

3-tier VPC setup on AWS.

Goal:

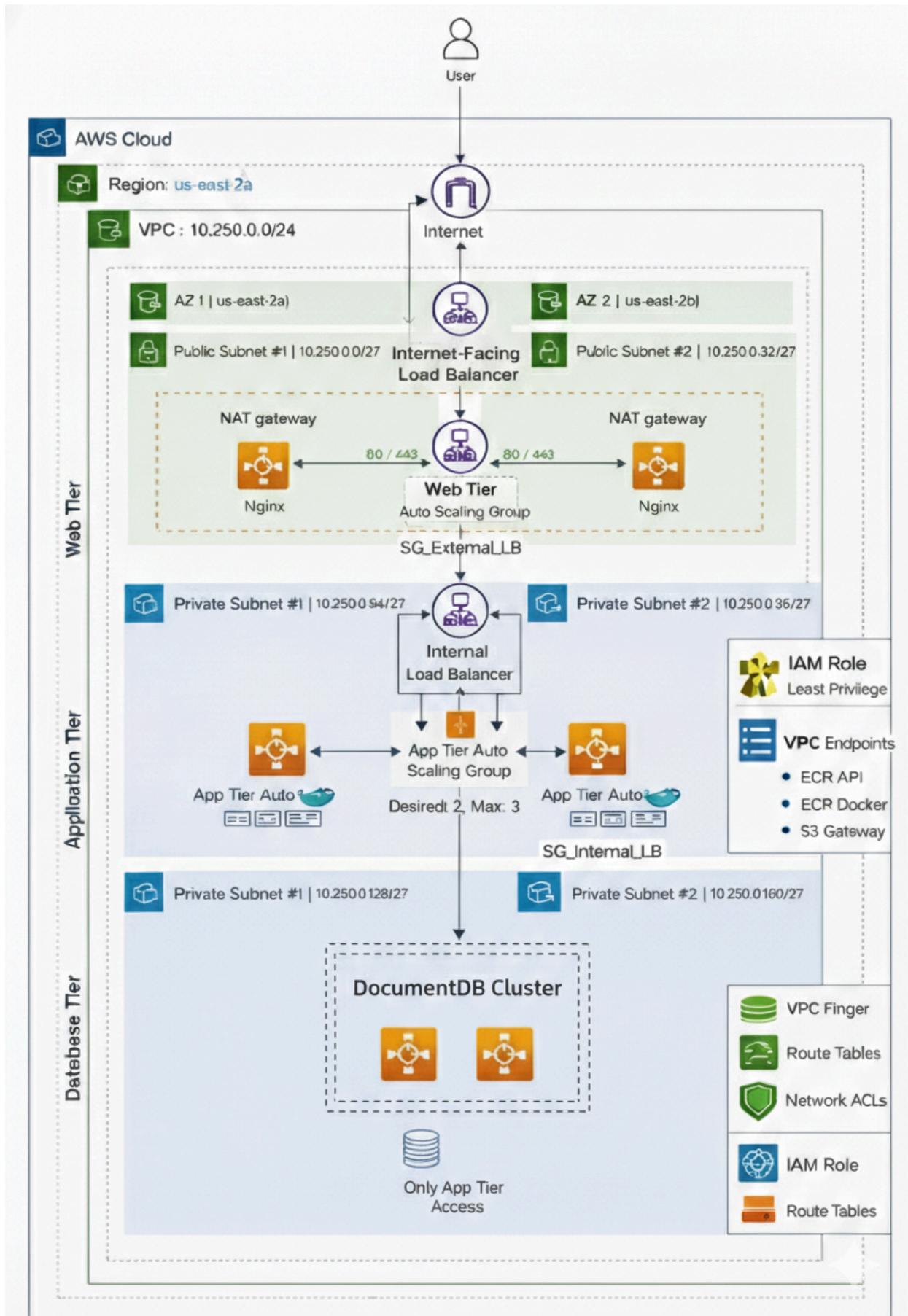
The objective of this project is to design and deploy a **secure, highly available, and scalable three-tier application architecture on AWS** using a **custom Virtual Private Cloud (VPC)**.

