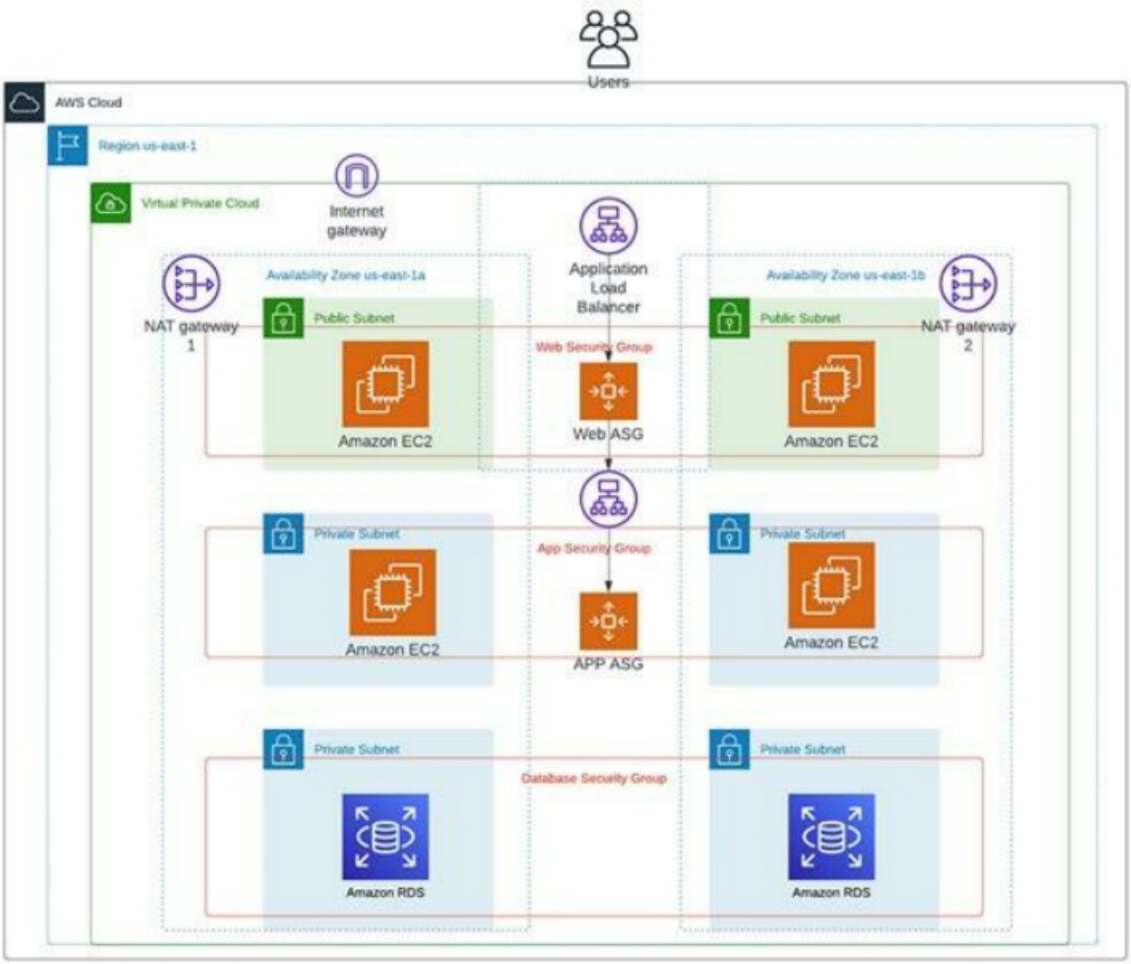
3-TIER ARCHITECTURE

NAME : CHINNAMSETTI VENKATARJA **BATCH NO :** 133

EMAIL ID: rajachinnamsetti2003@gmail.com **DATE:** 26-02-2025

TRAINER: Mr. Madhukar Reddy Sir

AIM: This project focuses on designing and implementing a **3-tier architecture** on **AWS** that ensures **scalability, security, and high availability** by separating the **web, application, and database layers**. The architecture enhances performance, fault tolerance, and follows cloud infrastructure best practices by leveraging AWS services such as **Amazon EC2, VPC, RDS, Auto Scaling Groups, and an Application Load**



1. INTRODUCTION:

1. 1 Project Overview

This project focuses on designing and implementing a **highly available, scalable, and secure 3-tier architecture on AWS** to host a web application efficiently. The architecture is structured into three distinct layers, ensuring optimal **performance, fault tolerance, and security** while following best practices for cloud infrastructure.

