Technical Project Report

AWS Lift and Shift: Multi-Tier Web Application Deployment

Project Overview

This report documents the lift and shift migration of the VProfile multi-tier web application onto Amazon Web Services. Lift and shift means the existing application architecture is moved to the cloud with minimal code changes. the application runs on EC2 virtual machines rather than being re architected into managed cloud services. The goal is to achieve cloud hosted scalability and availability while retaining the familiar server based deployment model.

The VProfile application is a Java based social networking platform. Its stack consists of four services: a Tomcat 10 application server that runs the WAR packaged Java application, a MySQL database that stores persistent user data, a Memcached server that caches database query results to reduce load, and a RabbitMQ message broker that handles asynchronous messaging between components. Each of these services runs on its own dedicated EC2 t3.micro instance.

On top of these EC2 instances, several AWS managed services are layered to provide production grade infrastructure: an Application Load Balancer handles incoming HTTPS traffic and distributes it to the Tomcat instance; an Auto Scaling Group ensures the application tier can scale out to handle demand; Amazon S3 stores the compiled application artifact; Route 53 provides private DNS resolution inside the VPC so services can reach each other by hostname; and AWS Certificate Manager supplies a wildcard SSL/TLS certificate that secures the public domain vprofileapp.theenuka.xyz.

1. Architecture and Technology Stack

The architecture separates concerns across three tiers. The presentation tier is the Application Load Balancer the only component exposed to the public internet. It accepts HTTPS requests on port 443 and forwards them to the application tier. The application tier is the Tomcat EC2 instance (project-app01), which processes requests, queries the database, reads from cache, and publishes messages. The data tier consists of three EC2 instances project-db01 (MySQL, port 3306), project-mc01 (Memcached, port 11211), and project-rmq01 (RabbitMQ, port 5672) none of which are reachable from the public internet.

All internal communication between the application tier and the data tier uses private IP addresses resolved through Route 53 private DNS records under the hosted zone vprofile.in. This means the application configuration refers to hostnames like db01.vprofile.in rather than hardcoded IP addresses, making the infrastructure easier to maintain and replace.

Component	Details
Application server	Apache Tomcat 10 on EC2 t3.micro - project-app01, us-east-2c
Database	MySQL on EC2 t3.micro - project-db01, port 3306, us-east-2c
Cache	Memcached on EC2 t3.micro - project-mc01 , port 11211, us-east-2c
Message broker	RabbitMQ on EC2 t3.micro - project-rmq01 , port 5672, us-east-2c
Load balancer	Application Load Balancer - vprofile-theenuka-elb (internet-facing, IPv4), 3 AZs
Target group	vprofile-theenuka-tg - protocol HTTP, port 8080
Auto Scaling Group	vprofile-theenuka-app-asg - min 1, desired 1, max 4 - 3 availability zones
Artifact store	S3 bucket - vprofile-theenu-artifacts - vprofile-v2.war
Private DNS	Route 53 private hosted zone - vprofile.in - 4 A records for backend hosts
SSL certificate	ACM wildcard certificate - *.theenuka.xyz - Amazon issued
Public domain	GoDaddy - theenuka.xyz - CNAME vprofileapp → ELB DNS endpoint
SSH key pair	project-prod-key (RSA)
VPC	vpc-default
AWS region	US East (Ohio) - us-east-2

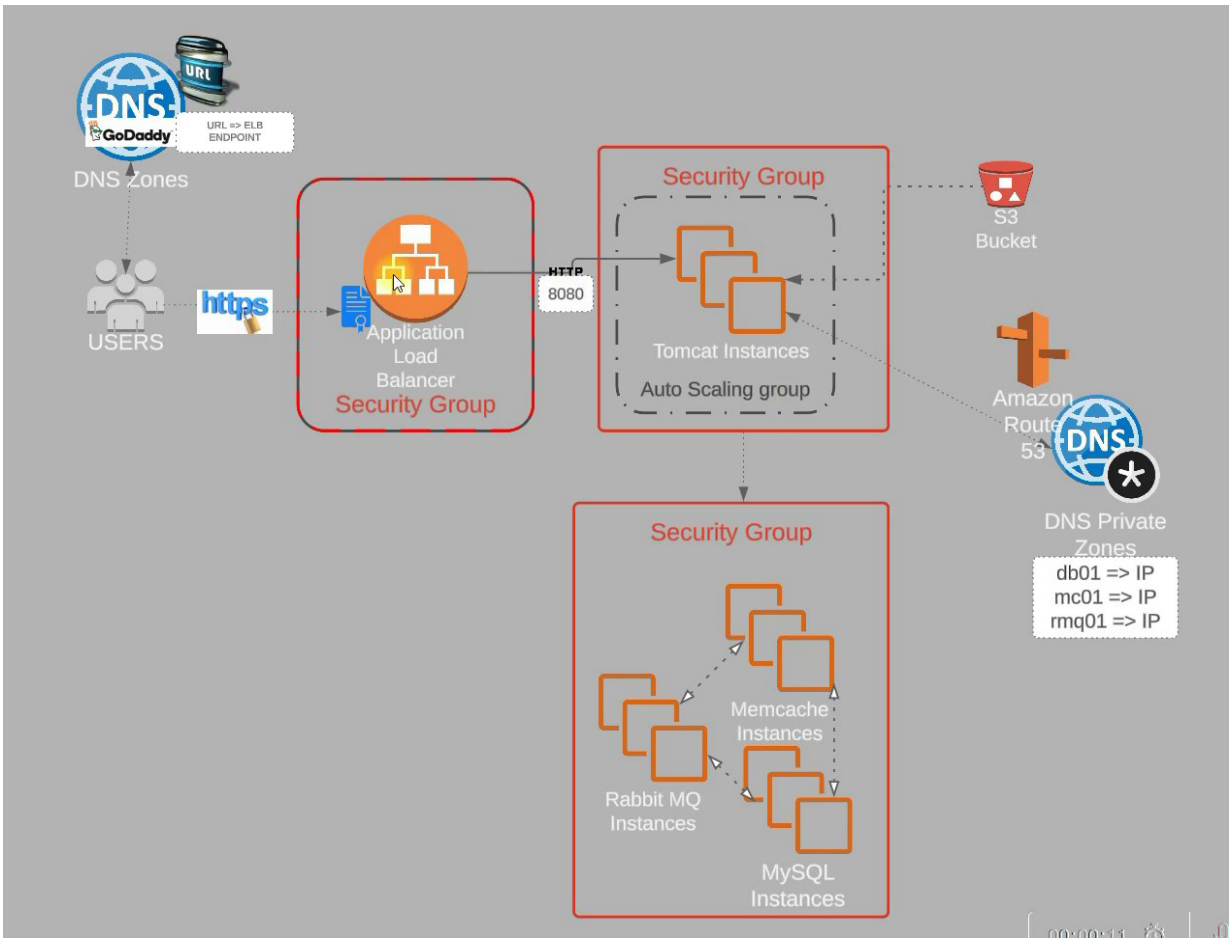

Figure 1 : Full project architecture diagram

2. Flow of Execution

The deployment was carried out in twelve sequential steps. Each step builds on the last infrastructure foundations are laid first, then instances are launched, then the application is built and deployed, and finally routing, security, and scaling are completed. A screenshot from each step is provided below.

Step 1 : Log in to AWS and Set the Region

The starting point was logging into the AWS Management Console and confirming the active region as US East (Ohio), us-east-2.

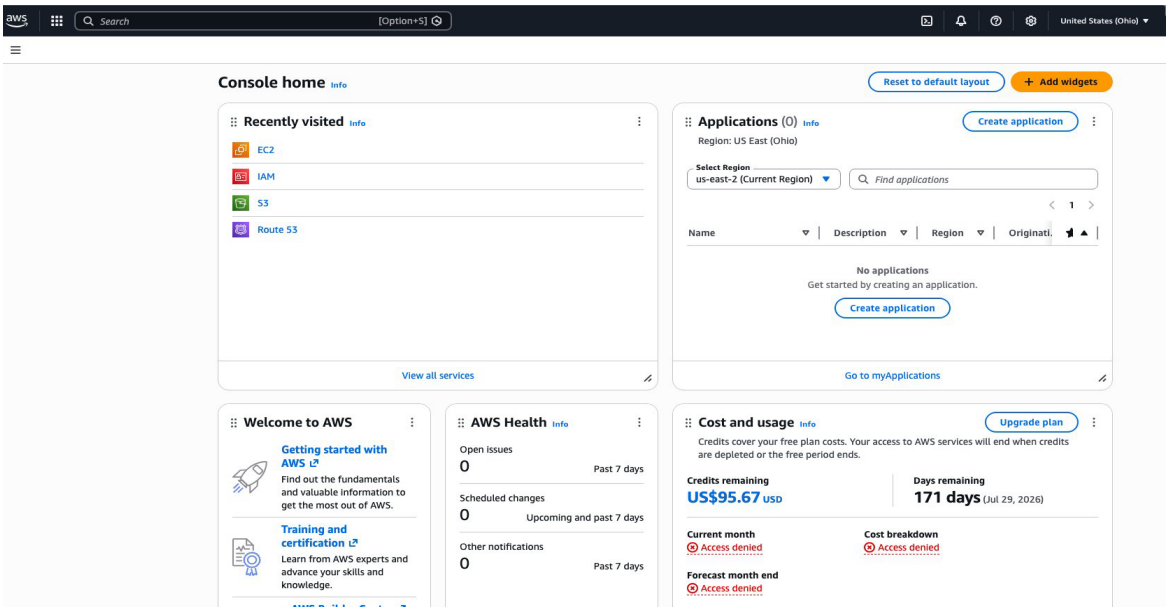

Figure 2 : AWS Console home showing recently visited services and us-east-2 as current region

Step 2 : Create a Key Pair

A key pair named project-prod-key was created in the EC2 console using the RSA algorithm. This single key is used for SSH access to all four EC2 instances throughout the project, keeping credential management straightforward during development.

