

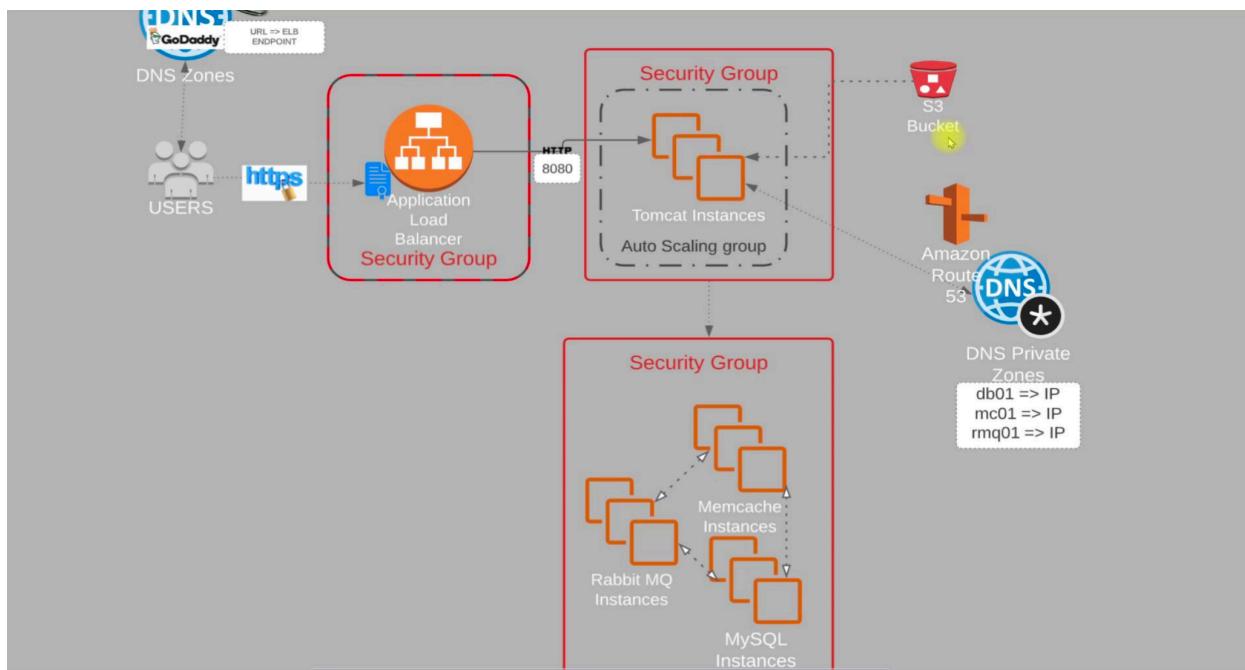
Complete Lift-and-Shift Migration of the vProfile Application to AWS (Step-by-Step with Full Commands & Outputs)

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.

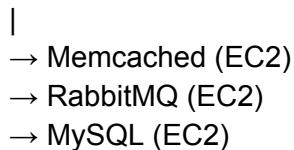
4. MySQL (Database Server)

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)



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 |



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/`

```

src > main > resources > application.properties
1 #JDBC Configuration for Database Connection
2 jdbc.driverClassName=com.mysql.cj.jdbc.Driver
3 jdbc.url=jdbc:mysql://db01.vprofile.in:3306/accounts?useUnicode=true&characterEncoding=UTF-8&zeroDateTimeBehavior=convertToNull
4 jdbc.username=admin
5 jdbc.password=admin123
6
7 #Memcached Configuration For Active and StandBy Host
8 #For Active Host
9 memcached.active.host=mc01.vprofile.in
10 memcached.active.port=1121
11 #For StandBy Host
12 memcached.standby.host=127.0.0.2
13 memcached.standby.port=11211
14
15 #RabbitMq Configuration
16 rabbitmq.address=rabbitmq1.vprofile.in
17 rabbitmq.port=5672
18 rabbitmq.username=test
19 rabbitmq.password=test
20
21 #Elasticsearch Configuration
22 elasticsearch.host=localhost
23 elasticsearch.port=9300
24 elasticsearch.cluster=profile
25 elasticsearch.node=vprofilenode
26
27
28 spring.servlet.multipart.max-file-size=128KB
29 spring.servlet.multipart.max-request-size=128KB
30
31 logging.level.org.springframework.security=DEBUG
32
33
34 # application.properties
35 spring.security.user.name=admin_vp
36 spring.security.user.password=admin_vp
37 spring.security.user.roles=ADMIN
38
39
40 spring.mvc.view.prefix=/WEB-INF/views/
41 spring.mvc.view.suffix=.jsp
42
43 #logging.level.root=OFF
44 #logging.level.org.springframework.web=OFF
45 #spring.main.banner-mode=OFF

```

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.

Create security group Info

A security group acts as a virtual firewall for your instance to control inbound and outbound traffic. To create a new security group, complete the fields below.

Basic details

Security group name Info
vprofile-ELB-SG
Name cannot be edited after creation.

Description Info
Security Group for the Load Balancer (ELB SG)

VPC Info
vpc-01005303ef99ae57b

Inbound rules Info

| Type | Protocol | Port range | Source | Description - optional |
|-------|----------|------------|---|---|
| HTTP | TCP | 80 | Anyw... <input type="button" value="Delete"/> | 0.0.0.0/0 <input type="button" value="Delete"/> |
| HTTP | TCP | 80 | Anyw... <input type="button" value="Delete"/> | ::/0 <input type="button" value="Delete"/> |
| HTTPS | TCP | 443 | Anyw... <input type="button" value="Delete"/> | 0.0.0.0/0 <input type="button" value="Delete"/> |
| HTTPS | TCP | 443 | Anyw... <input type="button" value="Delete"/> | ::/0 <input type="button" value="Delete"/> |

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.

A security group acts as a virtual firewall for your instance to control inbound and outbound traffic. To create a new security group, complete the fields below.

Basic details

Security group name [Info](#)
vprofile-app-sg
Name cannot be edited after creation.