The **three-tier architecture** consists of:

🟢 **Presentation Layer (Web Tier):** Public-facing EC2 instances deployed in a Public Subnet, managed by an Auto Scaling Group (ASG) and distributed using an Application Load Balancer (ALB) for handling incoming user requests efficiently.

🟠 **Application Layer (App Tier):** Private EC2 instances running business logic, deployed in a Private Subnet with restricted access for enhanced security.

🔵 **Data Layer (Database Tier):** A managed Amazon RDS instance deployed in a Private Subnet, ensuring data integrity, security, and performance with Multi-AZ replication.

1. 2 Project Objectives:

High Availability & Fault Tolerance:

- Deploy resources across multiple **Availability Zones (AZs)** for redundancy.
- Utilize **Multi-AZ RDS deployment** to ensure database failover protection.
- Implement **Elastic Load Balancing (ALB)** for intelligent traffic distribution.

Scalability & Performance Optimization:

- Use **Auto Scaling Groups (ASGs)** to dynamically adjust resources based on traffic demand.
- Optimize network latency and response time using **efficient load balancing strategies**.

Security & Network Isolation:

- Implement **VPC network segmentation** with **public and private subnets** to enforce isolation.
- Deploy **NAT Gateways** to allow private instances to securely access the internet without direct exposure.

2. ARCHITECTURE OVERVIEW

This architecture follows a **3-tier model**, ensuring **scalability, security, and high availability** by leveraging AWS services for **network segmentation, load balancing, auto-scaling, and database management**. The infrastructure is designed to optimize **performance, fault tolerance, and security** using best practices.

AWS Services Used:

- **Networking:** Uses **VPC, Subnets, NAT Gateway, and Internet Gateway** to segment and route network traffic securely.
- **Load Balancing:** An Application Load Balancer (ALB) efficiently distributes incoming traffic across multiple servers for better availability and performance.
- **Scaling:** EC2 instances managed by Auto Scaling Groups (ASG) dynamically adjust resources to handle varying traffic loads efficiently.
- **Database Tier:** Amazon RDS (Multi-AZ deployment) ensures high availability and reliable storage of persistent data.

Network Segmentation:

The architecture employs a **Virtual Private Cloud (VPC)** with multiple public and private subnets for isolation and security:

- **Public Subnets (Web Tier):** Hosts the web servers, allowing access from the internet via an **Application Load Balancer (ALB)**.
- **Private Subnets (Application Tier):** Hosts the backend application servers, restricting access to only the web tier and database tier.
- **Private Subnets (Database Tier):** Stores application data securely, accessible only by the application servers to prevent direct exposure.

Security Groups & Access Control:

To enhance security, different security groups are implemented for each layer:

- **Web Security Group:** Allows traffic only from the **ALB** to the web servers, preventing direct external access.
- **Application Security Group:** Restricts access to only requests coming from the Web Tier, ensuring no direct exposure.
- **Database Security Group:** Allows connections only from the Application Tier, preventing direct access from the web tier or the internet.

3. IMPLEMENTATION

STEP 1: VPC AND SUBNET SETUP

Create a Virtual Private Cloud (VPC)

- Define a **CIDR block** (e.g., 10.0.0.0/16) to establish a private network within AWS.

