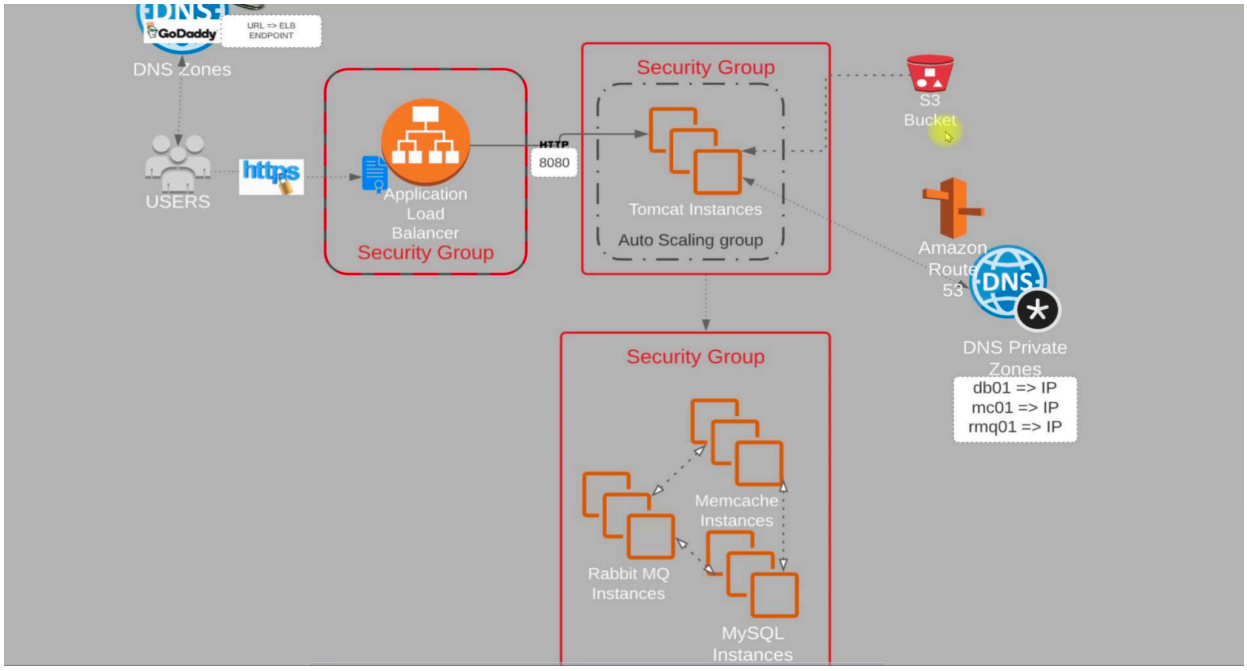
Complete Lift-and-Shift Migration of the VProfile Application to AWS (Step-by-Step with Full Commands & Outputs)

Cloud migration has become one of the most essential skills for DevOps and Cloud Engineers. Every company — big or small — is moving their applications from traditional data centers to the cloud for better scalability, automation, and cost efficiency.

In this project, we're going to do exactly that.

We will **lift** our traditional vProfile web application stack from local virtual machines and **shift** it to AWS Cloud — exactly how real-world cloud migrations happen.

If you're a beginner who wants to *actually understand* how to run multi-tier apps on AWS, load balancers, private DNS, caching, message brokers, and auto scaling…
this article is your complete guide.



Why Lift & Shift?

A traditional data center requires:

- SysAdmins
- Virtualization team
- Network team
- Hardware procurement
- Monitoring 24/7
- Manual scaling
- High cost
- Lots of downtime

And if suddenly your application gets more traffic, scaling up requires:

- Buying new hardware
- Installing it
- Configuring everything manually

Cloud solves all these problems beautifully.

In AWS, we get:

- **Pay-as-you-go** pricing
- **Elasticity** (auto scaling)
- **High availability**
- **Automation**
- **No upfront cost**
- **Managed services**

This project shows you how simple and powerful cloud computing can be.

Breaking Down the vProfile Application Stack

Before migrating, let's understand the components we're lifting to AWS.

1. Tomcat (Application Server)

This is where our Java application actually runs.
All user requests ultimately reach Tomcat.

On AWS, Tomcat will run on **EC2 instances**, behind a **Load Balancer** and **Auto Scaling**.

2. Memcached (Cache Server)

Memcached is a high-performance in-memory cache.

Why we use it:

- Stores frequently accessed data
- Reduces load on MySQL
- Speeds up application performance

Instead of hitting MySQL for every query, the app quickly grabs data from Memcached.

3. RabbitMQ (Message Broker)

RabbitMQ handles asynchronous tasks.

Example:
Sending emails, processing logs, running background processes — all these tasks go into RabbitMQ's queue and get processed smoothly without slowing down the user.

This makes the app more responsive and scalable.

4. MySQL (Database Server)

This is the main relational database.
Since this is a true **lift and shift**, we will continue using MySQL on EC2 instead of RDS.

Final AWS Architecture We Will Build

User → Domain (GoDaddy) → AWS Application Load Balancer → Tomcat (EC2 ASG)
|
→ Memcached (EC2)
→ RabbitMQ (EC2)
→ MySQL (EC2)

AWS Components Involved:

- **Application Load Balancer (HTTPS)**
- **Amazon Certificate Manager (SSL/TLS)**
- **Auto Scaling Group for Tomcat**
- **S3 for artifact storage**
- **Route 53 Private Hosted Zone**
- **EC2 backend instances**
- **Security Groups for every tier**

Security Groups Overview

Security Group	Allows	From
ALB SG	443	Internet

Tomcat SG	8080	ALB
Backend SG	Custom Ports	Tomcat

This ensures secure and controlled communication between tiers.

Step-by-Step Workflow

Here's exactly how we'll execute this project:

1. Login to AWS

2. Create Security Groups

3. Launch Backend EC2 Instances

4. Configure Route 53 Private Hosted Zone

5. Build the WAR file

6. Upload artifact to S3

7. Launch Tomcat Auto Scaling Group

8. Create Application Load Balancer

9. Test everything

10. Configure Auto Scaling Policies

Let's Get Started With Hands-On!

Now that you understand the entire architecture, components, flow, and AWS services involved —
it's time to roll up your sleeves and **start building it step-by-step on AWS**.

In the hands-on section, we will:

- Create the full network setup
- Launch all backend servers
- Configure Tomcat with user-data
- Set up Load Balancer + HTTPS
- Connect S3 artifact storage
- Test the application
- Finally create the Auto Scaling Group

This is where theory transforms into actual cloud engineering.

So grab a coffee ☕, open your AWS console, and let's begin the hands-on journey!

Below is the **updated Medium article**, with a new section added that explains *exactly what we are going to do next* based on the instructor's lecture about **security groups and key pairs**.

I've added it in a smooth Medium-style flow so it fits naturally into your article.

You can directly publish this version.

We Will Create Three Security Groups

AWS Security Groups act like virtual firewalls.
They control **which instance can talk to which instance**, and on **which ports**.

1. Load Balancer Security Group (vprofile-ELB-SG)

This security group allows public traffic from the internet.

Inbound rules:

- HTTP → port 80 → from anywhere

- HTTPS → port 443 → from anywhere

Later, we will remove HTTP and keep only HTTPS once our ACM certificate is attached.

Why?
Because users will access your website securely using SSL/TLS.

2. Tomcat Application Security Group (vprofile-app-sg)

This one is for Tomcat EC2 instances where our vProfile app will run.

Inbound rules:

- Port **8080** → from the **Load Balancer Security Group**
- Port **22** → from **My IP** (for SSH login)

Why 8080?
Tomcat listens on port 8080.
And we want only the **Load Balancer** to send requests to Tomcat — not anyone from the internet.

3. Backend Security Group (vprofile-backend-sg)

Backend servers will run:

- MySQL
- Memcached
- RabbitMQ

Inbound rules (from Tomcat SG):

- **3306** → MySQL
- **11211** → Memcached

- **5672** → RabbitMQ
- **22** → SSH from My IP

Where do these ports come from?
From the file:
`src/main/resources/application.properties`

This tells the Tomcat application how to connect to backend services.

Extra Rule for Backend → Backend Communication

We will add:

All traffic → Source: vprofile-backend-sg

This allows backend instances (MySQL, Memcache, RabbitMQ) to talk to each other when needed.

This rule is safe **as long as the instances themselves are secure**.

Next Step: Create a Key Pair

We will create a key pair named:

```
vprofile-prod-key.pem
```

This key will be used to SSH into all EC2 instances.

If you use:

- Terminal / Git Bash → use `.pem`

- PuTTY → use `.ppk`

Why This Step Matters

Most beginners skip networking rules because they look “boring” —
but **this is the exact place where 90% of real-world issues happen**.

If a port is wrong…
If a security group is attached incorrectly…
If outbound rules are changed accidentally…

Your entire architecture breaks.

As a DevOps/cloud engineer, understanding:

- **how traffic flows**
- **who can talk to whom**
- **what ports each service uses**

is the foundation of building secure and functional cloud systems.

Let’s Get Started With Hands-On!

Now that your fundamentals and plan are clear, it’s time to actually start building!

In the hands-on part, we will:

- Create all **three** security groups
- Add precise inbound rules
- Create a key pair

- Launch backend EC2 instances
- Configure user-data scripts
- Deploy Tomcat through Auto Scaling
- Attach ALB with HTTPS
- Connect S3 for artifacts
- Test the application end-to-end

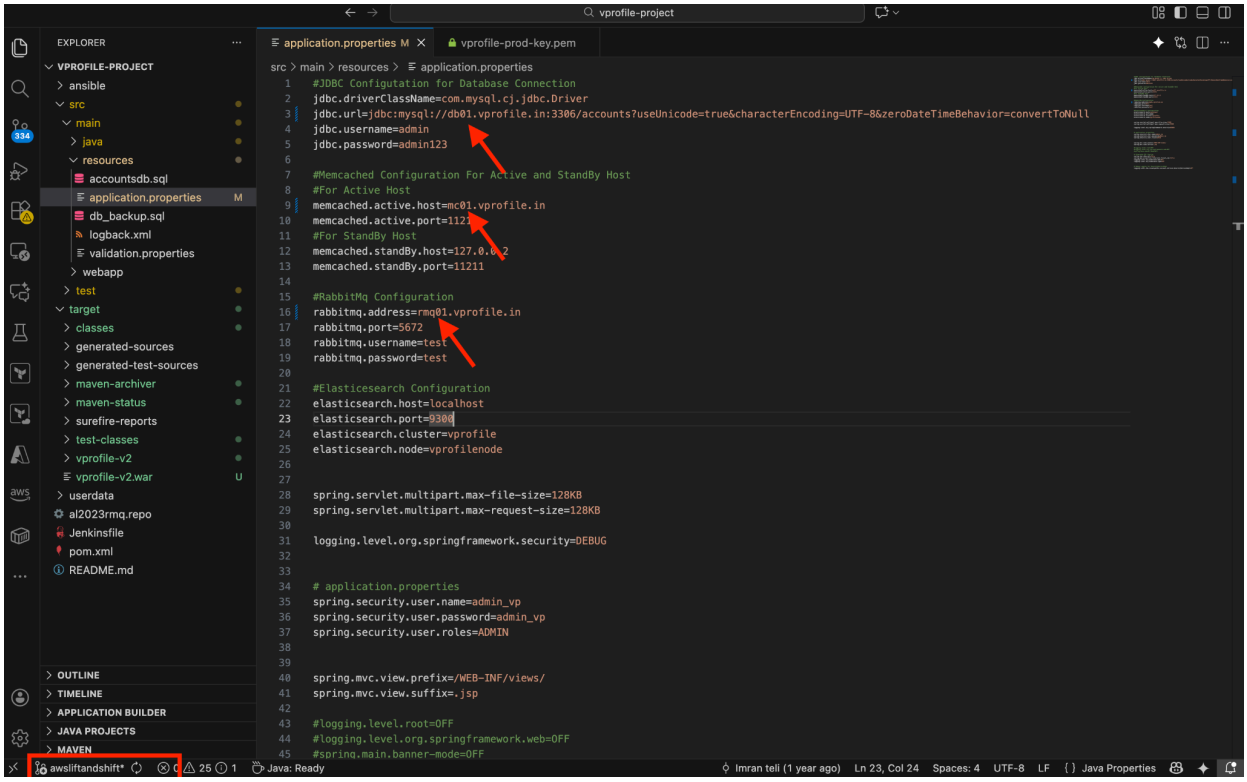
Let’s open the AWS console and begin our first step:
Security Groups + Key Pair Setup

Before We Begin the Hands-On

Before we jump into creating the security groups in AWS, the first thing we need to do is **open the project repository** and verify the backend service ports.

Go to your repository and navigate to:

src/main/resources/



Inside this directory, open the `application.properties` file. This file contains the exact ports that our application uses to connect to backend services:

- **MySQL → 3306**
- **Memcache → 11211**
- **RabbitMQ → 5672**

We will use these exact ports while configuring our security groups.

Now that we know which services our application communicates with, we can move on to creating the required AWS security groups.
In the hands-on section, we will create **three security groups**:

1. **Load Balancer Security Group (ELB SG)**
    - Allows HTTP (80) and HTTPS (443) from the internet.
2. **Application Security Group (Tomcat SG)**

- Allows port 8080 from the load balancer SG.
- Allows SSH (22) from your IP.

3. **Backend Security Group**
    - Allows MySQL (3306), Memcache (11211), and RabbitMQ (5672) **from the application SG**.
    - Allows SSH (22) from your IP.
    - Allows internal traffic within the backend SG itself.

With this understanding in place, you're ready to proceed.

Let’s Get Started with the Hands-On

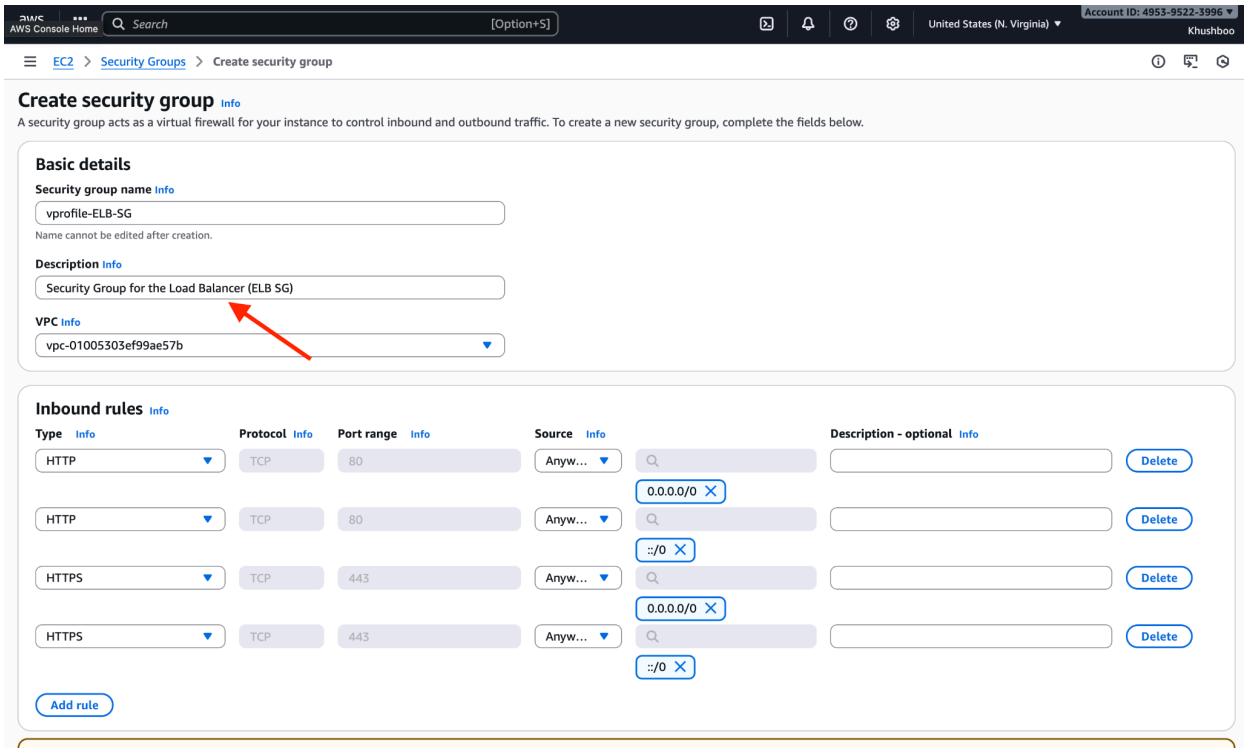
Head over to the **AWS EC2 service**, and we’ll begin by creating the security groups.

1. First: Create the Security Group for the Load Balancer (ELB SG)

This SG will receive traffic directly from the internet, so we will allow:

- **HTTP (80)** from anywhere
- **HTTPS (443)** from anywhere

Later, once our SSL certificate is configured, we will remove HTTP and keep only HTTPS.