Description [Info](#)
Security Group for the Application (Tomcat SG)

VPC Info
vpc-01005303ef99ae57b

Inbound rules [Info](#)

| Type | Protocol | Port range | Source | Description - optional |
|------------|----------|------------|--------|---|
| Custom TCP | TCP | 8080 | Custom | Allow traffic from vprofile load balancer sg-0a236098591d25264 |
| SSH | TCP | 22 | My IP | Allow traffic from vprofile load balancer 49.47.128.201/32 |

Add rule

Outbound rules [Info](#)

| Type | Protocol | Port range | Destination | Description - optional |
|-------------|----------|------------|-------------|------------------------|
| All traffic | All | All | Custom | 0.0.0.0/0 |

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.

The screenshot shows the 'Create security group' wizard in the AWS Management Console. The 'Basic details' section includes a 'Security group name' field with 'vprofile-backend-sg' and a 'Description' field with 'Security group for mysql, memcache & rabbitmq allowed from tomcat'. A red arrow points to the 'Description' field. The 'VPC info' section shows 'vpc-01005303ef99ae57b'. The 'Inbound rules' section lists five rules:

| Type | Protocol | Port range | Source | Description - optional |
|--------------|----------|------------|--------|-------------------------------|
| MySQL/Aurora | TCP | 3306 | Custom | sg-0b1e25b0e8a1a2d (Delete) |
| Custom TCP | TCP | 11211 | Custom | sg-0b1e25b0e8a1a2d63 (Delete) |
| Custom TCP | TCP | 5672 | Custom | sg-0b1e25b0e8a1a2d63 (Delete) |
| SSH | TCP | 22 | My IP | 49.47.128.201/32 (Delete) |

Inbound rules

Add rule

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.

The screenshot shows the AWS EC2 Key Pairs page. The left sidebar has a 'Instances' section expanded, showing options like Instances, Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, and Capacity Reservations. The main content area is titled 'Key pairs (1/1) Info' and shows a table with one row. The table columns are Name, Type, Created, Fingerprint, and ID. The single row contains 'vprofile-prod-key', 'rsa', '2025/12/07 18:17 GMT+5:30', '44:fe:f6:cc:01:e9:ad:cf:16:cb:a0:40...', and 'key-0106a4d9991de0fbf'. There are 'Actions' and 'Create key pair' buttons at the top right.

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.

```

EXPLORER application.properties M backend.sh vprofile-prod-key.pem
334
VPROFILE-PROJECT ...
> ansible
  > src
    > main
      > test /java/com/visualpathit...
      > target
        > userdata
          > application.properties
          $ backend.sh
          $ memcache.sh
          $ mysql.sh
          $ nginx.sh
          $ rabbitmq.sh
          $ tomcat_ubuntu.sh
alzuzamrpepo
Jenkinsfile
pom.xml
README.md
...
aws
...
> OUTLINE
> TIMELINE
> APPLICATION BUILDER
> JAVA PROJECTS
> MAVEN
awsfliftshift* Java: Ready
Ln 1, Col 1 Spaces: 4 UTF-8 LF () Shell Script

```

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.

The screenshot shows the AWS EC2 Instances page. On the left, there's a navigation sidebar with sections like EC2 Global View, Events, Instances (selected), Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations, and Capacity Manager. Below that are sections for Images, AMIs, AMI Catalog, Elastic Block Store, Volumes, Snapshots, Lifecycle Manager, Network & Security, Security Groups, Elastic IPs, Placement Groups, Key Pairs, Network Interfaces, and Load Balancing. At the bottom of the sidebar are CloudShell, Feedback, and Console Mobile App links.

The main content area shows a table titled "Instances (1/1) Info". It lists one instance: "vprofile-db01" (Instance ID: i-0e2c082e95d07cbe9). The instance is "Running" (Status check: Initializing) and has an "t2.micro" instance type. It is located in the "us-east-1c" availability zone. There are buttons for "Connect", "Actions", and "Launch instances".

Below the table, a detailed view for the instance "i-0e2c082e95d07cbe9 (vprofile-db01)" is shown. The "Details" tab is selected. The instance summary includes:

- Instance ID:** i-0e2c082e95d07cbe9
- IPv6 address:** -
- Hostname type:** IP name: ip-172-31-29-59.ec2.internal
- Answer private resource DNS name:** IPv4 (A)
- Auto-assigned IP address:** 98.91.2.78 [Public IP]
- Public IPv4 address:** 98.91.2.78 [open address]
- Instance state:** Running
- Private IP DNS name (IPv4 only):** ip-172-31-29-59.ec2.internal
- Instance type:** t2.micro
- VPC ID:** vpc-01005304ef99ae57h
- Public DNS:** ec2-98-91-2-78.compute-1.amazonaws.com [open address]
- Private IPv4 addresses:** 172.31.29.59
- Elastic IP addresses:** -
- AWS Compute Optimizer finding:** Opt-in to AWS Compute Optimizer for recommendations.

At the bottom of the page, there are links for CloudShell, Feedback, and Console Mobile App, along with copyright information: © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences.

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.

The screenshot shows the AWS EC2 Instances page. On the left, there's a sidebar with various navigation links: EC2 Global View, Events, Instances (selected), Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations, Capacity Manager, Images, AMIs, AMI Catalog, Elastic Block Store, Volumes, Snapshots, Lifecycle Manager, Network & Security, Security Groups, Elastic IPs, Placement Groups, Key Pairs, Network Interfaces, and Load Balancing. The main content area displays two instances: vprofile-mc01 (running, t2.micro, public IP 54.159.30.190) and vprofile-db01 (running, t2.micro, public IP 54.159.30.190). Below the instances, there's a detailed view for instance i-042de202838571397 (vprofile-mc01), showing its Public IPv4 address (54.159.30.190), Private IP DNS name (ip-172-31-31-171.ec2.internal), Instance type (t2.micro), and other configuration details like VPC ID and Subnet ID.

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.