Set Up Subnets for Different Tiers

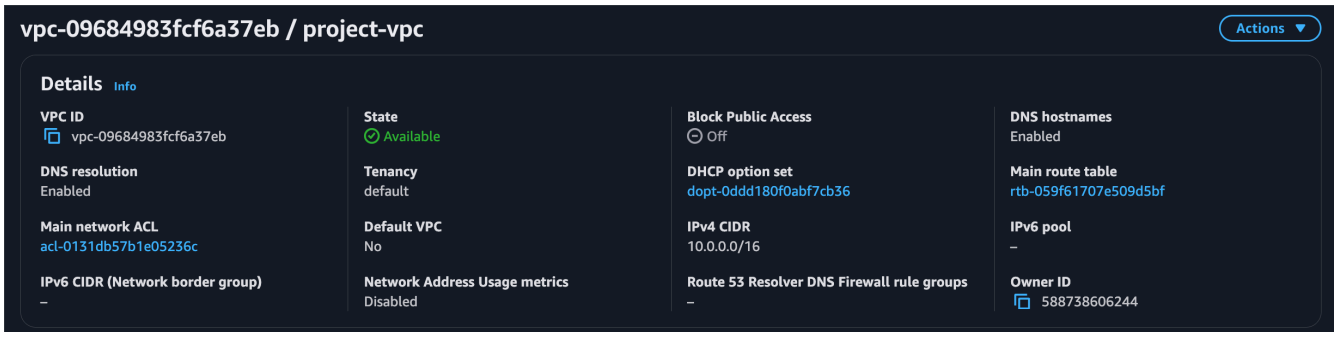
- **Public Subnets:** Deploy **web-tier instances** in these subnets, enabling internet access.
- **Private Subnets:**
  - **Application Tier:** Hosts backend application servers, restricting direct internet access.
  - **Database Tier:** Stores application data securely with controlled access.

Attach an Internet Gateway (IGW)

- Create and attach an **Internet Gateway (IGW)** to the VPC to enable external communication for public subnets.

Configure a NAT Gateway for Private Subnets

- Deploy a **NAT Gateway** in a **public subnet** to allow instances in private subnets to initiate outbound internet requests (e.g., software updates) while preventing inbound traffic.



STEP 2: SECURITY GROUP CONFIGURATION

To enforce **strict access control**, security groups are configured for each layer to regulate incoming and outgoing traffic.

Web Security Group (Web Tier)

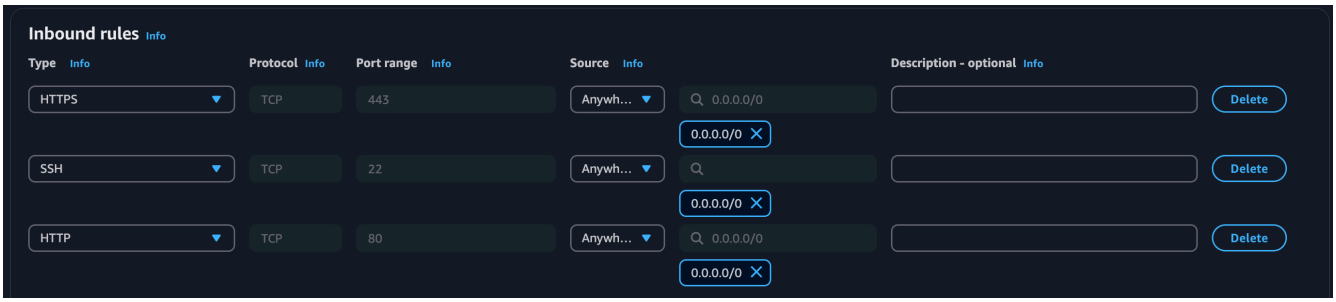
- **Allow HTTP (80) and HTTPS (443)** from **anywhere (0.0.0.0/0, ::/0)** to ensure global accessibility.
- **Allow SSH (22) only from a trusted source** (e.g., your office IP or a bastion host) for secure remote management.

App Security Group (Application Tier)

- **Allow inbound traffic only from the Web Tier** on required ports (e.g., **port 8080 or 5000**, depending on the application).
- **Deny direct access from the internet**, ensuring that only the web tier can communicate with the application servers.

Database Security Group (Database Tier)

- **Allow only MySQL (3306) traffic from the Application Tier**, ensuring that database access is strictly limited to backend servers.