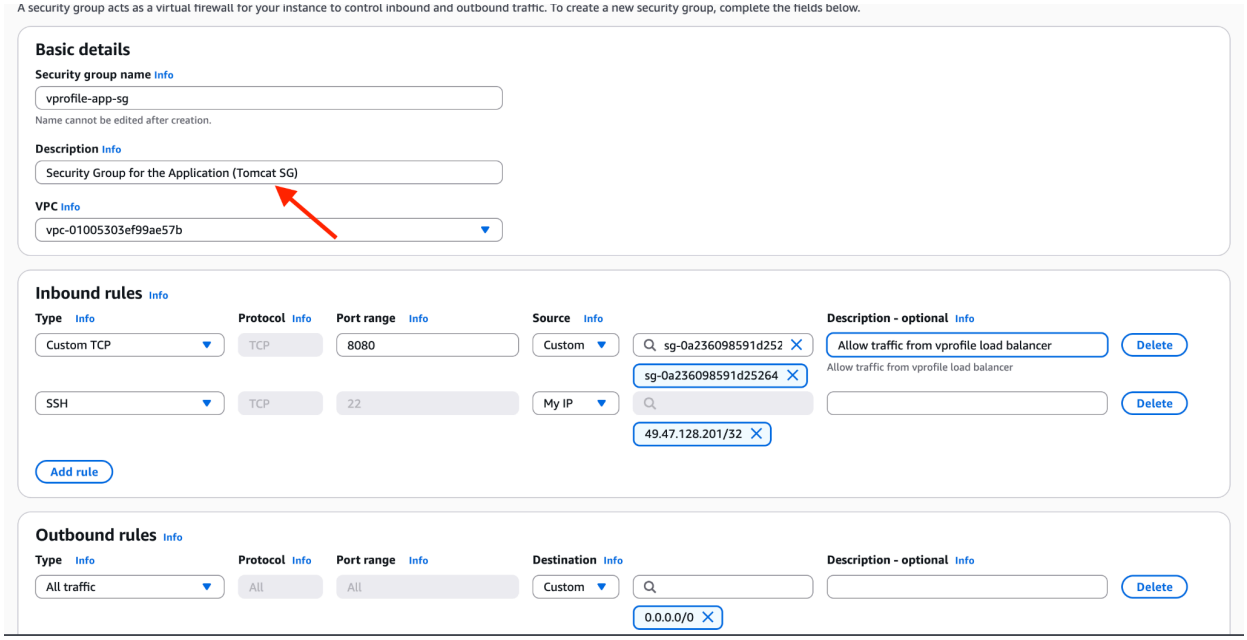


2. Second: Create the Security Group for the Application (Tomcat SG)

This SG will be attached to our Tomcat EC2 instances.
We will allow:

- **Port 8080** — but *only* from the Load Balancer SG
- **Port 22** — from *My IP* so we can SSH into the instance

This ensures only the load balancer can talk to Tomcat, not the entire internet.

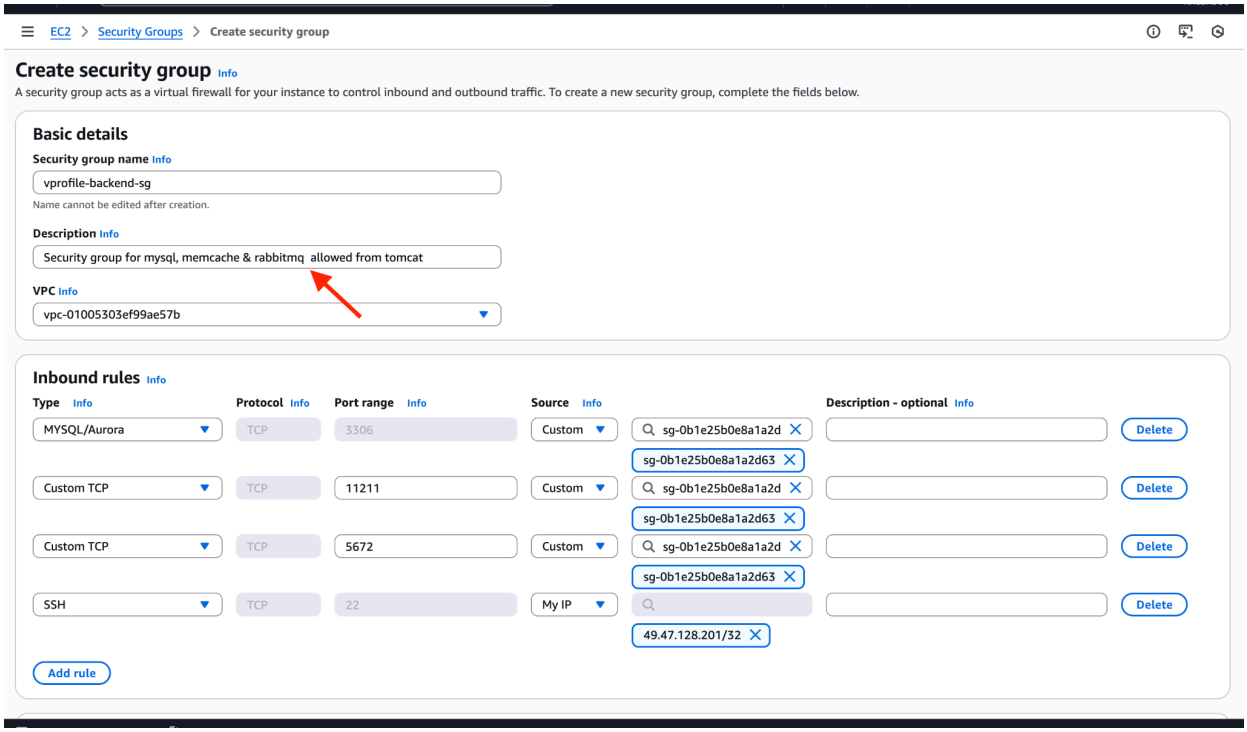


3. Third: Create the Security Group for Backend Services

This SG will be used by MySQL, Memcache, and RabbitMQ EC2 instances.
We will allow:

- **3306 (MySQL)** from the Tomcat SG
- **11211 (Memcache)** from the Tomcat SG
- **5672 (RabbitMQ)** from the Tomcat SG
- **22 (SSH)** from My IP
- **All traffic from itself** (backend servers need internal communication)

This tightly controls backend access so only the application layer can connect.

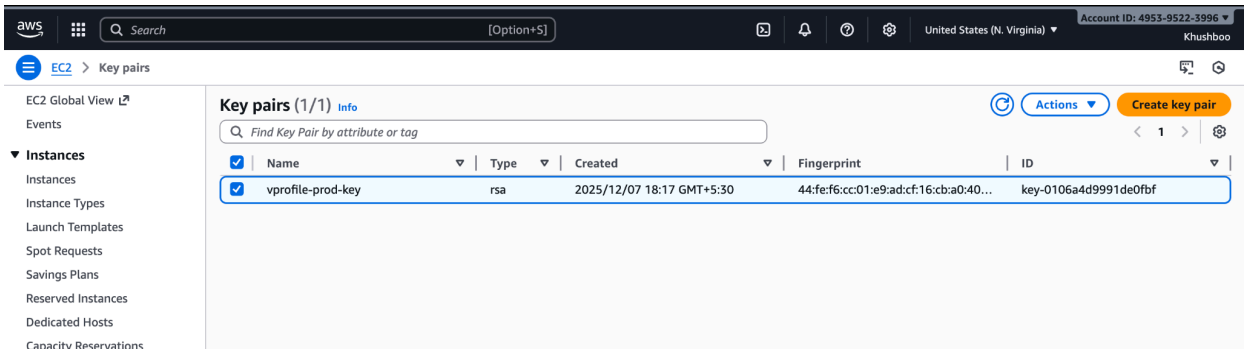


Step 2: Create the Key Pair

Now that our security groups are ready, the next step is to create a **Key Pair**.
This key pair will allow us to SSH into all our EC2 instances (Tomcat, MySQL, RabbitMQ, Memcache).

- Go to **EC2 → Key Pairs → Create Key Pair**
- Name it: **vprofile-prod-key**
- Download the `.pem` file (or `.ppk` if using PuTTY)
- Store it safely — you will need it to access your servers

With security groups and the login key pair in place, we're fully prepared to launch our EC2 instances.



Step 3: Launching EC2 Instances With User Data Scripts

Now that our Security Groups and Key Pair are ready, we're moving to the most important part of the setup — **launching four EC2 instances** and configuring each service automatically using **User Data scripts**.

We will create the following instances:

- **MySQL instance**
- **Memcache instance**
- **RabbitMQ instance**
- **Tomcat application instance**

Each instance will run its respective setup script during launch so everything installs and configures automatically — no manual provisioning required.

Security Group Mapping (Very Important)

Before launching, remember the correct SG placement:

- **MySQL, Memcache, RabbitMQ → Backend Security Group**
- **Tomcat → Application Security Group**

- **Load Balancer (later) → ELB Security Group**

Choosing the wrong SG will break connectivity, so double-check this in every launch.

Step 3.1: Clone the Project Repository

We first need access to the *user-data scripts* stored in the GitHub repository. These scripts automate the installation of MySQL, Memcache, RabbitMQ, and Tomcat.

1. Go to the GitHub repo:
   `https://github.com/khushboo-sah/vprofile-project`

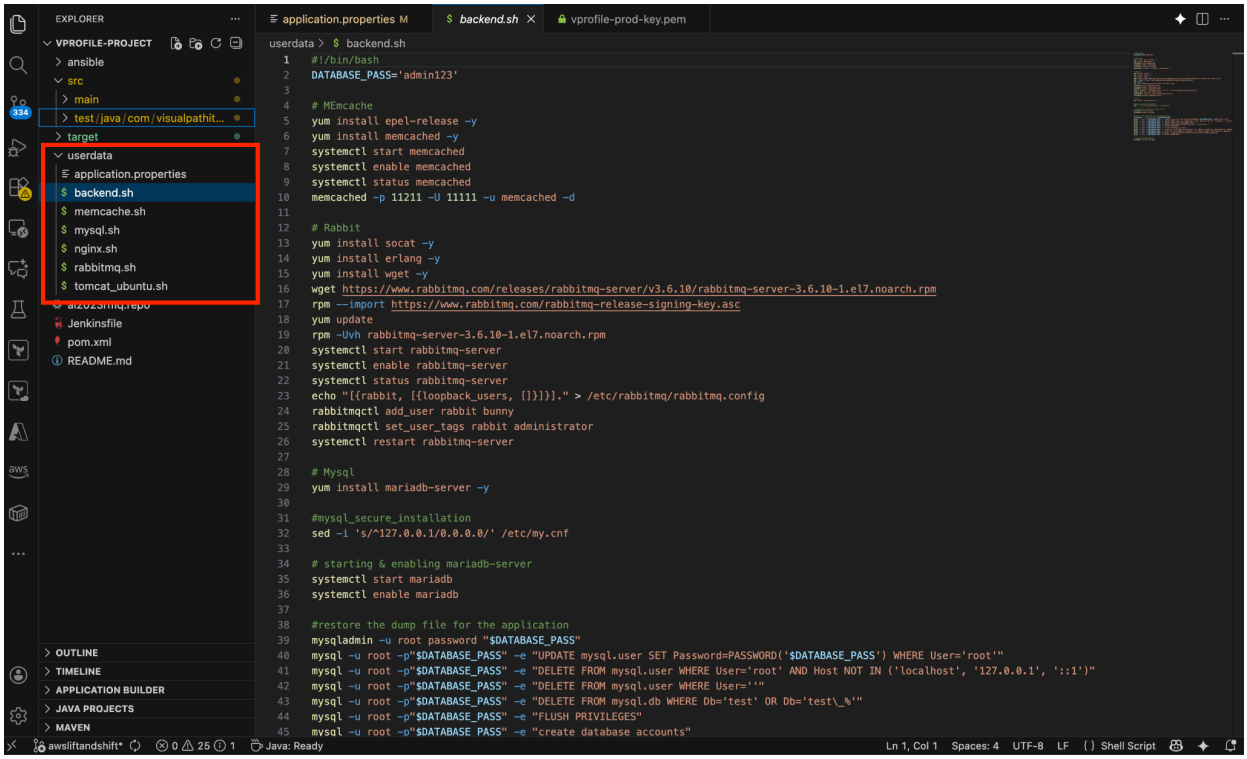
2. Clone it into your system (VS Code or terminal).

3. Switch to the branch:
   aws-liftandshift

4. Open the folder:
   `user-data/`
   This folder contains:

   - `mysql.sh`
   - `memcache.sh`
   - `rabbitmq.sh`
   - `tomcat_ubuntu.sh`

We will use each script while launching the respective EC2 instance.



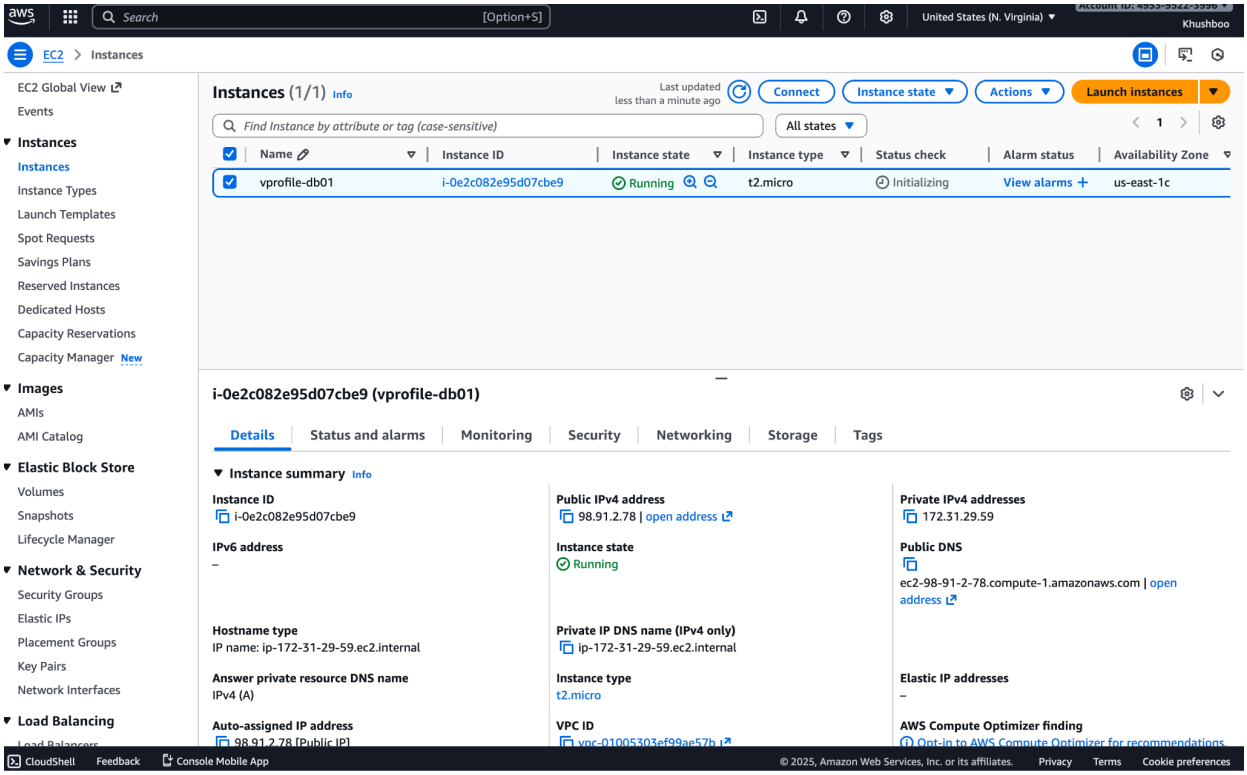
Step 3.2: Launch the MySQL EC2 Instance

We'll launch the first EC2 instance using the `mysql.sh` script.
This script installs MariaDB, secures the setup, creates the database, admin user, privileges, and loads the DB schema.

Key configuration:

- **AMI:** Amazon Linux 2023
- **Instance Type:** t2.micro / t3.micro
- **Key Pair:** vprofile-prod-key
- **Security Group:** vprofile-backend-sg
- **User Data:** paste the entire `mysql.sh` script

Then launch the instance.



Step 3.3: Launch the Memcache EC2 Instance

Next, we launch the Memcache server using the `memcache.sh` script.