The screenshot shows the AWS EC2 Instances page. On the left, there's a sidebar with navigation links for Global View, Events, Instances (selected), Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations, and Capacity Manager. Below that are sections for Images, Elastic Block Store, Network & Security, and Load Balancing. The main content area displays a table of instances. The first instance, `vprofile-rmq01`, is highlighted with a blue border. Its details are expanded in a modal window. The modal shows the following information for `i-015bbe4dcf9032632 (vprofile-rmq01)`:

| Details | | Status and alarms | Monitoring | Security | Networking | Storage | Tags |
|----------------------------------|---|---------------------------------|--|-----------------------|--|-------------------------------|--|
| Instance summary | Info | | | | | | |
| Instance ID | i-015bbe4dcf9032632 | Public IP4 address | 18.212.95.253 open address | Private IP4 addresses | 172.31.23.208 | | |
| IPv6 address | - | Instance state | Running | Public DNS | ec2-18-212-95-253.compute-1.amazonaws.com open address | | |
| Hostname type | IP name: ip-172-31-23-208.ec2.internal | Private IP DNS name (IPv4 only) | ip-172-31-23-208.ec2.internal | Instance type | t2.micro | Elastic IP addresses | - |
| Answer private resource DNS name | IPv4 (A) | | | VPC ID | vpc-01005303ef99ae57b | AWS Compute Optimizer finding | Opt-in to AWS Compute Optimizer for recommendations. |
| Auto-assigned IP address | 18.212.95.253 [Public IP] | | | Subnet ID | subnet-09317da9d93811654 | Learn more | |
| IAM Role | - | | | | | Auto Scaling Group name | - |

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.

The screenshot shows the AWS EC2 Instances page. On the left, there's a navigation sidebar with sections like EC2 Global View, Events, Instances (selected), Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations, Capacity Manager (New), Images (AMIs, AMI Catalog), Elastic Block Store (Volumes, Snapshots, Lifecycle Manager), Network & Security (Security Groups, Elastic IPs, Placement Groups, Key Pairs, Network Interfaces), and Load Balancing (Load Balancers). The main area displays a table of instances. The table has columns for Name, Instance ID, Instance state, Instance type, Status check, Alarm status, and Availability Zone. There are four instances listed: vprofile-rmq01, vprofile-mc01, vprofile-db01, and vprofile-app01. The vprofile-app01 instance is selected, indicated by a checked checkbox in the first column. Below the table, there's a detailed view for the selected instance (vprofile-app01). This view includes tabs for Details, Status and alarms, Monitoring, Security, Networking, Storage, and Tags. Under the Details tab, there's a section for Instance summary with fields like Instance ID (i-0eab7cc96578452ea), Public IPv4 address (54.197.205.182), Private IP4 address (172.31.29.69), Public DNS (ec2-54-197-205-182.compute-1.amazonaws.com), and Private IP DNS name (ip-172-31-29-69.ec2.internal). Other sections show Auto-assigned IP address (54.197.205.182 [Public IP]), IAM Role (none), and various AWS service links like VPC ID, Subnet ID, and Auto Scaling Group name.

✓ 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 SHA256:8DQ5x58LiFOJNY1EbRxYTAC/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:mysqld(8)
           https://mariadb.com/kb/en/library/systemd/
   Main PID: 27423 (mariadb)
     Status: "Taking your SQL requests now..."
       Tasks: 8 (limit: 1106)
      Memory: 67.4M
        CPU: 492ms
      CGroup: /system.slice/mariadb.service
              └─27423 /usr/libexec/mariadb --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:Ghi6/+OnrhmhQT/B4Alw50mlxk0qWZ15DsMRLe3CWg.
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
  ~\`-' '->
  ~~~  /
  ~..- _/
  /_`_/
  /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: 106)
       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:X8K2hATsfeJ7U70sgb6GIH0H1hTpCiwNlkUhJlo.
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
  ~\`-' '->
  ~~~  /
  ~..- _/
  /_`_/
  /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; 1min ago
     Main PID: 26400 (beam.smp)
        Tasks: 23 (limit: 106)
       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 -MKos ageffcbf -MBmbcs 512 -MMmc 30 -pc unicode -P 1048576 -t 5000000 -stbt db -zdbbl 128000 -sbwt none -s
             ├─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)**

```

Invaliid titles 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-172-31-29-69:~$ sudo -i
root@ip-172-31-29-69:~# systemctl status tomcat10
● tomcat10.service - Apache Tomcat 10 Web Application Server
   Loaded: loaded (/usr/lib/systemd/system/tomcat10.service; enabled; preset: enabled)
   Active: active (running) since Mon 2025-12-08 11:38:01 UTC; 1min ago
     Docs: https://tomcat.apache.org/tomcat-10.0-doc/index.html
Main PID: 13269 (java)
   Tasks: 1 (limit: 1121)
      Memory: 101.2MiB (peak: 109.2MiB)
         CPU: 8.246s
      CGroup: /system.slice/tomcat10.service
             └─13269 /usr/lib/jvm/java-17-openjdk-amd64/bin/java -Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager -Dj

Dec 08 11:38:08 ip-172-31-29-69 tomcat10[13269]: Deployment of deployment descriptor [/etc/tomcat10/Catalina/localhost/docs.xml] has finished in [1,109] ms
Dec 08 11:38:08 ip-172-31-29-69 tomcat10[13269]: Deploying deployment descriptor [/etc/tomcat10/Catalina/localhost/manager.xml]
Dec 08 11:38:08 ip-172-31-29-69 tomcat10[13269]: The path attribute with value [/manager] in deployment descriptor [/etc/tomcat10/Catalina/localhost/manager.xml] has been ignored
Dec 08 11:38:09 ip-172-31-29-69 tomcat10[13269]: At least one JAR was scanned for TLDs yet contained no TLDs. Enable debug logging for this logger for a complete list of JARs that were scanned but no TLDs were found in it.
Dec 08 11:38:09 ip-172-31-29-69 tomcat10[13269]: Deployment of deployment descriptor [/etc/tomcat10/Catalina/localhost/manager.xml] has finished in [1,157] ms
Dec 08 11:38:09 ip-172-31-29-69 tomcat10[13269]: Deploying web application directory [/var/lib/tomcat10/webapps/ROOT]
Dec 08 11:38:10 ip-172-31-29-69 tomcat10[13269]: At least one JAR was scanned for TLDs yet contained no TLDs. Enable debug logging for this logger for a complete list of JARs that were scanned but no TLDs were found in it.
Dec 08 11:38:10 ip-172-31-29-69 tomcat10[13269]: Deployment of web application directory [/var/lib/tomcat10/webapps/ROOT] has finished in [1,025] ms
Dec 08 11:38:10 ip-172-31-29-69 tomcat10[13269]: Starting ProtocolHandler ["http-nio-8080"]
Dec 08 11:38:10 ip-172-31-29-69 tomcat10[13269]: Server startup in [6333] milliseconds
lines 1-22/22 (END)