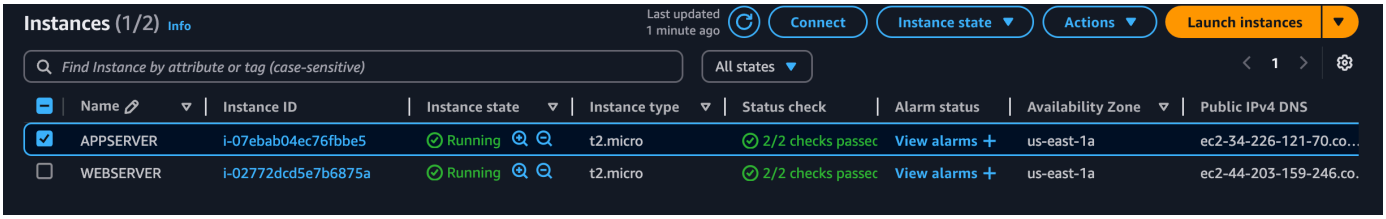
STEP 3: EC2 INSTANCE SETUP

To ensure a scalable and reproducible infrastructure, **Amazon Machine Images (AMIs) and Launch Templates** are used for deploying EC2 instances.

Web Tier Setup:

- Launch an **EC2 instance** in the **Public Subnet**.
- Install a web server (e.g., **NGINX or Apache**) to handle HTTP requests.

```
apt update -y
apt install nginx -y
systemctl start nginx
```



- Configure the firewall to allow **HTTP (80) and HTTPS (443)** traffic.
- Create an **AMI** of the configured instance for future scaling.
- Create a **Launch Template** using the AMI to enable **Auto Scaling**.

App Tier Setup:

- Launch an **EC2 instance** in the **Private Subnet**.
- Install required software for application processing (e.g., **Node.js, Python, or Java**) based on your app.
- If the application requires **MySQL**, install it on this instance or connect it to an **Amazon RDS** instance.

```
sudo apt update -y
sudo apt install mysql-client -y
```

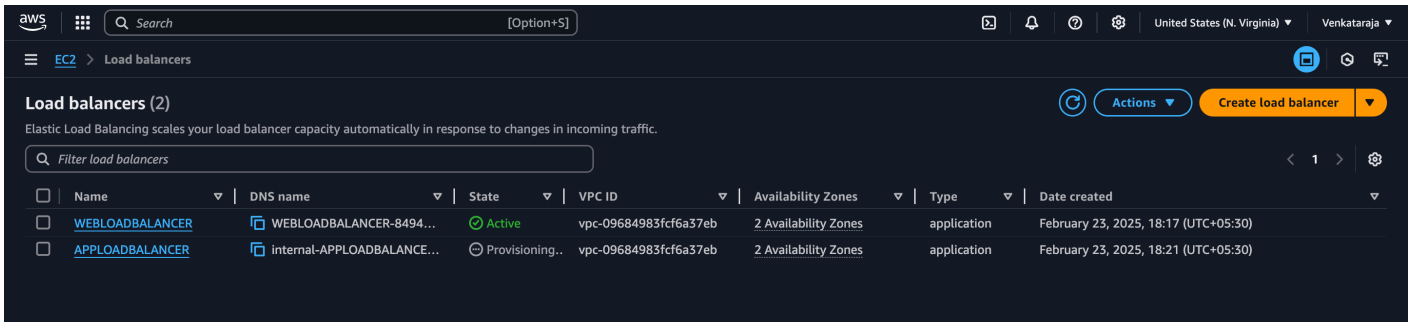
- Configure security to allow connections **only from the Web Tier**.
- Create an **AMI** of the configured instance for consistency.
- Create a **Launch Template** using the AMI to enable **Auto Scaling**.

STEP 4: APPLICATION LOAD BALANCER (ALB) SETUP

To efficiently distribute traffic and ensure high availability, **two Application Load Balancers (ALBs)** are configured—one for the **Web Tier** and another for the **App Tier**.

Create Target Groups:

- **Web Tier Target Group:**
  - Create a **target group** for the web tier with **HTTP (80) and HTTPS (443)** as listener ports.
  - Select **Instance or IP Mode** based on your deployment.
  - No instances will be added initially, as they will be registered dynamically via **Auto Scaling Groups**.
- **App Tier Target Group:**
  - Create a separate **target group** for the app tier with the required **port (e.g., 8080, 5000, or custom app port)**.
  - Set the **target type** as **Instance** and keep it empty initially (to be auto-registered later).

