REST API DEPLOYMENT

Deployed a containerized Node.js REST API on AWS using a scalable, production-ready architecture. The API is hosted on ECS Fargate, exposed securely through an Application Load Balancer, and managed via Amazon ECR.

Tech Stacks Used

- **Node.js (Express.js)** – REST API application
- **Docker** – Containerization
- **Amazon EC2** – Virtual server for Docker image build
- **Amazon ECR** – Container image repository
- **Amazon ECS (Fargate)** – Serverless container orchestration
- **Application Load Balancer (ALB)** – Traffic routing and health checks
- **AWS VPC & Security Groups** – Networking and access control
- **AWS CLI** – Image push and infrastructure interaction

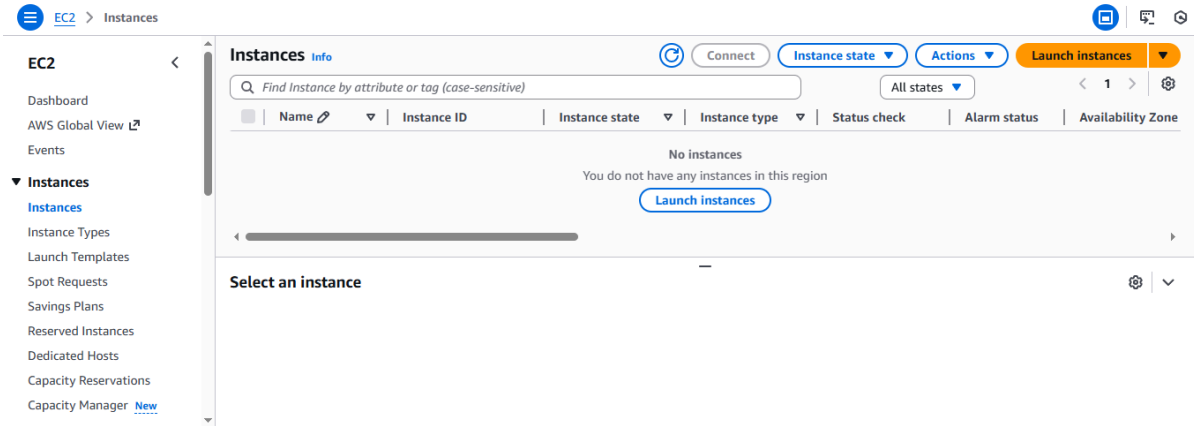
STEP 1: Launch an EC2 Virtual Server (Docker Build Server)

This EC2 instance will:

- Run Docker
- Build your website image
- Push image to ECR
  (No Docker Desktop needed)

1.1 Go to EC2 Console

AWS Console → **EC2** → **Instances** → **Launch instance**



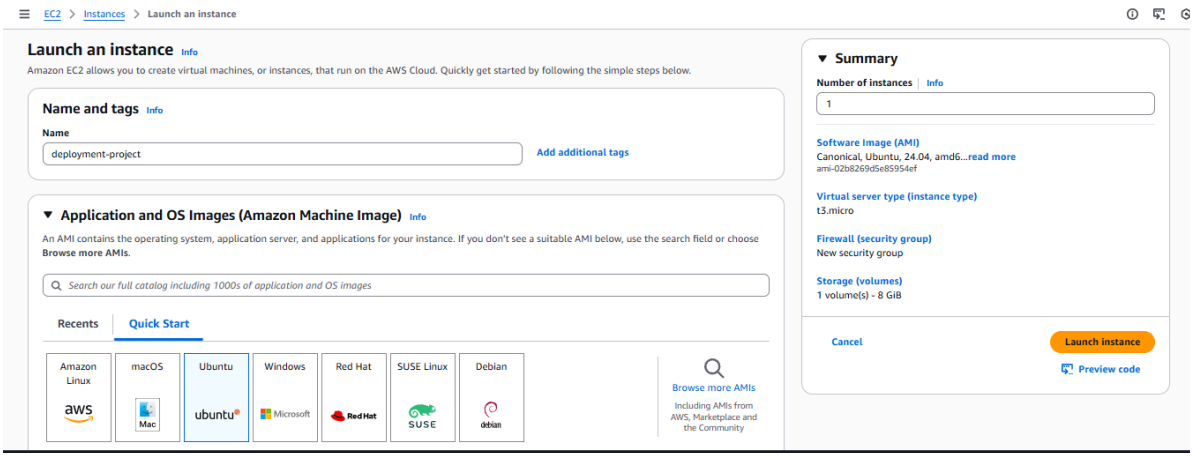
1.2 Choose AMI (Operating System)

Select:

Ubuntu

Why:

- AWS optimized
- Stable
- Docker works perfectly

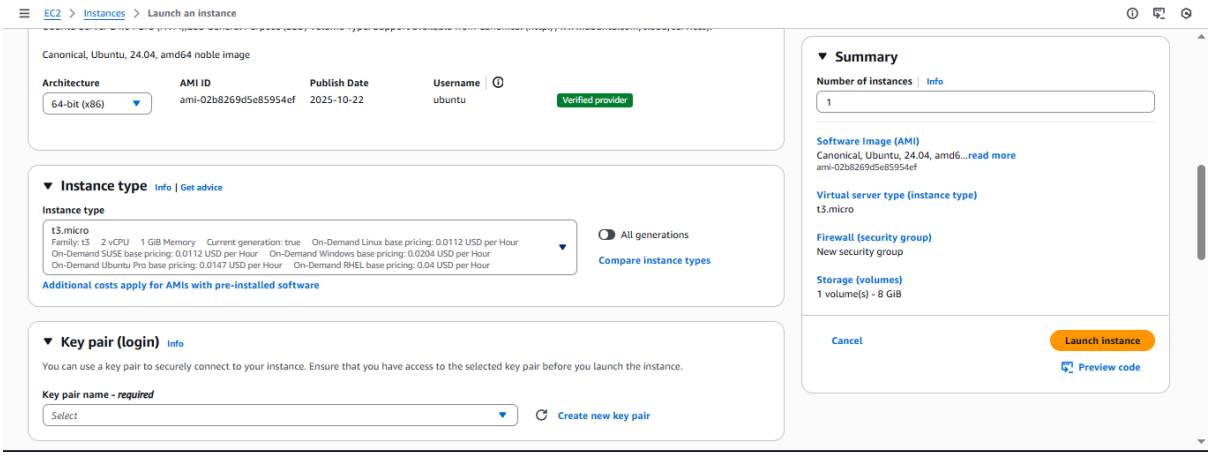


1.3 Choose Instance Type

Select:

t2.micro or **t3.micro**
(Free-tier eligible)

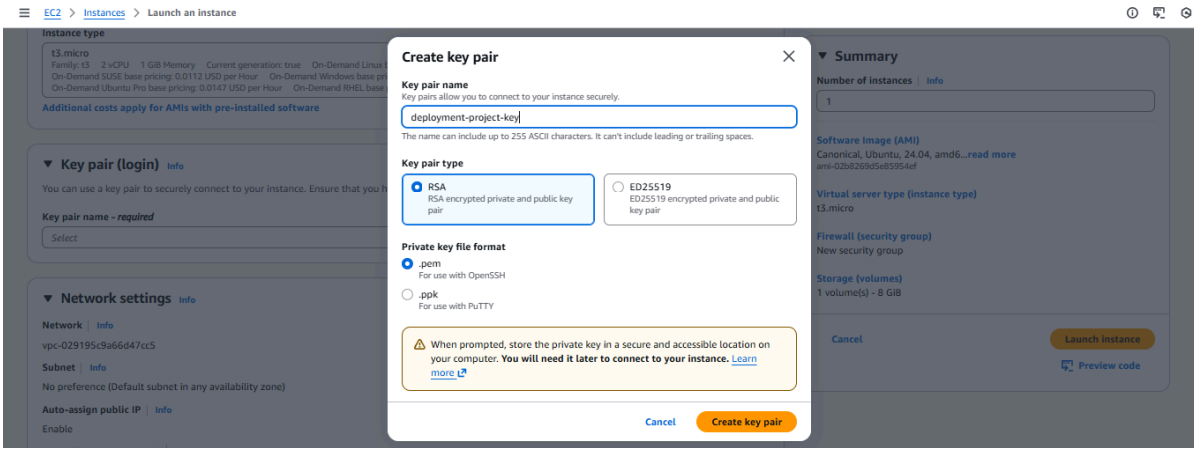
Click **Next / Continue**

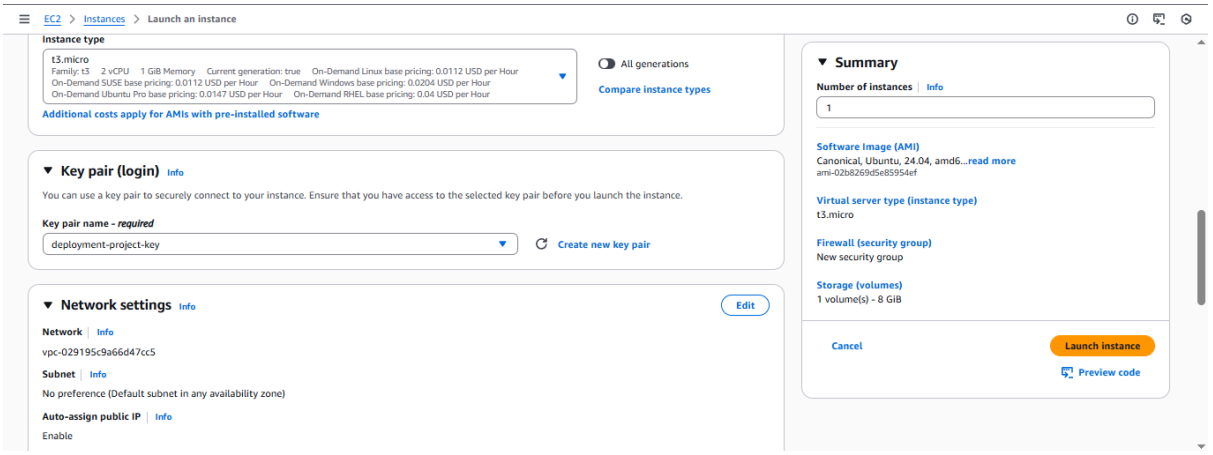


1.4 Key Pair (VERY IMPORTANT)

- Select **Create new key pair**
- Key pair type: RSA
- File format: `.pem`
- Name: `deployment-project-key`

Download and save this file safely
(You cannot download it again)

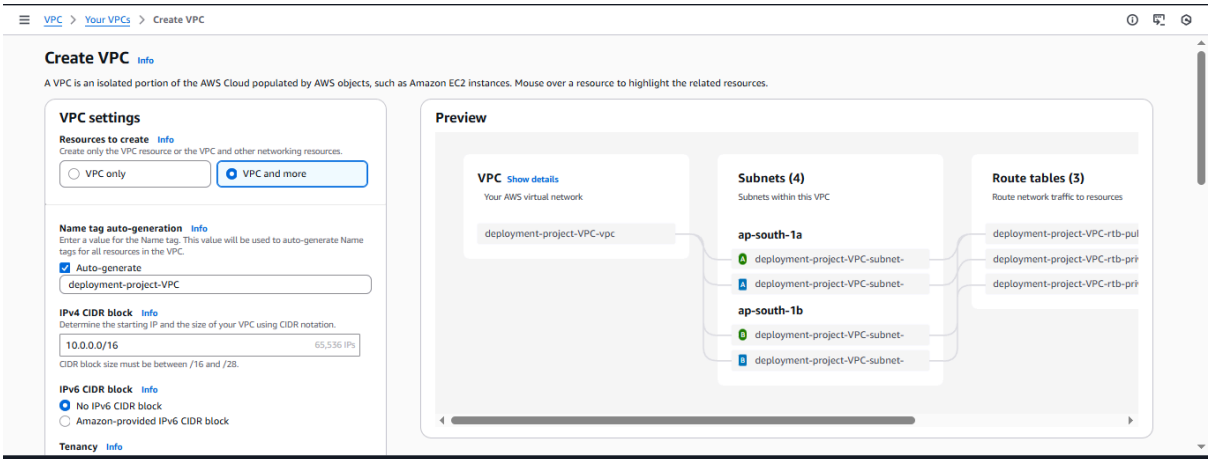


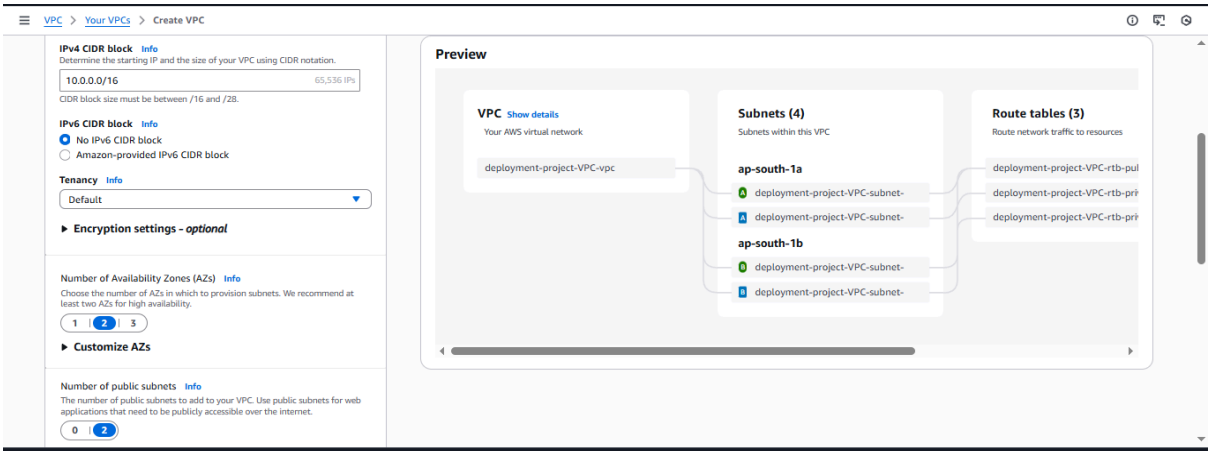


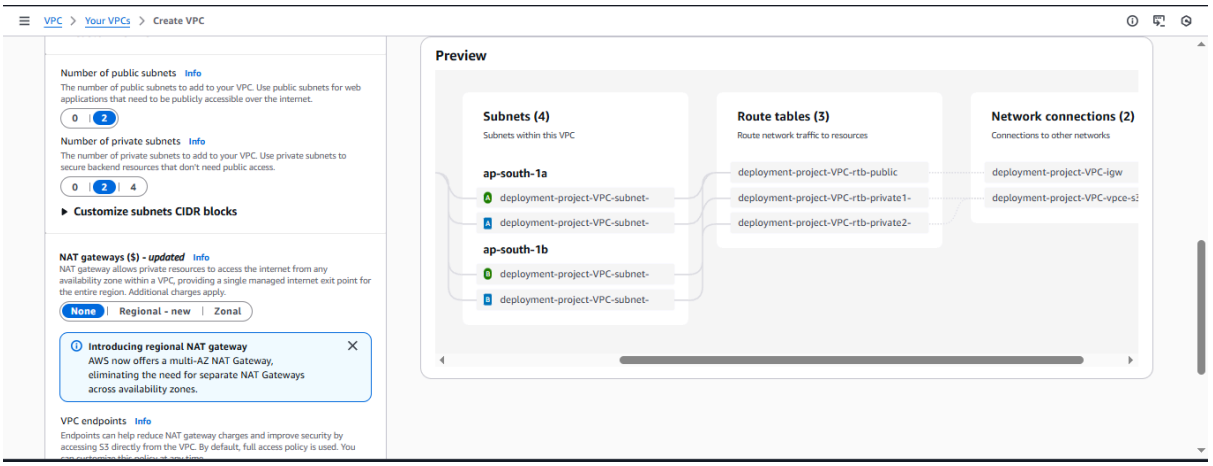
1.5 Network Settings

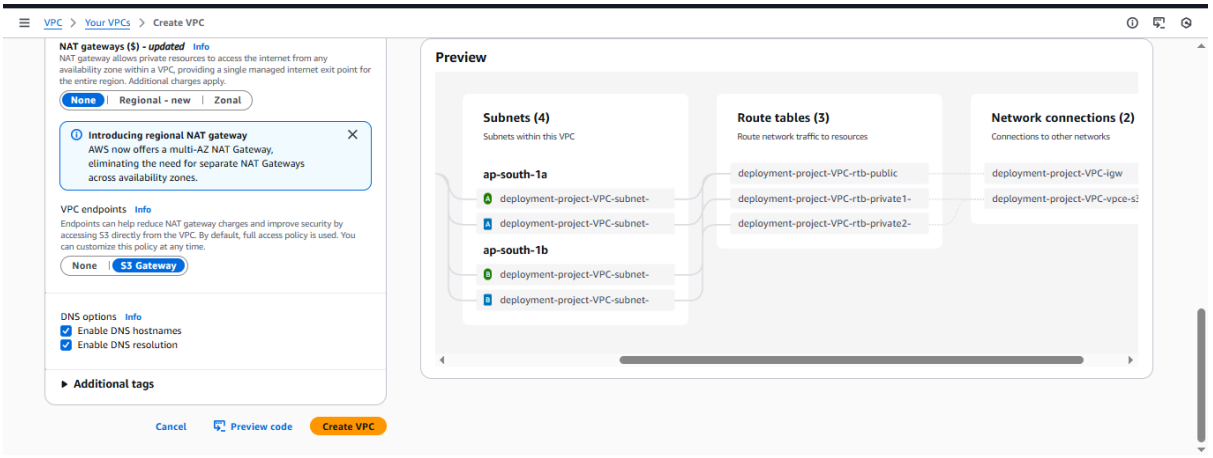
Configure:

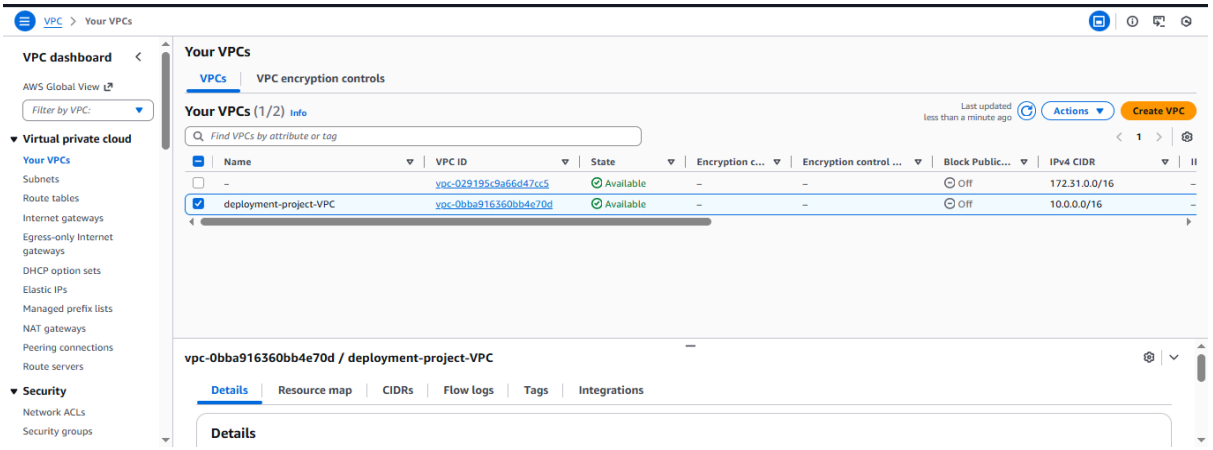
- VPC: Use your created VPC - `deployment-project-VPC`
- Subnet: Public subnet
- Auto-assign public IP: **Enabled**

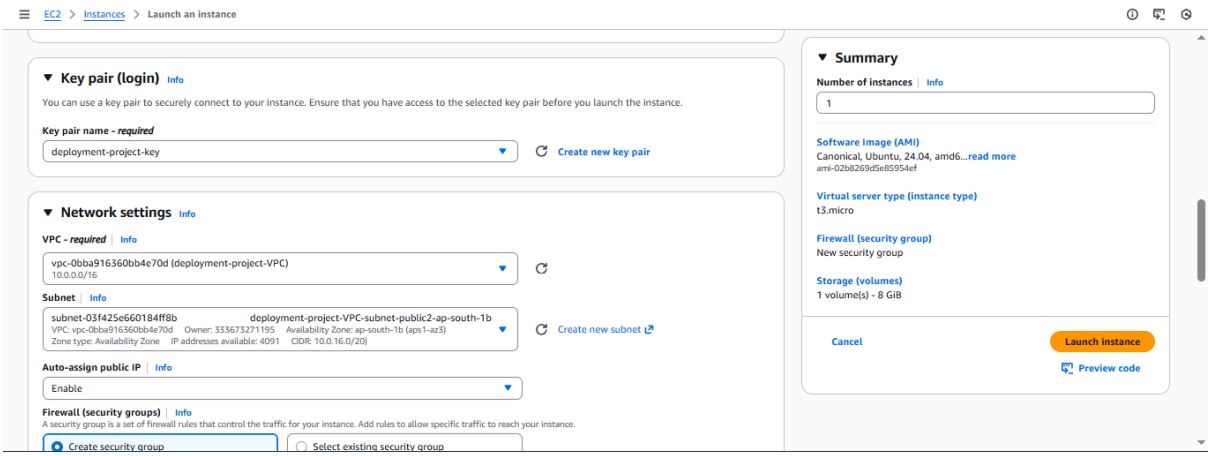



VPC > Your VPCs > Create VPC
IPv4 CIDR block Info
Determine the starting IP and the size of your VPC using CIDR notation.
10.0.0.0/16
65,536 IPs
CIDR block size must be between /16 and /28.
IPv6 CIDR block Info
No IPv6 CIDR block
Amazon-provided IPv6 CIDR block
Tenancy Info
Default
Encryption settings - optional
Number of Availability Zones (AZs) Info
Choose the number of AZs in which to provision subnets. We recommend at least two AZs for high availability.
1 2 3
Customize AZs
Number of public subnets Info
The number of public subnets to add to your VPC. Use public subnets for web applications that need to be publicly accessible over the internet.
0 2
Preview
VPC Show details
Your AWS virtual network
deployment-project-VPC-vpc
Subnets (4)
Subnets within this VPC
ap-south-1a
deployment-project-VPC-subnet-
deployment-project-VPC-subnet-
ap-south-1b
deployment-project-VPC-subnet-
deployment-project-VPC-subnet-
Route tables (3)
Route network traffic to resources
deployment-project-VPC-rtb-pul
deployment-project-VPC-rtb-pri
deployment-project-VPC-rtb-pri


VPC > Your VPCs > Create VPC
Number of public subnets Info
The number of public subnets to add to your VPC. Use public subnets for web applications that need to be publicly accessible over the internet.
0 2
Number of private subnets Info
The number of private subnets to add to your VPC. Use private subnets to secure backend resources that don't need public access.
0 2 4
Customize subnets CIDR blocks
NAT gateways ($) - updated Info
NAT gateway allows private resources to access the internet from any availability zone within a VPC, providing a single managed internet exit point for the entire region. Additional charges apply.
None Regional - new Zonal
Introducing regional NAT gateway
AWS now offers a multi-AZ NAT Gateway, eliminating the need for separate NAT Gateways across availability zones.
VPC endpoints Info
Endpoints can help reduce NAT gateway charges and improve security by accessing S3 directly from the VPC. By default, full access policy is used. You
Preview
Subnets (4)
Subnets within this VPC
ap-south-1a
deployment-project-VPC-subnet-
deployment-project-VPC-subnet-
ap-south-1b
deployment-project-VPC-subnet-
deployment-project-VPC-subnet-
Route tables (3)
Route network traffic to resources
deployment-project-VPC-rtb-public
deployment-project-VPC-rtb-private1-
deployment-project-VPC-rtb-private2-
Network connections (2)
Connections to other networks
deployment-project-VPC-igw
deployment-project-VPC-vpce-s3


VPC > Your VPCs > Create VPC
NAT gateways ($) - updated Info
NAT gateway allows private resources to access the internet from any availability zone within a VPC, providing a single managed internet exit point for the entire region. Additional charges apply.
None Regional - new Zonal
Introducing regional NAT gateway
AWS now offers a multi-AZ NAT Gateway, eliminating the need for separate NAT Gateways across availability zones.
VPC endpoints Info
Endpoints can help reduce NAT gateway charges and improve security by accessing S3 directly from the VPC. By default, full access policy is used. You can customize this policy at any time.
None S3 Gateway
DNS options Info
Enable DNS hostnames
Enable DNS resolution
Additional tags
Cancel
Preview code
Create VPC
Preview
Subnets (4)
Subnets within this VPC
ap-south-1a
deployment-project-VPC-subnet-
deployment-project-VPC-subnet-
ap-south-1b
deployment-project-VPC-subnet-
deployment-project-VPC-subnet-
Route tables (3)
Route network traffic to resources
deployment-project-VPC-rtb-public
deployment-project-VPC-rtb-private1-
deployment-project-VPC-rtb-private2-
Network connections (2)
Connections to other networks
deployment-project-VPC-igw
deployment-project-VPC-vpce-s3





1.6 Security Group Configuration

Create a new security group:

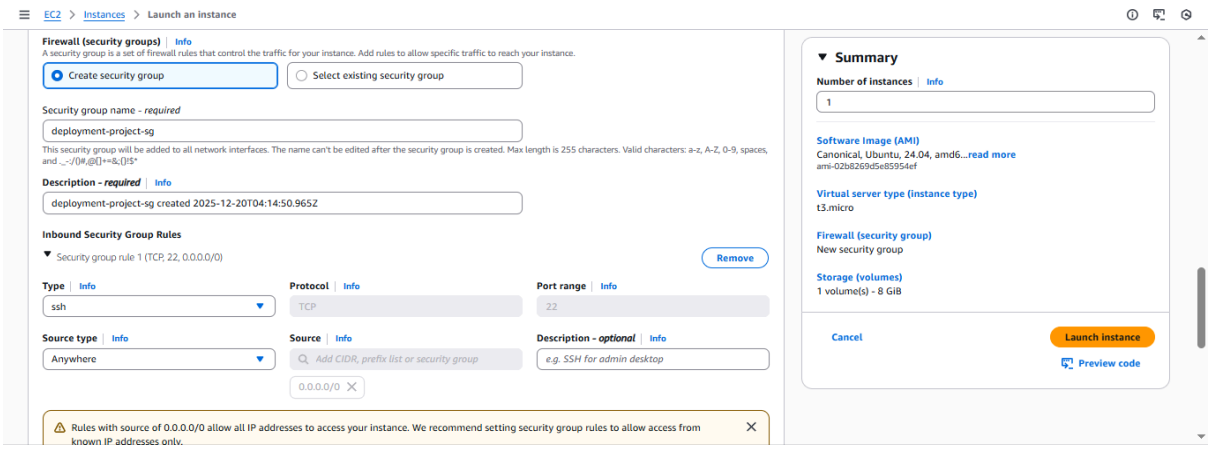
Inbound Rules

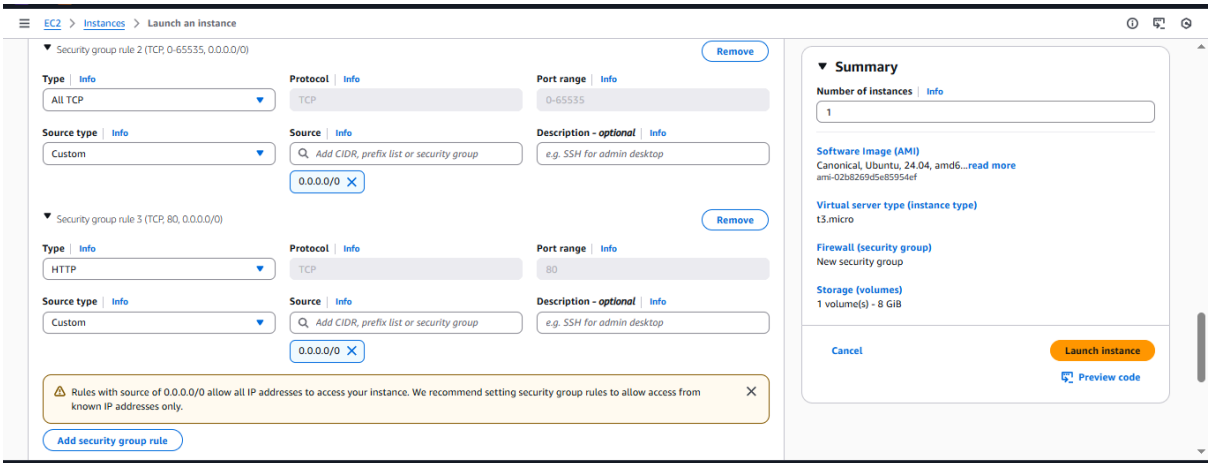
Add these rules:

```
SSH    | TCP | 22 | 0.0.0.0/0
HTTP   | TCP | 80 | 0.0.0.0/0
All TCP  | TCP | 0.0.0.0/0
```

Why:

- SSH → connect to EC2
- HTTP → test website temporarily

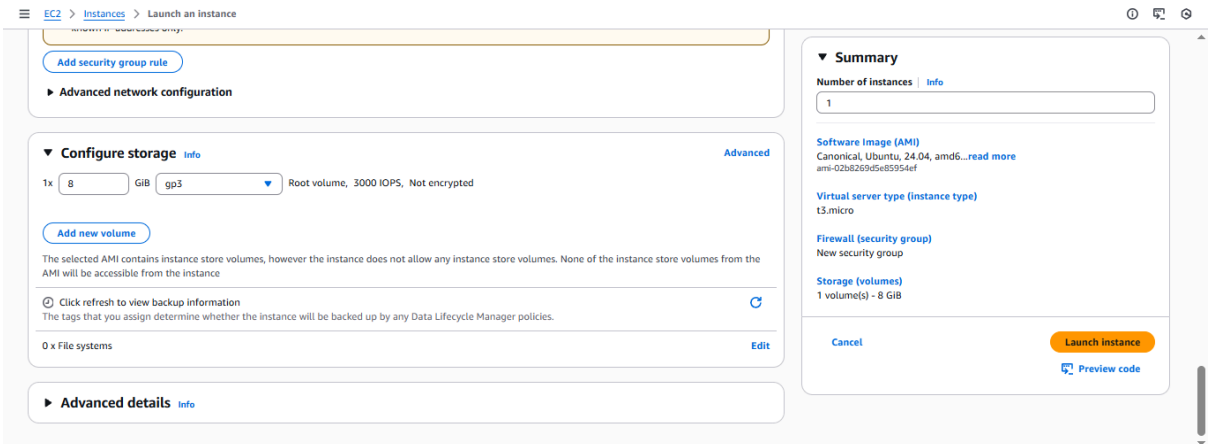




1.7 Storage

Leave default:

```
8 GB gp3
```

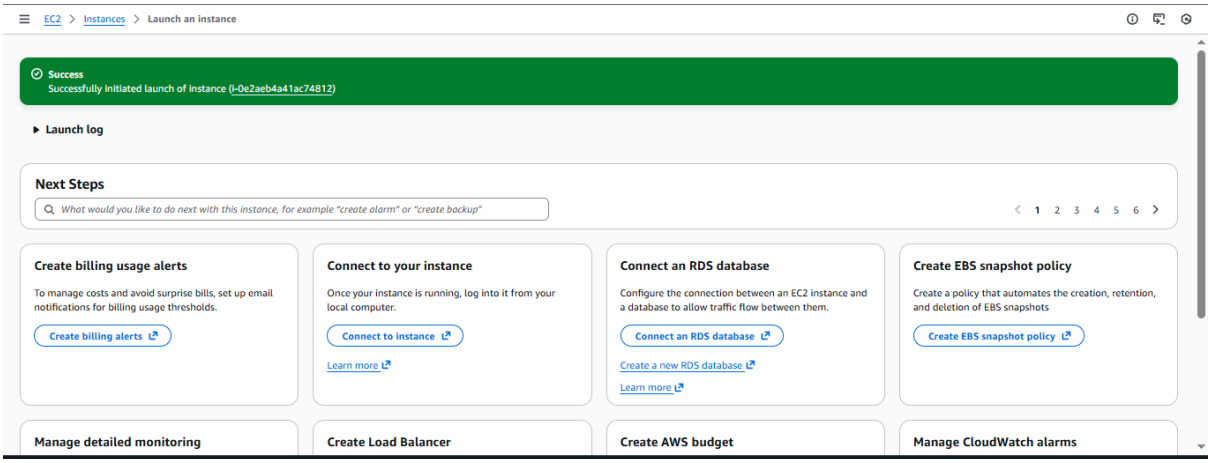


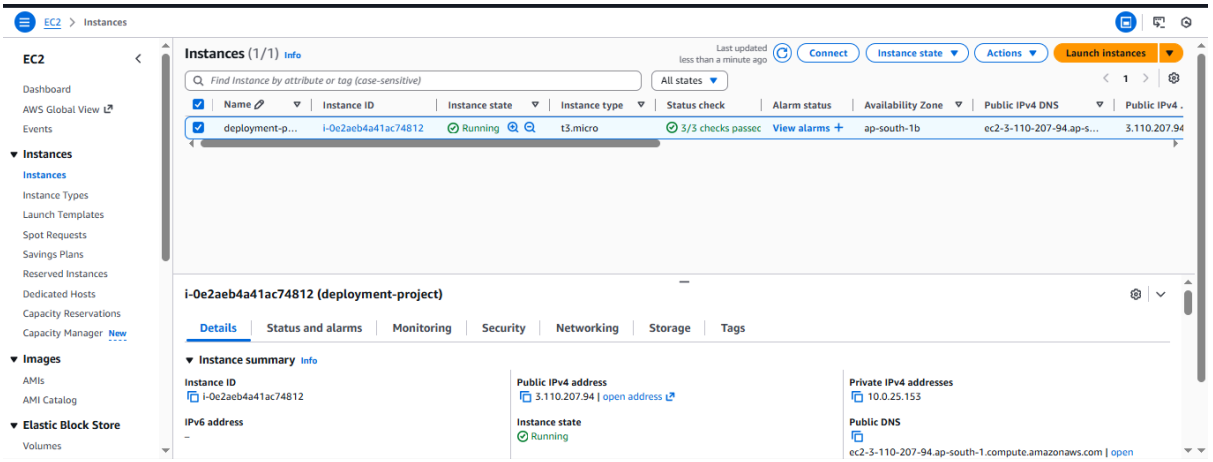
1.8 Launch Instance

Click **Launch Instance**

Wait until:

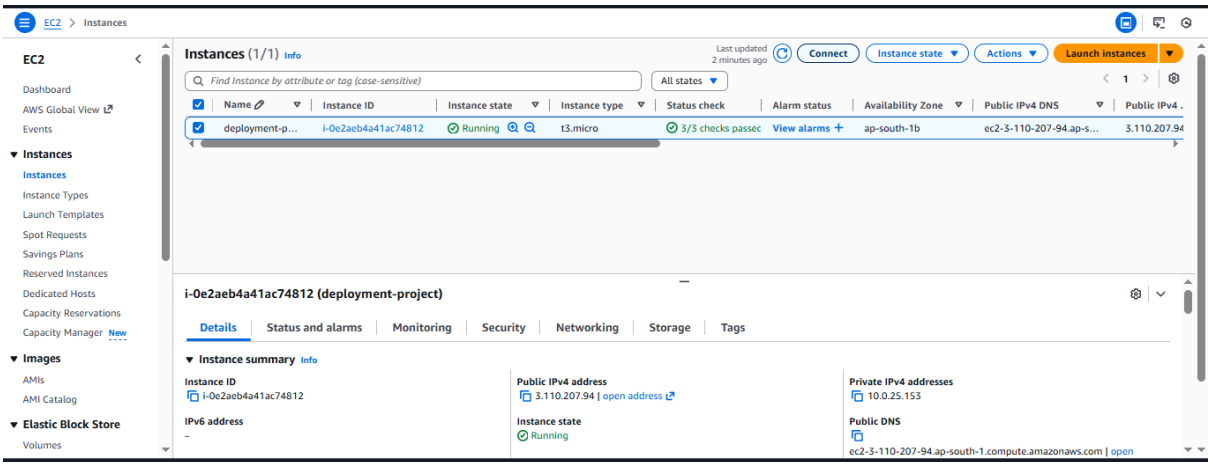
- Instance state = **Running**
- Status checks = **3/3 passed**

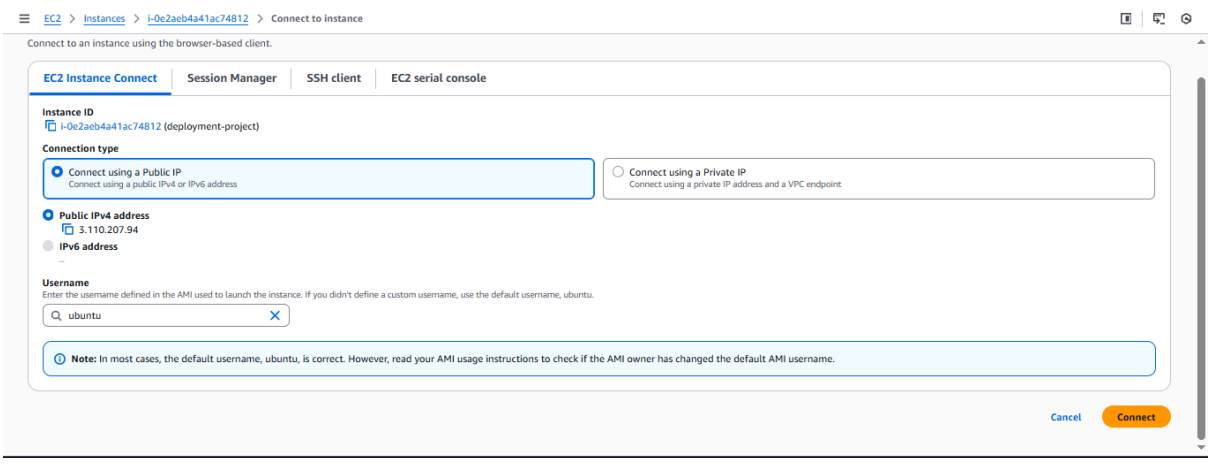


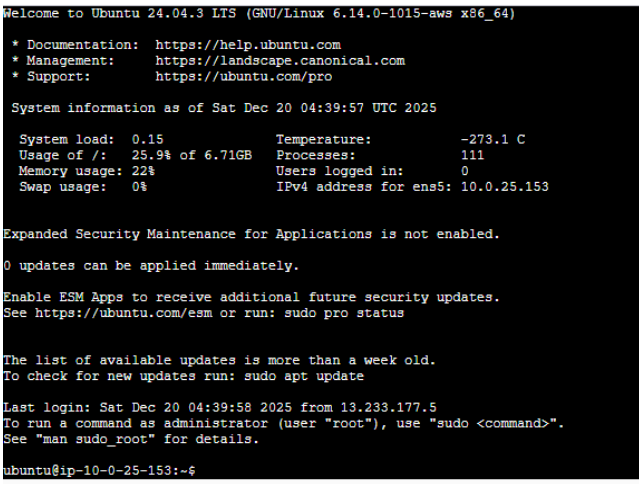


STEP 2: Connect to EC2 (Using AWS Console)

You are already inside the EC2 terminal.
Now we prepare this virtual server to act as a **Docker machine**.







STEP 3: Install Docker on EC2 (Virtual Docker)

3.1 Update the EC2 System

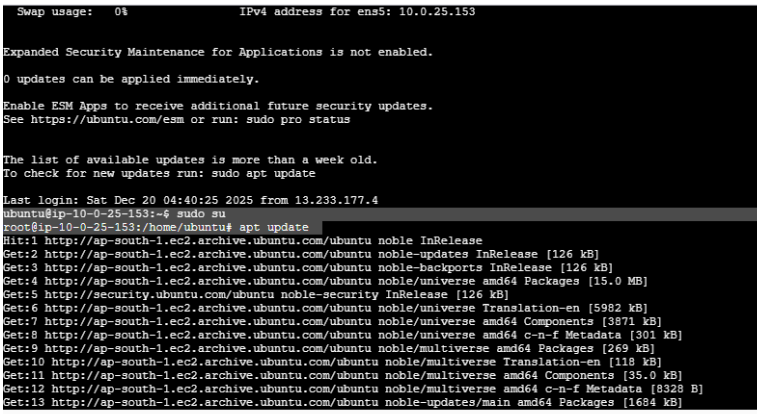
Run this **exact command** in the EC2 terminal:

```
sudo su
apt update
```

What this does

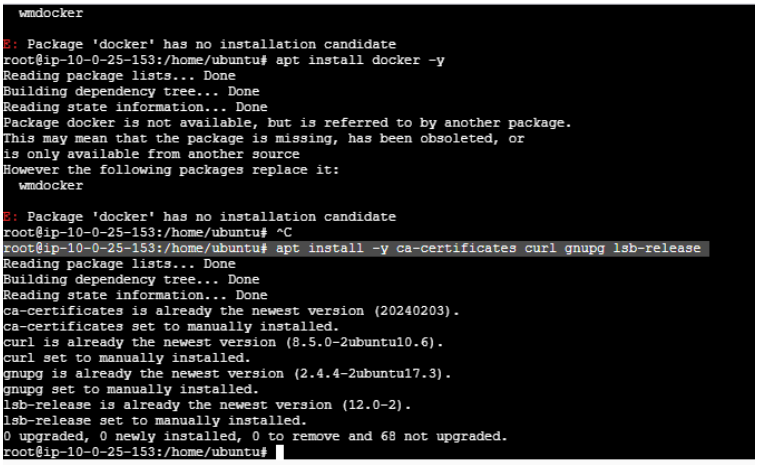
- Updates system packages
- Prevents Docker installation issues

Wait until it completes.



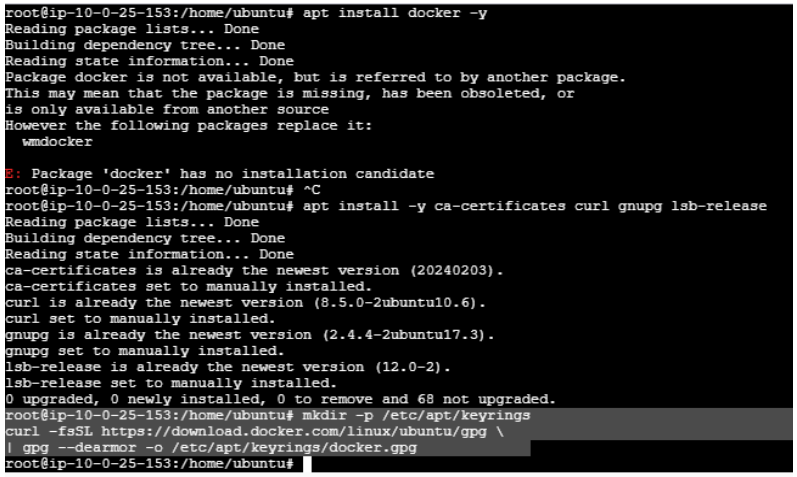
3.2 Install Required Packages

```
sudo apt install -y ca-certificates curl gnupg lsb-release
```



3.3 Add Docker's Official GPG Key

```
mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg \
| gpg --dearmor -o /etc/apt/keyrings/docker.gpg
```



3.4 Add Docker Repository

```
echo \
"deb [arch=$(dpkg --print-architecture)
signed-by=/etc/apt/keyrings/docker.gpg] \
https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) stable" \
| tee /etc/apt/sources.list.d/docker.list > /dev/null
```

```
is only available from another source
However the following packages replace it:
  wmdocker

E: Package 'docker' has no installation candidate
root@ip-10-0-25-153:/home/ubuntu# ^C
root@ip-10-0-25-153:/home/ubuntu# apt install -y ca-certificates curl gnupg lsb-release
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
ca-certificates is already the newest version (20240203).
ca-certificates set to manually installed.
curl is already the newest version (8.5.0-2ubuntu10.6).
curl set to manually installed.
gnupg is already the newest version (2.4.4-2ubuntu17.3).
gnupg set to manually installed.
lsb-release is already the newest version (12.0-2).
lsb-release set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 68 not upgraded.
root@ip-10-0-25-153:/home/ubuntu# mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg \
| gpg --dearmor -o /etc/apt/keyrings/docker.gpg
root@ip-10-0-25-153:/home/ubuntu#
echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] \
https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) stable" \
| tee /etc/apt/sources.list.d/docker.list > /dev/null
root@ip-10-0-25-153:/home/ubuntu#
```

3.5 Update Package List Again

`apt update`

```
curl is already the newest version (8.5.0-2ubuntu10.6).
curl set to manually installed.
gnupg is already the newest version (2.4.4-2ubuntu17.3).
gnupg set to manually installed.
lsb-release is already the newest version (12.0-2).
lsb-release set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 68 not upgraded.
root@ip-10-0-25-153:/home/ubuntu# mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg \
| gpg --dearmor -o /etc/apt/keyrings/docker.gpg
root@ip-10-0-25-153:/home/ubuntu#
echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] \
https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) stable" \
| tee /etc/apt/sources.list.d/docker.list > /dev/null
root@ip-10-0-25-153:/home/ubuntu# apt update
Hit:1 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:3 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease
Get:4 https://download.docker.com/linux/ubuntu noble InRelease [48.5 kB]
Get:5 https://download.docker.com/linux/ubuntu noble/stable amd64 Packages [41.1 kB]
Hit:6 http://security.ubuntu.com/ubuntu noble-security InRelease
Fetched 89.6 kB in 0s (185 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
68 packages can be upgraded. Run 'apt list --upgradable' to see them.
root@ip-10-0-25-153:/home/ubuntu#
```

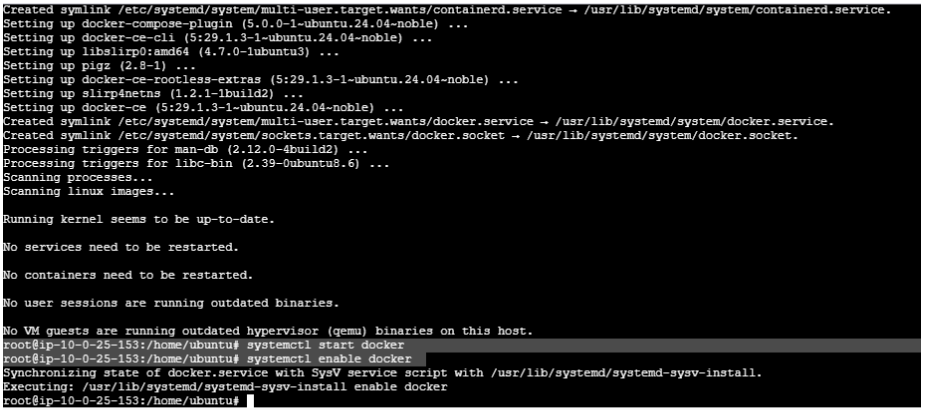
3.6 Install Docker Engine (Correct Package)

