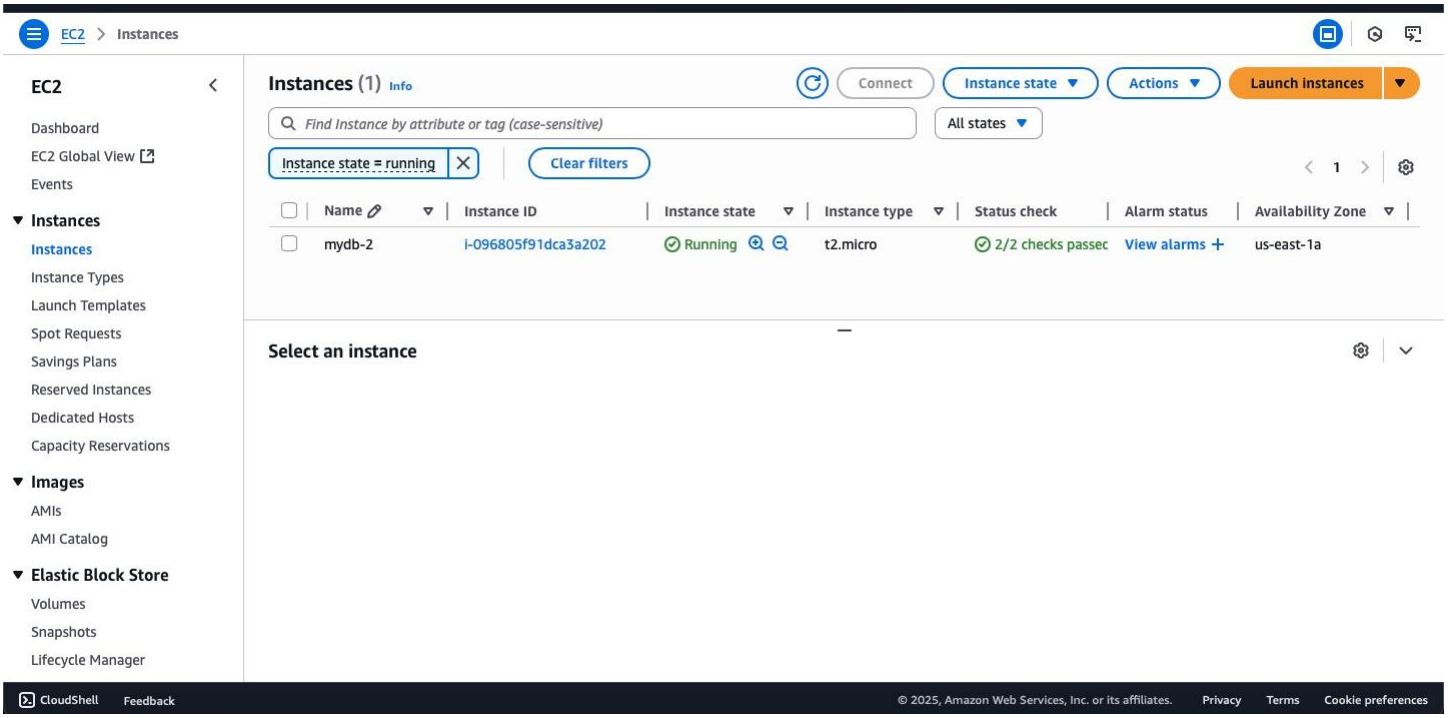
NullClass Cloud Technology (AWS)

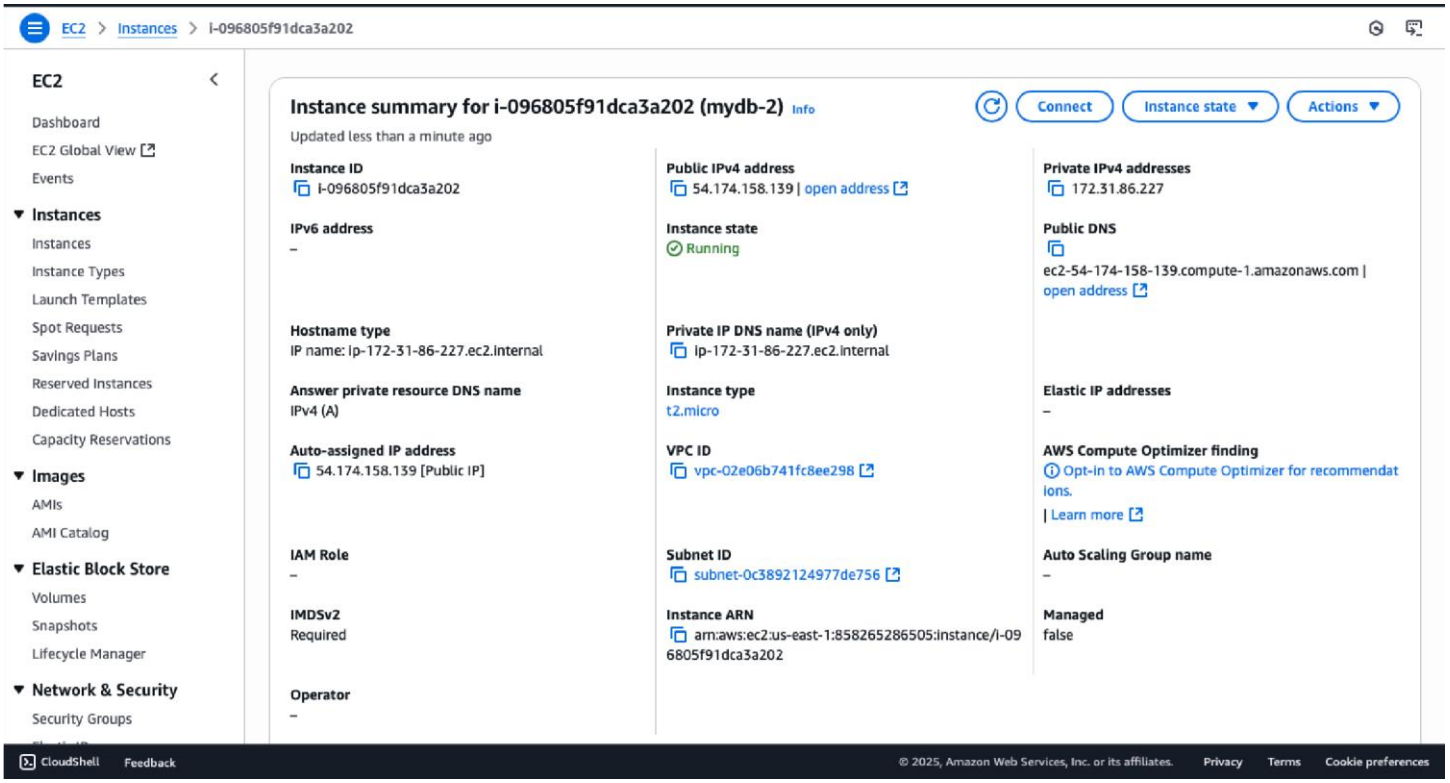
1.Create an RDS MySQL Database and Connect from EC2 Launch an RDS MySQL instance inside your VPC. Modify the security group to allow connections from your EC2 instance. Connect to the database from your EC2 instance using MySQL CLI.

This guide explains how to set up an **Amazon RDS MySQL database** and connect to it from an **EC2 instance**. We will launch an EC2 server, create an RDS MySQL database in the same VPC, configure security groups to allow communication, install the MySQL client on EC2, and then connect to the database. Finally, we will create a table, insert sample records, and verify the connection.

1. Launch an EC2 Instance

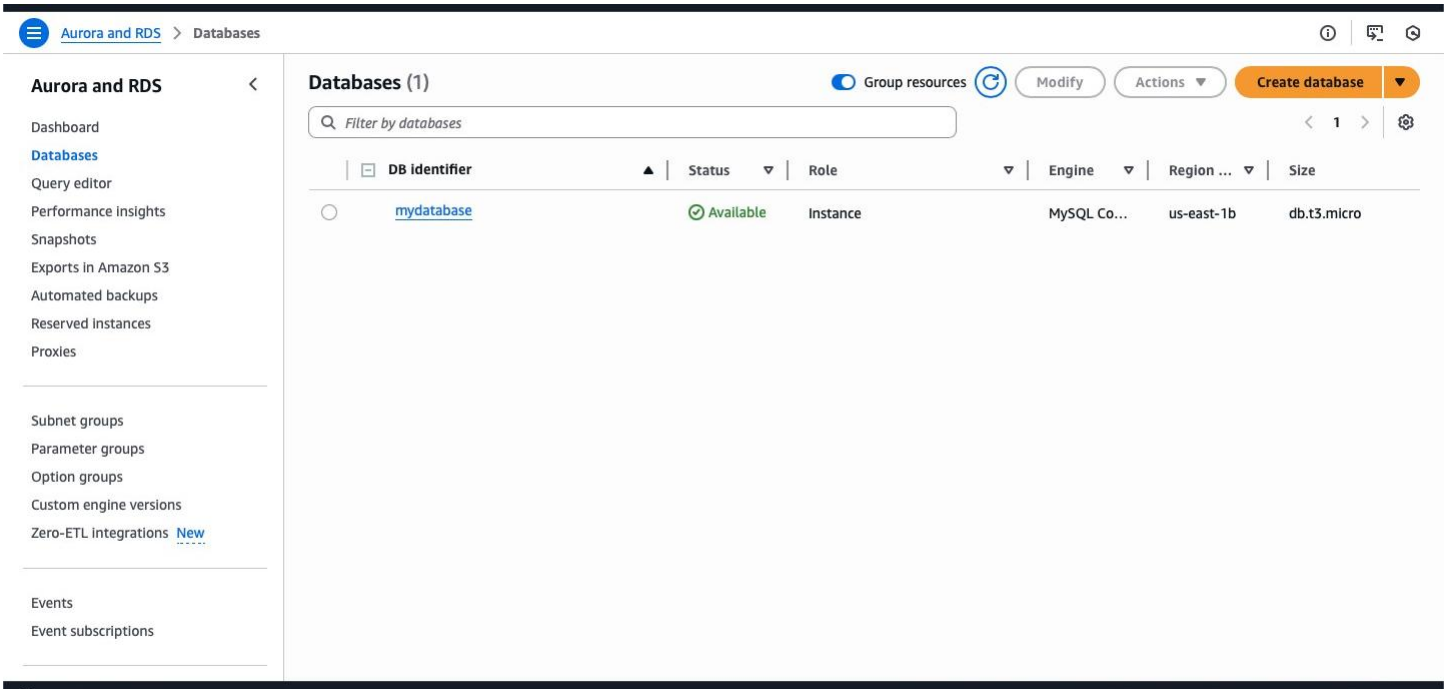
1. Log in to the **AWS Management Console**.
2. Navigate to **EC2 > Instances > Launch Instance**.
3. Choose an Amazon Machine Image (AMI):
   - Select **Amazon Linux 2023** (or Amazon Linux 2).
4. Choose instance type:
   - Example: **t2.micro** (Free Tier eligible).
5. **Key Pair**: Create or select an existing key pair (e.g., sonu-3.pem).
6. Network settings:
   - Attach to your default VPC.
   - Enable a public IP.
7. **Launch** the instance.

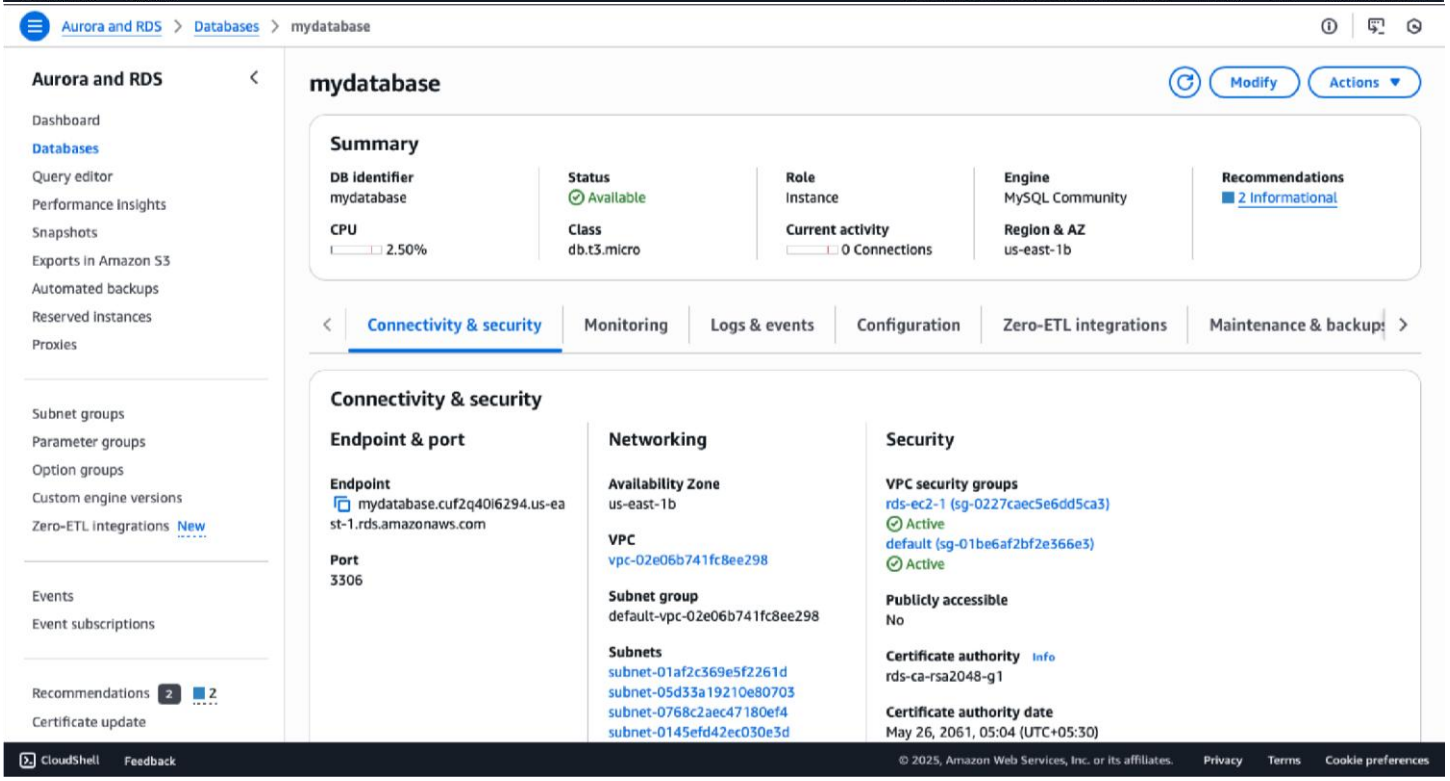




2. Launch an RDS MySQL Database

1. Go to **RDS > Databases > Create Database**.
2. Choose **Standard Create** → **MySQL**.
3. Select version: Example **MySQL 8.0**.
4. Choose **Free Tier** template.
5. DB instance identifier:
   - Example: mydatabase.
6. Set username & password:
   - Username: databasemysql. Enter a
   - secure password.
7. Connectivity:
   - VPC: Select the same VPC as your EC2 instance.
   - Subnet group: Default.
   - Public access: **No** (private DB).
   - Security group: Select or create one (we'll configure it next).
8. Create the database.





3. Configure Security Groups

For EC2 to connect to RDS, security rules must allow traffic:

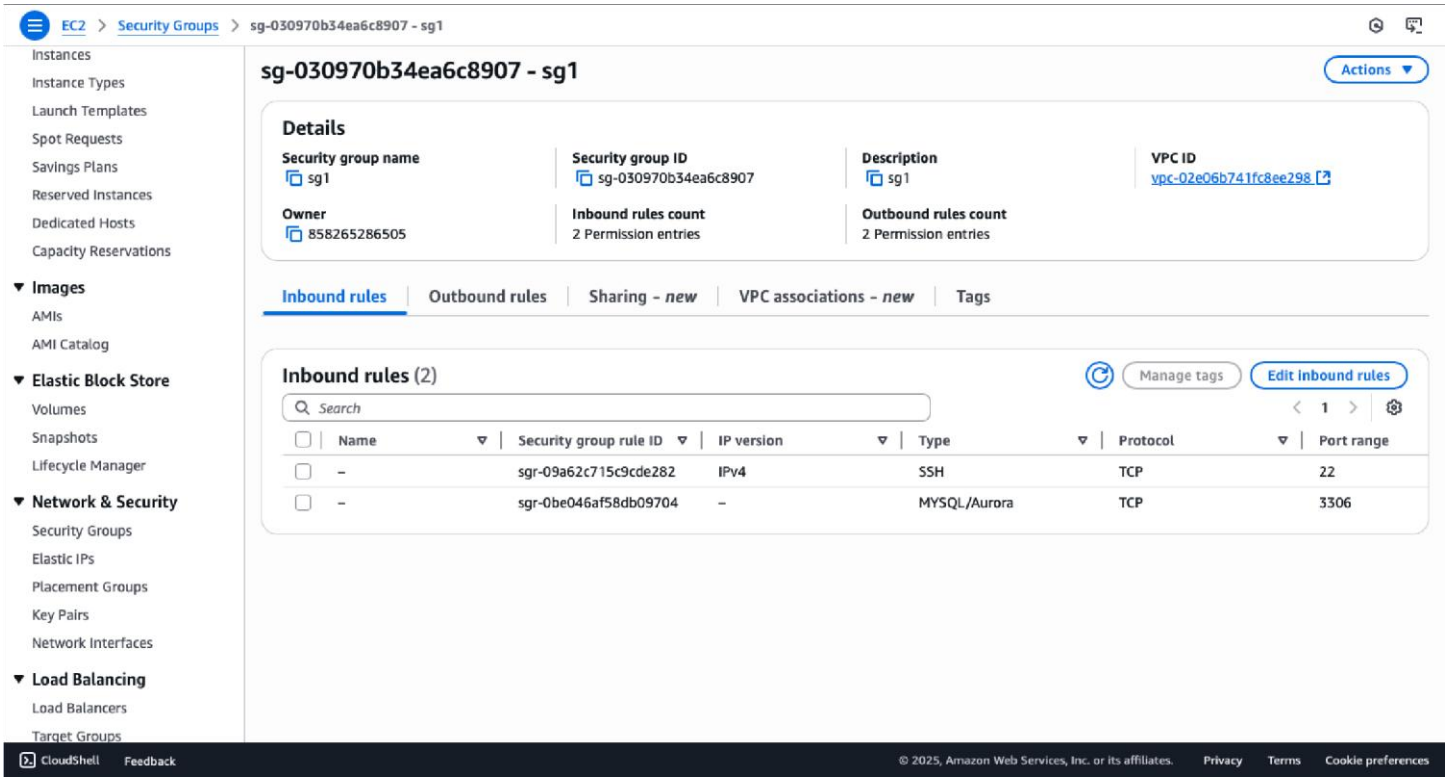
- **On RDS Security Group**:
  1. Go to **EC2 > Security Groups**.
  2. Find the RDS security group (e.g., rds-ec2-1).
  3. Edit inbound rules → Add rule:

- **Type**: MySQL/Aurora
- **Protocol**: TCP
- **Port**: 3306
- **Source**: Custom → Select your EC2's Security Group ●

On EC2 Security Group:

1. Add inbound rule:
   - **Type**: SSH
   - **Protocol**: TCP
   - **Port**: 22
   - **Source**: 0.0.0.0/0 (for testing; later restrict to your IP).

Now EC2 can reach RDS on port **3306**.



4. Connect to Your EC2 Instance via SSH

1. Open a terminal (Linux) or PowerShell (Windows).
2. Locate your key file: sonu-3.pem.
3. Set permissions so it's not publicly visible:
4. chmod 400 sonu-3.pem
5. Connect using EC2's **Public DNS**:
6. ssh -i "sonu-3.pem" ec2-user@ec2-54-174-158-139.compute-1.amazonaws.com
7. Once connected, you'll be inside your EC2 shell.

