

3-tier VPC setup on AWS.

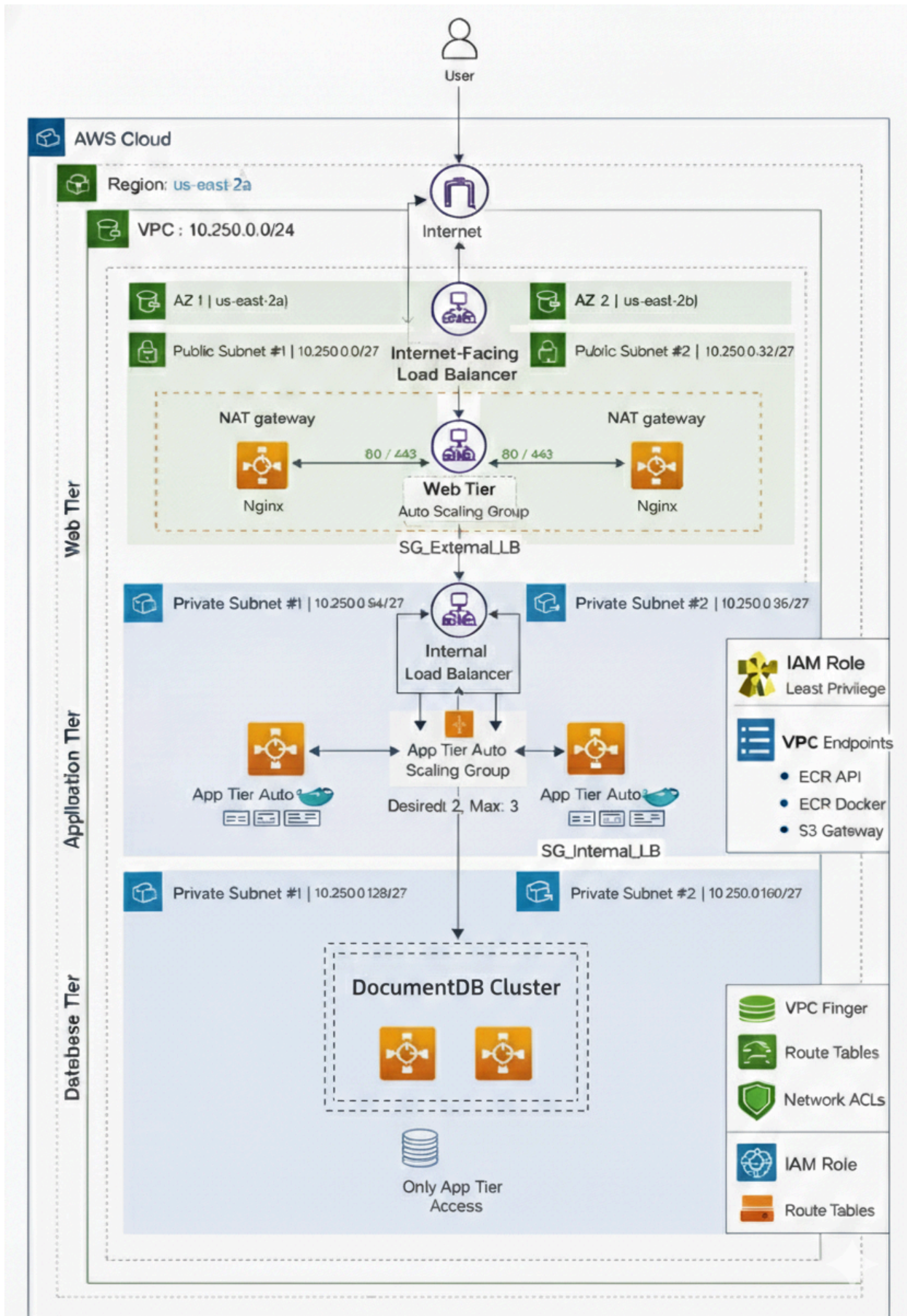
Goal:

The objective of this project is to design and deploy a **secure, highly available, and scalable three-tier application architecture on AWS** using a **custom Virtual Private Cloud (VPC)**.

The solution is architected with **clearly defined public and private subnets** to enforce strong network isolation, and leverages **Multi-Availability Zone (Multi-AZ) deployments** along with **Auto Scaling Groups (ASG)** to achieve resilience, fault tolerance, and horizontal scalability. **Layered security controls** are implemented using **Security Groups and Network ACLs** to strictly regulate inbound and outbound traffic between application tiers.

Access to AWS resources is governed through **IAM roles and policies** following the **principle of least privilege**, ensuring secure, auditable, and controlled access. The architecture is designed in alignment with **AWS production best practices**, emphasizing **security, availability, operational reliability, and future extensibility**.

Architecture Diagram:



Tier	Purpose	Placement
Web Tier	Frontend (Nginx) + External Load Balancer	Public Subnets
Application Tier	Backend services (Docker containers) + Internal Load Balancer	Private Subnets
Database Tier	MongoDB (Amazon DocumentDB)	Private Subnets

Step 1: Creation of VPC

A **Virtual Private Cloud (VPC)** is an AWS networking service that enables the creation of a **logically isolated virtual network** within the AWS cloud. It allows full control over network configuration by defining a **custom IPv4 CIDR block**, which forms the foundation for subnetting, routing, and security boundaries.