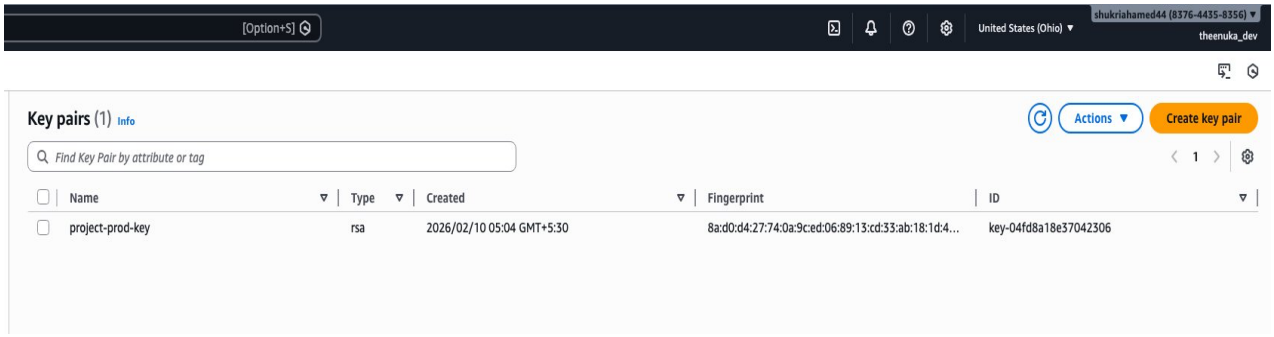

Figure 3 : EC2 Key Pairs: project-prod-key (RSA)

Step 3 : Create Security Groups

Four security groups were created inside VPC, each enforcing a specific set of access rules.

project-ELB-SG is the outermost layer with 4 inbound rules: HTTPS TCP 443 from 0.0.0.0/0, HTTPS TCP 443 from ::/0, HTTP TCP 80 from 0.0.0.0/0, and HTTP TCP 80 from ::/0. This makes the load balancer reachable from any IP on the internet over both IPv4 and IPv6.

project-app-SG has 3 inbound rules: Custom TCP port 8080 from my IP, Custom TCP port 8080 from the project-ELB-SG security group (allowing the load balancer to forward traffic), and SSH TCP 22 from my IP for administrative access. This means the Tomcat instance is not publicly accessible on port 8080 only the load balancer can reach it.

Project-backend SG has 6 inbound rules: SSH TCP 22 from my IP, All traffic from project-app-SG , Custom TCP 11211 (Memcached) from project-app-SG, MySQL/Aurora TCP 3306 from project-app-SG, Custom TCP 5672 (RabbitMQ) from project-app-SG, and All traffic from the backend SG itself (for inter-service communication). This keeps all three data-tier services completely isolated from the internet.

The default VPC security group has 1 inbound rule: All traffic from itself, which is standard AWS default VPC behaviour.

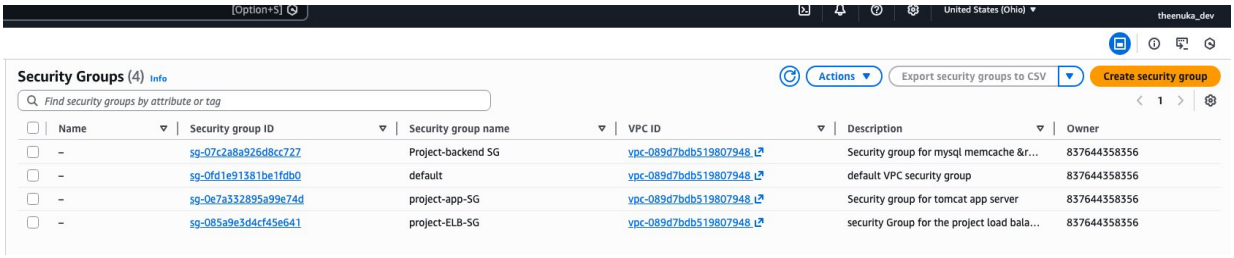

Figure 4 : Security Groups list: 4 groups in VPC vpc

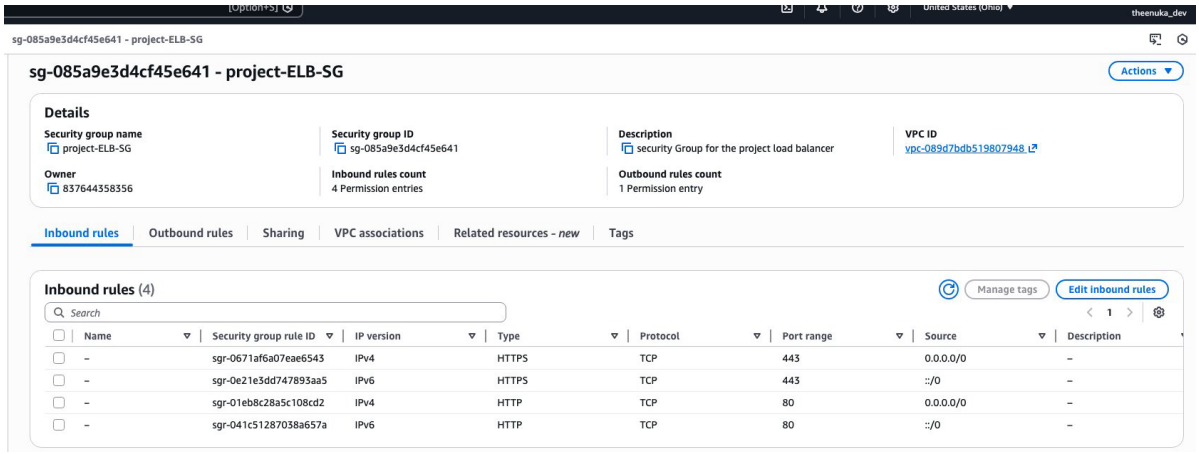

Figure 5 : project-ELB-SG: 4 inbound rules , HTTP 80 and HTTPS 443 from internet (IPv4 and IPv6)

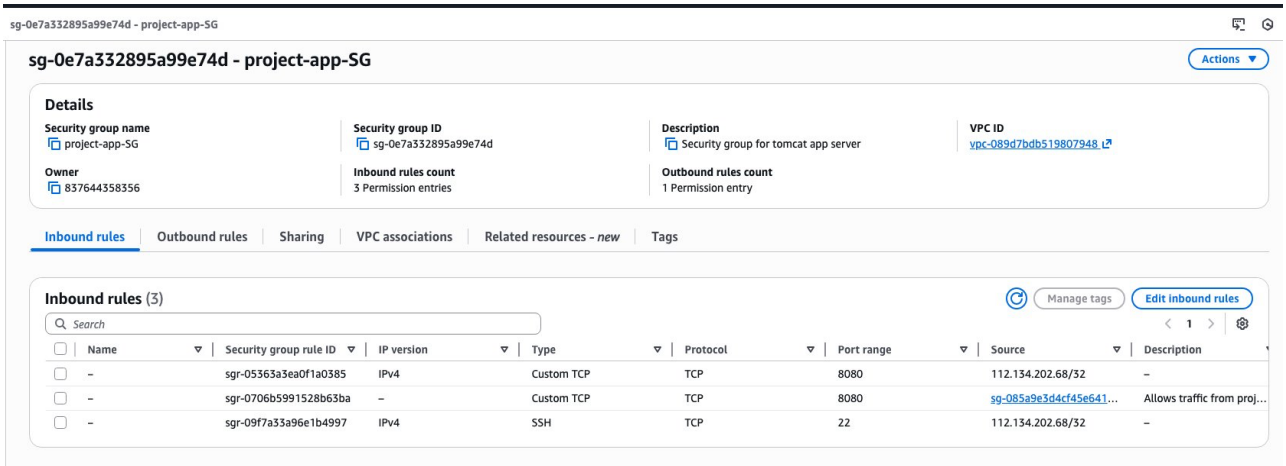

Figure 6 : project-app-SG: port 8080 from ELB-SG and management IP; SSH 22 from my IP