5. Install MySQL Client on EC2

On Amazon Linux 2023, run:

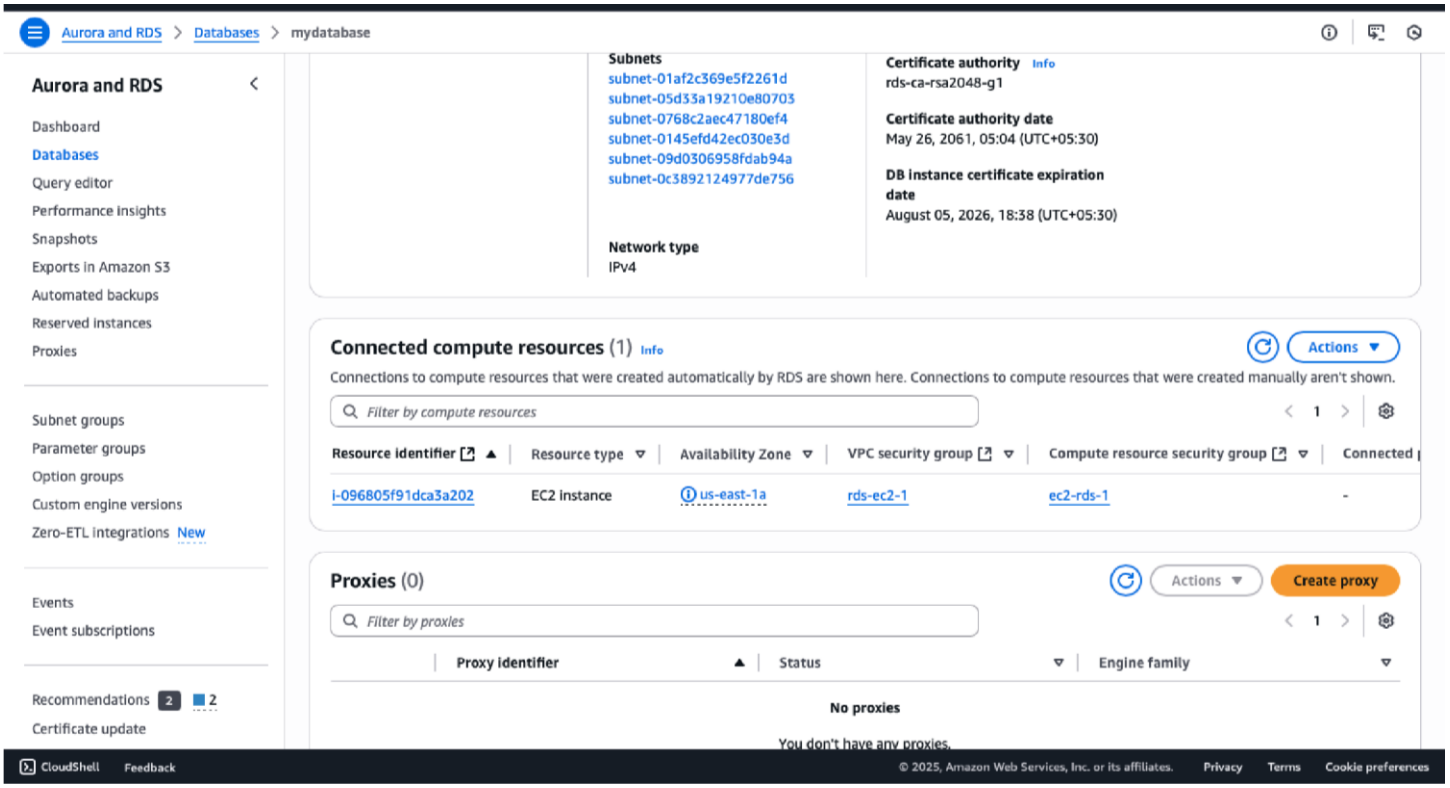
sudo dnf update -y sudo dnf

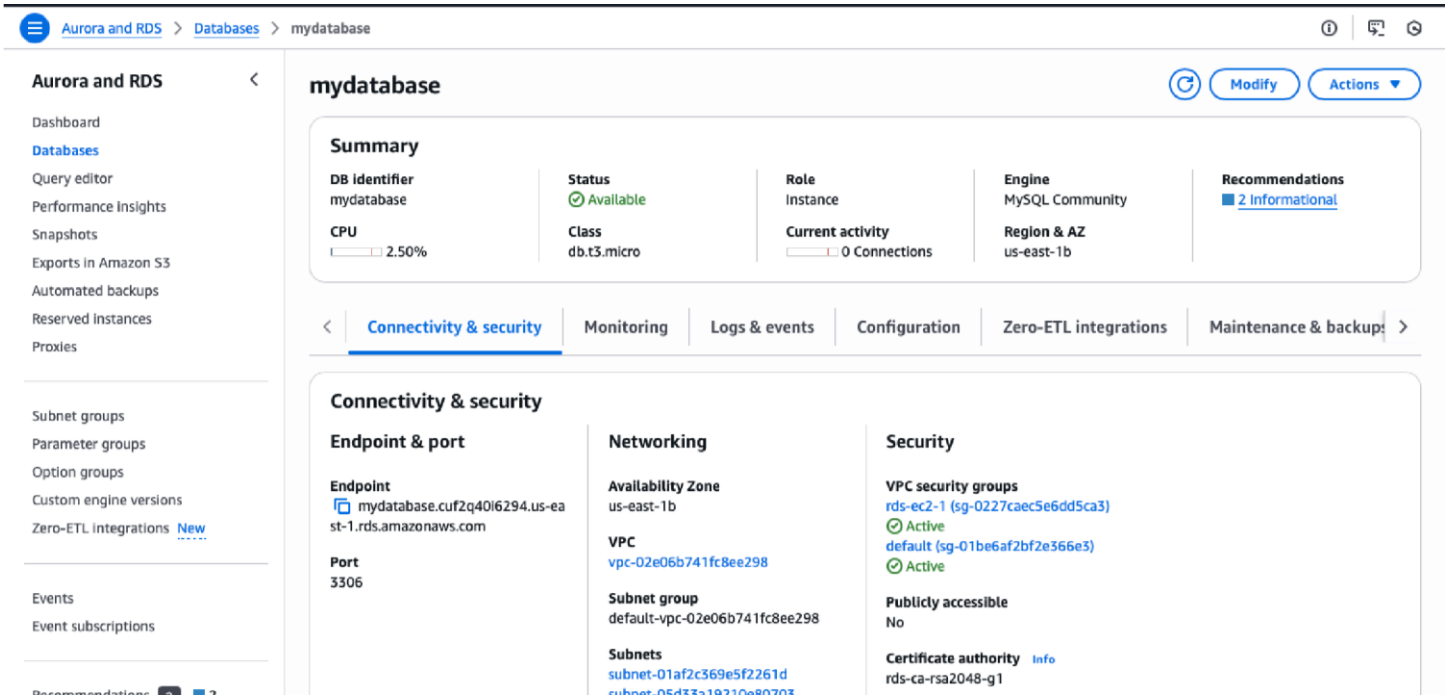
install -y mariadb105

This installs the MySQL/MariaDB client, which lets you connect to RDS.

6. Connect from EC2 to RDS

1. Go to **RDS > Databases > your database (mydatabase)**.
2. Copy the **endpoint** (e.g.,mydatabase.cuf2q40i6294.us-east-1.rds.amazonaws.com
3. From your EC2 terminal, connect:
4. mysql -h mydatabase.cuf2q40i6294.us-east-1.rds.amazonaws.com -u databasemysql -p
5. Enter your DB password when prompted.
6. You are now connected to your MySQL RDS database





7. Create a Database and Table

Inside the MySQL shell, run:

```
CREATE DATABASE company;
USE company;
```

```
CREATE TABLE users (     id INT
AUTO_INCREMENT PRIMARY KEY,     name
VARCHAR(50),     email VARCHAR(100)
);
```

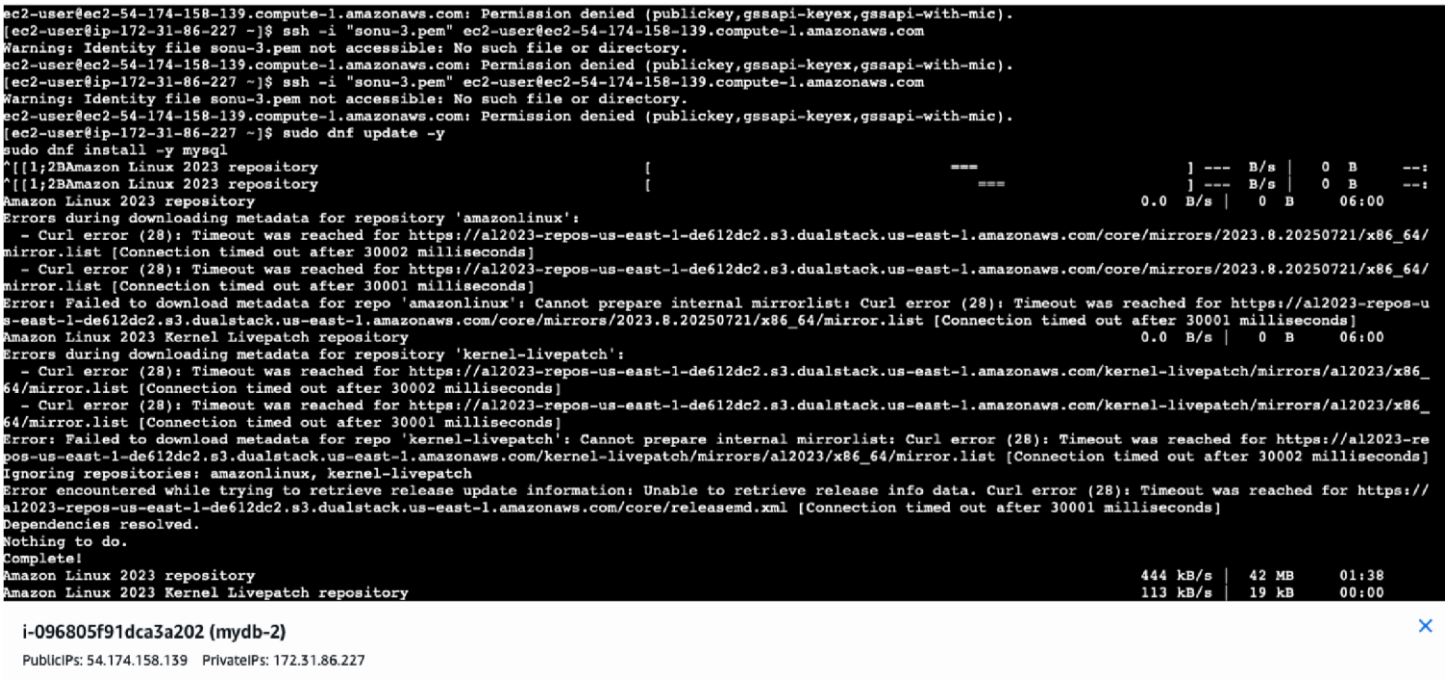
8. Insert and Verify Data

Insert some sample records:

```
INSERT INTO users (name, email) VALUES ('Alice', 'alice@example.com');
INSERT INTO users (name, email) VALUES ('Bob', 'bob@example.com');
INSERT INTO users (name, email) VALUES ('Charlie', 'charlie@example.com');
INSERT INTO users (name, email) VALUES ('Diana', 'diana@example.com');
```

Check the data:

SELECT * FROM users;



```
Complete!
[ec2-user@ip-172-31-86-227 ~]$ sudo yum install -y mysql
Last metadata expiration check: 0:00:56 ago on Mon Aug 18 20:49:20 2025.
No match for argument: mysql
Error: Unable to find a match: mysql
[ec2-user@ip-172-31-86-227 ~]$ sudo dnf install -y mariadb105
Last metadata expiration check: 0:01:34 ago on Mon Aug 18 20:49:20 2025.
Dependencies resolved.
================================================================================================================================
 Package                          Architecture          Version                              Repository                 Size
================================================================================================================================
Installing:
 mariadb105                       x86_64                3:10.5.29-1.amzn2023.0.1             amazonlinux               1.5 M
Installing dependencies:
 mariadb-connector-c              x86_64                3.3.10-1.amzn2023.0.1                amazonlinux               211 k
 mariadb-connector-c-config       noarch                3.3.10-1.amzn2023.0.1                amazonlinux               9.9 k
 mariadb105-common                x86_64                3:10.5.29-1.amzn2023.0.1             amazonlinux                28 k
 perl-Sys-Hostname                x86_64                1.23-477.amzn2023.0.7                amazonlinux                16 k

Transaction Summary
================================================================================================================================
Install  5 Packages

Total download size: 1.8 M
Installed size: 19 M
Downloading Packages:
(1/5): mariadb-connector-c-config-3.3.10-1.amzn2023.0.1.noarch.rpm                          204 kB/s | 9.9 kB     00:00
(2/5): mariadb-connector-c-3.3.10-1.amzn2023.0.1.x86_64.rpm                                 3.5 MB/s | 211 kB     00:00
(3/5): mariadb105-10.5.29-1.amzn2023.0.1.x86_64.rpm                                          18 MB/s | 1.5 MB     00:00
(4/5): mariadb105-common-10.5.29-1.amzn2023.0.1.x86_64.rpm                                  724 kB/s |  28 kB     00:00
(5/5): perl-Sys-Hostname-1.23-477.amzn2023.0.7.x86_64.rpm                                   347 kB/s |  16 kB     00:00
--------------------------------------------------------------------------------------------------------------------------------
Total                                                                                        13 MB/s | 1.8 MB     00:00
Running transaction check
Transaction check succeeded.
```

i-096805f91dca3a202 (mydb-2)

PublicIPs: 54.174.158.139   PrivateIPs: 172.31.86.227

```
  Verifying        : mariadb105-common-3:10.5.29-1.amzn2023.0.1.x86_64                                                     4/5
  Verifying        : perl-Sys-Hostname-1.23-477.amzn2023.0.7.x86_64                                                        5/5
================================================================================================================================
WARNING:
  A newer release of "Amazon Linux" is available.

  Available Versions:

  Version 2023.8.20250808:
    Run the following command to upgrade to 2023.8.20250808:

      dnf upgrade --releasever=2023.8.20250808

    Release notes:
     https://docs.aws.amazon.com/linux/al2023/release-notes/relnotes-2023.8.20250808.html

  Version 2023.8.20250818:
    Run the following command to upgrade to 2023.8.20250818:

      dnf upgrade --releasever=2023.8.20250818

    Release notes:
     https://docs.aws.amazon.com/linux/al2023/release-notes/relnotes-2023.8.20250818.html

================================================================================================================================

Installed:
  mariadb-connector-c-3.3.10-1.amzn2023.0.1.x86_64    mariadb-connector-c-config-3.3.10-1.amzn2023.0.1.noarch   mariadb105-3:10.5.29-1.amzn2023.0.1.x86_64
  mariadb105-common-3:10.5.29-1.amzn2023.0.1.x86_64   perl-Sys-Hostname-1.23-477.amzn2023.0.7.x86_64

Complete!
[ec2-user@ip-172-31-86-227 ~]$ mysql --version
mysql  Ver 15.1 Distrib 10.5.29-MariaDB, for Linux (x86_64) using  EditLine wrapper
[ec2-user@ip-172-31-86-227 ~]$ mysql -h mydatabase.cuf2q40i6294.us-east-1.rds.amazonaws.com -u mydatabase -p
Enter password:
```

i-096805f91dca3a202 (mydb-2)

PublicIPs: 54.174.158.139   PrivateIPs: 172.31.86.227

```
  Version 2023.8.20250818:
    Run the following command to upgrade to 2023.8.20250818:

      dnf upgrade --releasever=2023.8.20250818

    Release notes:
     https://docs.aws.amazon.com/linux/al2023/release-notes/relnotes-2023.8.20250818.html

================================================================================================================================================================

Installed:
  mariadb-connector-c-3.3.10-1.amzn2023.0.1.x86_64    mariadb-connector-c-config-3.3.10-1.amzn2023.0.1.noarch   mariadb105-3:10.5.29-1.amzn2023.0.1.x86_64
  mariadb105-common-3:10.5.29-1.amzn2023.0.1.x86_64   perl-Sys-Hostname-1.23-477.amzn2023.0.7.x86_64

Complete!
[ec2-user@ip-172-31-86-227 ~]$ mysql --version
mysql  Ver 15.1 Distrib 10.5.29-MariaDB, for Linux (x86_64) using  EditLine wrapper
[ec2-user@ip-172-31-86-227 ~]$ mysql -h mydatabase.cuf2q40i6294.us-east-1.rds.amazonaws.com -u mydatabase -p
Enter password:
ERROR 1045 (28000): Access denied for user 'mydatabase'@'172.31.86.227' (using password: YES)
[ec2-user@ip-172-31-86-227 ~]$ mysql -h mydatabase.cuf2q40i6294.us-east-1.rds.amazonaws.com -u mydatabase -p
Enter password:
ERROR 1045 (28000): Access denied for user 'mydatabase'@'172.31.86.227' (using password: YES)
[ec2-user@ip-172-31-86-227 ~]$ mysql -h mydatabase.cuf2q40i6294.us-east-1.rds.amazonaws.com -u mydatabase -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 1320
Server version: 8.0.41 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

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

MySQL [(none)]>
```

i-096805f91dca3a202 (mydb-2)

PublicIPs: 54.174.158.139 PrivateIPs: 172.31.86.227

```
MySQL [(none)]>
MySQL [(none)]> CREATE DATABASE testdb;
Query OK, 1 row affected (0.014 sec)

MySQL [(none)]>
MySQL [(none)]> USE testdb;
Database changed
MySQL [testdb]>
MySQL [testdb]> CREATE TABLE users (
    ->   id INT AUTO_INCREMENT PRIMARY KEY,
    ->   name VARCHAR(50) NOT NULL,
    ->   email VARCHAR(100) UNIQUE NOT NULL,
    ->   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    -> );
Query OK, 0 rows affected (0.063 sec)

MySQL [testdb]>
MySQL [testdb]> INSERT INTO users (name, email) VALUES ('Alice', 'alice@example.com');
Query OK, 1 row affected (0.009 sec)

MySQL [testdb]> INSERT INTO users (name, email) VALUES ('Bob', 'bob@example.com');
Query OK, 1 row affected (0.007 sec)

MySQL [testdb]>
MySQL [testdb]> SELECT * FROM users;
+----+-------+-------------------+---------------------+
| id | name  | email             | created_at          |
+----+-------+-------------------+---------------------+
|  1 | Alice | alice@example.com | 2025-08-18 21:02:26 |
|  2 | Bob   | bob@example.com   | 2025-08-18 21:02:26 |
+----+-------+-------------------+---------------------+
2 rows in set (0.001 sec)

MySQL [testdb]>
```

i-096805f91dca3a202 (mydb-2)

PublicIPs: 54.174.158.139 PrivateIPs: 172.31.86.227

```
MySQL [testdb]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| testdb             |
+--------------------+
5 rows in set (0.001 sec)

MySQL [testdb]> CREATE DATABASE projectdb;
Query OK, 1 row affected (0.006 sec)

MySQL [testdb]> USE projectdb;
Database changed
MySQL [projectdb]> CREATE TABLE users (
    ->   id INT AUTO_INCREMENT PRIMARY KEY,
    ->   name VARCHAR(50) NOT NULL,
    ->   email VARCHAR(100) UNIQUE NOT NULL,
    ->   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    -> );
Query OK, 0 rows affected (0.038 sec)

MySQL [projectdb]> INSERT INTO users (name, email) VALUES ('Alice', 'alice@example.com');
Query OK, 1 row affected (0.005 sec)

MySQL [projectdb]> INSERT INTO users (name, email) VALUES ('Bob', 'bob@example.com');
Query OK, 1 row affected (0.004 sec)

MySQL [projectdb]> INSERT INTO users (name, email) VALUES ('Charlie', 'charlie@example.com');
Query OK, 1 row affected (0.005 sec)

MySQL [projectdb]> INSERT INTO users (name, email) VALUES ('Diana', 'diana@example.com');
```

i-096805f91dca3a202 (mydb-2)