In this project, a **custom VPC** is created to serve as the **core networking layer** for the three-tier architecture, enabling secure communication between application components while maintaining isolation from other AWS environments.

Parameter	Value
VPC Name	VPC_ue2_prod_ecommerce_app
Region	us-east-2
IPv4 CIDR	10.250.0.0/24

Naming Convention as per AWS recommends:

vpc-RegionCode-EnvironmentCode-ApplicationStackCode

Create VPC [Info](#)

A VPC is an isolated portion of the AWS Cloud populated by AWS objects, such as Amazon EC2 instances.

VPC settings

Resources to create [Info](#)
Create only the VPC resource or the VPC and other networking resources.

☒ VPC only ☐ VPC and more

Name tag - optional [Info](#)
Creates a tag with a key of "Name" and a value that you specify.

VPC_ue2_prod_ecommerce_app

IPv4 CIDR block [Info](#)
☒ IPv4 CIDR manual input
☐ IPAM-allocated IPv4 CIDR block

IPv4 CIDR
10.250.0.0/24
CIDR block size must be between /16 and /28.

IPv6 CIDR block [Info](#)
☒ No IPv6 CIDR block
☐ IPAM-allocated IPv6 CIDR block
☐ Amazon-provided IPv6 CIDR block
☐ IPv6 CIDR owned by me

Tenancy [Info](#)
Default

VPC encryption control (\$) [Info](#)
Monitor mode provides visibility into encryption status without blocking traffic. Enforce mode prevents unencrypted traffic. [Additional charges apply](#)

☒ None ☐ Monitor mode ☐ Enforce mode

Tags
A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.

Key	Value - optional	
Q Name	X	Q VPC_ue2_prod_ecommerce_app X Remove tag

Step 2: Subnet Design & Creation

Subnet: A subnet is the subdivision of a network. It is the range of IP addresses in your VPC

Subnet Strategy

- **2 Public Subnets** (Web Tier)
- **2 Private Subnets** (Application Tier)
- **2 Private Subnets** (Database Tier)
- Distributed across **2 Availability Zones** for high availability

AWS Reserved IP Addresses

In every subnet:

- First **4 IPs** and the **last IP** are reserved by AWS

Example (10.250.0.32/27):

.32 → Network Address
.33 → VPC Router

.34 → Amazon DNS
.35 → AWS Reserved
.63 → Broadcast

Subnet Allocation

Subnet	CIDR	AZ
Public-AZ1	10.250.0.0/27	us-east-2a
Public-AZ2	10.250.0.32/27	us-east-2b
Private-App-AZ1	10.250.0.64/27	us-east-2a
Private-App-AZ2	10.250.0.96/27	us-east-2b
Private-DB-AZ1	10.250.0.128/27	us-east-2a
Private-DB-AZ2	10.250.0.160/27	us-east-2b

VPC > Subnets > Create subnet

10.250.0.0/24

Subnet settings

Specify the CIDR blocks and Availability Zone for the subnet.

Subnet 1 of 1

Subnet name
Create a tag with a key of 'Name' and a value that you specify.
subnet_public_E_Commerce_Store_AZ1
The name can be up to 256 characters long.

Availability Zone Info
Choose the zone in which your subnet will reside, or let Amazon choose one for you.
United States (Ohio) / us-east-2a

IPv4 VPC CIDR block Info
Choose the VPC's IPv4 CIDR block for the subnet. The subnet's IPv4 CIDR must lie within this block.
10.250.0.0/24

IPv4 subnet CIDR block
10.250.0.0/27 32 IPs

Tags - optional

Key	Value - optional
Name	subnet_public_E_Commerce_Store_AZ1

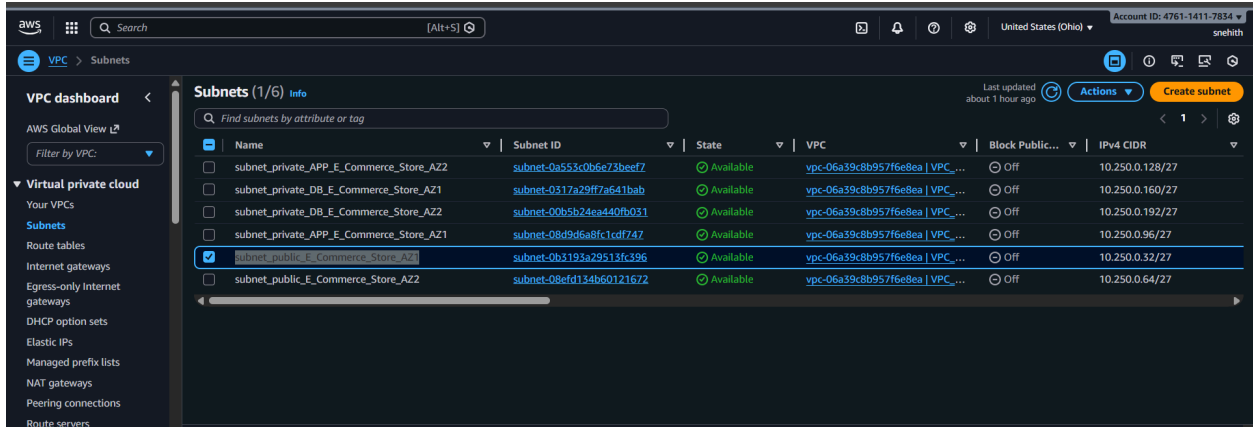
Add new tag
You can add 49 more tags.