This script:

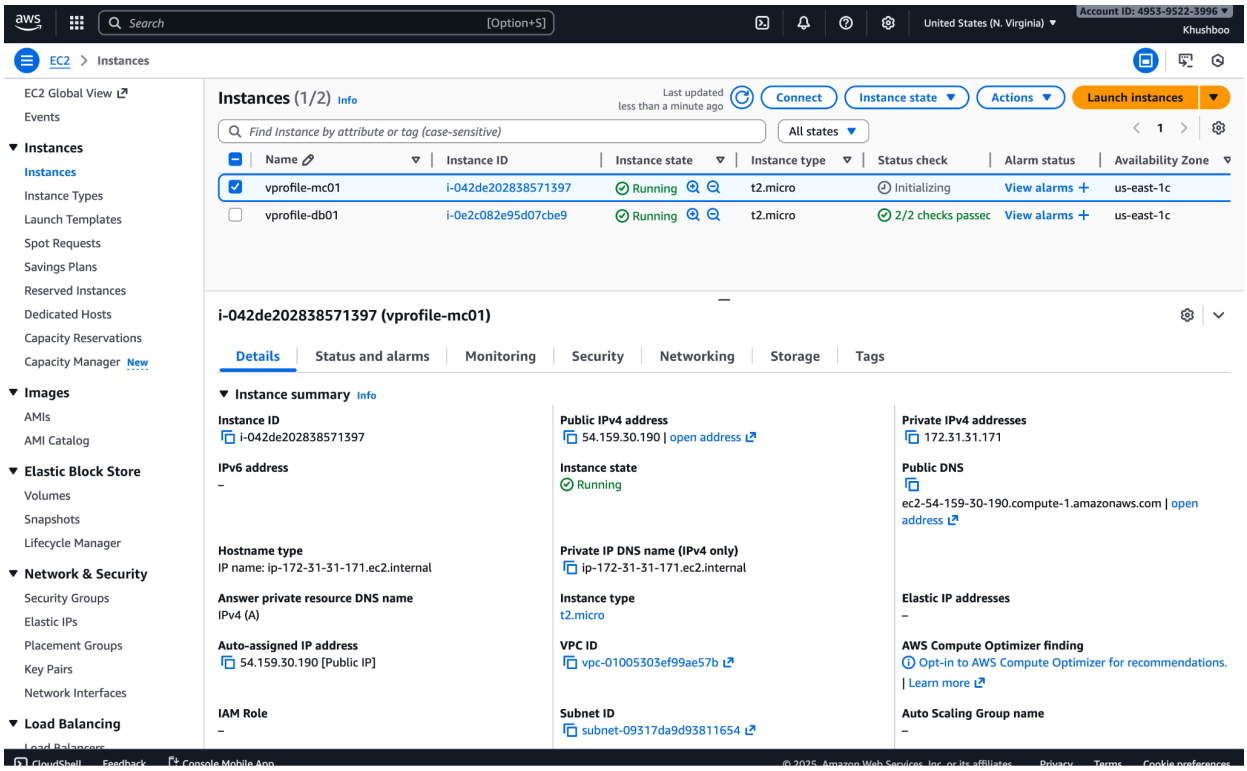
- Installs memcached
- Updates config to listen on all IPs (`0.0.0.0`)
- Enables remote connections
- Restarts the service

Configuration:

- **AMI:** Amazon Linux 2023
- **Security Group:** vprofile-backend-sg

- **User Data:** `memcache.sh`

Launch the instance.



Step 3.4: Launch the RabbitMQ EC2 Instance

This script is different because Amazon Linux 2023 uses different repositories.
The script:

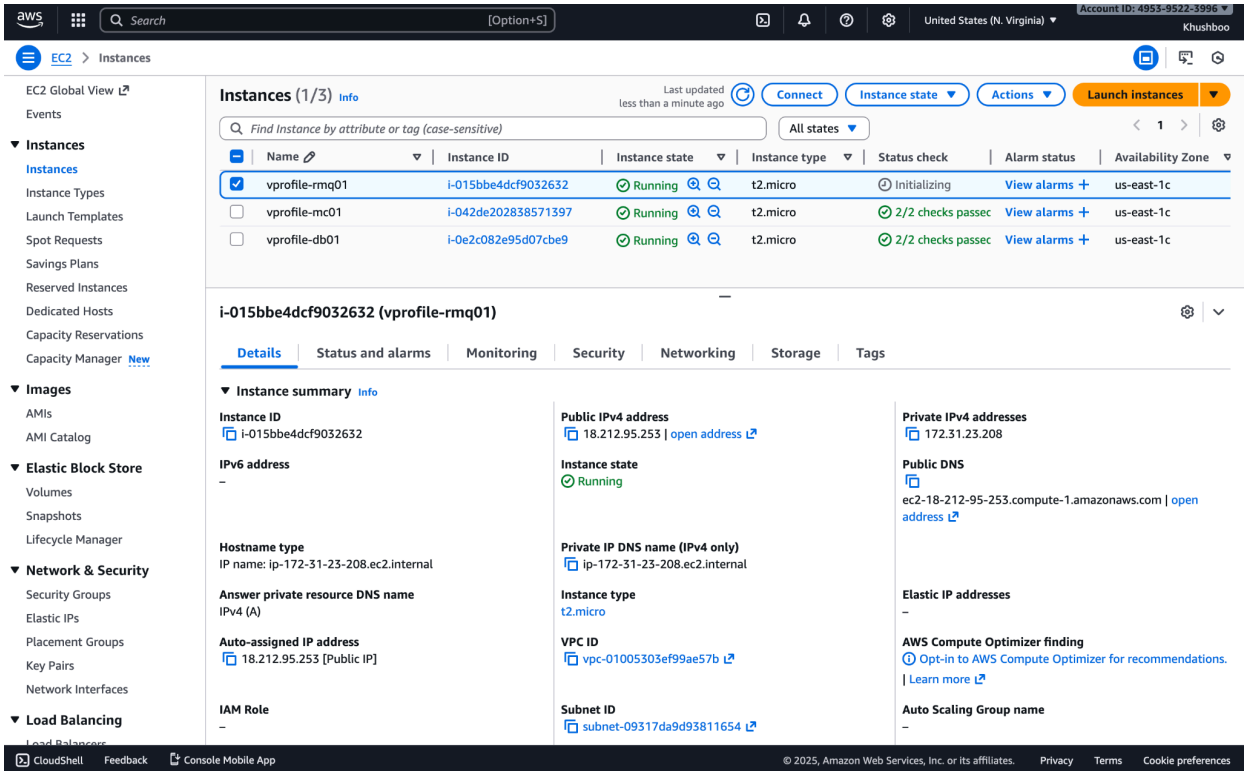
- Imports RabbitMQ & Erlang repo keys
- Downloads repo configuration file from the GitHub repo
- Installs all dependencies
- Installs Erlang + RabbitMQ automatically
- Adds user, gives admin privileges

- Updates config

Configuration:

- **AMI:** Amazon Linux 2023
- **Security Group:** vprofile-backend-sg
- **User Data:** `rabbitmq.sh`

Launch the instance.



Step 3.5: Launch the Tomcat EC2 Instance (Ubuntu)

For Tomcat, we use **Ubuntu 24.04** instead of Amazon Linux.
Why?
Because Ubuntu has **Tomcat10** available directly in its repository — no manual download, no extraction, no service creation.

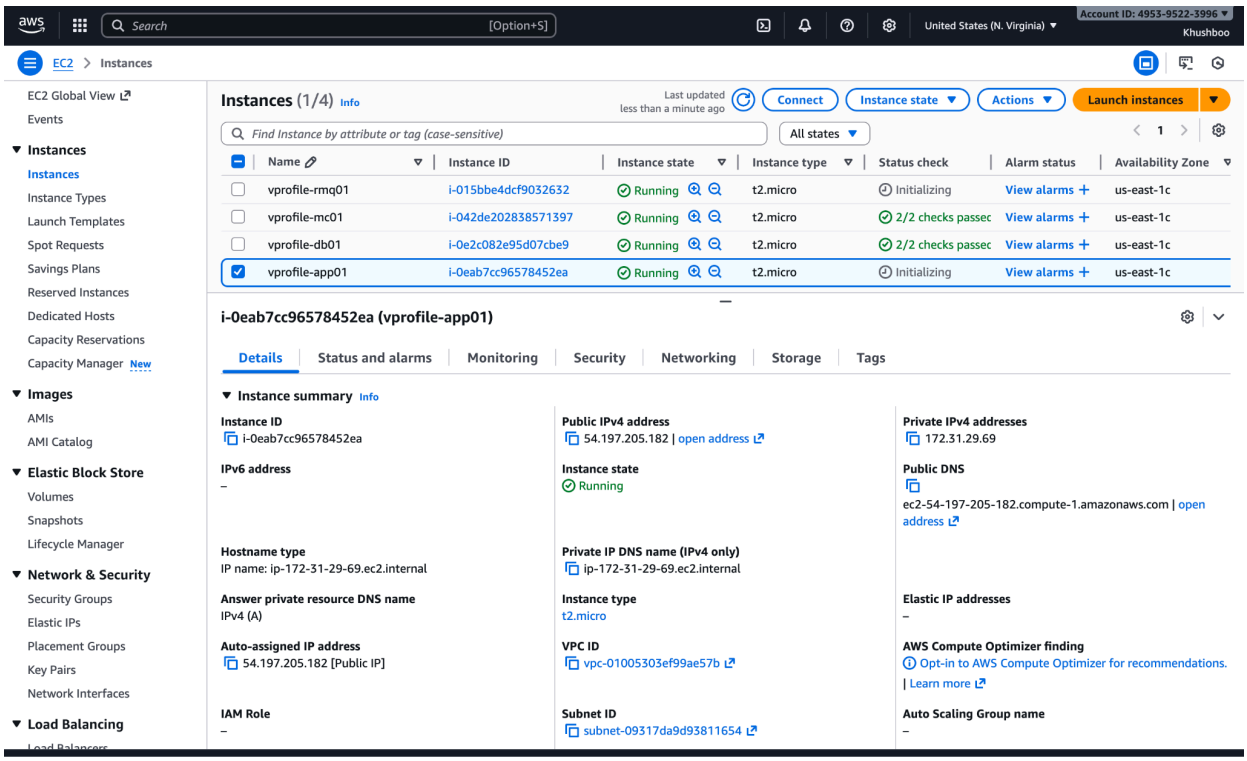
The script:

- Installs Tomcat10
- Starts & enables the service

Configuration:

- **AMI:** Ubuntu Server 24.04 LTS
- **Security Group:** vprofile-app-sg
- **User Data:** `tomcat_ubuntu.sh`

Launch the instance.



✔️ Step 4: Verify All Services

Once all instances are running, SSH into each one (using the key pair) and verify:

- **MySQL:** `systemctl status mariadb`

```
Last login: Sun Dec  7 20:19:37 on ttys004
khushboo@Mac ~ % cd downloads
khushboo@Mac downloads % ssh -i "vprofile-prod-key.pem" ec2-user@ec2-98-91-2-78.compute-1.amazonaws.com
The authenticity of host 'ec2-98-91-2-78.compute-1.amazonaws.com (98.91.2.78)' can't be established.
ED25519 key fingerprint is SHA256:8DQSx58LiFOJNY1EbRxYTAC/sDH7o3+9ekmc7tQzaK8.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-98-91-2-78.compute-1.amazonaws.com' (ED25519) to the list of known hosts.
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-172-31-29-59 ~]$ sudo -i
[root@ip-172-31-29-59 ~]# systemctl status mariadb
● mariadb.service - MariaDB 10.5 database server
     Loaded: loaded (/usr/lib/systemd/system/mariadb.service; enabled; preset: disabled)
     Active: active (running) since Mon 2025-12-08 11:28:25 UTC; 10min ago
       Docs: man:mariadbd(8)
             https://mariadb.com/kb/en/library/systemd/
   Main PID: 27423 (mariadbd)
     Status: "Taking your SQL requests now..."
      Tasks: 8 (limit: 1106)
     Memory: 67.4M
        CPU: 492ms
     CGroup: /system.slice/mariadb.service
             └─27423 /usr/libexec/mariadbd --basedir=/usr

Dec 08 11:28:25 ip-172-31-29-59.ec2.internal mariadb-prepare-db-dir[27380]: The second is mysql@localhost, it has no password either, but
Dec 08 11:28:25 ip-172-31-29-59.ec2.internal mariadb-prepare-db-dir[27380]: you need to be the system 'mysql' user to connect.
Dec 08 11:28:25 ip-172-31-29-59.ec2.internal mariadb-prepare-db-dir[27380]: After connecting you can set the password, if you would need to be
Dec 08 11:28:25 ip-172-31-29-59.ec2.internal mariadb-prepare-db-dir[27380]: able to connect as any of these users with a password and without sudo
Dec 08 11:28:25 ip-172-31-29-59.ec2.internal mariadb-prepare-db-dir[27380]: See the MariaDB Knowledgebase at https://mariadb.com/kb
Dec 08 11:28:25 ip-172-31-29-59.ec2.internal mariadb-prepare-db-dir[27380]: Please report any problems at https://mariadb.org/jira
Dec 08 11:28:25 ip-172-31-29-59.ec2.internal mariadb-prepare-db-dir[27380]: The latest information about MariaDB is available at https://mariadb.org/.
Dec 08 11:28:25 ip-172-31-29-59.ec2.internal mariadb-prepare-db-dir[27380]: Consider joining MariaDB's strong and vibrant community:
Dec 08 11:28:25 ip-172-31-29-59.ec2.internal mariadb-prepare-db-dir[27380]: https://mariadb.org/get-involved/
Dec 08 11:28:25 ip-172-31-29-59.ec2.internal systemd[1]: Started mariadb.service - MariaDB 10.5 database server.
[root@ip-172-31-29-59 ~]#
```

```
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 14
Server version: 10.5.29-MariaDB MariaDB Server

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

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

MariaDB [accounts]> show tables;
+--------------------+
| Tables_in_accounts |
+--------------------+
| role               |
| user               |
| user_role          |
+--------------------+
3 rows in set (0.000 sec)

MariaDB [accounts]>
```

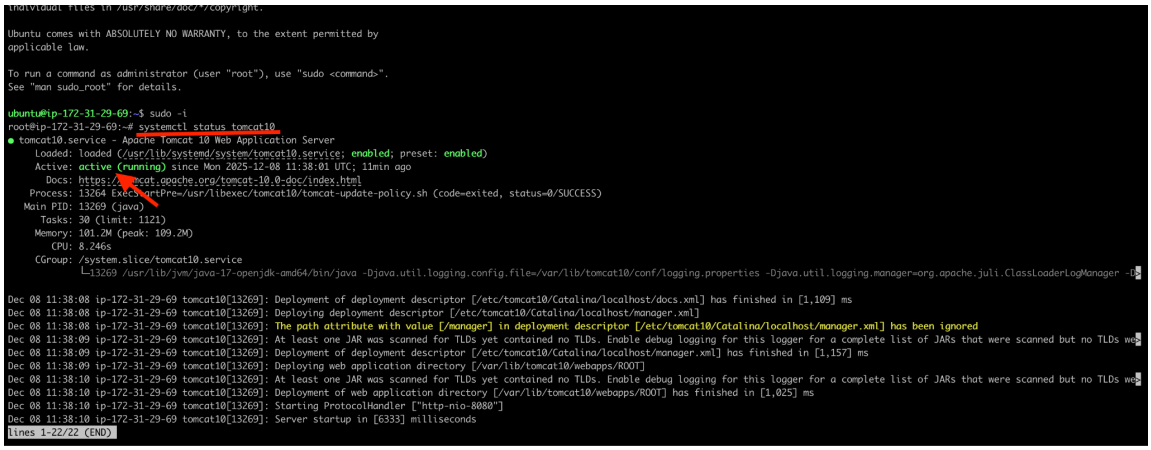
- **Memcache:** `systemctl status memcached`

```
khushboo@Mac downloads % ssh -i "vprofile-prod-key.pem" ec2-user@ec2-54-159-30-190.compute-1.amazonaws.com
The authenticity of host 'ec2-54-159-30-190.compute-1.amazonaws.com (54.159.30.190)' can't be established.
ED25519 key fingerprint is SHA256:GhiY6/+OnhmhQT/B4Alw50vmlxk0qWZ1SDsMRLe3CWg.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-54-159-30-190.compute-1.amazonaws.com' (ED25519) to the list of known hosts.
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-172-31-31-171 ~]$ sudo -i
[root@ip-172-31-31-171 ~]# systemctl status memcached
● memcached.service - memcached daemon
     Loaded: loaded (/usr/lib/systemd/system/memcached.service; enabled; preset: disabled)
     Active: active (running) since Mon 2025-12-08 11:30:49 UTC; 13min ago
   Main PID: 4979 (memcached)
      Tasks: 10 (limit: 1106)
     Memory: 1.8M
        CPU: 238ms
     CGroup: /system.slice/memcached.service
             └─4979 /usr/bin/memcached -p 11211 -u memcached -m 64 -c 1024 -l 0.0.0.0,::1

Dec 08 11:30:49 ip-172-31-31-171.ec2.internal systemd[1]: Started memcached.service - memcached daemon.
[root@ip-172-31-31-171 ~]#
```