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

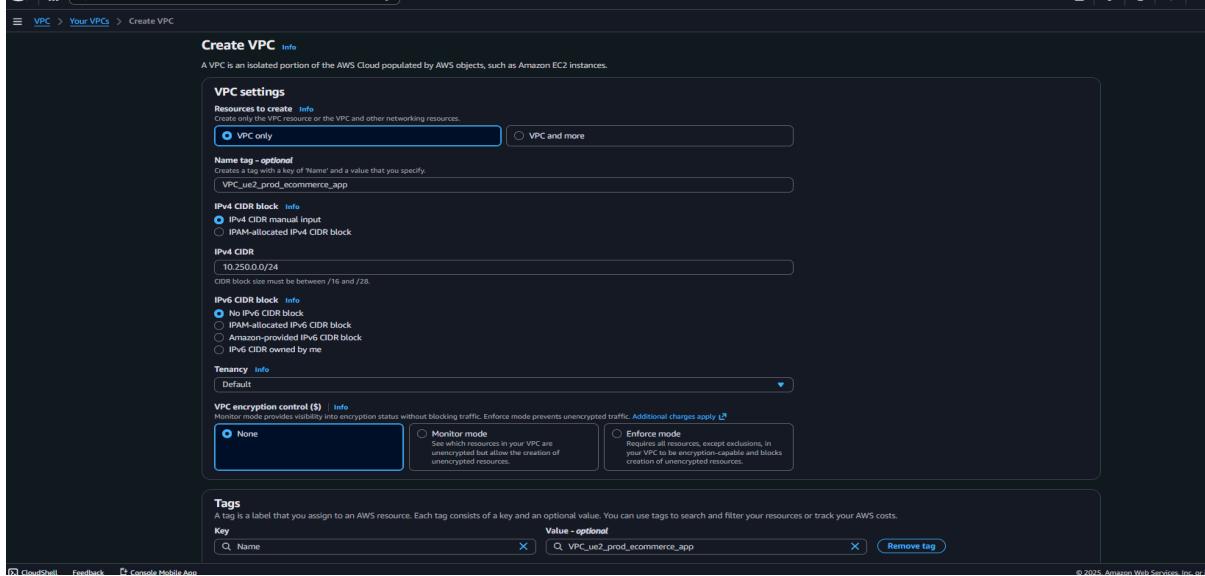
A **Virtual Private Cloud (VPC)** is an AWS networking service that enables the creation of a **logically isolated virtual network** within the AWS cloud. It allows full control over network configuration by defining a **custom IPv4 CIDR block**, which forms the foundation for subnetting, routing, and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