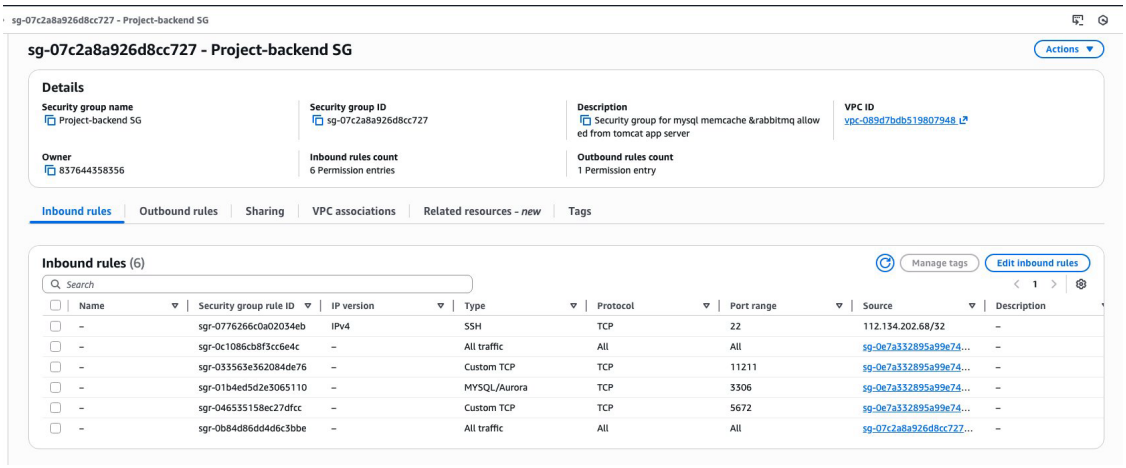

Figure 7 : Project-backend SG: MySQL 3306, Memcached 11211, RabbitMQ 5672 all from project-app-SG only

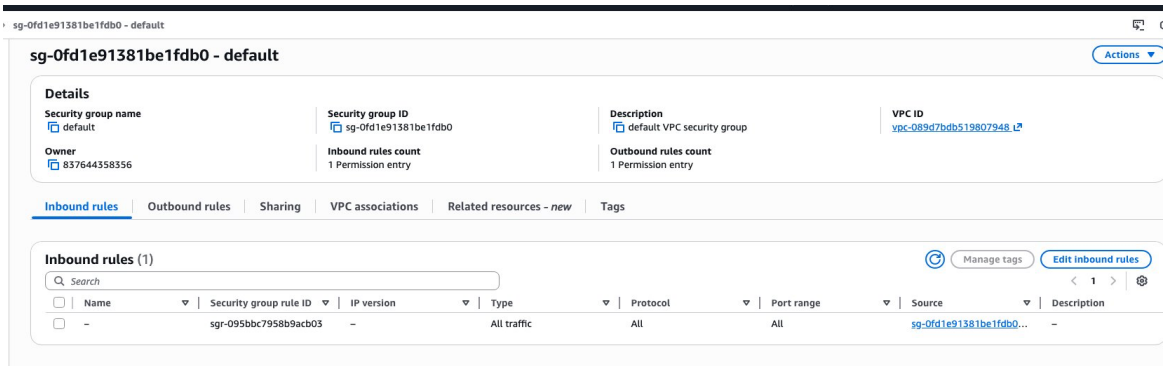

Figure 8 : Default VPC security group: all traffic from within the same group only

Step 4 : Launch EC2 Instances Using User Data Scripts

Four EC2 t3.micro instances were launched project-app01, project-rmq01, project-db01, and project-mc01. Each instance was given a bash script via the user data field at launch time. These scripts run automatically on first boot and install and configure the required software without any manual intervention after launch. All four instances landed in the us-east-2c availability zone, received public IPv4 addresses, and passed all 3 of 3 EC2 system and instance status checks, confirming that the underlying hardware and the operating system are both healthy.

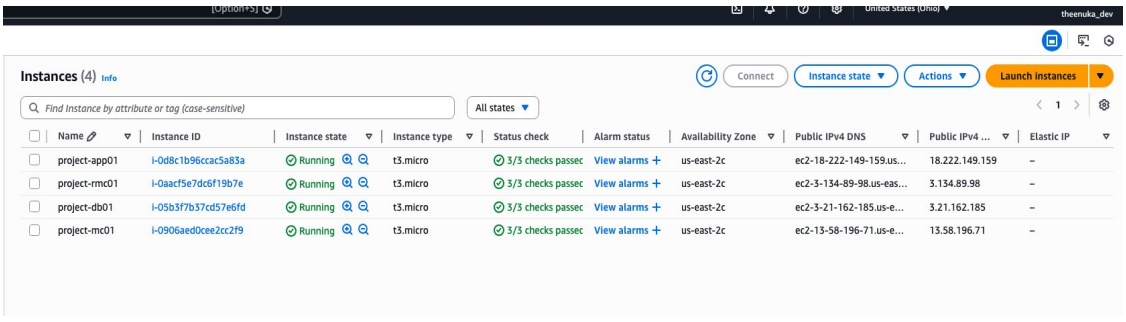

Figure 9 : All four EC2 instances running with 3/3 status checks passed, us-east-2c

Step 5 : Configure Route 53 Private DNS

A private hosted zone named vprofile.in was created in Route 53 and associated with the project VPC. Six records are present in total: an NS record and an SOA record (both created automatically by Route 53), plus four A records added manually app01.vprofile.in pointing to private ip of app instance, db01.vprofile.in to private ip of db instance, mc01.vprofile.in to private ip of memcached instance, and rmq01.vprofile.in to private ip of rabbitmq instance. All four A records have a TTL of 300 seconds. The Tomcat application uses these hostnames in its application.properties configuration file, so if an instance IP changes, only the Route 53 record needs updating.

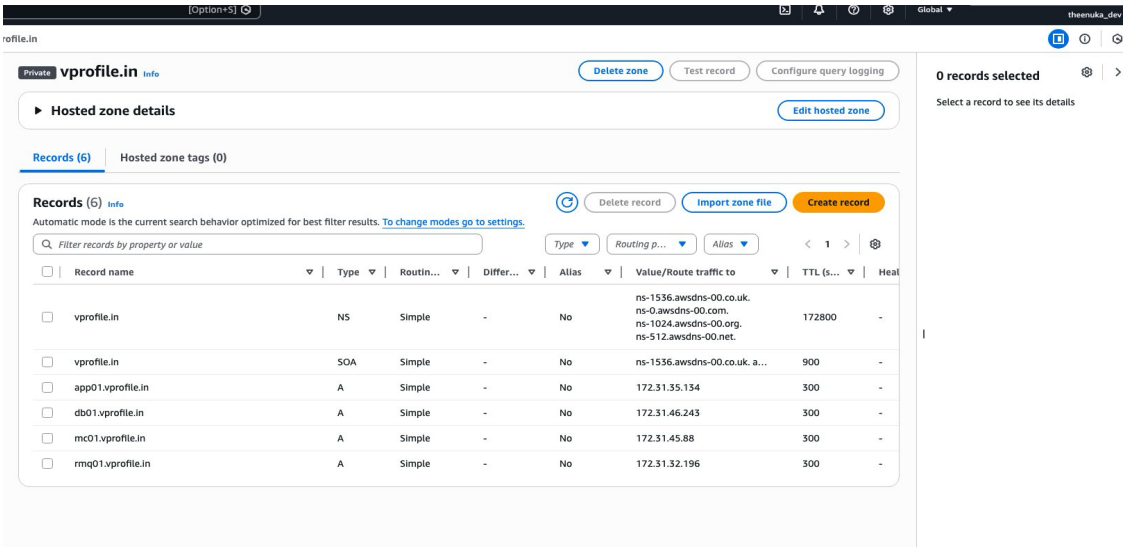

Figure 10 : Route 53 private hosted zone vprofile.in: NS, SOA, and 4 A records with TTL 300

Step 6 : Build the Application from Source Code

The VProfile Java application source code was compiled on a local machine using Apache Maven with the command mvn install. Maven scanned the pom.xml, resolved dependencies, copied resources, compiled the Java classes ran the test suite,packaged the webapp into a WAR file, and installed it to the local Maven repository.

```
(base) theenukabandara@theenukas-MacBook-Pro vprofile-project % mvn install
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------------< com.visualpathit:vprofile >-----------------------
[INFO] Building Visualpathit VProfile Webapp v2
[INFO]   from pom.xml
[INFO] --------------------------------[ war ]---------------------------------
[WARNING] The artifact org.hibernate:hibernate-validator:jar:6.2.0.Final has been relocated to org.hibernate.validator:hibernate-validator:jar:6
inal
[WARNING] The artifact mysql:mysql-connector-java:jar:8.0.33 has been relocated to com.mysql:mysql-connector-j:jar:8.0.33: MySQL Connector/J art
 moved to reverse-DNS compliant Maven 2+ coordinates.
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ vprofile ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 5 resources from src/main/resources to target/classes
[INFO]
[INFO] --- compiler:3.13.0:compile (default-compile) @ vprofile ---
[INFO] Nothing to compile - all classes are up to date.
[INFO]
[INFO] --- resources:3.3.1:testResources (default-testResources) @ vprofile ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory /Users/theenukabandara/Desktop/AWS Lift and Shift project/vprofile-project/src/test/resources
```