```

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

The screenshot shows the AWS Route 53 interface. On the left, there's a navigation sidebar with sections like Route 53, Hosted zones, Global Resolver, VPC Resolver, Domains, IP-based routing, and Traffic flow. The main area is titled 'vprofile.in' and shows 'Hosted zone details'. Under 'Records (5)', there's a table with columns: Record name, Type, Value/Route traffic to, TTL, and Health. The table includes rows for vprofile.in (NS), db01.vprofile.in (A, highlighted with a red box), mc01.vprofile.in (A), and rmq01.vprofile.in (A).

| Record name | Type | Value/Route traffic to | TTL | Health |
|-------------------|------|--|--------|--------|
| vprofile.in | NS | ns-1536.awsdns-00.co.uk. ns-0.awsdns-00.com. ns-1024.awsdns-00.org. ns-512.awsdns-00.net. | 172800 | - |
| db01.vprofile.in | A | 172.31.29.59 | 300 | - |
| mc01.vprofile.in | A | 172.31.31.171 | 300 | - |
| rmq01.vprofile.in | A | 172.31.23.208 | 300 | - |

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.

```
khushboo@Mac downloads % ssh -i "vprofile-prod-key.pem" ubuntu@ec2-54-197-205-182.compute-1.amazonaws.com
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 6.14.0-1015-aws x86_64)

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

System information as of Mon Dec  8 15:55:36 UTC 2025

System load:  0.0          Processes:      109
Usage of /:   42.2% of 6.71GB  Users logged in:  0
Memory usage: 36%           IPv4 address for enX0: 172.31.29.69
Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

5 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

*** System restart required ***
Last login: Mon Dec  8 11:49:01 2025 from 49.128.201
ubuntu@ip-172-31-29-69:~$ ping -c 4 db01.vprofile.in
PING db01.vprofile.in (172.31.29.59) 56(84) bytes of data.
64 bytes from ip-172-31-29-59.ec2.internal (172.31.29.59): icmp_seq=1 ttl=127 time=1.67 ms
64 bytes from ip-172-31-29-59.ec2.internal (172.31.29.59): icmp_seq=2 ttl=127 time=0.737 ms
64 bytes from ip-172-31-29-59.ec2.internal (172.31.29.59): icmp_seq=3 ttl=127 time=0.478 ms
64 bytes from ip-172-31-29-59.ec2.internal (172.31.29.59): icmp_seq=4 ttl=127 time=1.72 ms

--- db01.vprofile.in ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3027ms
rtt min/avg/max/mdev = 0.478/1.151/1.717/0.551 ms
ubuntu@ip-172-31-29-69:~$
```

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.

The screenshot shows the AWS S3 Buckets page. At the top, there's a success message: "Successfully created bucket 'vprofile-lab-artifacts-1234'". Below that, the General purpose buckets section lists one bucket named "vprofile-lab-artifacts-1234" in the US East (N. Virginia) region, created on December 8, 2025, at 21:30:23 (UTC+05:30). There are buttons for "Copy ARN", "Empty", "Delete", and "Create bucket". To the right, there are two informational boxes: "Account snapshot" (updated daily) and "External access summary - new" (updated daily).

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

The screenshot shows the AWS IAM User Details page for the user 'vprofile-s3-admin'. The 'Permissions' tab is active. In the 'Permissions policies' section, there is one policy listed: 'AmazonS3FullAccess'. In the 'Permissions boundary (not set)' section, there is a note about generating a policy based on CloudTrail events.

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

The screenshot shows the AWS IAM Roles page. On the left, there's a sidebar with navigation links like Identity and Access Management (IAM), Dashboard, Access management, Roles, Access reports, and IAM Identity Center. The main area displays the details for the 's3-admin' role. The role has an ARN of arn:aws:iam::495395223996:role/s3-admin and was created on December 07, 2025, at 23:05 UTC+05:30. It has a maximum session duration of 1 hour. The 'Permissions' tab is selected, showing one managed policy attached: 'AmazonS3FullAccess'. There are tabs for Trust relationships, Tags, Last Accessed, and Revoke sessions. Below the permissions section, there are sections for Permissions boundary (not set) and Generate policy based on CloudTrail events.

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.