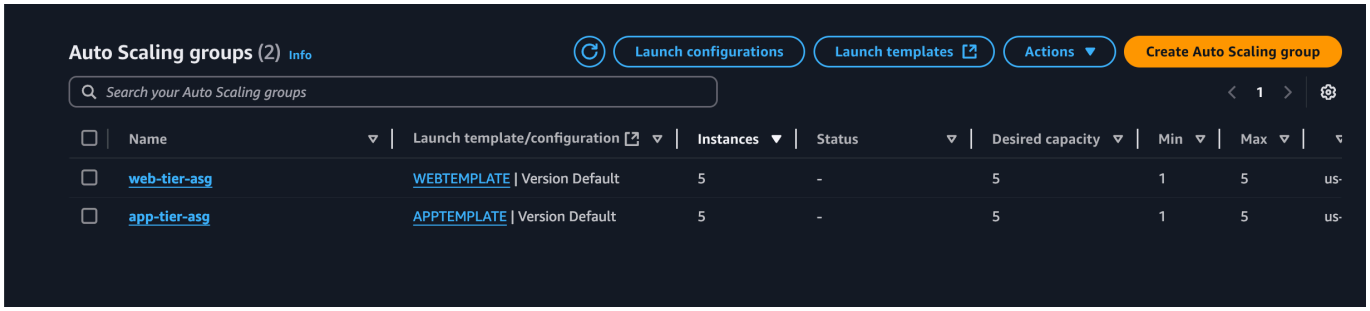


Deploy Two Application Load Balancers (ALBs)

- **Web Tier ALB:**
  - Deploy an **Application Load Balancer (ALB)** in **public subnets**.
  - Attach the **Web Tier Target Group** to route traffic from the internet to web instances.
  - Enable listeners for **HTTP (80) and HTTPS (443)**.
- **App Tier ALB:**
  - Deploy another **Application Load Balancer (ALB)** in **private subnets**.
  - Attach the **App Tier Target Group** to route requests from the **Web Tier to the App Tier**.
  - Set up a listener on the **application's port** (e.g., **8080, 5000**).

STEP 5: AUTO SCALING GROUPS (ASGS) CONFIGURATION

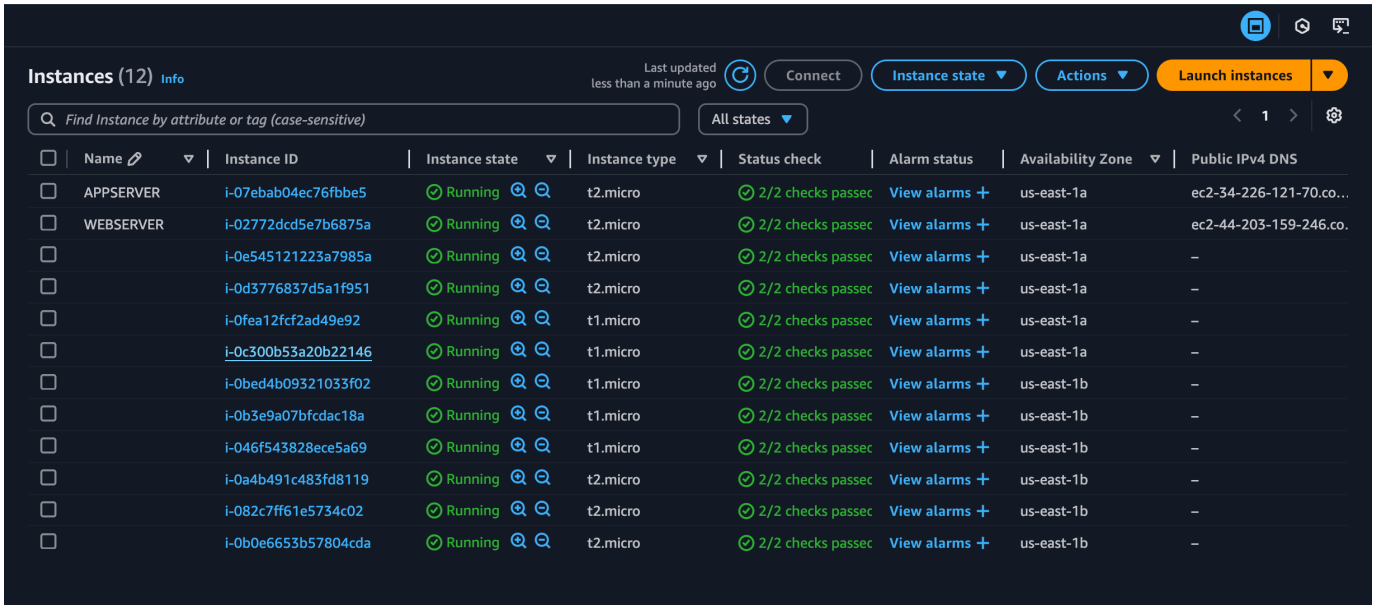
To ensure **high availability, fault tolerance, and scalability**, **Auto Scaling Groups (ASGs)** are set up for both the **Web Tier** and **App Tier** using the previously created **Load Balancers and Target Groups**.