- **RabbitMQ:** `systemctl status rabbitmq-server`

```
khushboo@Mac downloads % ssh -i "vprofile-prod-key.pem" ec2-user@ec2-18-212-95-253.compute-1.amazonaws.com
The authenticity of host 'ec2-18-212-95-253.compute-1.amazonaws.com (18.212.95.253)' can't be established.
ED25519 key fingerprint is SHA256:XBK2hATsfejT7UJ70sgU6IHQH1hJtprCiw8hUkuH1Jo.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-18-212-95-253.compute-1.amazonaws.com' (ED25519) to the list of known hosts.
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-172-31-23-208 ~]$ sudo -i
[root@ip-172-31-23-208 ~]# systemctl status rabbitmq-server
● rabbitmq-server.service - Open source RabbitMQ server
     Loaded: loaded (/usr/lib/systemd/system/rabbitmq-server.service; enabled; preset: disabled)
     Active: active (running) since Mon 2025-12-08 11:35:22 UTC; 11min ago
   Main PID: 26400 (beam.smp)
      Tasks: 23 (limit: 1106)
     Memory: 70.0M
        CPU: 3.475s
     CGroup: /system.slice/rabbitmq-server.service
             ├─26400 /usr/lib64/erlang/erts-15.2.7.4/bin/beam.smp -W w -MBas ageffcbf -MHas ageffcbf -MBlmbcs 512 -MHlmbcs 512 -MMmcs 30 -pc unicode -P 1048576 -t 5000000 -stbt db -zdbbl 128000 -sbwt none -sbwtdcpu none -sbwtdio none
             ├─26413 erl_child_setup 32768
             ├─26428 /usr/lib64/erlang/erts-15.2.7.4/bin/inet_gethost 4
             ├─26429 /usr/lib64/erlang/erts-15.2.7.4/bin/inet_gethost 4
             ├─26440 /usr/lib64/erlang/erts-15.2.7.4/bin/epmd -daemon
             └─26459 /bin/sh -s rabbit_disk_monitor

Dec 08 11:35:21 ip-172-31-23-208.ec2.internal rabbitmq-server[26400]:   Doc guides:  https://www.rabbitmq.com/docs
Dec 08 11:35:21 ip-172-31-23-208.ec2.internal rabbitmq-server[26400]:   Support:     https://www.rabbitmq.com/docs/contact
Dec 08 11:35:21 ip-172-31-23-208.ec2.internal rabbitmq-server[26400]:   Tutorials:   https://www.rabbitmq.com/tutorials
Dec 08 11:35:21 ip-172-31-23-208.ec2.internal rabbitmq-server[26400]:   Monitoring:  https://www.rabbitmq.com/docs/monitoring
Dec 08 11:35:21 ip-172-31-23-208.ec2.internal rabbitmq-server[26400]:   Upgrading:   https://www.rabbitmq.com/docs/upgrade
Dec 08 11:35:21 ip-172-31-23-208.ec2.internal rabbitmq-server[26400]:   Logs: /var/log/rabbitmq/rabbit@ip-172-31-23-208.log
Dec 08 11:35:21 ip-172-31-23-208.ec2.internal rabbitmq-server[26400]:         <stdout>
Dec 08 11:35:21 ip-172-31-23-208.ec2.internal rabbitmq-server[26400]:   Config file(s): /etc/rabbitmq/rabbitmq.config
Dec 08 11:35:22 ip-172-31-23-208.ec2.internal rabbitmq-server[26400]:   Starting broker... completed with 0 plugins.
Dec 08 11:35:22 ip-172-31-23-208.ec2.internal systemd[1]: Started rabbitmq-server.service - Open source RabbitMQ server.
lines 1-25/25 (END)
```

- **Tomcat:** (verify later after deployment)



If any instance fails, simply **terminate and relaunch** (no need to troubleshoot because the setup is 100% automated).

Step 3: Configure Route 53 Private Hosted Zone (Internal DNS Resolution)

Our `vprofile` application (running on **app01**) needs to connect to its backend services:

- MySQL (`db01`)
- Memcache (`mc01`)
- RabbitMQ (`rmq01`)

But how does the application know **which IP** each service is running on?

Let's break this down.

Why We Need Internal DNS (Instead of IPs)

If you open the source code:

src/main/resources/application.properties

You will notice entries like:

db01
mc01
rmq01

These are **hostnames**, not IPs.
And that's a good thing.

If we directly put **IP addresses** here, then any time a backend instance is replaced or recreated, the IP will change and the application would break. We would have to manually edit the configuration every time — NOT a good practice.

Instead, configuration files should always use **names**, not IPs.

Earlier in local setups we used `/etc/hosts` to map names → IPs, but in AWS that is NOT a scalable or professional approach.

We need a proper private DNS to resolve:

db01 → private IP of db01 server
mc01 → private IP of memcache server
rmq01 → private IP of rabbitmq server

Solution: AWS Route 53 Private Hosted Zone

AWS Route 53 is the DNS service we use to create internal name-to-IP mappings.

We don't need a public DNS here.
We only want private, internal resolution for our VPC.

Steps to Create the Private Hosted Zone

1. Go to **Route 53** → **Hosted Zones**
2. Click **Create Hosted Zone**

Enter the domain name.
Example:

vprofile.in

3.
4. Choose **Private Hosted Zone**
5. Select your **VPC (Default VPC)**
6. Click **Create**

Now your private DNS zone is ready.

Create A-Records for Each Backend Server

Now open each backend instance and copy **its private IP** (never the public IP!).

We will create A-records (Name → IP) for:

1. MySQL Server

Record Name:

db01.vprofile.in

Record Type: A
Value: Private IP of db01

2. Memcache Server

Record Name:

mc01.vprofile.in

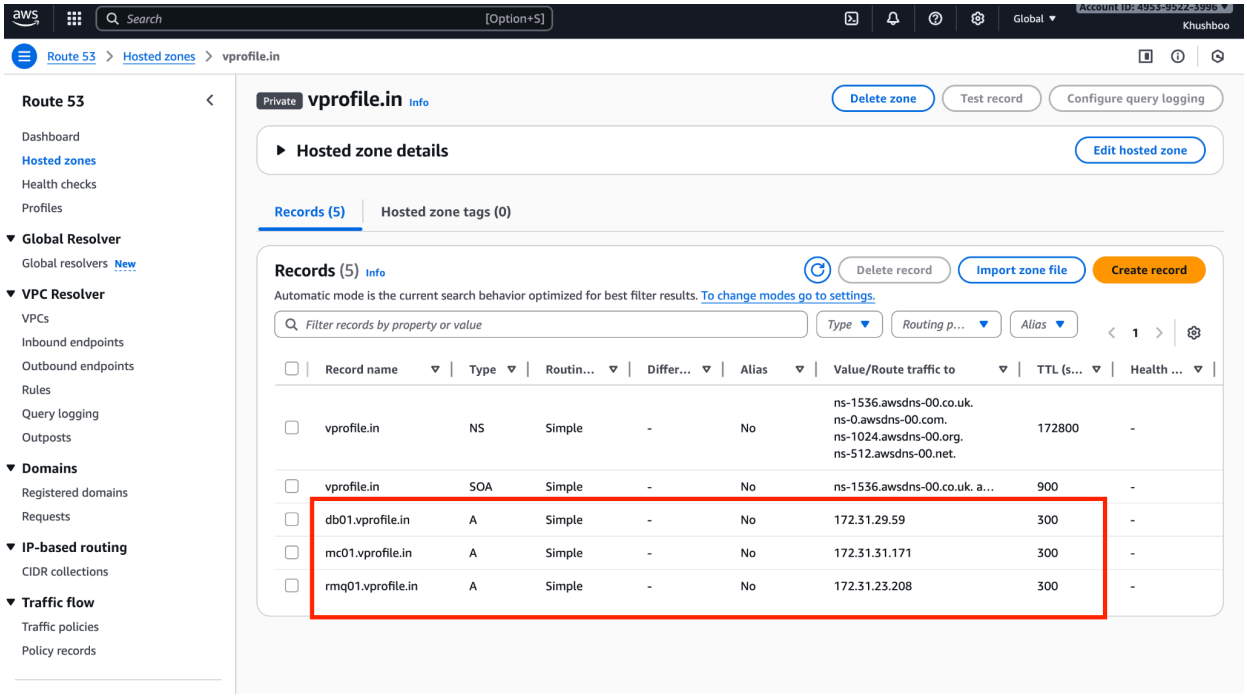
Record Type: A
Value: Private IP of mc01

3. RabbitMQ Server

Record Name:

rmq01.vprofile.in

Record Type: A
Value: Private IP of rmq01



Validate Your DNS Records

Once all records are created, SSH into `app01`.

Use its **public IP** to connect:

ssh ubuntu@<app01-public-ip>

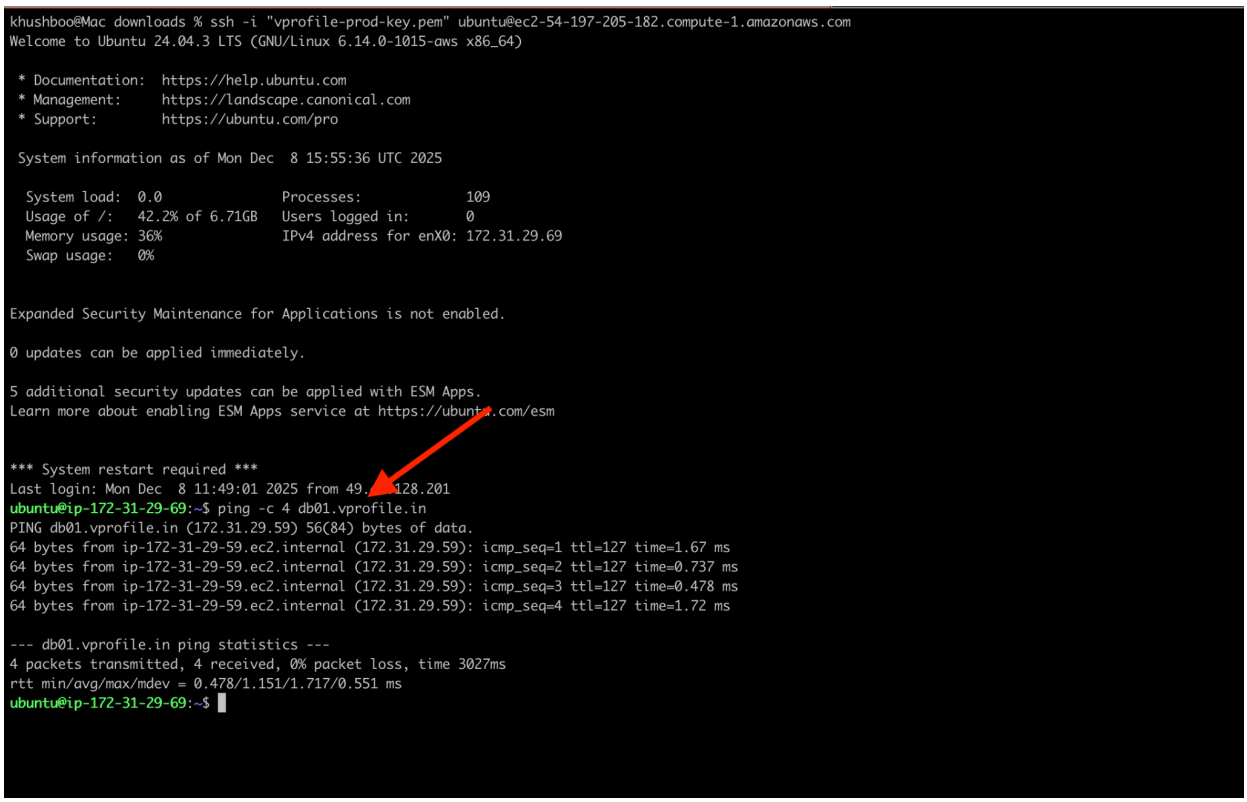
Now test DNS resolution:

ping -c 4 db01.vprofile.in

You should see replies from the **private IP of db01**.
Validate the same for:

mc01.vprofile.in
rmq01.vprofile.in

If all look correct and resolving properly, you're ready to move forward.



Building & Deploying the Artifact to Tomcat Using S3

Now that our backend setup is complete, the next major step is to **build the application artifact**, upload it to **Amazon S3**, and finally **deploy it to our Tomcat server (app01)**.

This entire workflow happens in three stages:

1. Prepare AWS components (S3 bucket, IAM user/keys, IAM role)
2. Build the `.war` file on your local machine using Maven
3. Upload to S3 → Download on EC2 → Deploy to Tomcat

Let’s break it down.

Step 1: Create AWS Resources (S3 Bucket, IAM Keys, IAM Role)

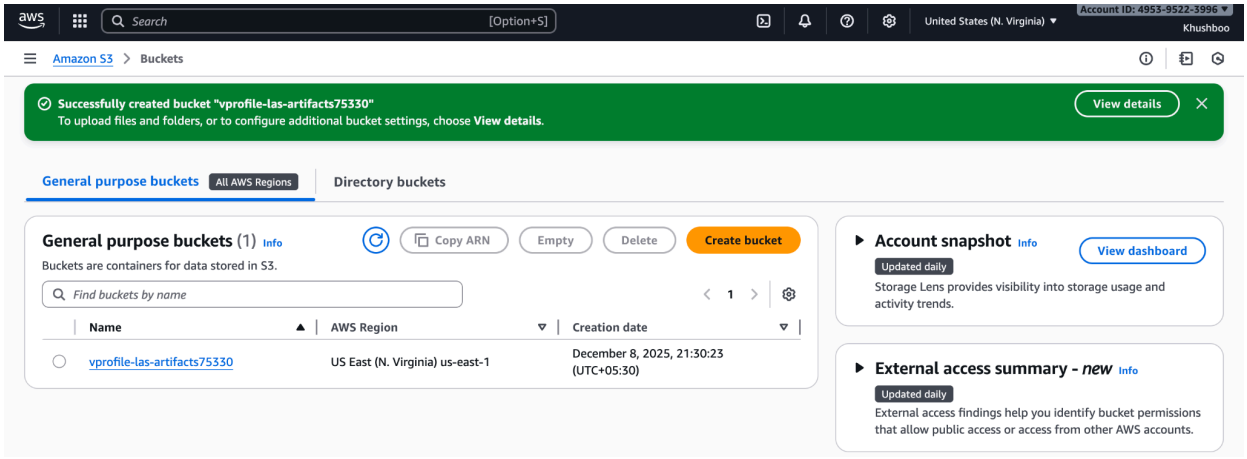
We start with AWS because both our **local machine** and **Tomcat instance** need authenticated access to S3.

1. Create an S3 Bucket

Go to **S3 → Create bucket**

- Name it something unique, e.g. `vprofile-lab-artifacts-1234`
- Keep all default settings
- Create the bucket

Take note of the bucket name — we'll use it in CLI commands.

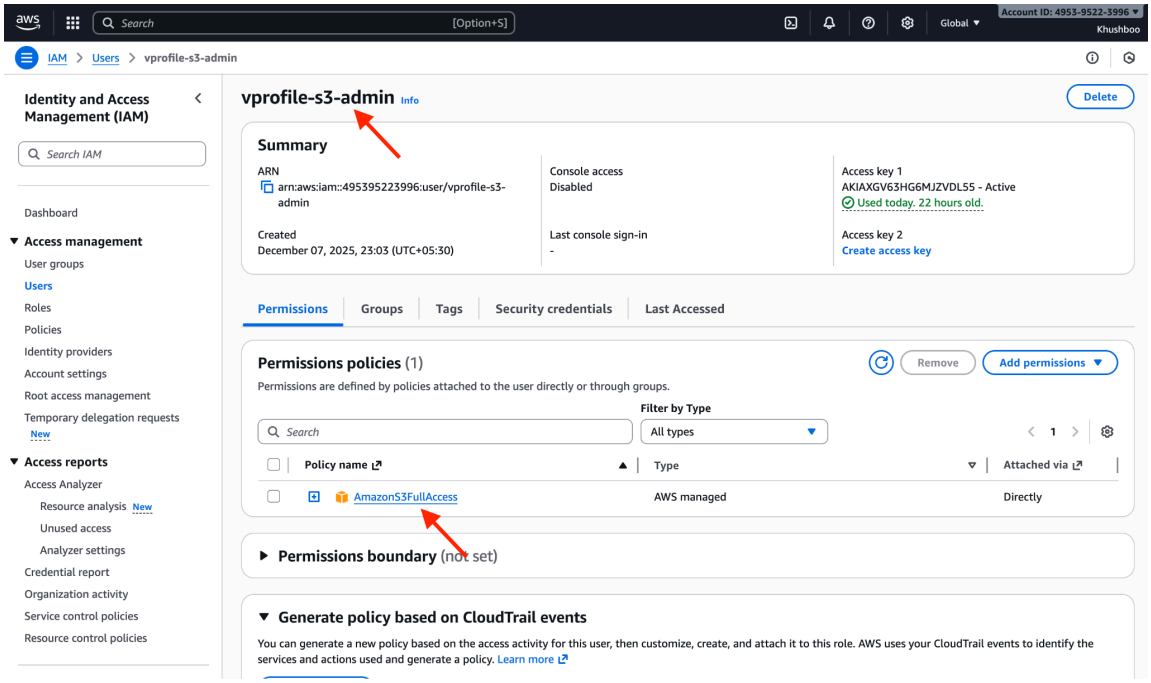


2. Create IAM User (For Local Machine Access)

We need programmatic access from our laptop to push the artifact to S3.