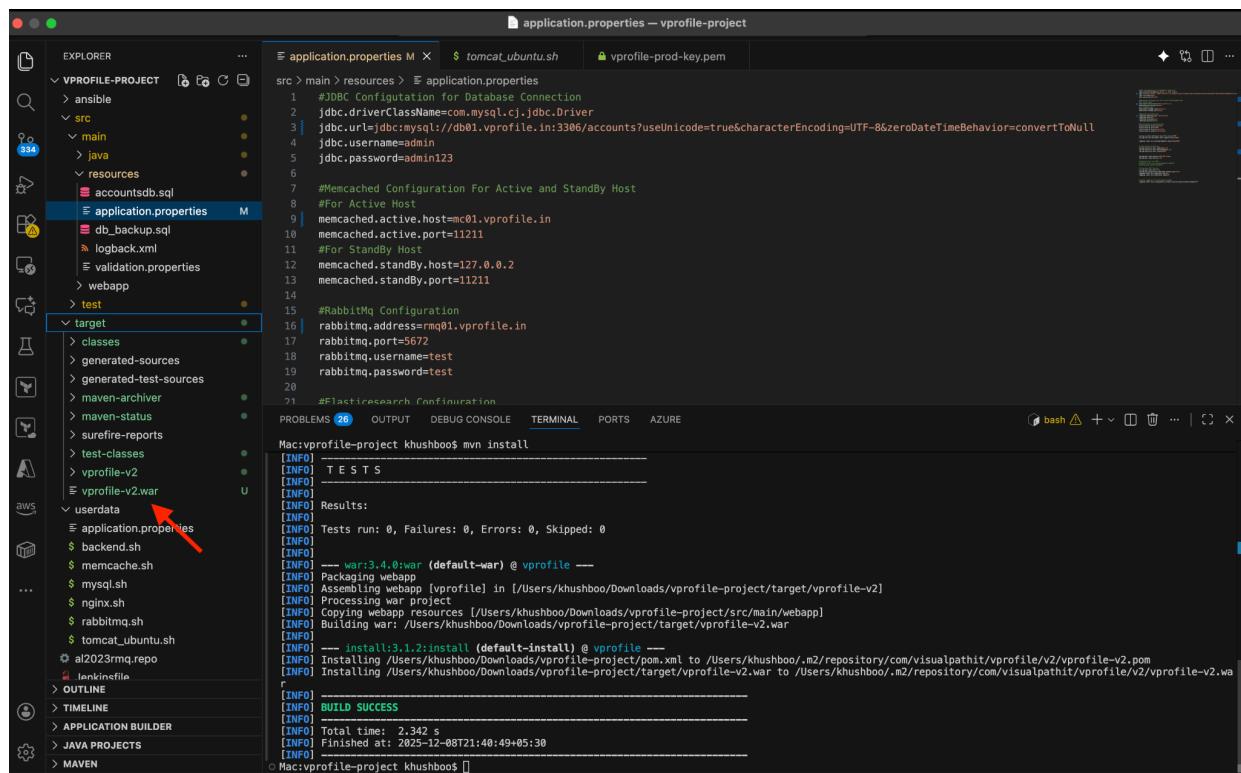
Step 4: Build the Artifact Using Maven

Navigate to the project folder and run:

`mvn install`

After successful build, you will see:

`target/vprofile-v2.war`



The screenshot shows a code editor interface with several tabs open. The left sidebar shows a project structure with a red arrow pointing to the 'userdata' folder. The main editor area contains an 'application.properties' file with configuration for database connections, memcached hosts, and RabbitMQ. Below the editor is a terminal window showing the command 'mvn install' being run, followed by the Maven build output. The output shows the creation of a 'vprofile-v2.war' file in the 'target' directory and its successful installation into the local Maven repository.

```
Mac:vprofile-project khushboo$ mvn install
[INFO] ------------------------------------------------------------------------
[INFO] T E S T S
[INFO] ------------------------------------------------------------------------
[INFO] Results:
[INFO]
[INFO] Tests run: 0, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] --- var3.4.0:war (default-war) @ vprofile ---
[INFO] Packaging webapp [vprofile] in [/Users/khushboo/Downloads/vprofile-project/target/vprofile-v2]
[INFO] Processing war project
[INFO] Assembling webapp [vprofile] in [/Users/khushboo/Downloads/vprofile-project/src/main/webapp]
[INFO] Copying webapp resources [/Users/khushboo/Downloads/vprofile-project/src/main/webapp]
[INFO] Building war: /Users/khushboo/Downloads/vprofile-project/target/vprofile-v2.war
[INFO]
[INFO] --- install3.1.2:install (default-install) @ vprofile ---
[INFO] Installing /Users/khushboo/Downloads/vprofile-project/pom.xml to /Users/khushboo/.m2/repository/com/visualpathit/vprofile/v2/vprofile-v2.pom
[INFO] Installing /Users/khushboo/Downloads/vprofile-project/target/vprofile-v2.war to /Users/khushboo/.m2/repository/com/visualpathit/vprofile/v2/vprofile-v2.war
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2,342 s
[INFO] Finished at: 2025-12-08T21:40:49+05:30
[INFO] ------------------------------------------------------------------------
Mac:vprofile-project khushboo$
```

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/
```

The screenshot shows a terminal window within a code editor interface. The terminal is executing the command:

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

An arrow points from the bottom left towards the command line, indicating where the user should type their bucket name.

Check upload:

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

Step 6: Download Artifact on Tomcat Server

SSH into the Tomcat instance:

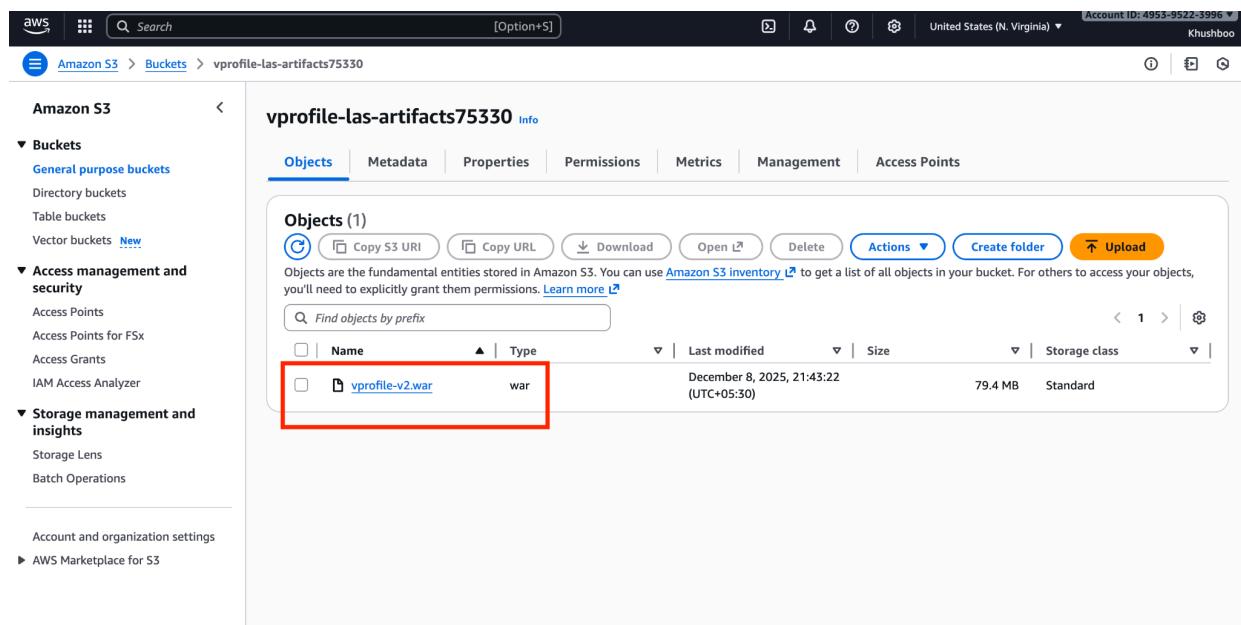
```
ssh -i ~/Downloads/vprofile-prod-key.pem ubuntu@<Public-IP>
```

Install AWS CLI (Ubuntu):