Figure 11 : Maven build: dependency resolution, resource copying, and compilation phase

```
[INFO]
[INFO] Tests run: 0, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- war:3.4.0:war (default-war) @ vprofile ---
[INFO] Packaging webapp
[INFO] Assembling webapp [vprofile] in [/Users/theenukabandara/Desktop/AWS Lift and Shift project/vprofile-project/target/vprofile-v2]
[INFO] Processing war project
[INFO] Copying webapp resources [/Users/theenukabandara/Desktop/AWS Lift and Shift project/vprofile-project/src/main/webapp]
[INFO] Building war: /Users/theenukabandara/Desktop/AWS Lift and Shift project/vprofile-project/target/vprofile-v2.war
[INFO]
[INFO] --- install:3.1.2:install (default-install) @ vprofile ---
[INFO] Installing /Users/theenukabandara/Desktop/AWS Lift and Shift project/vprofile-project/pom.xml to /Users/theenukabandara/.m2/repository/com/visu
alpathit/vprofile/v2/vprofile-v2.pom
[INFO] Installing /Users/theenukabandara/Desktop/AWS Lift and Shift project/vprofile-project/target/vprofile-v2.war to /Users/theenukabandara/.m2/repo
sitory/com/visualpathit/vprofile/v2/vprofile-v2.war
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.281 s
[INFO] Finished at: 2026-02-10T07:58:03+05:30
[INFO] ------------------------------------------------------------------------
```

Figure 12 : Maven BUILD SUCCESS: vprofile-v2.war packaged and installed

Step 7 : Upload the Artifact to S3

After the build succeeded, the ls target/ command confirmed that vprofile-v2.war was present in the target directory alongside compiled classes and other build outputs. The artifact was then uploaded to S3 with: aws s3 cp target/vprofile-v2.war s3://vprofile-theenu-artifacts. The upload progress confirmed the transfer completed. Running aws s3 ls s3://vprofile-theenu-artifacts afterwards showed a single object: vprofile-v2.war. The S3 console confirmed the object in the bucket vprofile-theenu-artifacts with Standard storage class.

```
(base) theenukabandara@theenukas-MacBook-Pro vprofile-project % ls target/
classes                 generated-test-sources  maven-status            test-classes            vprofile-v2.war
generated-sources       maven-archiver          surefire-reports        vprofile-v2
(base) theenukabandara@theenukas-MacBook-Pro vprofile-project % aws s3 cp target/vprofile-v2.war s3://vprofile-theenu-artifacts
upload: target/vprofile-v2.war to s3://vprofile-theenu-artifacts/vprofile-v2.war
(base) theenukabandara@theenukas-MacBook-Pro vprofile-project % aws s3 ls s3://vprofile-theenu-artifacts
2026-02-10 08:02:15   83275478 vprofile-v2.war
(base) theenukabandara@theenukas-MacBook-Pro vprofile-project %
```

Figure 13 : CLI upload: aws s3 cp and verification with aws s3 ls

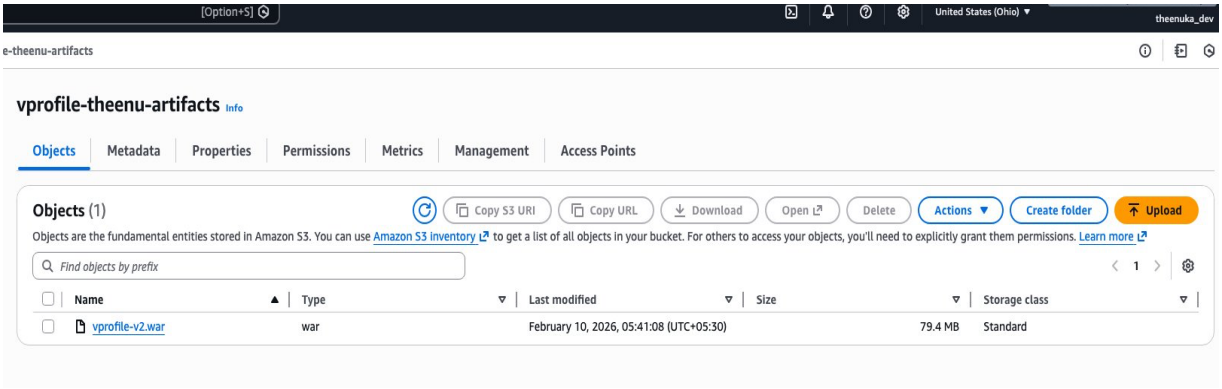

Figure 14 : S3 bucket vprofile-theenu-artifacts with vprofile-v2.war

Step 8 : Deploy the Artifact to the Tomcat Instance

SSH was used to connect to project-app01. The following commands were run in sequence: systemctl daemon-reload to reload unit files, systemctl stop tomcat10 to stop the running Tomcat service, ls /var/lib/tomcat10/webapps/ which showed the existing ROOT directory, rm -rf /var/lib/tomcat10/webapps/ROOT to remove it, cp /tmp/vprofile-v2.war /var/lib/tomcat10/webapps/ROOT.war to place the new artifact as ROOT.war so the application serves at the root path, and systemctl start tomcat10 to restart the service. A final ls

/var/lib/tomcat10/webapps/ confirmed both ROOT and ROOT.war were present, showing that Tomcat had already begun extracting the WAR file.

```
[root@ip-172-31-35-134:~# systemctl daemon-reload
 root@ip-172-31-35-134:~# systemctl stop tomcat10
[root@ip-172-31-35-134:~# ls /var/lib/tomcat10/webapps/
[ROOT
[root@ip-172-31-35-134:~# rm -rf /var/lib/tomcat10/webapps/ROOT
 root@ip-172-31-35-134:~# cp /tmp/vprofile-v2.war /var/lib/tomcat10/webapps/ROOT.war
[root@ip-172-31-35-134:~# systemctl start tomcat10
[root@ip-172-31-35-134:~# ls /var/lib/tomcat10/webapps/
 ROOT  ROOT.war
```

Figure 15 : Tomcat deployment: stop service, remove old ROOT, copy new ROOT.war, restart service

Step 9 : Set Up the Application Load Balancer with HTTPS

First, a wildcard SSL/TLS certificate for *.theenuka.xyz was requested from AWS Certificate Manager. The certificate is of type Amazon Issued and reached a status of Issued, meaning AWS validated domain ownership and the certificate is active. The certificate is shown as being in use, confirming it was attached to the load balancer.

The Application Load Balancer vprofile-theenuka-elb was created as an internet-facing IPv4 load balancer in VPC, spanning three subnets across availability zones us-east-2a, us-east-2b, and us-east-2c.

A target group named vprofile-theenuka-tg was configured with target type Instance, protocol HTTP, port 8080, and protocol version HTTP1. The target group is associated with vprofile-theenuka-elb.

Two listeners were configured: HTTP:80 forwarding to target group vprofile-theenuka-tg at 100%, and an HTTPS listener using the ACM certificate.

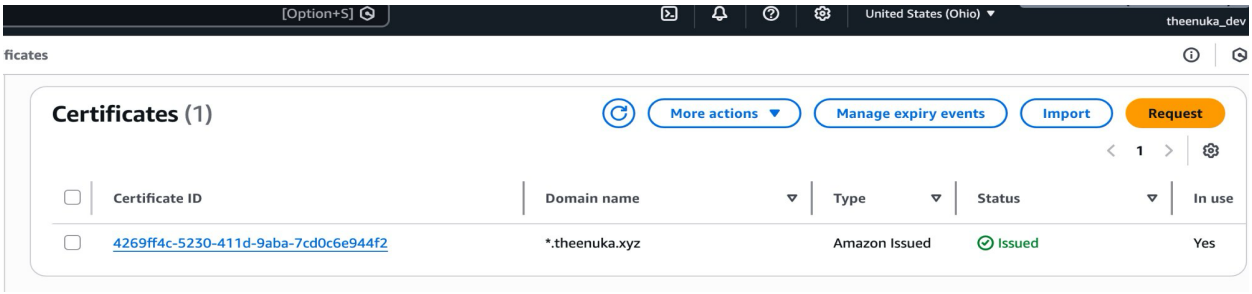

*Figure 16 : ACM certificate *.theenuka.xyz*

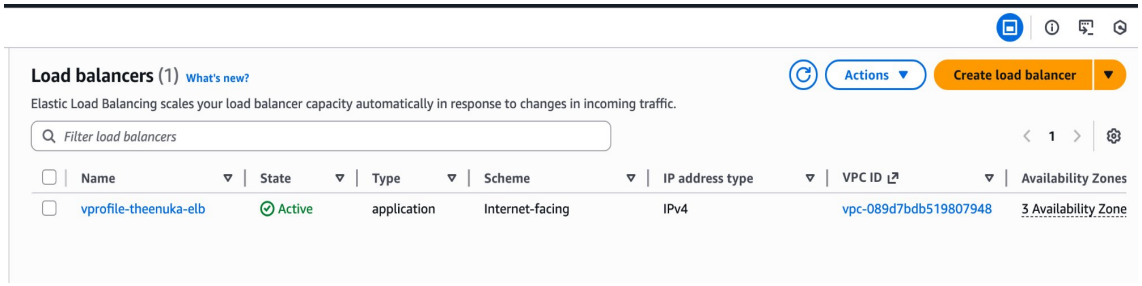

Figure 17 : Load balancers list: vprofile-theenuka-elb, Active, application type, internet-facing