Remove

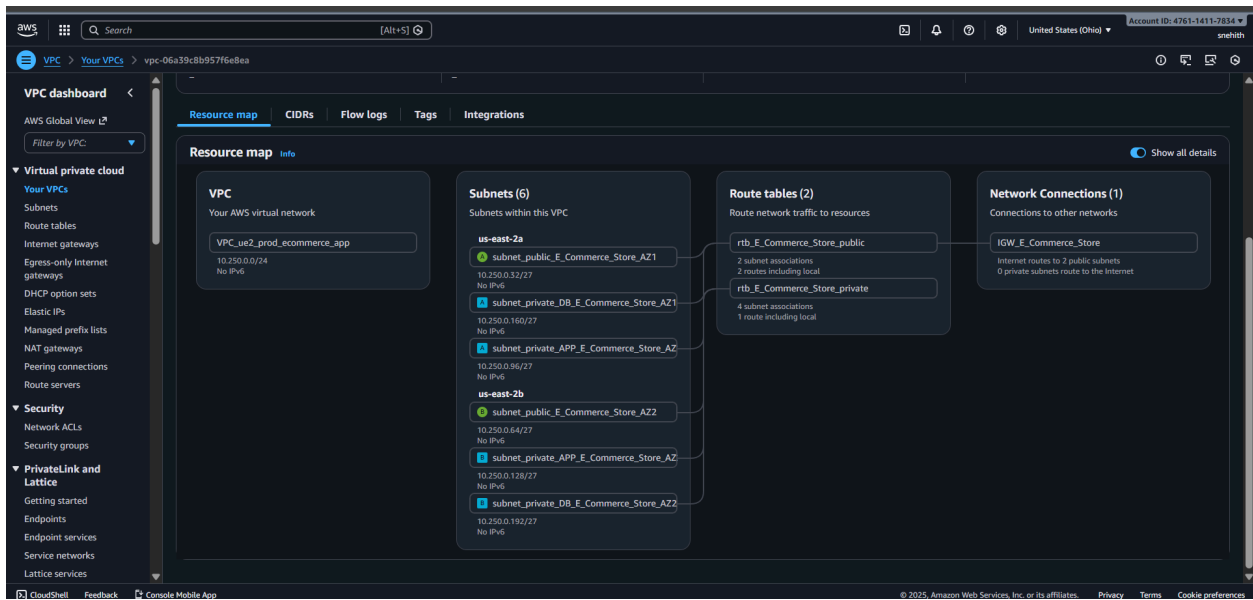
CloudShell Feedback Console Mobile App

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Create the remaining 5 subnets in the same way.



VPC Resource map:



Step 3: Security Group Design

Security Groups enforce **resource-level access control**.

Security Groups Used

Security Group	Purpose
SG_External_LB_e_commerce_store	Allow Internet traffic on 80/443
SG_web_e_commerce_store	Allow traffic only from External LB (port 80)
SG_Internal_LB_e_commerce_store	Allow traffic from Web Tier (8080/4443)
SG_APP_e_commerce_store	Allow traffic only from Internal LB

SG_DB_e_commerce_store

Allow traffic only from App Tier to DB

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](#)

SG_External_LB_e_commerce_store

Name cannot be edited after creation.

Description [Info](#)

Allow access from Internet on port 80 & 443

VPC [Info](#)

vpc-06a39c8b957f6e8ea (VPC_ue2_prod_ecommerce_app) ▼

Inbound rules [Info](#)

Type	Protocol	Port range	Source	Description - optional
Custom TCP ▼	TCP	80	Anywhere... ▼	Allow access from Internet on port 80 & 443
			0.0.0.0/0 ✕	Allow access from Internet on port 80 & 443

[Add rule](#) [Delete](#)

Step 4: Database Tier – Amazon DocumentDB

Why Amazon DocumentDB?

- MongoDB is compatible with our application
- Managed, highly available
- Multi-AZ support