PublicIPs: 54.174.158.139 PrivateIPs: 172.31.86.227

2.Deploy an E-commerce Application on EC2 Install necessary dependencies (Node.js, MySQL, etc.). Clone the Evershop repository and configure the database connection. Start the application and make it accessible over the public IP

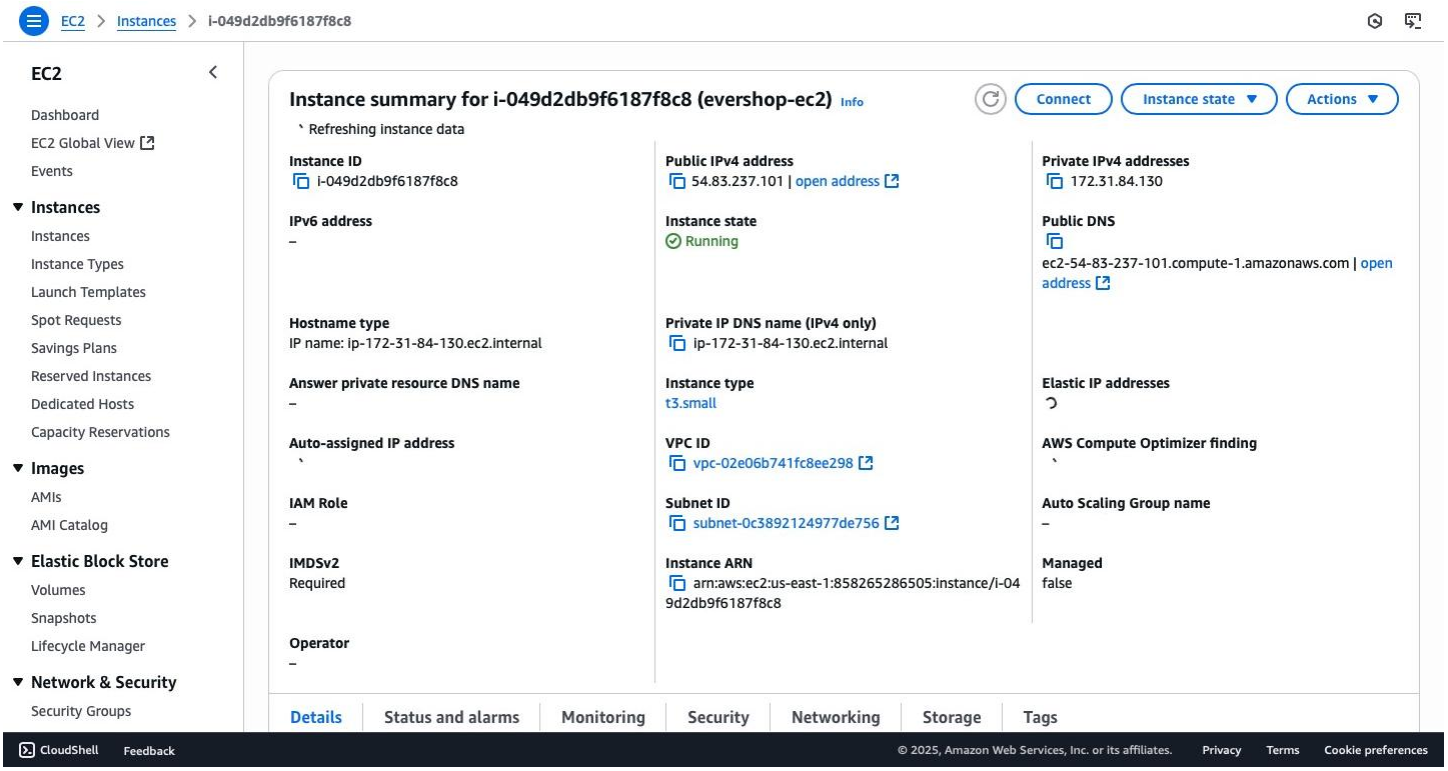
This is a clean, reproducible log of the exact steps we followed to deploy **EverShop** on an **Ubuntu EC2** instance,

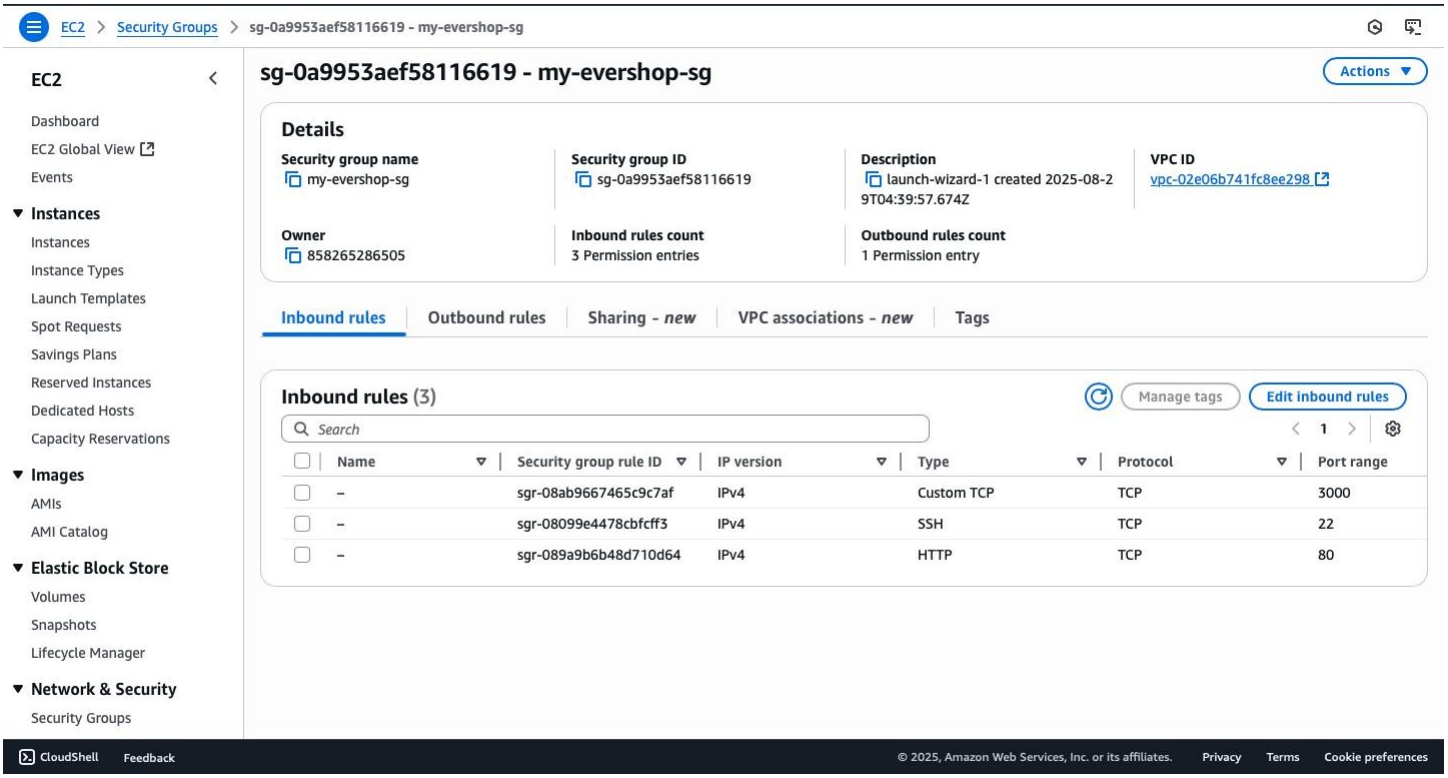
1) Launch EC2

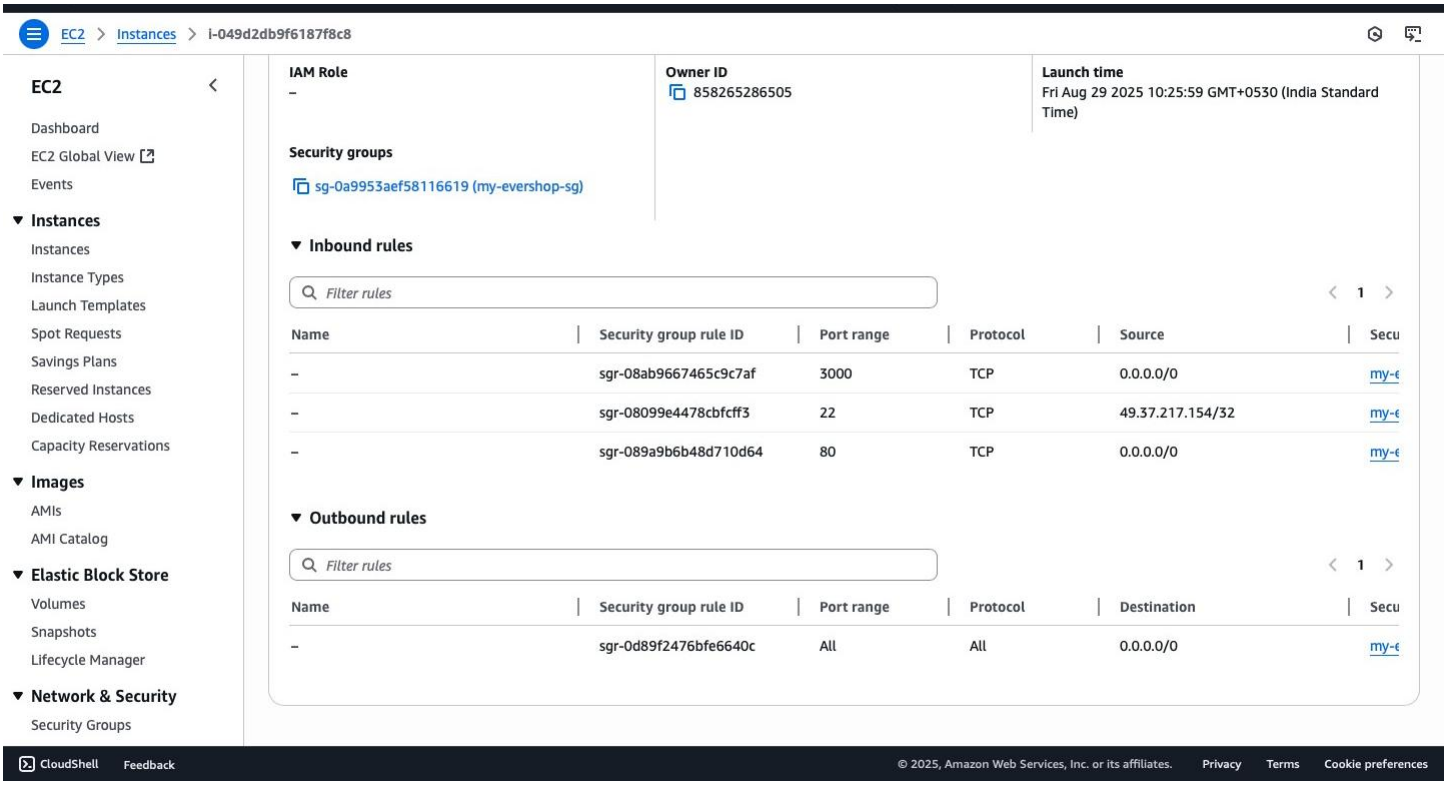
- **AMI**: Ubuntu Server 22.04 LTS
- **Instance type**: t3.small (or t3.micro for quick tests)
- **Key pair**: create/download a .pem
- **Security Group (inbound rules)**
  - SSH: 22/tcp from **your IP only** (e.g., 49.37.217.154/32)
  - HTTP: 80/tcp from 0.0.0.0/0
  - App test (optional): 3000/tcp from 0.0.0.0/0
- **Note** the **Public IPv4 address** (used as PUBLIC_IP below).

SSH in:

```
ssh -i "sonu-2.pem" ubuntu@ec2-54-83-237-101.compute-1.amazonaws.com
```







2) System prep

sudo apt update && sudo apt -y upgrade sudo

apt -y install git curl build-essential

3) Install Node.js 20 + npm

curl -fsSL https://deb.nodesource.com/setup_20.x | sudo -E bash sudo

apt -y install nodejs node -v npm -v

4) Install PostgreSQL

sudo apt -y install postgresql postgresql-contrib sudo

systemctl enable --now postgresql systemctl status

postgresql --no-pager

Create DB + user (we used Test1234 during setup — replace with a strong password):

sudo -u postgres psql -c "CREATE USER evershop WITH PASSWORD 'Test1234';" sudo -u

postgres psql -c "CREATE DATABASE evershop OWNER evershop;"

5) Create the EverShop app

Recommended scaffold (or you can clone your own repo):

mkdir -p ~/apps && cd ~/apps npx

create-evershop-app my-shop cd

~/apps/my-shop

Install dependencies:

npm install

6) Configure environment

Create .env in ~/apps/my-shop:

cat > .env <<'EOF'
DB HOST 1270 01
DB_HOST=127.0.0.1
DB_PORT=5432
DB_NAME=evershop
DB_USER=evershop
DB_PASSWORD=Test1234
DB_SSLMODE=disable

HOST=0.0.0.0
PORT=3000 EOF

Verify DB connectivity from the app host:

PGPASSWORD='Pass1234' psql \

"host=127.0.0.1 port=5432 dbname=evershop user=evershop sslmode=disable" \ -c '\\conninfo'

Expected output (example):

You are connected to database "evershop" as user "evershop" on host "127.0.0.1" at port "5432".

7) Build the app

If you hit Node.js heap OOM during build on small instances, add swap and increase Node heap:

Create 2G swap (one-time):

sudo fallocate -l 2G /swapfile sudo chmod 600 /swapfile sudo

mkswap /swapfile sudo swapon /swapfile echo '/swapfile

none swap sw 0 0' | sudo tee -a /etc/fstab

Build with more heap:

NODE_OPTIONS="--max-old-space-size=2048" npm run build

8) Start the app (foreground test)

npm run start

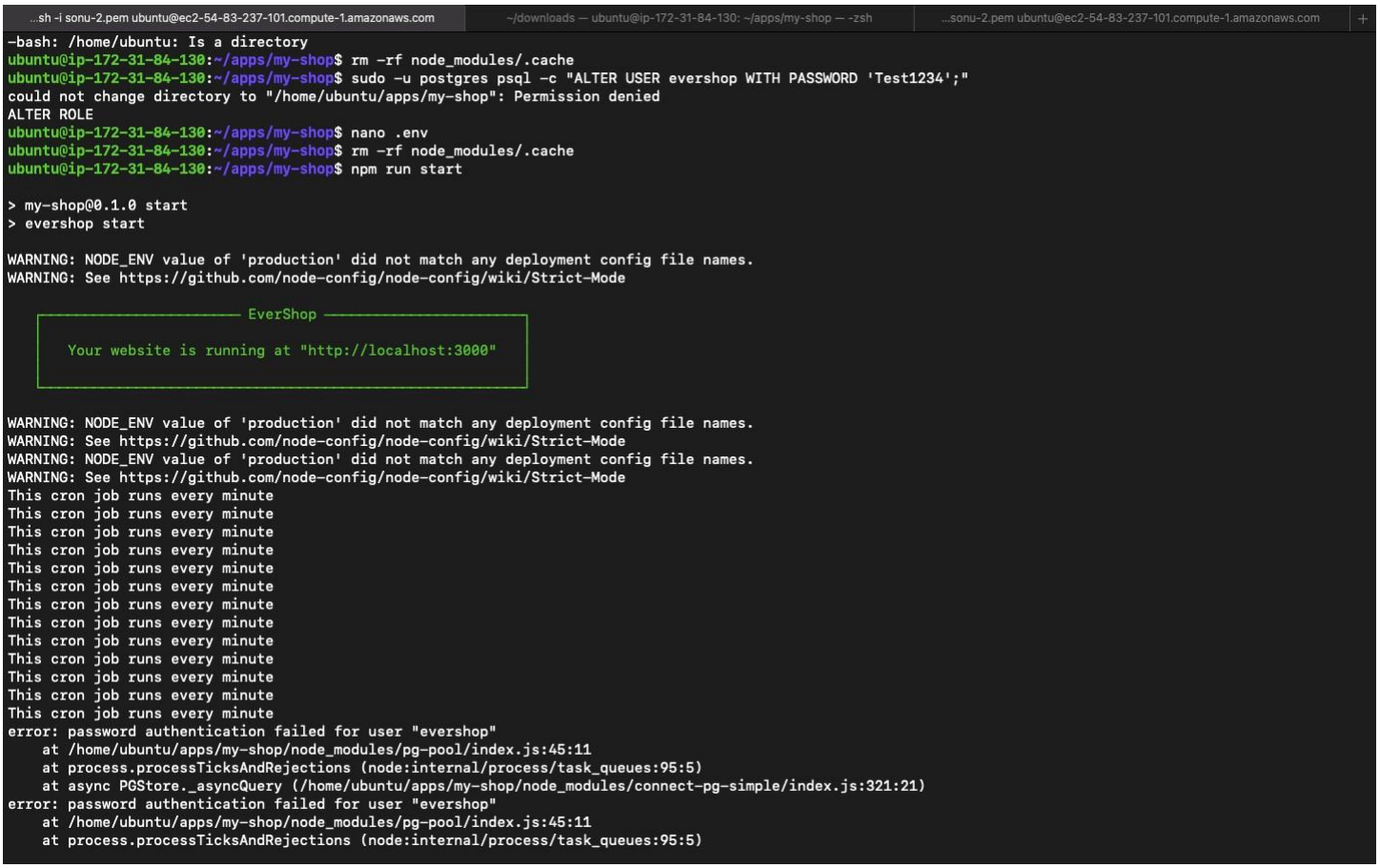
You should see something like:

Your website is running at "http://localhost:3000"

Because HOST=0.0.0.0, you can reach it from your computer at:
http://PUBLIC_IP:3000

If port is in use: sudo

lsof -i :3000 kill -9 <PID>



9) Create an admin user

Option A — EverShop CLI (preferred if available in your scaffold) npx

```
evershop user:create \
--name "Admin" \
--email admin@example.com \
--password 'Admin@123' \
--role admin
```

Option B — Direct SQL (works on any install)

Generate a bcrypt hash (example for Admin@123):

```
node -e "console.log(require('bcryptjs').hashSync(process.argv[1],10))" 'Admin@123'
# copy the printed hash into the SQL below as <BCRYPT_HASH>
```

Insert the admin row:

```
sudo -u postgres psql -d evershop -c "
INSERT INTO admin_user (email, password, status, full_name, created_at, updated_at) VALUES
('admin@example.com','<BCRYPT_HASH>', true, 'Admin', NOW(), NOW());"
```

Log in at:

http://PUBLIC_IP:3000/admin/login

http://54.83.237.101:3000

http://54.83.237.101

10) Keep the app running with PM2

Install and register PM2:

```
sudo npm install -g pm2 pm2 start "npm run
start" --name my-shop pm2 save pm2
startup systemd
# run the command PM2 prints, e.g.:
# sudo env PATH=$PATH:/usr/bin pm2 startup systemd -u ubuntu --hp /home/ubuntu
```

Check:

```
pm2 status pm2
logs my-shop
```

11) (Optional) NGINX reverse proxy (no :3000 in URL)

Install NGINX:

```
sudo apt -y install nginx
sudo nano /etc/nginx/sites-available/evershop
server {    listen 80;
  server_name 54.83.237.101 ec2-54-83-237-101.compute-1.amazonaws.com;
  location / {       proxy_pass http://127.0.0.1:3000;
proxy_http_version 1.1;       proxy_set_header
Upgrade $http_upgrade;       proxy_set_header
Connection 'upgrade';       proxy_set_header Host
$host;       proxy_cache_bypass $http_upgrade;
```

```
    }
}
```

Apply changes

Save the file.

