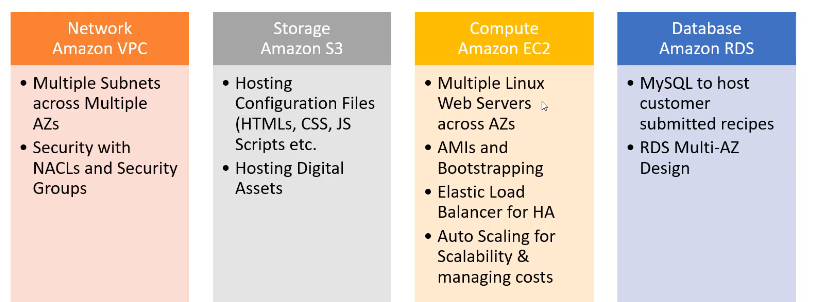
Fully Server-Based AWS Three-Tier Architecture with Auto Scaling

## **Introduction:**

🡺 This project demonstrates the use of all the AWS services I've learned so far.  


**Services Used:**

* **EC2 (Elastic Compute Cloud)** – Virtual servers used for hosting the application. A **user data script** is configured to automate server-side setup each time an instance is launched, ensuring seamless deployment during auto-scaling.
* **VPC (Virtual Private Cloud)** – Provides a logically isolated network for secure communication between application components, including public and private subnets.
* **S3 (Simple Storage Service)** – Stores static website files and application code, allowing EC2 instances to fetch necessary resources.
* **IAM (Identity and Access Management)** – Manages permissions and security policies, enabling EC2 instances to securely access S3 buckets for retrieving website assets.
* **Session Manager** – Provides secure, agent-based access to EC2 instances without requiring an SSH key or bastion host, simplifying troubleshooting and administration.
* **Auto Scaling** – Automatically adjusts the number of EC2 instances based on traffic demand, ensuring high availability and cost efficiency.
* **CloudWatch** – Monitors EC2 instances, logs performance metrics, and sets up alarms to trigger scaling actions or notify administrators of potential issues.

## **Low-Level Design (LLD) Overview & Services Used:**

**1. Purpose of the Architecture**

This AWS-based **three-tier architecture** is designed to host a web application with **auto-scaling** for cost efficiency. The architecture ensures high availability by leveraging **multiple Availability Zones (AZs)** while maintaining security at different layers.

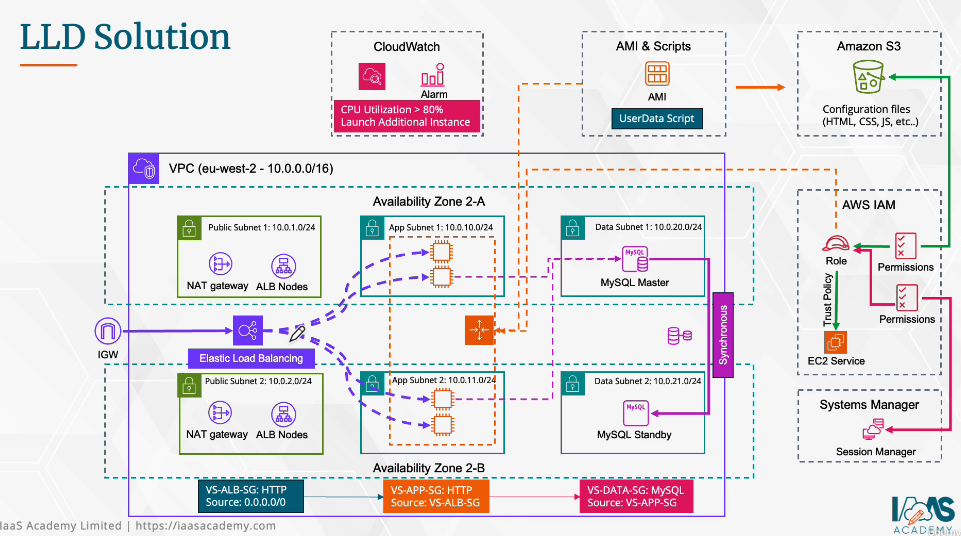
**2. Architecture Breakdown**

**Networking & Load Balancing**

* The architecture operates within a **Virtual Private Cloud (VPC)**, ensuring network isolation and security.
* Public subnets house **NAT Gateways** and **Application Load Balancer (ALB) nodes** to distribute traffic efficiently.
* The **Elastic Load Balancer (ELB)** acts as a single entry point, intelligently routing requests to available application servers.

**Compute & Scaling**

* **EC2 instances** in the application tier handle web traffic and execute dynamic content processing.
* A **user data script** automates server configuration on launch, ensuring seamless deployment when instances scale.
* **Auto Scaling** dynamically adjusts the number of EC2 instances based on real-time demand, optimizing cost and performance.



**Database Layer (High Availability & Failover)**

* The database layer follows a **Master-Standby architecture** within private subnets.
* If the **primary database** fails, ownership is transferred to the standby instance in the opposite region.
* A new standby database is automatically created in the previously failed region to maintain redundancy.

**Security & Access Control**

* **IAM roles and policies** control access between EC2 and S3 for secure file retrieval.
* **Session Manager** provides secure, SSH-free access to EC2 instances for administration.
* The **Application Tier** enforces strict security policies, validating incoming requests before processing.

**Monitoring & Alerts**

* **CloudWatch** monitors system health, tracks performance metrics, and triggers auto-scaling when CPU utilization exceeds 80%.

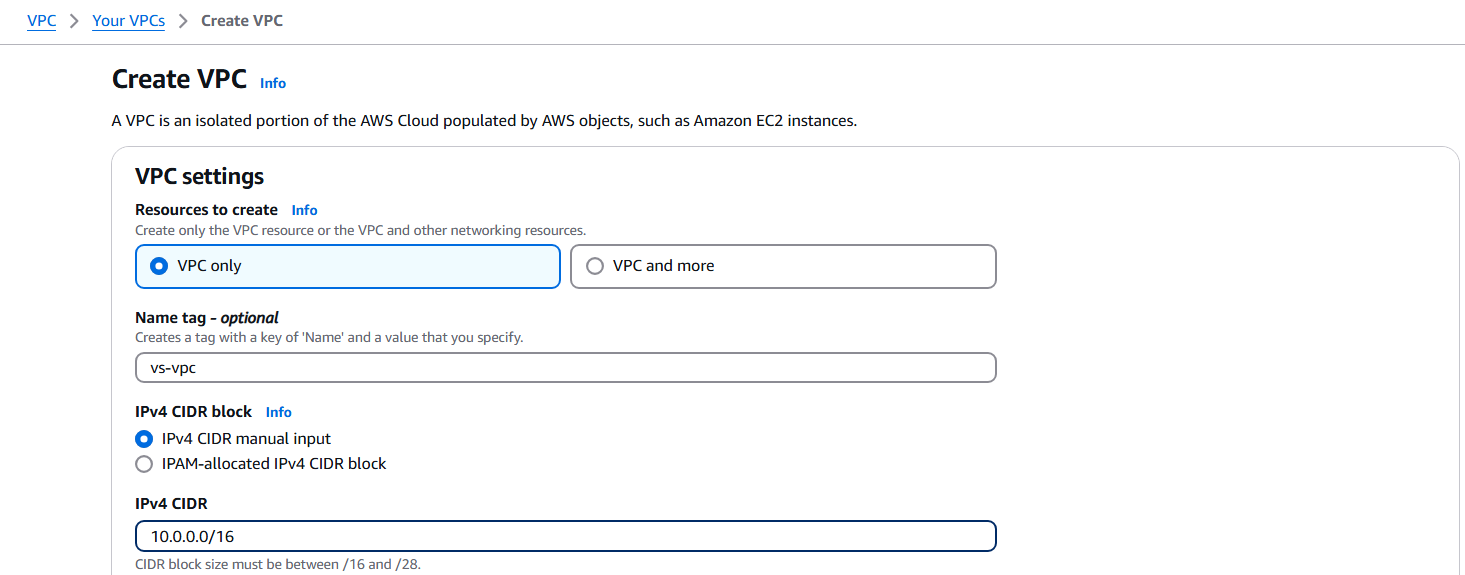
**3. Request Flow (Step-by-Step Workflow)**

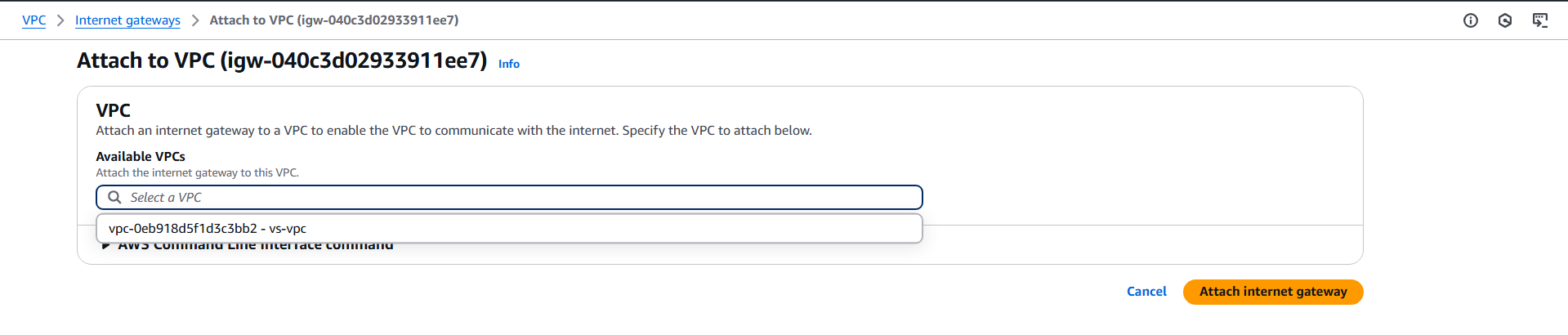
1. User accesses the web application through a public endpoint.
2. The Elastic Load Balancer (ELB) distributes traffic to EC2 instances in different AZs.
3. EC2 instances fetch static content from S3 and process dynamic content.
4. The processed request is forwarded to the database layer for data retrieval/storage.
5. In case of database failure, ownership shifts to the standby instance in another region.
6. The response is sent back to the user through the load balancer.

This ensures a **resilient, scalable, and secure** web hosting solution!

### Project-Work-Line:

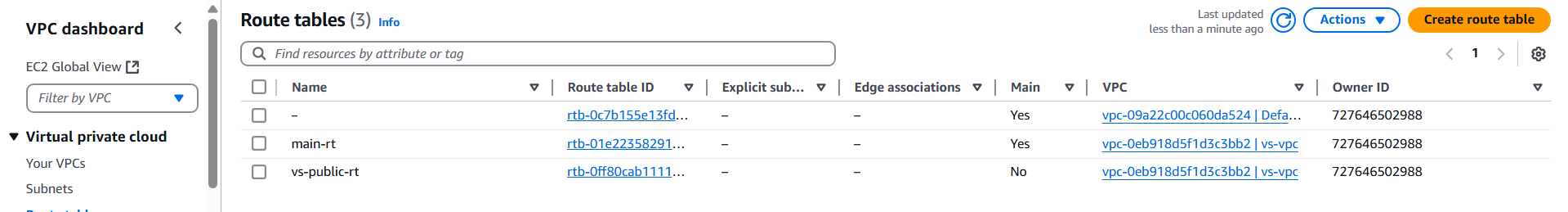
### Setting Up VPC:





🡺Create main route table for private subnets

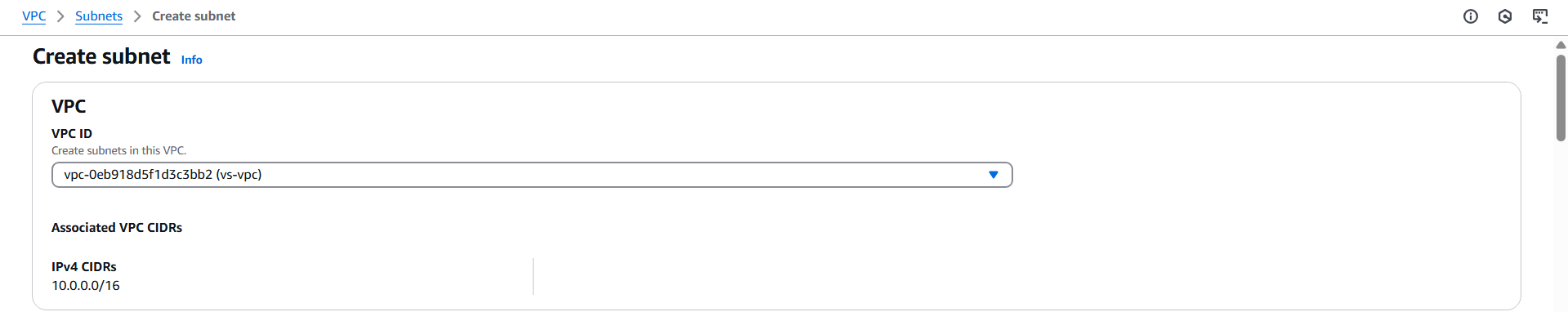
🡺And public route table for public subnets

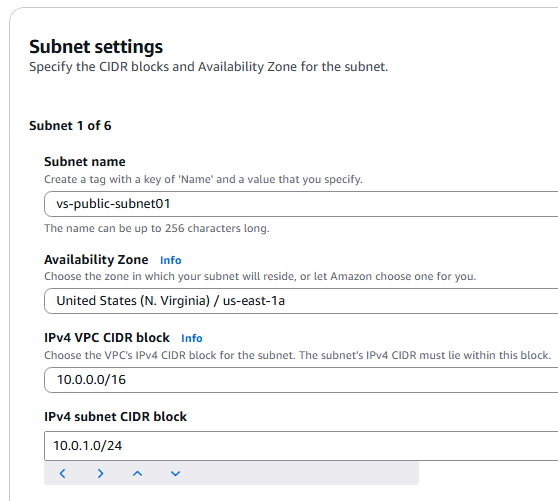
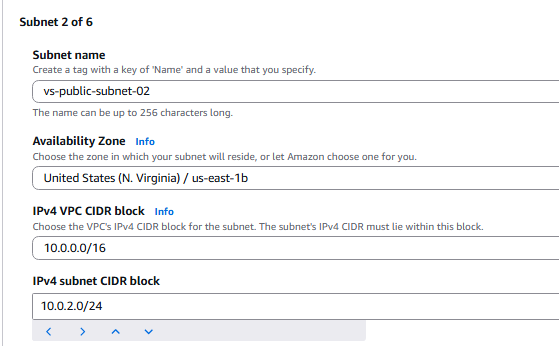


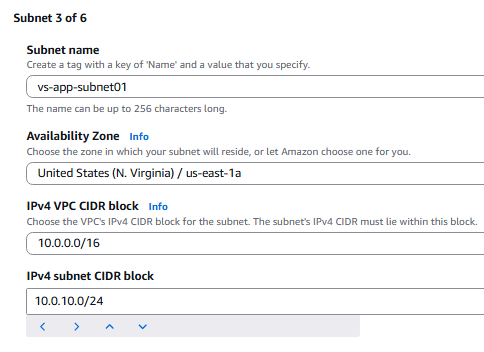
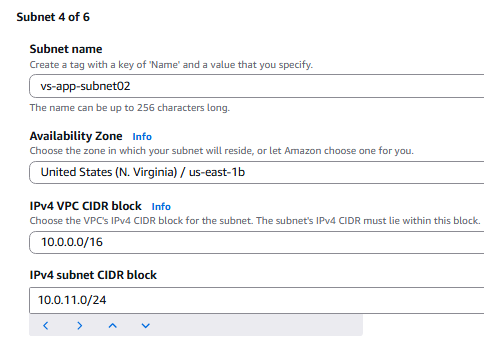
🡺Create six subnets for a 3-tier architecture across two Availability Zones (AZs) in the **us-east-1** region:

* **In us-east-1a**:
  + **Web Subnet** (Public)
  + **App Subnet** (Private)
  + **Database Subnet** (Private)
* **In us-east-1b**:
  + **Web Subnet** (Public)
  + **App Subnet** (Private)
  + **Database Subnet** (Private)

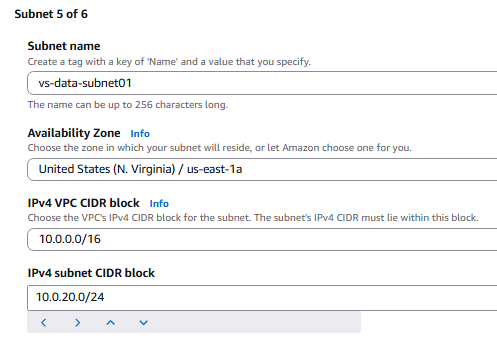
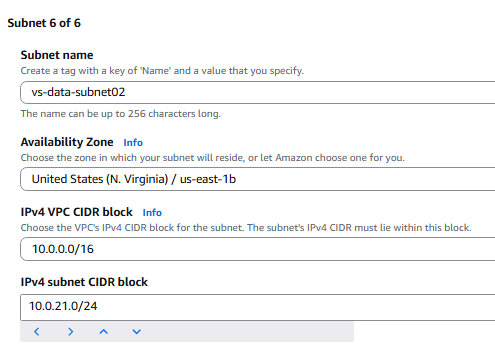
Each tier is deployed in both AZs to ensure high availability and fault tolerance.