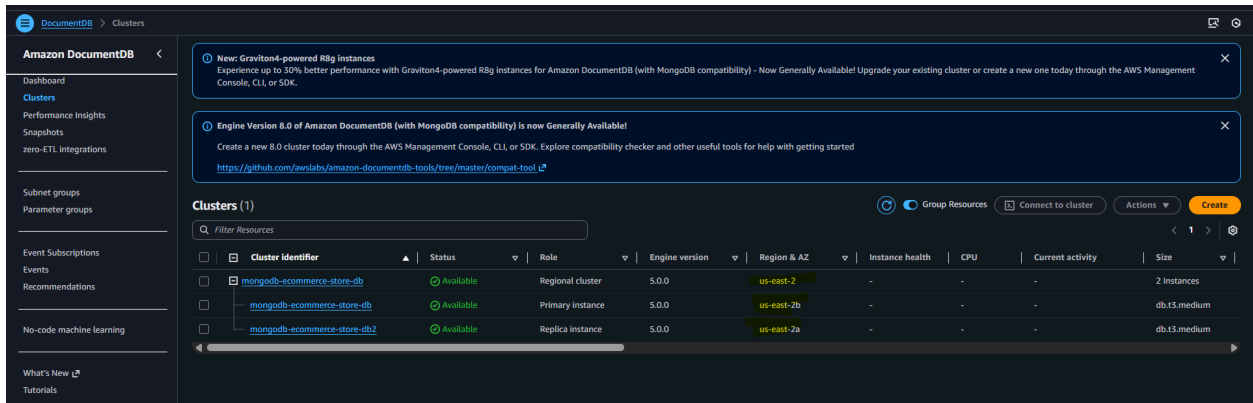
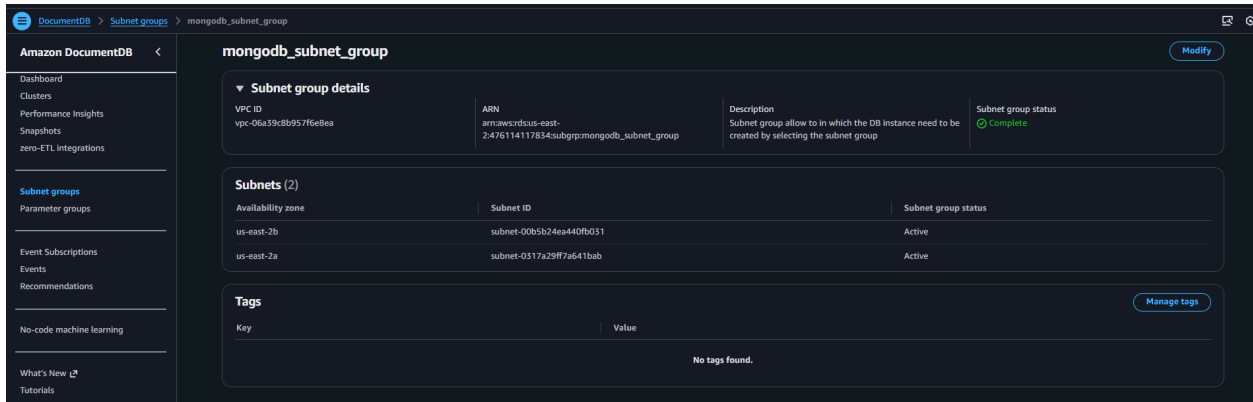
Subnet Group Creation:

Since DocumentDB requires a subnet group:

- Subnet Group Name: mongodb_subnet_group
- Select VPC
- Add DB private subnets
- Create subnet group

Cluster Configuration

- Cluster identifier: Give a unique name
- Replica Instances: 2 (Multi-AZ)
- Username: Provide username
- Authentication: AWS Secrets Manager
- Security Group: SG_DB_e_commerce_store

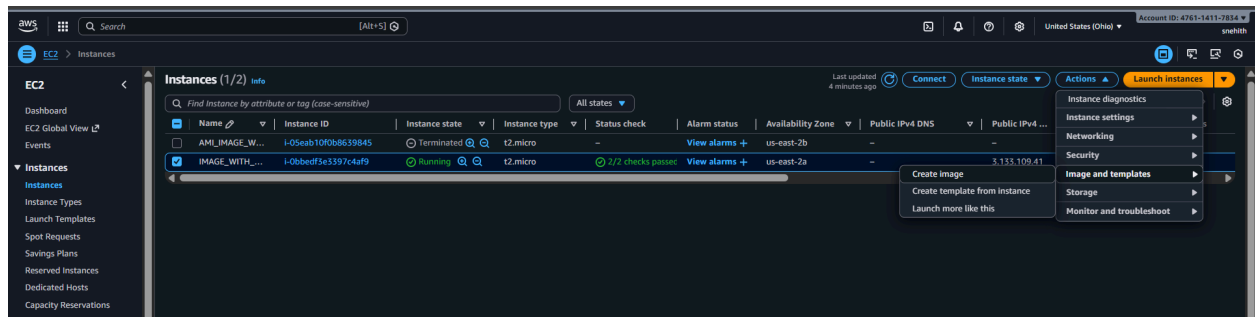


Step 5: Application Tier Setup

1. Creating the AMI (With docker and AWS CLI installed) & creating the Launch template
2. User data (On boot will pull the image from ECR and start the container)
3. Set up the ASG for the auto scaling.