- Go to **IAM → Users → Create User**

- Name: `vprofile-s3-admin`
- Choose **Attach policies directly**
- Search **S3** → Select **AmazonS3FullAccess**
- Create user



Now generate the **Access Key**:

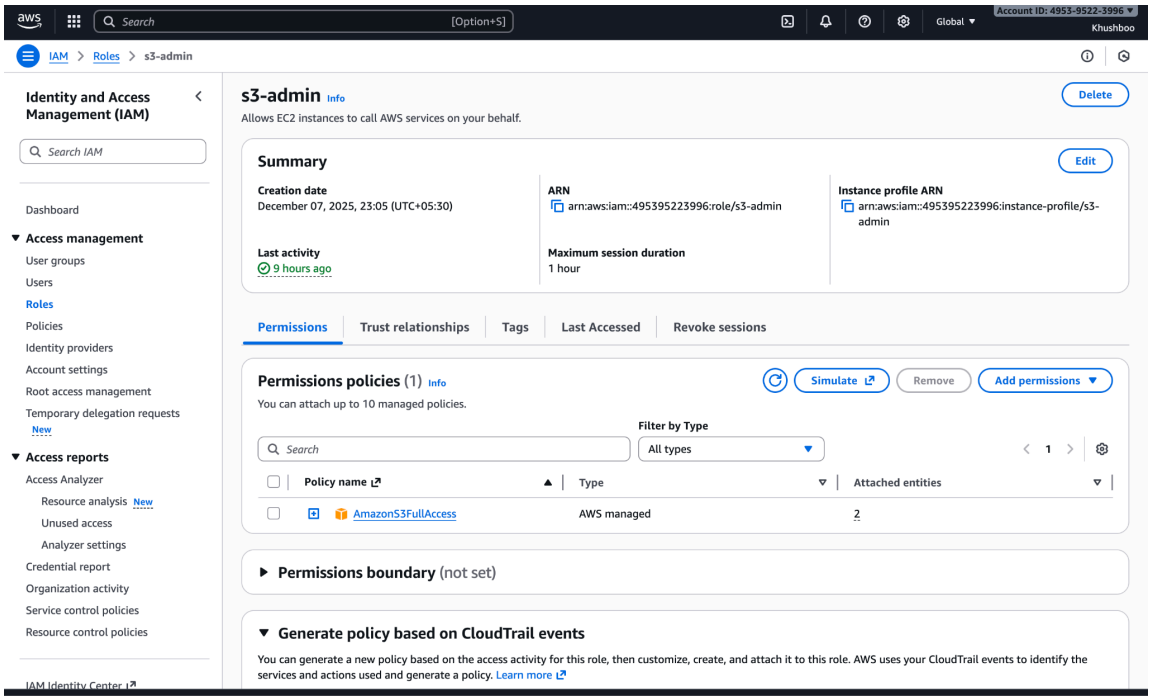
- Open the user → **Security credentials**
- Click **Create access key**
- Choose **CLI**
- Download the CSV file

💡 *Store it securely. If lost, you must generate new keys.*

3. Create IAM Role (For Tomcat EC2 Instance)

This role allows the EC2 instance to **pull artifacts from S3**.

- Go to **IAM → Roles → Create Role**
- Use case: **EC2**
- Add permission: **AmazonS3FullAccess**
- Name it: `S3-admin-role`
- Create role



Apply the role to Tomcat instance:

- Go to **EC2 → Instances → app01**
- Actions → Security → **Modify IAM Role**
- Select the role

Now the instance can access S3 without any keys.

Step 2: Prepare Your Local Machine (Maven, JDK, AWS CLI)

To build the application:

- **Maven** (3.9.9 recommended)
- **JDK 17**
- **AWS CLI**

Check versions inside VS Code terminal (or Git Bash):

mvn -version
java -version
aws --version

```
Mac:vprofile-project khushboo$ mvn --version
Apache Maven 3.9.10 (5f519b97e944483d878815739f519b2eade0a91d)
Maven home: /opt/homebrew/Cellar/maven/3.9.10/libexec
Java version: 17.0.16, vendor: Homebrew, runtime: /opt/homebrew/Cellar/openjdk@17/17.0.16/libexec/openjdk.jdk/Contents/Home
Default locale: en_IN, platform encoding: UTF-8
OS name: "mac os x", version: "15.6", arch: "aarch64", family: "mac"
Mac:vprofile-project khushboo$
```

```
Mac:vprofile-project khushboo$ aws

usage: aws [options] <command> <subcommand> [<subcommand> ...] [parameters]
To see help text, you can run:

  aws help
  aws <command> help
  aws <command> <subcommand> help

aws: error: the following arguments are required: command

Mac:vprofile-project khushboo$
```

If versions differ, install them from the prerequisites section.

Step 3: Update Backend DNS in Code

Before building the artifact, update backend hostnames in:

`src/main/resources/application.properties`

Make sure they match the Route 53 private hosted zone:

db01.vprofile.in
mc01.vprofile.in

rmq01.vprofile.in

Save the file.

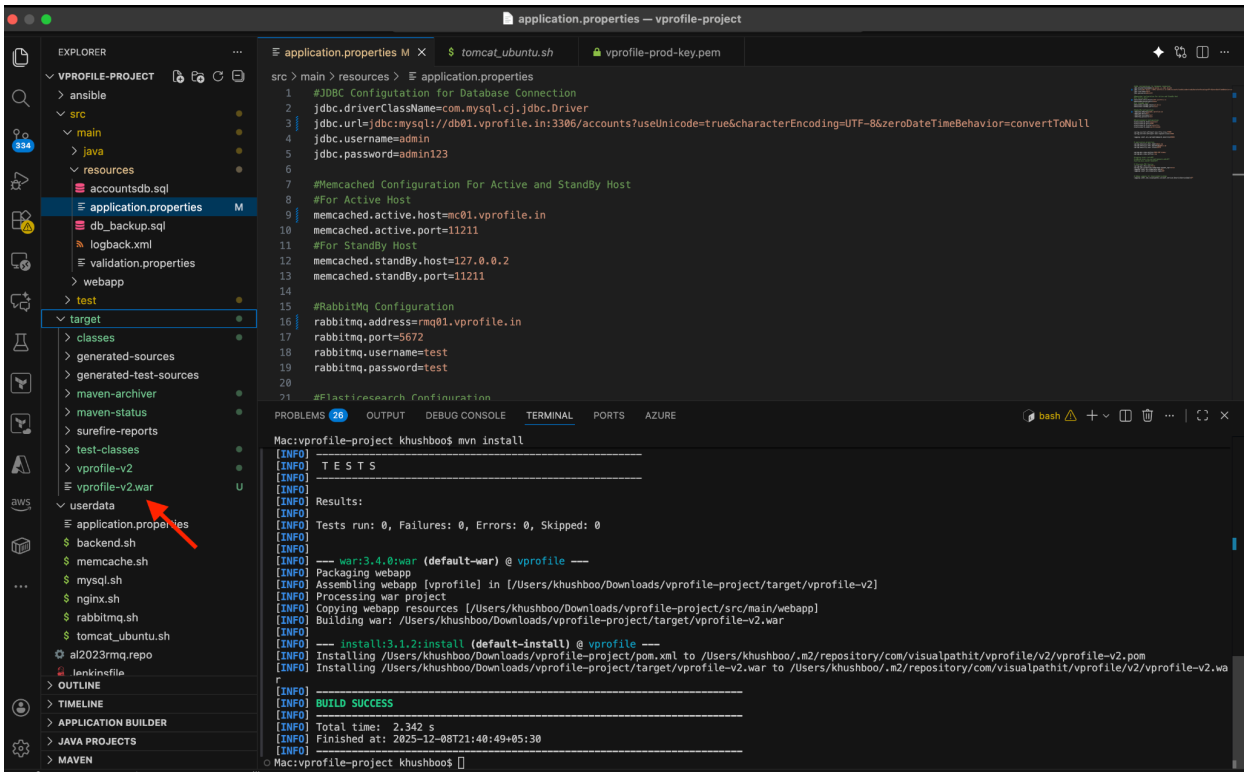
Step 4: Build the Artifact Using Maven

Navigate to the project folder and run:

mvn install

After successful build, you will see:

target/vprofile-v2.war



This is the artifact we will upload.

Step 5: Upload the Artifact to S3

First configure your access keys:

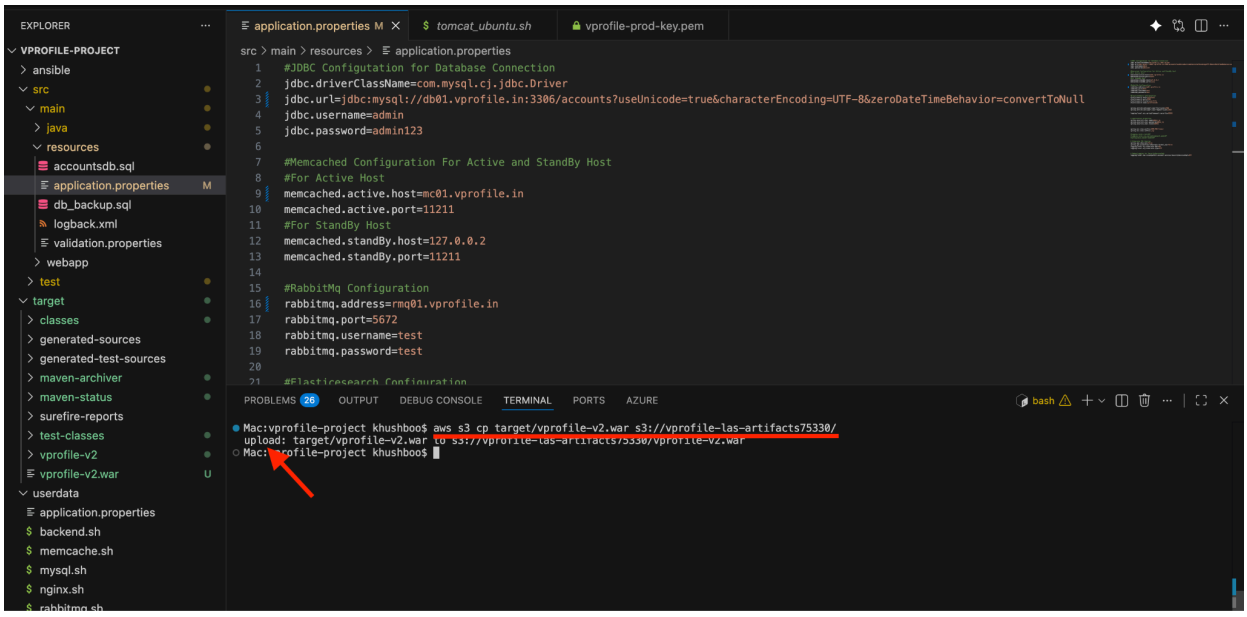
aws configure

Enter:

- Access Key
- Secret Key
- Region (e.g., `us-east-1`)
- Output format: `json`

Now upload:

aws s3 cp target/vprofile-v2.war s3://your-bucket-name/



Check upload:

aws s3 ls s3://your-bucket-name/

Step 6: Download Artifact on Tomcat Server

SSH into the Tomcat instance:

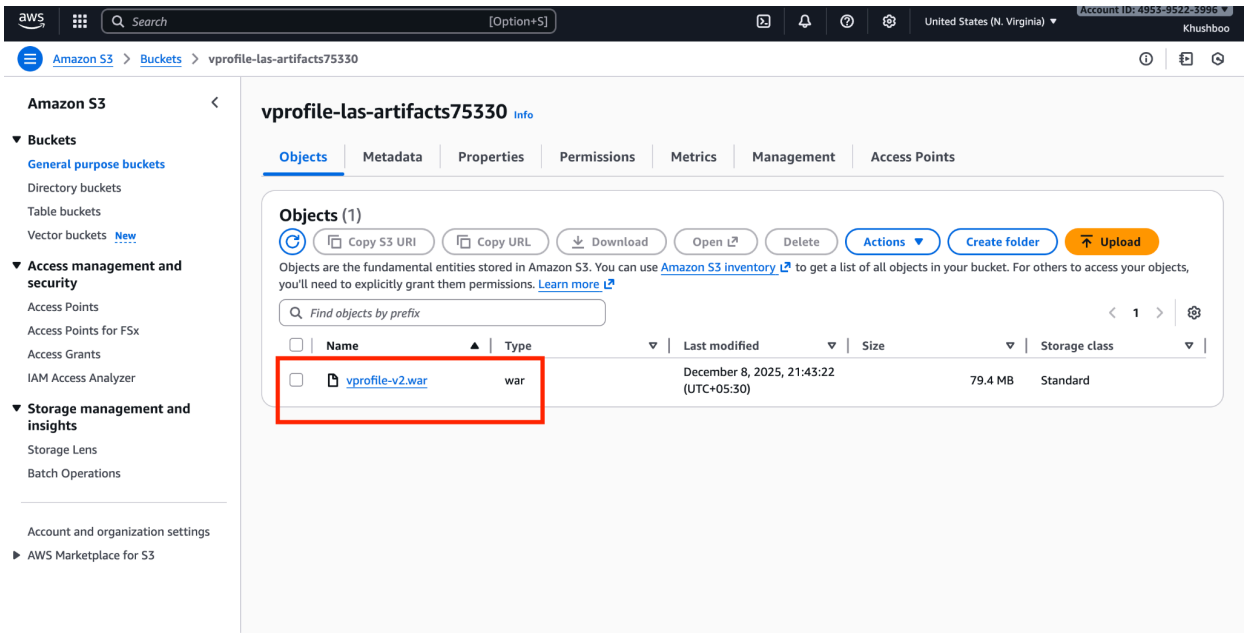
ssh -i ~/Downloads/vprofile-prod-key.pem ubuntu@<Public-IP>

Install AWS CLI (Ubuntu):

sudo snap install aws-cli --classic

Download the artifact:

aws s3 cp s3://your-bucket-name/vprofile-v2.war /tmp/



Step 7: Deploy the Artifact to Tomcat

```
download: s3://vprofile-las-artifacts75330/vprofile-v2.war to ../tmp/vprofile-v2.war
root@ip-172-31-29-69:~# ls -l /tmp/
total 81356
drwxr-xr-x 2 root root     4096 Dec  8 11:37 hsperfdata_root
drwx------ 2 root root     4096 Dec  8 11:36 snap-private-tmp
drwx------ 3 root root     4096 Dec  8 11:36 systemd-private-9af56b6b22974a879c120afca0c28be1-ModemManager.service-xAPKEo
drwx------ 3 root root     4096 Dec  8 11:36 systemd-private-9af56b6b22974a879c120afca0c28be1-chrony.service-jbISkI
drwx------ 3 root root     4096 Dec  8 11:36 systemd-private-9af56b6b22974a879c120afca0c28be1-polkit.service-APgfV7
drwx------ 3 root root     4096 Dec  8 11:36 systemd-private-9af56b6b22974a879c120afca0c28be1-systemd-logind.service-xw2Psq
drwx------ 3 root root     4096 Dec  8 11:36 systemd-private-9af56b6b22974a879c120afca0c28be1-systemd-resolved.service-zBmpca
drwx------ 3 root root     4096 Dec  8 11:38 systemd-private-9af56b6b22974a879c120afca0c28be1-tomcat10.service-4DKE3l
-rw-r--r-- 1 root root 83275399 Dec  8 16:13 vprofile-v2.war
root@ip-172-31-29-69:~# systemctl stop tomcat10
Warning: The unit file, source configuration file or drop-ins of tomcat10.service changed on disk. Run 'systemctl daemon-reload' to reload units.
root@ip-172-31-29-69:~# systemctl daemon-reload
root@ip-172-31-29-69:~#
```

Stop Tomcat:

sudo systemctl stop tomcat10
sudo systemctl daemon-reload

Remove default ROOT app:

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

```
root@ip-172-31-29-69:~# ls -l /var/lib/tomcat10/webapps/
total 4
drwxr-xr-x 3 root root 4096 Dec  8 11:38 ROOT
root@ip-172-31-29-69:~# rm -rf /var/lib/tomcat10/webapps/ROOT
root@ip-172-31-29-69:~# ls /var/lib/tomcat10/webapps/
root@ip-172-31-29-69:~#
```

Deploy new artifact:

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

Start Tomcat:

sudo systemctl start tomcat10

Tomcat will automatically extract the WAR:

ls -l /var/lib/tomcat10/webapps/

```
drwxr-xr-x 3 root root 4096 Dec  8 11:38 ROOT
root@ip-172-31-29-69:~# rm -rf /var/lib/tomcat10/webapps/ROOT
root@ip-172-31-29-69:~# ls /var/lib/tomcat10/webapps/
root@ip-172-31-29-69:~# cp /tmp/vprofile-v2.war /var/lib/tomcat10/webapps/ROOT.war
root@ip-172-31-29-69:~# systemctl start tomcat10
root@ip-172-31-29-69:~# ls /var/lib/tomcat10/webapps/
ROOT  ROOT.war
root@ip-172-31-29-69:~#
```

You should now see a `ROOT` directory created from the WAR.