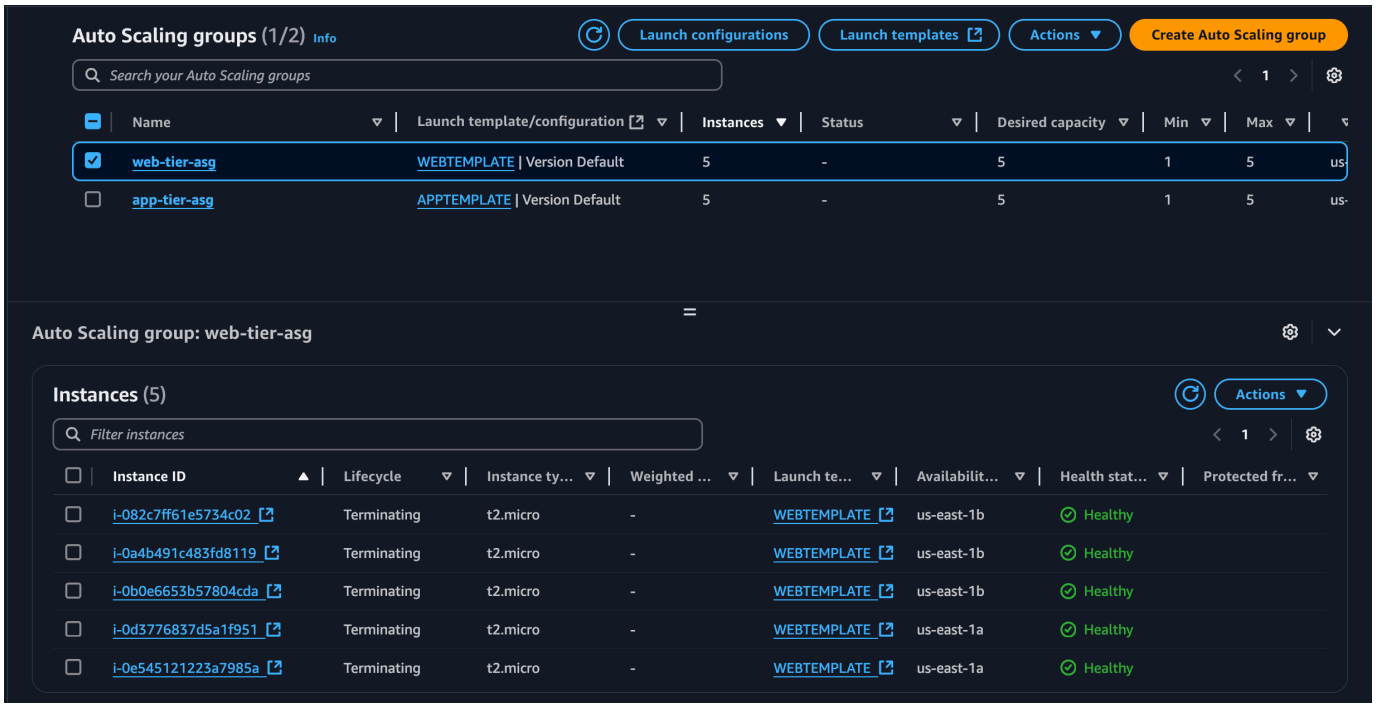


Create an Auto Scaling Group for the Web Tier

- Use the **Launch Template** created for the Web Tier (AMI with NGINX or Apache installed).
- Attach the **Web Tier Target Group** to allow instances to register dynamically.
- Configure the **Application Load Balancer (ALB) for the Web Tier** to distribute incoming traffic.
- Select **Public Subnets** to allow external access.