8.1 AMI Creation

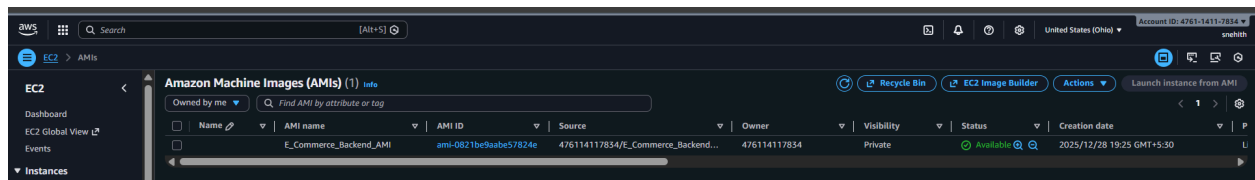
- Launch EC2
- Install:
- Docker
- AWS CLI
- Convert instance to custom AMI



Once AMI is created, create the launch template

8.2 Launch Template

Setting	Value
Template Name	Backend_Template
AMI	Backend AMI
Subnet	Do not include
Key Pair	Do not include
Security Group	SG_APP_e_commerce_store
Attach role:	E_commerce_role



8.3 VPC Endpoints (Private Connectivity)

Since App EC2 instances are private, the following VPC endpoints are created for connecting the ECR to pull/push the image:

- ECR API Endpoint
- ECR Docker Endpoint
- S3 Gateway Endpoint

1. Creating the ECR_API_Endpoint

Setting	Value
Name	ECR_DKR_ENDPOINT
service	com.amazonaws.us-east-2.ecr.api
Security Group	SG_APP_e_commerce_store

Create endpoint [info](#)
Create the type of VPC endpoint that supports the service, service network or resource to which you want to connect.

Endpoint settings
Specify a name and select the type of endpoint.

Name tag - optional
Creates a tag with a key of 'Name' and a value that you specify. Tags help you find and manage your endpoint.

ECR_DKR_ENDPOINT

Type [info](#)
Select a category

- ☒ **AWS services**
Connect to services provided by Amazon with an interface endpoint, or a Gateway endpoint
- ☐ **PrivateLink Ready partner services**
Connect to SaaS services which have AWS Service Ready designation with an Interface endpoint. Uses AWS PrivateLink.
- ☐ **AWS Marketplace services**
Connect to SaaS services that you have purchased through AWS Marketplace with an Interface Endpoint
- ☐ **EC2 Instance Connect Endpoint**
An elastic network interface that allows you to connect to resources in a private subnet
- ☐ **Resources**
Connect to resources like Amazon Relational Database Services (RDS) with a Resource endpoint. Uses AWS PrivateLink.
- ☐ **Service networks**
Connect to VPC Lattice service networks with a Service network endpoint. Uses AWS PrivateLink.
- ☐ **Endpoint services that use NLBs and GWLBs**
Find services shared with you by service name. Connect to a Network Loadbalancer (NLB) service with an Interface endpoint or to a Gateway Loadbalancer (GWLB) service with a Gateway Load Balancer endpoint

Service Region

☐ **Enable Cross Region endpoint** [info](#)
Connect to cross Region enabled services.

Showing services available in service region: United States (Ohio) (us-east-2)

Services (1/1)

Search

Service Name: com.amazonaws.us-east-2.ecr.dkr X Clear filters

Service Name	Owner	Type
com.amazonaws.us-east-2.ecr.dkr	amazon	Interface

Network settings
Select the VPC in which to create the endpoint

VPC
Create the VPC endpoint in the VPC in the same AWS Region from which you will access a resource.

vpc-06a39c8b957f6e8ea (VPC_ue2_prod_ecommerce_app)

Additional settings

Private DNS name

☒ **Enable private DNS name** [info](#)
Associates a private hosted zone with the VPC that contains a record set that enables you to leverage Amazon's private network connectivity to the service while making requests to the service's default public endpoint DNS name. To use this feature, ensure that the attributes 'Enable DNS hostnames' and 'Enable DNS support' are enabled for your VPC.

DNS record IP type

In sameway, create for the other endpoints as well.

8.4 User Data Script (Backend EC2)

Attach below userdata:

```
#!/bin/bash
REGION="us-west-2"
ACCOUNT_ID="476114117834"
ECR_REPO="e_commerce_store"
```

```

IMAGE_TAG="latest"

sudo usermod -aG docker ubuntu

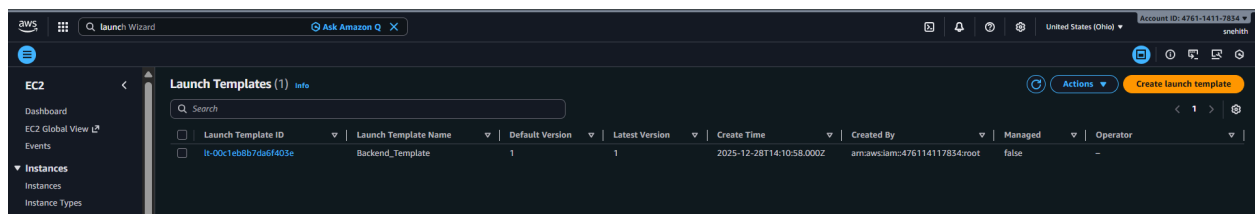
# Login to ECR
aws ecr get-login-password --region $REGION | docker login \
  --username AWS \
  --password-stdin ${ACCOUNT_ID}.dkr.ecr.${REGION}.amazonaws.com

# Pull image
docker pull
${ACCOUNT_ID}.dkr.ecr.${REGION}.amazonaws.com/${ECR_REPO}:${IMAGE_TAG}

# Stop old container if exists
docker rm -f
${ACCOUNT_ID}.dkr.ecr.${REGION}.amazonaws.com/${ECR_REPO}:${IMAGE_TAG} ||
true

# Run container
docker run -d --name e_commerce_store_backend -p 5000:5000 \
  ${ACCOUNT_ID}.dkr.ecr.${REGION}.amazonaws.com/${ECR_REPO}:${IMAGE_TAG}