```
sudo snap install aws-cli --classic
```

Download the artifact:

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



The screenshot shows the AWS S3 console interface. On the left, there's a navigation sidebar with sections like 'Amazon S3', 'Buckets', 'Access management and security', and 'Storage management and insights'. The main area shows the contents of the 'vprofile-las-artifacts75330' bucket. The 'Objects' tab is active, showing a single item: 'vprofile-v2.war'. This file is a WAR file, last modified on December 8, 2025, at 21:43:22 (UTC+05:30), and is 79.4 MB in size. The file name is highlighted with a red box.

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:~# systemctl stop tomcat10
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**

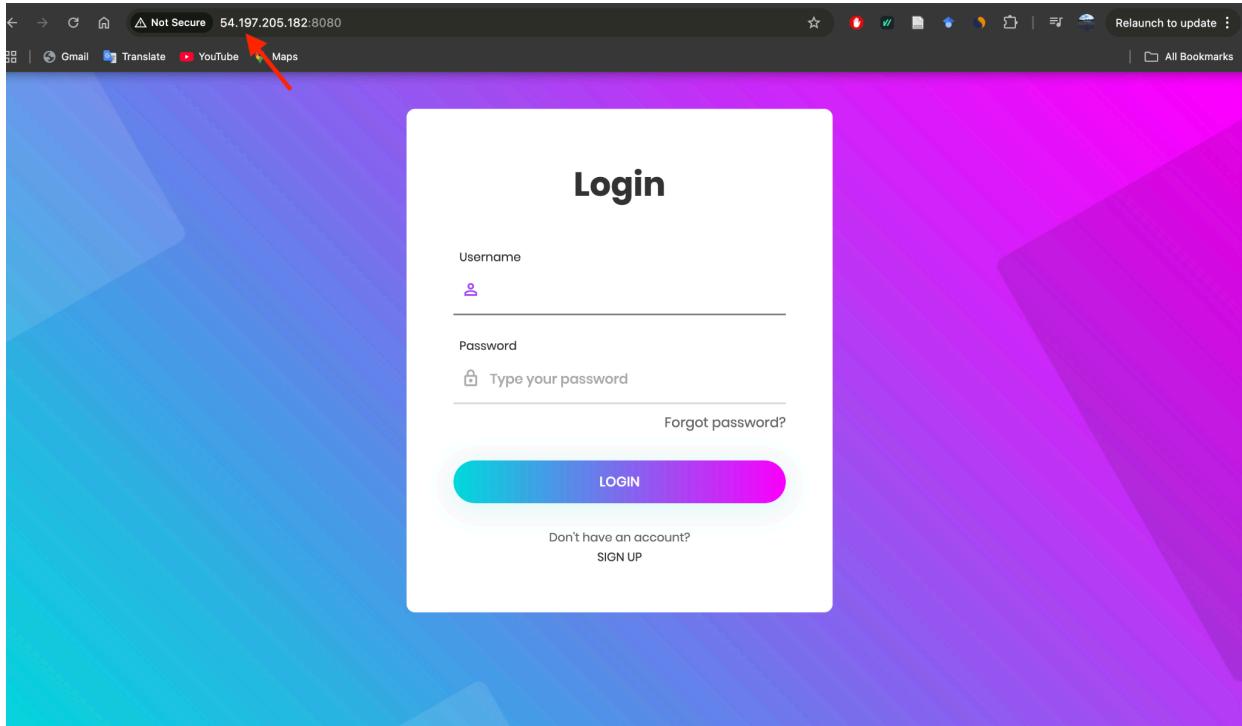
This is temporary, only for verification.

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.

Review and create
Review your target group configuration before creating

Step 1: Target group details

| | | | |
|--|-------------------------|-------------------------------|---------------------------|
| Name vprofile-lsa-TG | Target type Instance | Protocol : Port HTTP: 8080 | Protocol version HTTP1 |
| VPC vpc-01005303ef99ae57b | IP address type IPv4 | | |

Health check details

| | | | |
|-------------------------------|------------------------|---------------------------|------------------------|
| Health check protocol HTTP | Health check path / | Health check port 8080 | Interval 30 seconds |
| Timeout 5 seconds | Healthy threshold 5 | Unhealthy threshold 2 | Success codes 200 |

Step 2: Register targets

| Targets (1) | | | |
|-------------------------------------|----------------|------|------------|
| Instance ID | Name | Port | Zone |
| i-0eab7cc96578452ea | vprofile-app01 | 8080 | us-east-1c |

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.

The screenshot shows the AWS EC2 Load Balancers console with the following details:

- Load balancer type:** Application
- Scheme:** Internet-facing
- Status:** Active
- VPC:** vpc-01005303ef99ae57b
- Hosted zone:** Z35SXDOTRQ7X7K
- Availability Zones:**
 - subnet-074fa6ba74a7d9e9 (us-east-1d)
 - subnet-09317da9d93811654 (us-east-1c)
 - subnet-0619ef68aab67c43e (us-east-1a)
 - subnet-08188346c7640d202 (us-east-1f)
 - subnet-07e36632d08c6a346 (us-east-1e)
 - subnet-0f5598876e1ce1044 (us-east-1b)
- Load balancer IP address type:** IPv4
- Date created:** December 8, 2025, 22:39 (UTC+05:30)
- Listeners and rules:** The tab is currently selected.
- Network mapping:** Available but not selected.
- Resource map:** Available but not selected.
- Security:** Available but not selected.
- Monitoring:** Available but not selected.
- Integrations:** Available but not selected.
- Attributes:** Available but not selected.
- Capacity:** Available but not selected.
- Tags:** Available but not selected.

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

vprofile-lsa-TG

Details

| | | | |
|-------------------------|--|--|--|
| Target type Instance | Protocol : Port HTTP: 8080 | Protocol version HTTP1 | VPC vpc-01005303ef99ae57b |
| IP address type IPv4 | Load balancer None associated | | |
| 1 Total targets | 1 Healthy | 0 Unhealthy | 0 Unused |
| | 0 Anomalous | | 0 Initial |
| | | | 0 Draining |

Distribution of targets by Availability Zone (AZ)
Select values in this table to see corresponding filters applied to the Registered targets table below.

Targets | Monitoring | Health checks | Attributes | Tags

Registered targets (1) [Info](#) (Anomaly mitigation: Not applicable) [C](#) [Deregister](#) [Register targets](#)