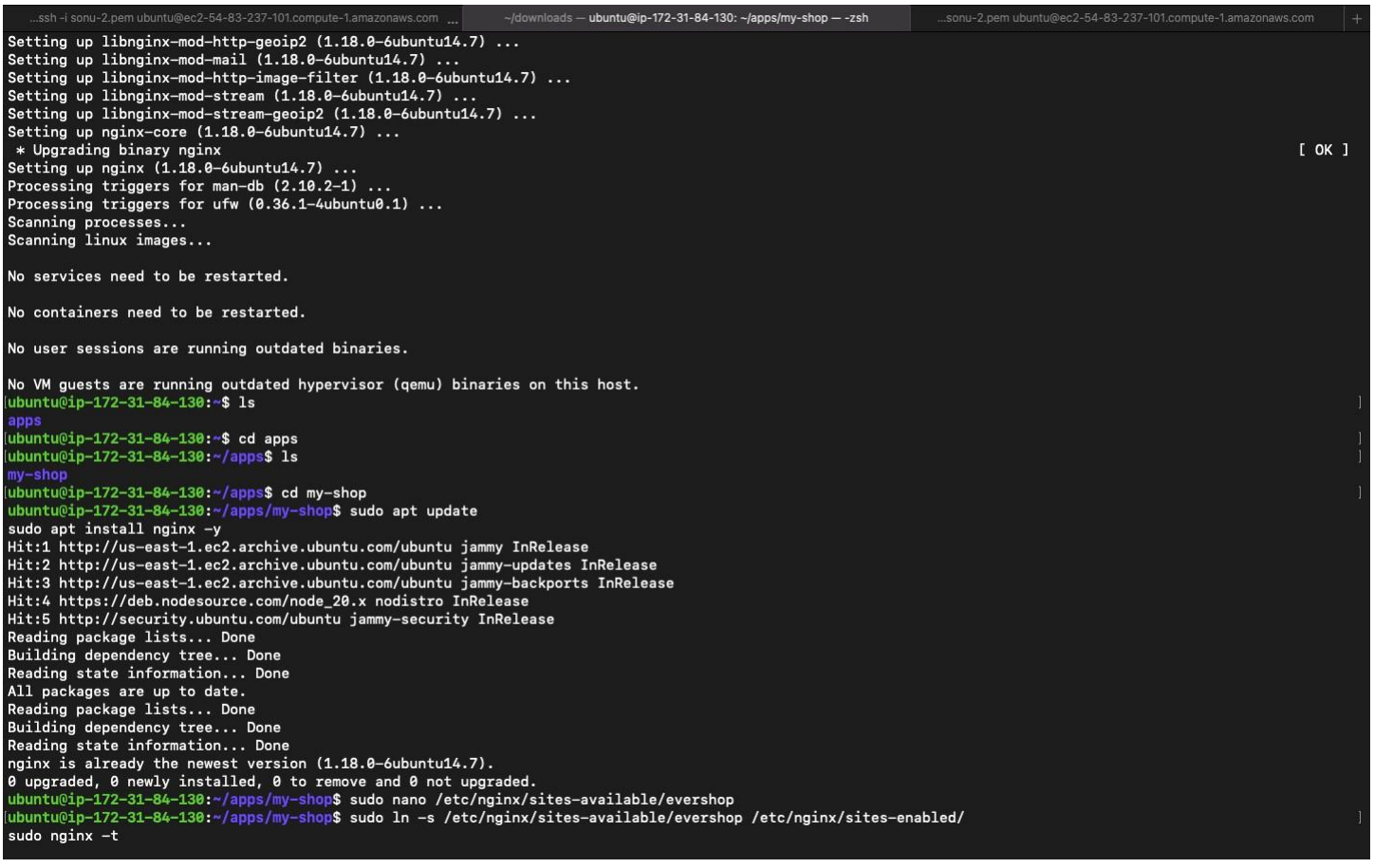
Test configuration:

```
sudo nginx -t
```

Reload Nginx:

```
sudo systemctl reload nginx
```

http://ec2-54-83-237-101.compute-1.amazonaws.com



12) commands

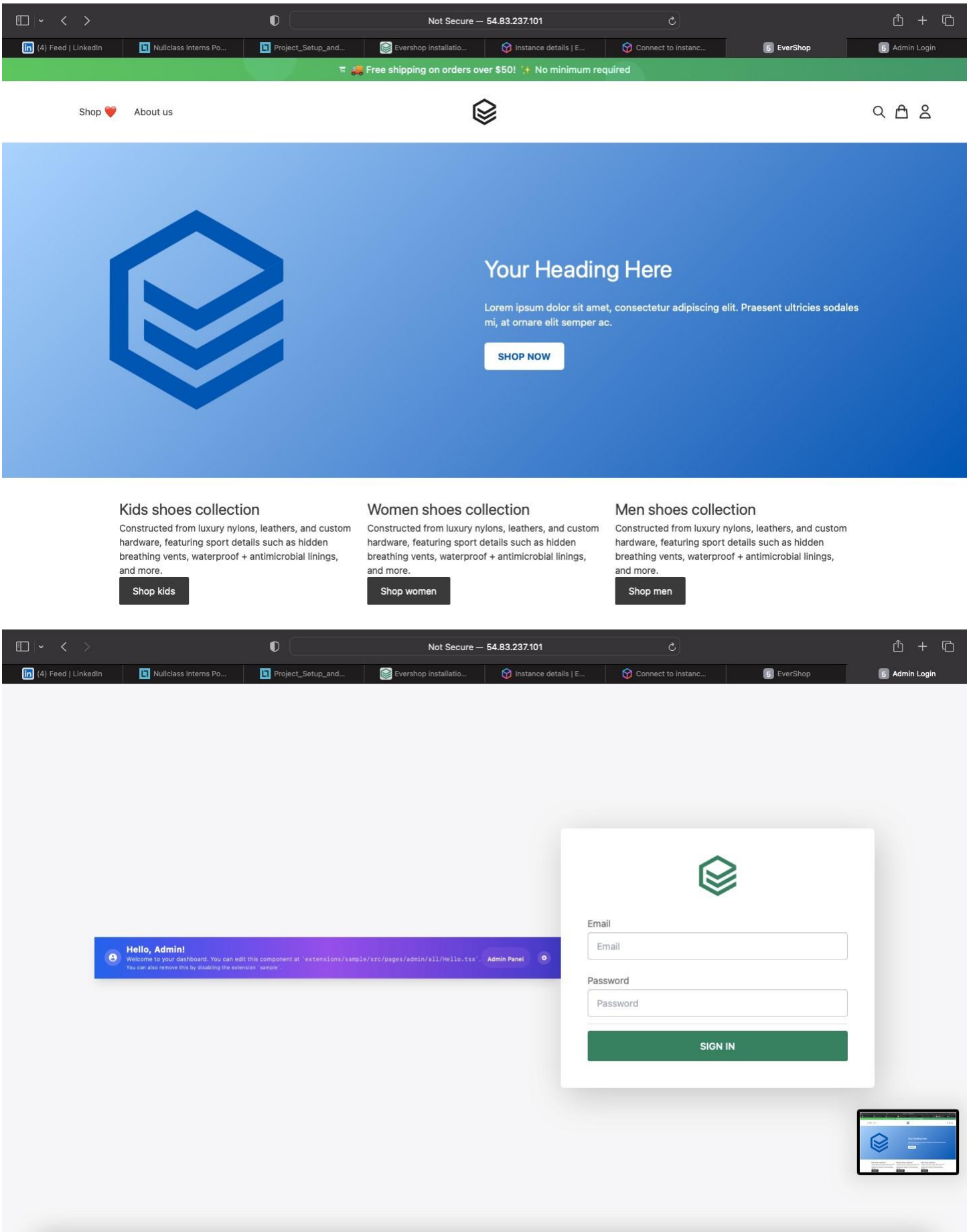
```
# App (PM2) pm2 status pm2

logs my-shop pm2 restart

my-shop # PostgreSQL

systemctl status postgresql

sudo -u postgres psql
```

DB backup

PGPASSWORD='Test1234' pg_dump -U evershop -h 127.0.0.1 evershop > ~/evershop-$(date +%F).sql # NGINX

sudo nginx -t sudo systemctl reload nginx

journalctl -u nginx --no-pager | tail -n 50


Not Secure — 54.83.237.101
(4) Feed | LinkedIn
Nullclass Interns Po...
Project_Setup_and...
Evershop installatio...
Instance details | E...
Connect to instanc...
EverShop
Admin Login
Free shipping on orders over $50! ✨ No minimum required
Shop ❤️
About us
Your Heading Here
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent ultricies sodales mi, at ornare elit semper ac.
SHOP NOW
Kids shoes collection
Constructed from luxury nylons, leathers, and custom hardware, featuring sport details such as hidden breathing vents, waterproof + antimicrobial linings, and more.
Shop kids
Women shoes collection
Constructed from luxury nylons, leathers, and custom hardware, featuring sport details such as hidden breathing vents, waterproof + antimicrobial linings, and more.
Shop women
Men shoes collection
Constructed from luxury nylons, leathers, and custom hardware, featuring sport details such as hidden breathing vents, waterproof + antimicrobial linings, and more.
Shop men
Not Secure — 54.83.237.101
(4) Feed | LinkedIn
Nullclass Interns Po...
Project_Setup_and...
Evershop installatio...
Instance details | E...
Connect to instanc...
EverShop
Admin Login
Hello, Admin!
Welcome to your dashboard. You can edit this component at `extensions/sample/src/pages/admin/all/Hello.tsx`.
You can also remove this by disabling the extension `sample`.
Admin Panel
Email
Email
Password
Password
SIGN IN

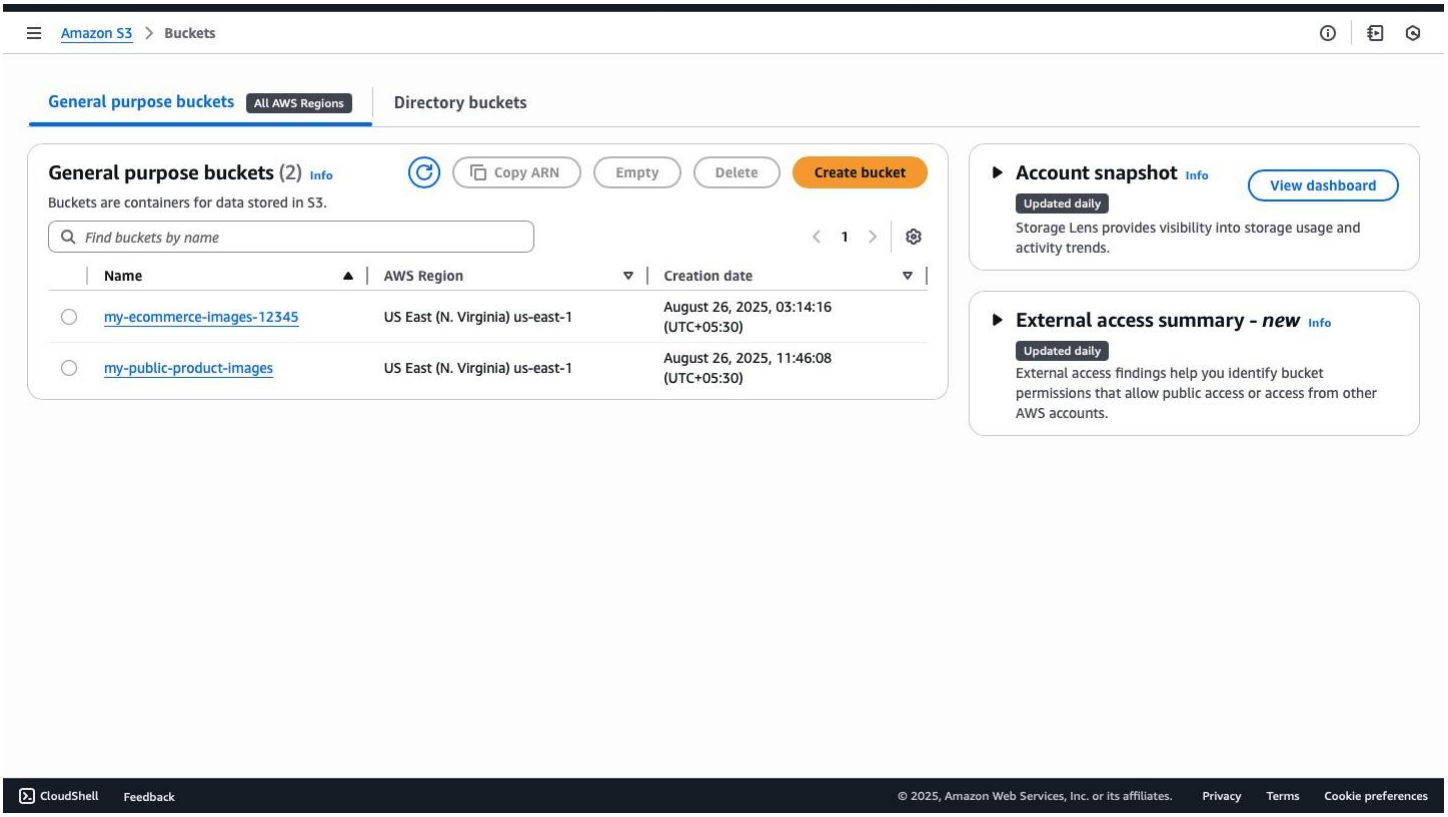
3.Implement an S3 Bucket for Storing Product Images Create an S3 bucket and enable public read access. Upload sample product images. Modify the e-commerce application to fetch images from S3 instead of the local server.

This guide explains how to create an Amazon S3 bucket with **public read access** for product images, upload sample images, and configure an e-commerce application to fetch images directly from S3 instead of the local server.

Step 1: Create an S3 Bucket

1. Log in to the **AWS Management Console**.
2. Navigate to **S3 → Create Bucket**.
3. Configure the bucket:
   - **Bucket Name**: my-public-product-images (must be globally unique).
   - **Region**: Choose the closest to your location (e.g., *US East – N. Virginia*).
   - **Bucket Type**: General Purpose.
   - **Object Ownership**: Bucket owner enforced.
   - **Block Public Access**: **Turn OFF** (required for public read access).
   - **Default Encryption**: SSE-S3 (Amazon-managed).
4. Click **Create Bucket**.

Outcome: An S3 bucket ready to store publicly accessible product images.

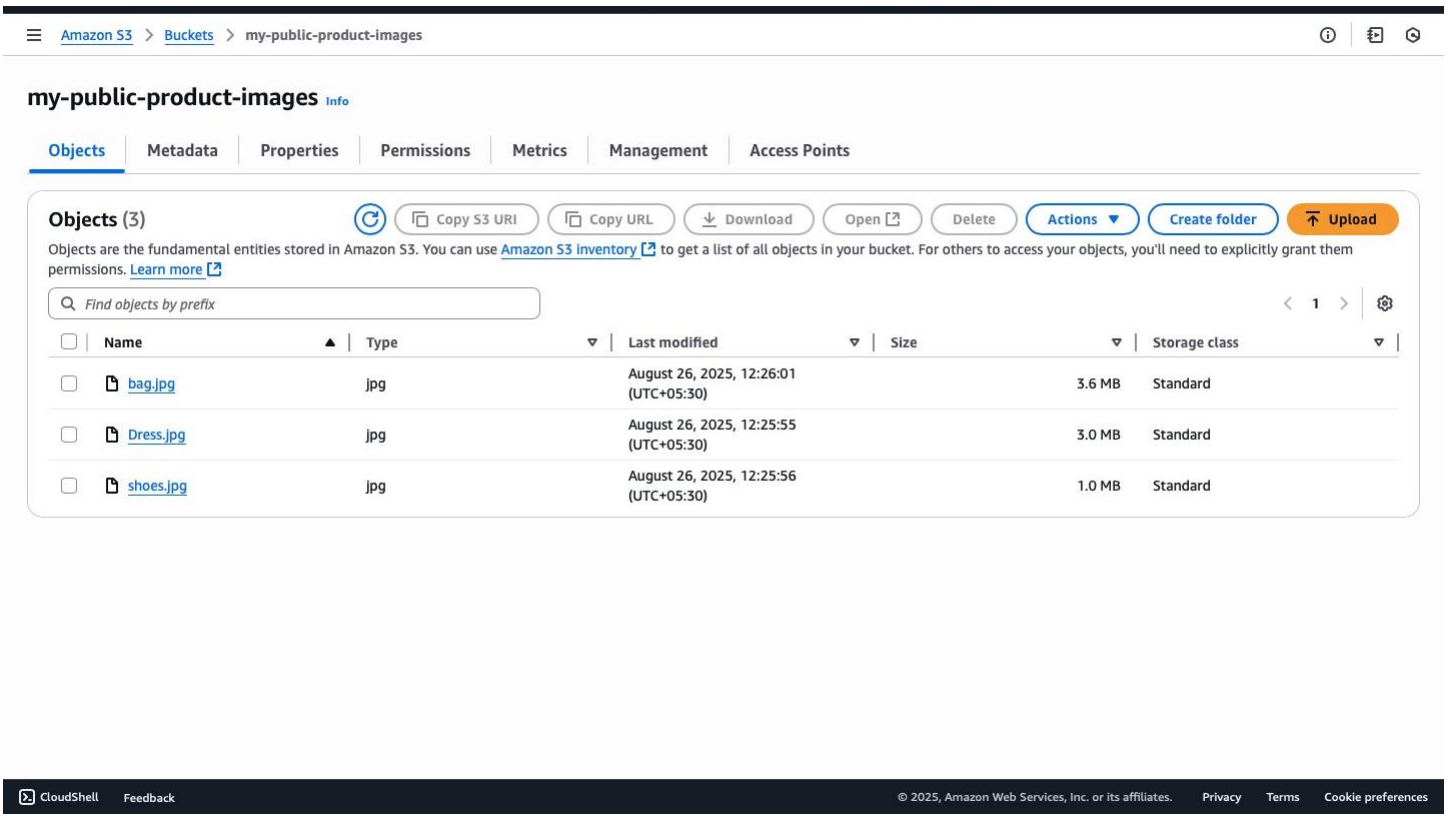


Step 2: Upload Product Images to S3

1. Open your bucket (my-public-product-images).
2. Click **Upload → Add files**.
3. Select product images (e.g., dress.jpg, shoes.jpg, bag.jpg).
4. Under **Permissions**, check **Grant public read access**.

5. Click **Upload**.

Outcome: Product images are now publicly accessible from your S3 bucket.