```



Creating the EC2 instance with ASG (for auto scaling feature) and attaching the Internal LB

1.1 Internal Load Balancer & ASG (App Tier)

1.1.0 Creating the target group for attaching the LB

Target Group

- Name: Internal_LB_Target_Group_For_APP_TIER
- Port: 5000 (Application container port)
- Subnets: App Tier

1.1.1 Creation of Internal Application Load Balancer

- Type: Internal
- Subnets: App Tier
- Security Group: SG_Internal_LB_e_commerce_store

1.2. Create an Auto Scaling group

- Name: APP_TIER_ASG_GROUP
- Select launch Template: Backend_Template
- Under Network (Select VPC and Subnet (private_APP_subnet)

Network Info

For most applications, you can use multiple Availability Zones and let EC2 Auto Scaling balance your instances across the zones. The default VPC and default subnets are suitable for getting started quickly.

VPC
Choose the VPC that defines the virtual network for your Auto Scaling group.

vpc-06a39c8b957f6e8ea (VPC_ue2_prod_ecommerce_app)
10.250.0.0/24

[Create a VPC](#)

Availability Zones and subnets
Define which Availability Zones and subnets your Auto Scaling group can use in the chosen VPC.

Select Availability Zones and subnets

use2-az1 (us-east-2a) | subnet-08d9d6a8fc1cdf747
(subnet_private_APP_E_Commerce_Store_AZ1)
10.250.0.96/27

use2-az2 (us-east-2b) | subnet-0a553c0b6e73beef7
(subnet_private_APP_E_Commerce_Store_AZ2)
10.250.0.128/27

[Create a subnet](#)

Availability Zone distribution - new
Auto Scaling automatically balances instances across Availability Zones. If launch failures occur in a zone, select a strategy.

☒ **Balanced best effort**
If launches fail in one Availability Zone, Auto Scaling will attempt to launch in another healthy Availability Zone.

☐ **Balanced only**
If launches fail in one Availability Zone, Auto Scaling will continue to attempt to launch in the unhealthy Availability Zone to preserve balanced distribution.

[Cancel](#) [Skip to review](#) [Previous](#) [Next](#)

© 2025, Amazon Web Services, Inc. or its affiliates. [Privacy](#) [Terms](#) [Cookie preferences](#)

Auto Scaling Group

Parameter	Value
Desired	2
Min	2
Max	3

Subnets	Private App Subnets
---------	---------------------

Once ASG is created the it will launch the desired number of instance which we have i.e 2 (desired capacity) in the APP_Tier subnet

The screenshot shows the 'Create Auto Scaling group' wizard in the AWS Management Console, specifically Step 4: 'Configure group size and scaling'. The left sidebar shows the progress of the wizard, with Step 4 highlighted. The main content area is divided into three sections:

- Group size**: This section includes a 'Desired capacity type' dropdown set to 'Units (number of instances)'. Below it, the 'Desired capacity' is set to 2.
- Scaling**: This section includes 'Scaling limits' with 'Min desired capacity' set to 2 and 'Max desired capacity' set to 3. It also has an 'Automatic scaling - optional' section with two radio buttons: 'No scaling policies' (selected) and 'Target tracking scaling policy'.
- Instance maintenance policy**: This section includes a 'Choose a replacement behavior depending on your availability requirements' section with three radio buttons: 'No policy' (selected), 'Launch before terminating', and 'Terminate and launch'.

The bottom of the console shows the footer with '© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms'.