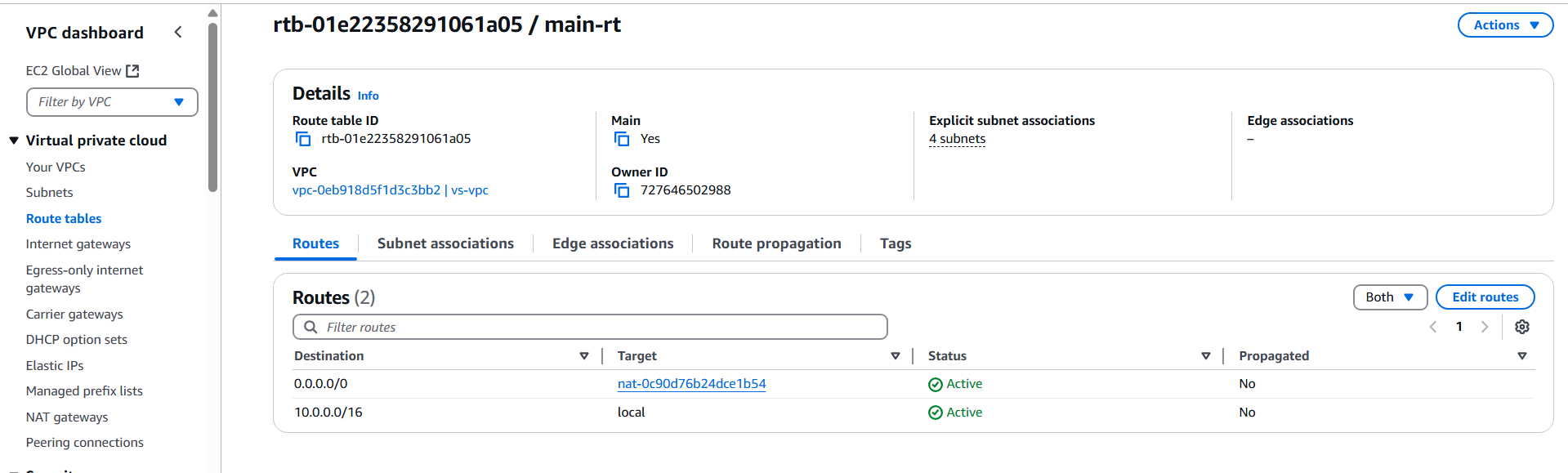


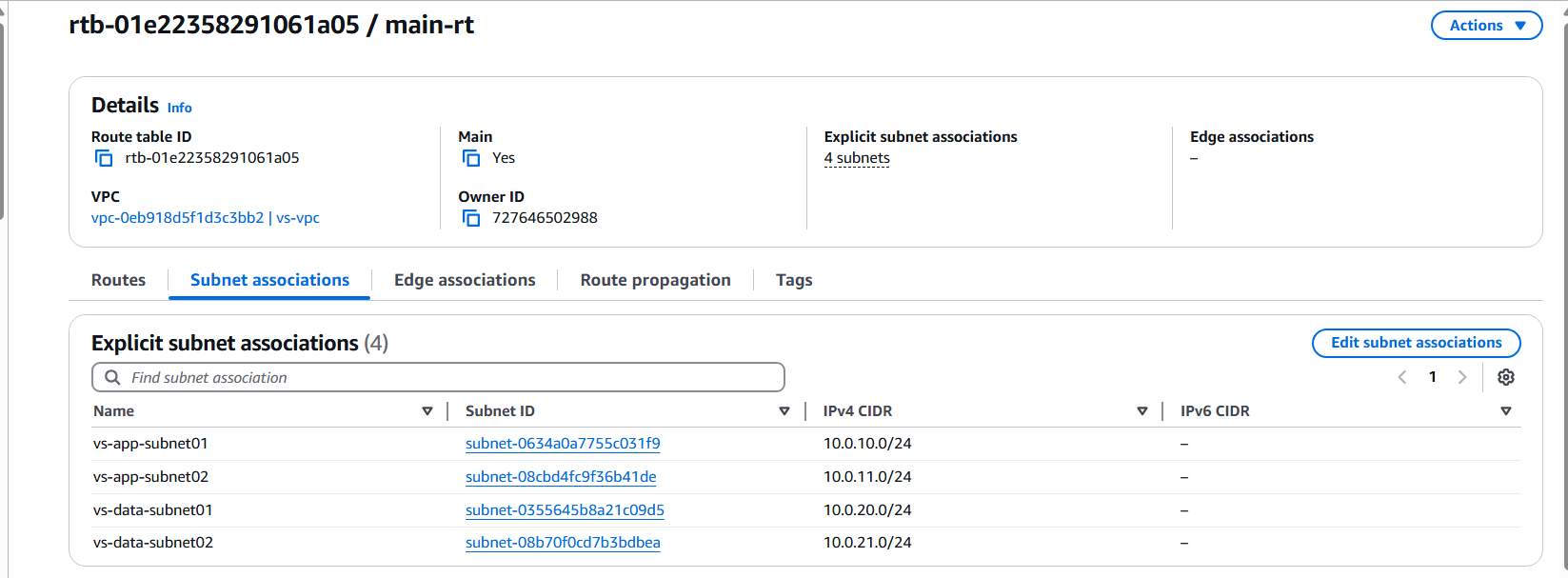
 

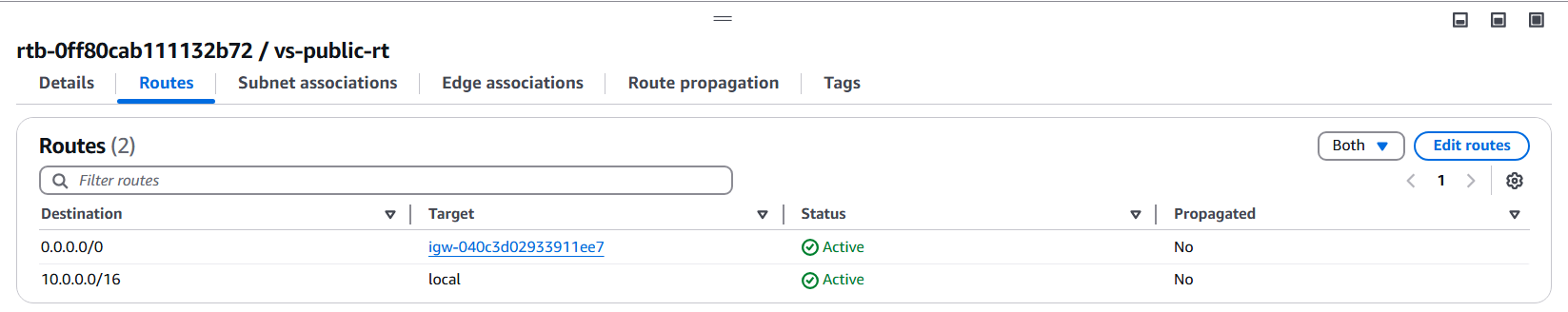
**Route Table Association Guide**

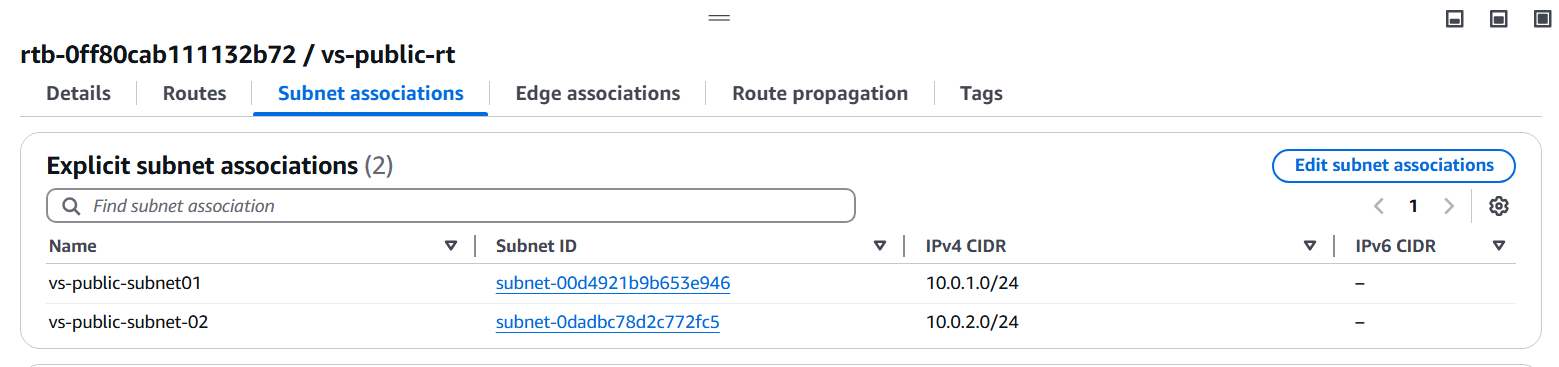
1. **Main Route Table (Private Subnets with NAT in us-east-1a)**
   * The **main route table** should be associated with **private subnets** in both **us-east-1a and us-east-1b**.
   * A **NAT Gateway** is deployed in **us-east-1a (Public Subnet)** to allow private instances to access the internet securely.
   * The private subnets use this **NAT Gateway** as the default route (0.0.0.0/0) for outbound internet access.
2. **Public Route Table (For Public Subnets)**
   * A **separate public route table** should be created and associated with the **public subnets** in **us-east-1a and us-east-1b**.
   * This public route table should have a **default route (0.0.0.0/0) pointing to the Internet Gateway (IGW)**, allowing public instances direct internet access.

This setup ensures:  
✅ **Private subnets remain secure** (outbound traffic via NAT, no direct internet access).  
✅ **Public subnets have direct internet access** through IGW.  
✅ **High availability** with resources spread across two AZs.





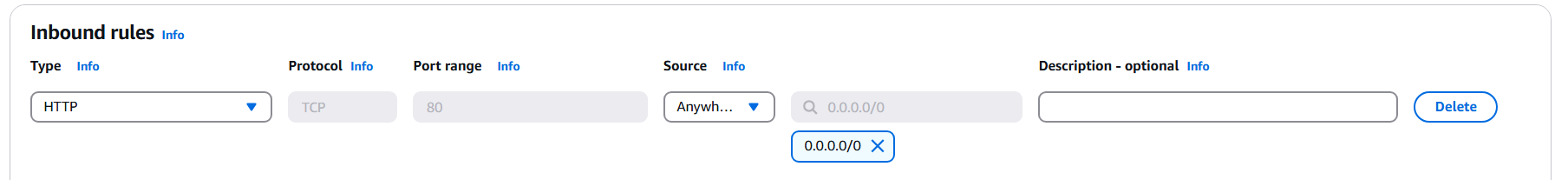


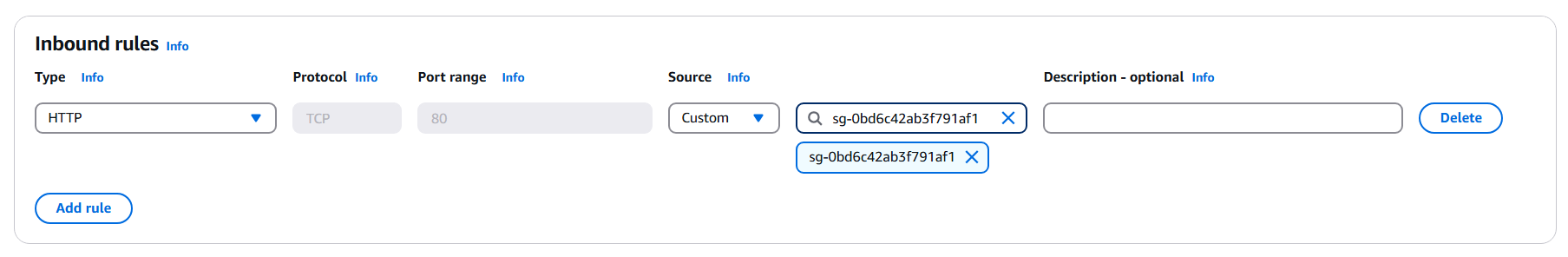


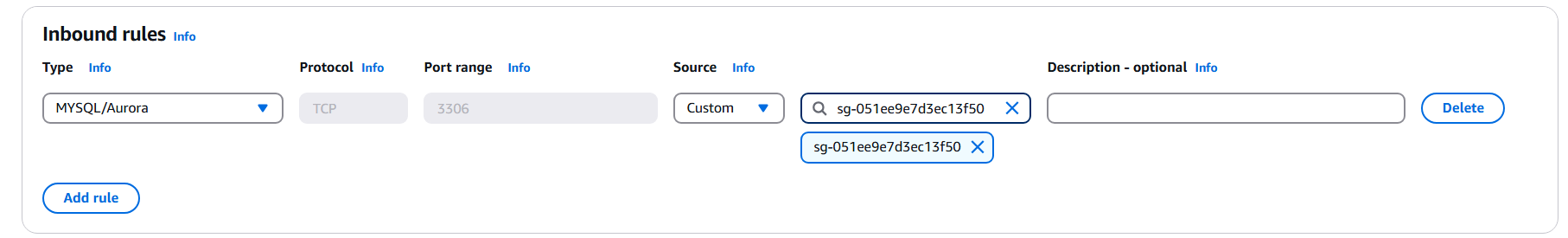
**Security Group Configuration for 3-Tier Architecture (Only HTTP & MySQL)**

1. **Web Security Group (SG-Web)**
   * Allows **Inbound:**
     + **HTTP (port 80)** from **anywhere (0.0.0.0/0)**.
   * Allows **Outbound:**
     + **HTTP (port 80)** to **SG-App** (Application Security Group).
2. **App Security Group (SG-App)**
   * Allows **Inbound:**
     + **HTTP (port 80)** only from **SG-Web** (Web Security Group).
   * Allows **Outbound:**
     + **MySQL (port 3306)** to **SG-Database** (Database Security Group).
3. **Database Security Group (SG-Database)**
   * Allows **Inbound:**
     + **MySQL (port 3306)** only from **SG-App** (Application Security Group).
   * Allows **Outbound:**
     + Traffic as needed for database operations (e.g., backups, replication).

