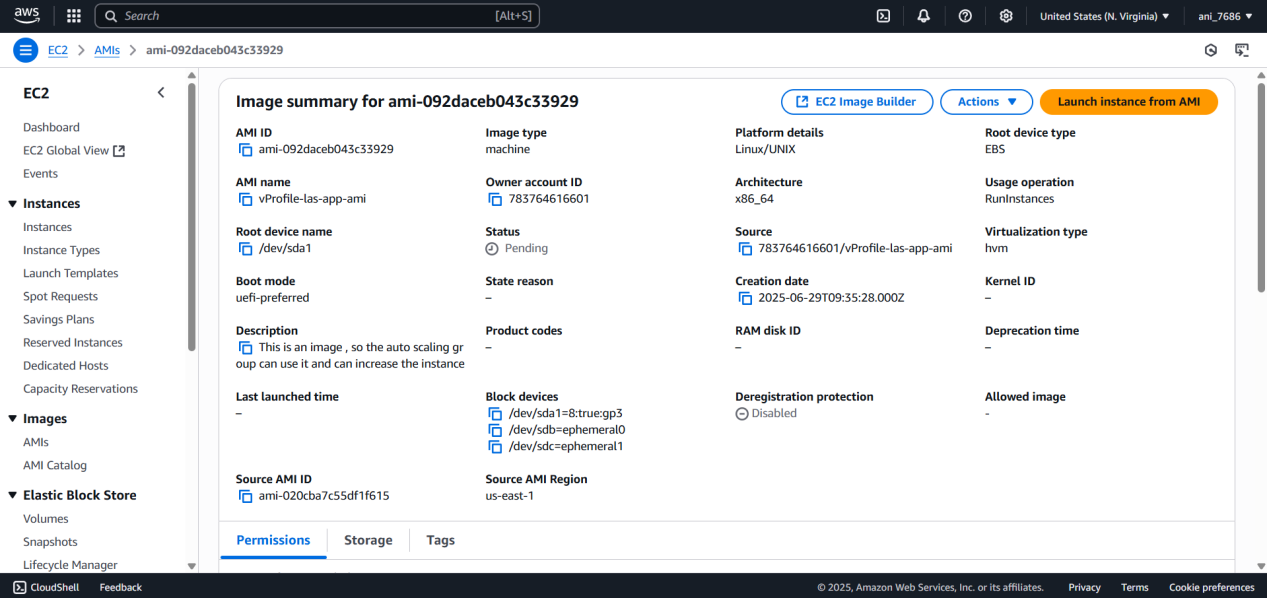
Select the vprofile-app01 instance.

Go to Actions > Image and Templates > Create Image.

Name the AMI as: vProfile-las-app-ami

Click **Create Image**.

**Why AMI?** Auto Scaling launches new instances using this AMI so they are exact replicas of the current working app instance.



## Step 2: Create Launch Template

Go to **Launch Templates** in EC2 console.

Click **Create Launch Template**.

Name it: vProfile-las-app-LT

Select the AMI created above

Instance Type: t2.micro or t3.micro

Select Key Pair: vProfile-prod-key

Network Settings:

Choose **App Security Group**

Tags:

Name = vprofile-app

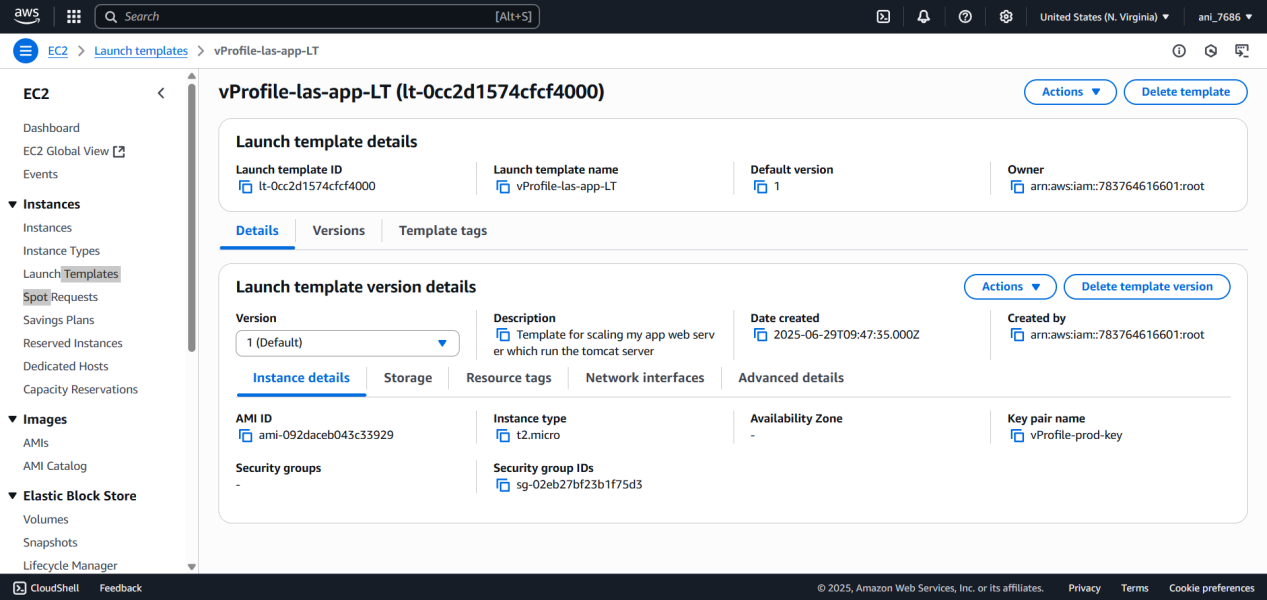
Project = vprofile

Enable **volume tags**

(Optional) Add IAM Role if needed

Click **Create Launch Template**

🧠 **Why Launch Template?** It defines how Auto Scaling should launch new instances: which AMI, what type, what SG, key pair, and so on



## Step 3: Create Auto Scaling Group

Go to **Auto Scaling Groups**

Click **Create Auto Scaling Group**

Name: vprofile-las-app-asg

Choose Launch Template: vprofile-las-app-LT

Select VPC and Availability Zones

Attach to existing Load Balancer:

Select **Target Group**: vprofile-las-tg

Turn ON Health Checks:

Enable **Elastic Load Balancing health checks**

Configure Group Size:

Desired: 1

Min: 1

Max: 4

Scaling Policy:

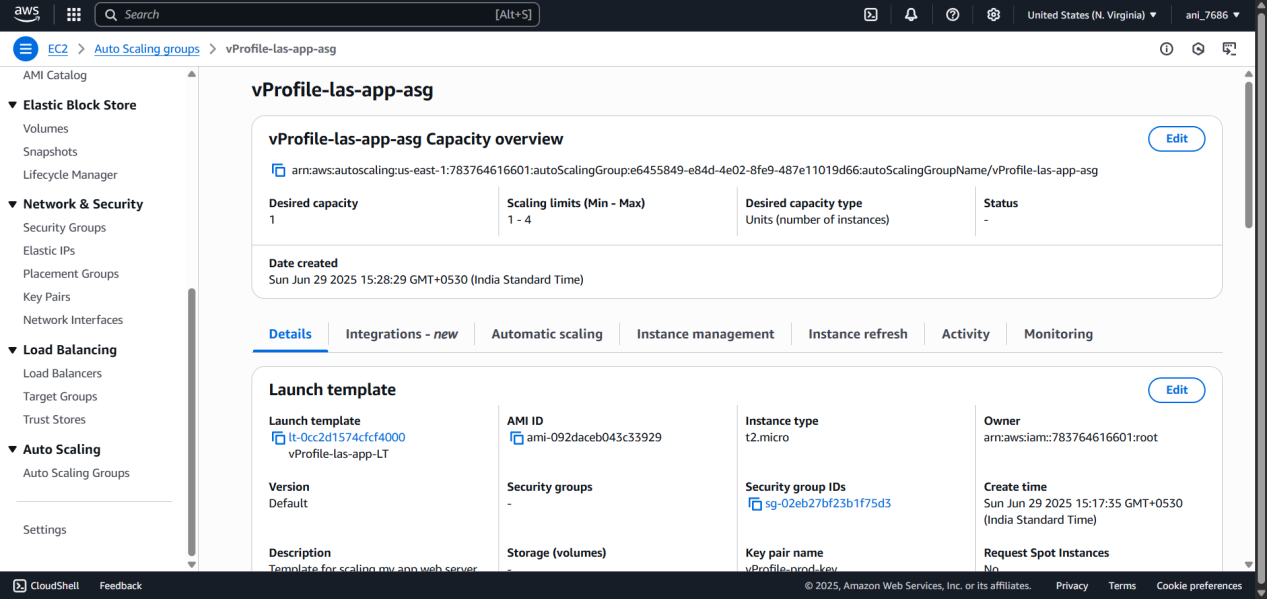
Use **Target Tracking**

Metric: CPU Utilization

Target Value: 50%

(Optional) Notification Settings via SNS

Click **Create Auto Scaling Group**



## Lift-and-Shift Scope

Application was migrated **without actual data**.

**No** DB sync or S3 sync needed.

This was a **code and infrastructure migration only**.

This used **Infrastructure as a Service (IaaS)** — all services like DB, Memcache, etc., were installed and managed manually on EC2.

## Final Architecture Review

**Security Groups & Key Pairs**: Created per role (app, backend, LB)

**EC2 Instances**: Configured with user-data scripts

**Maven Build + S3 Upload**: Java WAR built locally, uploaded to S3

**Tomcat Server**: Retrieved WAR from S3 and deployed

**Load Balancer**: ALB with HTTP(80) and HTTPS(443) endpoints

**ACM Certificate**: Configured for HTTPS

**Target Group**: Used for health checks and routing

**Route 53 Private Hosted Zone**: Enabled DNS resolution among services

**Auto Scaling Group**: Based on Launch Template and custom AMI

**Stickiness**: Enabled due to app-specific session handling

Summary :

This project successfully lifted and shifted a locally running Java-based application (vProfile) to the AWS cloud infrastructure, using best practices in provisioning, deployment, scalability, and security.