Step 3: Configure Bucket Policy for Public Read Access

Attach the following **bucket policy** to allow public read access to objects in the bucket:

"Version": "2012-10-17",

"Statement": [

{

"Sid": "PublicReadGetObject",

"Effect": "Allow",

"Principal": "*",

"Action": "s3:GetObject",

"Resource": "arn:aws:s3:::my-public-product-images/*"

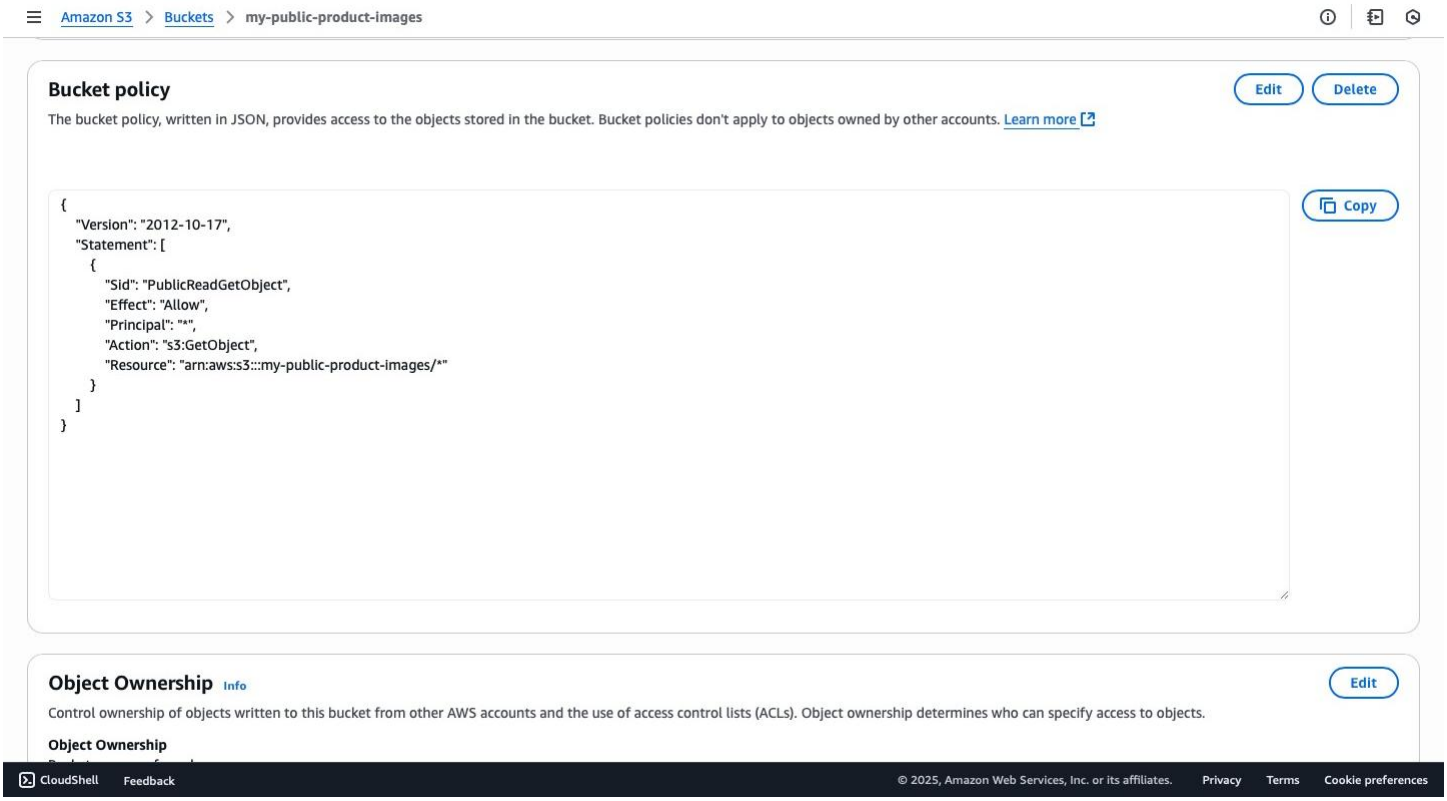
}
]

}

Replace:

- my-public-product-images with your actual bucket name.

Outcome: Anyone with the S3 object URL can view product images.



Step 4: Access Images via Public URLs

After uploading, each image will have a public URL in the format: https://my-public-product-images.s3.amazonaws.com/<filename>

Example:

https://my-public-product-images.s3.us-east-1.amazonaws.com/shoes.jpg https://my-public-product-images.s3.us-east-1.amazonaws.com/Dress.jpg https://my-public-product-images.s3.us-east-1.amazonaws.com/bag.jpg

shoes.jpg Info

Copy S3 URI | Download | Open | Object actions

Properties | Permissions | Versions

Object overview

Owner
9cb513322107863b27203aabce11f0d5d8646e5c789d4f9cbff37d7ee5621a0d

AWS Region
US East (N. Virginia) us-east-1

Last modified
August 26, 2025, 12:25:56 (UTC+05:30)

Size
1.0 MB

Type
jpg

Key
shoes.jpg

S3 URI
s3://my-public-product-images/shoes.jpg

Amazon Resource Name (ARN)
arn:aws:s3:::my-public-product-images/shoes.jpg

Entity tag (Etag)
a71ba6844629720ca78803599a6e690a

Object URL
https://my-public-product-images.s3.us-east-1.amazonaws.com/shoes.jpg

Object management overview

The following bucket properties and object management configurations impact the behavior of this object.

Bucket properties

Bucket Versioning

Management configurations

Replication status

Amazon S3

General purpose buckets
Directory buckets
Table buckets
Vector buckets
Access Grants
Access Points (General Purpose Buckets, FSx file systems)
Access Points (Directory Buckets)
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
IAM Access Analyzer for S3

Block Public Access settings for this account

Storage Lens
Dashboards
Storage Lens groups
AWS Organizations settings

Feature spotlight 11

Dress.jpg Info

Copy S3 URI | Download | Open | Object actions

Properties | Permissions | Versions

Object overview

Owner
sahanabraj234

AWS Region
US East (N. Virginia) us-east-1

Last modified
August 26, 2025, 12:25:55 (UTC+05:30)

Size
3.0 MB

Type
jpg

Key
Dress.jpg

S3 URI
s3://my-public-product-images/Dress.jpg

Amazon Resource Name (ARN)
arn:aws:s3:::my-public-product-images/Dress.jpg

Entity tag (Etag)
d10ca819d26d24e903cd4d3c7ab337d2

Object URL
https://my-public-product-images.s3.us-east-1.amazonaws.com/Dress.jpg

Object management overview

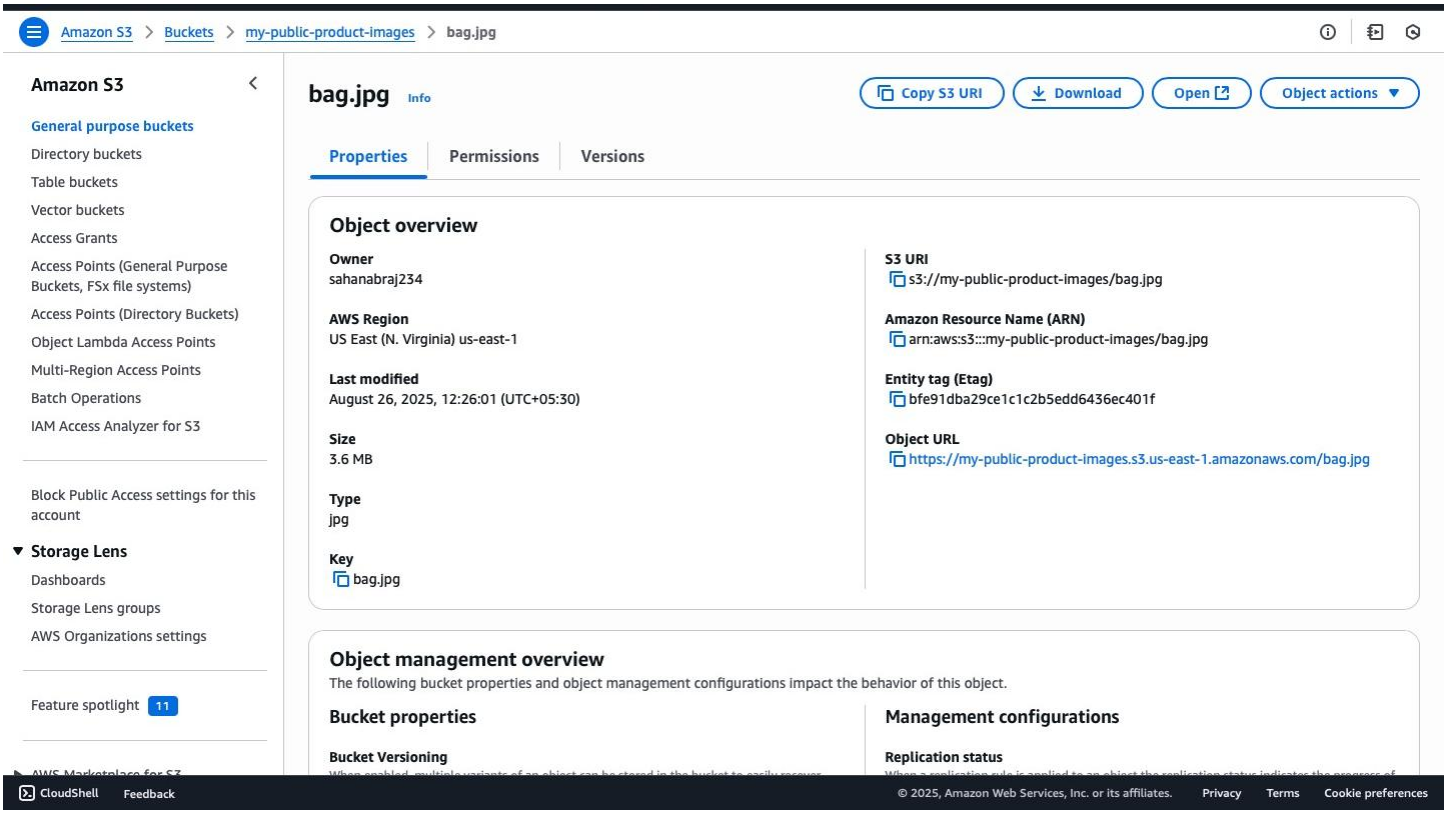
The following bucket properties and object management configurations impact the behavior of this object.

Bucket properties

Bucket Versioning

Management configurations

Replication status



Step 5: Modify the E-Commerce Application

Replace local image paths in your application with S3 URLs.

Before (local server):

<img src="/images/dress.jpg" alt="Dress"> **After**

(S3 bucket):

<img src="https://my-public-product-images.s3.amazonaws.com/dress.jpg" alt="Dress">

4.Configure Auto Scaling and Load Balancing Create a launch template for EC2 instances. Set up an auto-scaling group with minimum and maximum instance limits.
Attach an application load balancer to distribute traffic.

Auto Scaling Group + Load Balancer on AWS

This guide explains **from scratch** how to set up an Auto Scaling Group (ASG) with a Load Balancer (ALB) to run a simple website on EC2 instances.

We'll do this in 5 main parts:

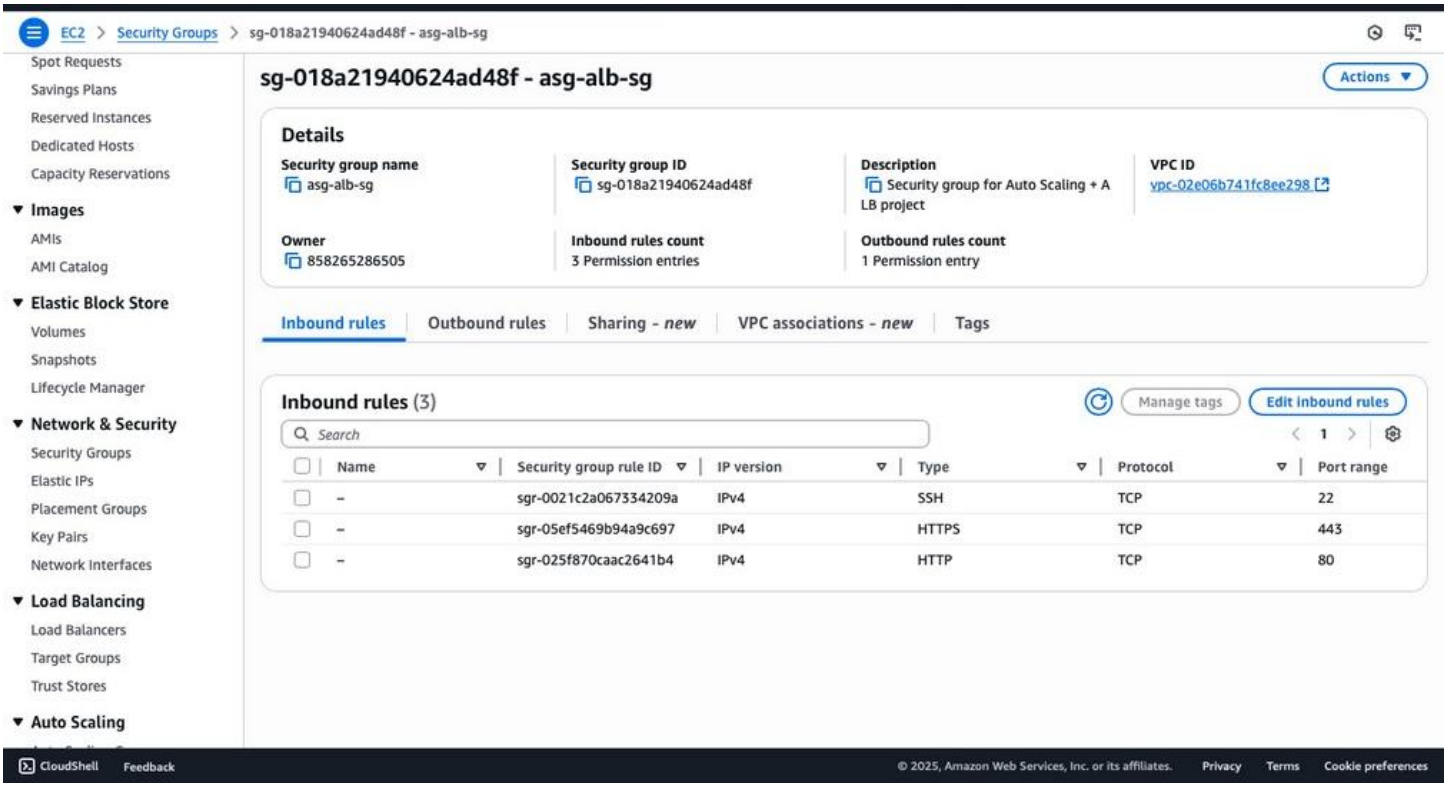
1. **Launch Template** → Blueprint for EC2.
2. **Security Group** → Firewall for EC2.
3. **Auto Scaling Group (ASG)** → Manages EC2 automatically.
4. **Application Load Balancer (ALB)** → Distributes traffic.
5. **Install Apache Web Server** → Host a webpage.

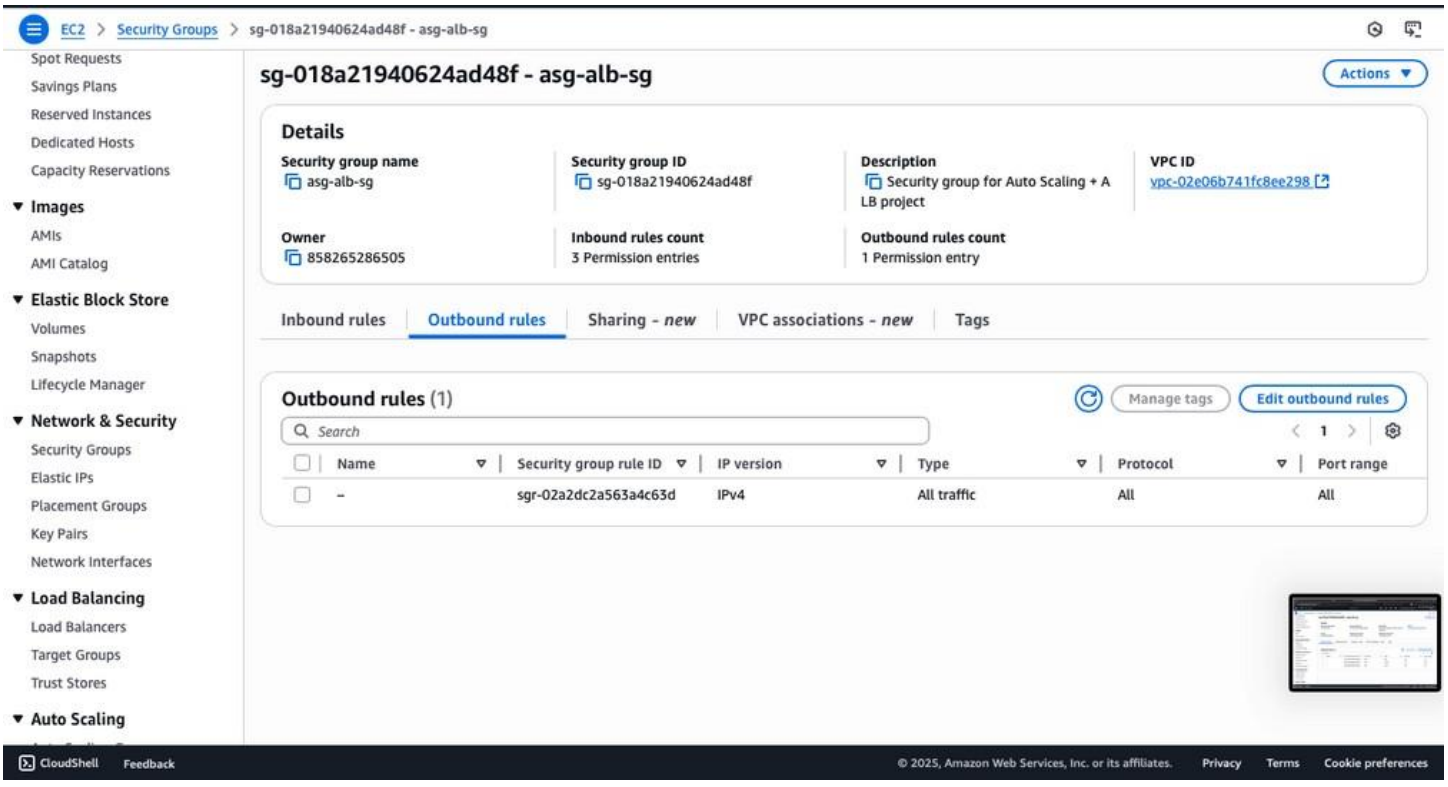
Step 1: Create a Security Group

A **Security Group** acts like a firewall. It controls what traffic can enter and leave your EC2 instance.

1. Go to **EC2 → Security Groups → Create Security Group**.
2. Fill details:
   - **Name:** asg-alb-sg
   - **Description:** Security Group for ASG + ALB
   - **VPC:** Default VPC
   3. Under **Inbound Rules**, add:
   - **SSH (22)** → Source: My IP (to connect to instance)
   - **HTTP (80)** → Source: Anywhere (0.0.0.0/0)
4. Leave **Outbound Rules** as default (all traffic allowed).
5. Click **Create Security Group**

This allows us to connect via SSH and serve a website.





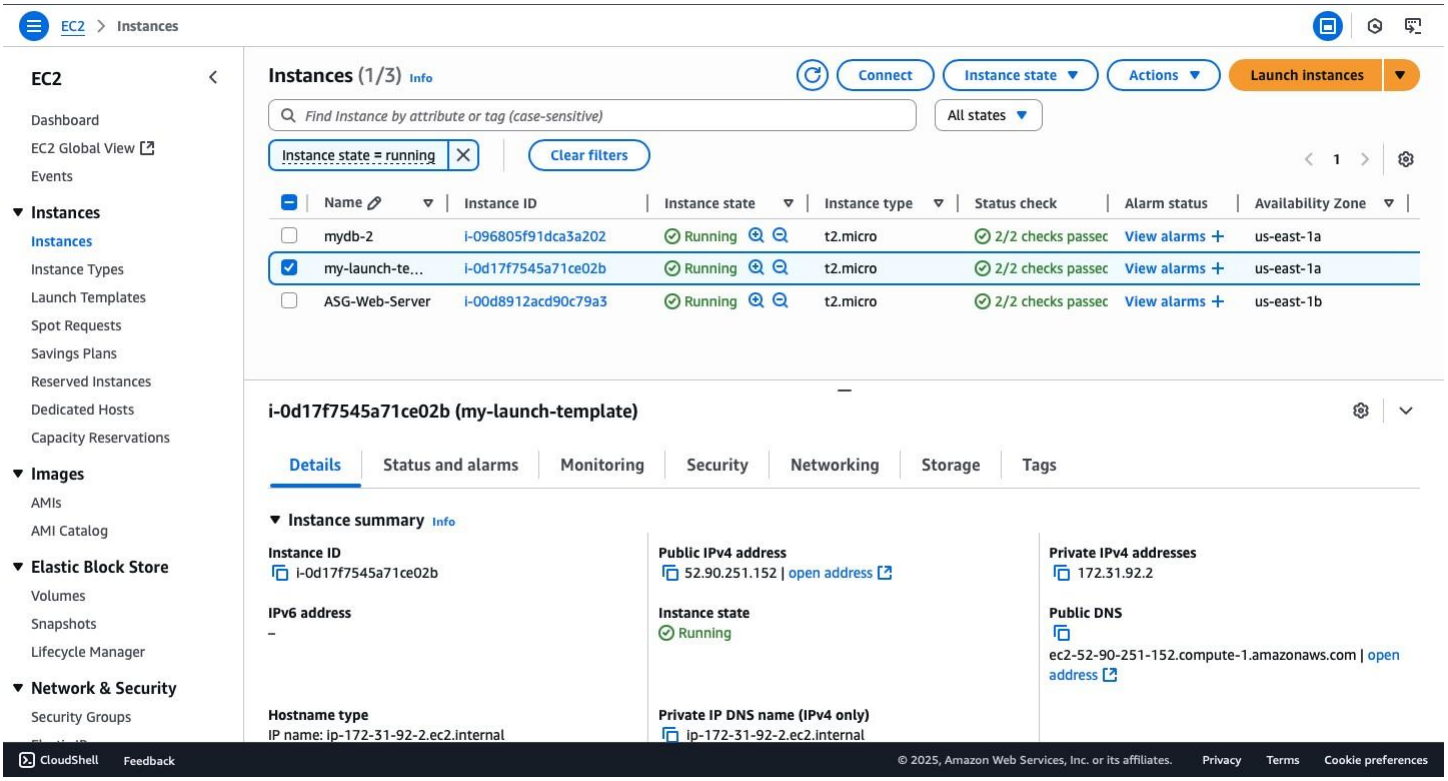
Step 2: Create a Launch Template

A **Launch Template** is a blueprint. It tells AWS how new EC2 instances should be created (OS, size, key, firewall).