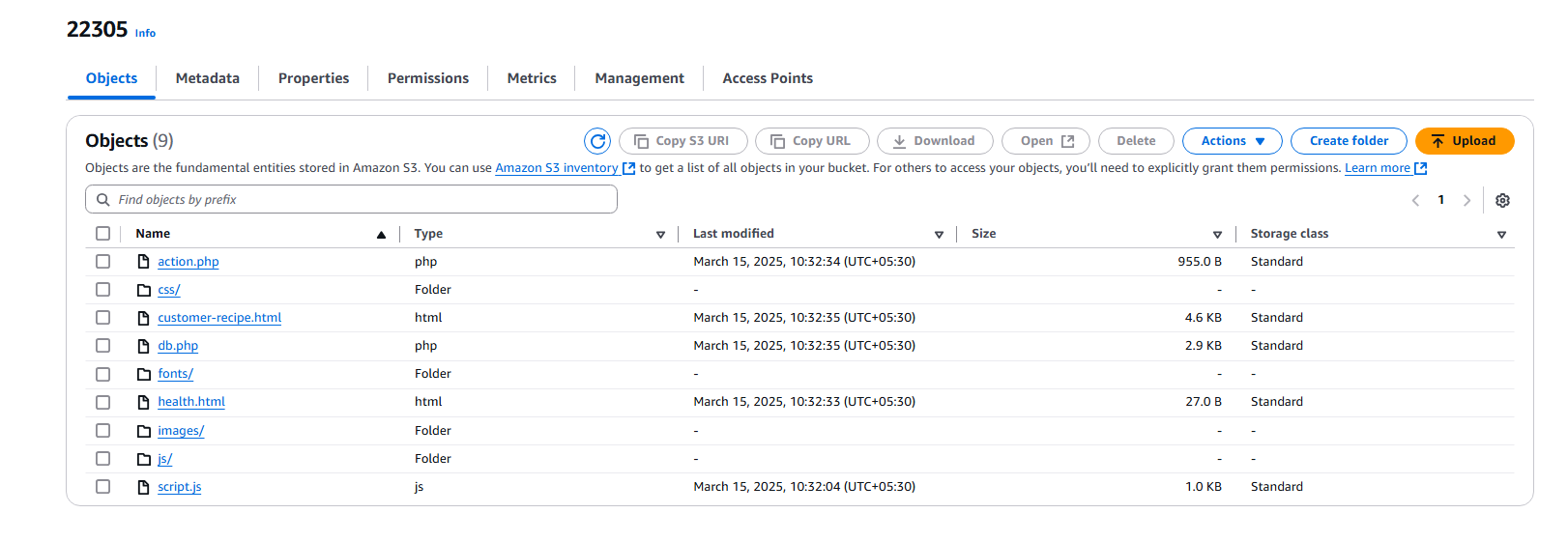
This setup ensures:  
✅ **Web servers accept only HTTP traffic from the internet**.  
✅ **App servers only communicate with Web & Database layers**.  
✅ **Database remains private, accessible only from the App layer**.







### Setting Up S3 Bucket for s3 hosting:



### IAM Roles:

**S3 Permissions:**

* s3:GetObject → Read objects from S3.
* s3:ListBucket → List objects in S3.

**SSM Permissions:**

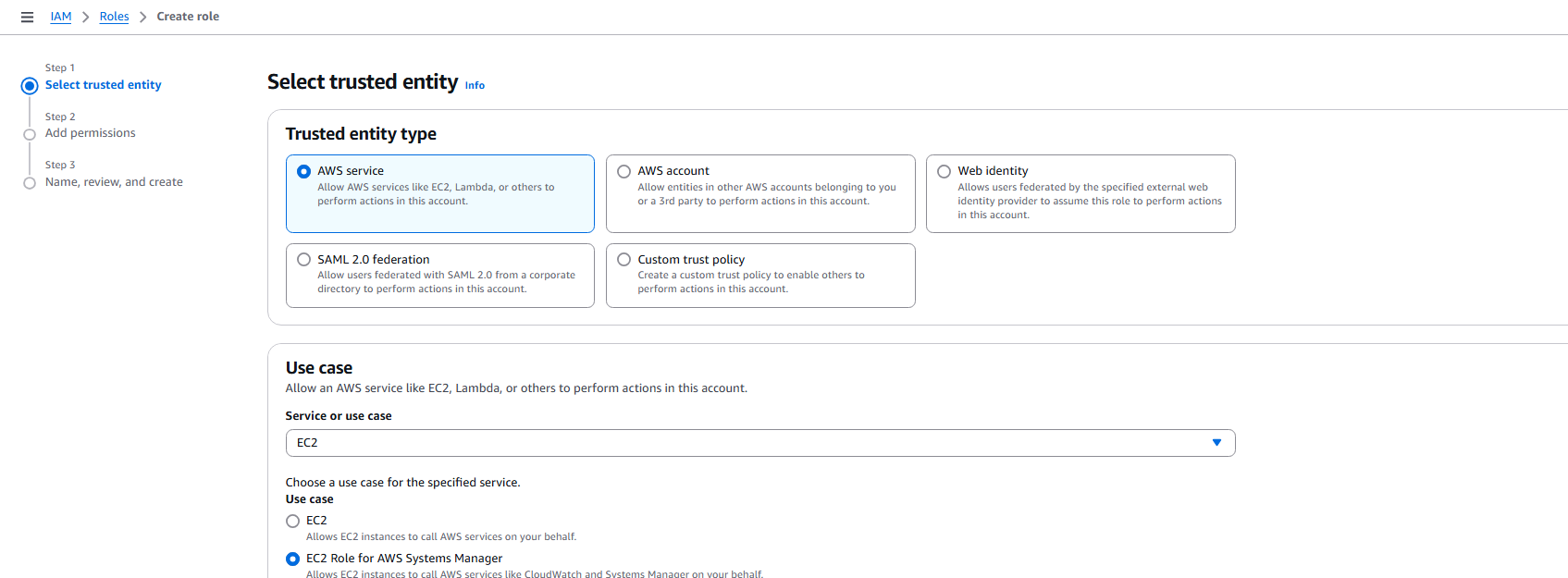
* Attach **AmazonSSMManagedInstanceCore** policy to enable:
  + AWS Systems Manager Session Manager (remote access without SSH).
  + SSM Agent for automation, patching, and monitoring.

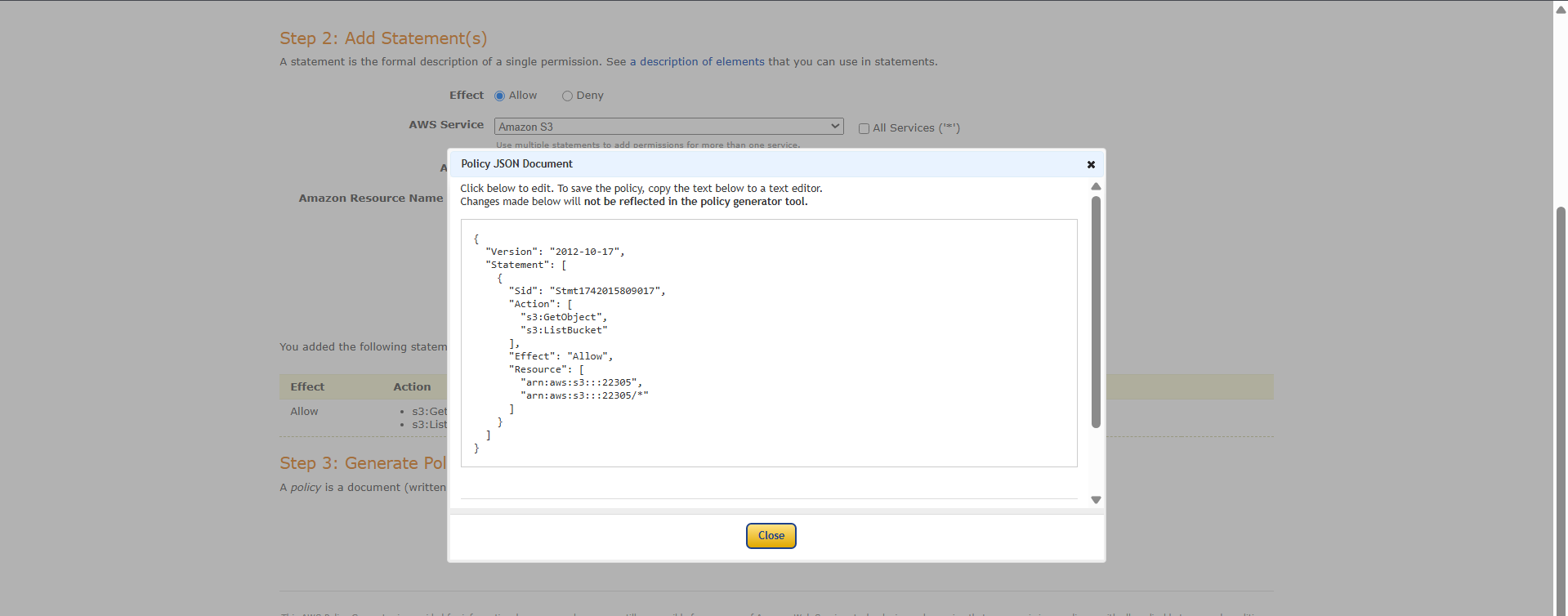
**IAM Role Setup:**

1. Create an **IAM role** for EC2.
2. Attach:
   * **Custom S3 Policy** (s3:GetObject, s3:ListBucket for a specific bucket).
   * **AmazonSSMManagedInstanceCore** for SSM access.
3. Assign the IAM role to the **EC2 Template**.

**Purpose:**

* Secure **S3 access** without credentials.
* Manage EC2 using **SSM Session Manager** instead of SSH.





### Setting Up DataBase:

**Setting Up Database (MySQL) in AWS RDS**

**1. Create a Subnet Group**

* Go to **RDS Console** → **Subnet Groups** → **Create DB Subnet Group**.
* Select **at least two subnets** in different AZs for high availability.

**2. Create MySQL Database**

* Navigate to **RDS Console** → **Create Database**.
* **Engine:** Choose **MySQL**.
* **Template:** Select **Dev/Test** for a non-production environment.

**3. Configure Database Settings**

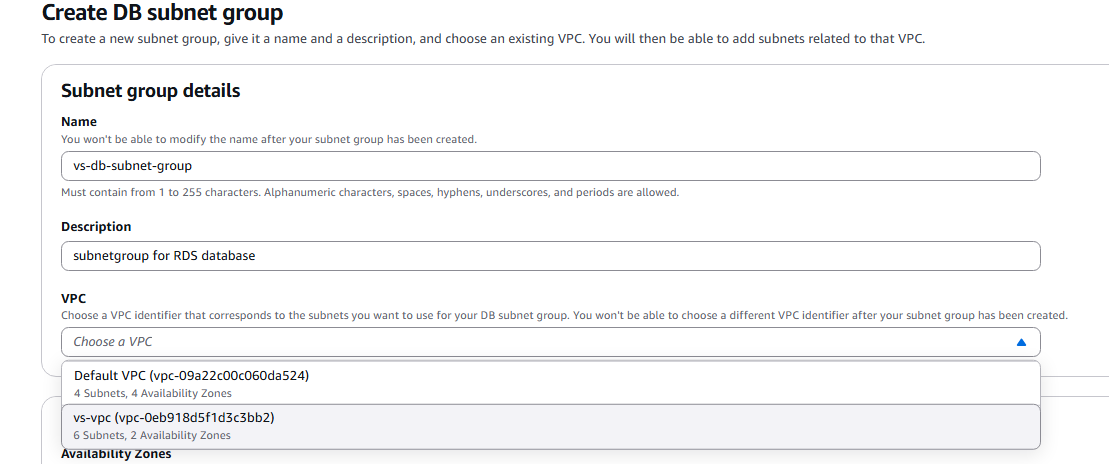
* **DB Instance Class:** Choose based on workload (e.g., db.t3.micro for testing).
* **Storage:** Select size and enable **Auto Scaling** if needed.
* **VPC & Subnet Group:** Choose the **previously created subnet group**.
* **Public Access:** Set to **No** (Private Database).
* **VPC Security Group:** Allow inbound **3306** access from the application layer.

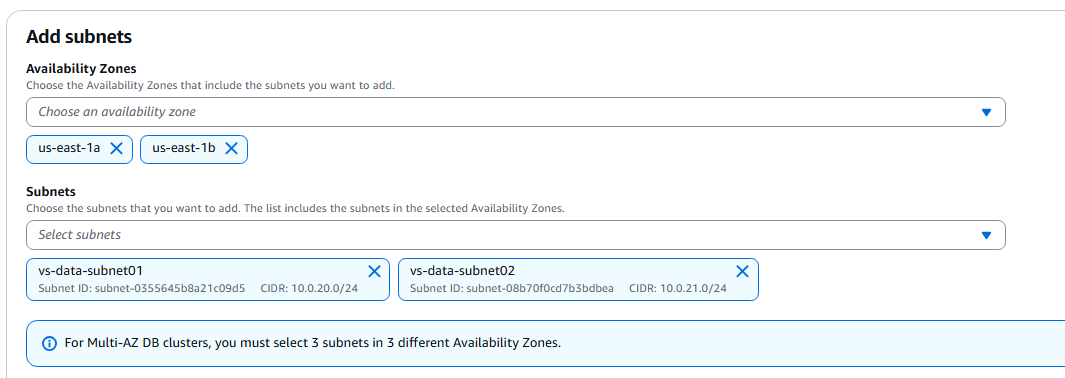
**4. Authentication & Connectivity**

* **Username & Password:** Set master credentials.
* **Parameter Group:** Use default or create a custom one.
* **Backup & Maintenance:** Configure automated backups and monitoring as needed.