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

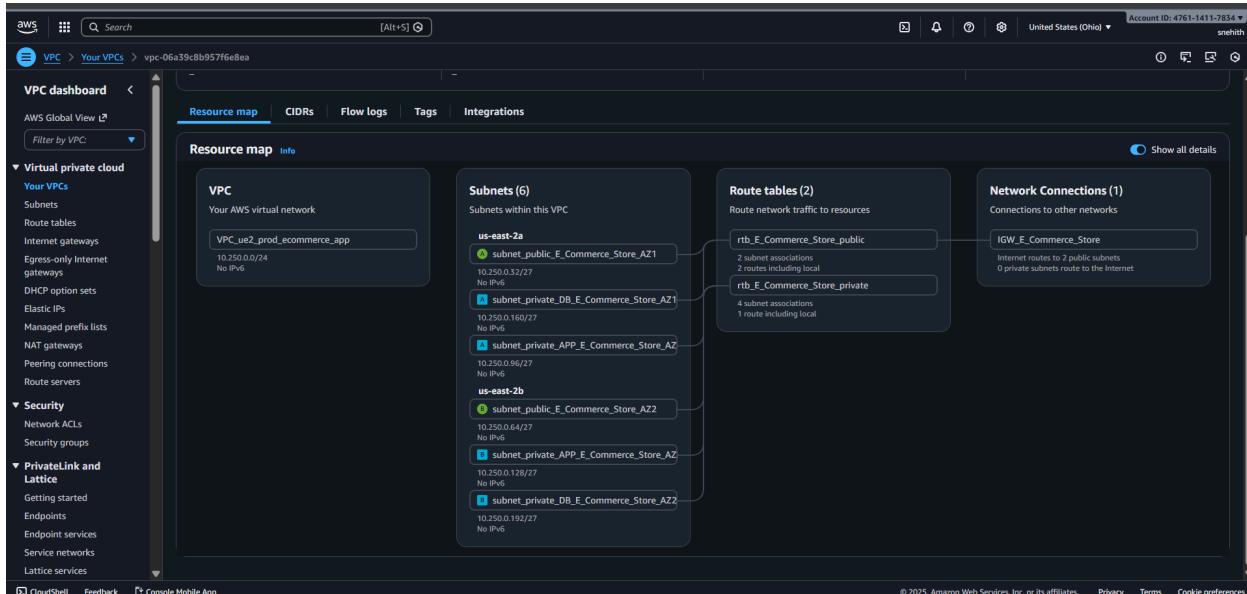
Subnet	CIDR	AZ
Public-AZ1	10.250.0.0/27	us-east-2a
Public-AZ2	10.250.0.32/27	us-east-2b
Private-App-AZ1	10.250.0.64/27	us-east-2a
Private-App-AZ2	10.250.0.96/27	us-east-2b
Private-DB-AZ1	10.250.0.128/27	us-east-2a
Private-DB-AZ2	10.250.0.160/27	us-east-2b

The screenshot shows the AWS VPC Subnets creation interface. The top navigation bar includes 'VPC', 'Subnets', and 'Create subnet'. A search bar at the top right contains '10.250.0.0/24'. The main form area is titled 'Subnet settings' with the sub-section 'Specify the CIDR blocks and Availability Zone for the subnet.' Below this, 'Subnet 1 of 1' is listed. The 'Subnet name' field contains 'subnet_public_E_Commerce_Store_AZ1'. The 'Availability Zone' dropdown is set to 'United States (Ohio) / us-east-2a (us-east-2a)'. The 'IPv4 VPC CIDR block' dropdown is set to '10.250.0.0/24'. The 'IPv4 subnet CIDR block' dropdown is set to '10.250.0.0/27'. Under 'Tags - optional', there is a single tag 'Name' with value 'subnet_public_E_Commerce_Store_AZ1'. The bottom of the screen shows standard AWS footer links: CloudShell, Feedback, Console Mobile App, © 2025, Amazon Web Services, Inc. or its affiliates., Privacy, Terms, and Cookie preferences.

Create the remaining 5 subnets in the same way.

Subnets (1/6) Info						
	Name	Subnet ID	State	VPC	Block Public...	IPv4 CIDR
<input checked="" type="checkbox"/>	subnet_private_APP_E_Commerce_Store_AZ2	subnet-0a553c0be73beef7	Available	vpc-06a39c8b957f6e8ea VPC...	Off	10.250.0.128/27
<input type="checkbox"/>	subnet_private_DB_E_Commerce_Store_AZ1	subnet-0317a29ff7a641bab	Available	vpc-06a39c8b957f6e8ea VPC...	Off	10.250.0.160/27
<input type="checkbox"/>	subnet_private_DB_E_Commerce_Store_AZ2	subnet-00b5b24e9440fb031	Available	vpc-06a39c8b957f6e8ea VPC...	Off	10.250.0.192/27
<input type="checkbox"/>	subnet_private_APP_E_Commerce_Store_AZ1	subnet-08d9d6a8fc1cdf747	Available	vpc-06a39c8b957f6e8ea VPC...	Off	10.250.0.96/27
<input checked="" type="checkbox"/>	subnet_public_E_Commerce_Store_AZ2	subnet-0b3193a29513fc396	Available	vpc-06a39c8b957f6e8ea VPC...	Off	10.250.0.32/27
<input type="checkbox"/>	subnet_public_E_Commerce_Store_AZ2	subnet-08efd134b60121672	Available	vpc-06a39c8b957f6e8ea VPC...	Off	10.250.0.64/27