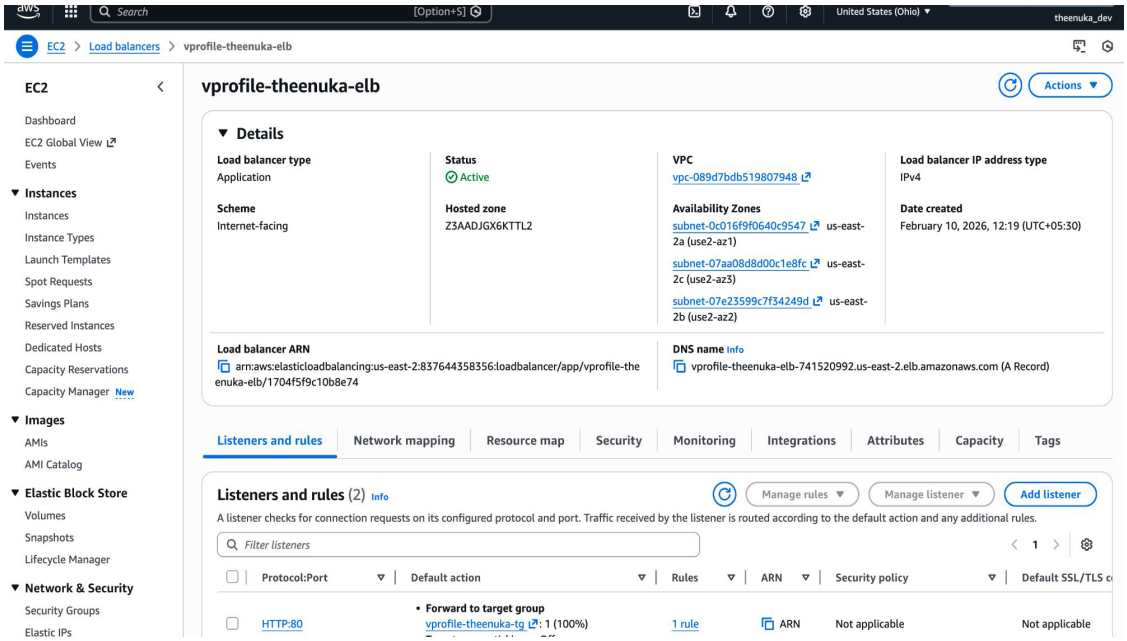

Figure 18 : ELB details: DNS name, hosted zone, 3 AZs, listeners (HTTP:80 and HTTPS)

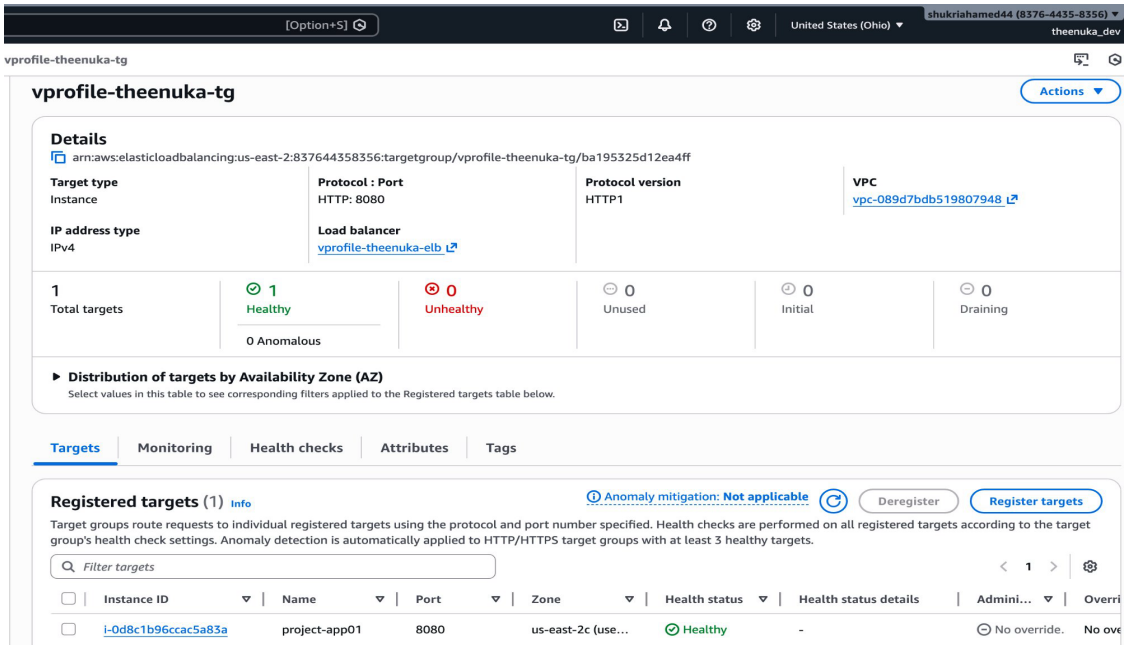

Figure 19 : Target group vprofile-theenuka-tg: project-app01 on port 8080 showing Healthy status

Step 10 : Point the Domain to the Load Balancer via GoDaddy DNS

In GoDaddy's DNS management panel for the domain theenuka.xyz, a CNAME record was added with name vprofileapp and value vprofile-theenuka-elb-741520992.us-east-2.elb.amazonaws.com, with a TTL of 1 hour. This record routes anyone visiting vprofileapp.theenuka.xyz to the load balancer. Two other CNAME records are also visible: a www record pointing to theenuka.xyz itself (for general website use), and a long ACM validation CNAME (name starting _d85e30f3cd7a52e…, value starting _35d2bd1dc87eaaafa…jkddzztszm.acm-validations.aws) which is what allowed the ACM certificate request to be validated via DNS.

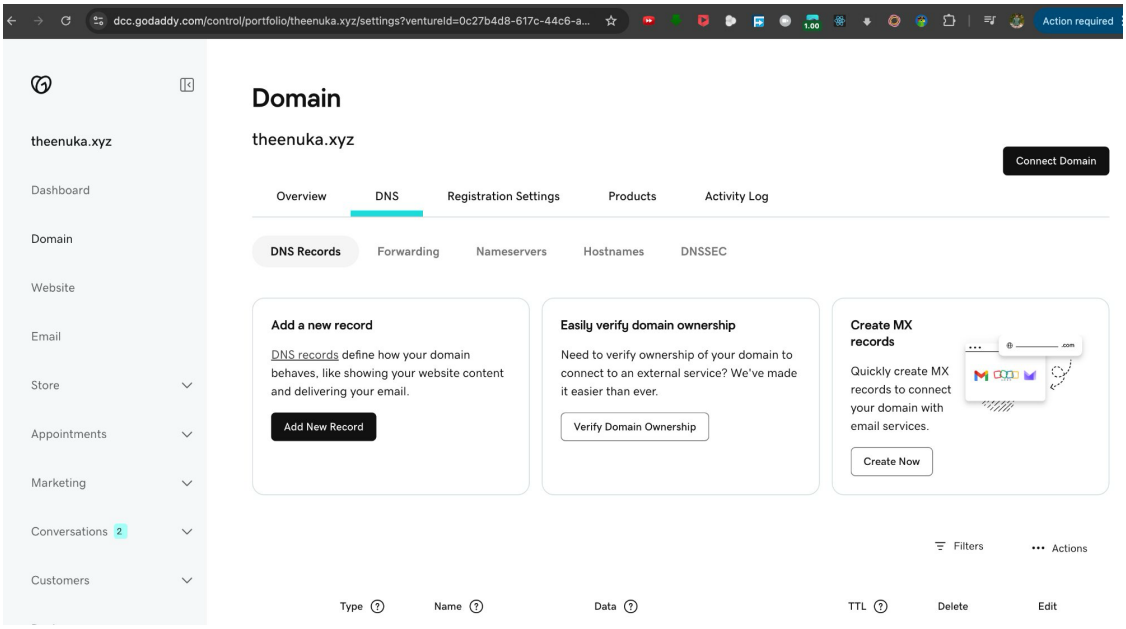

Figure 20 : GoDaddy DNS management panel for theenuka.xyz showing all DNS records

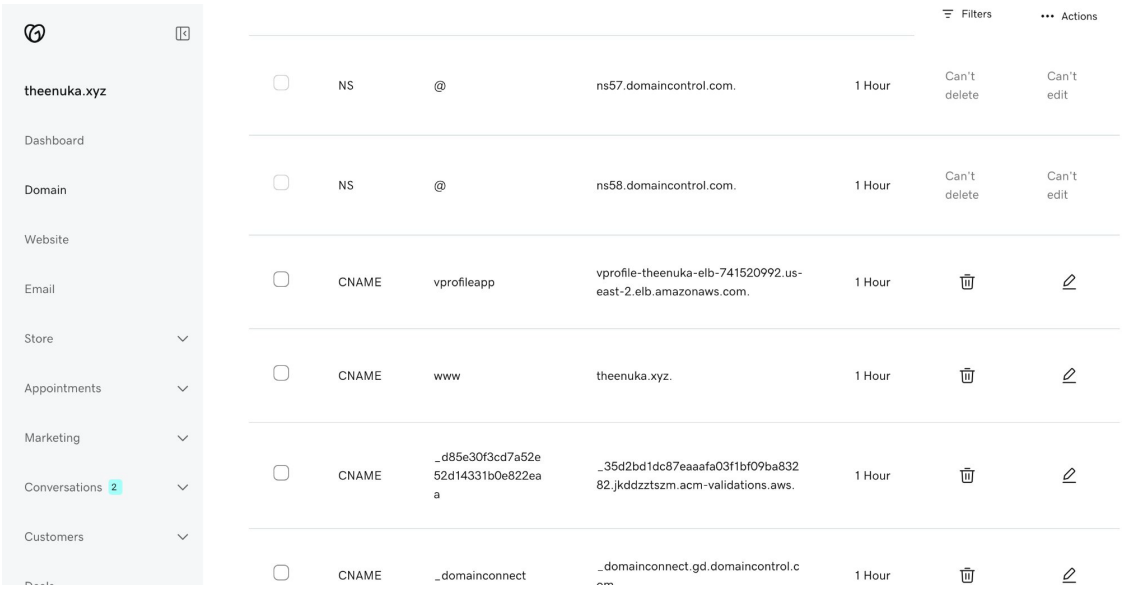

Figure 21 : CNAME records: vprofileapp → ELB endpoint, www → theenuka.xyz, ACM validation record

Step 11 : Verify the Full Deployment

A complete end-to-end verification was carried out through the browser.