Step 4: Testing the Application & Creating the Application Load Balancer (ALB)

Now that all backend services are installed and configured on **db01**, **mc01**, **rmq01**, and our application is deployed on **app01**, it's time to verify that our Tomcat application is running correctly. Once verified, we'll configure the **AWS Application Load Balancer (ALB)** so users can access the application through a single, secure endpoint.

1. Test the Tomcat Application Running on app01

Before building the Load Balancer, we must ensure Tomcat is responding.

Update Security Group Temporarily

Go to the **app security group** (vprofile-app-sg) and add:

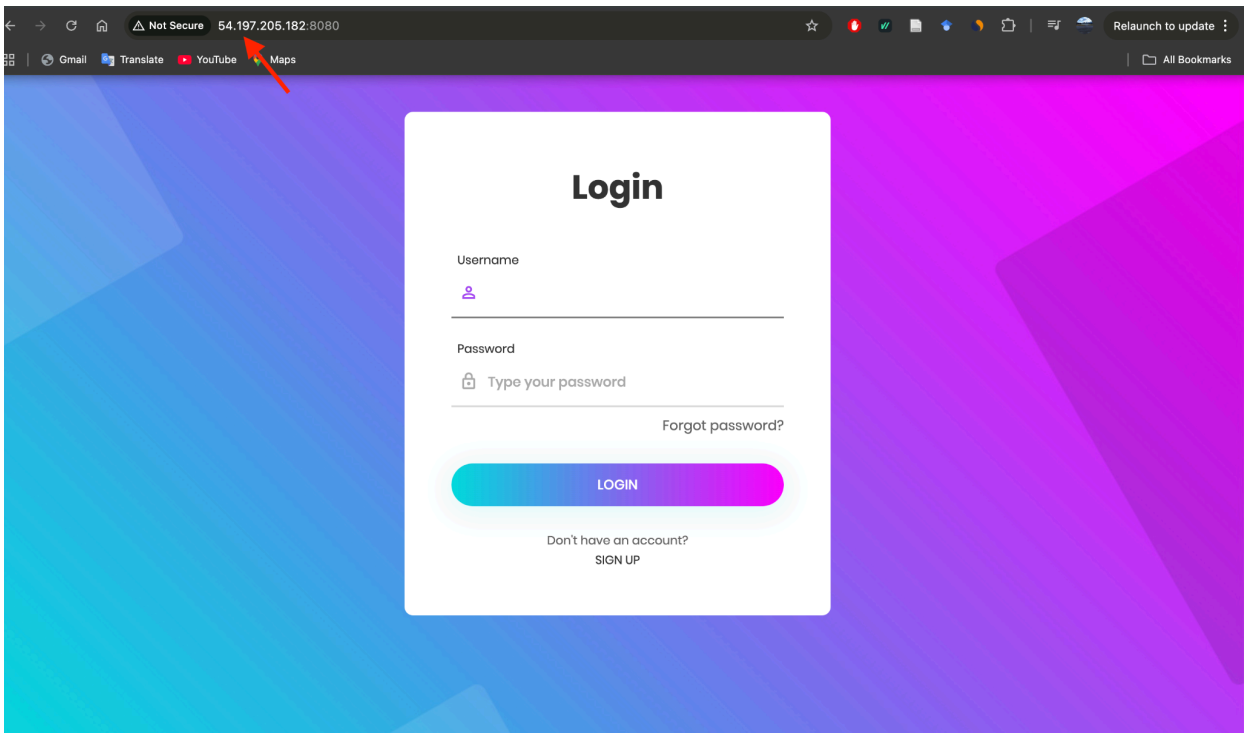
- **Port 8080** → Source: **My IP**

This is temporary, only for verification.

Test in Browser

1. Copy the **public IP of app01**.
   Open your browser and enter: http://<APP01_PUBLIC_IP>:8080

2. You should see the **vProfile login page**, confirming Tomcat + WAR deployment is working.



Once confirmed, close the tab—we will not access the app directly anymore.

2. Create the Target Group for the Load Balancer

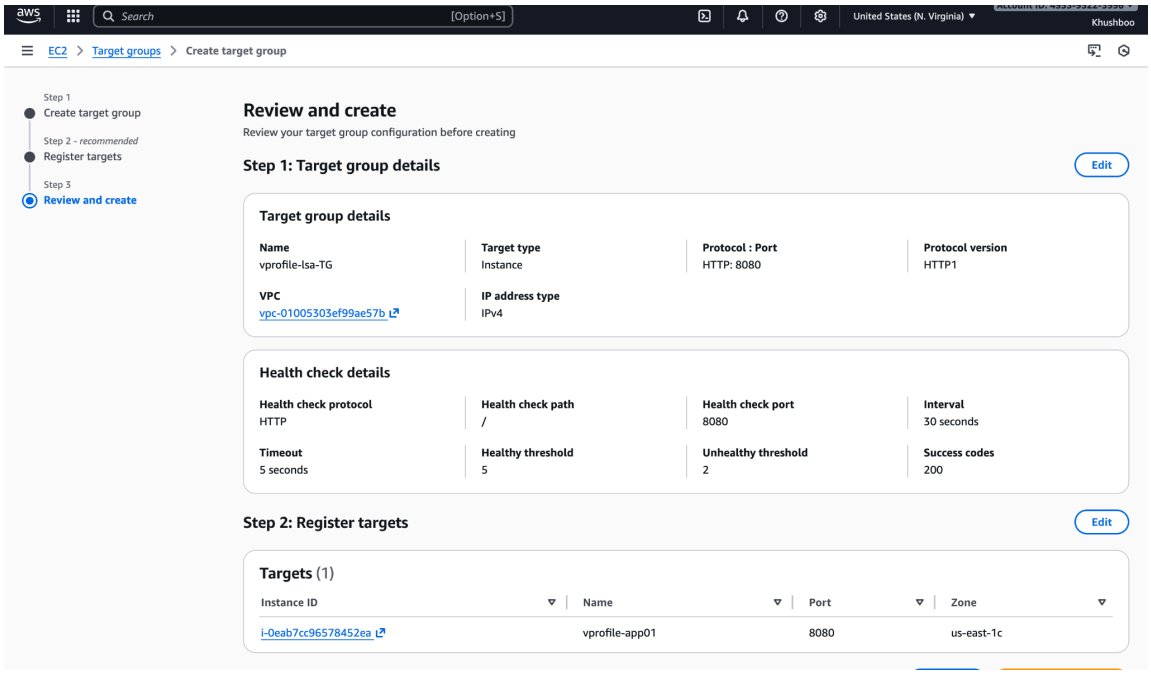
The Load Balancer will forward traffic to app01 through a **Target Group**.

Steps

1. Go to **EC2 → Load Balancing → Target Groups → Create Target Group**.
2. Select **Instances** as the target type.
3. Name it: **vprofile-las-tg**
4. Protocol: **HTTP**, Port: **8080**
5. Scroll down → **Health Check > Advanced settings**

- Override port to **8080** (default is 80)

6. Click **Next**.
7. Register **app01** and ensure:
   - **Port = 8080**
8. Create the Target Group.



3. Create the Application Load Balancer (ALB)

We'll configure **HTTP (80)** and **HTTPS (443)** listeners.
HTTPS requires an **ACM certificate**, which we created earlier.

If you don't have a domain or certificate, you can still proceed with HTTP only.

Steps

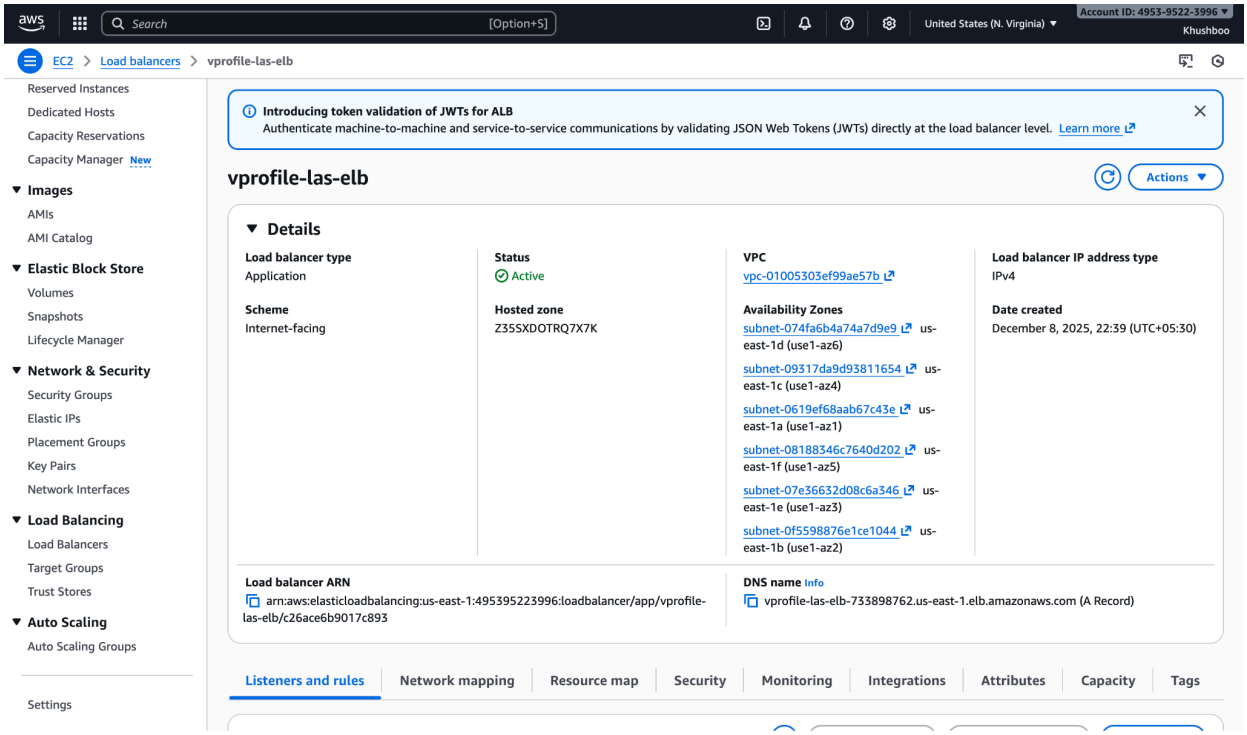
1. Go to **Load Balancers → Create Load Balancer**
2. Choose **Application Load Balancer**

3. Name it: **vprofile-las-elb**
4. Scheme: **Internet-facing**
5. Select your **VPC** and all **Availability Zones**
6. Remove default SG → Add your **ELB security group**

Configure Listeners

- Listener 1: **HTTP 80** → forward to **vprofile-las-tg**
- Listener 2: **HTTPS 443**
    - Forward to same target group
    - Choose ACM certificate from dropdown
    - Keep default SSL policy

Create the load balancer.



ALB status will show **provisioning**, then become **active**.

4. (Optional) Map Your Domain Using CNAME

If you purchased a domain (GoDaddy, Namecheap, etc.):

1. Go to your domain's DNS panel.
2. Create a **CNAME record**:
    - **Name:** vprofileapp
    - **Value:** *Load Balancer DNS name*
3. Save.

Now your application is accessible at:

https://vprofileapp.<your-domain>.com

If you didn't buy a domain, simply use the **ALB DNS endpoint** in your browser.

5. Verify ALB Health Checks

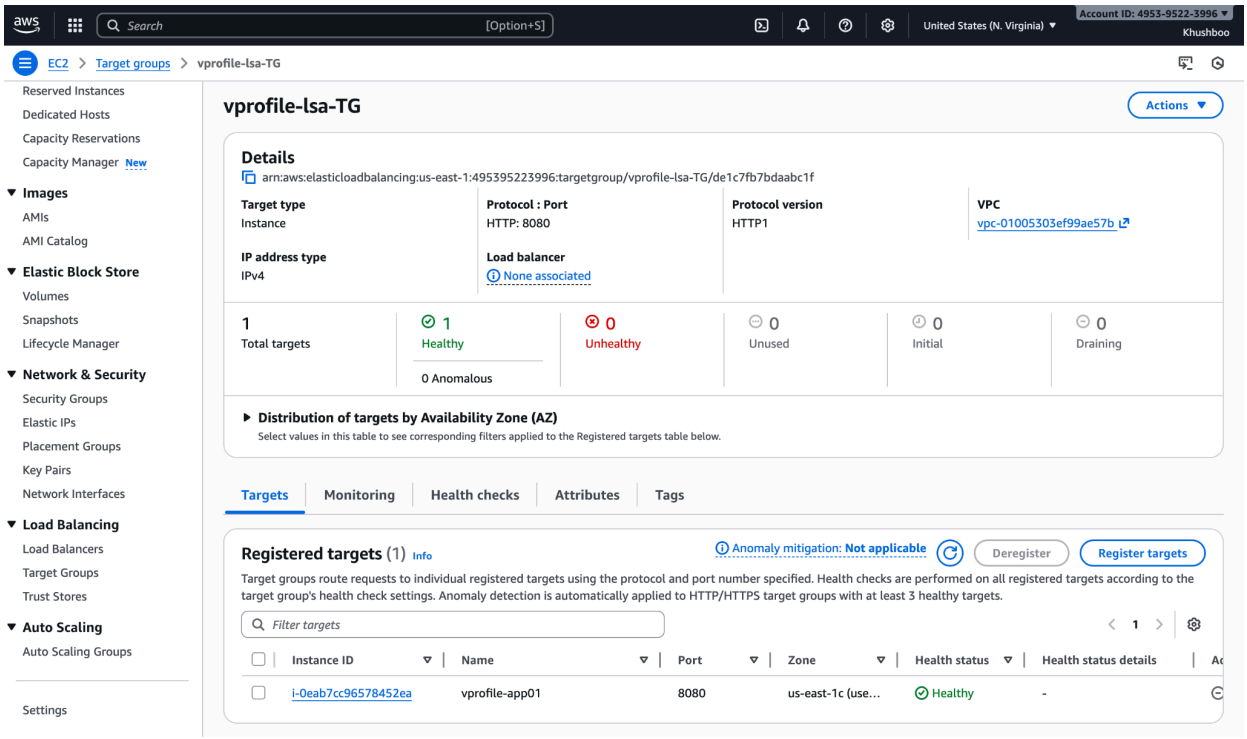
Go to:
EC2 → Target Groups → vprofile-las-tg → Targets

Status should show:

healthy

If not, check:

- Tomcat service running
- SG of app01 allows 8080 from ELB SG
- Health check port = 8080