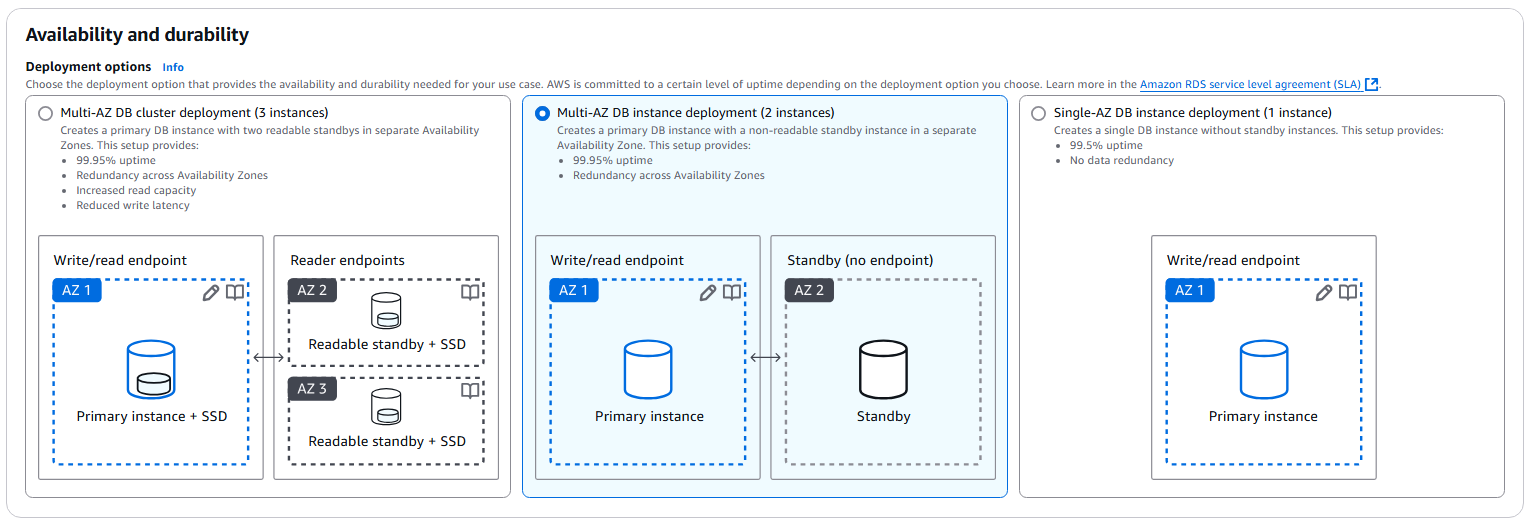
**5. Deploy & Connect**

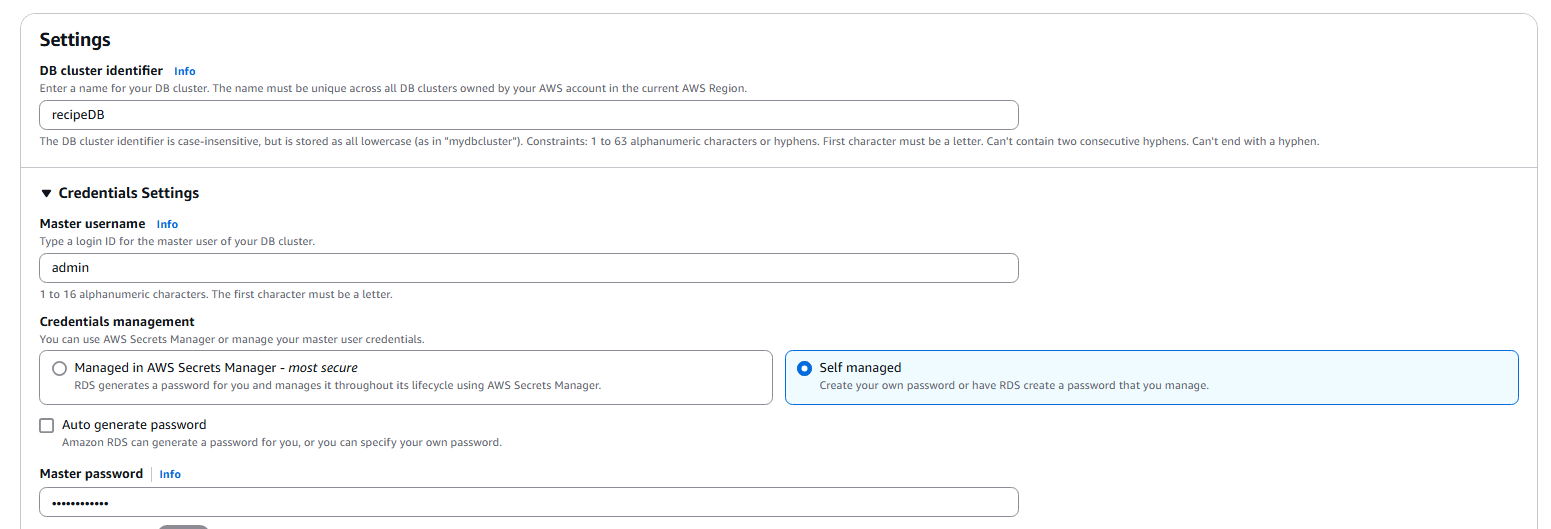
* Click **Create Database** and wait for the instance to be available.
* Use mysql -h <endpoint> -u <username> -p to connect from the app layer.

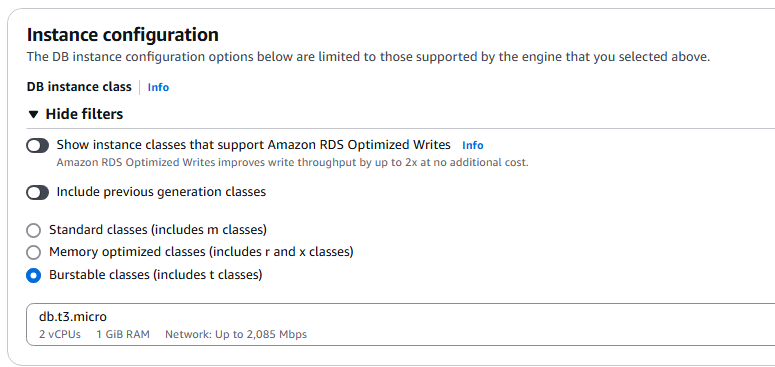


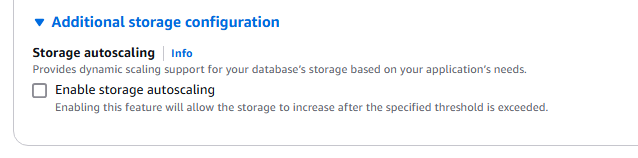


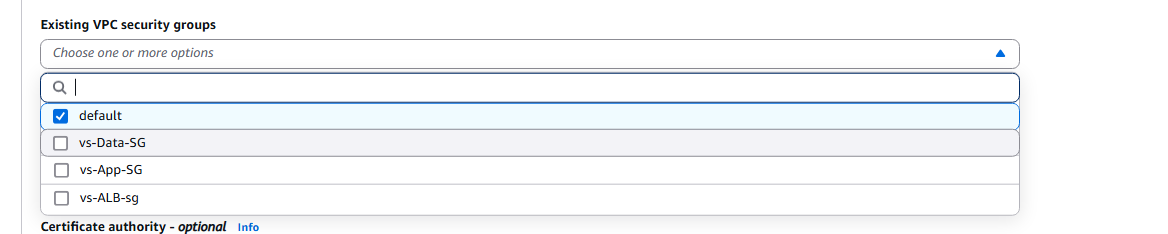
🡺Create Database>Choose MySql DataBase>Dev/Test:











### Amazon ELB:

🡺Application Load Balancer to distribute the connection request in a balanced way.

🡺First thing you should create the target group to define the targets, that can be combination of instances.

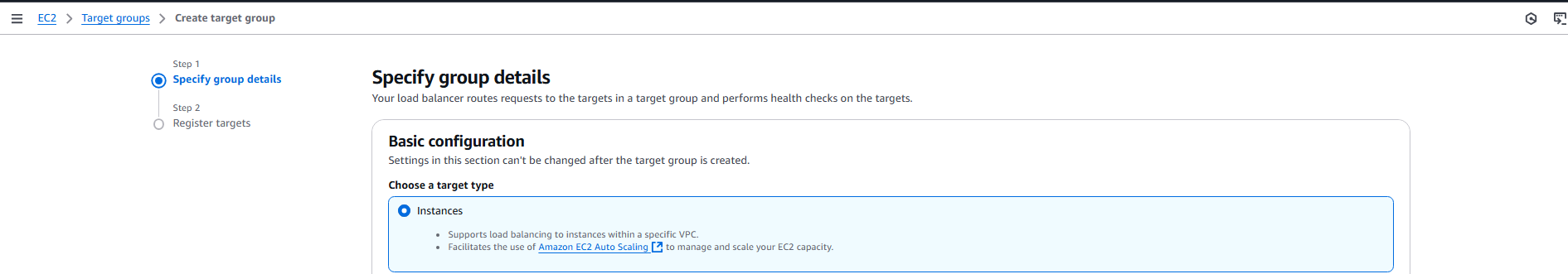
**Guide for Setting Up an Elastic Load Balancer (ELB)**

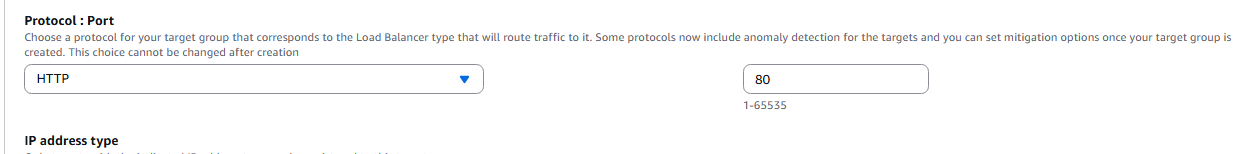
**1. Create a Target Group**

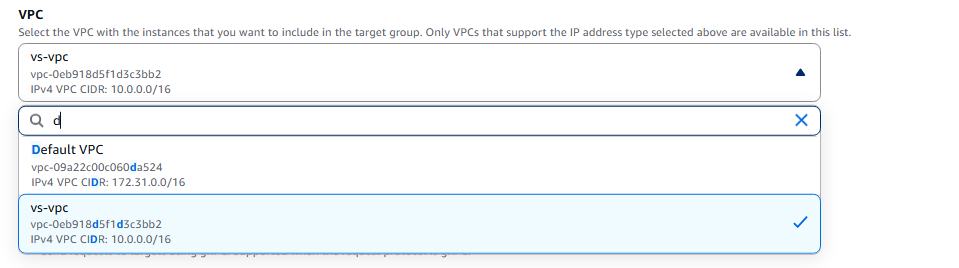
* Define a **Target Type** (Instance or IP).
* Choose the **protocol (HTTP/HTTPS)** and port (**80/443**).
* Attach instances or Auto Scaling Group to the **target group**.
* Set up **health checks** (e.g., HTTP:80 on / for web apps).

**2. Create an Elastic Load Balancer (ELB)**

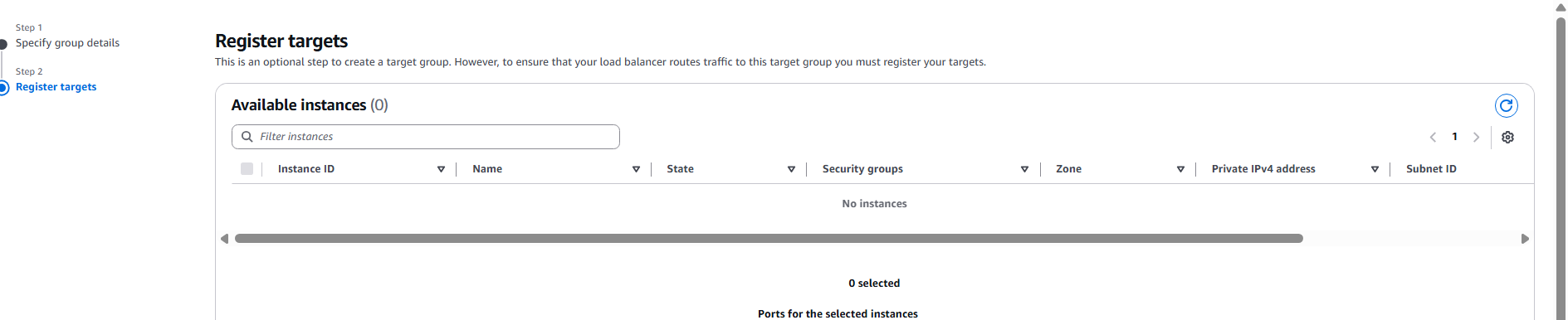
* Choose **Application Load Balancer (ALB)** for HTTP/HTTPS traffic.
* Select **at least two subnets** for high availability.
* Attach the previously created **target group**.
* Configure **listeners** (e.g., HTTP:80 → forward to target group).
* Enable **security settings** (SSL certificates for HTTPS if required).