The login page of the VProfile application loaded correctly at the URL vprofileapp.theenuka.xyz, confirming that the DNS CNAME record resolved successfully, the load balancer forwarded the request, and Tomcat served the application. The browser address bar showed the HTTPS padlock icon. Opening the browser security panel confirmed the message Connection is secure, with a note that Certificate is valid.

Viewing the full certificate in the certificate viewer confirmed the following: it was issued to *.theenuka.xyz, issued by Amazon RSA 2048 M04 (Organisation: Amazon), with a validity period. This confirmed that the ACM wildcard certificate was correctly serving traffic through the load balancer.

After logging in, the HKH Infotech social media dashboard loaded at the /welcome route, showing posts, user activity, photos, and contacts, confirming that the Tomcat application was processing authenticated requests and rendering dynamic content correctly.

Navigating to /user/rabbit showed the RabbitMQ status page with the message RabbitMQ Initiated Generated 5 Connections 3 Channels, 6 Exchange, and 1 Queues. This confirmed that the Tomcat application had successfully connected to the RabbitMQ instance on project-rmq01 and that the message broker was handling connections.

Opening the users list at /users showed a table of user records fetched from MySQL on project-db01, confirming database connectivity. Clicking on the first user (Hibo Prince, ID 4) produced two consecutive tests. The first load of the user detail page showed the red header Data is From DB and Data Inserted In Cache, meaning the application queried MySQL for the user data and simultaneously wrote it into Memcached. Navigating to the same URL again showed Data is From Cache, confirming that Memcached on project-mc01 returned the cached result without touching the database. This validated the complete data flow: MySQL for persistence, Memcached for caching, and correct cache-aside pattern implementation.

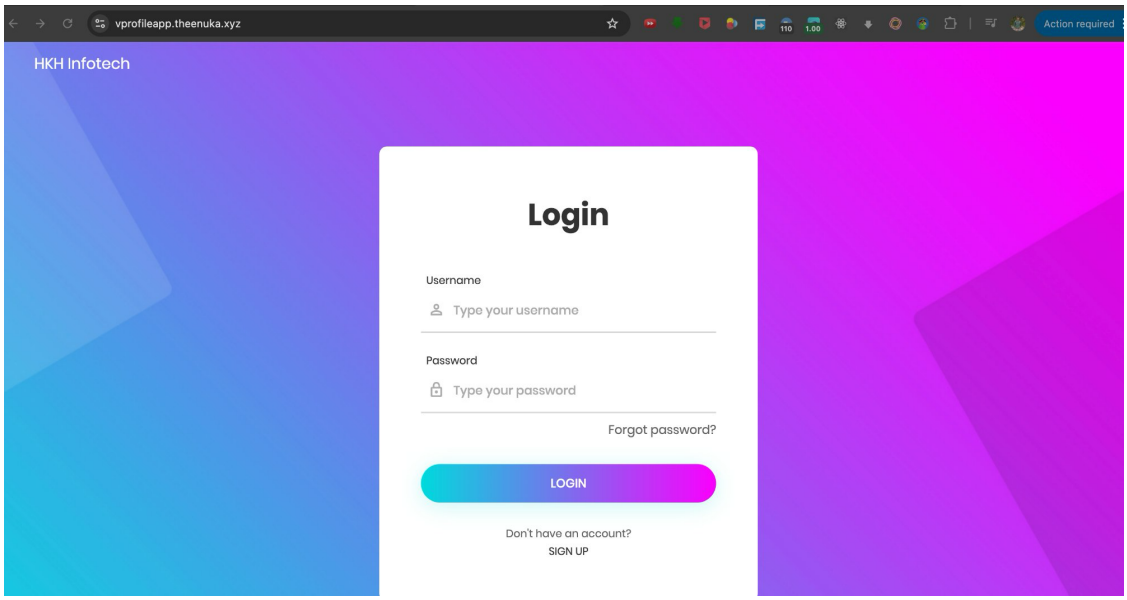

Figure 22 : VProfile application login page loading at vprofileapp.theenuka.xyz over HTTPS

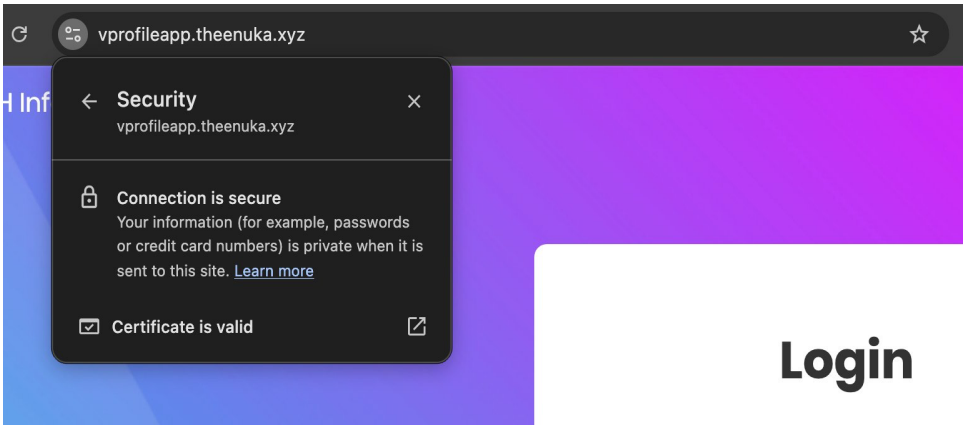

Figure 23 : Browser security panel: Connection is secure, Certificate is valid

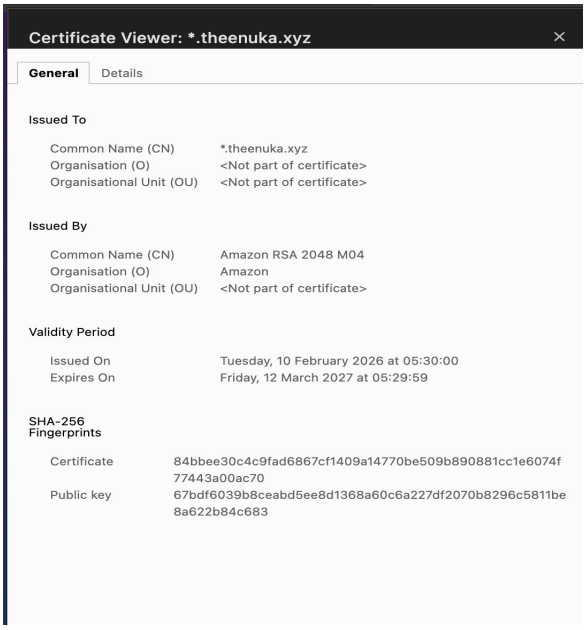

*Figure 24 : Certificate Viewer: *.theenuka.xyz, issued by Amazon RSA 2048 M04*

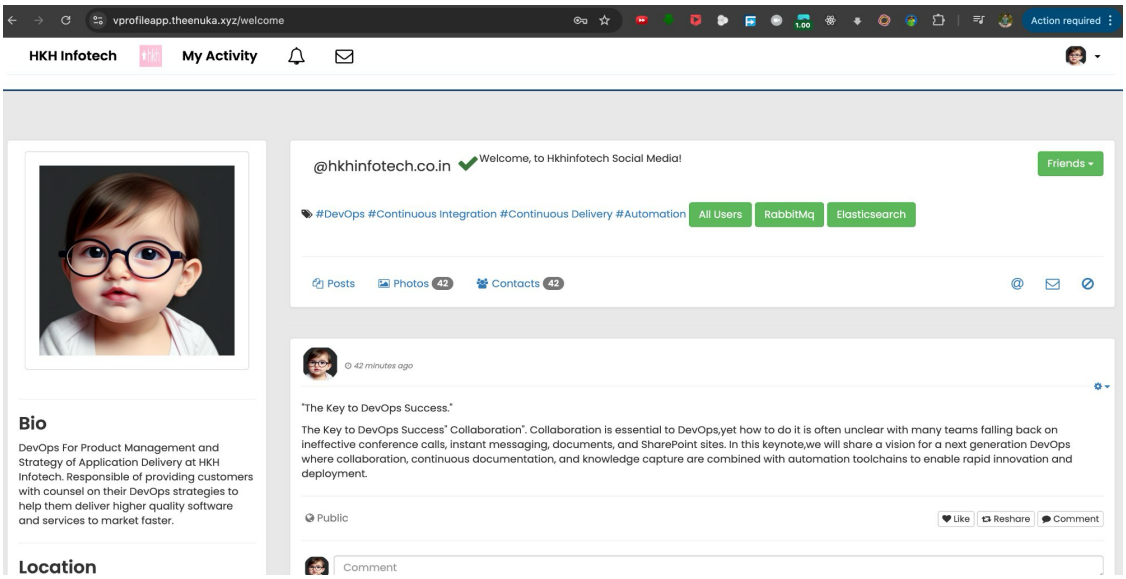

Figure 25 : Application dashboard (HKH Infotech) after successful login at /welcome