Target groups route requests to individual registered targets using the protocol and port number specified. Health checks are performed on all registered targets according to the target group's health check settings. Anomaly detection is automatically applied to HTTP/HTTPS target groups with at least 3 healthy targets.

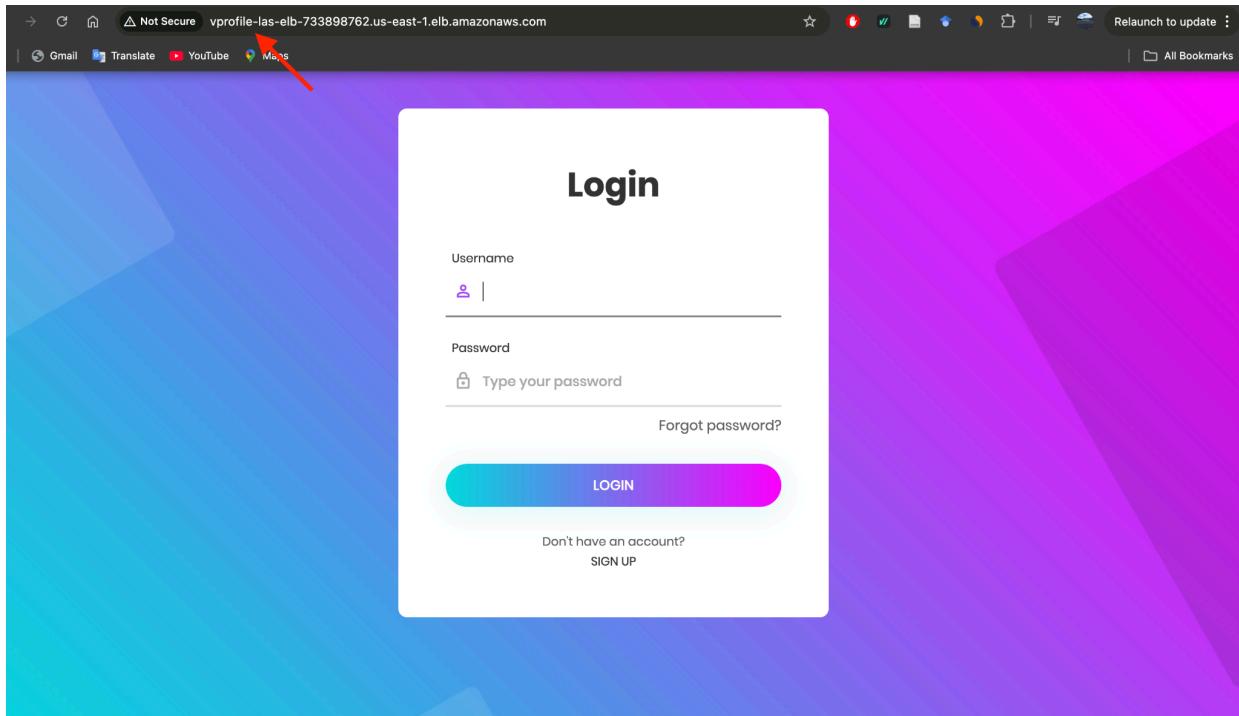
| <input type="checkbox"/> Instance ID | Name | Port | Zone | Health status | Health status details | Action |
|--------------------------------------|----------------|------|---------------------|--|-----------------------|----------------------|
| i-0eab7cc96578452ea | vprofile-app01 | 8080 | us-east-1c (use...) | Healthy | - | Edit |

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

Bio
DevOps For Product Management and Strategy of Application Delivery at HKH Infotech. Responsible of providing customers with counsel on their DevOps strategies to help them deliver higher quality software and services to market faster.

Location
Earth

Gender
Unknown

"The Key to DevOps Success." The Key to DevOps Success" Collaboration". Collaboration is essential to DevOps, yet how to do it is often unclear with many teams falling back on ineffective conference calls, instant messaging, documents, and SharePoint sites. In this keynote, we will share a vision for a next generation DevOps where collaboration, continuous documentation, and knowledge capture are combined with automation toolchains to enable rapid innovation and deployment.

- RabbitMQ connection → Queue appears

RabbitMQ Initiated

Generated 6 Connections

6 Channels, 7 Exchange, and 9 Queues

- Memcache connection

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

| User Name | User Id |
|---------------|---------|
| Hibo Prince | 4 |
| Aejaz Habeeb | 5 |
| Jackie | 6 |
| admin_vp | 7 |
| Abrar Nirban | 8 |
| Amayra Fatima | 9 |
| Aron | 10 |
| Kiran Kumar | 11 |
| Balbir Singh | 12 |

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**

Screenshot of the AWS EC2 Instances page showing a list of running instances. An arrow points to the 'Actions' dropdown menu for the selected instance 'vprofile-app01'. The dropdown menu includes options like 'Create image', 'Create template from instance', and 'Launch more like this'.

Screenshot of the AWS EC2 AMIs page showing a list of Amazon Machine Images (AMIs). An arrow points to the 'Actions' dropdown menu for the selected AMI 'vprofile-las-app-ami'. The dropdown menu includes options like 'Recycle Bin', 'EC2 Image Builder', and 'Launch instance from AMI'.

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

The screenshot shows the AWS EC2 console with the 'Launch Templates' section selected. The left sidebar includes links for Dashboard, EC2 Global View, Events, Instances (selected), Images, Elastic Block Store, and Network & Security. The main content area displays the 'Launch Templates (1/1)' page. A table lists one launch template:

| Launch Template ID | Launch Template Name | Default Version | Latest Version | Create Time | Created By |
|----------------------|----------------------|-----------------|----------------|--------------------------|--------------------------------|
| lt-086d95a7934558330 | vprofile-las-APP-LT | 1 | 1 | 2025-12-08T17:36:16.000Z | arn:aws:iam::495395223996:root |

Below the table, the details for the launch template 'vprofile-las-APP-LT' are shown. The 'Launch template details' section includes fields for Launch template ID (lt-086d95a7934558330), Launch template name (vprofile-las-APP-LT), Default version (1), and Owner (arn:aws:iam::495395223996:root). The 'Launch template version details' section shows a single version (1 Default) with a description of 'vprofile-las-APP-LT', created on 2025-12-08T17:36:16.000Z by the same owner.

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**