- Define **scaling policies** (e.g., scale out when CPU utilization > 70%, scale in when CPU < 30%).
- Configure **minimum, desired, and maximum instance count** (e.g., min: 2, desired: 3, max: 5).





Create an Auto Scaling Group for the App Tier

- Use the **Launch Template** created for the App Tier (AMI with backend software installed).
- Attach the **App Tier Target Group** to allow instances to register dynamically.
- Configure the **Application Load Balancer (ALB) for the App Tier** to route traffic from the Web Tier.
- Select **Private Subnets** to **restrict direct external access**.
- Define **scaling policies** based on application load (e.g., request rate, CPU usage).
- Set up **min/max instance count** based on workload demand (e.g., min: 2, desired: 3, max: 5).

STEP 6: RDS DATABASE CONFIGURATION

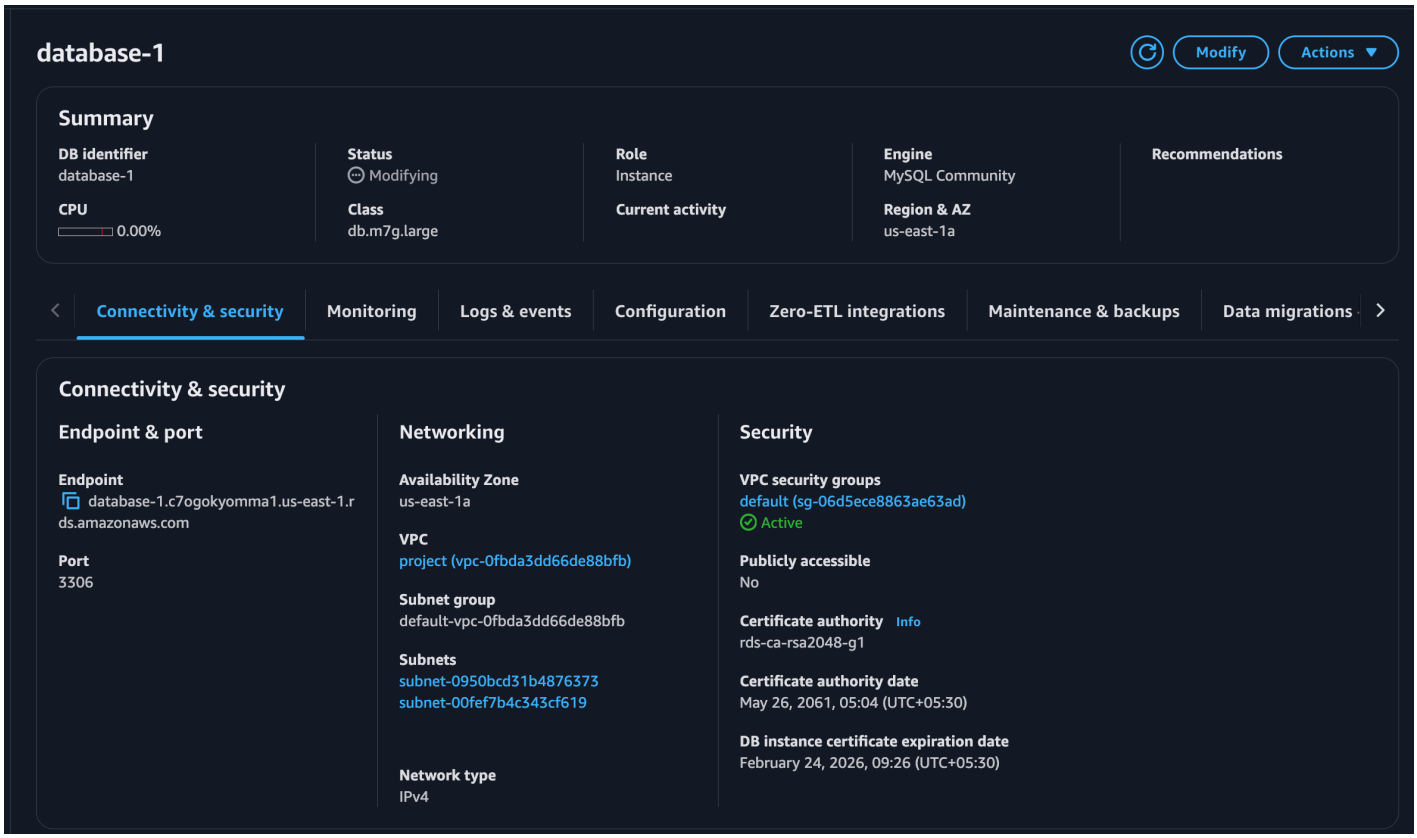
To ensure **data reliability, high availability, and security**, an **Amazon RDS (MySQL) instance** is deployed in the **private subnet** with **Multi-AZ** replication.