Figure 26 : RabbitMQ status at /user/rabbit: 5 connections, 3 channels, 6 exchanges, 1 queue

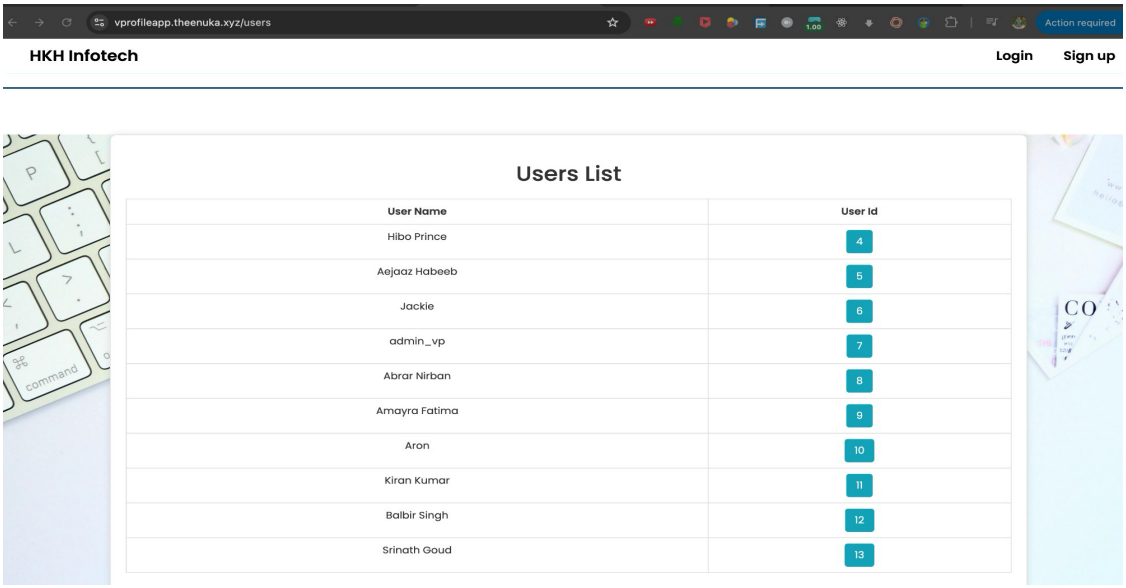

Figure 27 : Users list page at /users: MySQL returning user records correctly

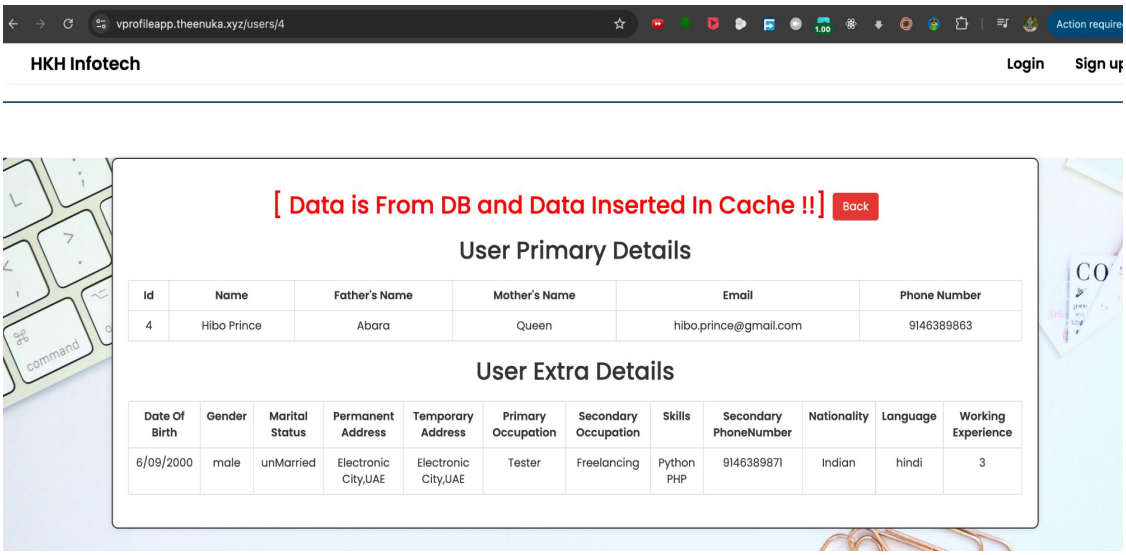

Figure 28 : First request to user/4: data fetched from MySQL and inserted into Memcached

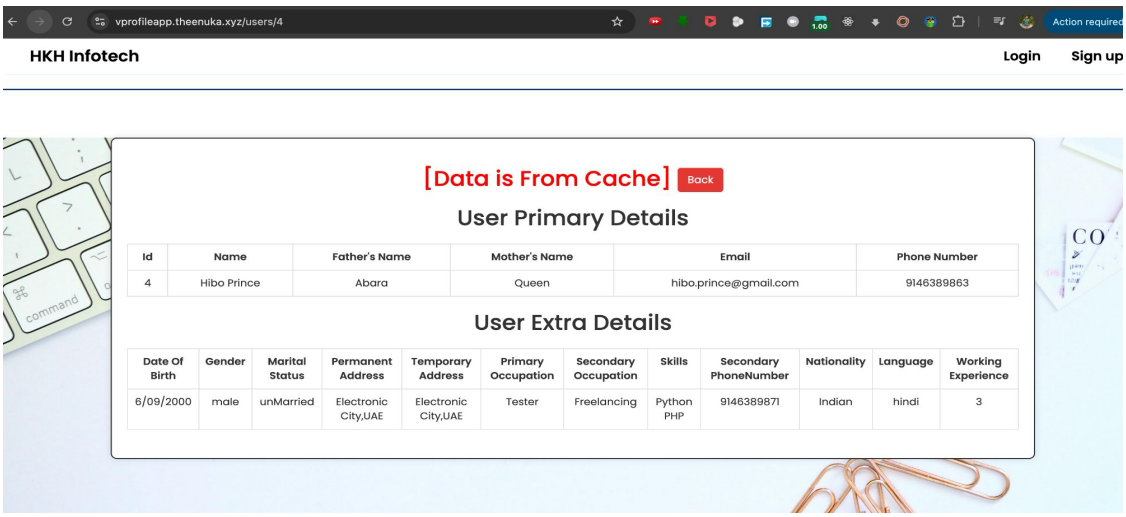

Figure 29 : Second request to same URL: data served from Memcached without hitting the database

Step 12 : Build an Auto Scaling Group for the Application Tier

To enable automatic scaling of the Tomcat application tier, three resources were created in sequence.

First, an Amazon Machine Image (AMI) named vprofile-theenuka-app-ami was created from the fully configured and deployed project-app01 instance. The AMI is private and owned by account. It captures the complete disk state of the instance at the time of creation, including the OS, installed packages, Tomcat configuration, and the deployed vprofile-v2.war, so any new instance launched from it is ready to serve traffic immediately.

Second, a Launch Template named vprofile-theenuka-app-LT) was created. According to the Auto Scaling Group detail view, this template specifies AMI ID, instance type t3.micro, key pair project-prod-key, and security group (project-app-SG).

Third, an Auto Scaling Group named vprofile-theenuka-app-asg was created using this Launch Template. It was configured with a desired capacity of 1 instance, a minimum of 1, and a maximum of 4, across 3 availability zones. The ASG is associated with the load balancer so that any instance it launches is automatically registered with the target group and receives traffic as soon as it passes health checks. At the time of the screenshot the status showed Updating capacity with 0 current instances, indicating the ASG had just been created and was provisioning its first instance.