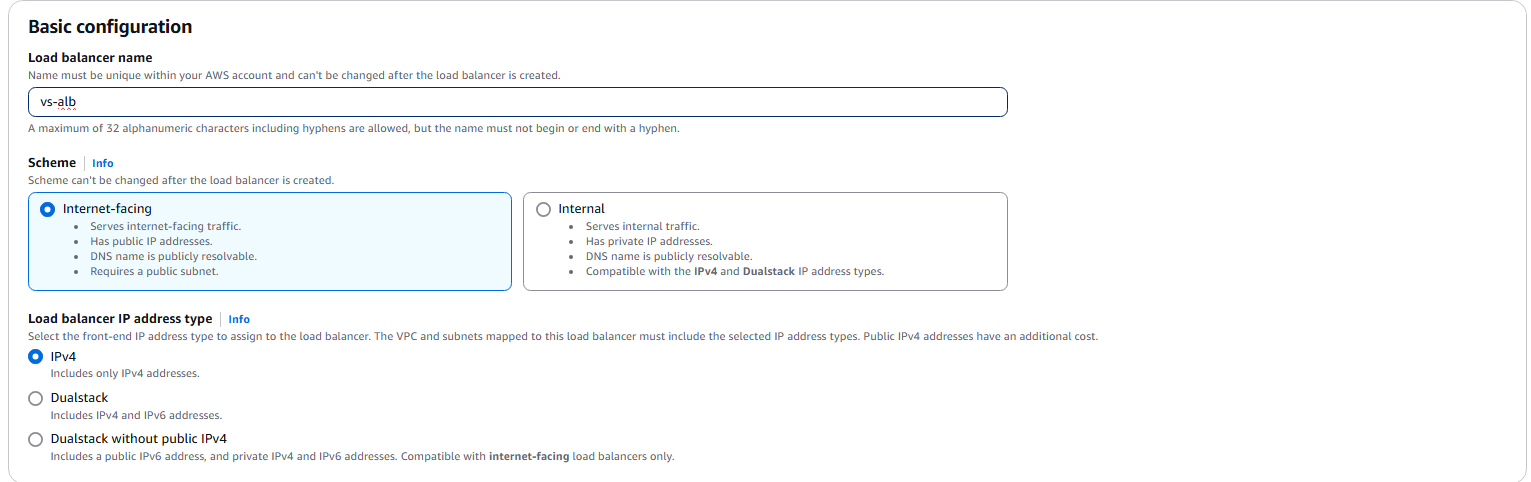


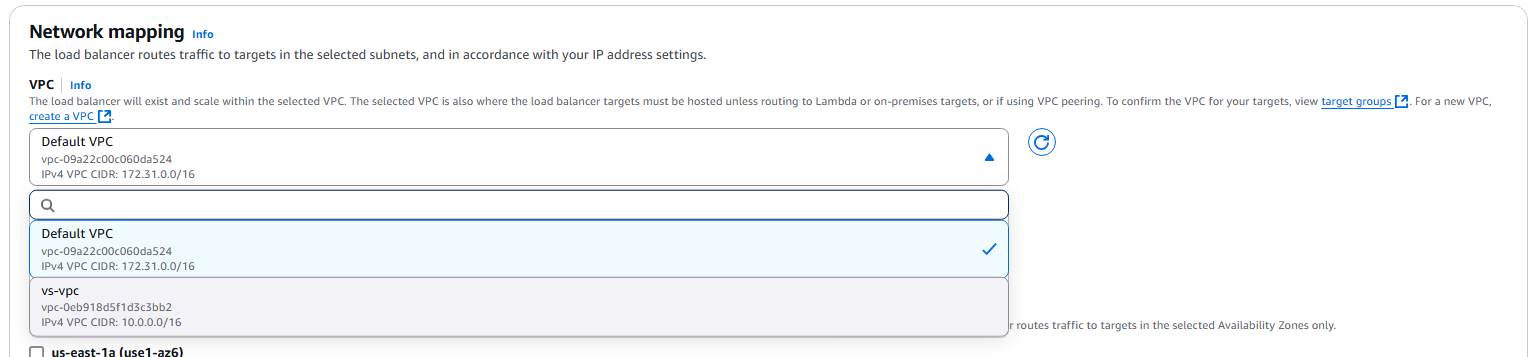


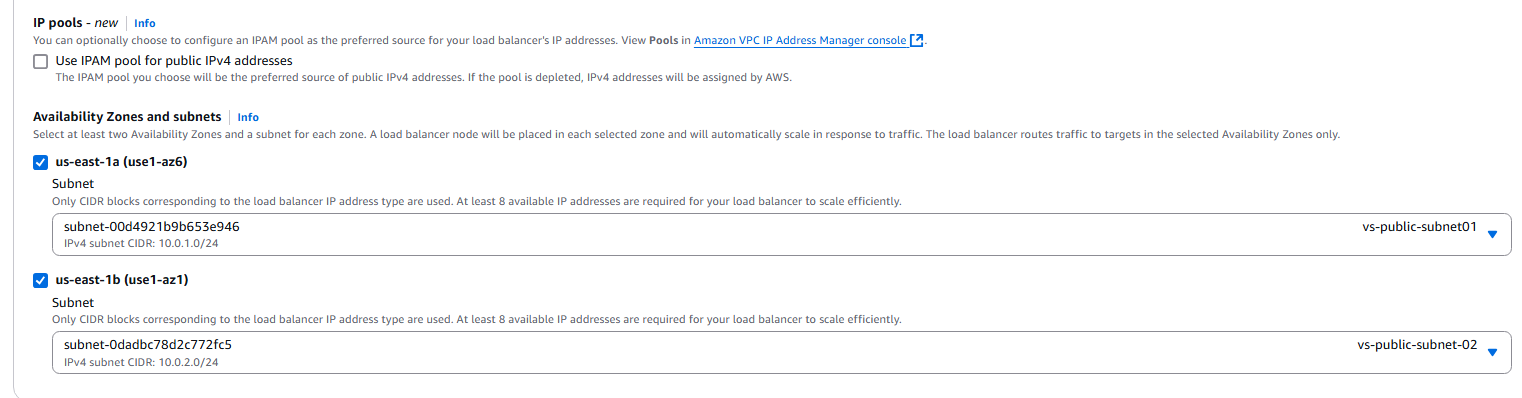


🡺We have no target, but target group is in place:

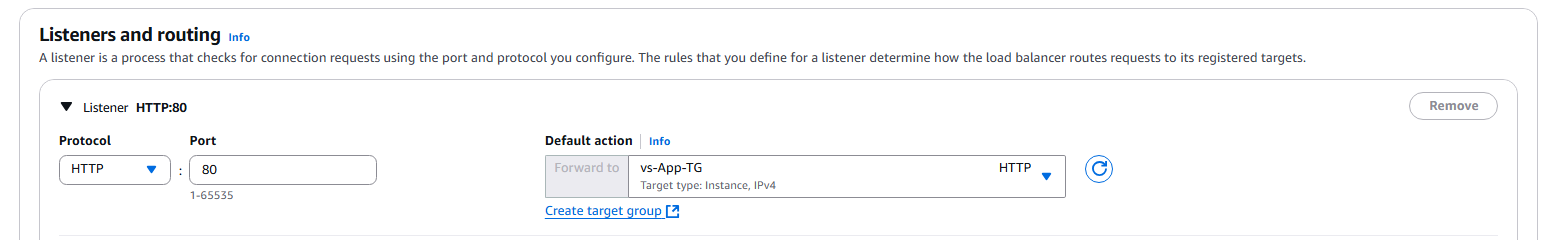
🡺Create Application Load Balancer:

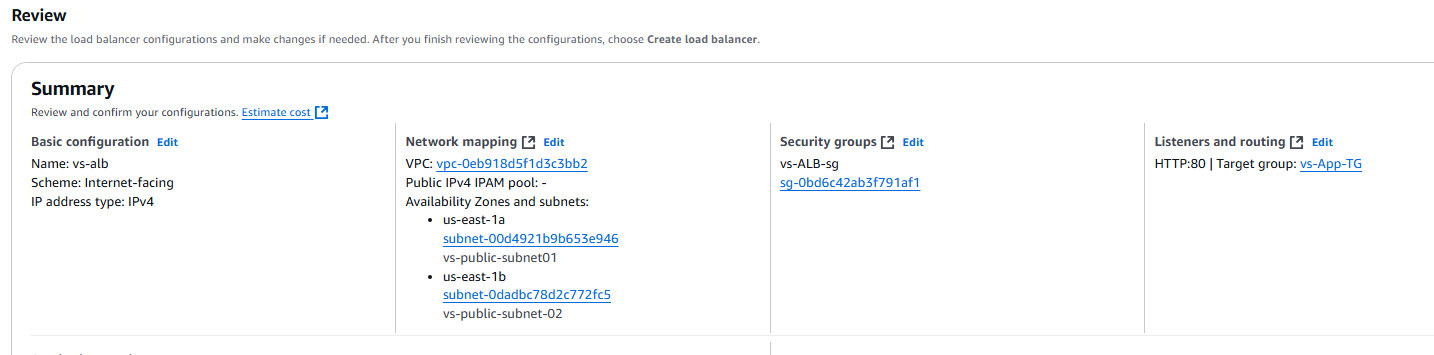




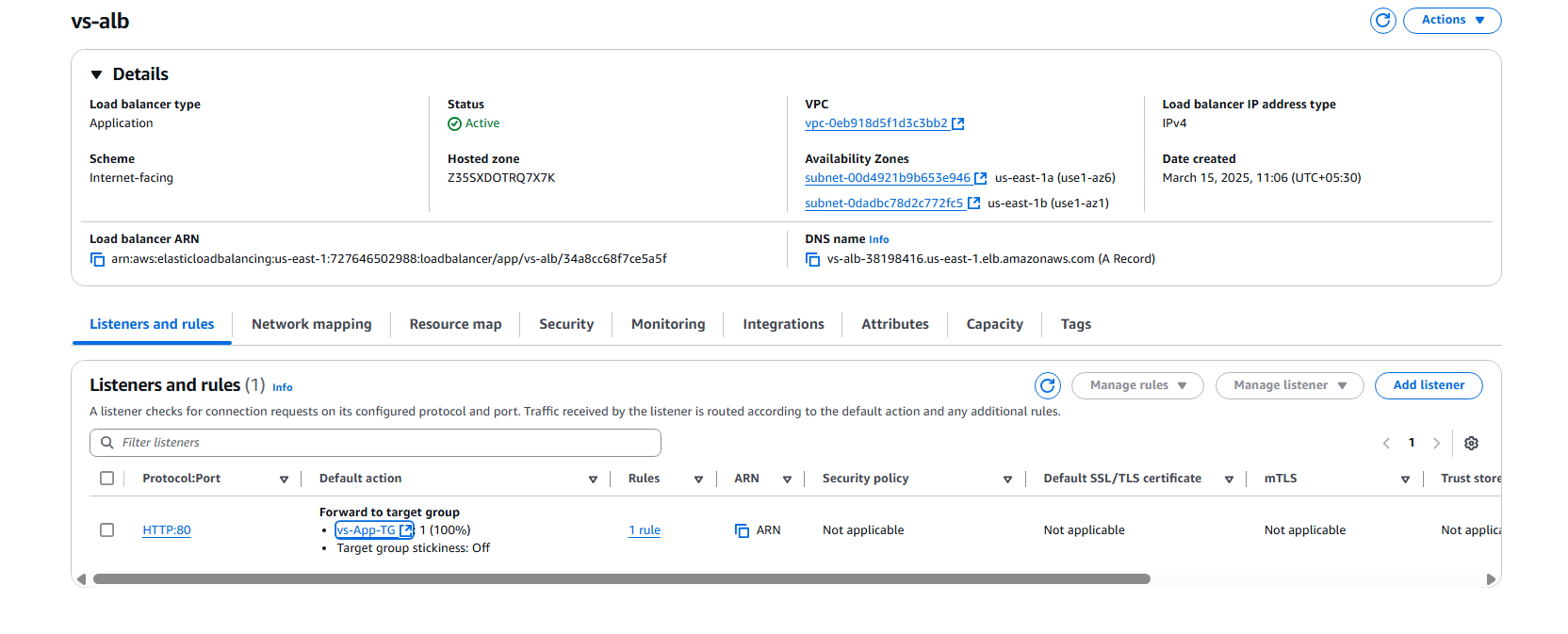








🡺ALB Created Successfully.

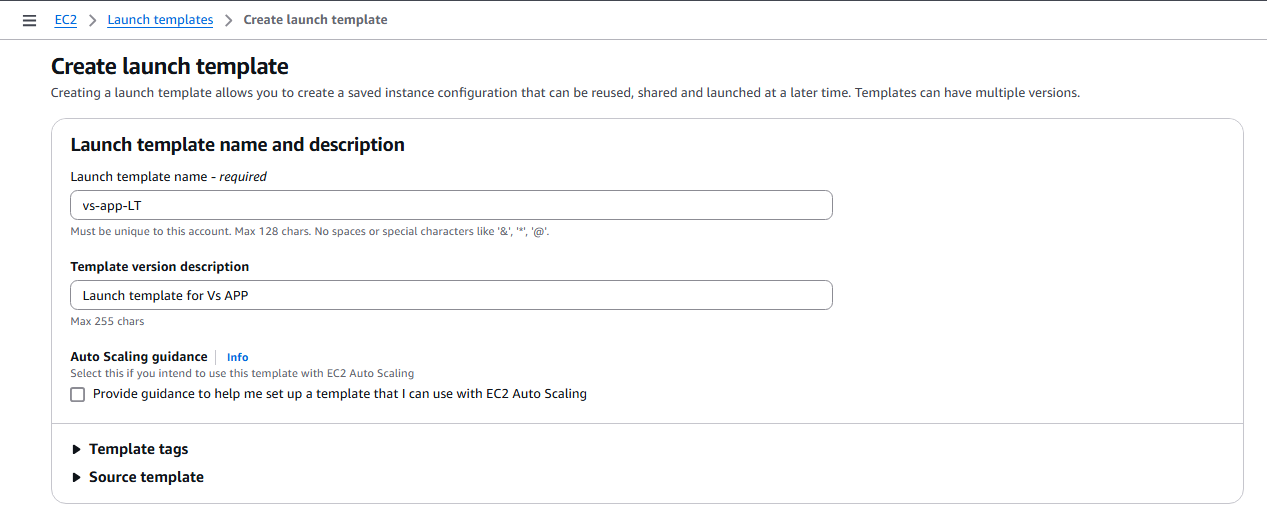


### Configuring Autoscaling Services:

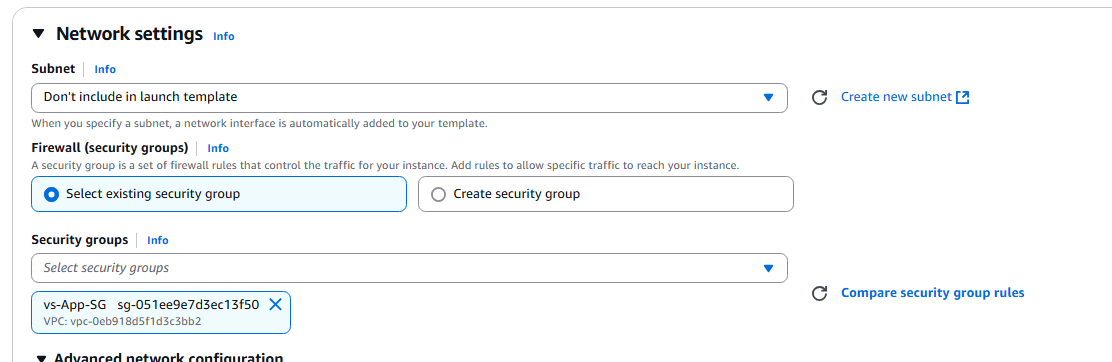
1.Create Launch Template

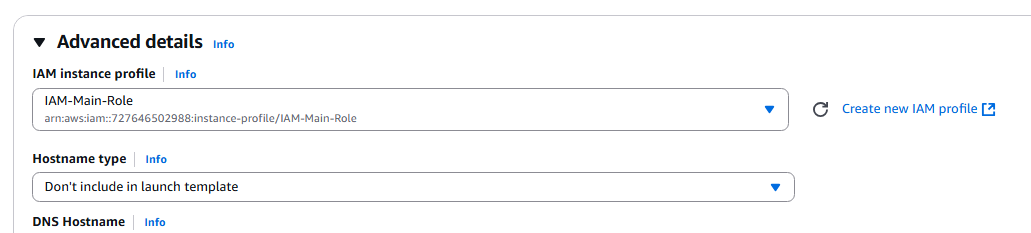
2.Actual configuration of the AutoScaling.