1. Go to **EC2 → Launch Templates → Create Launch Template**.
2. Fill details:

- **Name:** my-launch-template
- **AMI (OS):** Amazon Linux 2023 (free tier eligible)
- **Instance type:** t2.micro
- **Key pair:** Choose your .pem key (example: sonu-3.pem)
- **Security Group:** Select asg-alb-sg (created above)
- **Storage:** 8GB (default EBS volume)

3. Click **Create Launch Template**



Now AWS knows how to launch new EC2 instances automatically.

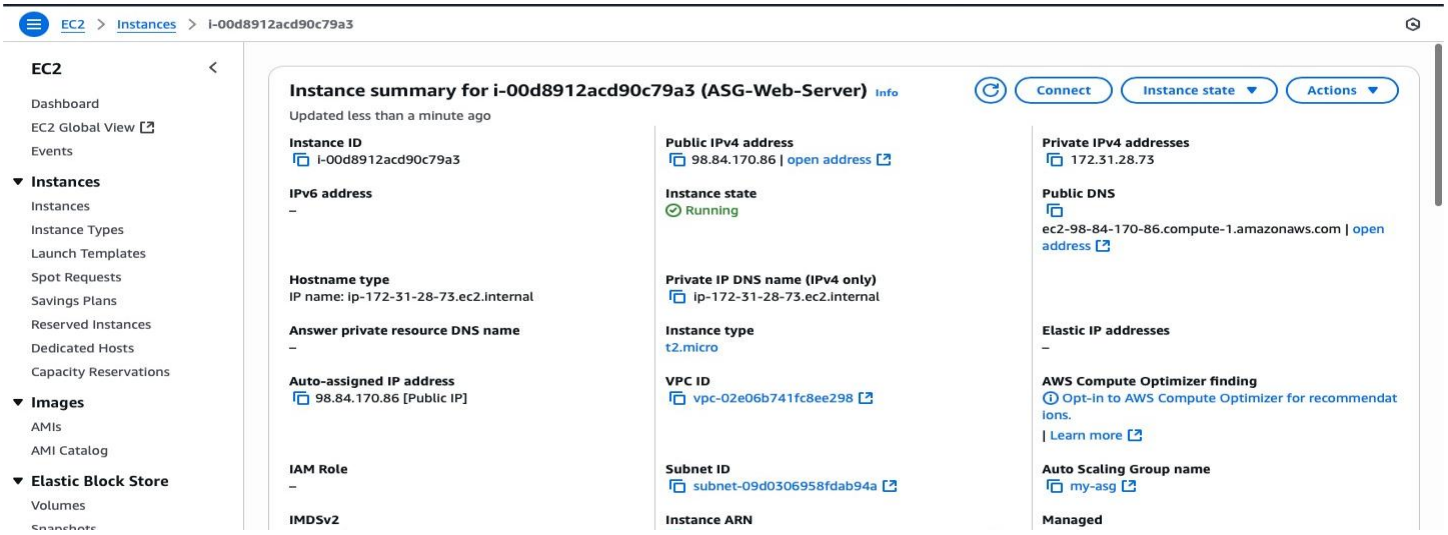
Step 3: Create an Auto Scaling Group (ASG)

An **Auto Scaling Group** ensures the right number of EC2 instances are always running. If one crashes, it creates a new one. If load increases, it adds more.

1. Go to **EC2 → Auto Scaling Groups → Create Auto Scaling Group**.
2. Select:
   - **Launch Template:** my-launch-template **Name:**
   - my-asg
3. Select **VPC:** Default VPC
   - Choose **at least 2 subnets** in different Availability Zones (e.g. us-east-1a and us-east-1b) → This ensures high availability.
4. Under **Load Balancing**, select:
   - **Attach to an existing load balancer** → (We'll create ALB next)

- Or choose **Create new ALB** directly here.

5. Configure **Group Size**:
   - **Desired capacity:** 1 (start with one instance)
   - **Min:** 1
   - **Max:** 3 (allows scaling up to 3 instances)
6. Add **Scaling Policy**:
   - **Target tracking policy** → Keep average CPU utilization at **50%**
7. Notifications (Optional):
   - Choose or create **SNS topic** → Get emails when instances launch/terminate.
8. Add **Tags**:
   - Example → Key: Name | Value: ASG-Web-Server



Click **Create Auto Scaling Group**

Now AWS will launch your first EC2 instance automatically.

Step 4: Create an Application Load Balancer (ALB)

A **Load Balancer** makes sure user traffic is distributed across healthy instances. If one fails, traffic goes to another.

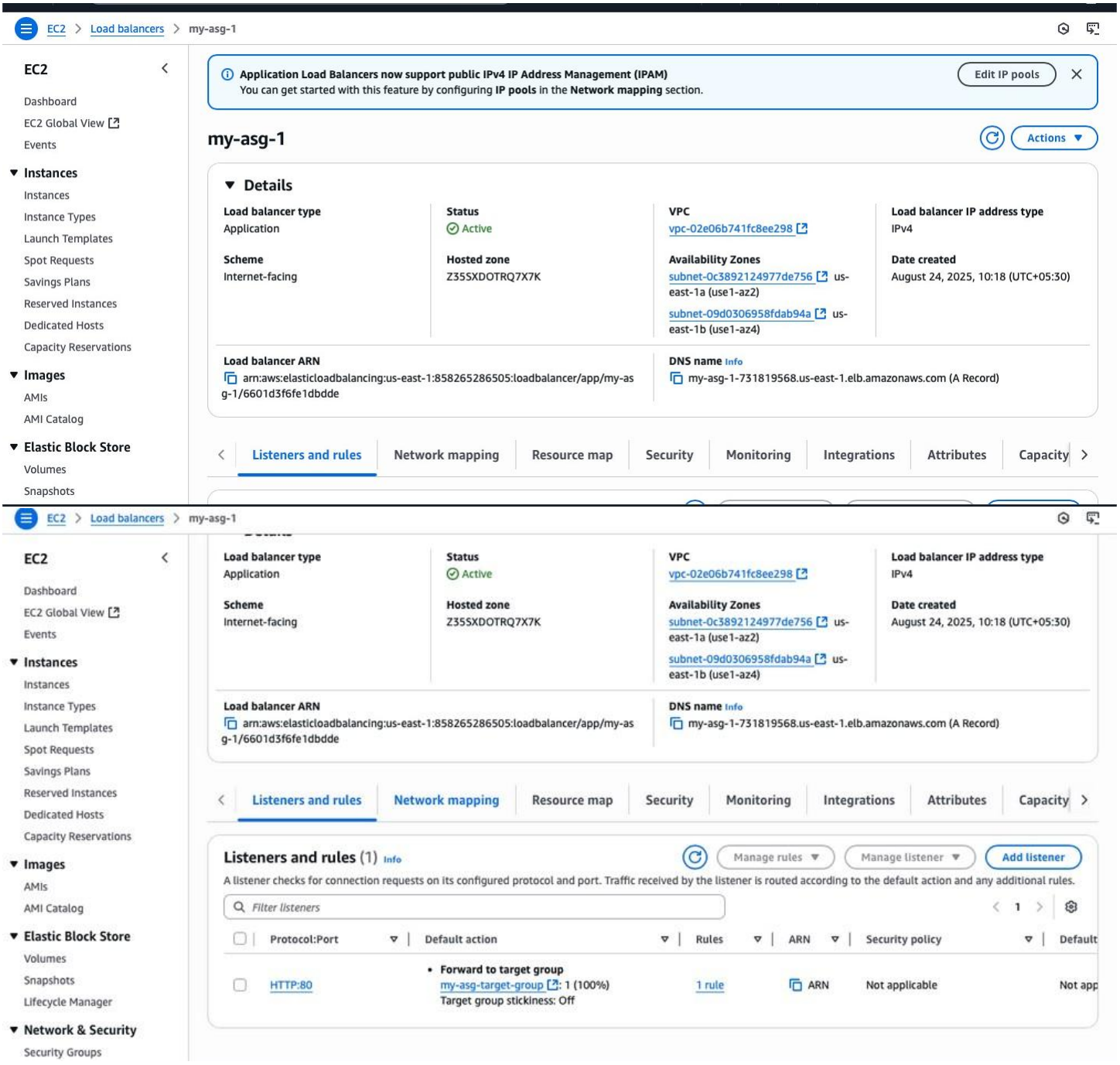
1. Go to **EC2 → Load Balancers → Create Load Balancer → Application Load Balancer**.
2. Fill details:
   - **Name:** my-asg-alb
   - **Scheme:** Internet-facing
   - **IP type:** IPv4
3. Network:
   - Choose **VPC:** Default
   - Select **2 subnets** in different Availability Zones (same as ASG).
4. Security Group:
   - Select **asg-alb-sg** (so it allows HTTP).
5. Listener:
   - Add **HTTP on port 80**.
6. Create **Target Group**: **Name:** my-asg-target-group
   -
   -

Target type: Instances

Protocol: HTTP:80

Health Check Path: / (root page)

7. Register instances → Select EC2 instances from your ASG.
8. Click **Create Load Balancer**



Now ALB sits in front of your ASG and distributes traffic.

Step 5: Install Apache on EC2 Instance

Now we'll install a **web server** (Apache) to serve a webpage.

1. Go to **EC2 → Instances → Select instance → Connect**
   (or SSH from terminal: ssh -i sonu-3.pem ec2-user@<Public-IP>).

2. Run these commands one by one: # Update all packages sudo yum

update -y

Install Apache (httpd) sudo

yum install -y httpd

Start Apache sudo systemctl

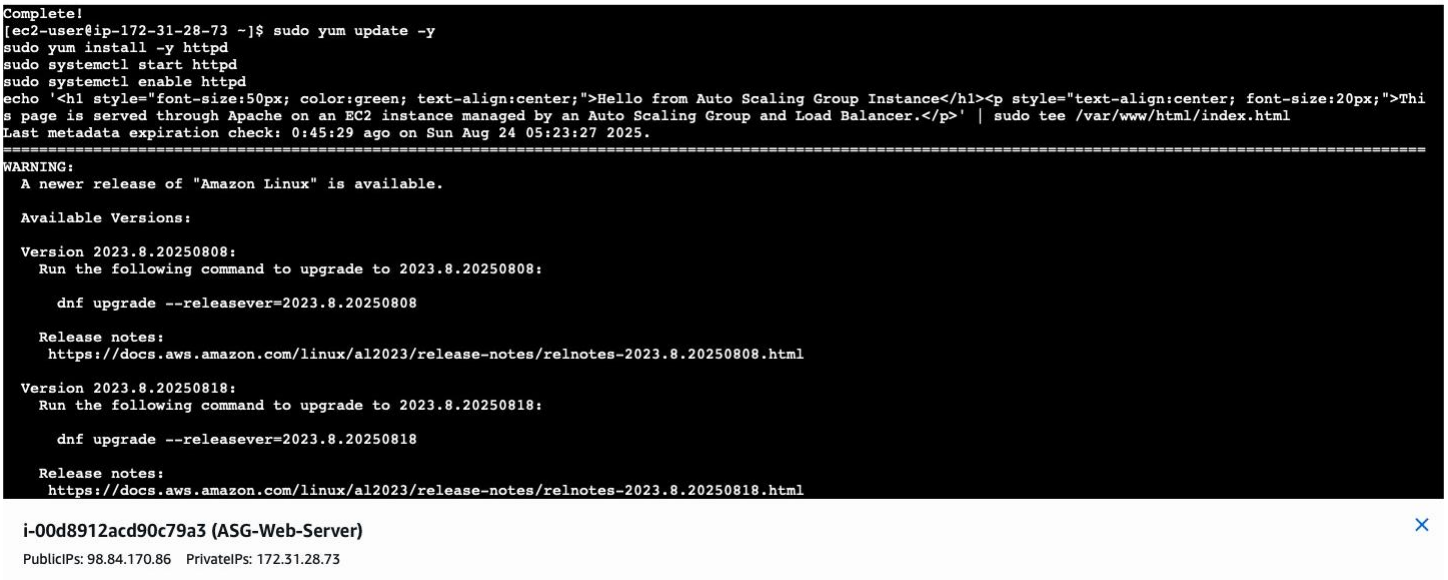
start httpd sudo systemctl

enable httpd

Create a custom HTML page

echo '<h1 style="font-size:50px; color:green; text-align:center;">Hello from Auto Scaling Group Instance</h1>

This page is served through Apache on an EC2 instance managed by an Auto Scaling Group and Load Balancer. ' | sudo tee /var/www/html/index.html



```
  Version 2023.8.20250808:
    Run the following command to upgrade to 2023.8.20250808:

      dnf upgrade --releasever=2023.8.20250808

    Release notes:
     https://docs.aws.amazon.com/linux/al2023/release-notes/relnotes-2023.8.20250808.html

  Version 2023.8.20250818:
    Run the following command to upgrade to 2023.8.20250818:

      dnf upgrade --releasever=2023.8.20250818

    Release notes:
     https://docs.aws.amazon.com/linux/al2023/release-notes/relnotes-2023.8.20250818.html

================================================================================================================================================================
Dependencies resolved.
Nothing to do.
Complete!
Last metadata expiration check: 0:45:30 ago on Sun Aug 24 05:23:27 2025.
Package httpd-2.4.62-1.amzn2023.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
<h1 style="font-size:50px; color:green; text-align:center;">Hello from Auto Scaling Group Instance</h1><p style="text-align:center; font-size:20px;">This page
 is served through Apache on an EC2 instance managed by an Auto Scaling Group and Load Balancer.</p>
[ec2-user@ip-172-31-28-73 ~]$
```

i-00d8912acd90c79a3 (ASG-Web-Server)

PublicIPs: 98.84.170.86 PrivateIPs: 172.31.28.73

Now your EC2 is serving a webpage.

Step 6: Test Your Setup

1. Copy the **DNS of your Load Balancer** (looks like my-asg-1-731819568.us-east-1.elb.amazonaws.com).
2. Open it in your browser.
3. You should see:

"**Hello from Auto Scaling Group Instance**" in **big green letters**.

If you refresh multiple times, ALB may send requests to different EC2 instances.

If one instance fails, ASG will replace it automatically.

Hello from Auto Scaling Group Instance

This page is served through Apache on an EC2 instance managed by an Auto Scaling Group and Load Balancer.

5.Configure Route 53 with Your Custom Domain and SSL using ACM Purchase a domain (or use an existing one) and configure it in Route 53. Request an SSL certificate using ACM and associate it with the load balancer. Ensure HTTPS traffic is properly routed to the application.

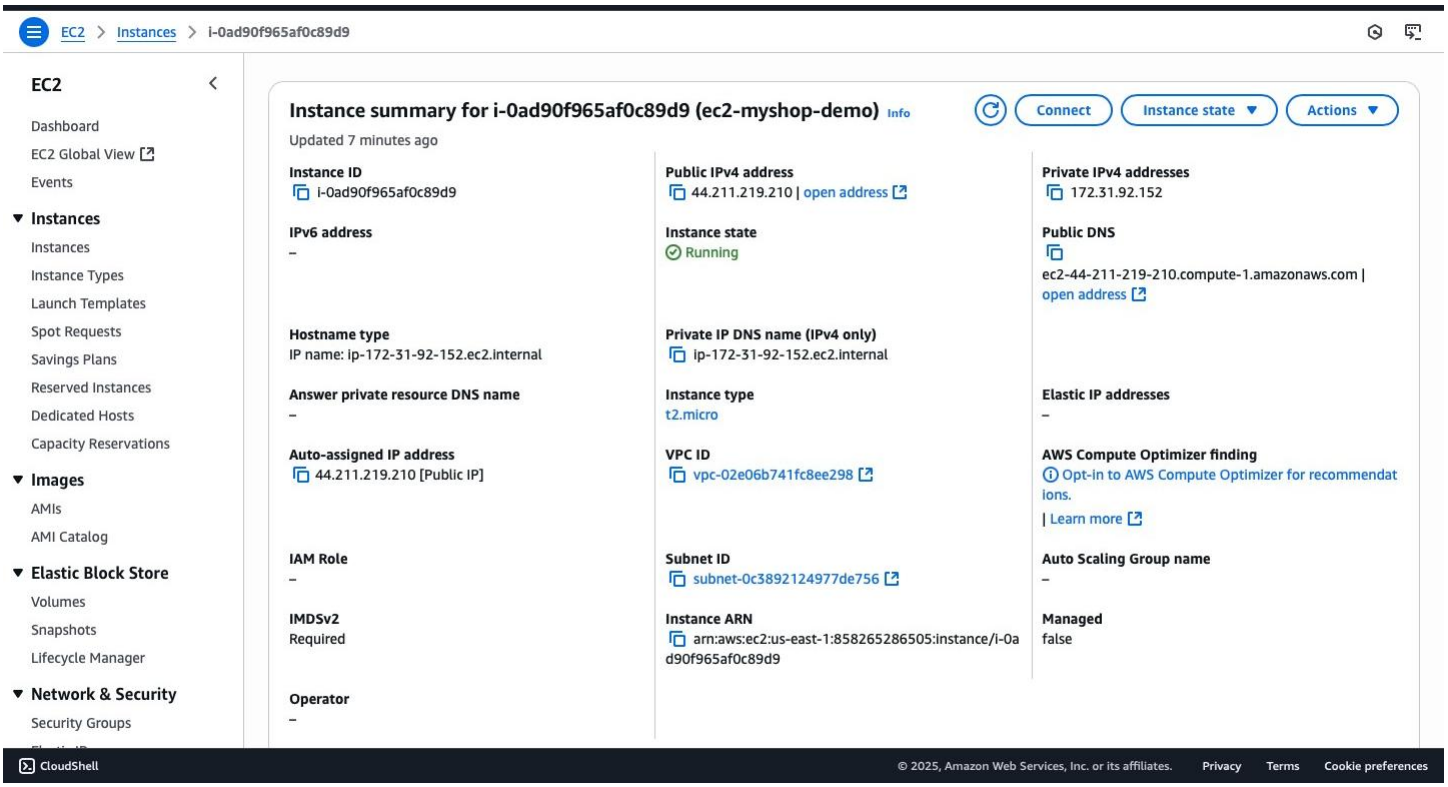
Secure Application Hosting on AWS (EC2 + ALB + CloudFront + ACM + Route 53)

This documentation explains step by step how to:

1. Launch an **EC2 instance** to host a simple web application.
2. Configure an **Application Load Balancer (ALB)** to distribute traffic.
3. Use **CloudFront** to provide global delivery and free HTTPS.
4. Request and attach an SSL certificate with **AWS Certificate Manager (ACM)**.
5. Configure **Route 53** with a custom domain in production.
6. Attach the ACM certificate directly to the **ALB HTTPS listener** (as the requirement states).