`apt install -y docker-ce docker-ce-cli containerd.io`

```
lsb-release set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 68 not upgraded.
root@ip-10-0-25-153:/home/ubuntu# mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg \
| gpg --dearmor -o /etc/apt/keyrings/docker.gpg
root@ip-10-0-25-153:/home/ubuntu#
echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] \
https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) stable" \
| tee /etc/apt/sources.list.d/docker.list > /dev/null
root@ip-10-0-25-153:/home/ubuntu# apt update
Hit:1 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:3 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease
Get:4 https://download.docker.com/linux/ubuntu noble InRelease [48.5 kB]
Get:5 https://download.docker.com/linux/ubuntu noble/stable amd64 Packages [41.1 kB]
Hit:6 http://security.ubuntu.com/ubuntu noble-security InRelease
Fetched 89.6 kB in 0s (185 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
68 packages can be upgraded. Run 'apt list --upgradable' to see them.
root@ip-10-0-25-153:/home/ubuntu# apt install -y docker-ce docker-ce-cli containerd.io
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  docker-buildx-plugin docker-ce-rootless-extras docker-compose-plugin libslirp0 pigz slirp4netns
Suggested packages:
```

3.7 Start and Enable Docker

```
systemctl start docker
systemctl enable docker
```

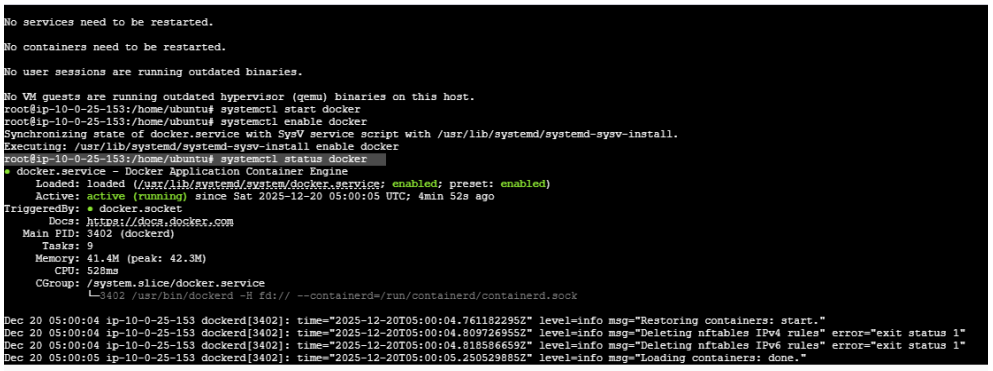


Check status:

```
systemctl status docker
```

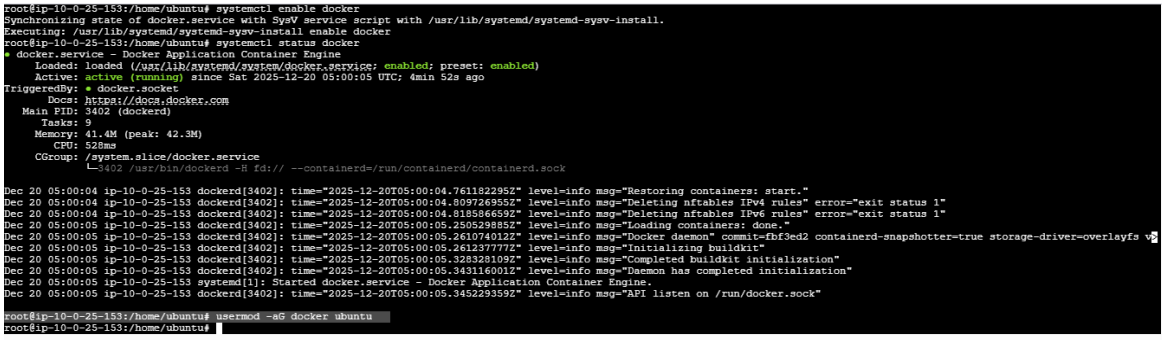
You should see:

```
active (running)
```

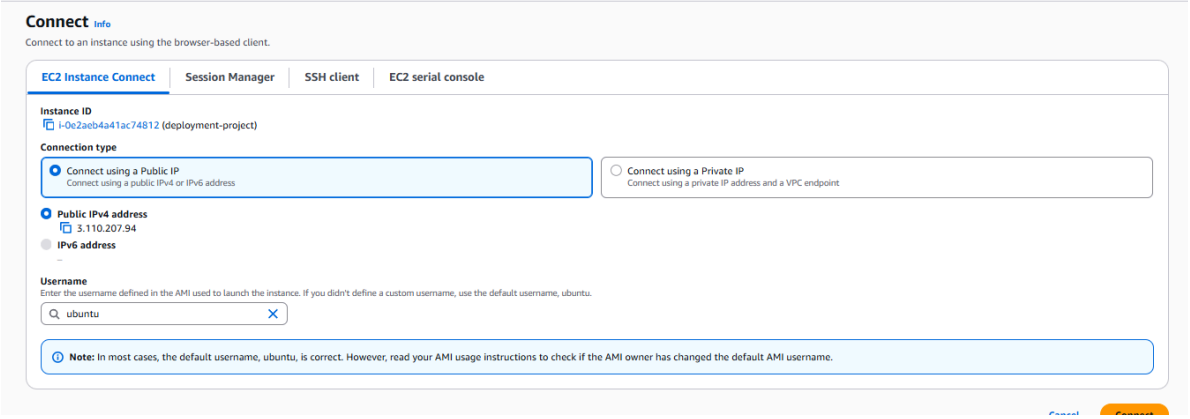


3.8 Allow Docker Without sudo (VERY IMPORTANT)

```
usermod -aG docker ubuntu
```



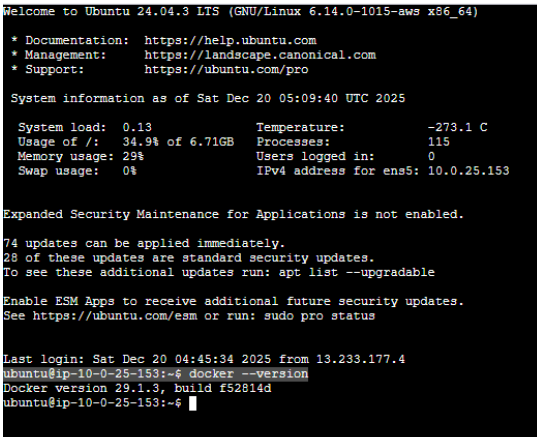
Now **disconnect and reconnect** the EC2 console session.



3.9 Verify Docker Installation

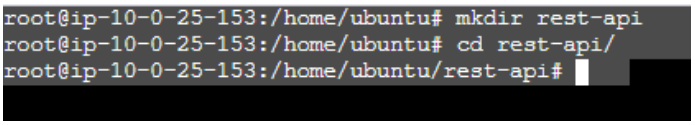
After reconnecting, run:

```
docker --version
```



STEP 4: Create REST API Project Directory

```
mkdir rest-api
cd rest-api
```



STEP 5: Create a Simple REST API (Node.js – Industry Standard)

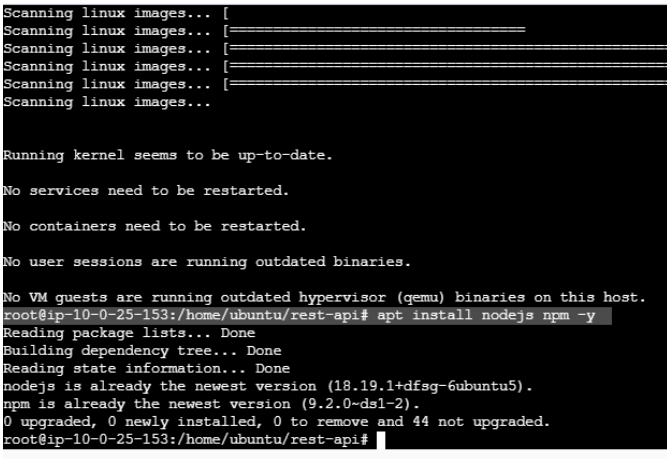
5.1 Install Node.js (Only if not installed)

Check first:

```
node -v
```

If not installed:

```
sudo apt install nodejs npm -y
```

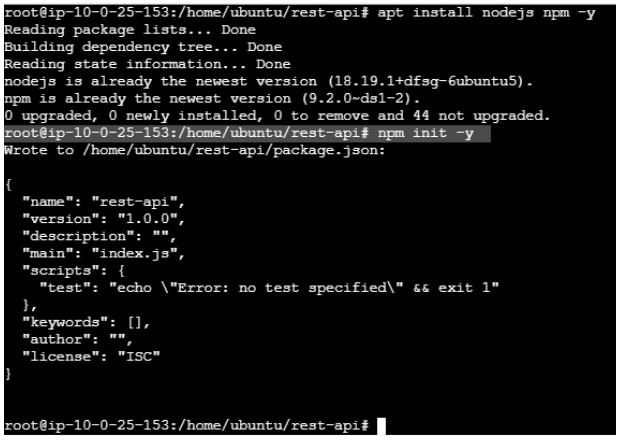


5.2 Initialize Node Project

```
npm init -y
```

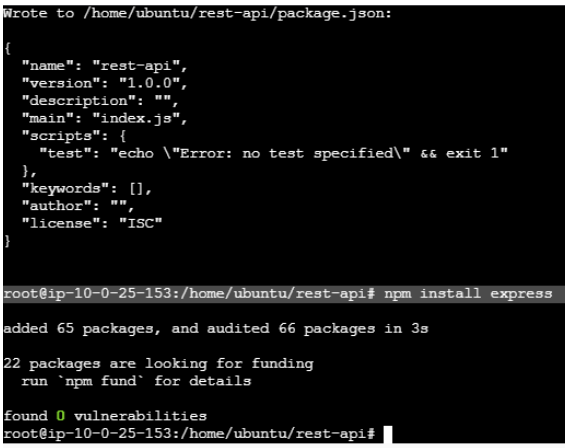
This creates:

```
package.json
```



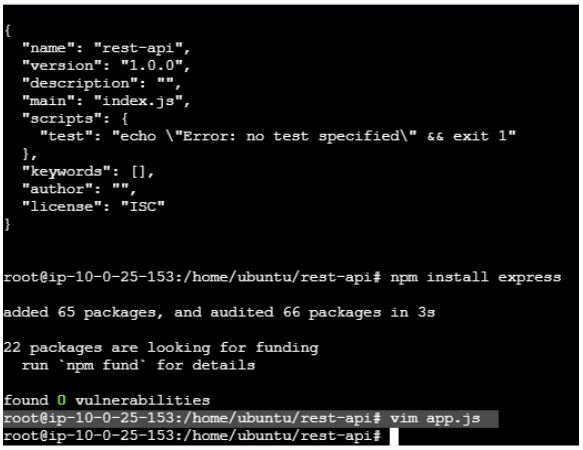
5.3 Install Express (REST framework)

```
npm install express
```



5.4 Create API File

```
vim app.js
```

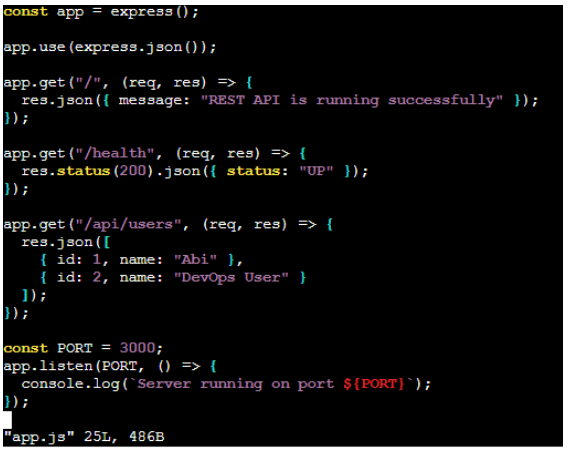


Paste this **simple REST API**:

```
const express = require("express");
const app = express();

app.use(express.json());

app.get("/", (req, res) => {
  res.json({ message: "REST API is running successfully" });
});
```

```
app.get("/health", (req, res) => {
  res.status(200).json({ status: "UP" });
});

app.get("/api/users", (req, res) => {
  res.json([
    { id: 1, name: "jeevadharsan },
    { id: 2, name: "user" }
  ]);
});

const PORT = 3000;
app.listen(3000, "0.0.0.0", () => {
  console.log("Server running on port 3000");
});
```

Save & exit.



STEP 6: Test REST API Locally (WITHOUT DOCKER)

```
node app.js
```

In another terminal tab (or stop and use curl):

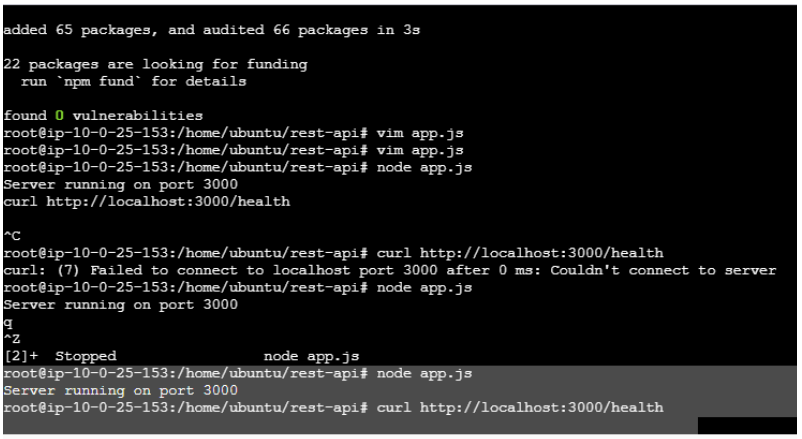
```
curl http://localhost:3000/health
```

Expected output:

```
{"status":"UP"}
```

Stop the app:

`Ctrl + C`



We don't get any output in this step, because our security group is not listening to port 3000, it's listening to 80. So, if we have a `server running on port 3000`, it's correct.

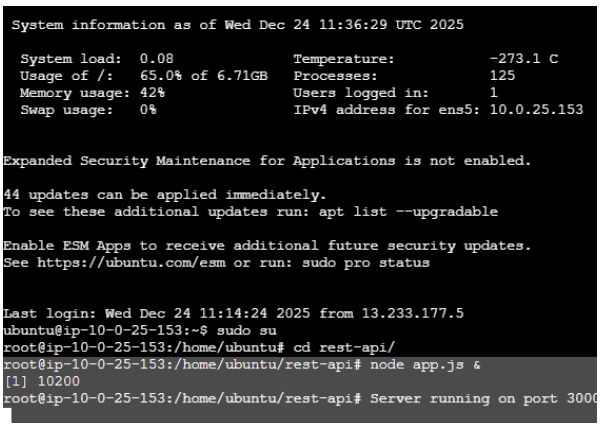
STEP 7: Create Dockerfile for REST API

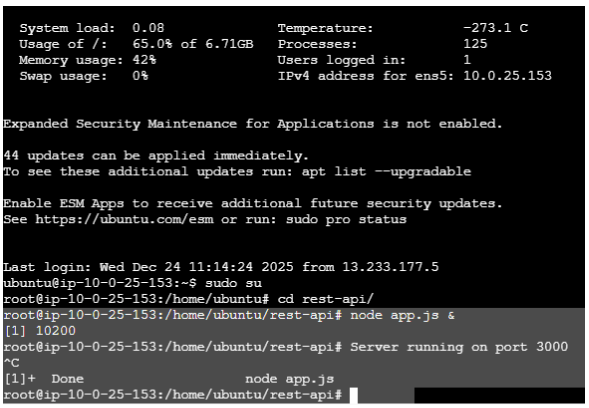
Now we move from **application validation → containerization**

STEP 7.1: Stop the API

Make sure the app is stopped before Docker build:

`Ctrl + C`





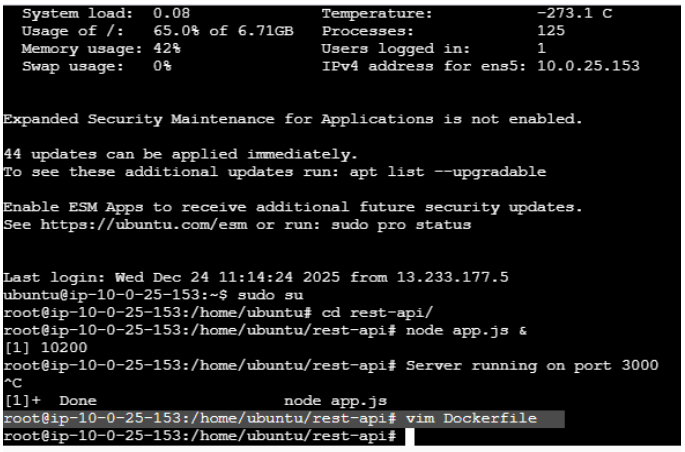
STEP 7.2: Create Dockerfile

Ensure you are inside `rest-api` directory:

```
cd ~/rest-api
```

Create Dockerfile:

```
vim Dockerfile
```

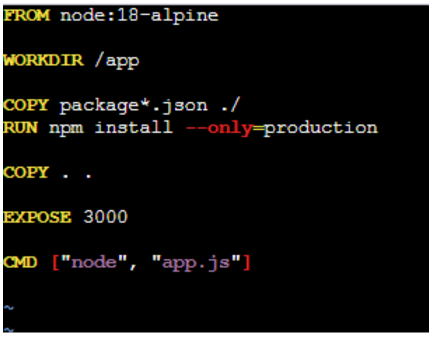


Paste this **exactly**:

```
FROM node:18-alpine
WORKDIR /app

COPY package*.json ./
RUN npm install --only=production

COPY . .
```

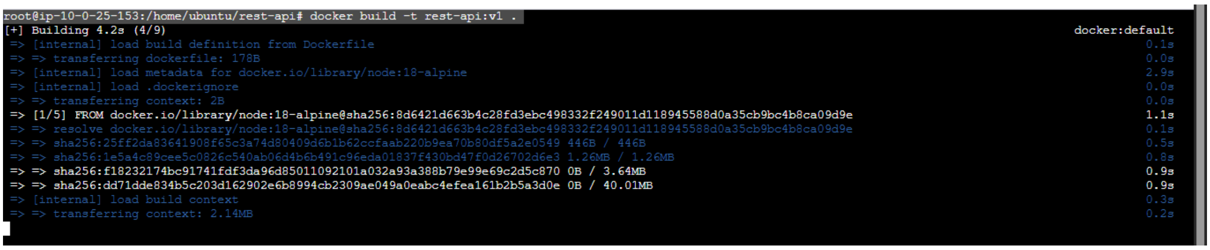
```
EXPOSE 3000
CMD ["node", "app.js"]
```

Save and exit.



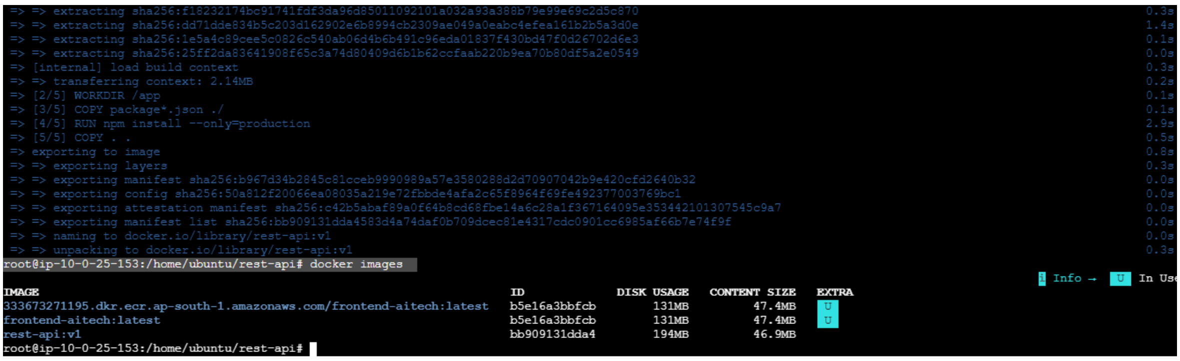
STEP 7.3: Build Docker Image

```
docker build -t rest-api:v1 .
```



Verify:

```
docker images
```



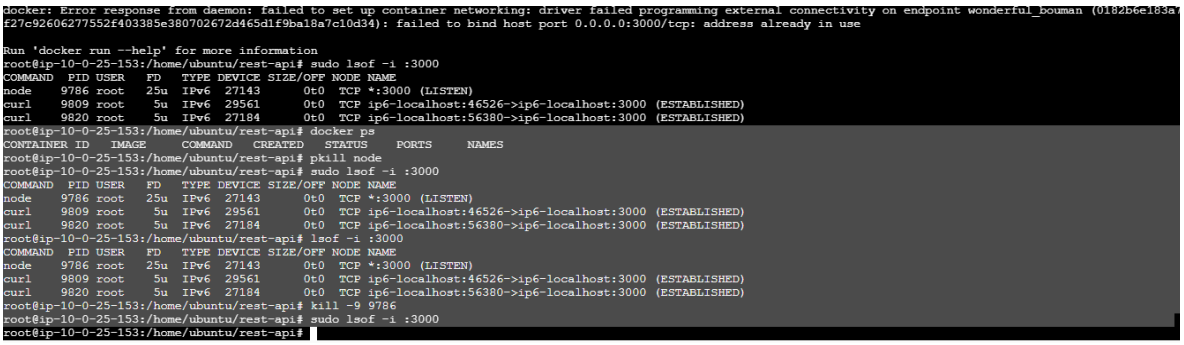
You should see:

```
rest-api    v1
```

STEP 7.4: Run Container Locally

```
docker run -d -p 3000:3000 rest-api:v1
```

When you run the above command, you'll get an error sometimes. Verify the below image, if there is any error.

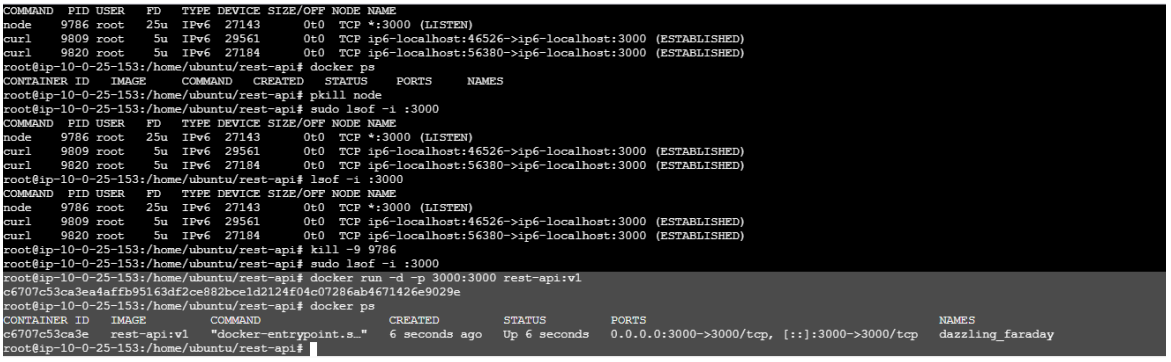


In this, kill -9 <PID>.
Again, run this command:

```
docker run -d -p 3000:3000 rest-api:v1
```

Verify container:

```
docker ps
```

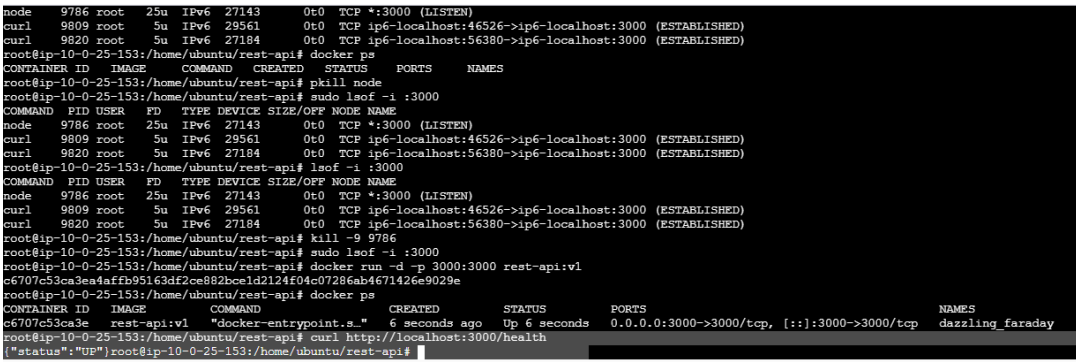


STEP 7.5: Test Containerized API

```
curl http://localhost:3000/health
```

Expected:

```
{"status":"UP"}
```



STEP 8: Push REST API Docker Image to Amazon ECR

Now we move from **local container → AWS-managed container registry**.

This step is **mandatory** for ECS Fargate.

STEP 8.1: Create ECR Repository (REST API)

AWS Console → **ECR** → **Create repository**

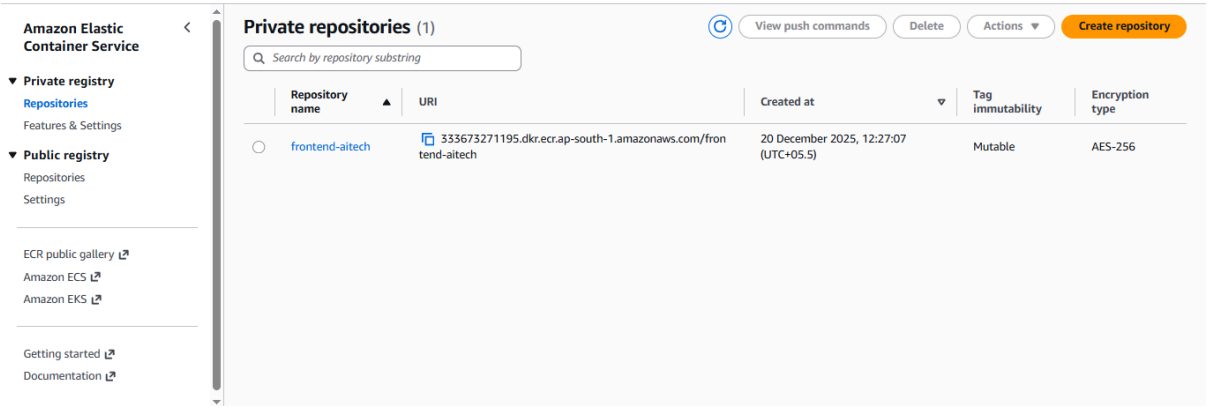
Repository name:

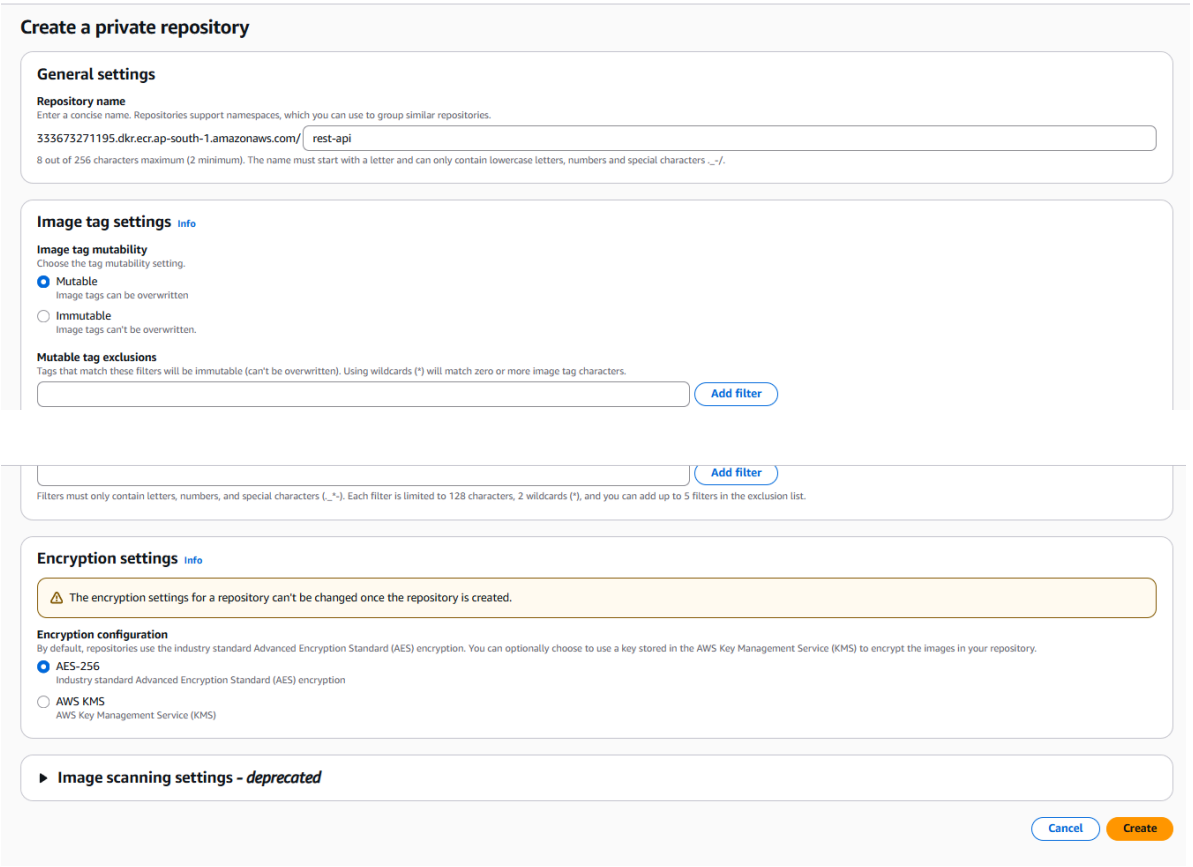
```
rest-api
```

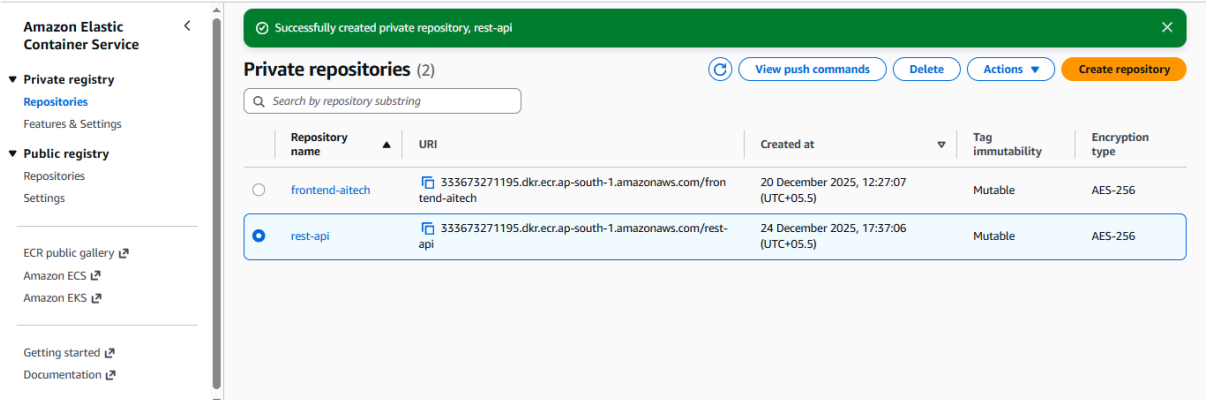
- Visibility: Private
- Encryption: Default
- Click **Create repository**

📌 Copy the repository URI, it will look like:

```
333673271195.dkr.ecr.ap-south-1.amazonaws.com/rest-api
```


Amazon Elastic Container Service
Private registry
Repositories
Features & Settings
Public registry
Repositories
Settings
ECR public gallery
Amazon ECS
Amazon EKS
Getting started
Documentation
Private repositories (1)
View push commands
Delete
Actions
Create repository
Search by repository substring
Repository name
URI
Created at
Tag immutability
Encryption type
frontend-aitech
333673271195.dkr.ecr.ap-south-1.amazonaws.com/frontend-aitech
20 December 2025, 12:27:07 (UTC+05.5)
Mutable
AES-256