Create an Amazon RDS (MySQL) Instance:

- Navigate to **Amazon RDS** in the AWS Console.
- Select **MySQL** as the database engine.
- Choose **"Production" template** for best performance settings.
- Deploy the RDS instance **inside the private subnet** of the **VPC** (to prevent direct internet access).
- Set up a **username and password** for database authentication.
- Configure the **database port (3306, default for MySQL)**.

Enable Multi-AZ Deployment for High Availability:

- Select **Multi-AZ Deployment** to ensure automatic failover in case of an availability zone failure.
- AWS will create a **standby replica** in a different AZ for redundancy.



```
root@ip-10-0-0-23: /home/ut
     Active: active (running) since Tue 2025-02-25 17:58:53 UTC; 15min ago
    Process: 4233 ExecStartPre=/usr/share/mysql/mysql-systemd-start pre (code=exited, status=0/SUCCESS)
   Main PID: 4243 (mysqld)
     Status: "Server is operational"
      Tasks: 37 (limit: 1130)
     Memory: 383.2M (peak: 400.1M)
        CPU: 4.381s
     CGroup: /system.slice/mysql.service
             └─4243 /usr/sbin/mysqld

Feb 25 17:58:52 ip-10-0-0-23 systemd[1]: Starting mysql.service - MySQL Community Server...
Feb 25 17:58:53 ip-10-0-0-23 systemd[1]: Started mysql.service - MySQL Community Server.
root@ip-10-0-0-23:/home/ubuntu# mysql -h database.clmk6gm8aug9.us-east-1.rds.amazonaws.com -u dhanush -p
Enter password:
ERROR 2003 (HY000): Can't connect to MySQL server on 'database.clmk6gm8aug9.us-east-1.rds.amazonaws.com:3306' (110)
root@ip-10-0-0-23:/home/ubuntu# mysql -h database.clmk6gm8aug9.us-east-1.rds.amazonaws.com -u dhanush -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 51
Server version: 8.0.40 Source distribution

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql>
```

```
root@ip-10-0-0-47: /home/ut
root@ip-10-0-0-47:/home/ubuntu# mysql -h database.clmk6gm8aug9.us-east-1.rds.amazonaws.com -u dhanush -p
Enter password:
ERROR 1045 (28000): Access denied for user 'dhanush'@'10.0.0.47' (using password: YES)
root@ip-10-0-0-47:/home/ubuntu# mysql -h database.clmk6gm8aug9.us-east-1.rds.amazonaws.com -u dhanush -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 54
Server version: 8.0.40 Source distribution

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| dhanush            |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)

mysql>
```

CONCLUSION

This 3-tier architecture provides a scalable and resilient cloud infrastructure. Features like Auto Scaling, RDS Multi-AZ, and ALB ensure high availability and fault tolerance. Security measures such as private subnets, security groups, and IAM roles safeguard data and resources. This architecture lays a strong foundation for building secure, scalable, and reliable applications on AWS.