With this demo, you'll achieve secure access (HTTPS) using AWS's free CloudFront domain. For realworld production, you would purchase a domain, configure Route 53, request ACM certificates, and associate them with the ALB. **Step 1: Launch an EC2 Instance**

1. **Go to EC2 → Launch Instance.**
2. **Name: ec2-myshop-demo**
3. **AMI: Amazon Linux 2 (free tier eligible)** 4. **Instance type: t2.micro (free tier eligible)**
5. **Key pair: Select or create a key pair.**
6. **Network settings: VPC: default VPC**
   - **Subnet: choose a public subnet**
   - **Auto-assign public IP: enabled**
   - **Security group: create sg-ec2-web with inbound rules:**
   - **HTTP (80) from 0.0.0.0/0**
     - **SSH (22) from your IP or 0.0.0.0/0 (optional, for admin)**
7. **Launch instance.**



install a web server (after SSH into EC2):

sudo yum update -y sudo yum install -y httpd sudo systemctl start httpd sudo

systemctl enable httpd echo "<h1>Hello from EC2 backend</h1>" | sudo tee

/var/www/html/index.html

or

HTML

```
<!DOCTYPE html>

<html lang="en">

<head>

 <meta charset="UTF-8">

 <meta name="viewport" content="width=device-width, initial-scale=1.0">

 <title>MyShop Demo</title>
 <style>    body {      margin: 0;      font-family: Arial, sans-

serif;      background: linear-gradient(120deg, #4facfe,

#00f2fe);      color: #333;      display: flex;      flex-direction:

column;      justify-content: center;      align-items: center;

height: 100vh;      text-align: center;

  }    h1 {      font-size: 3em;

margin-bottom: 0.2em;

    color: #fff;

    text-shadow: 2px 2px 5px rgba(0,0,0,0.3);

  }    p {      font-

size: 1.2em;

color: #fefefe;

  }

  .card {
```

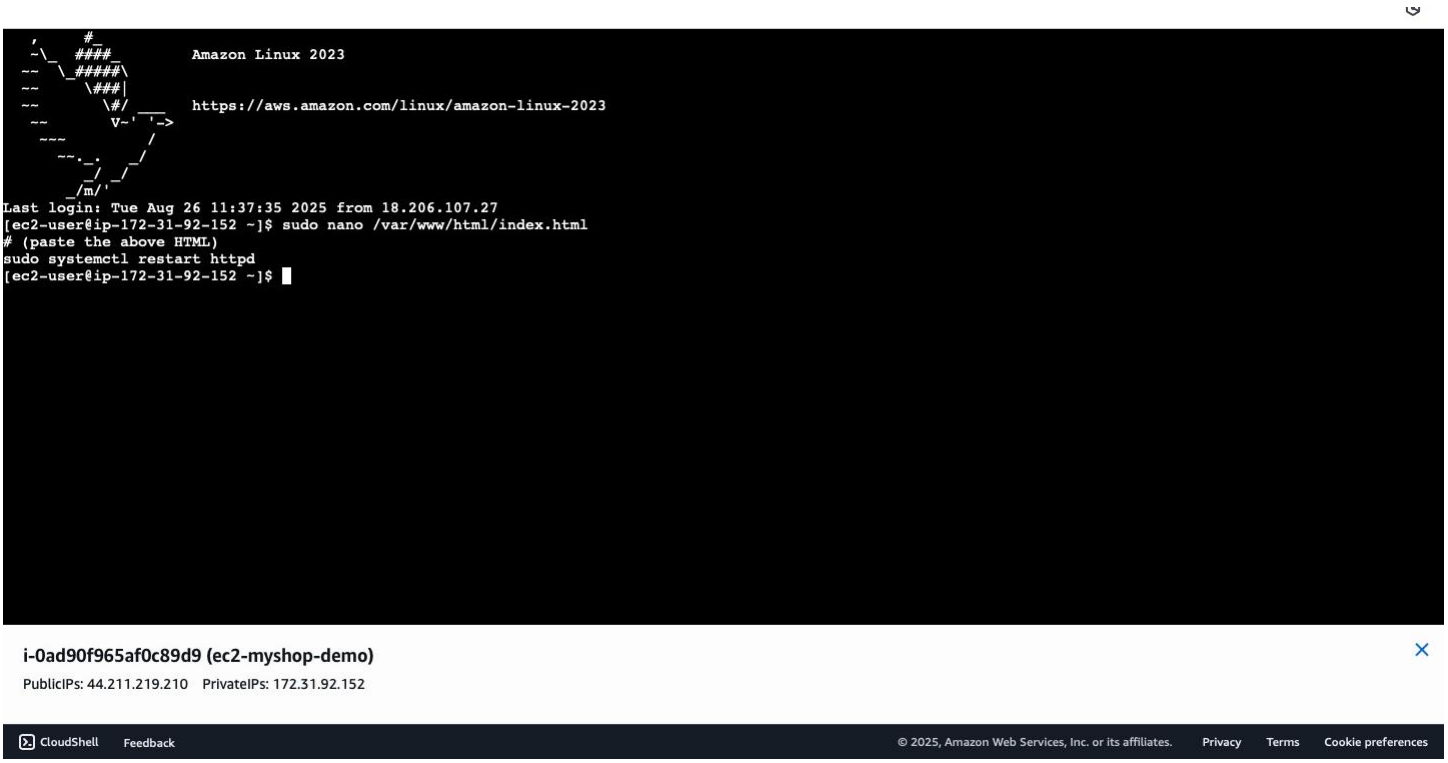
```
    background: rgba(255,255,255,0.15);
padding: 20px 40px;      border-radius:
12px;
    backdrop-filter: blur(6px);      box-shadow:
0 8px 16px rgba(0,0,0,0.2);
  }
  footer {      position:
absolute;      bottom:
20px;      font-size:
0.9em;      color:
#fefefe;
  }
 </style>
</head>
<body>
 <div class="card">
  <h1>🚀 Welcome to MyShop Demo</h1>
  <p>Your EC2 instance is running successfully behind an ALB + CloudFront!</p>
 </div>
 <footer>
  Powered by <strong>AWS EC2</strong> | Secure with <strong>CloudFront & ACM</strong>  </footer>
</body>
</html>
```

On your EC2:

sudo nano /var/www/html/index.html

 # (paste the above HTML) sudo

systemctl restart httpd

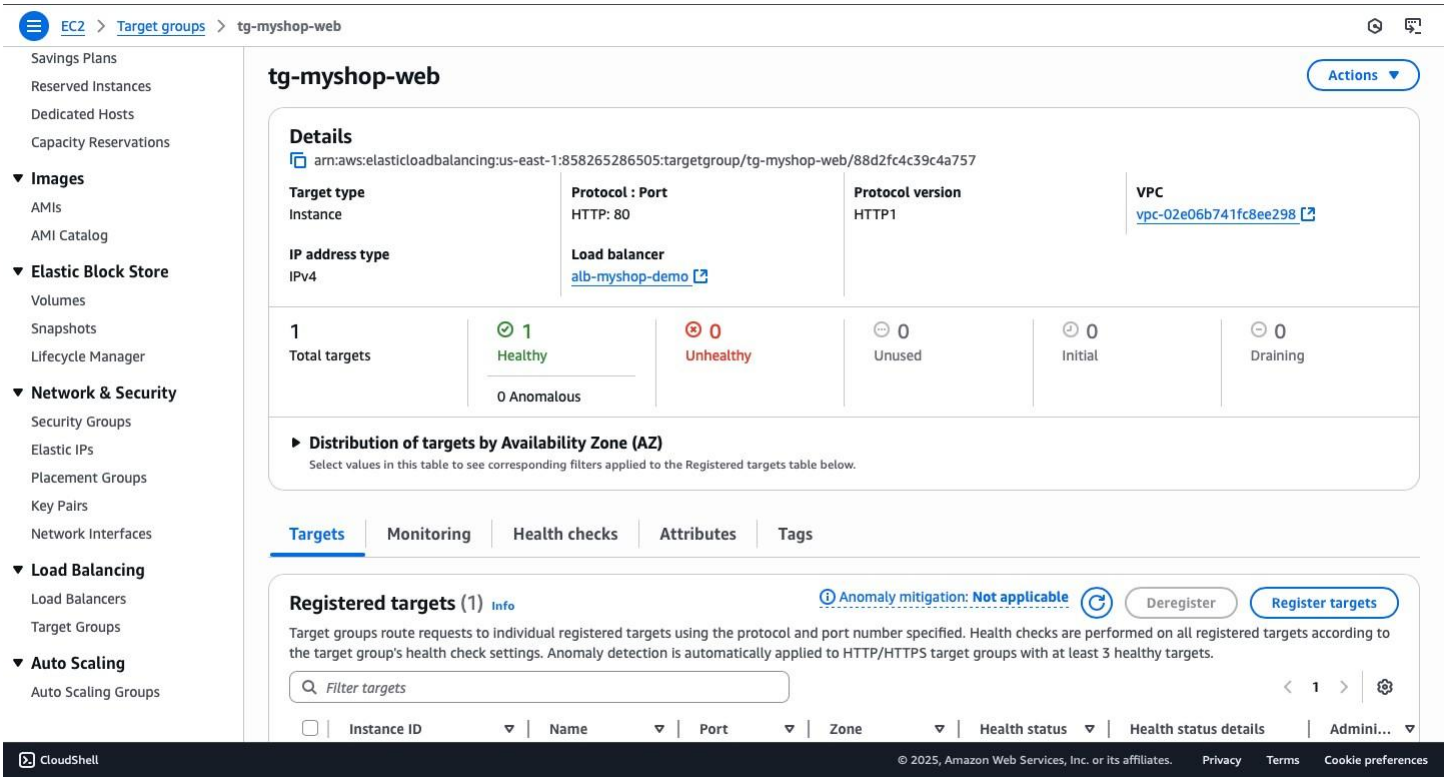


Outcome: Access http://44.211.219.210 → should display *Hello from EC2 backend*

Step 2: Create a Target Group

1. Go to **EC2 → Target Groups → Create Target Group**.
2. **Name:** tg-myshop-web
3. **Target type:** Instances
4. **Protocol / Port:** HTTP : 80
5. **VPC:** Select your VPC
6. **Health check path:** /
7. Register your EC2 instance on port 80.

Outcome: EC2 instance shows as **Healthy**.

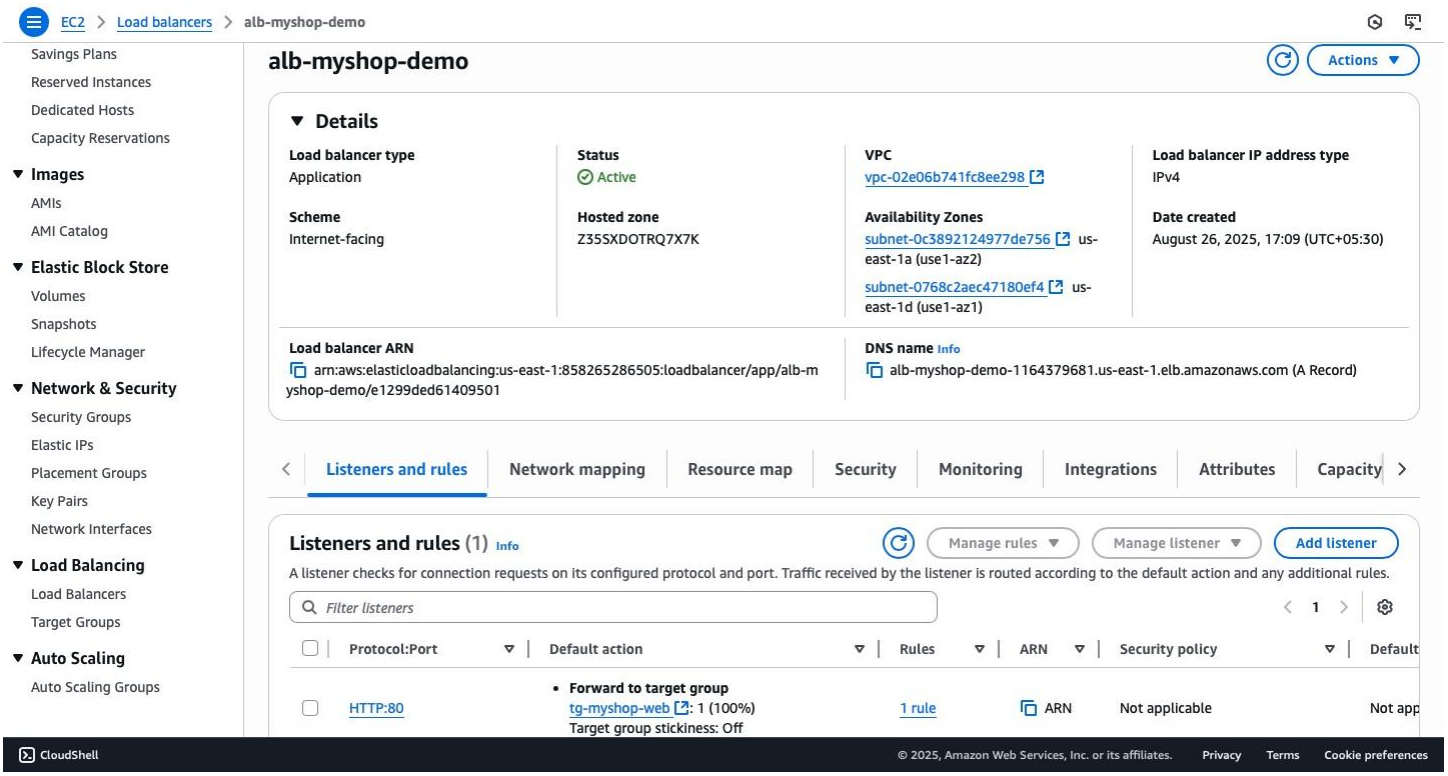


Step 3: Create an Application Load Balancer (ALB)

1. Go to **EC2 → Load Balancers → Create Load Balancer → Application Load Balancer**.
2. **Name:** alb-myshop-demo
3. **Scheme:** Internet-facing
4. **IP address type:** IPv4
5. **Network mapping:** Select two public subnets in different AZs.
6. **Security group:** Create sg-alb-public with inbound rule: HTTP (80) from 0.0.0.0/0
7. **Listeners:**
   - Add HTTP : 80 → forward to tg-myshop-web
8. Click **Create load balancer**.

Outcome: ALB DNS (example):

http://alb-myshop-demo-1164379681.us-east-1.elb.amazonaws.com



Visit http://<ALB-DNS> → loads the EC2 page.

Step 4: Create a CloudFront Distribution (HTTPS for Demo)

1. Go to **CloudFront → Create Distribution**.
2. **Origin domain:** Enter your ALB DNS name.
3. **Origin name:** origin-myshop-alb
4. **Origin protocol policy:** HTTP only (for demo).
5. **Default cache behavior:**
   - Viewer protocol policy: Redirect HTTP to HTTPS
   - Allowed methods: GET, HEAD (or ALL if needed) Cache
   - policy: CachingDisabled (for dynamic apps)

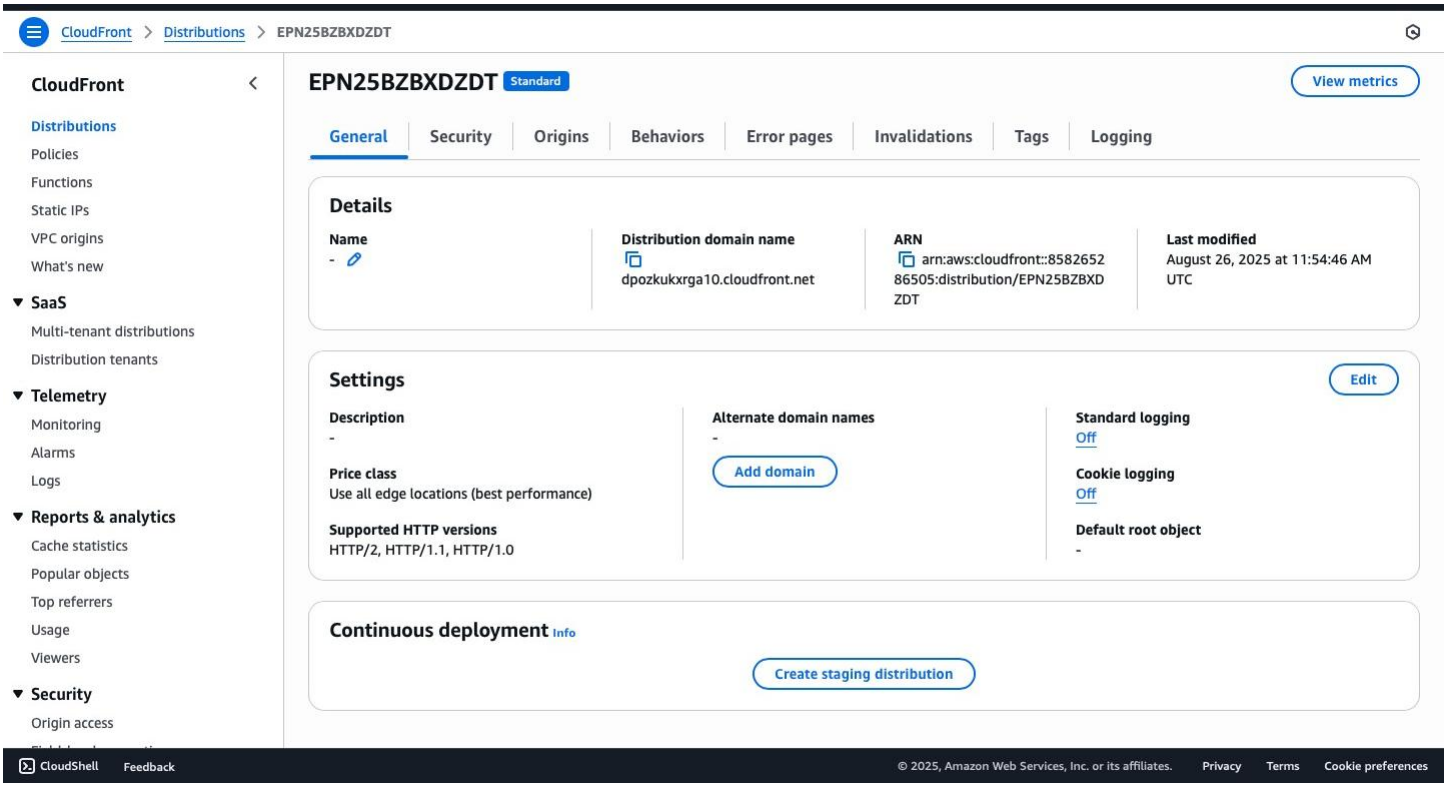
   6. **Settings:**
   - Alternate domain names: *(leave blank for demo)*
   - SSL certificate: Use default CloudFront certificate (*.cloudfront.net)

Click **Create distribution**.