Create a private repository
General settings
Repository name
Enter a concise name. Repositories support namespaces, which you can use to group similar repositories.
333673271195.dkr.ecr.ap-south-1.amazonaws.com/
rest-api
8 out of 256 characters maximum (2 minimum). The name must start with a letter and can only contain lowercase letters, numbers and special characters ._-/.
Image tag settings Info
Image tag mutability
Choose the tag mutability setting.
Mutable
Image tags can be overwritten
Immutable
Image tags can't be overwritten.
Mutable tag exclusions
Tags that match these filters will be immutable (can't be overwritten). Using wildcards (*) will match zero or more image tag characters.
Add filter
Add filter
Filters must only contain letters, numbers, and special characters (._*-). Each filter is limited to 128 characters, 2 wildcards (*), and you can add up to 5 filters in the exclusion list.
Encryption settings Info
The encryption settings for a repository can't be changed once the repository is created.
Encryption configuration
By default, repositories use the industry standard Advanced Encryption Standard (AES) encryption. You can optionally choose to use a key stored in the AWS Key Management Service (KMS) to encrypt the images in your repository.
AES-256
Industry standard Advanced Encryption Standard (AES) encryption
AWS KMS
AWS Key Management Service (KMS)
Image scanning settings - deprecated
Cancel
Create


Amazon Elastic Container Service
Private registry
Repositories
Features & Settings
Public registry
Repositories
Settings
ECR public gallery
Amazon ECS
Amazon EKS
Getting started
Documentation
Successfully created private repository, rest-api
Private repositories (2)
View push commands
Delete
Actions
Create repository
Search by repository substring
Repository name
URI
Created at
Tag immutability
Encryption type
frontend-aitech
333673271195.dkr.ecr.ap-south-1.amazonaws.com/frontend-aitech
20 December 2025, 12:27:07 (UTC+05.5)
Mutable
AES-256
rest-api
333673271195.dkr.ecr.ap-south-1.amazonaws.com/rest-api
24 December 2025, 17:37:06 (UTC+05.5)
Mutable
AES-256

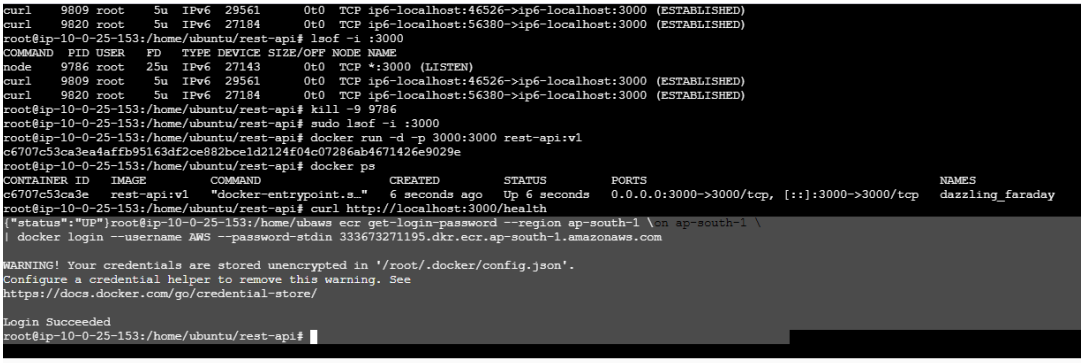
STEP 8.2: Authenticate Docker to ECR

Run on EC2:

```
aws ecr get-login-password --region ap-south-1 \
| docker login --username AWS --password-stdin
333673271195.dkr.ecr.ap-south-1.amazonaws.com
```

Expected:

```
Login Succeeded
```

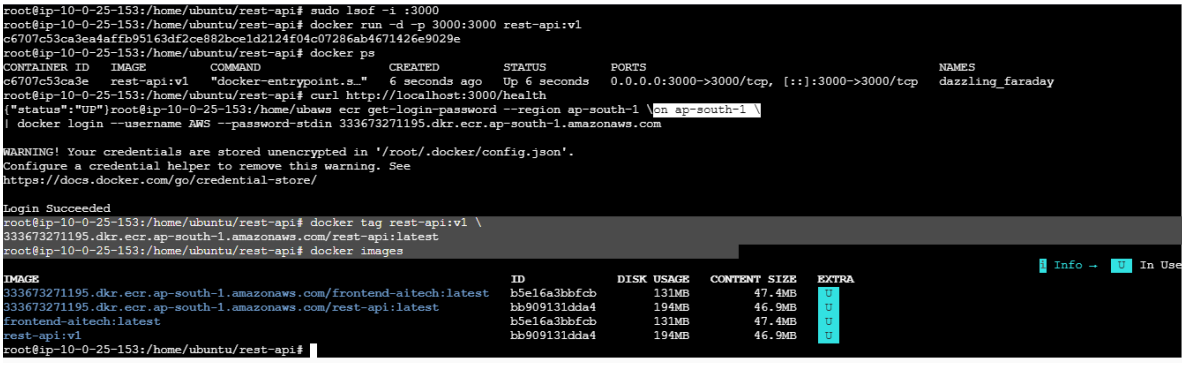


STEP 8.3: Tag REST API Image

```
docker tag rest-api:v1 \
333673271195.dkr.ecr.ap-south-1.amazonaws.com/rest-api:latest
```

Verify:

```
docker images
```



STEP 8.4: Push Image to ECR

```
docker push
333673271195.dkr.ecr.ap-south-1.amazonaws.com/rest-api:latest
```

```
https://docs.docker.com/go/credential-store/

Login Succeeded
root@ip-10-0-25-153:/home/ubuntu/rest-api# docker tag rest-api:v1 \
333673271195.dkr.ecr.ap-south-1.amazonaws.com/rest-api:latest
root@ip-10-0-25-153:/home/ubuntu/rest-api# docker images
                                                                                                    i Info →   U In Use
IMAGE                                                                ID             DISK USAGE   CONTENT SIZE   EXTRA
333673271195.dkr.ecr.ap-south-1.amazonaws.com/frontend-aitech:latest   b5e16a3bbfcb       131MB         47.4MB    U
333673271195.dkr.ecr.ap-south-1.amazonaws.com/rest-api:latest          bb909131dda4       194MB         46.9MB    U
frontend-aitech:latest                                               b5e16a3bbfcb       131MB         47.4MB    U
rest-api:v1                                                          bb909131dda4       194MB         46.9MB    U
root@ip-10-0-25-153:/home/ubuntu/rest-api# docker push 333673271195.dkr.ecr.ap-south-1.amazonaws.com/rest-api:latest
The push refers to repository [333673271195.dkr.ecr.ap-south-1.amazonaws.com/rest-api]
36124f414498: Pushed
11b279d0ef34: Pushed
f18232174bc9: Pushed
dd71dde834b5: Pushed
ba40c37a9e9c: Pushed
25ff2da83641: Pushed
b83a1367db4e: Pushed
c22ed34e5763: Pushed
1e5a4c89cee5: Pushed
latest: digest: sha256:bb909131dda4583d4a74daf0b709dcec81e4317cdc0901cc6985af66b7e74f9f size: 856
root@ip-10-0-25-153:/home/ubuntu/rest-api#
```

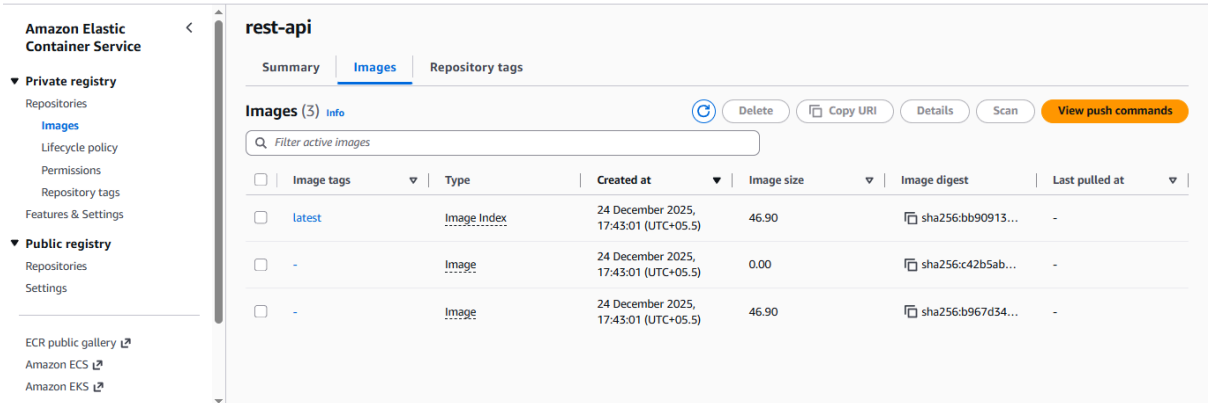
STEP 8.5: Verify in AWS Console

Go to:

`ECR → rest-api → Images`

You should see:

- Image tag: `latest`
- Status: Available

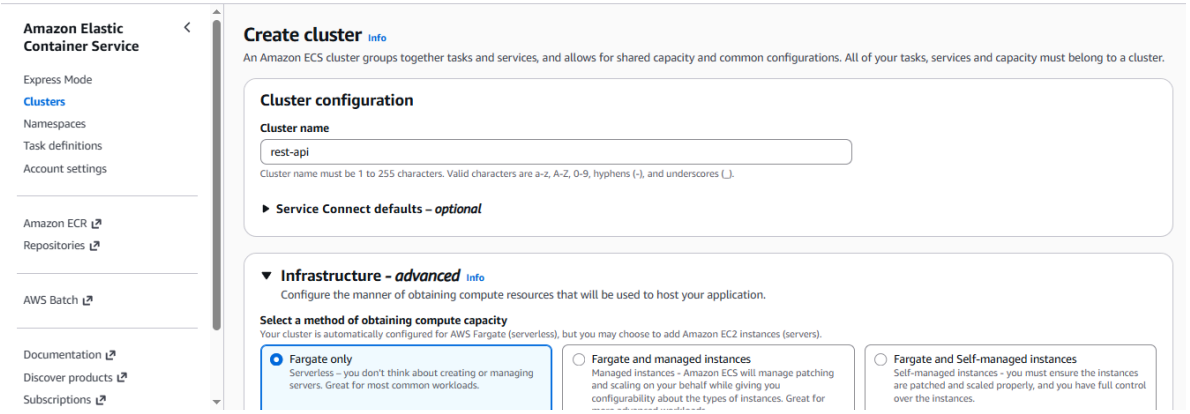


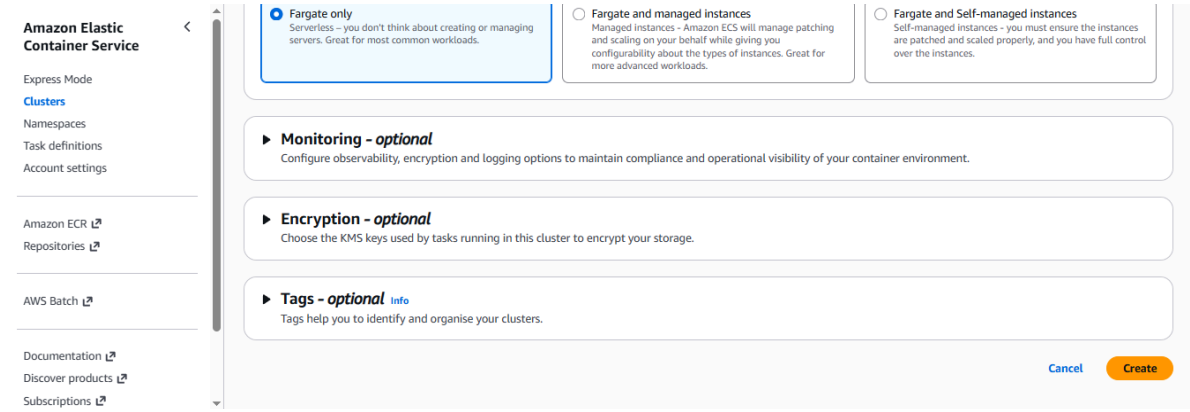
STEP 9: Create ECS Task Definition (REST API)

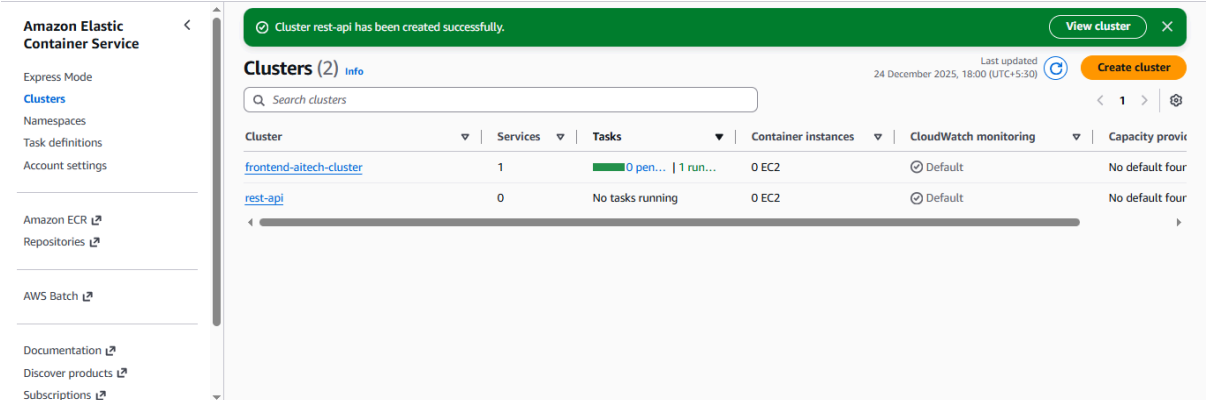
The **task definition** is the **blueprint** that tells ECS:

- Which image to run

- Which port to expose
- How much CPU/RAM to allocate
- How logs are handled







STEP 9.1: Open Task Definitions

AWS Console → **ECS** → **Task definitions** → **Create new task definition**

Choose:

- **Launch type:** `AWS Fargate`

STEP 9.2: Task Definition Configuration

Setting	Value
Task definition family	`rest-api-task`
Launch type	AWS Fargate
Operating system	Linux
Architecture	x86_64
Network mode	`awsvpc`

STEP 9.3: Task Size

Resource	Value
CPU	`0.25 vCPU`
Memory	`0.5 GB`

(Perfect for a lightweight REST API)

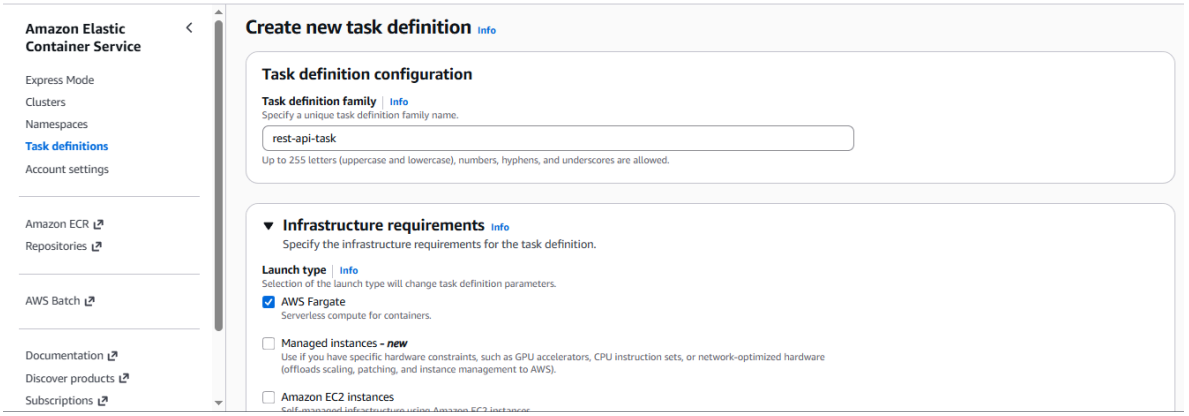
STEP 9.4: Task Execution Role

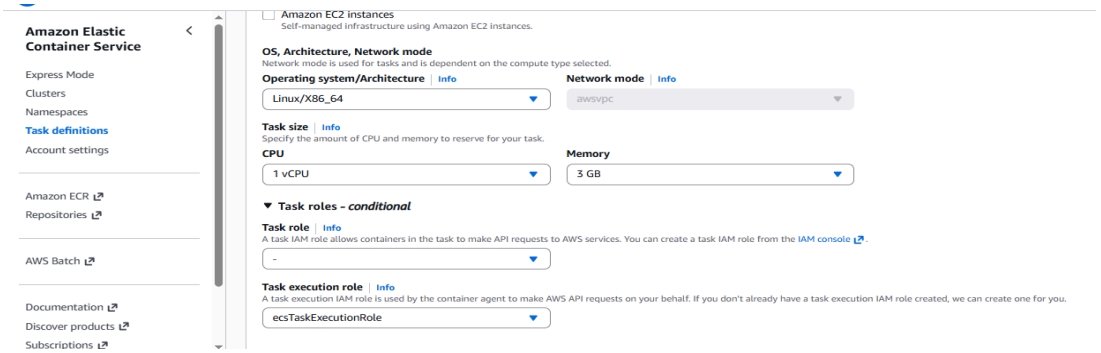
When prompted:

✔ Select **Create new role**
(or)
✔ Select **ecsTaskExecutionRole** (recommended)

This role allows ECS to:

- Pull image from **ECR**
- Send logs to **CloudWatch**





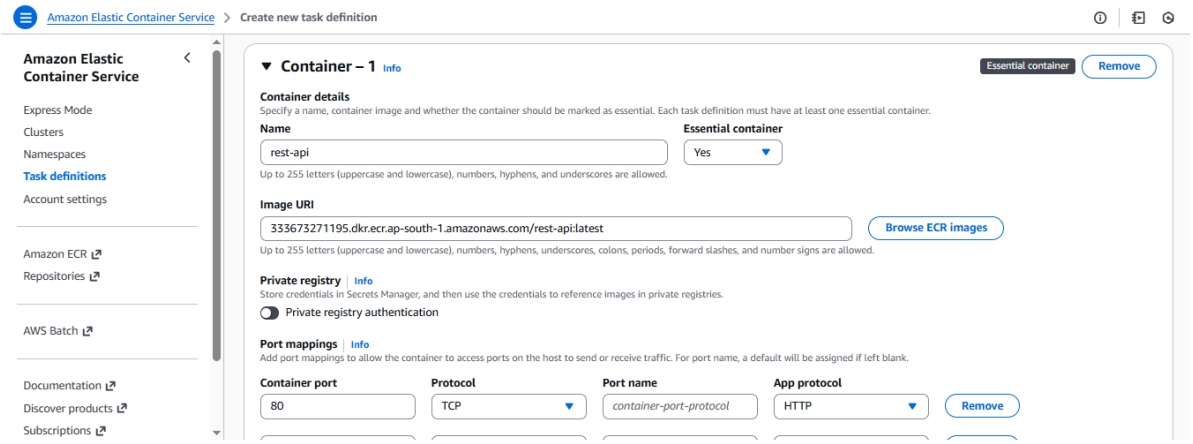
STEP 9.5: Container Configuration

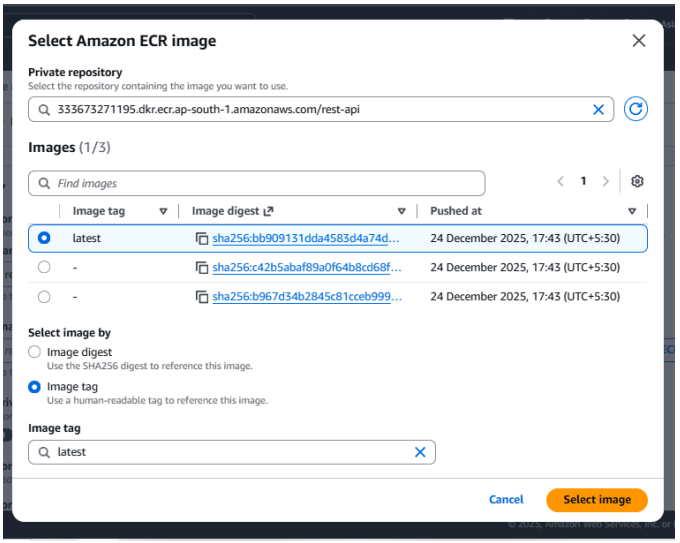
Click **Add container**

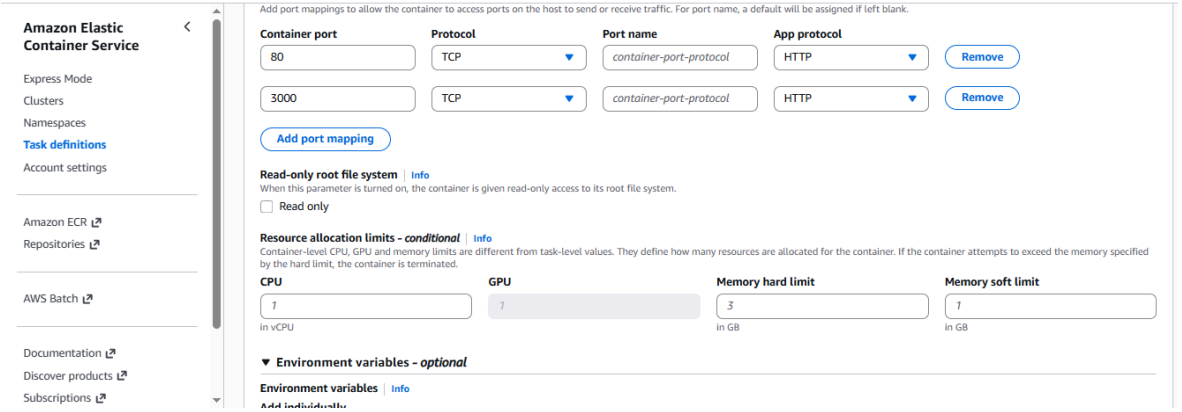
Field	Value
Container name	`rest-api`
Image URI	`333673271195.dkr.ecr.ap-south-1.amazonaws.com/rest-api:latest`
Essential	Yes

Port Mapping

Setting	Value
Container port	`3000`
Protocol	TCP





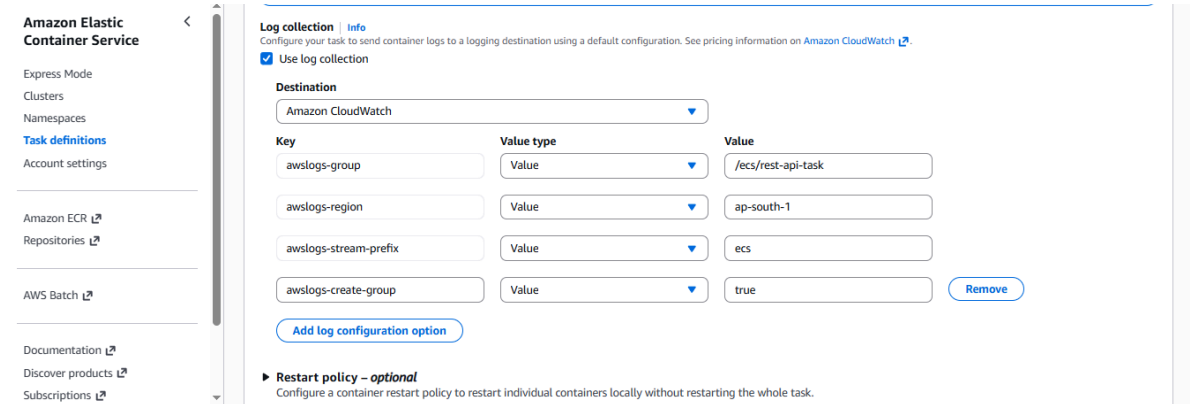


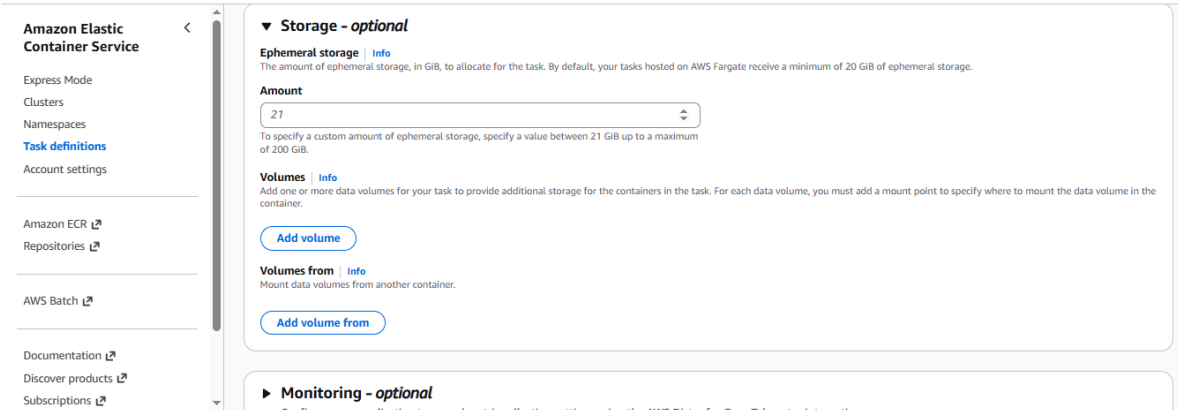
STEP 9.6: Logging (IMPORTANT)

Enable **CloudWatch Logs**

Setting	Value
Log group	`/ecs/rest-api-task`

Region	`ap-south-1`
Stream prefix	`ecs`
Auto-create group	Yes



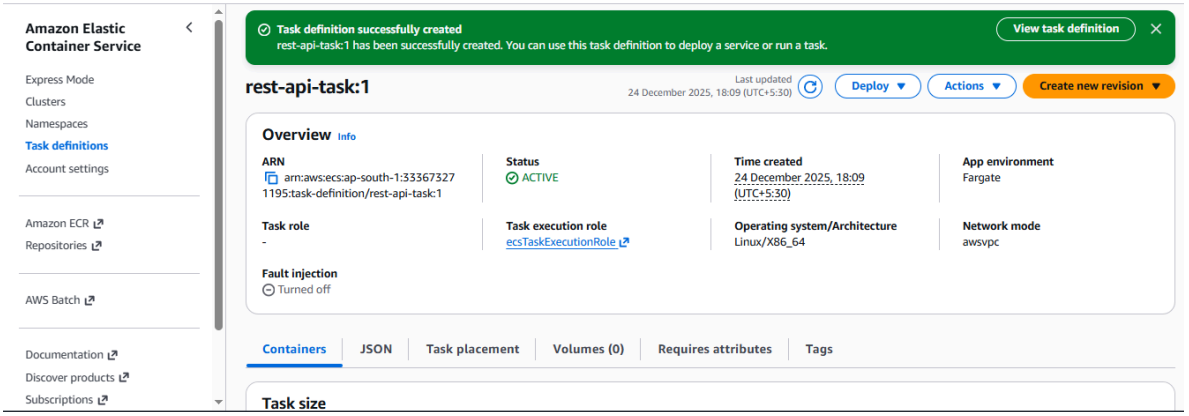


STEP 9.7: Create Task Definition

Click **Create**

✅ You should now see:

`rest-api-task : Revision 1`



STEP 10: Create ECS Service (REST API) + Attach to ALB

The **ECS Service**:

- Keeps your REST API running
- Replaces crashed containers automatically
- Connects your API to the **Application Load Balancer**

STEP 10.1: Create ECS Service

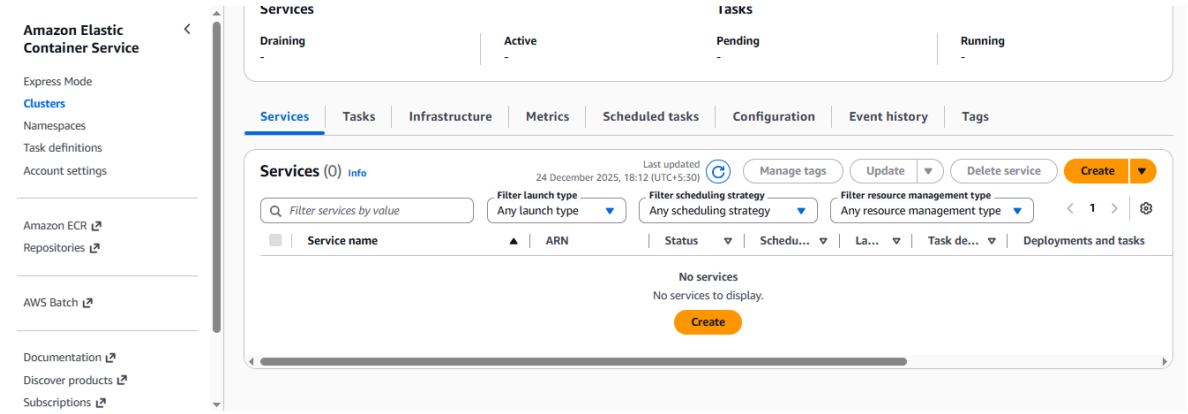
Go to:

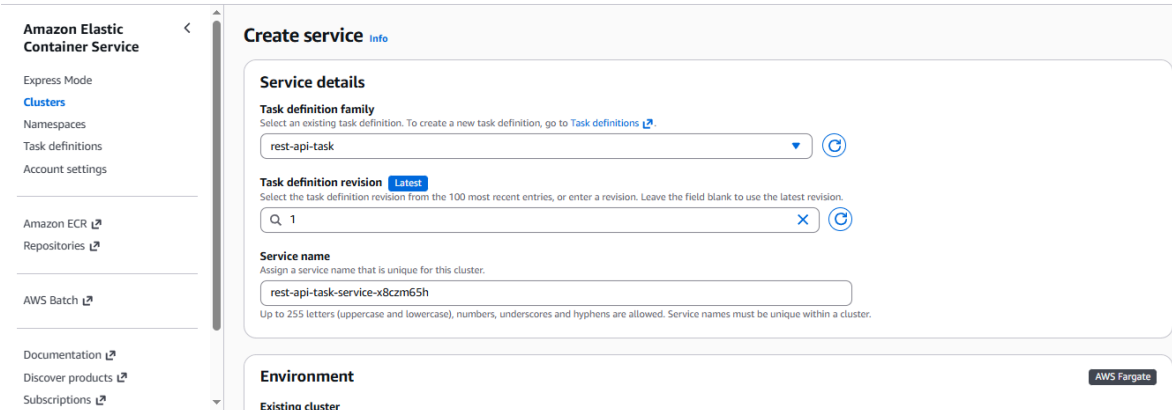
AWS Console → ECS → Clusters → *your cluster* → Create service

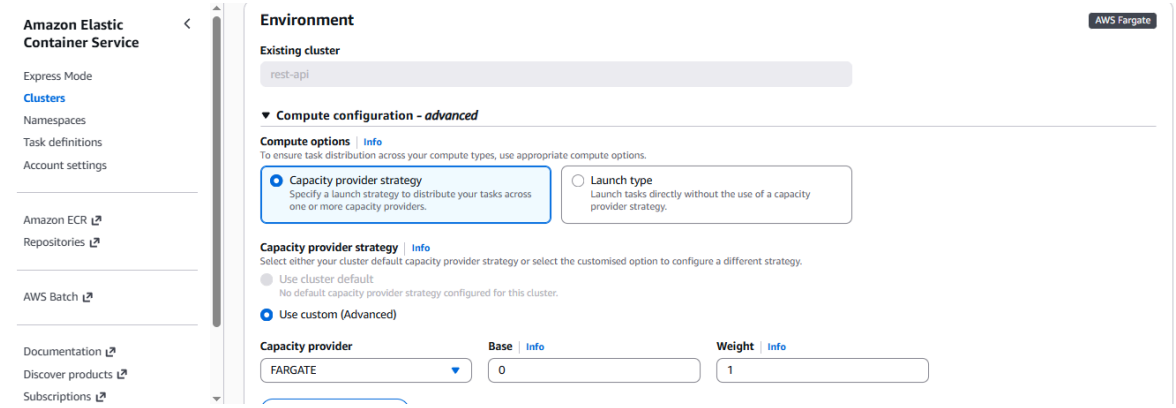
Choose:

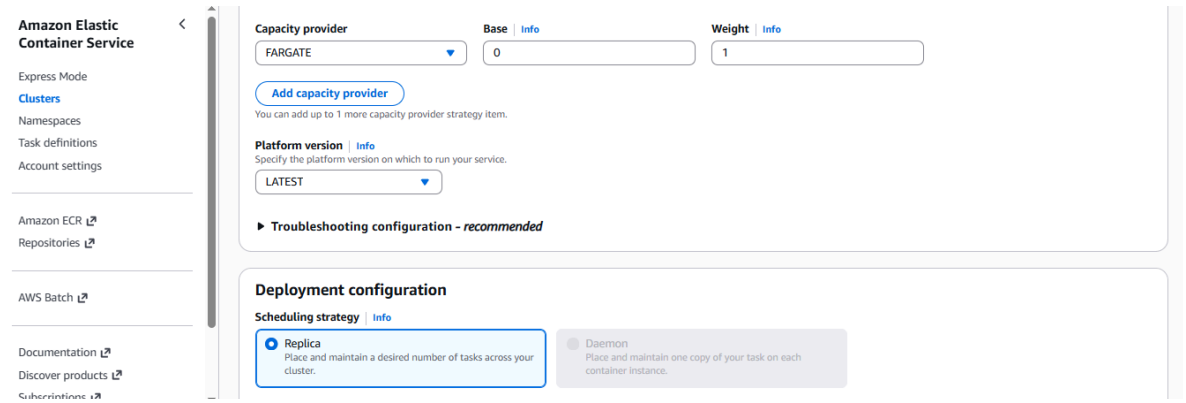
Setting	Value
Launch type	**Fargate**
Task definition family	`rest-api-task`
Revision	Latest
Service name	`rest-api-service`

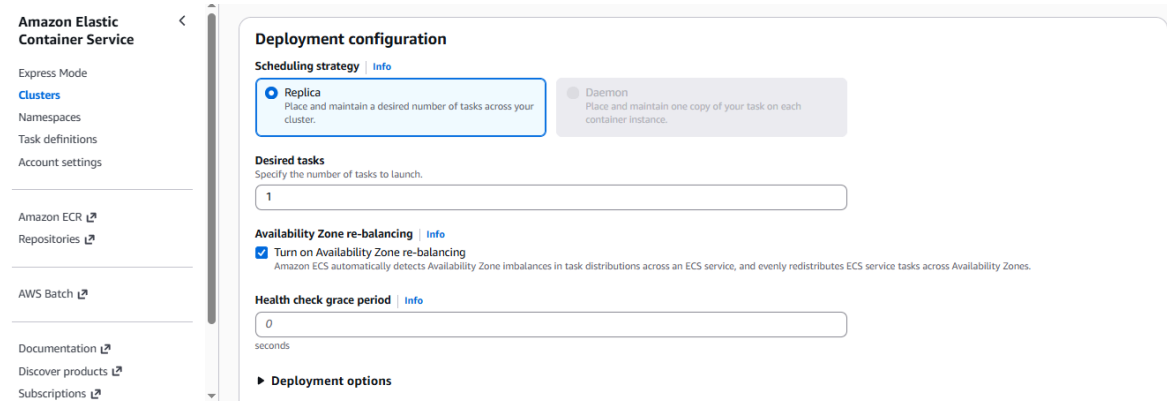
Desired tasks	1









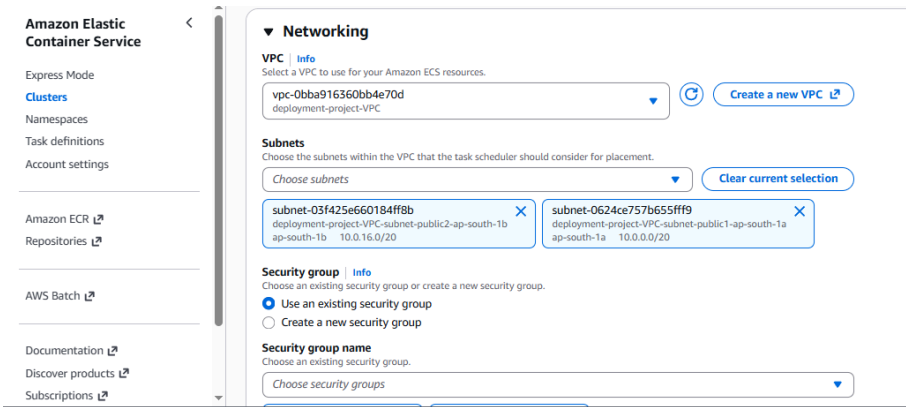


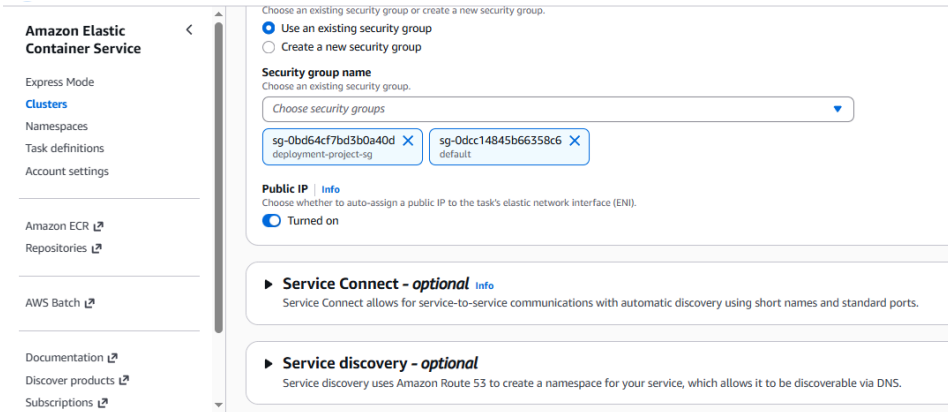
STEP 10.2: Networking Configuration

Setting	Value
VPC	`deployment-project-VPC`
Subnets	**Public subnets (same as ALB)**
Security group	`deployment-project-sg`
Auto-assign public IP	**Enabled**

Make sure:

- Subnets are **public**
- Security group allows **port 3000** from ALB SG



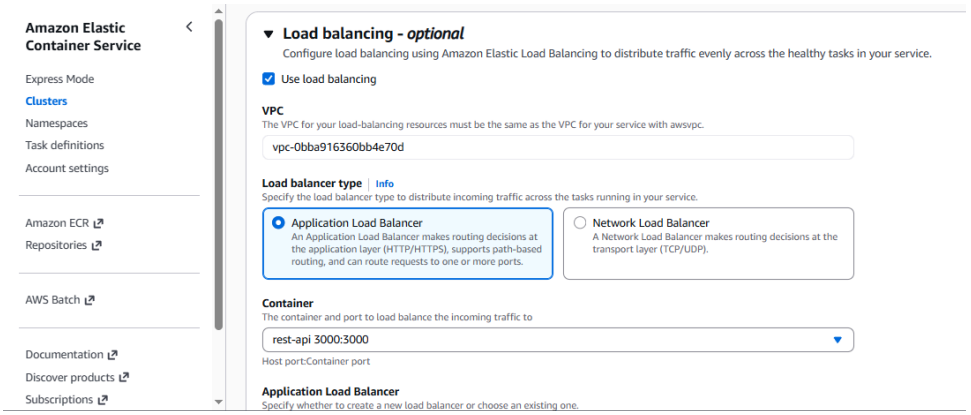


STEP 10.3: Load Balancer Configuration (IMPORTANT)

Enable load balancing.

Choose:

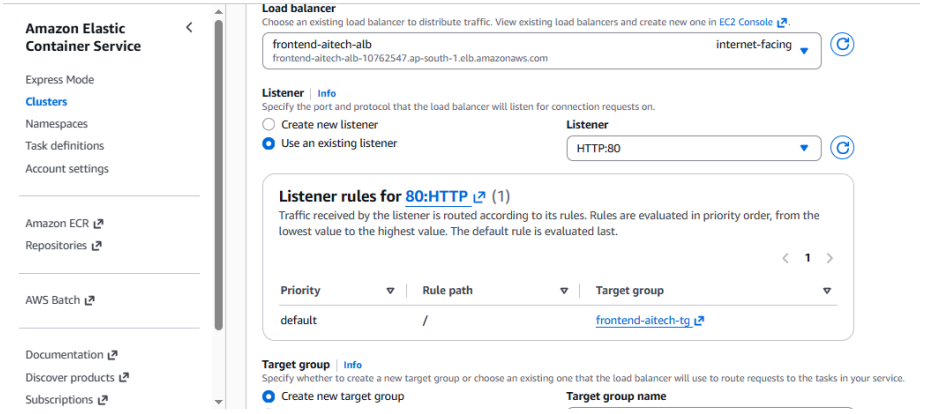
- **Application Load Balancer**
- Select existing ALB:
  `frontend-aitech-alb`

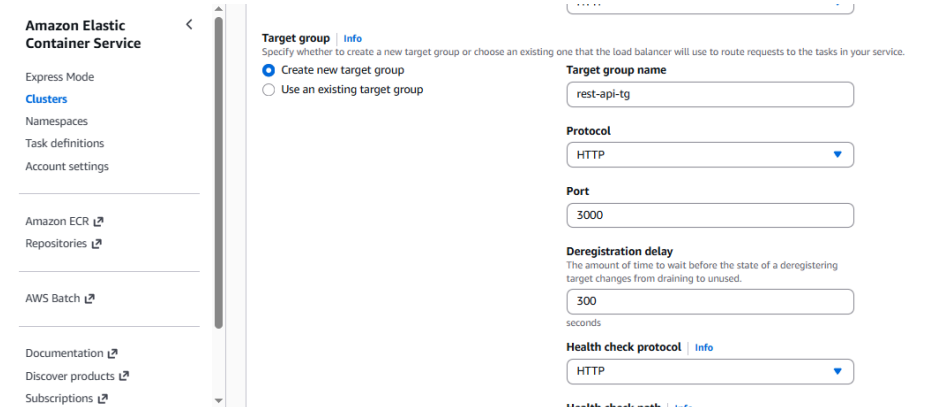


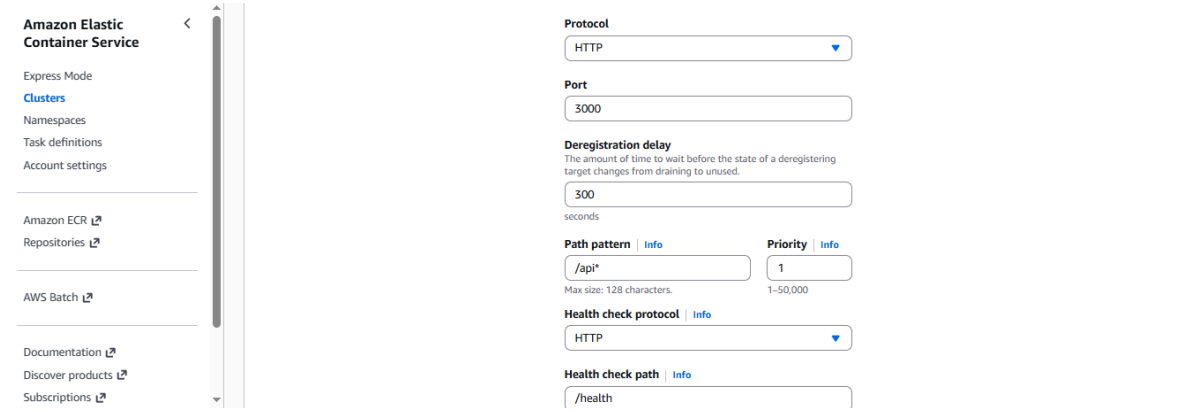
STEP 10.4: Target Group Configuration

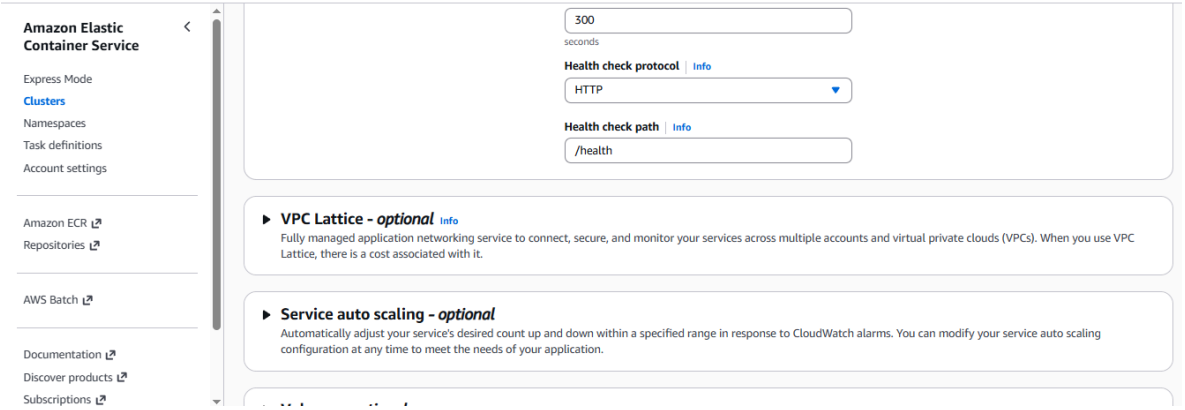
Setting	Value
Target type	**IP**
Target group name	`rest-api-tg`
Protocol	HTTP
Port	`3000`
Health check path	`/health`

✔ This is critical — your API health endpoint is `/health`







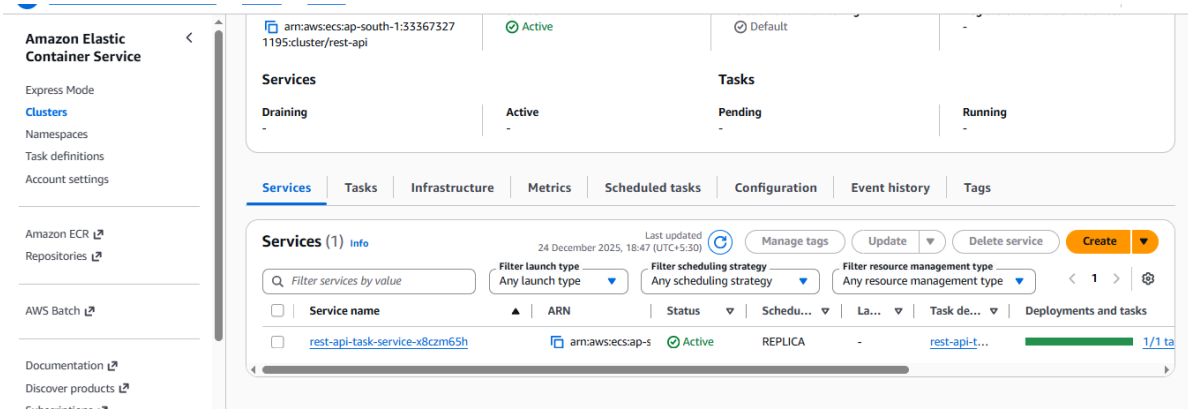


STEP 10.6: Create Service

Click **Create service**

Wait until:

- Desired tasks: 1
- Running tasks: 1
- Health status: **Healthy**



✅ VERIFY DEPLOYMENT

1️⃣ Get ALB DNS name

```
frontend-aitech-alb-xxxx.ap-south-1.elb.amazonaws.com
```

2️⃣ Test REST API via ALB

```
http://<ALB-DNS>/api/health
```

Expected output:

```
{"status":"UP"}
```