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
------------------------	--

The screenshot shows the 'Create security group' page in the AWS VPC console. The security group name is 'SG_External_LB_e_commerce_store'. Under 'Basic details', there is a note: '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.' The 'Inbound rules' section contains one rule: 'Custom TCP' on port '80' from 'Anywhere' ('0.0.0.0/0') to '0.0.0.0/0'. The rule description is 'Allow access from Internet on port 80 & 443'. There is also a note at the bottom: 'Allow traffic only from App Tier to DB'.

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

The screenshot shows the 'Subnet groups' section of the Amazon DocumentDB console. It displays a single subnet group named 'mongodb_subnet_group'. The 'Subnet group details' section includes the VPC ID ('vpc-06a59c8b957f6e8ea'), ARN ('arn:aws:rds:us-east-2:476114117834:subgrp:mongodb_subnet_group'), a description ('Subnet group allow to in which the DB instance need to be created by selecting the subnet group'), and a status of 'Complete'. The 'Subnets' section lists two subnets: 'us-east-2b' and 'us-east-2a', both marked as 'Active'. The 'Tags' section shows no tags found.

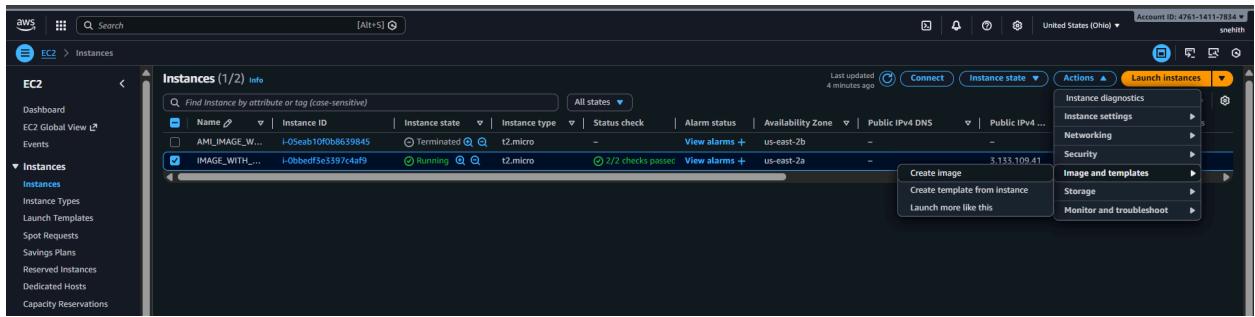
The screenshot shows the 'Clusters' section of the Amazon DocumentDB console. It displays a single cluster named 'mongodb-commerce-store-db'. The cluster table shows three instances: 'Cluster identifier' (Regional cluster, 5.0.0, us-east-2, 2 instances), 'mongodb-commerce-store-db' (Primary instance, 5.0.0, us-east-2b, db.t3.medium), and 'mongodb-commerce-store-db2' (Replica instance, 5.0.0, us-east-2a, db.t3.medium). There are two notifications at the top: one about Graviton4-powered R8g instances and another about Engine Version 8.0. A 'Create' button is visible in the top right corner.