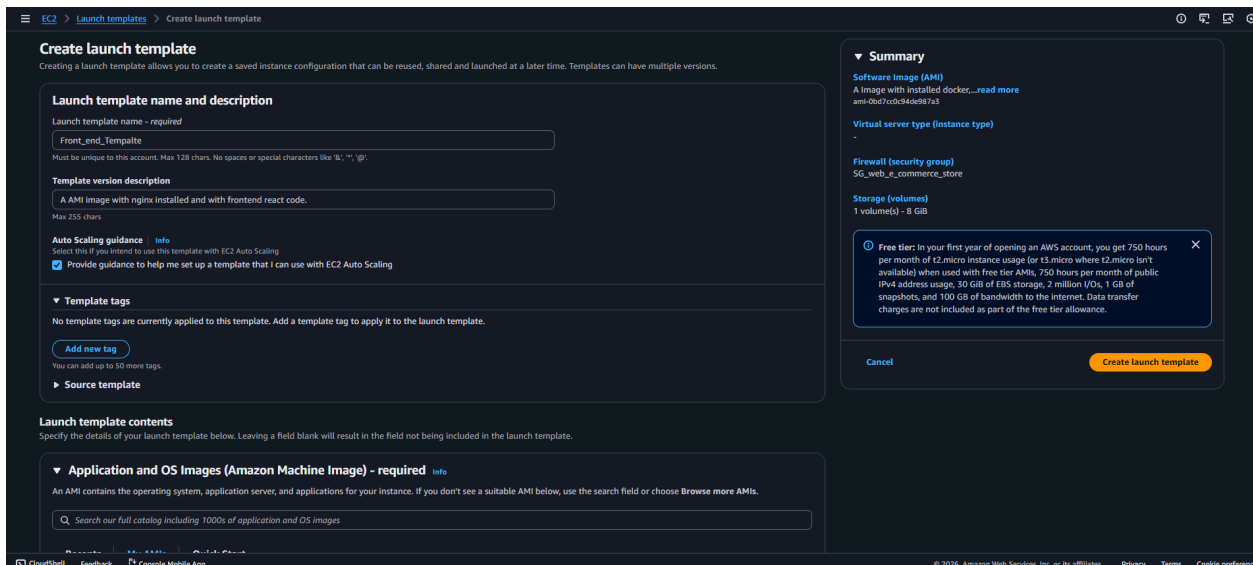
Web Tier Setup

Creating the Web-tier with External Load Balancers

Web AMI

- Install Nginx
- Place frontend code
- Update Nginx configuration
- Create a custom AMI

Nginx config file :



External Load Balancer

- Type: Internet Facing
- Port: 80
- Subnets: Public Subnets
- Security Group: SG_External_LB_e_commerce_store

Target Group

- Port: 8081
- VPC: select VPC

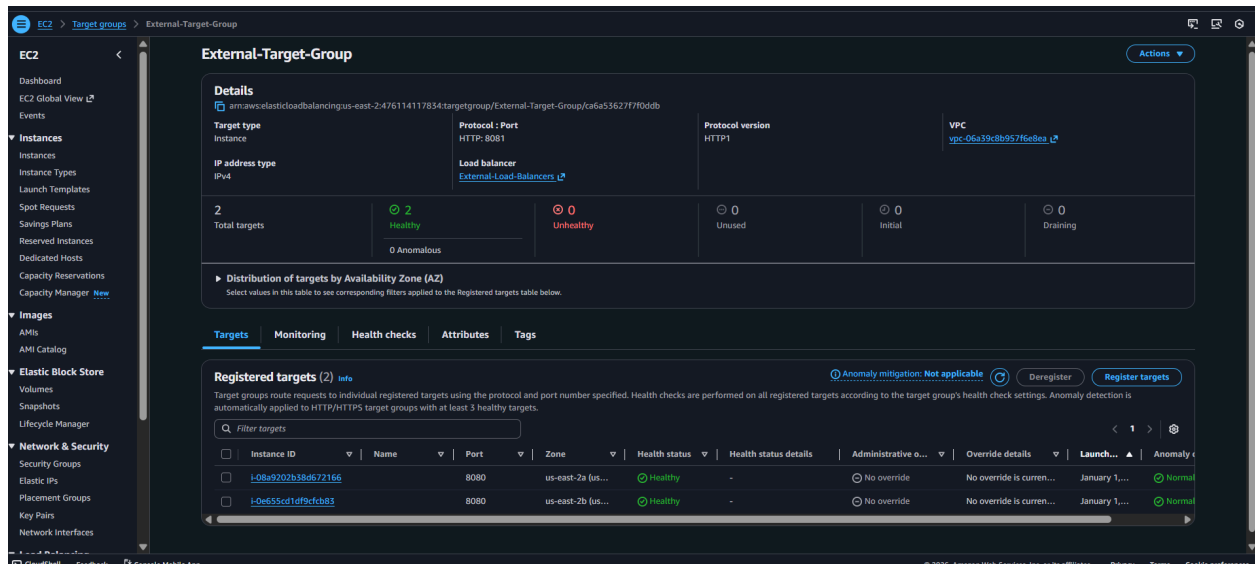
Creation of web_tier_ASG

- Name: Web_Tier_ASG
- Launch template: Front_end_template(select)
- Select VPC and public subnets
- Attach External target group: : External-Target-Group