The screenshot shows the AWS EC2 Auto Scaling Groups page. On the left, there's a sidebar with navigation links like Reserved Instances, Dedicated Hosts, Capacity Reservations, Capacity Manager, Images (AMIs, AMI Catalog), Elastic Block Store (Volumes, Snapshots, Lifecycle Manager), Network & Security (Security Groups, Elastic IPs, Placement Groups, Key Pairs, Network Interfaces), Load Balancing (Load Balancers, Target Groups, Trust Stores), and Auto Scaling (Auto Scaling Groups). Below these are Settings and Tags.

The main content area is titled "vprofile-las-asg" and contains a "vprofile-las-asg Capacity overview" section. It shows Desired capacity: 1, Scaling limits (Min - Max): 1 - 2, Desired capacity type: Units (number of instances), and Status: -. There's also a Date created: Mon Dec 08 2025 23:12:05 GMT+0530 (India Standard Time).

Below this is a "Launch template" section. It lists the Launch template: lt-086d95a7934558330 (vprofile-las-APP-LT), AMI ID: ami-026efc6744b8d7b84, Instance type: t2.micro, Owner: arn:aws:iam::495395223996:root, Version: Default, Security groups: -, Security group IDs: sg-0b1e25b0e8a1a2d63, Create time: Mon Dec 08 2025 23:06:16 GMT+0530 (India Standard Time), Description: vprofile-las-APP-LT, Storage (volumes): -, Key pair name: vprofile-prod-key, and Request Spot Instances: No.

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

| Name | ARN | Port | Protocol | Target type | Load balancer |
|-----------------|--|------|----------|-------------|------------------|
| vprofile-lsa-TG | arn:aws:elasticloadbalancing:us-east-1:123456789012:targetgroup/vprofile-lsa-TG/1234567890123456 | 8080 | HTTP | Instance | vprofile-las-elb |

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 screenshot shows the AWS CloudWatch Metrics Groups page. A target group named "vprofile-lsa-TG" is selected. The "Targets" tab is active, displaying two registered targets:

| Instance ID | Port | Zone | Health Status |
|---------------------|------|-----------------------|---------------|
| i-081c52d38fc0957 | 8080 | us-east-1a (use1-az1) | Healthy |
| i-0eab7cc96578452ea | 8080 | us-east-1c (use1-az4) | Healthy |

The screenshot shows the AWS EC2 Instances page. A success message "Successfully initiated termination (deletion) of i-0eab7cc96578452ea" is displayed. The main table lists five instances:

| Name | Instance ID | Instance State | Status Check | Alarm Status | Availability Zone |
|----------------|---------------------|----------------|-------------------|---------------|-------------------|
| vprofile-app | i-081c52d38fc0957 | Running | 2/2 checks passed | View alarms + | us-east-1a |
| vprofile-rmq01 | i-015bbe4dcf9032632 | Running | 2/2 checks passed | View alarms + | us-east-1c |
| vprofile-mc01 | i-042de202838571397 | Running | 2/2 checks passed | View alarms + | us-east-1c |
| vprofile-db01 | i-0e2c082e95d07cbe9 | Running | 2/2 checks passed | View alarms + | us-east-1c |
| vprofile-app01 | i-0eab7cc96578452ea | Shutting-down | - | View alarms + | us-east-1c |

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**

@hkhinfotech.co.in ✓ Welcome, to Hkhinfotech Social Media!

#DevOps #Continuous Integration #Continuous Delivery #Automation All Users RabbitMq Elasticsearch

Posts Photos 42 Contacts 42

42 minutes ago

"The Key to DevOps Success." Collaboration is essential to DevOps, yet how to do it is often unclear with many teams falling back on ineffective conference calls, instant messaging, documents, and SharePoint sites. In this keynote, we will share a vision for a next generation DevOps where collaboration, continuous documentation, and knowledge capture are combined with automation toolchains to enable rapid innovation and deployment.

Public Like Reshare Comment

Comment

42 minutes ago

Ahar nirban

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.