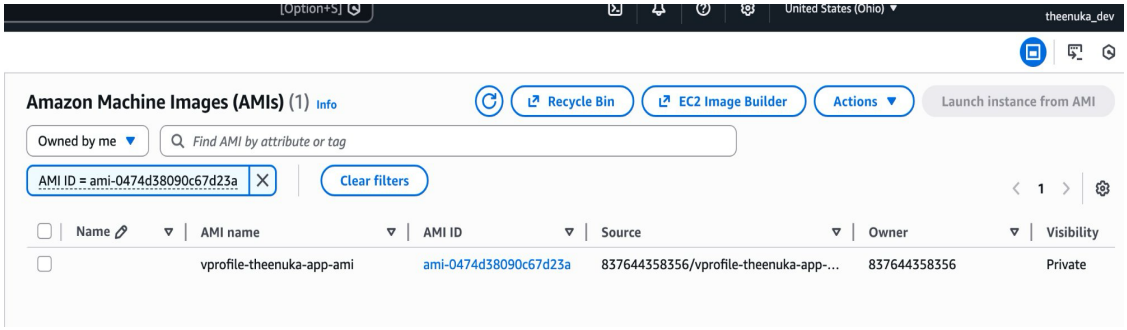

Figure 30 : AMI: vprofile-theenuka-app-ami

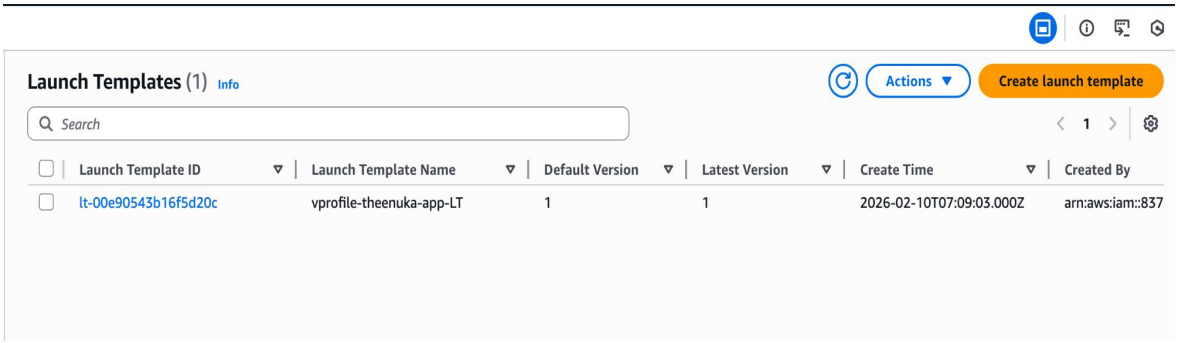

Figure 31 : Launch Template: vprofile-theenuka-app-LT

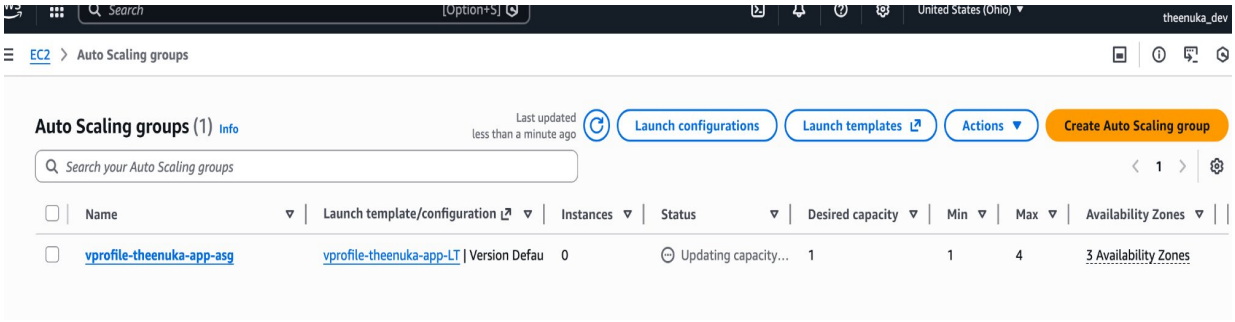

Figure 32 : Auto Scaling Group: vprofile-theenuka-app-asg

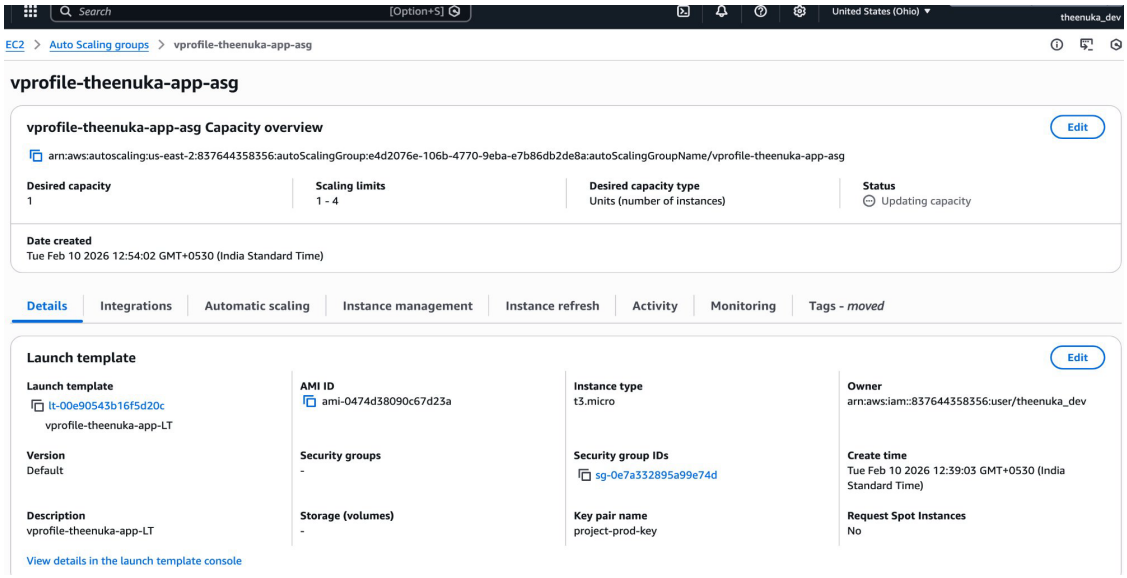

Figure 33 : ASG detail: launch template, AMI, instance type t3.micro, security group, key pair project-prod-key

3. Technical Skills Demonstrated

AWS Infrastructure Provisioning

Provisioned and interconnected the full set of AWS resources required for a production deployment: EC2 instances, key pairs, security groups, S3 bucket, Route 53 private hosted zone, Application Load Balancer with target group, ACM wildcard certificate, custom AMI, Launch Template, and Auto Scaling Group. Each resource was configured with accurate IDs and dependencies, reflecting a practical understanding of how AWS services relate to one another.

Linux Administration and Service Automation

Configured four distinct Linux server roles using bash user data scripts for automated first-boot provisioning, eliminating manual post launch setup. Performed direct server administration including service lifecycle management with systemctl, file system operations, and application deployment via SSH. The deployment sequence daemon-reload, stop service, remove old deployment, copy new artifact, start service follows standard Tomcat deployment practice.

Layered Network Security

Designed a three-tier security group architecture that enforces strict traffic flow between layers. The ELB SG is the only group open to the internet. The app SG accepts port 8080 only from the ELB SG. The backend SG accepts database, cache, and broker ports only from the app SG. Each data-tier service is completely unreachable from the internet or from any source outside the app tier. The SSH access from a specific management IP (My IP) rather than from 0.0.0.0/0 demonstrates awareness of access control best practice.

Build Pipeline and Artifact Management

Executed the full software delivery workflow: compiling a Java web application from source with Maven, generating a WAR artifact, uploading it to an S3 bucket for centralised and durable storage, and deploying it to a Tomcat server on EC2. Storing the artifact in S3 separates the build environment from the deployment target and makes the same artifact available for any instance the Auto Scaling Group might launch in the future.

DNS and Certificate Management

Implemented a two-layer DNS architecture: Route 53 private hosted zone for internal VPC service discovery using A records with short TTLs, and GoDaddy public DNS CNAME records for external domain routing to the load balancer. Obtained and validated a wildcard ACM certificate via DNS challenge, which is the recommended validation method for production environments because it does not require a running web server and supports automated renewal.

High Availability and Auto Scaling

Deployed the Application Load Balancer across three availability zones so that a failure of a single AWS data centre does not take the application offline. Created a custom AMI from the production instance to act as the golden image for scaling, ensuring that new instances launched by the Auto Scaling Group (up to a maximum of four) are identical to the original and ready to serve traffic immediately after passing health checks.

End-to-End Application Validation

Verified every component in the stack independently and as an integrated whole: HTTPS termination through ACM certificate at the load balancer, dynamic HTML rendering by Tomcat, SQL query execution by MySQL, cache population and cache hit serving by Memcached (confirmed by the DB→Cache and Cache-only indicator messages), and asynchronous messaging by RabbitMQ (confirmed by connection count and channel statistics). This systematic verification approach mirrors the acceptance testing phase of a real production deployment.

4. Conclusion

This project demonstrates a complete, working lift-and-shift migration of the VProfile multi-tier web application to AWS. The application is live and accessible at https://vprofileapp.theenuka.xyz, served securely over HTTPS with an Amazon-issued wildcard certificate, through an internet-facing Application Load Balancer that spans three availability zones. Behind it, four EC2 instances handle application serving (Tomcat), data persistence (MySQL), result caching (Memcached), and asynchronous messaging (RabbitMQ), with all inter-service traffic flowing through private IP addresses resolved by Route 53 internal DNS.

The entire data flow was verified from a user logging in, to the application querying MySQL and writing to Memcached, to a second request being served entirely from cache. The message broker was confirmed operational. The infrastructure was secured at every layer with purpose built security groups. An Auto Scaling Group backed by a custom AMI and Launch Template stands ready to scale the application tier out to four instances in response to demand, with new instances automatically registering with the load balancer and serving traffic without any manual intervention.

Building this project developed practical skills across a range of disciplines that directly apply to cloud engineering and DevOps work: AWS infrastructure provisioning, Linux server administration, Java application build pipelines with Maven, network security design, DNS architecture, SSL/TLS certificate management, and systematic end-to-end deployment validation.