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

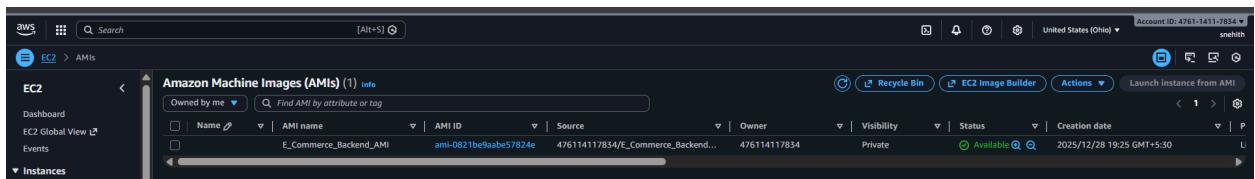
- Launch EC2
- Install:
- Docker
- AWS CLI
- Convert instance to custom AMI



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

The screenshot shows the 'Create endpoint' wizard in the AWS Management Console. The first step, 'Endpoint settings', is completed with the name 'ECR_DKR_ENDPOINT'. The second step, 'Service Region', has the 'Enable Cross Region endpoint' checkbox unchecked. The third step, 'Services', lists a single service named 'com.amazonaws.us-east-2.ecr.dkr'. The fourth step, 'Network settings', shows a VPC selection dropdown containing 'vpc-06a39cd8b957f6e8ea (VPC_ue2_prod_ecommerce_app)'. The fifth step, 'Additional settings', includes options for 'Private DNS name' (checked) and '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

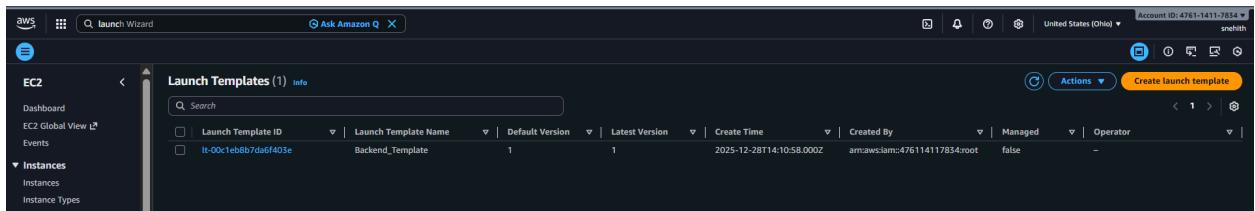
# Login to ECR
aws ecr get-login-password --region $REGION | docker login \
--username AWS \
--password-stdin ${ACCOUNT_ID}.dkr.ecr.${REGION}.amazonaws.com

# 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 i

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 i

use2-az1 (us-east-2a) | subnet-08d9d6a8fc1cd747
(subnet_private_APP_E_Commerce_Store_AZ1)
10.250.0.96/27

use2-az2 (us-east-2b) | subnet-0a5530b6e73beef7
(subnet_private_APP_E_Commerce_Store_AZ2)
10.250.0.128/27

Create a subnet i

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 at Step 4: Configure group size and scaling. The 'Desired capacity type' is set to 'Units (number of instances)' with a value of 2. The 'Scaling' section shows 'Min desired capacity' as 2 and 'Max desired capacity' as 3. Under 'Automatic scaling - optional', the 'No scaling policies' option is selected. The 'Instance maintenance policy' section shows 'Mixed behavior' as the chosen replacement behavior. The bottom of the screen includes standard AWS navigation links like 'Feedback', 'Console Mobile App', and copyright information.

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_tempalte(select)
- Select VPC and public subnets
- Attach External target group: : External-Target-Group

ASG created the two web_tier instances

Instances (3) Info											Actions
	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 IP	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-eas...	13.58.238.8	-	-
		i-0e655cd1df9cfcb83	Running	t2.micro	2/2 checks passed	View alarms +	us-east-2b	-	-	-	-
		i-08a9202b58d672166	Running	t2.micro	2/2 checks passed	View alarms +	us-east-2a	-	-	-	-

Target Group status:

External-Target-Group

Details

arn:aws:elasticloadbalancing:us-east-2:476114117834:targetgroup/External-Target-Group/ca6a55627f7f0ddb