ASG created the two web_tier instances

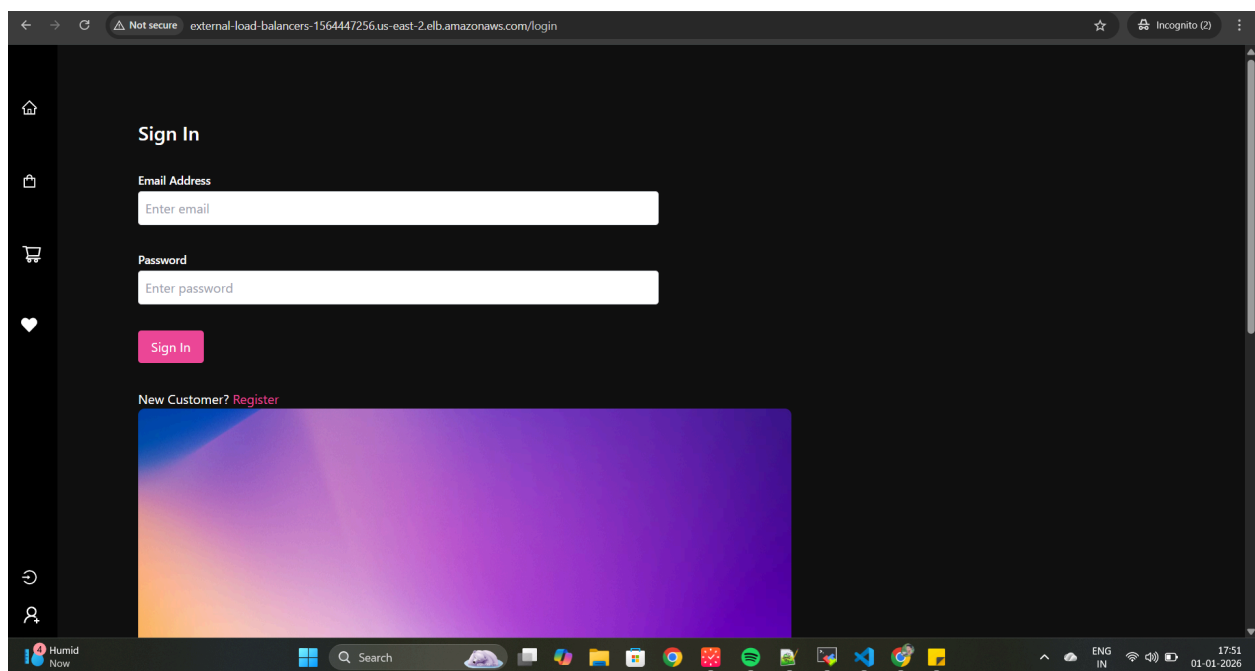
Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP	IPv6 IPs
test_boston	i-062b4098f7163547c	Running	t2.micro	2/2 checks passed	View alarms +	us-east-2a	ec2-13-58-238-8.us-east-2a	13.58.238.8	-	-
	i-0e655cd1df9cfc883	Running	t2.micro	2/2 checks passed	View alarms +	us-east-2b	-	-	-	-
	i-08a9202b38d672166	Running	t2.micro	2/2 checks passed	View alarms +	us-east-2a	-	-	-	-

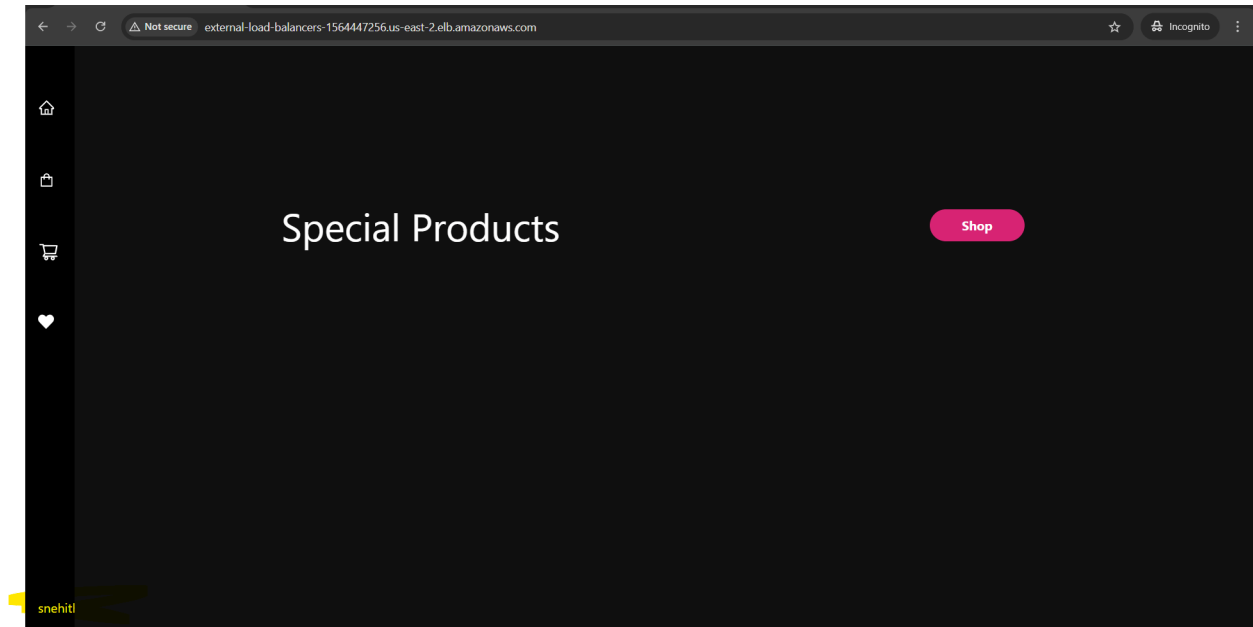
Target Group status:



11. Application Access & Validation

- Access the application using the **External Load Balancer DNS**
- Frontend loads successfully
- Backend connectivity verified
- User registration & login tested





Note: Image loading issue due to missing proxy configuration.

12. Conclusion

This project successfully demonstrates:

- Secure 3-tier AWS architecture
- Proper public/private subnet isolation
- High availability using Multi-AZ
- Scalable compute using ASG
- Secure database deployment in private subnets

The core goal of **secure, production-ready network architecture** is fully achieved.

Next Steps

- Fix frontend proxy/image loading
- Addition of security (WAF) and caching for less latency (CDN : cloudfront)
- Enable CloudWatch alarms
- Provisioning the resources using the terraform