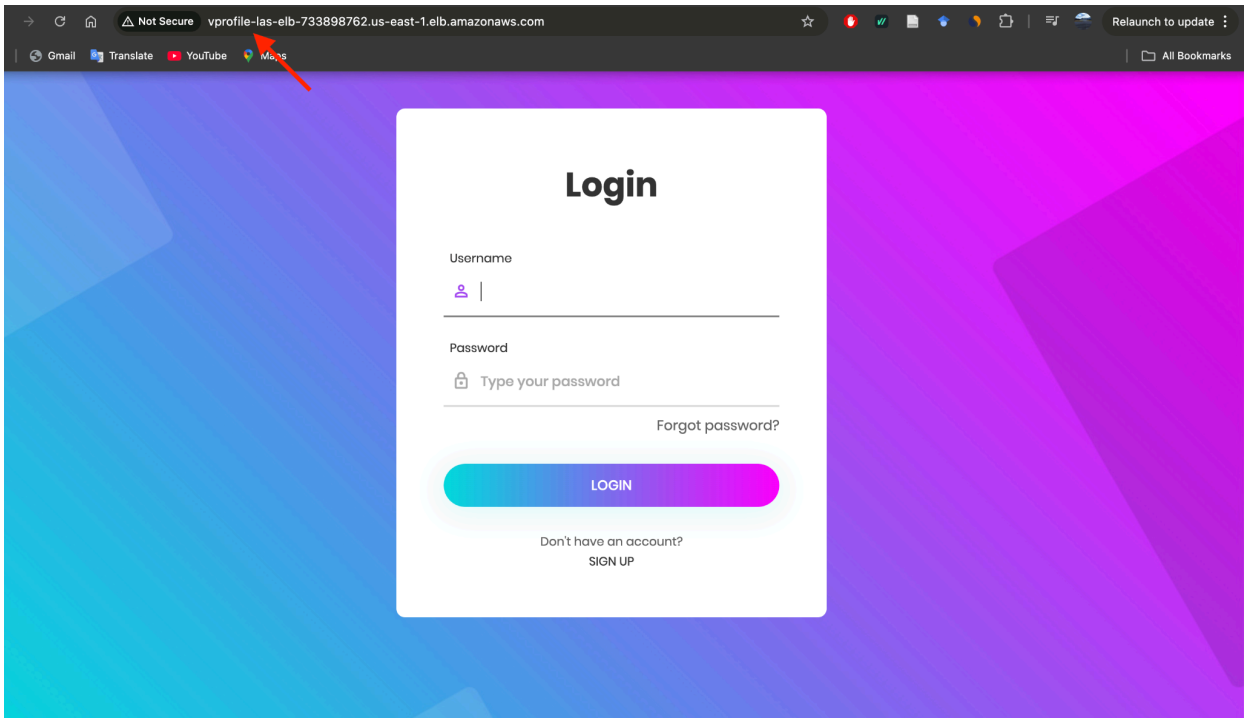


6. Test the Application Through the Load Balancer

HTTP Test

Use the ALB DNS name:

http://<ALB-DNS-NAME>

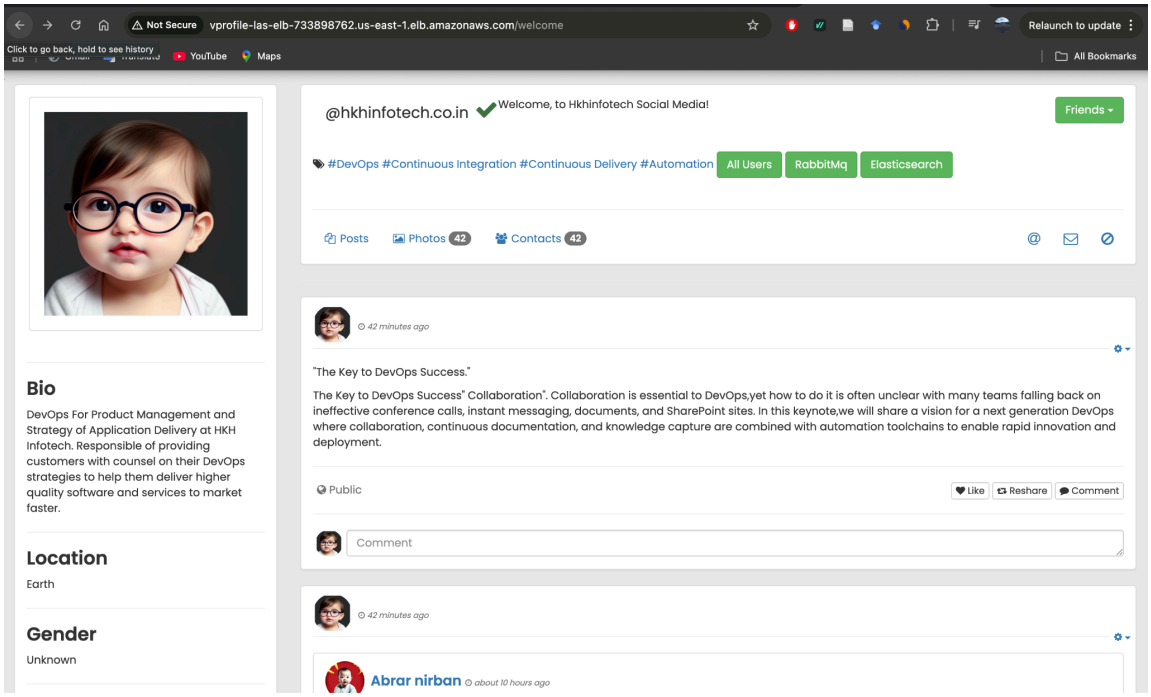


You should see the login page (but “Not secure”).

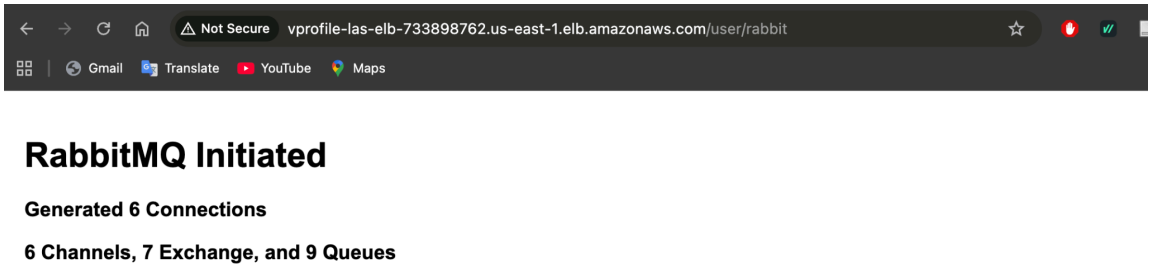
7. Validate Backend Connectivity

After logging in:

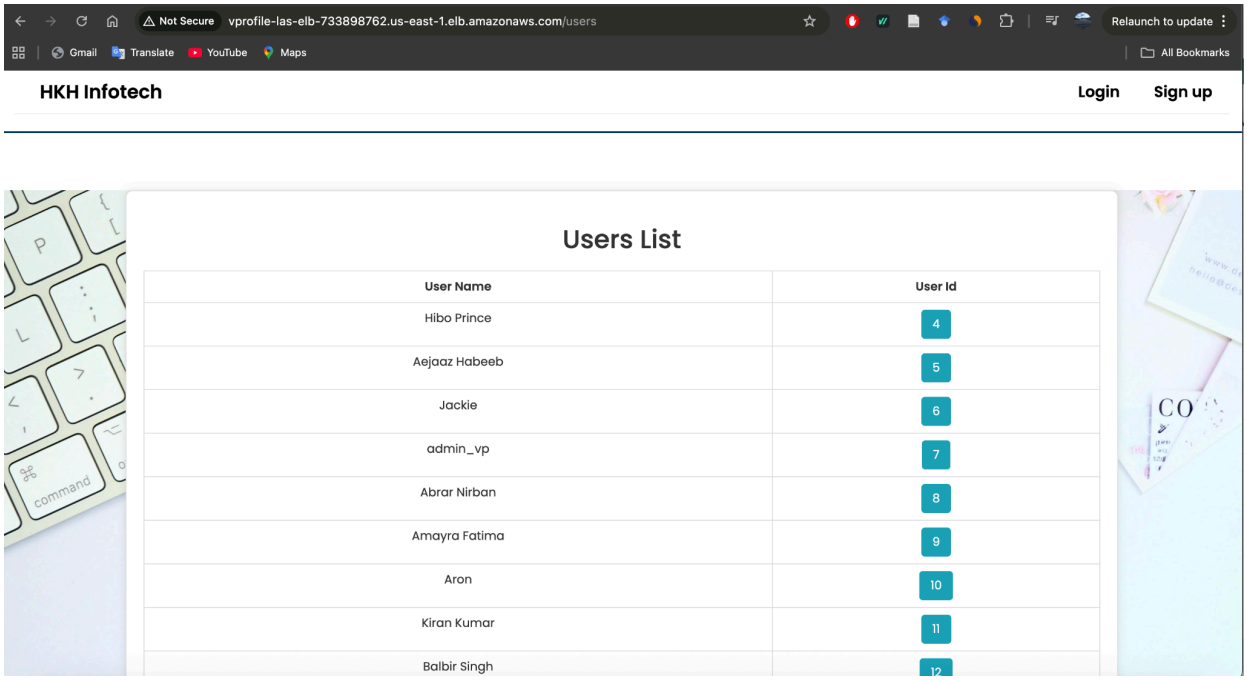
- **DB connection** works → Login success



- **RabbitMQ connection** → Queue appears



- **Memcache connection**
    - First fetch → “data is from DB”
    - Second fetch → “data is from cache”



This confirms every component—DB, cache, queue, app server, ALB—is successfully wired together.

Step 8 — Creating the AMI, Launch Template & Auto Scaling Group (ASG)

Now that our application is running successfully on the app server, it is time to make the environment **scalable**.
To achieve this, we will configure an **Auto Scaling Group (ASG)** so that new instances can be created automatically when the load increases — and removed when the load decreases.

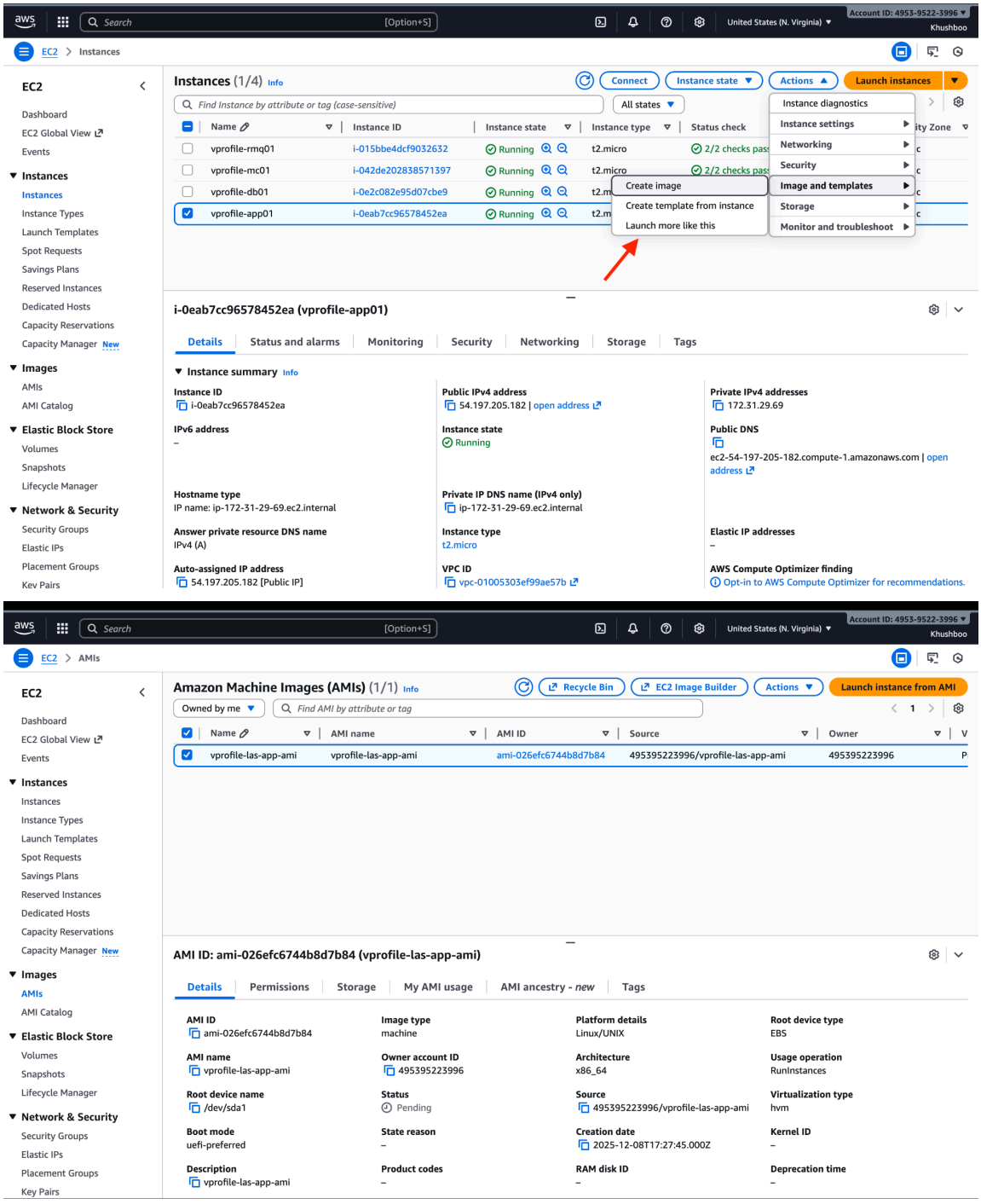
AWS Auto Scaling requires the following three components:

1. **AMI (Amazon Machine Image)** – a full snapshot of our app server
2. **Launch Template** – instructions on how new instances should be created
3. **Auto Scaling Group** – the scaling logic and capacity rules

Let's walk through each step.

1. Create an AMI of the App Server

1. Go to the **EC2 console** → select your **app01** instance
2. Click **Actions** → **Image and templates** → **Create image**
3. Give it a name:
   `vprofile-las-app-ami`
4. Add the same value in the description
5. Click **Create image**

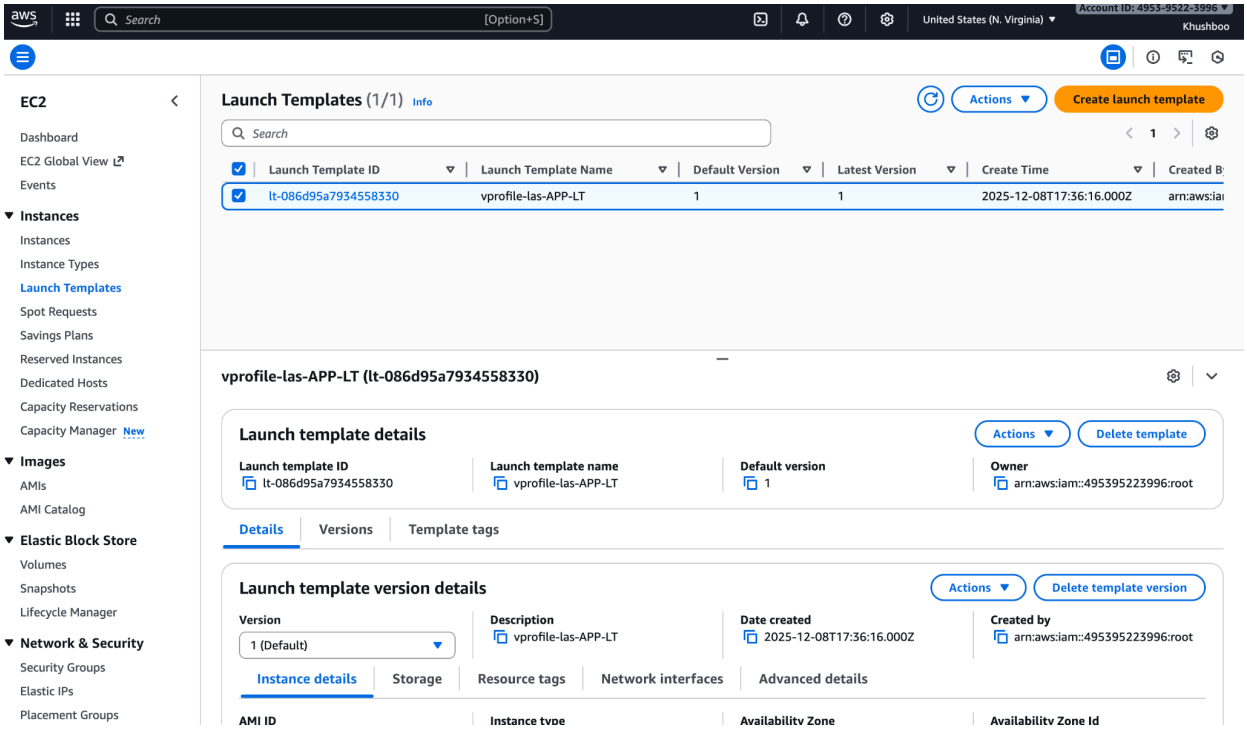


You will be redirected to the AMI page.
The image will stay in **pending** state for a few minutes.

2. Create a Launch Template

While the AMI is being created, we can prepare the **Launch Template**.

1. Go to **EC2** → **Launch Templates** → **Create launch template**
2. Name the template:
   `vprofile-las-app-LT`
3. Under **AMI**, select **My AMIs** and choose the image we just created
4. Choose the instance type — `t2.micro` or `t3.micro`
5. Select the key pair: `vprofile-prod-key`
6. Under **Network Settings**, choose:
   - **App Security Group**
7. Add tags:
   - `Name = vprofile-app`
   - `Project = vprofile`
     (Tag both **instances** and **volumes**)
8. Expand **Advanced details**
   - IAM role is optional



Click **Create launch template**.