Target type	Protocol : Port	Protocol version	VPC
Instance	HTTP: 8081	HTTP/1	vpc-06a39c8b957f6e8ea
IP address type	IPv4	Load balancer	External-Load-Balancers

2 Total targets | 0 Healthy | 0 Unhealthy | 0 Unused | 0 Initial | 0 Draining | 0 Anomalous

Distribution of targets by Availability Zone (AZ)

Availability Zone	Count
us-east-2a	1
us-east-2b	1

Targets | Monitoring | Health checks | Attributes | Tags

Registered targets (2) Info

Anomaly mitigation: Not applicable

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

Instance ID	Name	Port	Zone	Health status	Health status details	Administrative o...	Override details	Last check	Anomaly
i-08a9202b58d672166		8080	us-east-2a (us...)	Healthy	-	No override	No override is curren...	January 1, 2024	Normal
i-0e655cd1df9efcb83		8080	us-east-2b (us...)	Healthy	-	No override	No override is curren...	January 1, 2024	Normal

Filter targets | Deregister | Register targets

11. Application Access & Validation

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

Not secure external-load-balancers-1564447256.us-east-2.elb.amazonaws.com/login

Sign In

Email Address

Enter email

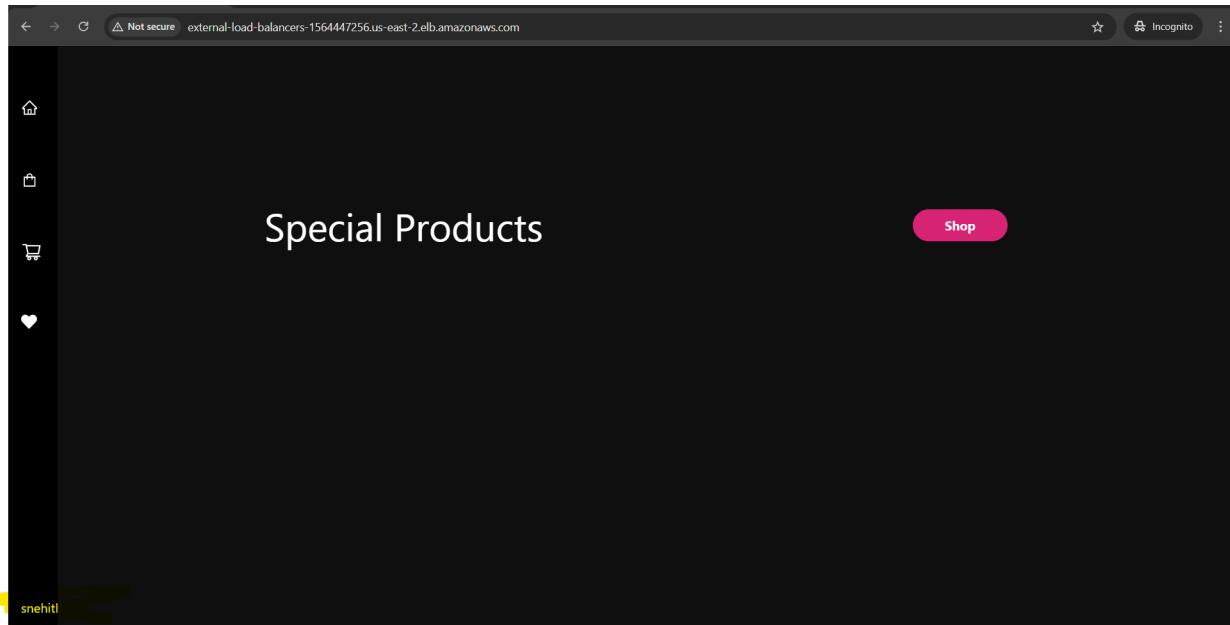
Password

Enter password

Sign In

New Customer? [Register](#)

Humid Now



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