Outcome: CloudFront domain (example):

dpozkukxrga10.cloudfront.net

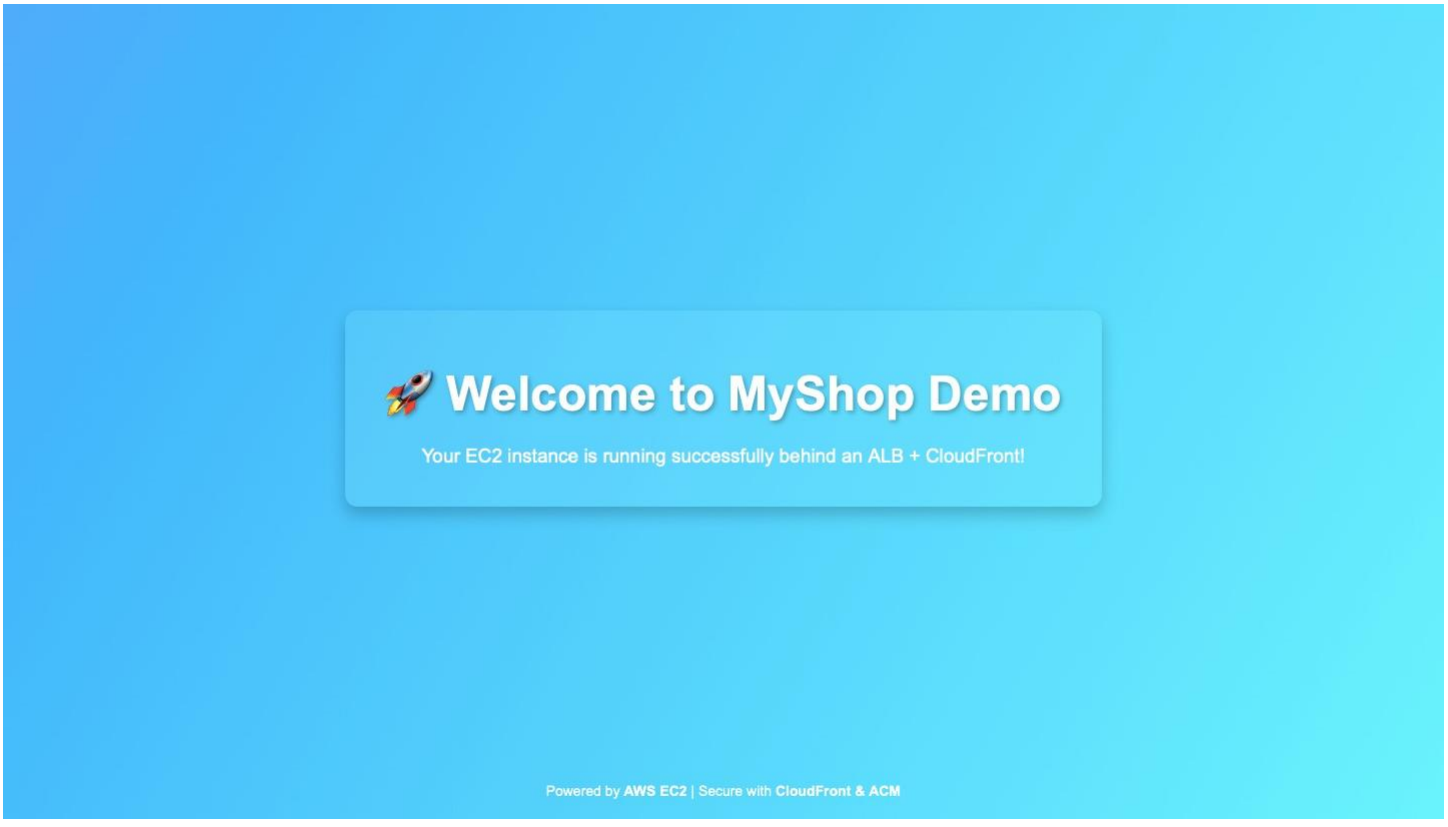
Visit https://dpozkukxrga10.cloudfront.net



Step 5: Verify End-to-End Flow

- Test EC2 directly → http://44.211.219.210
- Test ALB →http://alb-myshop-demo-1164379681.us-east-1.elb.amazonaws.com
- Test CloudFront → https://dpozkukxrga10.cloudfront.net
- Test HTTP→HTTPS redirect on CloudFront → automatically redirects.

Outcome: Application is secure and globally available.



Step 6: Production Setup with Route 53 + Custom Domain + ACM

If you purchase a domain (e.g., myshop.com):

6.1 Route 53 Setup

1. Create a **Route 53 Hosted Zone** for myshop.com.
2. Update your registrar's nameservers to Route 53.

6.2 Request ACM Certificates

1. Go to **ACM (us-east-1 for CloudFront)** → Request a public certificate for www.myshop.com.
2. Validate via DNS (CNAME record in Route 53).
3. Go to **ACM (same region as ALB, e.g., ap-south-1)** → Request a public certificate for app.myshop.com.
4. Validate via DNS.

6.3 Attach Certificate to ALB

1. In **EC2 → Load Balancers → Listeners**, edit or add **HTTPS (443)** listener.
2. Choose the ACM certificate for app.myshop.com (regional cert).
3. Default action: Forward to tg-myshop-web.

Outcome: The ALB can now terminate HTTPS traffic with your ACM certificate.

6.4 Update CloudFront

1. Add www.myshop.com as an **Alternate Domain Name (CNAME)**.
2. Attach the ACM certificate (us-east-1) for www.myshop.com.
3. Origin protocol policy → HTTPS only (CloudFront → ALB).

6.5 Route 53 DNS Records

1. Add **A/AAAA Alias Record**: www.myshop.com → CloudFront distribution.
2. (Optional) Add apex myshop.com → CloudFront distribution.

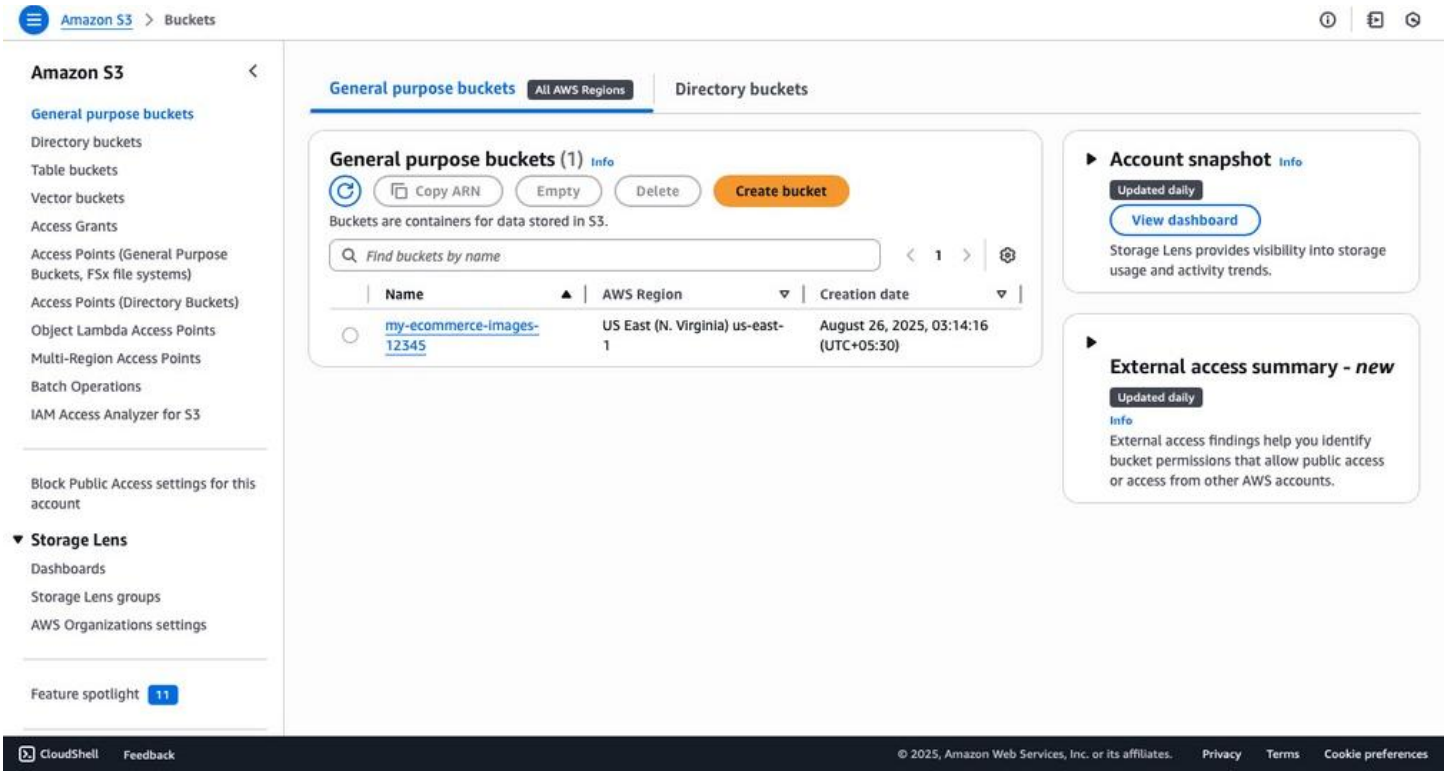
Outcome: Users access your app via https://www.myshop.com (CloudFront) → https://app.myshop.com (ALB) → EC2 backend.

6.Integrate CloudFront with S3 for Faster Content Delivery Configure a CloudFront distribution to serve static assets from S3. Restrict direct S3 access and enforce traffic via CloudFront. Test performance improvements by comparing direct S3 vs. CloudFront load times.

This guide explains how to set up an Amazon S3 bucket for product images, configure a CloudFront distribution to deliver them globally, and enforce secure access through bucket policies.

Step 1: Create an S3 Bucket

1. **Log in to the AWS Management Console.**
2. **Go to S3 → Create Bucket.**
3. **Configure the bucket:**
   - **Bucket Name: my-product-images-12345 (must be globally unique).**
   - **Region: Closest to your location (e.g., *US East – N. Virginia*).**
   - **Bucket Type: General Purpose.**
   - **Object Ownership: Bucket owner enforced.**
   - **Block Public Access: Keep all ON (CloudFront will handle access).**
   - **Default Encryption: SSE-S3 (Amazon-managed).**
4. **Click Create Bucket.**

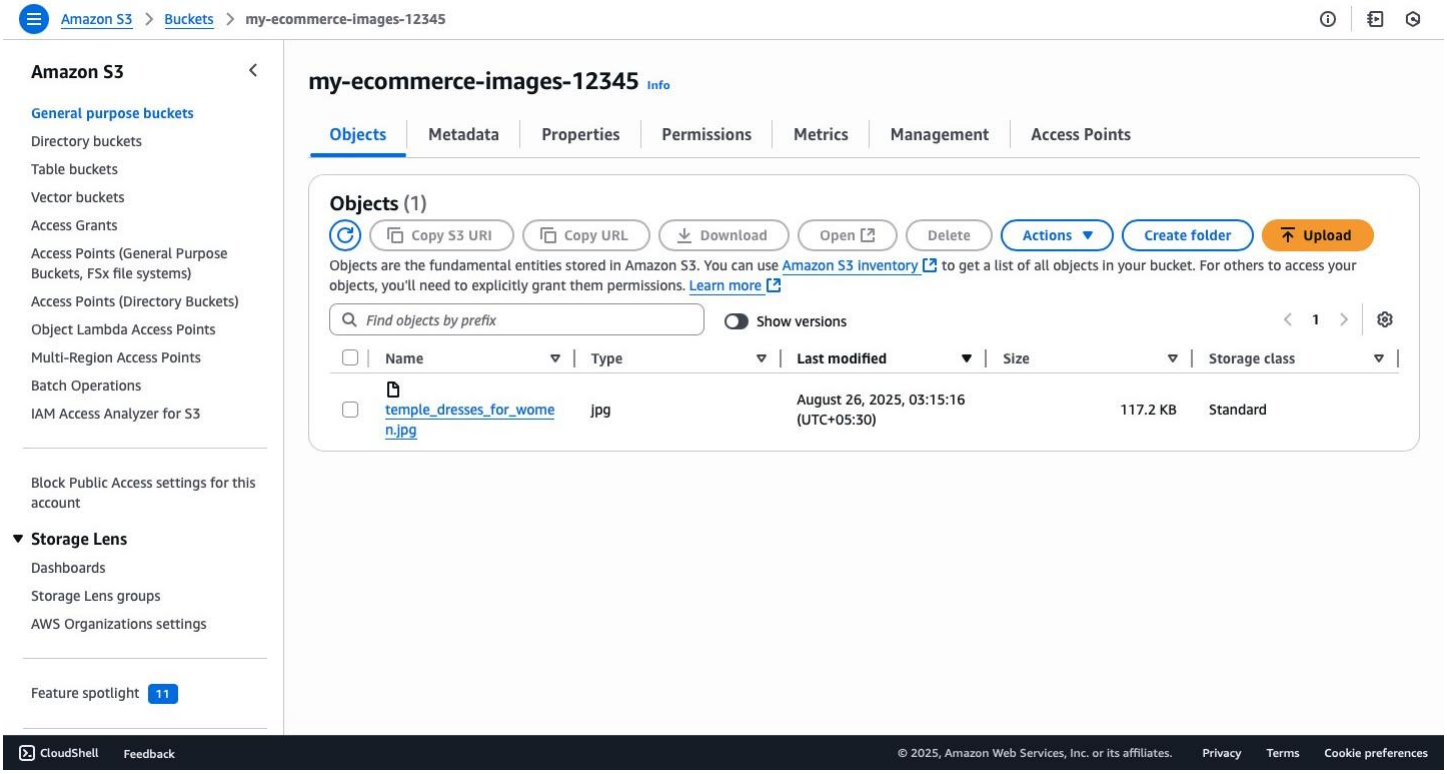


Step 2: Upload Product Images to S3

1. **Open your bucket (my-product-images-12345).**
2. **Click Upload → Add files.**
3. **Select product images (e.g., temple_dresses_for_women.jpg, shirt.jpg).**
4. **Click Upload.**

Note: Images remain private; CloudFront will handle secure delivery.

Outcome: Product images are now stored in S3.

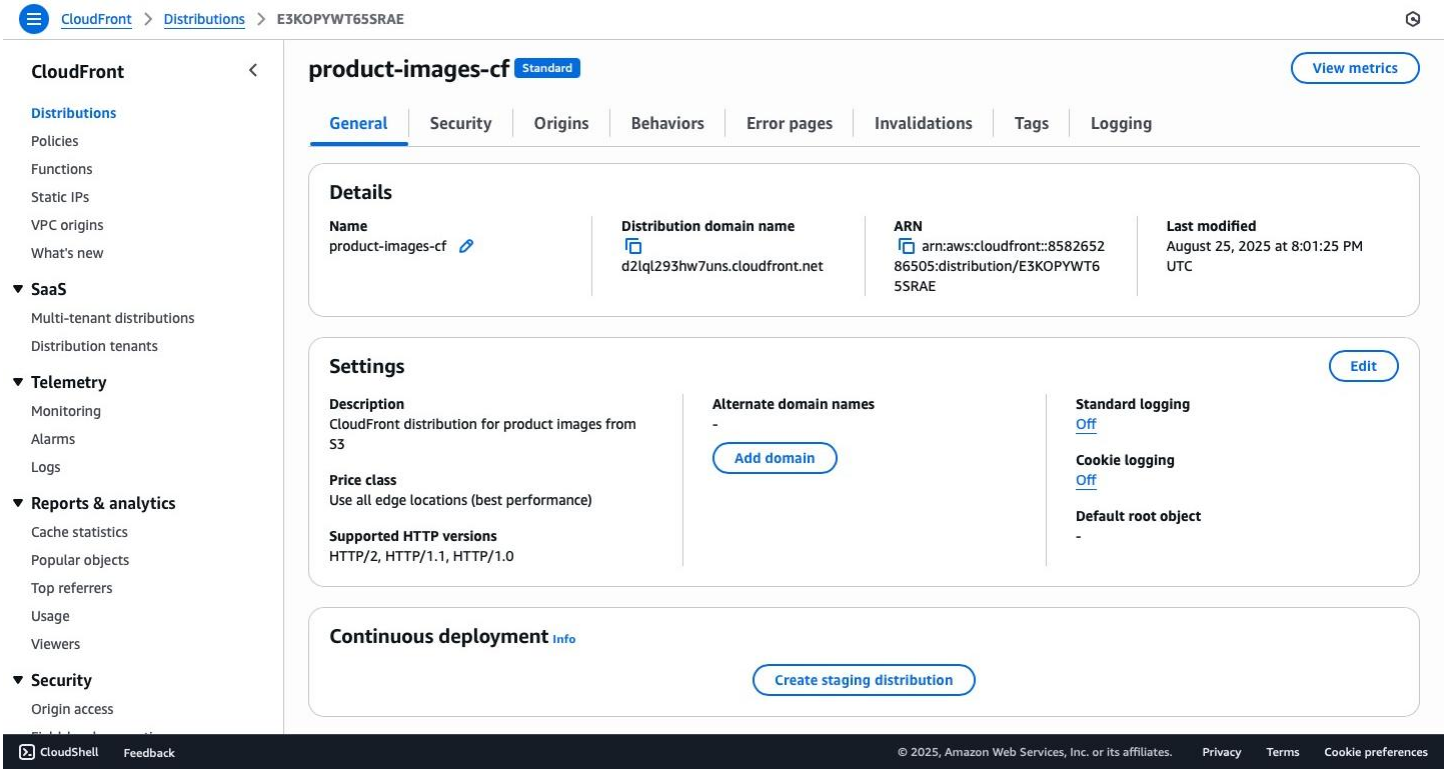


Step 3: Create a CloudFront Distribution

1. **Go to CloudFront → Create Distribution.**
2. **Configure the distribution:**
   - **Distribution Name: product-images-cf.**
   - **Origin Type: Amazon S3 → select your bucket my-product-images-123.**
   - **Grant CloudFront access to origin: Yes (creates an Origin Access Control, OAC).**
   - **Viewer Protocol Policy: Redirect HTTP → HTTPS (recommended).**
   - **Cache Settings: Use recommended defaults.**
3. **Click Create Distribution.**
4. **Wait 10–15 minutes for deployment.**
5. **Copy the CloudFront domain name (example):**

   https://d2lql293hw7uns.cloudfront.net

Outcome: CloudFront is ready to deliver your images globally, faster than direct S3 access.



Step 4: Test Images via CloudFront

1. Open a browser.
2. Enter a CloudFront image URL
3. The image should load successfully.

Outcome: CloudFront fetches and serves images securely from S3.

Step 5: Use CloudFront Images in Your App

Replace local image paths with CloudFront URLs.

Example:

<img src="https://d2lql293hw7uns.cloudfront.net/temple_dresses_for_women.jpg" alt="Temple Dress">

Benefits:

- Images load faster worldwide.
- Reduces storage/load on your app server.
- Images are served securely with CloudFront access policies.

Step 6: Apply an S3 Bucket Policy

To ensure only CloudFront can access your bucket, attach the following **bucket policy** (OAC-based example):

{

"Version": "2012-10-17",

```
  "Id": "PolicyForCloudFrontPrivateContent",
  "Statement": [
    {
      "Sid": "AllowCloudFrontServicePrincipal",
      "Effect": "Allow",
      "Principal": {
        "Service": "cloudfront.amazonaws.com"
      },
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::my-ecommerce-images-12345/*",
      "Condition": {
        "StringEquals": {
          "AWS:SourceArn": "arn:aws:cloudfront::858265286505:distribution/E3KOPYWT65SRAE"
        }
      }
    }
  ]
}
```

Amazon S3

General purpose buckets
Directory buckets
Table buckets
Vector buckets
Access Grants
Access Points (General Purpose Buckets, FSx file systems)
Access Points (Directory Buckets)
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
IAM Access Analyzer for S3

Block Public Access settings for this account

Storage Lens
Dashboards
Storage Lens groups
AWS Organizations settings

Feature spotlight 11

Bucket policy

Edit Delete

The bucket policy, written in JSON, provides access to the objects stored in the bucket. Bucket policies don't apply to objects owned by other accounts. Learn more

Copy


```
{
  "Version": "2012-10-17",
  "Id": "PolicyForCloudFrontPrivateContent",
  "Statement": [
    {
      "Sid": "AllowCloudFrontServicePrincipal",
      "Effect": "Allow",
      "Principal": {
        "Service": "cloudfront.amazonaws.com"
      },
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::my-ecommerce-images-12345/*",
      "Condition": {
        "StringEquals": {
          "AWS:SourceArn": "arn:aws:cloudfront::858265286505:distribution/E3KOPYWT65SRAE"
        }
      }
    }
  ]
}
```


Object Ownership Info

Edit