3. Create the Auto Scaling Group (ASG)

1. Go to **Auto Scaling Groups → Create Auto Scaling Group**
2. Name it:
   `vprofile-las-app-asg`
3. Select the launch template created above
4. Choose your VPC and **all Availability Zones**
5. Attach the existing **Application Load Balancer**
    - Choose the **target group** you created earlier

6. Enable **Elastic Load Balancing health checks**
   This ensures ASG replaces even *unhealthy* instances (not just stopped ones)

Set scaling values:

- **Desired capacity:** 1
- **Minimum capacity:** 1
- **Maximum capacity:** 2

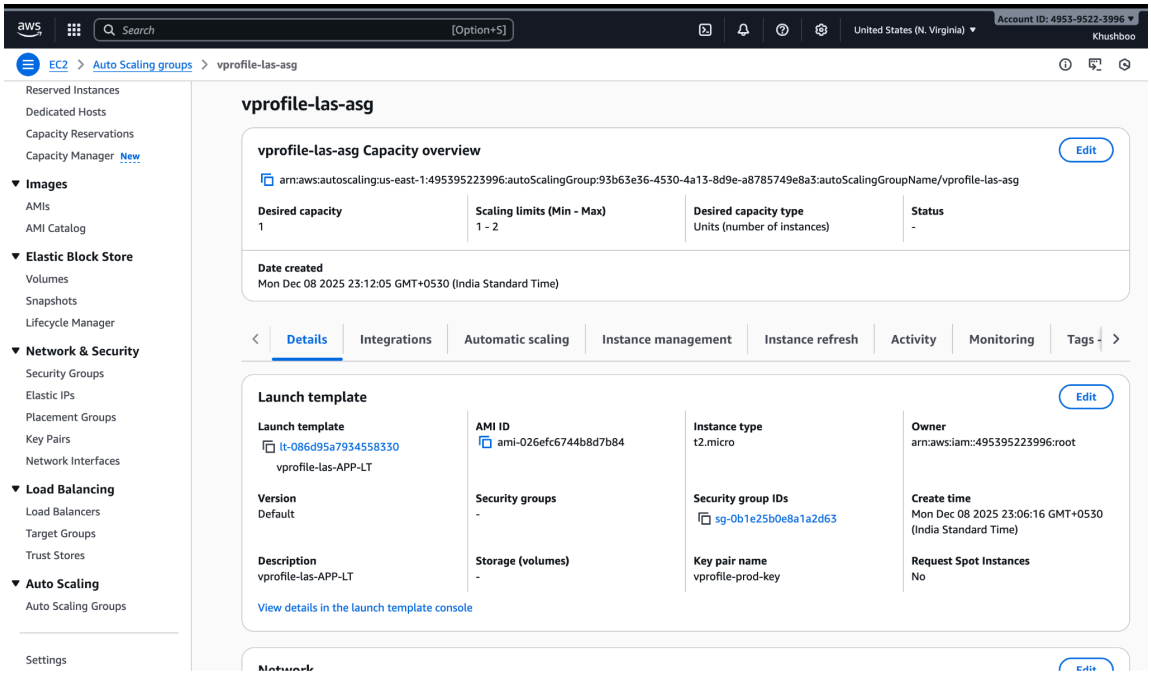
Configure scaling policy:

Use **Target Tracking Scaling Policy**

- Metric: **Average CPU Utilization**
- Target value: **50%**

This means:

- If CPU > 50% → **scale out**
- If CPU < 50% → **scale in**



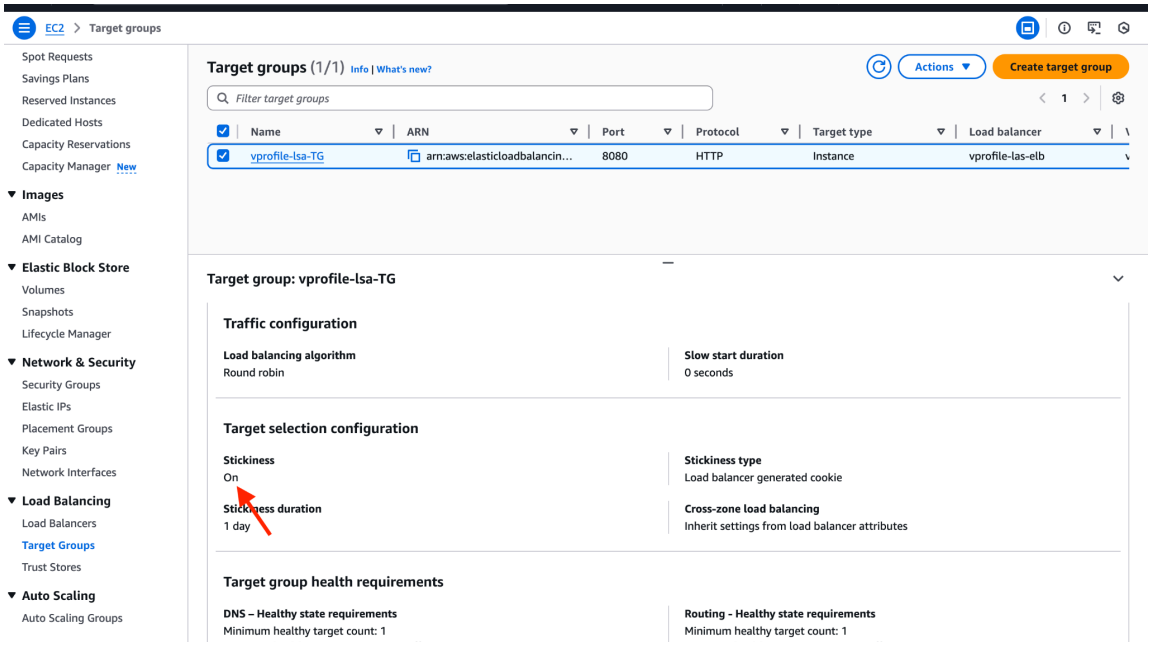
Click **Next** and create the ASG.

4. Enable Stickiness on the Target Group (Important for vProfile)

This app requires session stickiness, otherwise users get logged out when switching between instances.

1. Go to **EC2 → Target Groups**
2. Select your target group
3. Open **Attributes → Edit**
4. Enable **Stickiness**

5. Save changes



This ensures every user is routed to the same app instance.

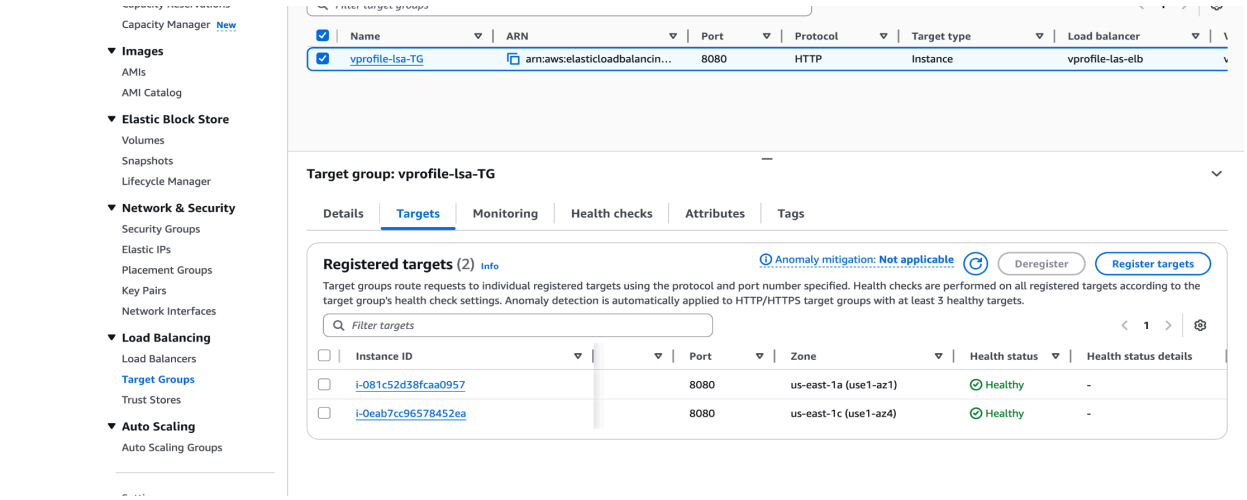
5. Clean Up the Old App Server

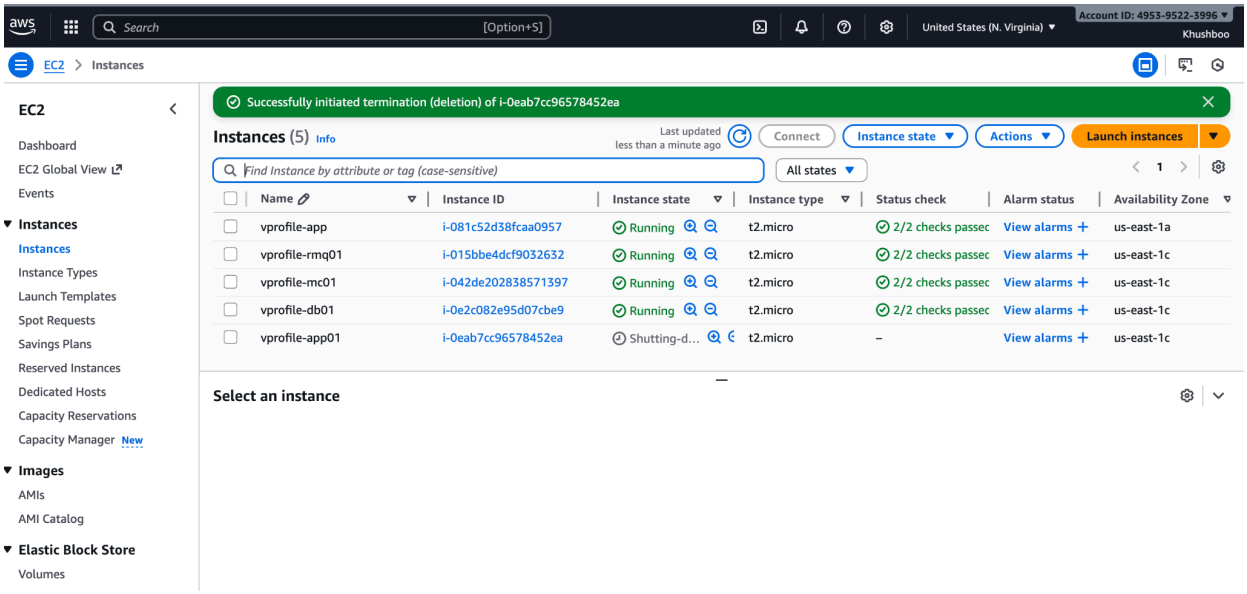
Now the ASG has launched a new app instance.
To avoid conflicts:

1. Go to **Target Groups → Targets**

2. Deregister the old **app01** instance

3. After draining completes, go to EC2 and **terminate app01**





The Auto Scaling Group will now manage all app instances automatically.

Step 9 — Validating the Deployment & Cleaning Up AWS Resources

At this point, your Auto Scaling Group should be running **one healthy app instance** registered under the target group. The original **app01** instance should already be terminated, and all traffic is now handled by the ASG-managed instance.

Before closing this project, let's validate the deployment and then walk through a clean and safe teardown procedure.

Validate the Application via Load Balancer

If you **don't** have a domain name configured:

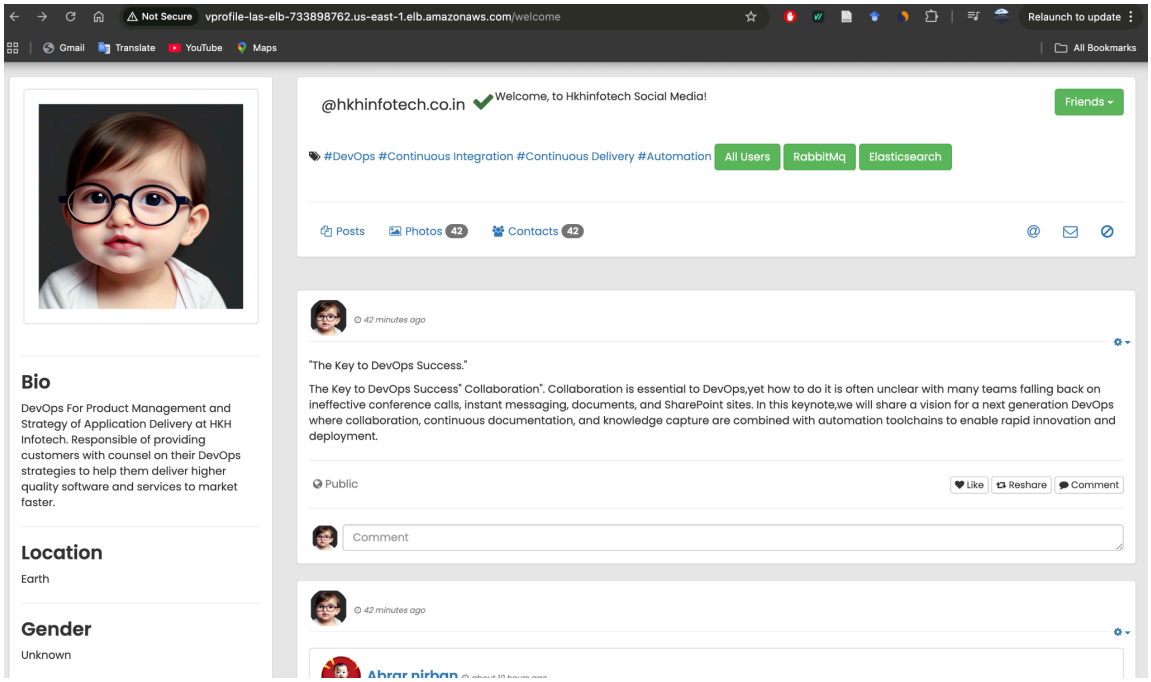
- Simply copy your **Load Balancer DNS name**
- Paste it in the browser
- Access the vProfile application directly

Use the login credentials:

- **Username:** `admin_vp`
- **Password:** `admin_vp`

Once logged in, verify that all components are working:

- **RabbitMQ**
- **Memcached**
- **Database connectivity**



If the login page refreshes without logging you in, it means **Target Group Stickiness** is disabled. For multi-instance environments, stickiness must be enabled to maintain session consistency.

What We Achieved

You've successfully taken a previously local application and deployed it end-to-end on AWS using real cloud services.
This was a **lift-and-shift migration**, meaning we replicated the existing architecture rather than redesigning it.

Later, in the next project, you will transform this application into a more cloud-native architecture.

Step 10 — Clean Up AWS Resources (Avoid Unwanted Costs)

Cleaning up is important because Auto Scaling, ALB, and EC2 resources can continue to incur charges if left running.

Follow this order **exactly**:

1. Delete the Auto Scaling Group

If you skip this step, AWS will recreate terminated instances automatically.

- Go to **EC2 → Auto Scaling Groups**
- Select your ASG
- Click **Delete**

This will also terminate any active app instances launched by the ASG.

2. Terminate Remaining EC2 Instances

After the ASG is deleted:

- Go to **EC2 → Instances**
- Select any remaining instances
- Click **Instance state → Terminate**

Wait a few seconds until they show as *terminated*.

3. Delete the Load Balancer

- Go to **EC2 → Load Balancers**
- Select your ALB
- Click **Delete**

ALBs incur hourly cost, so it’s important to remove them.

4. Delete or Keep Target Groups

Target groups do not cost anything.

You may keep them for reference or delete them as part of cleanup.

5. Cleanup Route 53 Private Hosted Zones

DNS hosted zones **cannot** be deleted until all records inside them are removed.

- Go to **Route 53 → Hosted Zones**
- Select your private hosted zone
- Delete all custom records
- Then delete the hosted zone itself

6. Remove Unused Key Pairs & Security Groups (optional)

They don’t cost anything, but for a clean project:

- Delete unused key pairs
- Delete security groups created during the project

7. Deregister the AMI and Delete Snapshots

AMI itself cannot be deleted — it is simply a mapping to snapshots.

Do this:

- Go to **EC2 → AMIs**

- Select your custom AMI
- Click **Deregister**

Then delete the underlying snapshot:

- Go to **Snapshots**
- Select the snapshot created during AMI creation
- **Delete**

8. Empty & Delete the S3 Bucket (optional)

If you created an S3 bucket for artifacts:

- Open the bucket
- Click **Empty**
- Confirm by typing *permanently delete*
- Delete the bucket

Architecture Recap — What We Built

Here's everything we accomplished in this project:

- Set up **Security Groups** and **Key Pairs**
- Launched all required **EC2 instances**
- Bootstrapped them using **user data scripts**
- Built the application artifact and deployed it using **S3**

- Created an **HTTPS Load Balancer** with **ACM certificate**
- Implemented **Target Groups** and health checks
- Added **Route 53 private DNS** for internal communication
- Created an **AMI** of app01
- Built a **Launch Template** from the AMI
- Configured an **Auto Scaling Group** to handle load automatically

This was a complete lift-and-shift migration running entirely on AWS infrastructure.

Final Thoughts

If you've followed the project to this point — congratulations.
This was not a small task. You built a full, production-style environment end-to-end on AWS.