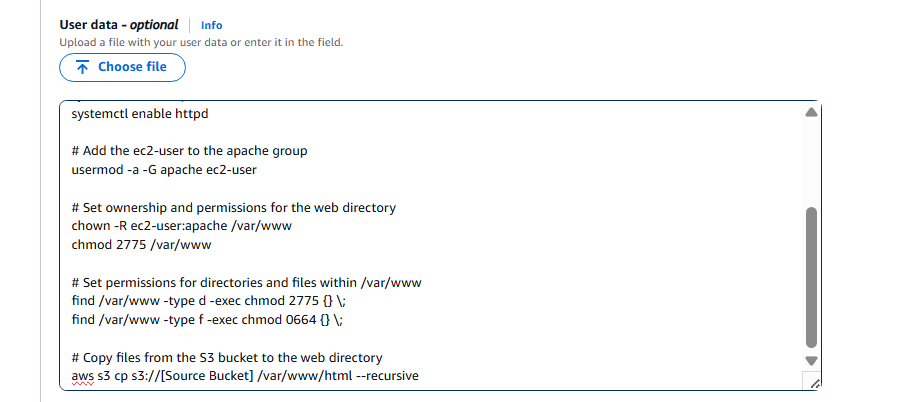
🡺Update the db.php

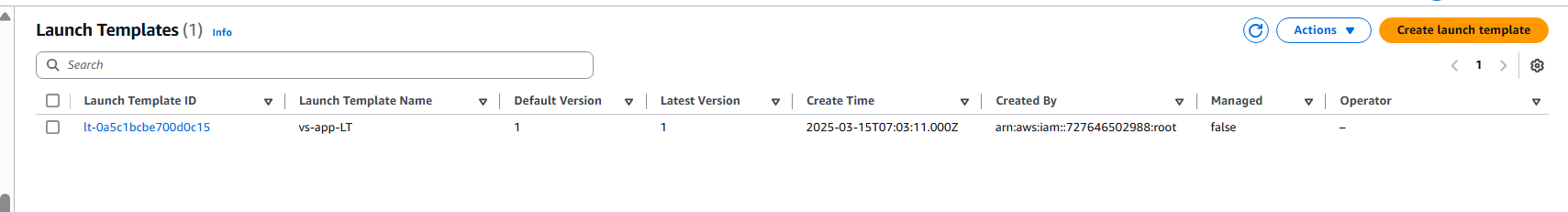


🡪Select your AMI (amazon ami)>t2.micro



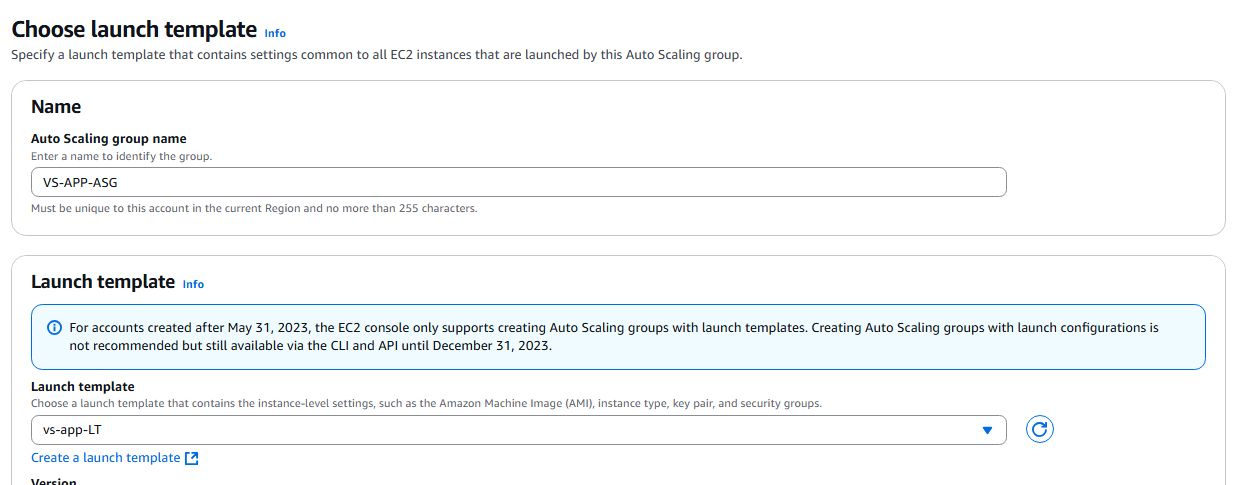




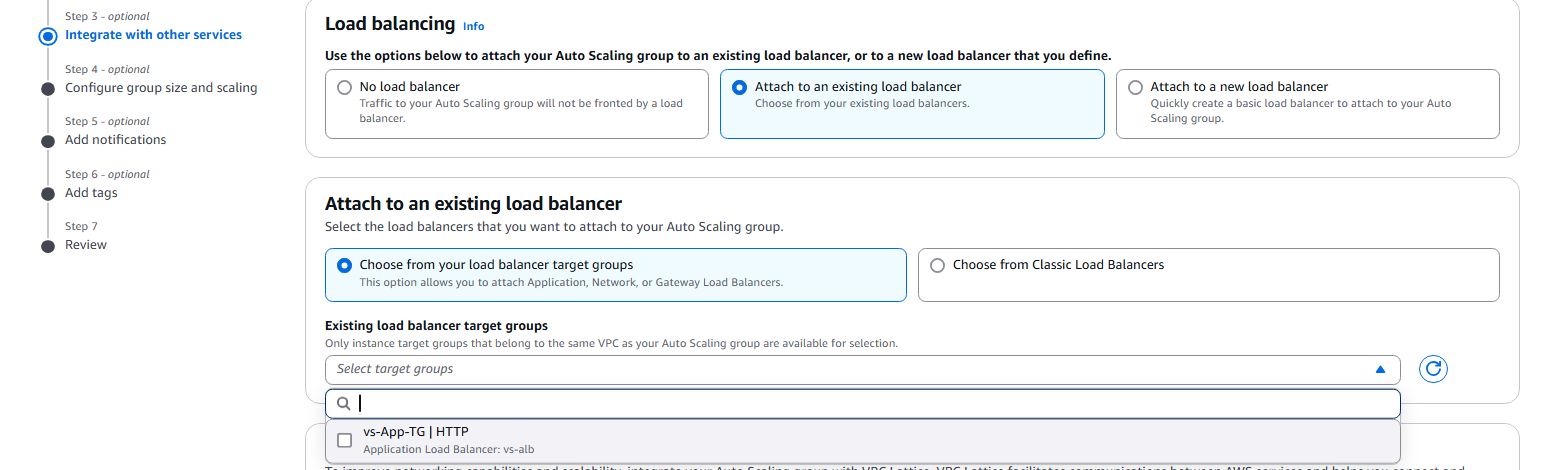


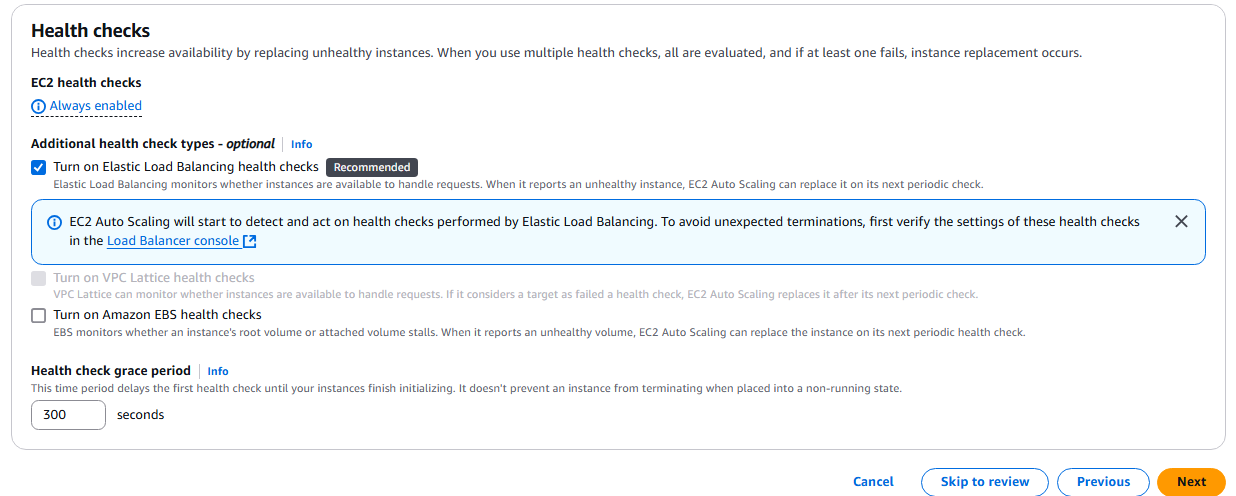
**5-Step Guide to Setting Up an Auto Scaling Group**

1. **Create a Launch Template**
   * Define the EC2 instance type, AMI, key pair, and security groups.
2. **Configure Auto Scaling Group**
   * Set the **minimum (2)** and **maximum (4)** instance limits.
   * Select the VPC and subnets for instance deployment.
3. **Define Scaling Policies**
   * Use target tracking, step scaling, or scheduled scaling based on CPU utilization or request count.
4. **Attach Load Balancer (Optional)**
   * Integrate an **Application Load Balancer (ALB)** for distributing traffic across instances.
5. **Review and Create**
   * Verify all settings, create the Auto Scaling Group, and monitor scaling activity.

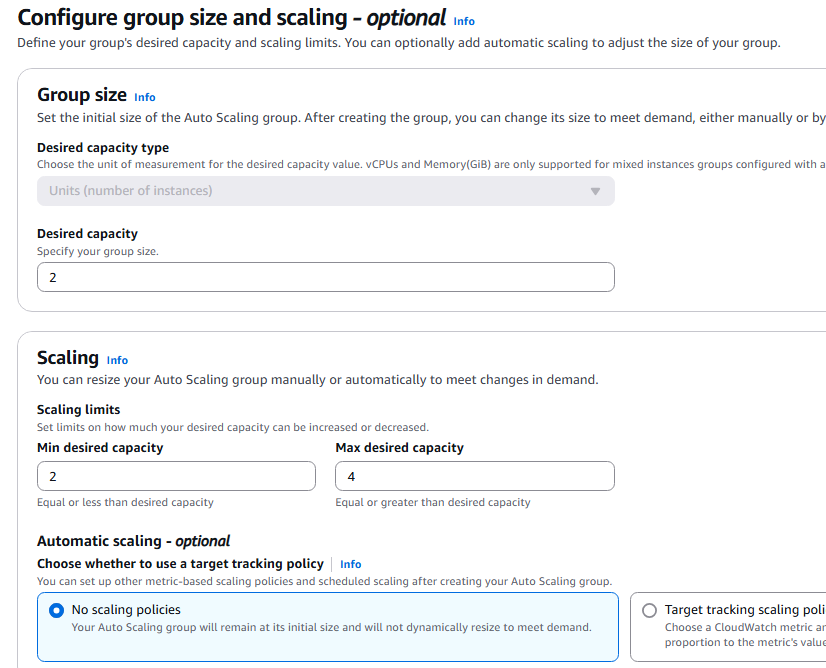


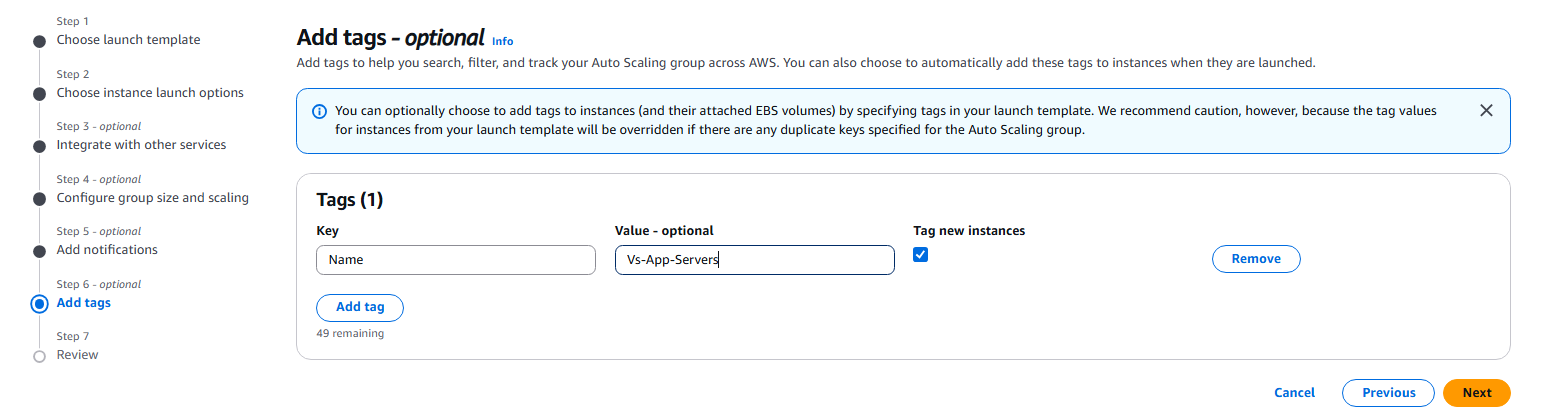
🡺Attach our load balancer’s target group which contains our launch template

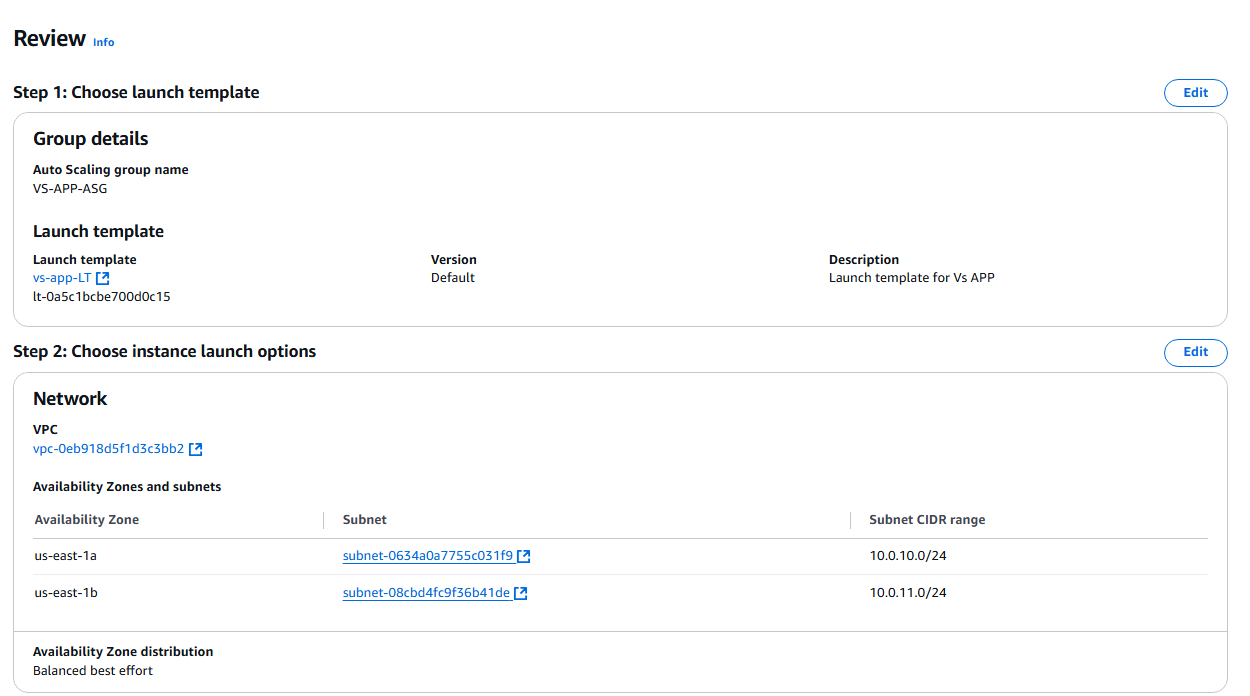


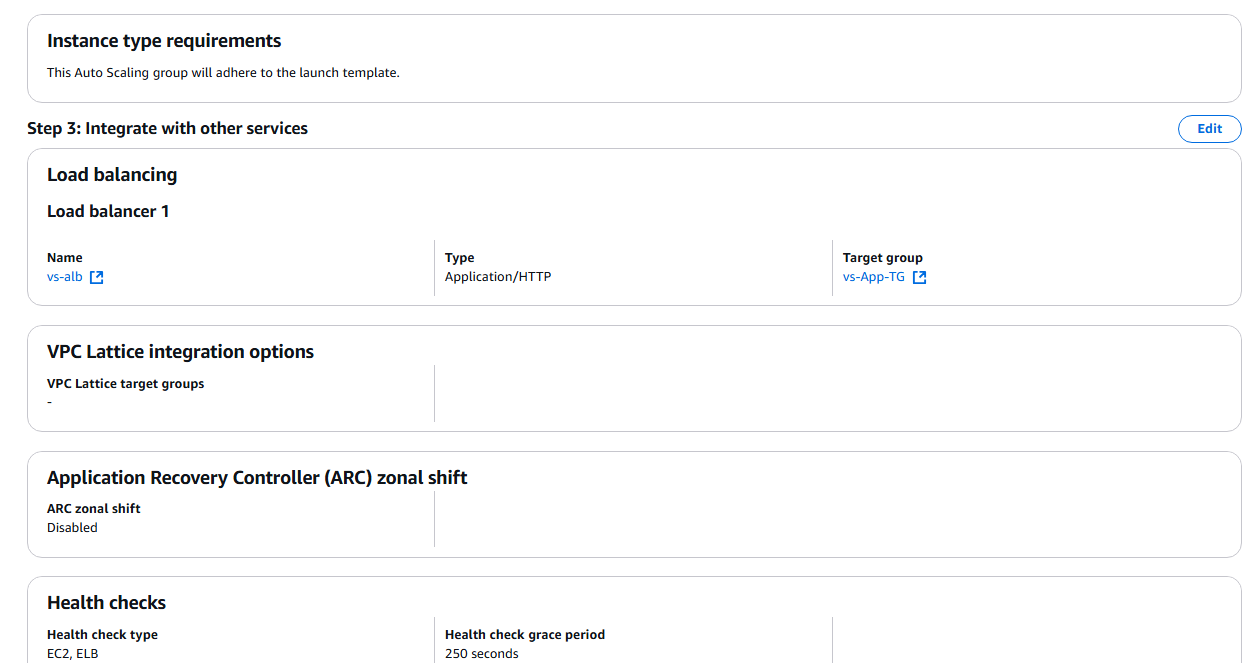


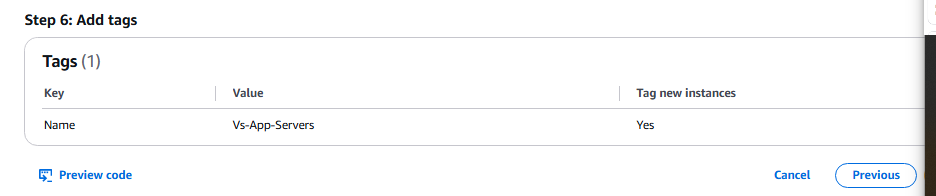
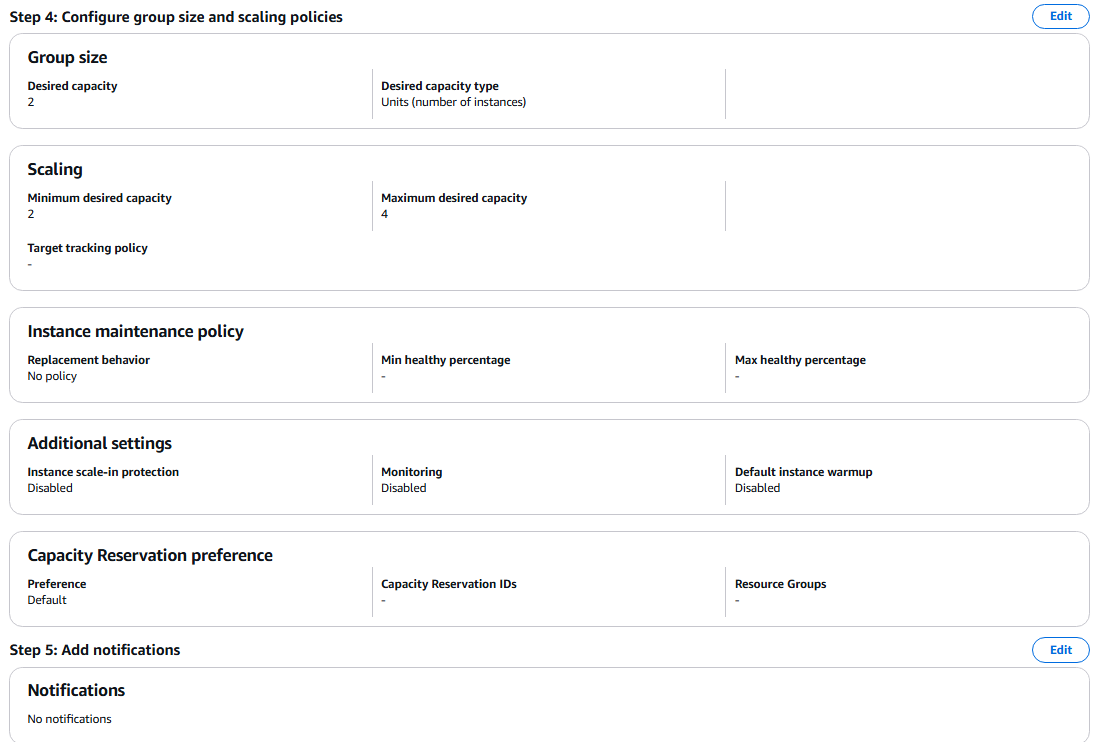
🡺Set up an **Auto Scaling Group** with a **minimum of 2 instances** to ensure high availability and a **maximum of 4 instances** to handle traffic spikes efficiently.

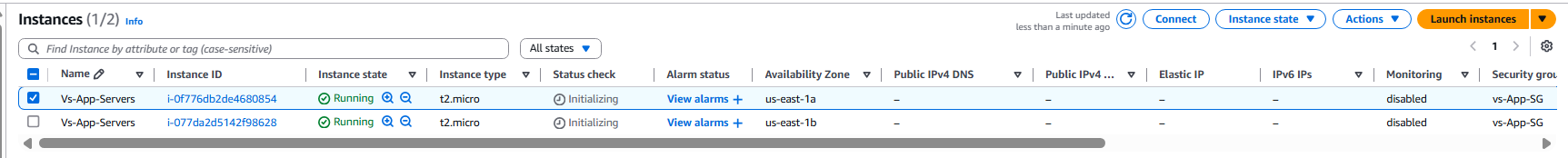










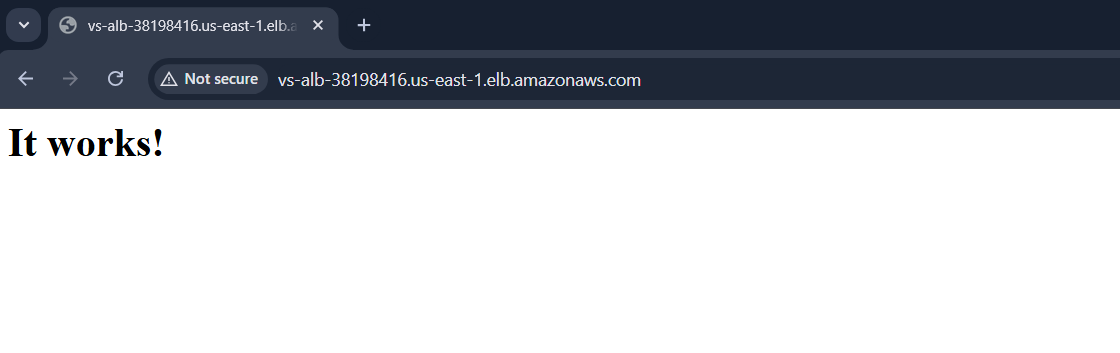


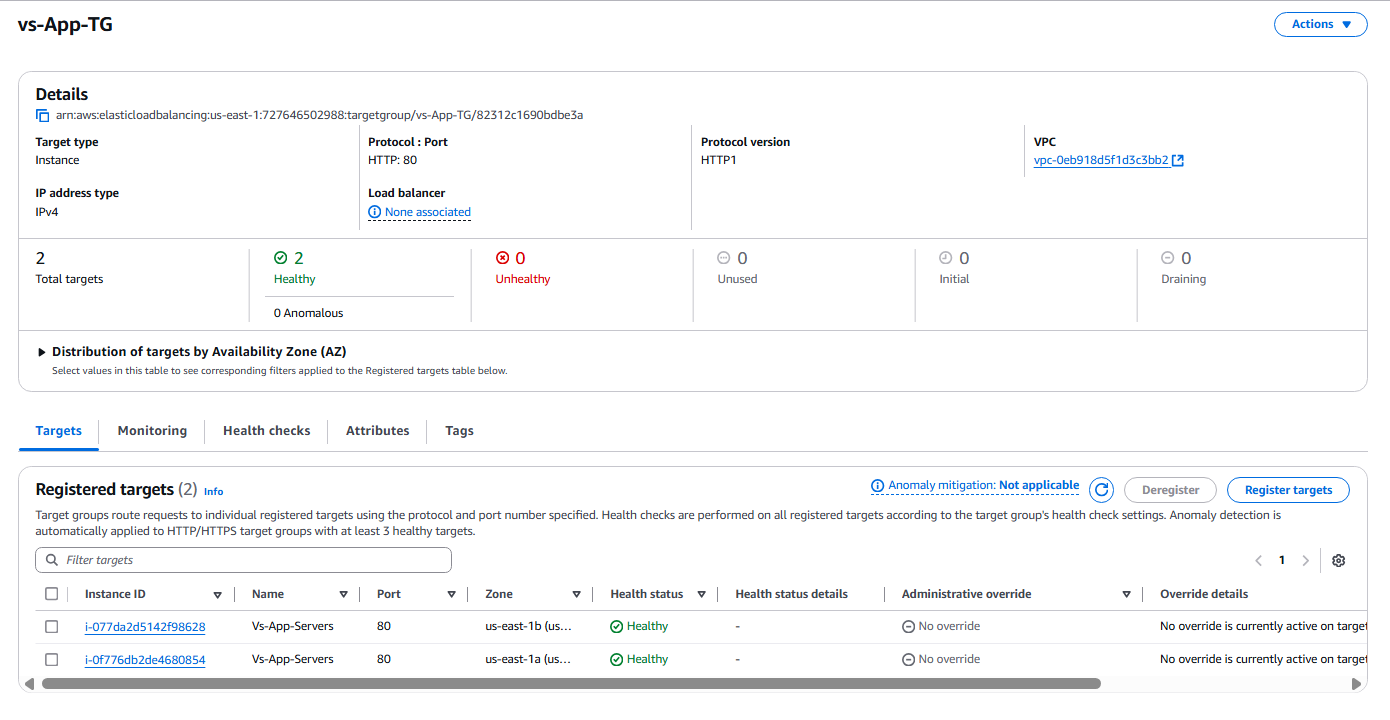
🡺Here our Instances (Website) getting launched using autoscaling group



**Final Steps to Check Application is Working**

1. **Copy the ALB DNS Name**
   * Go to **EC2 Console** → **Load Balancers** → Select your **Application Load Balancer (ALB)**.
   * Copy the **DNS name** from the **Description tab**.
2. **Open in Browser**
   * Paste the **ALB DNS name** into a browser.
   * If the application is working, the webpage should load successfully.

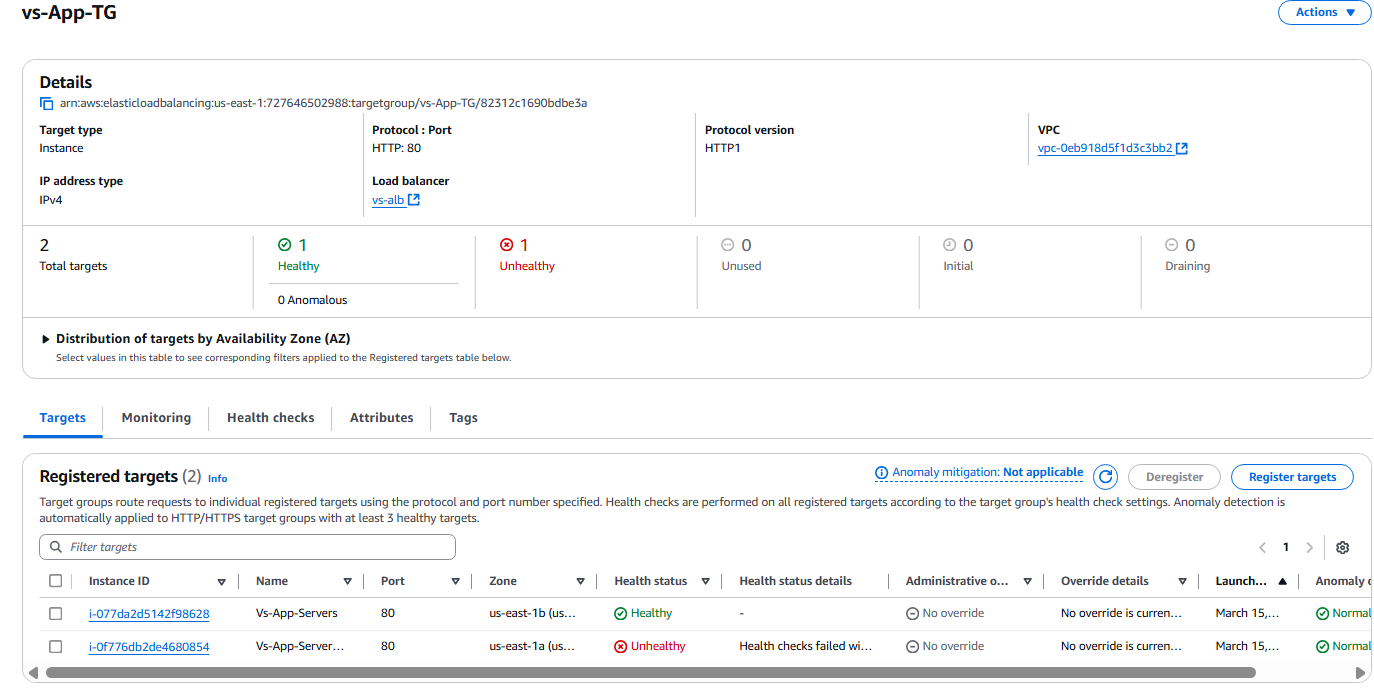


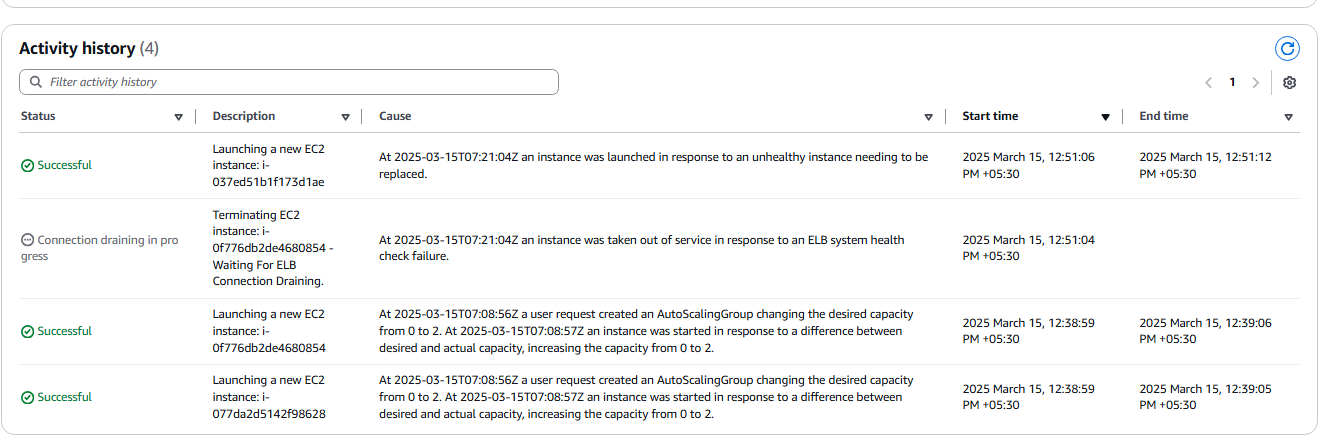


**Verifying Health Check by Manually Deleting health.htm via SSM**

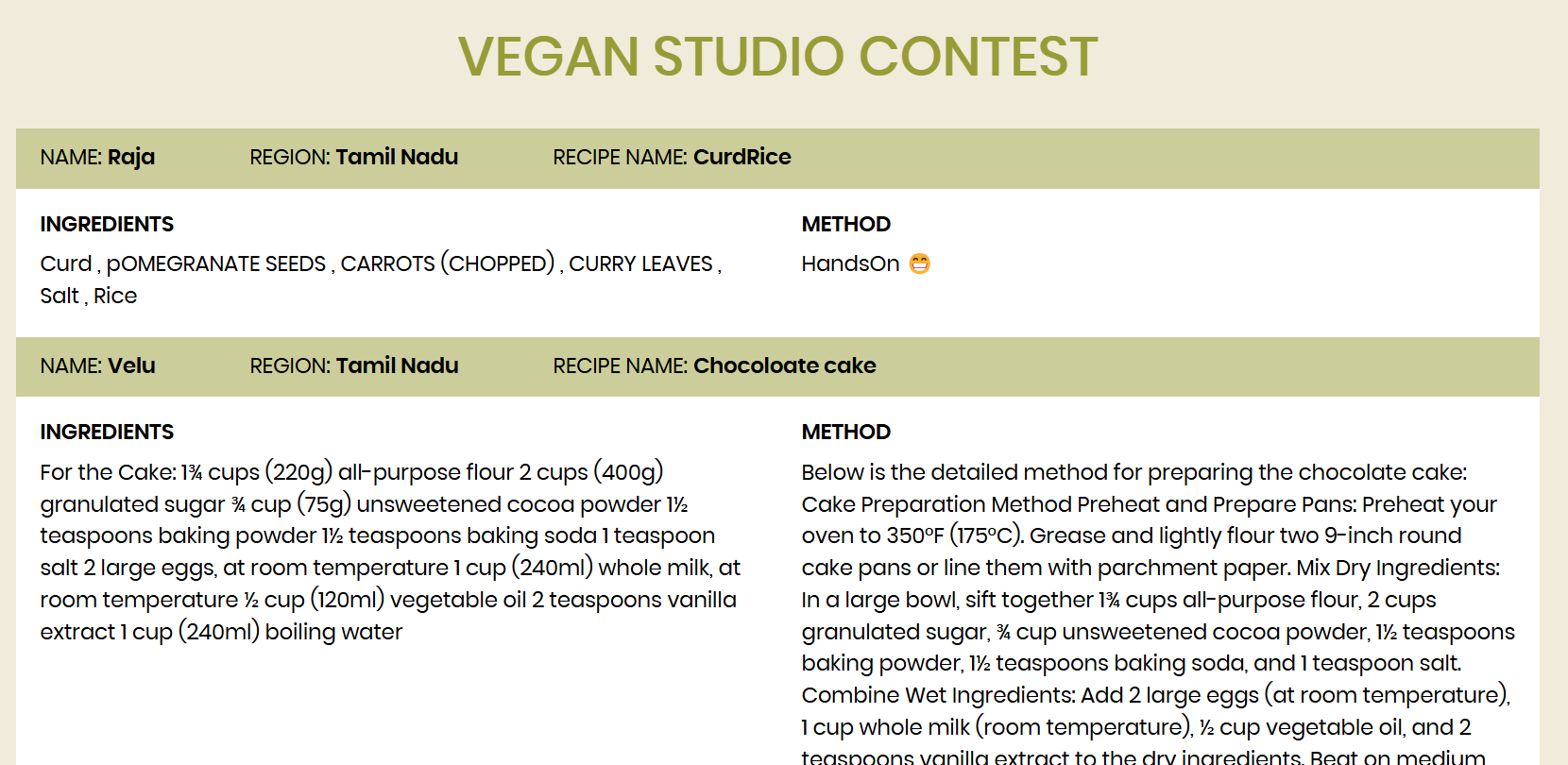
1. Connect using SSM for simple and secured connection without keypair



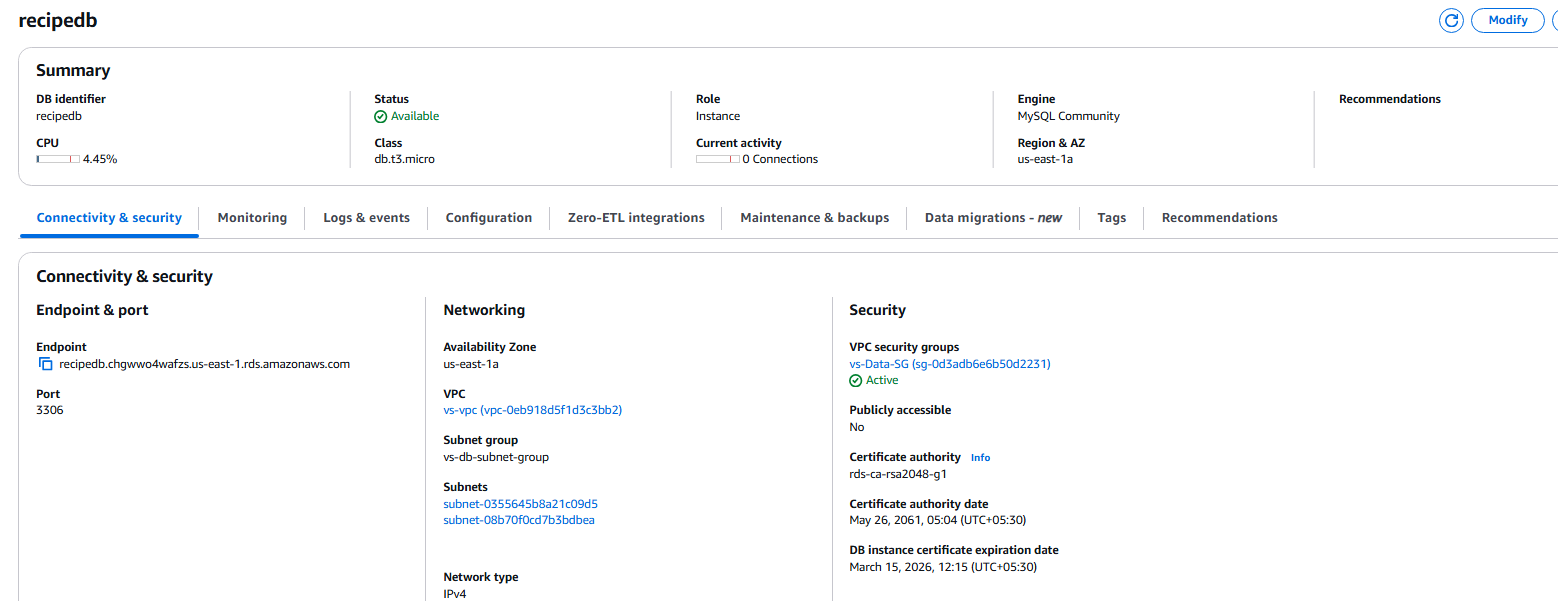




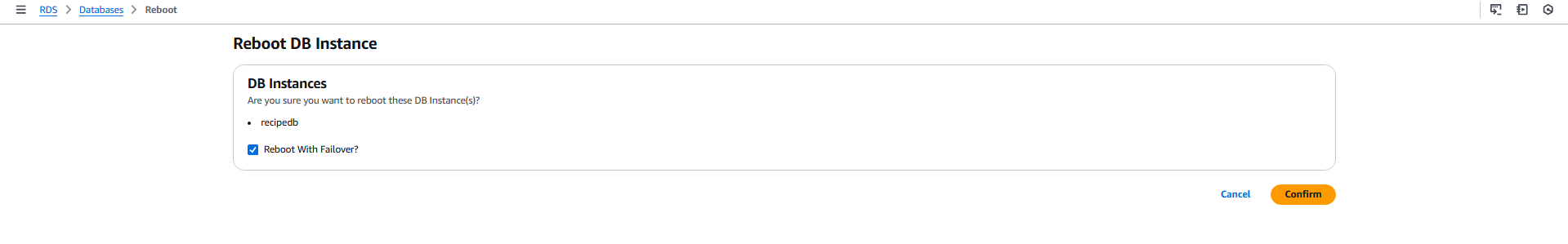
Going inside elb.dns.name/website-code.html

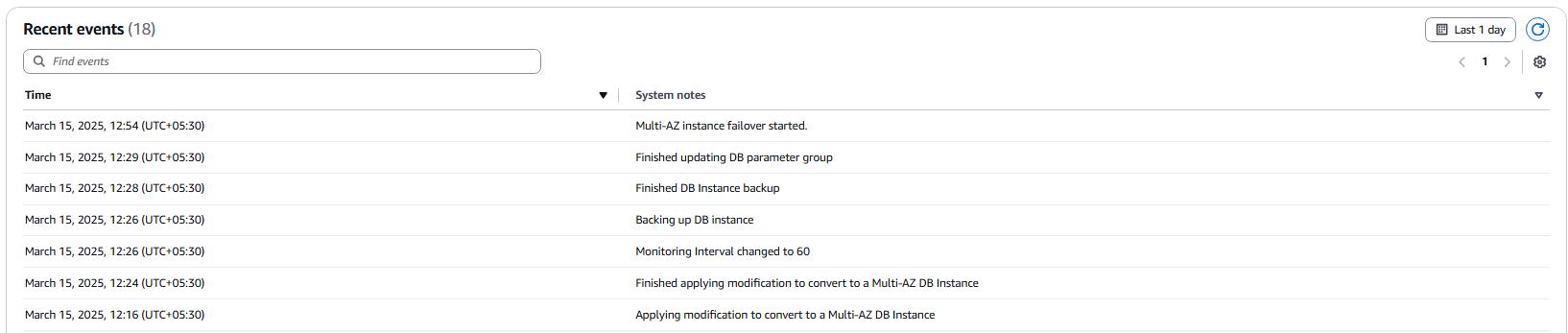


🡺AZ failover test

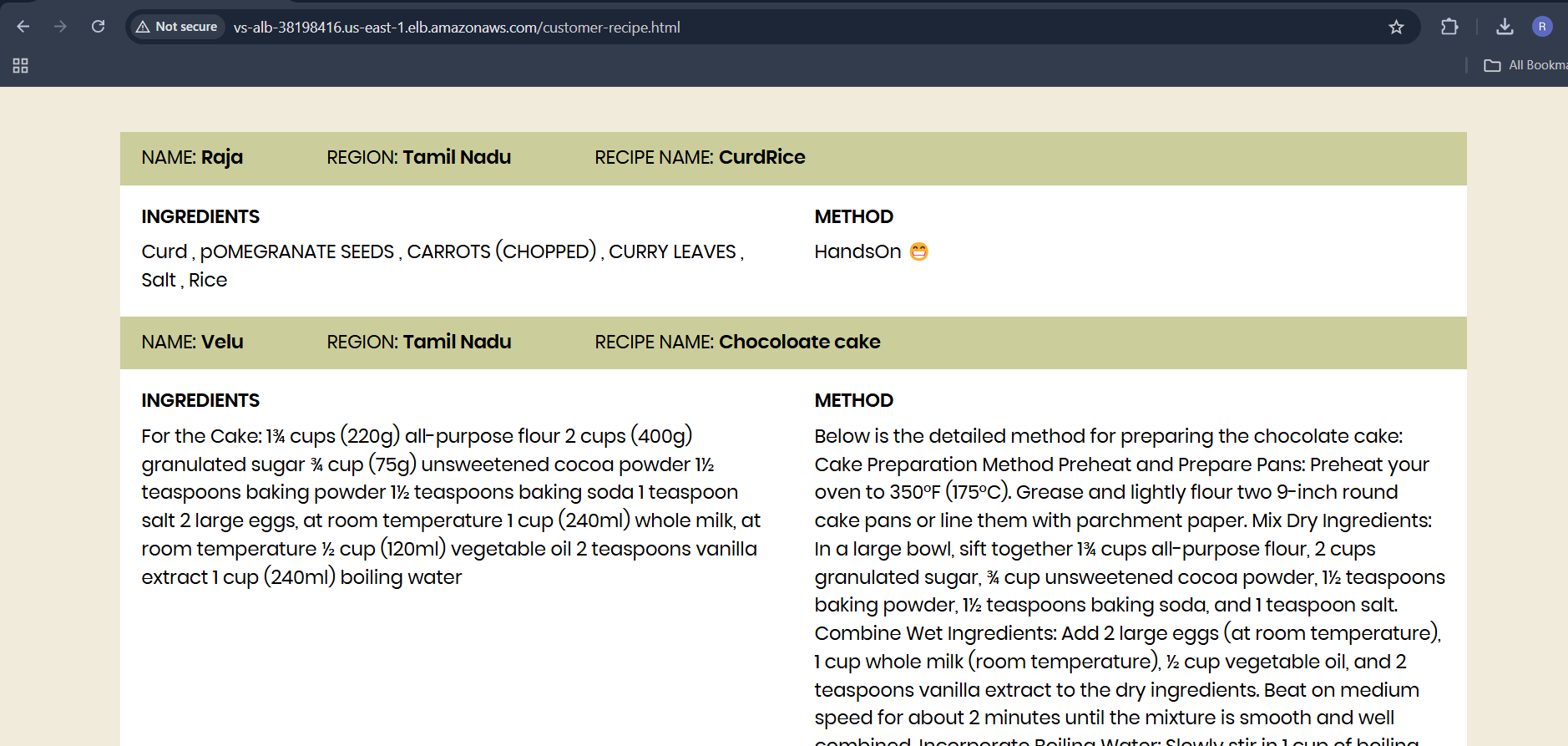


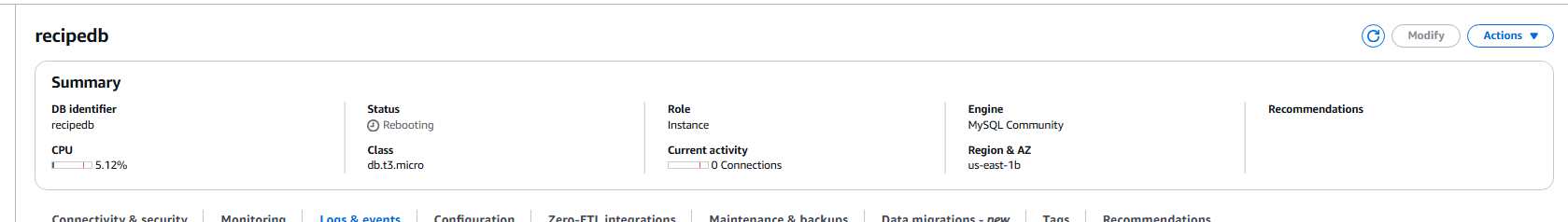
🡺performing reboot operation manually





🡺Checking Data Base Setup:





🡺Our DataBase Is still on Synchronous with user even though one of the az is down:

