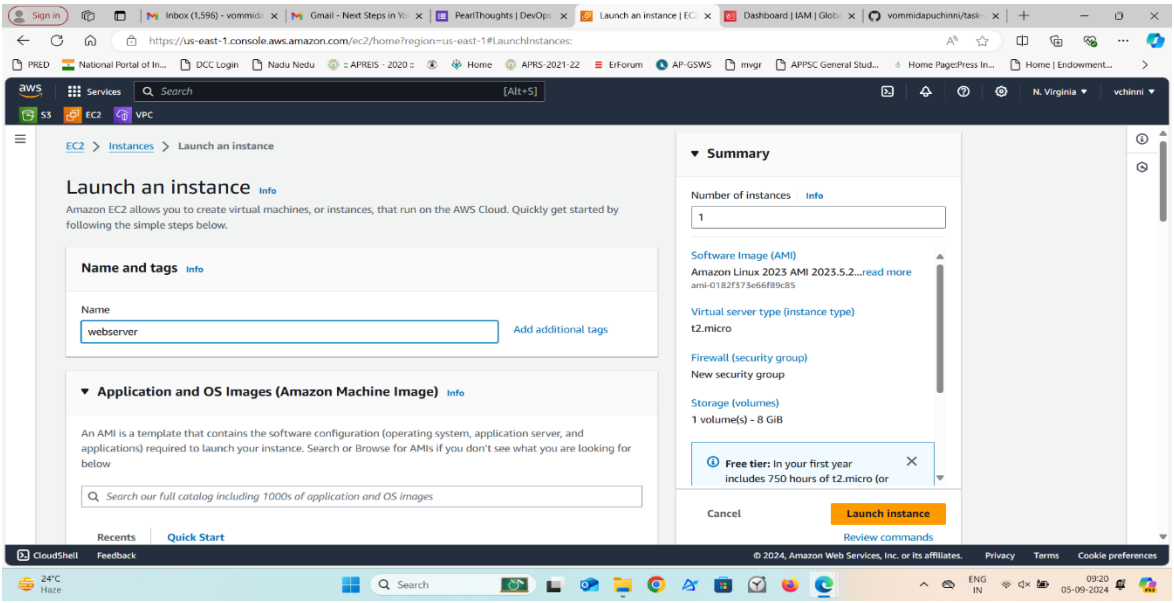
Medusa Backend Deployment on AWS Using Terraform, ECS, and GitHub Actions

Git repo: https://github.com/vommidapuchinni/pearl-task.git

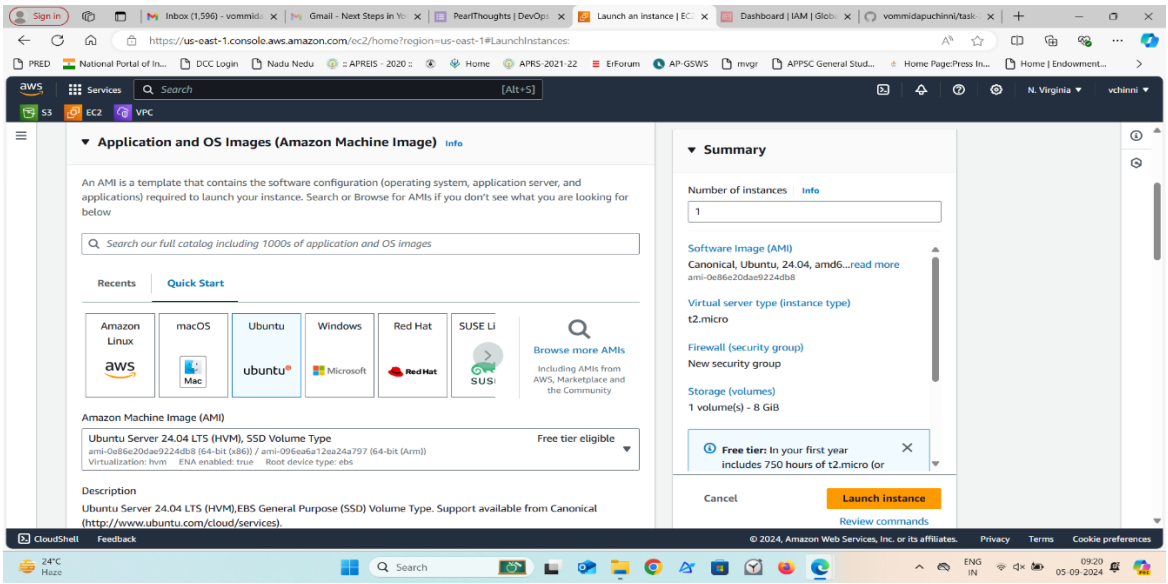
Step 1: Launch EC2 Instances

Launch the Instance:

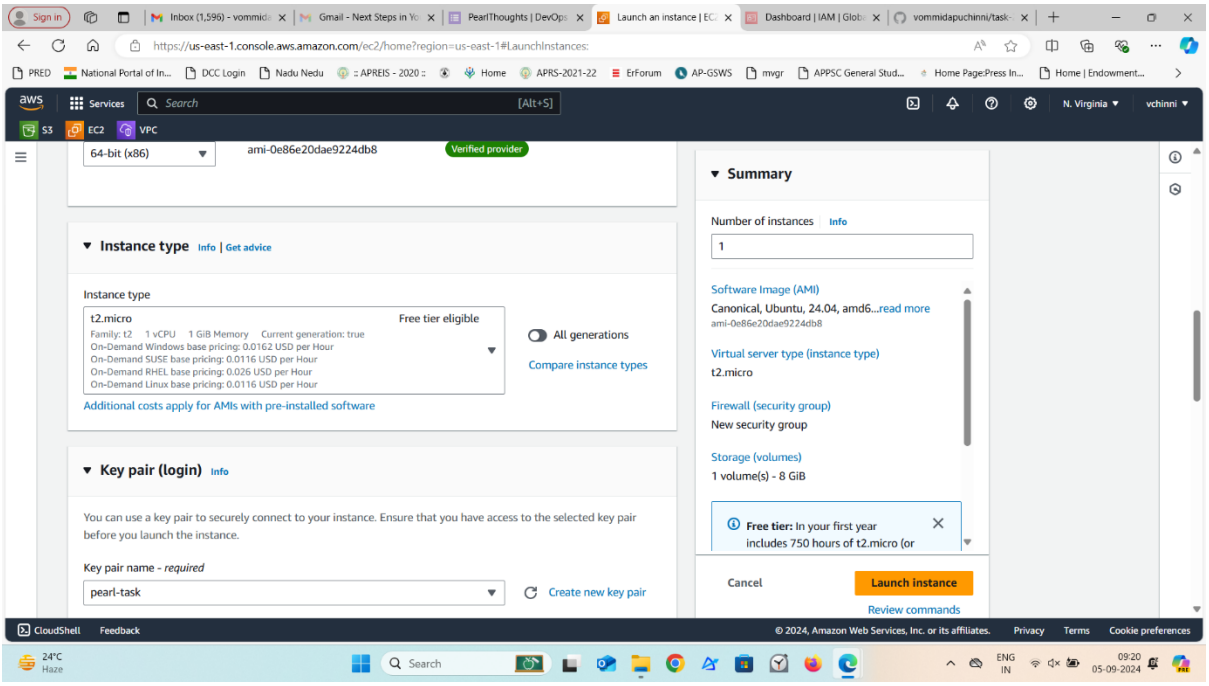
- Go to the AWS EC2 Console.
- Click Launch Instance
- Name and Tags: Name the instance webserver.

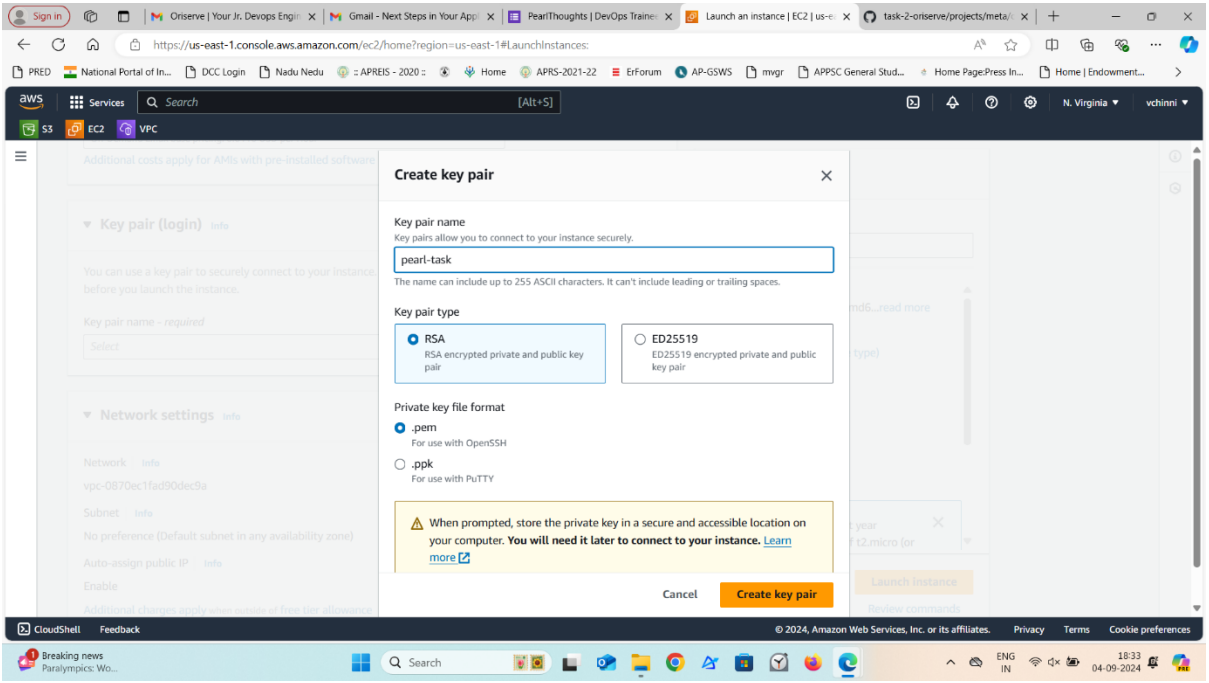


- AMI: Choose Ubuntu Server 20.04 LTS (or any preferred version).

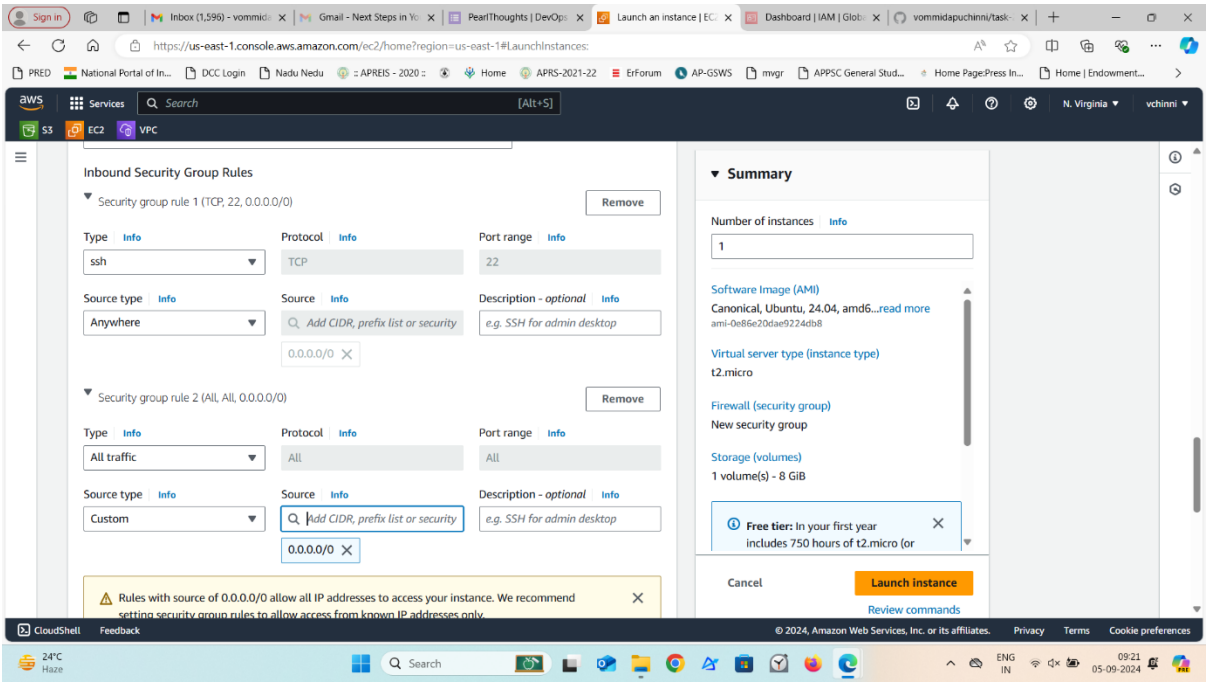


- Instance Type: Choose t2.micro for testing (eligible for free tier).
- Key Pair: create a new one.

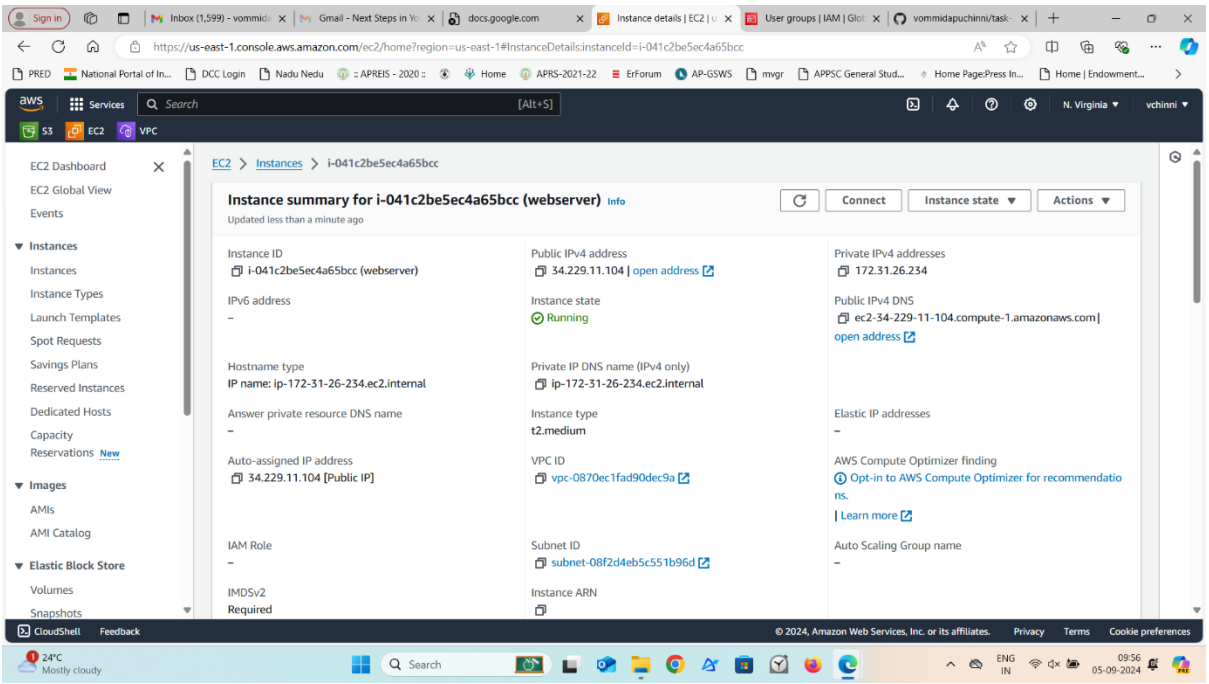




- Network Settings:
  - Create a new security group allowing:
    - SSH (port 22) from your IP for secure access.
    - I gave all traffic.

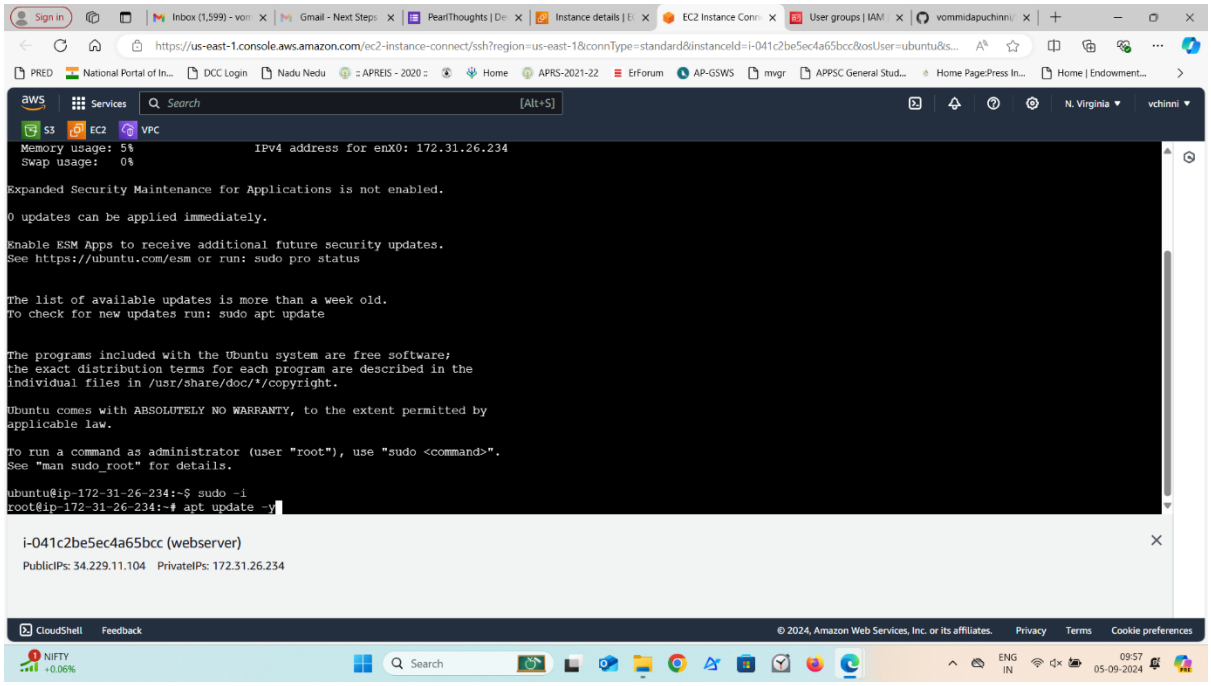


- Click Launch Instance.
- Connect to the Instance

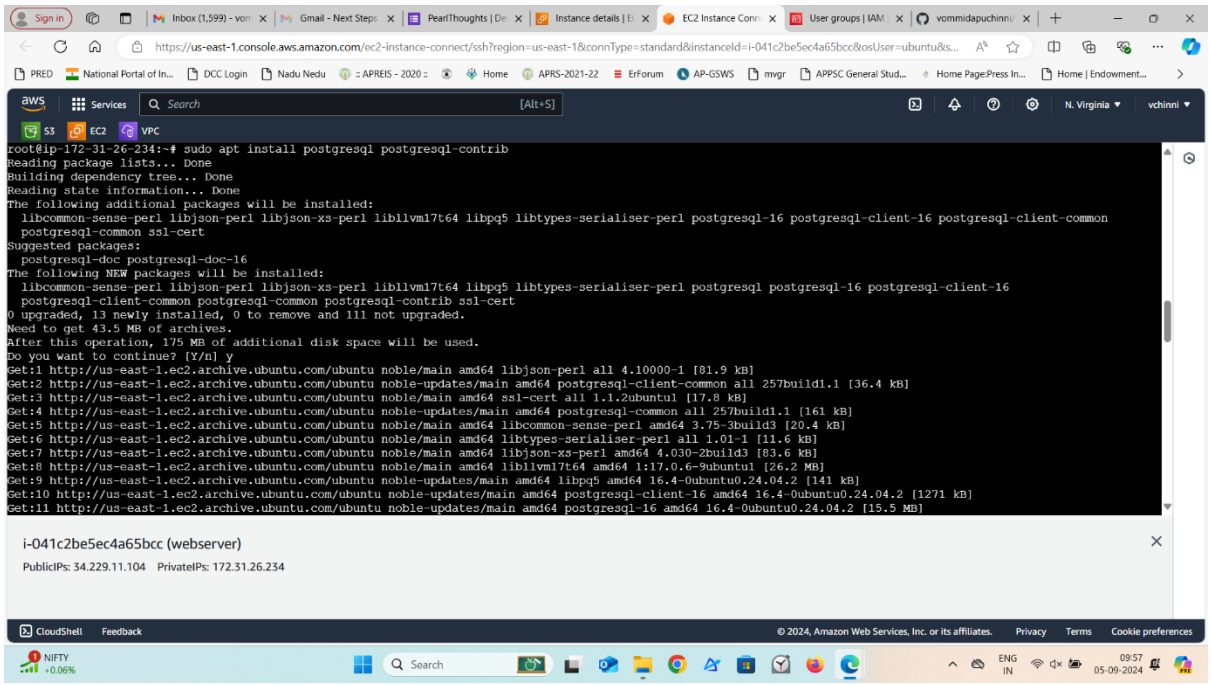


Step 2: Set Up PostgreSQL

1. Install PostgreSQL: Ensure PostgreSQL is installed on your server. If not, install it using:
   - sudo apt update



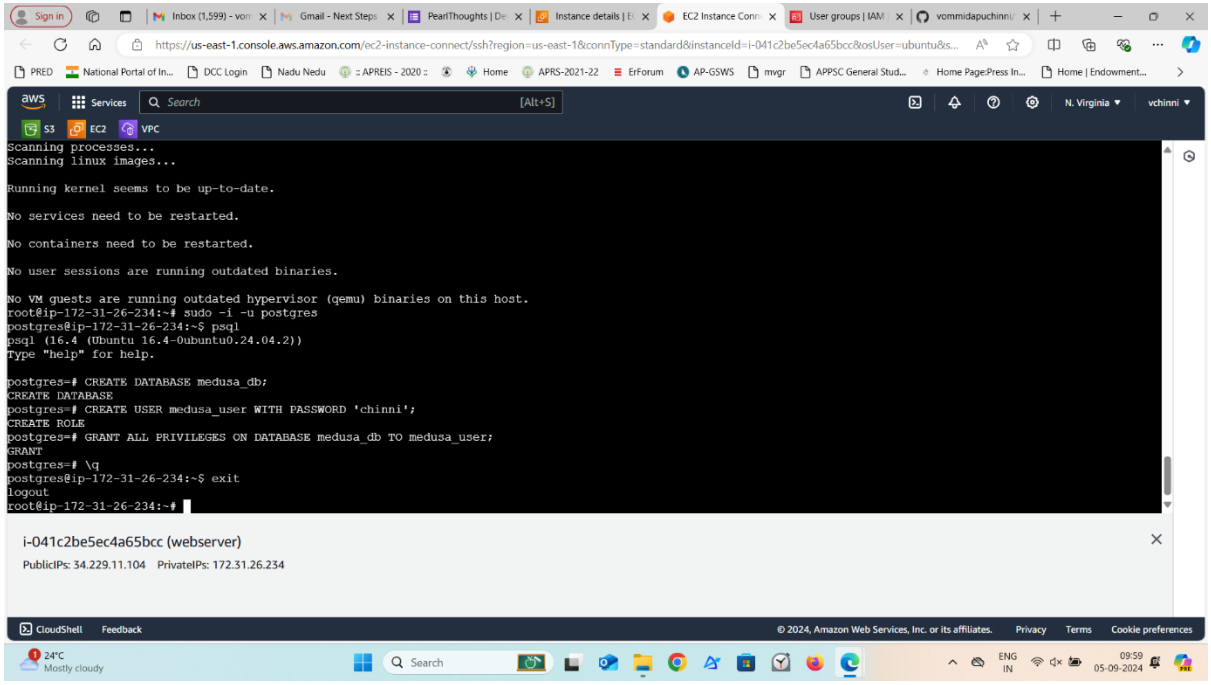
- sudo apt install postgresql postgresql-contrib



2. Start and Enable PostgreSQL: Start and enable PostgreSQL to run on boot:
   - sudo systemctl start postgresql
   - sudo systemctl enable postgresql
3. Switch to the PostgreSQL User: Switch to the postgres user:
   - sudo -i -u postgres
4. Access PostgreSQL CLI: Launch the psql command-line interface:

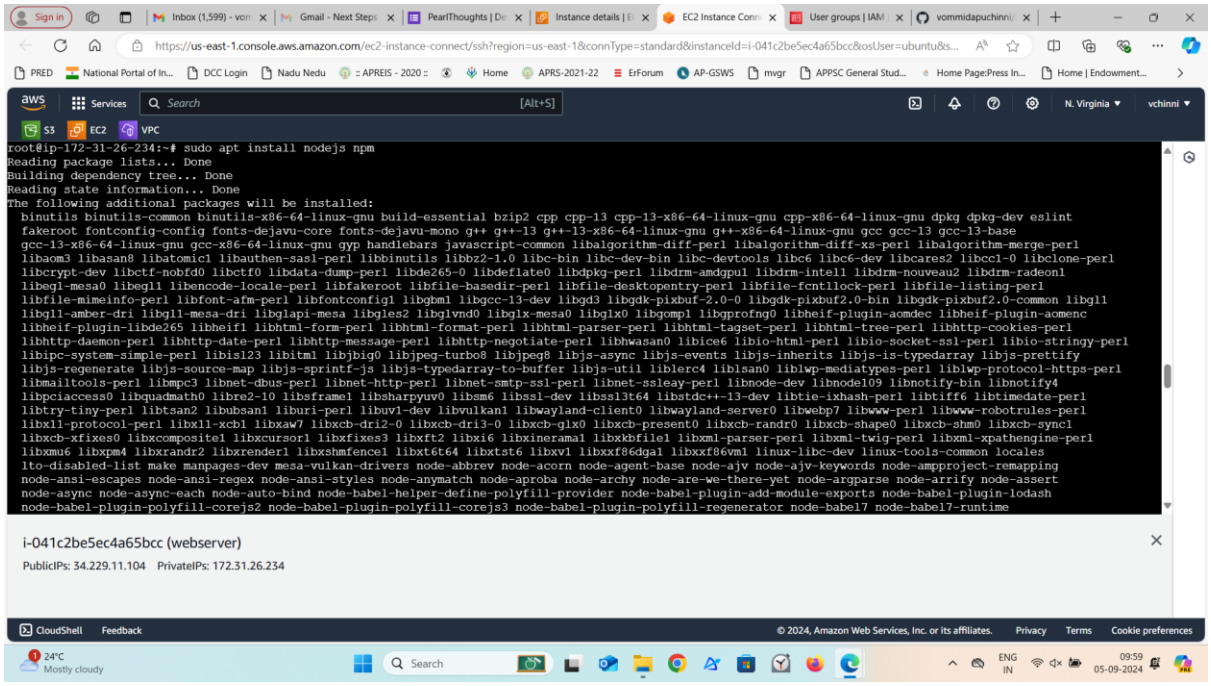
- psql

5. Create a Database and User: Execute the following commands in the psql prompt:
   - CREATE DATABASE medusa_db;
   - CREATE USER medusa_user WITH ENCRYPTED PASSWORD 'your_password';
   - GRANT ALL PRIVILEGES ON DATABASE medusa_db TO medusa_user;
   - \q
   - Replace 'your_password' with a strong password of your choice.
6. Exit PostgreSQL User: Exit from the postgres user:
   - Exit

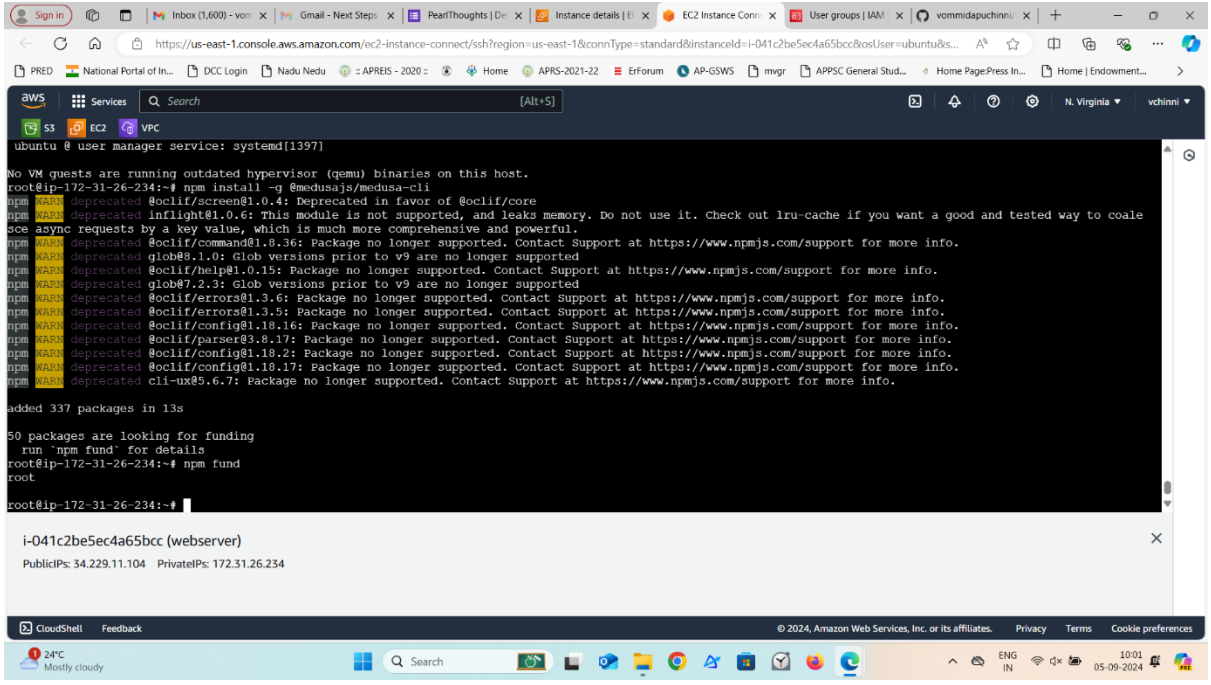


Step 3: Set Up Medusa Backend

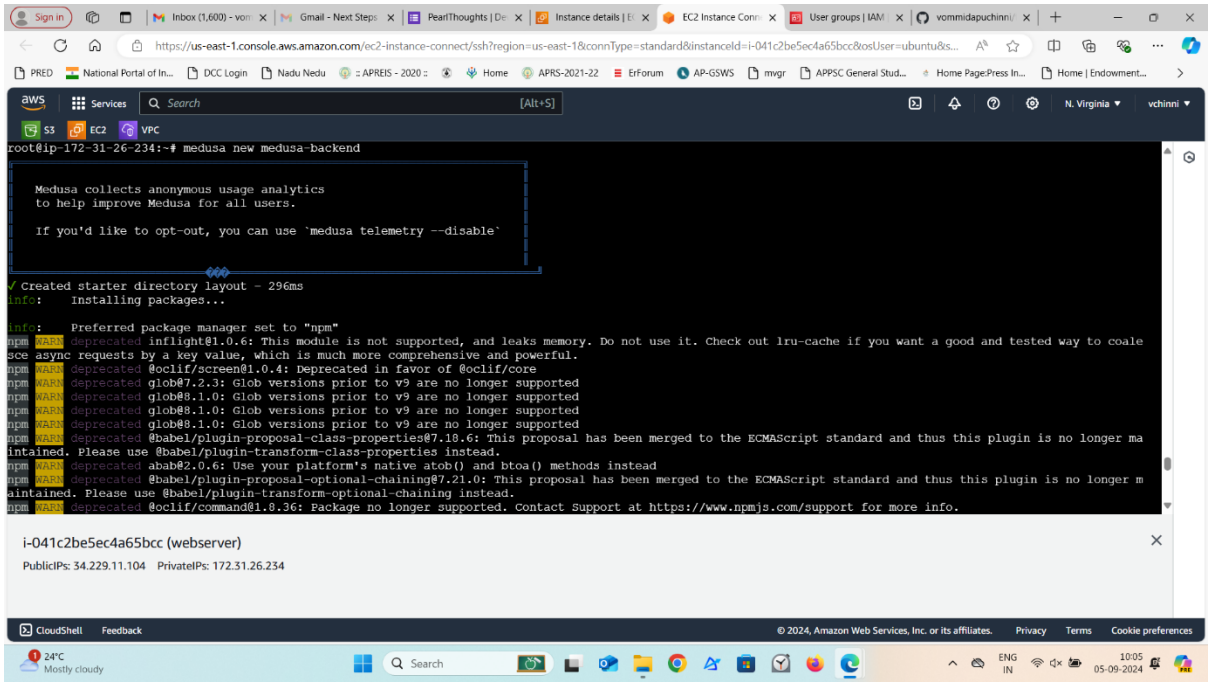
➢ Install Node.js and npm: Ensure Node.js and npm are installed. If not, install them using:
   - sudo apt update
   - sudo apt install nodejs npm



- Install Medusa CLI: Install the Medusa CLI globally:
  - npm install -g @medusajs/medusa-cli



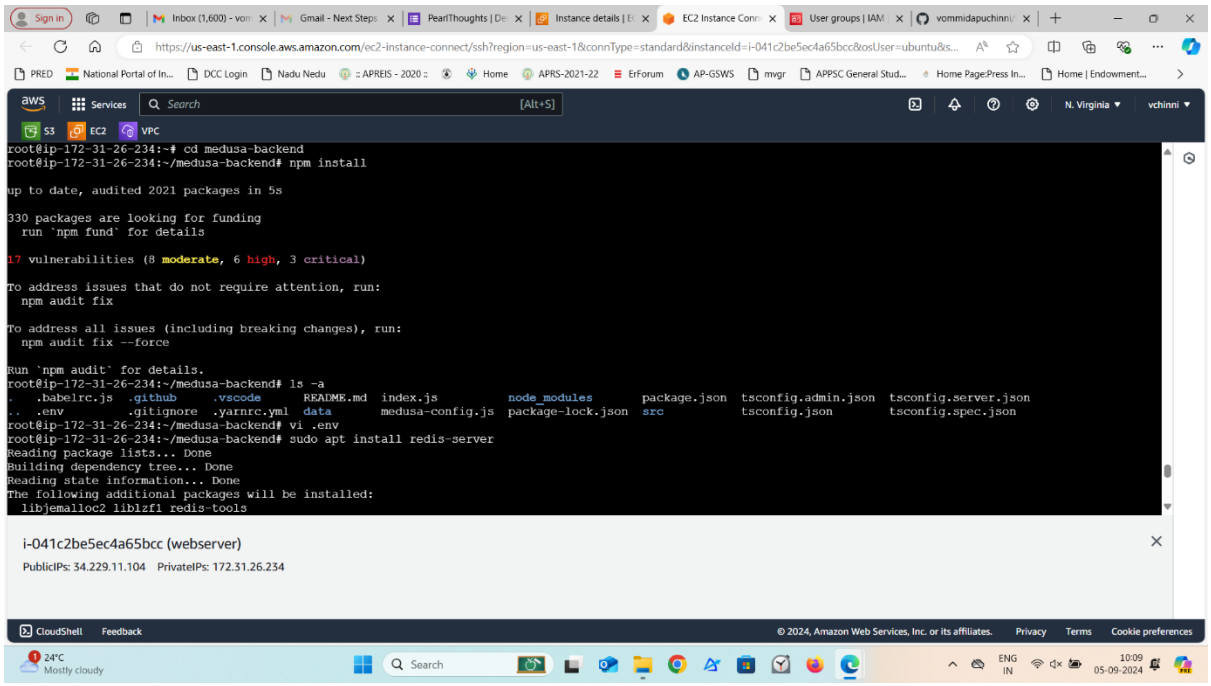
- Create a New Medusa Project: Create a new Medusa project:
  - medusa new medusa-backend



- cd medusa-backend

➢ Install Project Dependencies: Install the required dependencies for the project:

- npm install



5. Configure Medusa

Update .env File: Create or update the .env file in your project root with PostgreSQL credentials:

- DATABASE_TYPE=postgres
- DATABASE_URL=postgres://medusa_user:your_password@localhost:5432/medusa_db
- JWT_SECRET=your_jwt_secret
- COOKIE_SECRET=your_cookie_secret
- NODE_ENV=production
- NPM_CONFIG_PRODUCTION=false

➢ **Update medusa-config.js**: Ensure your medusa-config.js is configured to use the PostgreSQL database

Step 4: setup all thing

Basic .env File Content

Database Configuration

DATABASE_URL=postgres://medusa_user:your_password@localhost:5432/medusa_db

Redis Configuration (if using Redis)

REDIS_URL=redis://localhost:6379

Other Medusa Configuration (if needed)

Example: PORT=9000

PORT=9000

Ensure Redis is installed and running:

sudo apt update && sudo apt install redis-server && sudo systemctl start redis-server

Connect to PostgreSQL as a Superuser

First, connect to your PostgreSQL database as a superuser (like postgres):

sudo -u postgres psql

2. Grant Necessary Permissions

Once you're in the PostgreSQL shell, run the following commands to grant the medusa_user the required permissions on the public schema:

GRANT ALL PRIVILEGES ON SCHEMA public TO medusa_user;

ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT ALL ON TABLES TO medusa_user;

3. Verify Permissions

You can verify that the user now has the correct permissions by running:

\dn+

This will list the schemas and the privileges granted to each role.

4. Exit the PostgreSQL Shell

Type \q to exit the PostgreSQL shell.

Ensure that the redis_url property in the projectConfig is properly set. And update package.json and medusa-config.js

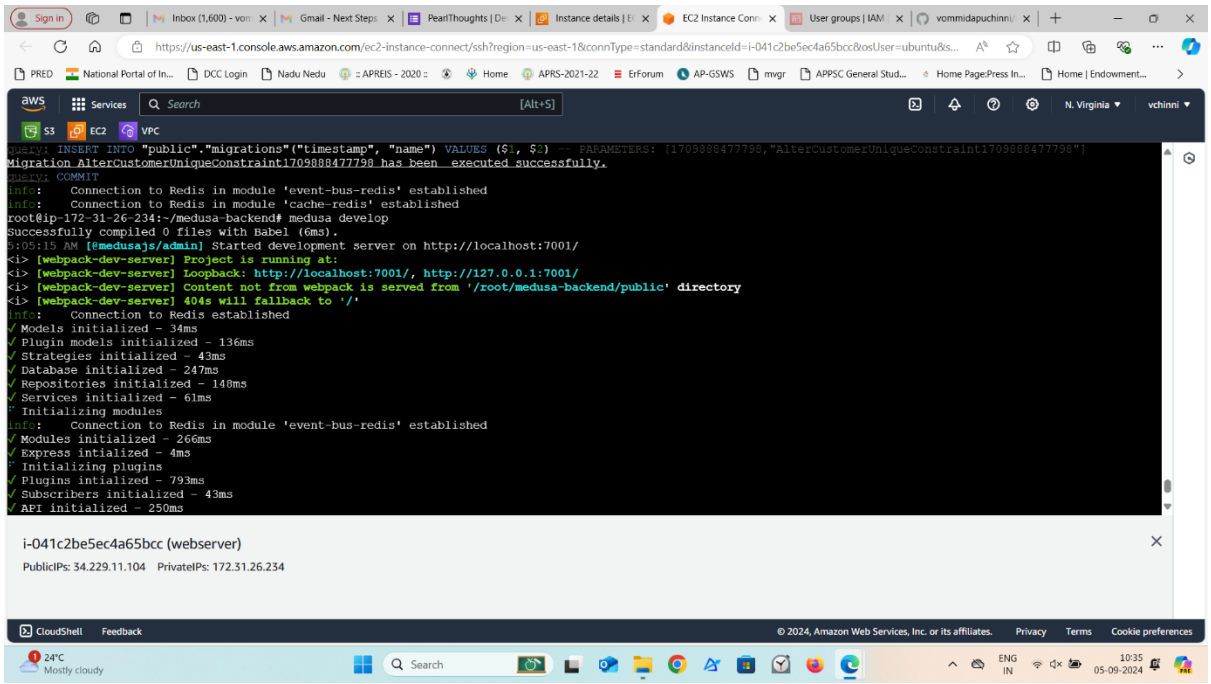
Run Migrations: If you haven't done so already, run the migrations to ensure the database schema is up to date:

medusa migrations run

Start the Medusa Backend

Ensure the Medusa backend is running. Depending on how you are managing services, you might start it with a command like:

medusa develop



Rebuild the Project: Make sure to rebuild the project to include any changes made to the package.json scripts and medusa-config.js file.

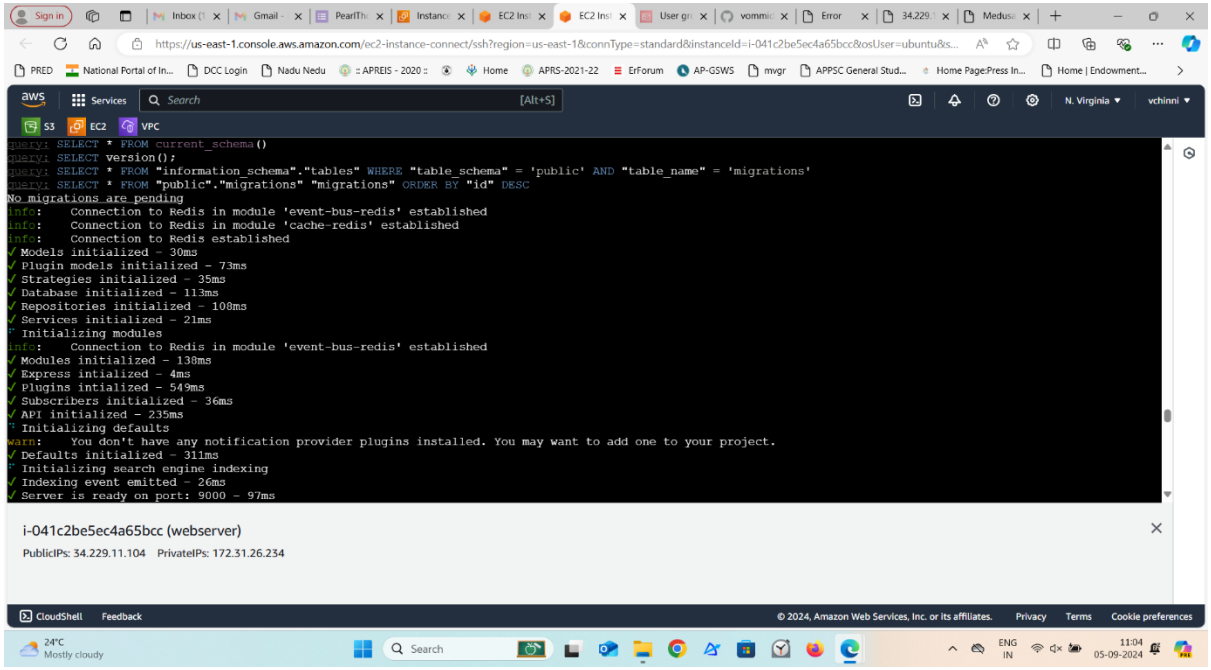
npm run build

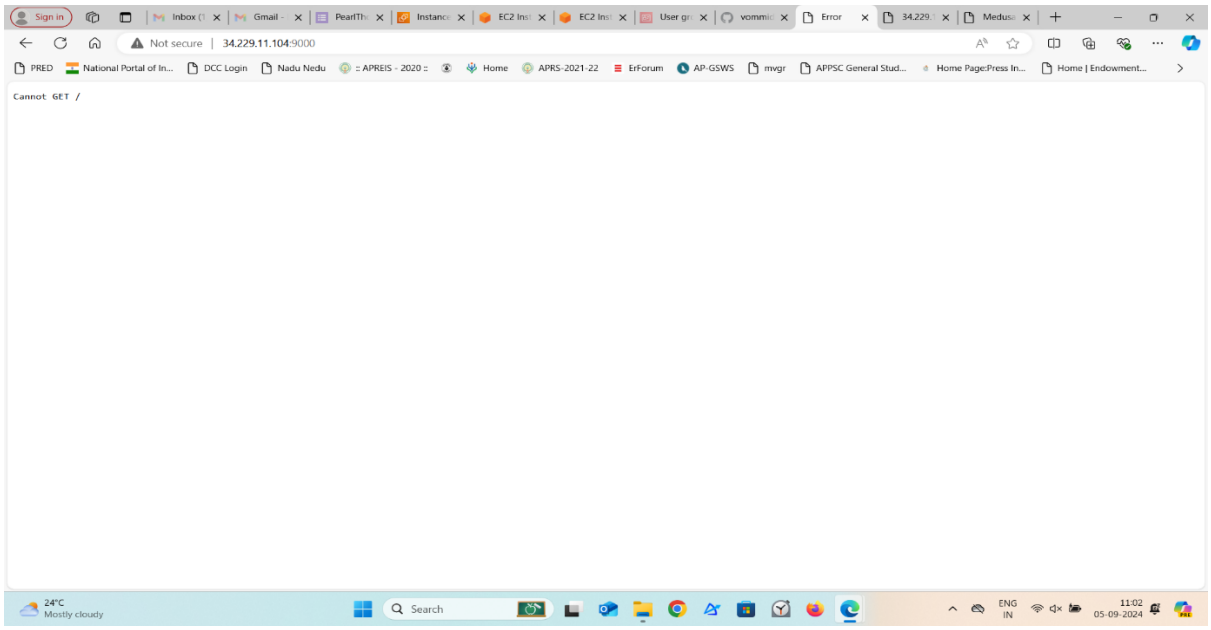
Run Migrations: Apply any pending database migrations. This ensures that your database schema is up to date.

npx medusa migrations run

Start the Backend: Start your Medusa backend. This will use the updated configuration and ensure that the server is running with the latest changes.

npm start

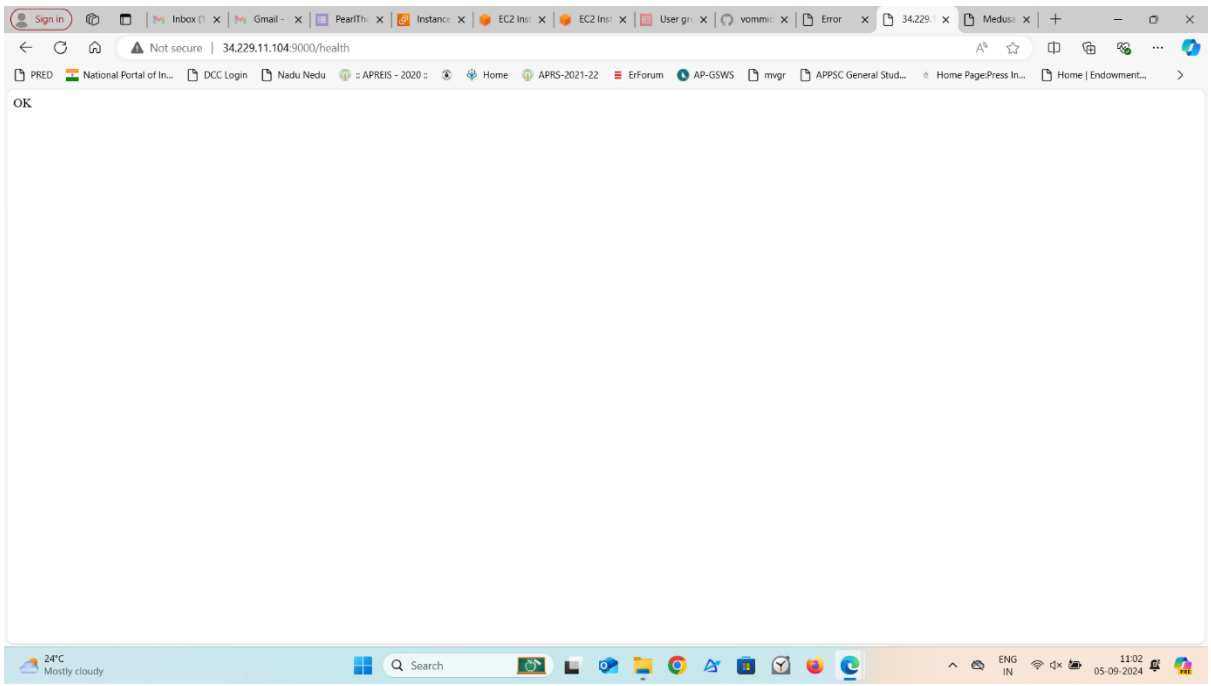




Verify the Deployment: Check if the backend is running properly by accessing the health check endpoint.

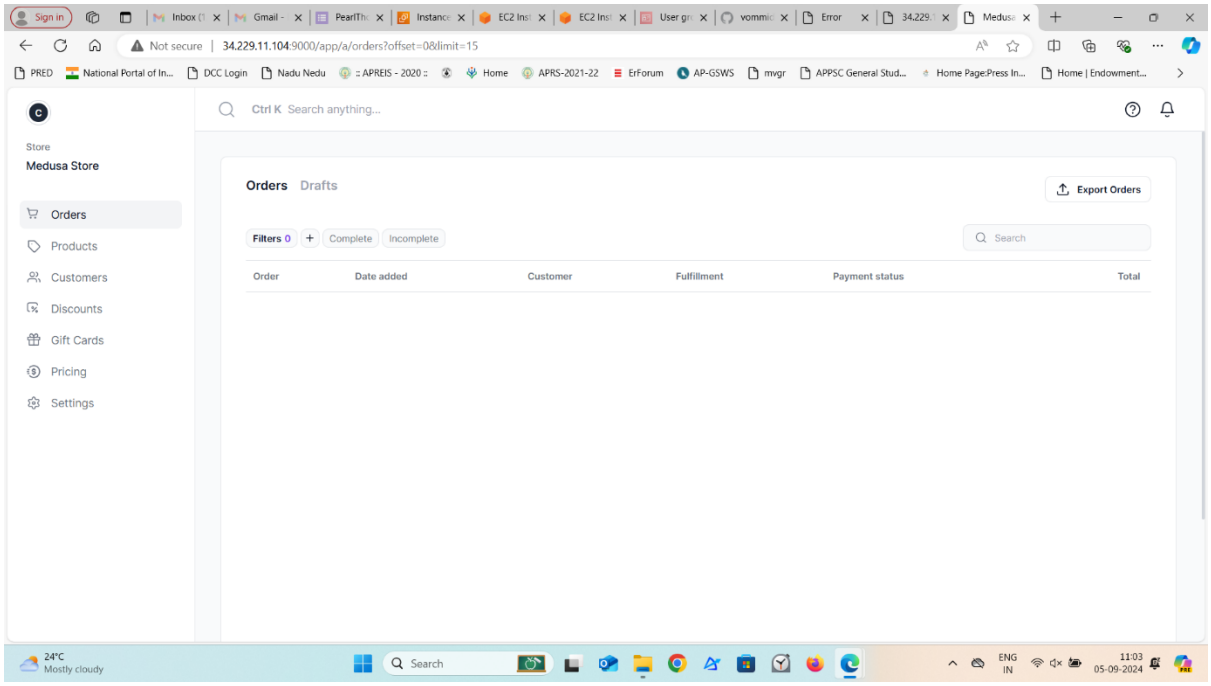
curl http://localhost:9000/health

Replace localhost:9000 with the appropriate URL if your backend is running on a different host or port.



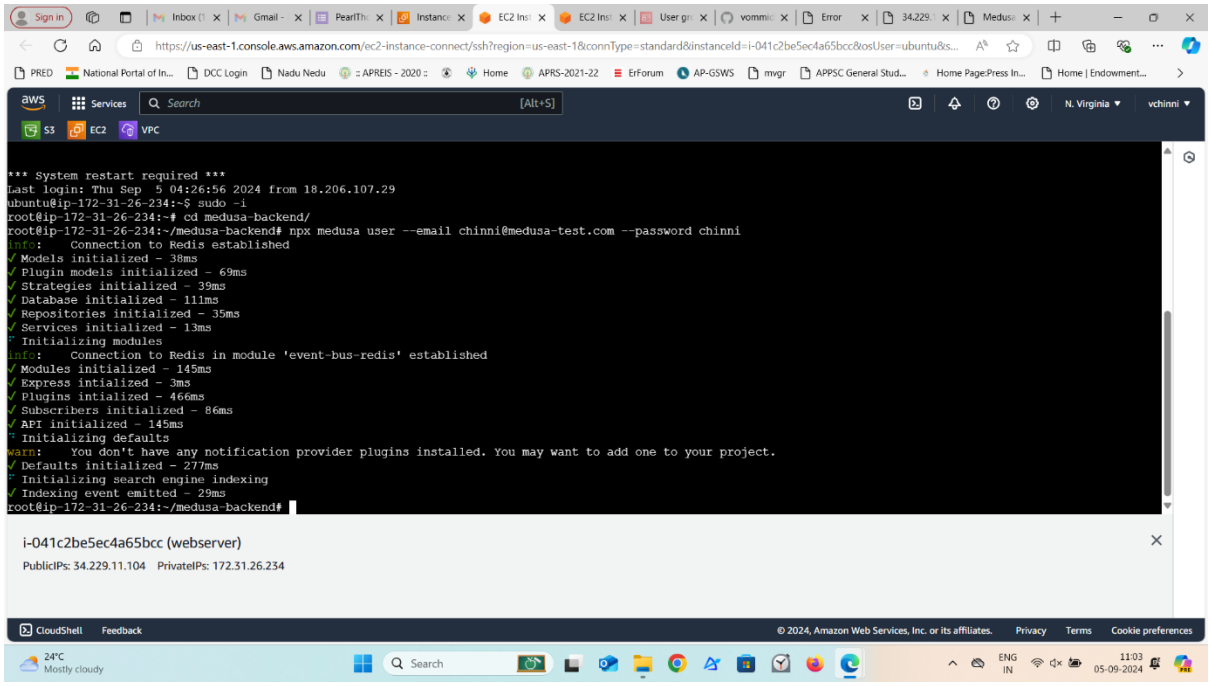
Check the Admin Dashboard (if deployed): If you deployed the admin dashboard with the backend, check if it's available at:

http://localhost:9000/app

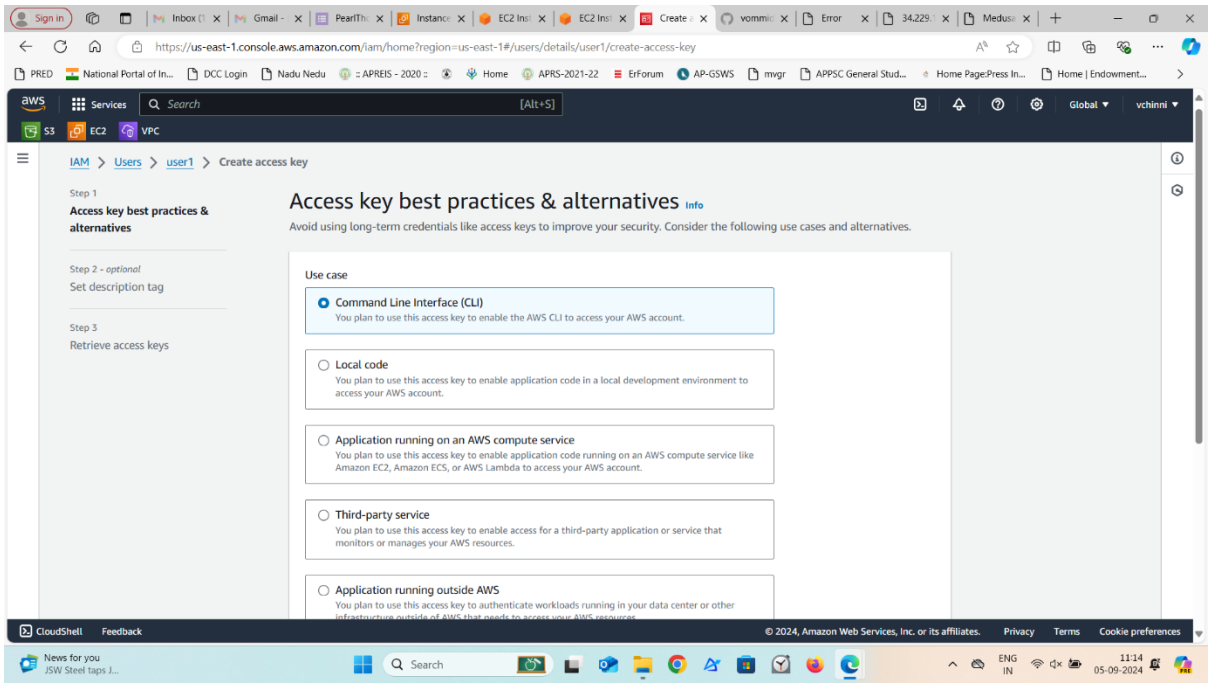


Create Admin User (if needed): If you need to create an admin user, you can do so by running:

npx medusa user --email admin@medusa-test.com --password supersecret



Create user and from that create access keys



Step 5: Install needed packages

1. Install Docker: Docker is essential for building and running containerized applications, including the Medusa backend.

    1. Install Docker: sudo apt update && sudo apt install -y docker.io
    2. Start Docker and Enable on Boot: sudo systemctl start docker && sudo systemctl enable docker

3. Add Your User to Docker Group (Optional): sudo usermod -aG docker $USER
4. Verify Docker Installation: docker --version

2. Install Docker Compose

Docker Compose is used to define and run multi-container Docker applications.

1. Download and Install Docker Compose: sudo apt install -y unzip

   curl -L "https://github.com/docker/compose/releases/download/$(curl -s https://api.github.com/repos/docker/compose/releases/latest | jq -r .tag_name)/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
2. Set Permissions: sudo chmod +x /usr/local/bin/docker-compose
3. Verify Docker Compose Installation: docker-compose --version
4. Install AWS CLI: The AWS CLI is already configured with your credentials.
5. Install Terraform:
   Terraform is used for managing infrastructure as code.
   Download and Install Terraform: sudo apt install -y unzip

   curl -LO "https://releases.hashicorp.com/terraform/$(curl -s https://checkpoint-api.hashicorp.com/v1/check/terraform | jq -r .current_version)/terraform_$(curl -s https://checkpoint-api.hashicorp.com/v1/check/terraform | jq -r .current_version)_linux_amd64.zip"

sudo unzip terraform_*.zip -d /usr/local/bin/

Verify Terraform Installation: terraform --version

You should create your Dockerfile in the root directory of your project. This is typically where your main application files and configuration are located, and it makes it easier to manage and build your Docker image.

Step 6: Steps to Create and Use the Dockerfile

1. Create the Dockerfile:
   - Navigate to the root of your project directory.
   - Create a file named Dockerfile using a text editor.

2. Add the content to the file.

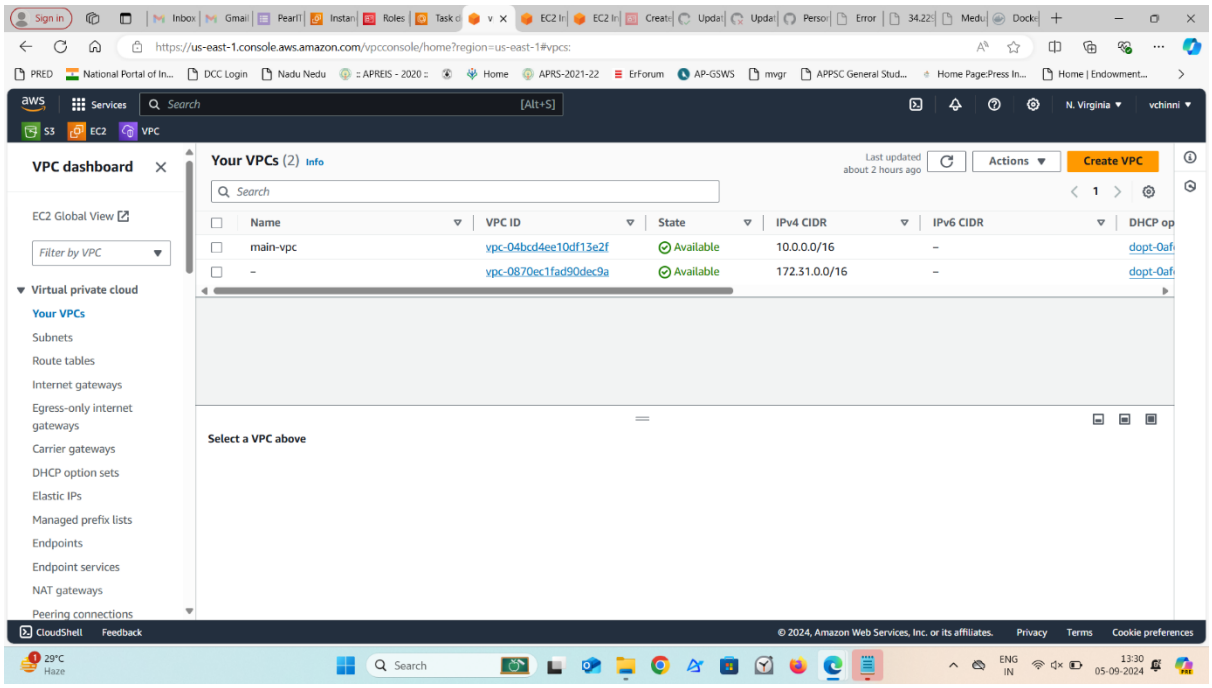
Step 7: Design the IaC (Terraform, Aws ECS) for Medusa open source

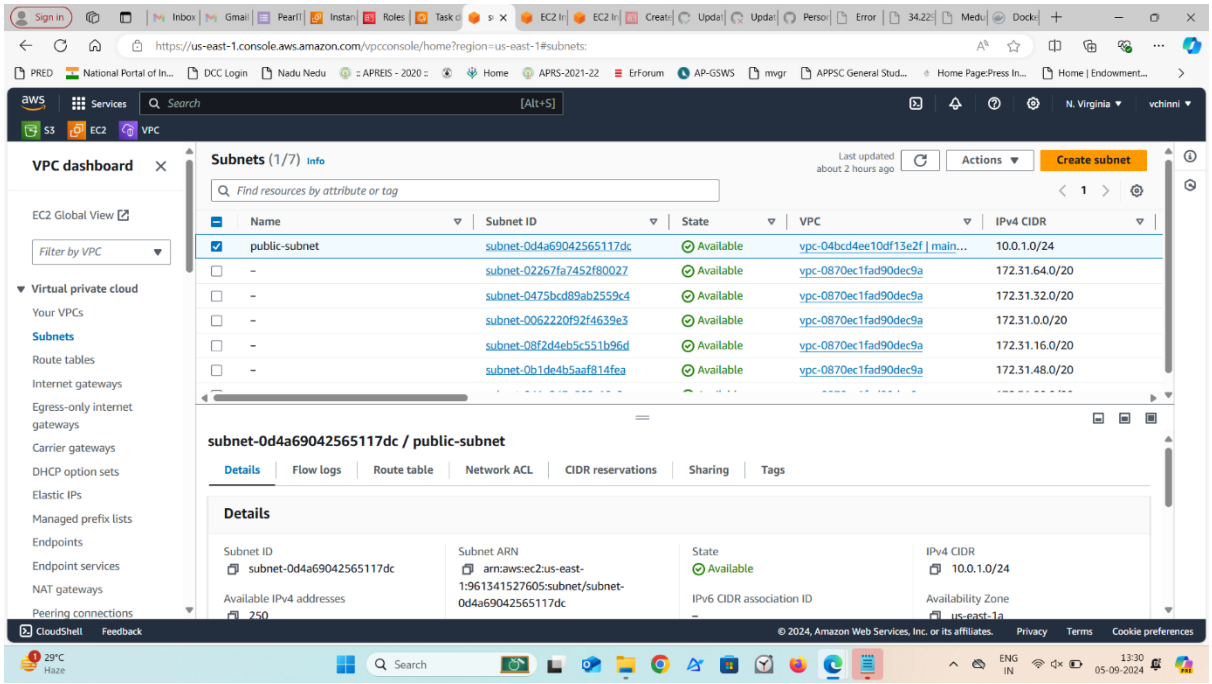
1. provider.tf

This file sets up the AWS provider.

2. networking.tf

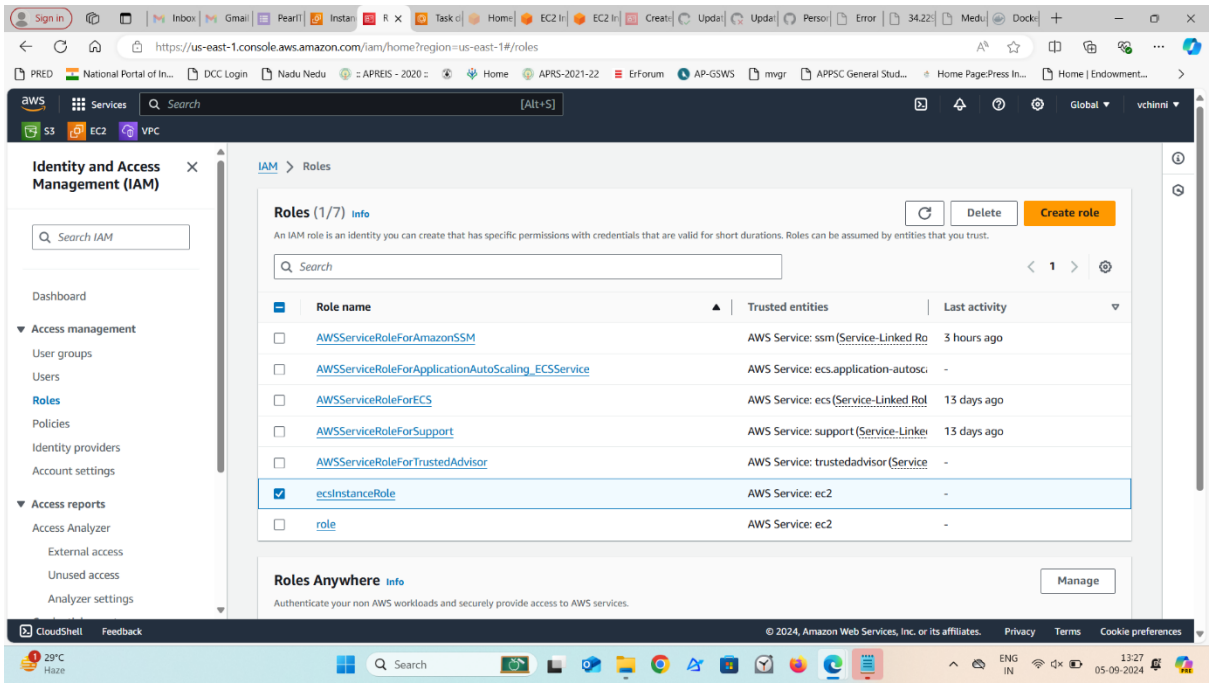
This file sets up the VPC, subnet, and security group.





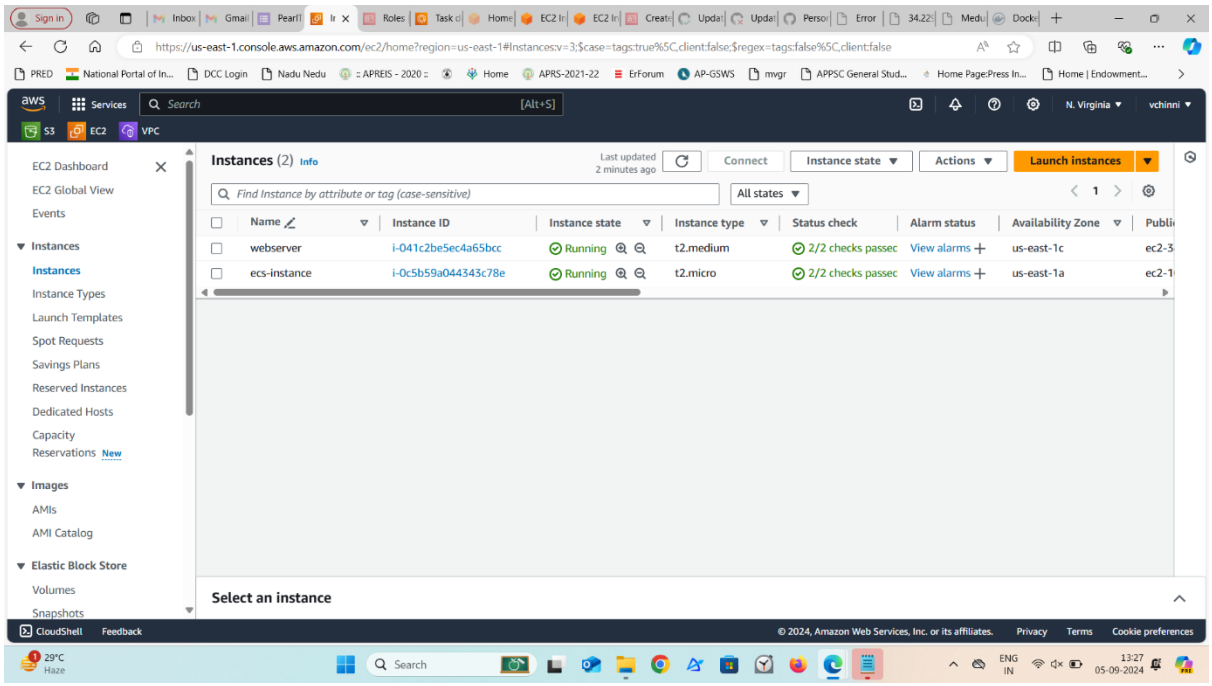
3. iam.tf

This file sets up the IAM roles and instance profile.



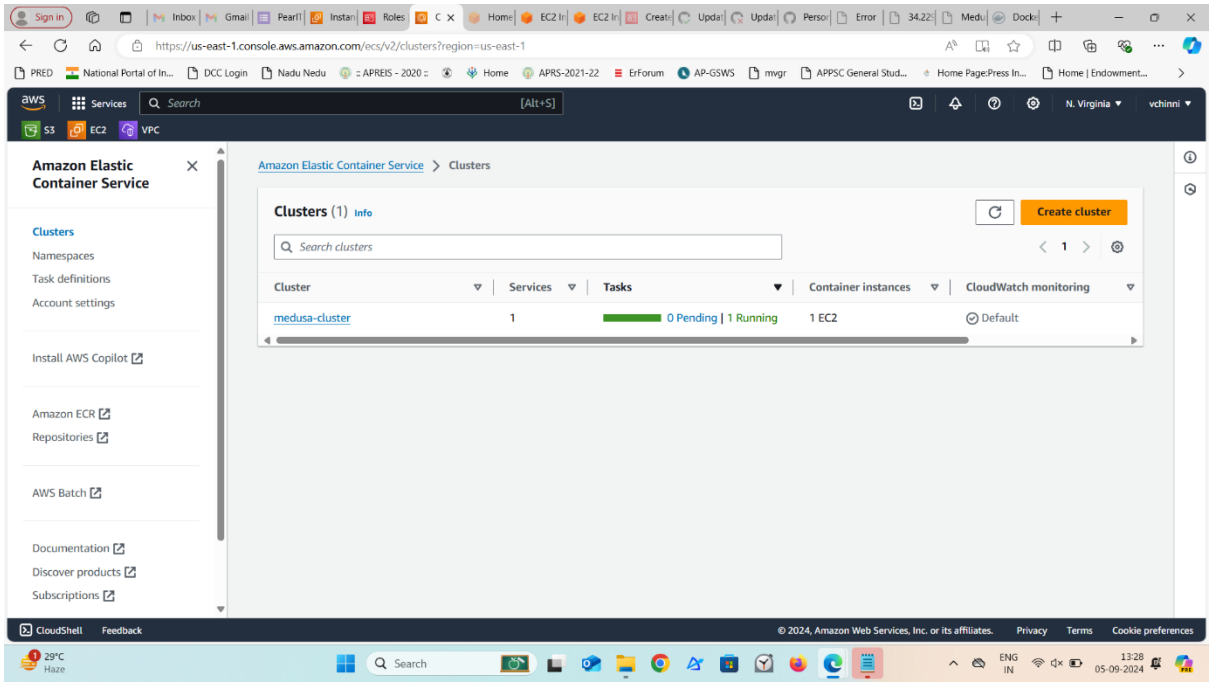
4. ec2.tf

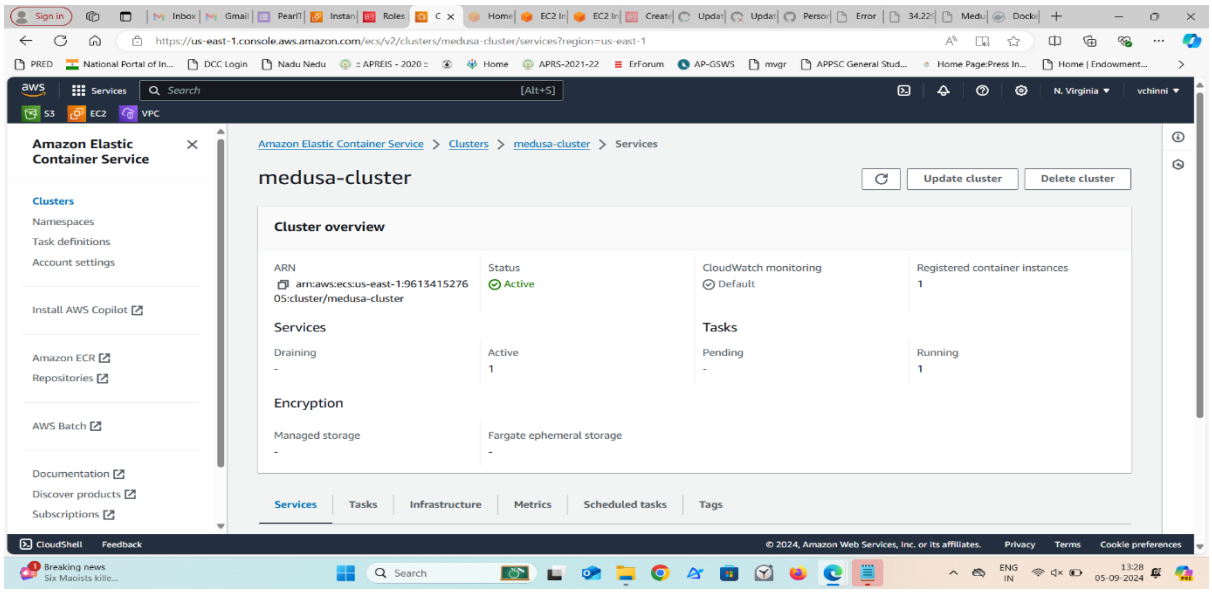
This file sets up the EC2 instance.

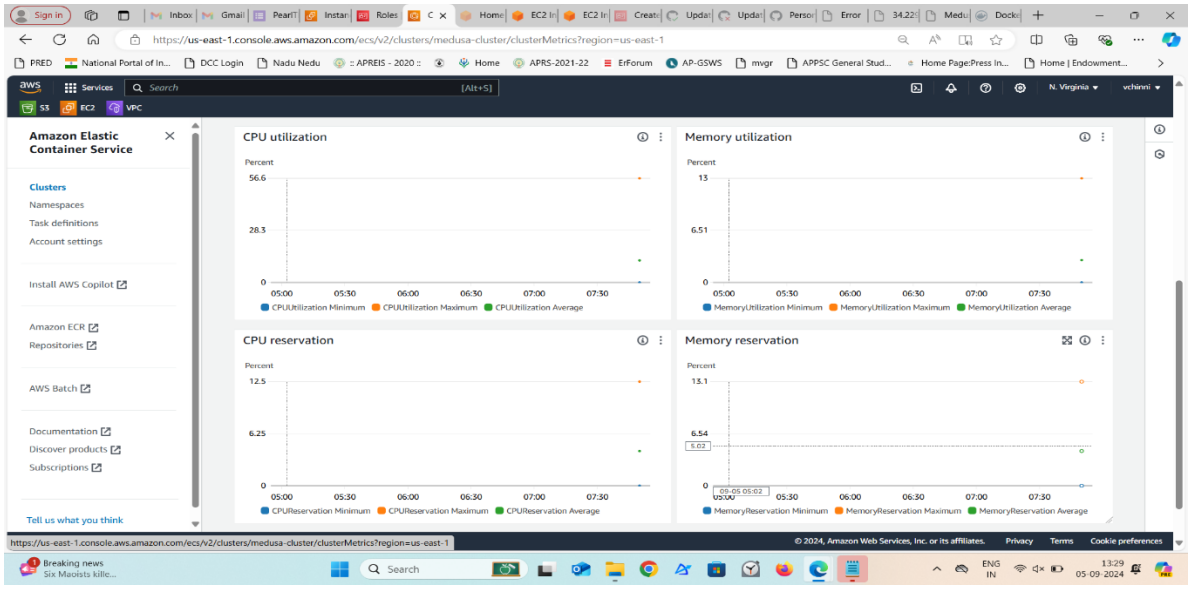


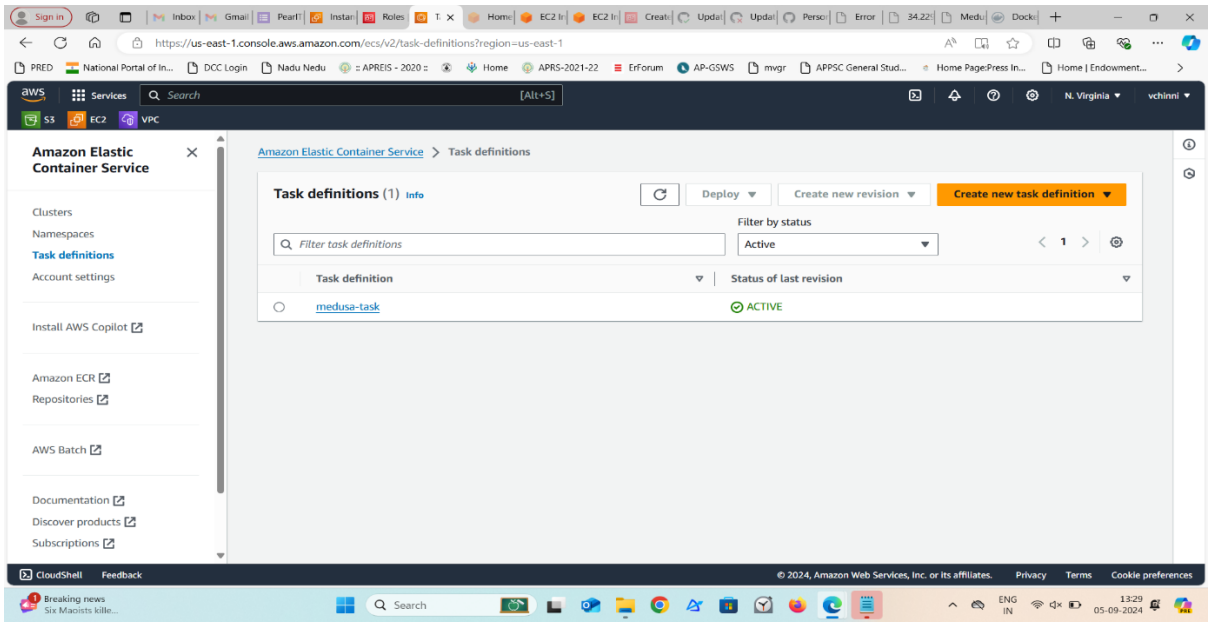
5. ecs.tf

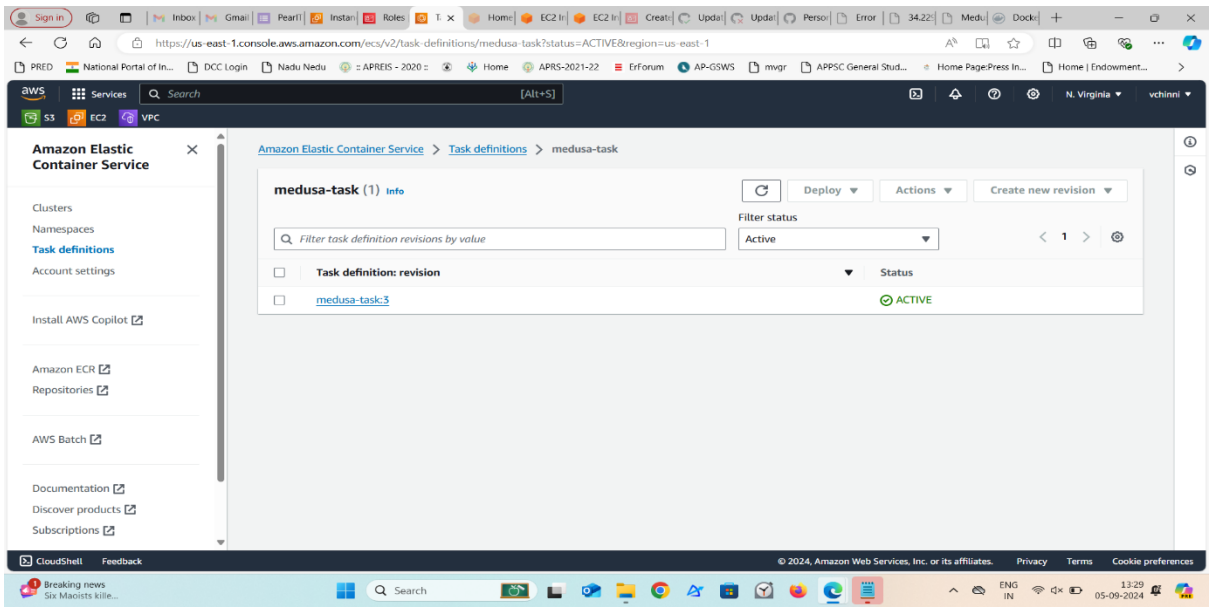
This file sets up the ECS cluster, task definition, and service.

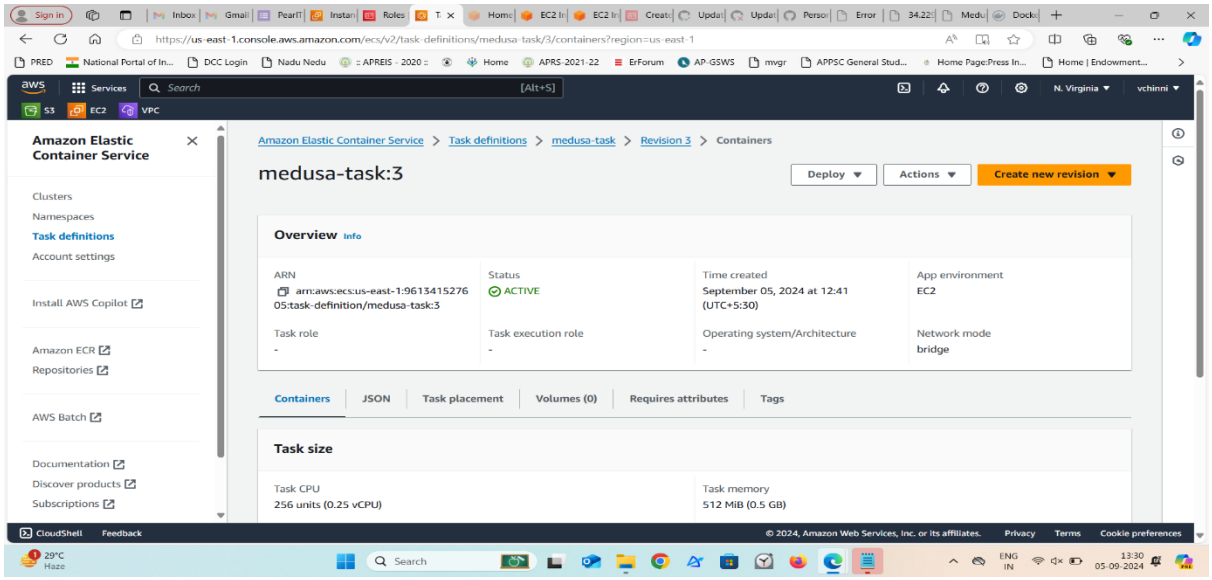


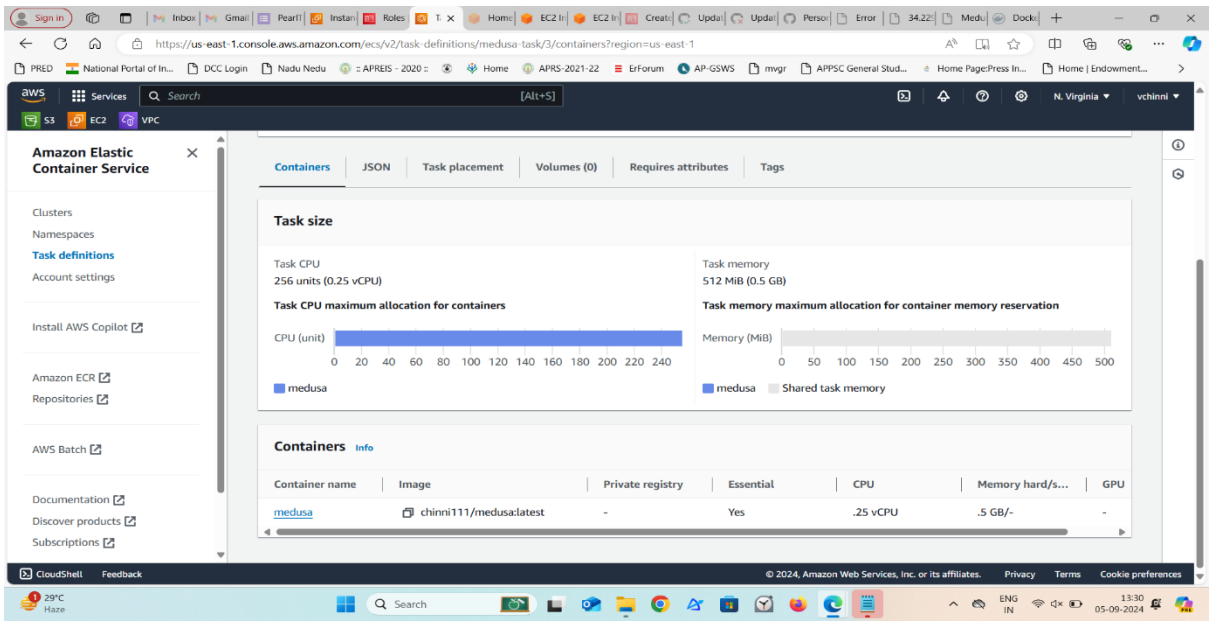



Amazon Elastic Container Service
Clusters
Namespaces
Task definitions
Account settings
Install AWS Copilot
Amazon ECR
Repositories
AWS Batch
Documentation
Discover products
Subscriptions
Tell us what you think
CPU utilization
Memory utilization
CPU reservation
Memory reservation
Percent
CPUUtilization Minimum
CPUUtilization Maximum
CPUUtilization Average
MemoryUtilization Minimum
MemoryUtilization Maximum
MemoryUtilization Average
CPUReservation Minimum
CPUReservation Maximum
CPUReservation Average
MemoryReservation Minimum
MemoryReservation Maximum
MemoryReservation Average


Amazon Elastic Container Service
Clusters
Namespaces
Task definitions
Account settings
Install AWS Copilot
Amazon ECR
Repositories
AWS Batch
Documentation
Discover products
Subscriptions
Amazon Elastic Container Service > Task definitions
Task definitions (1) Info
Deploy
Create new revision
Create new task definition
Filter by status
Filter task definitions
Active
Task definition
Status of last revision
medusa-task
ACTIVE
CloudShell
Feedback

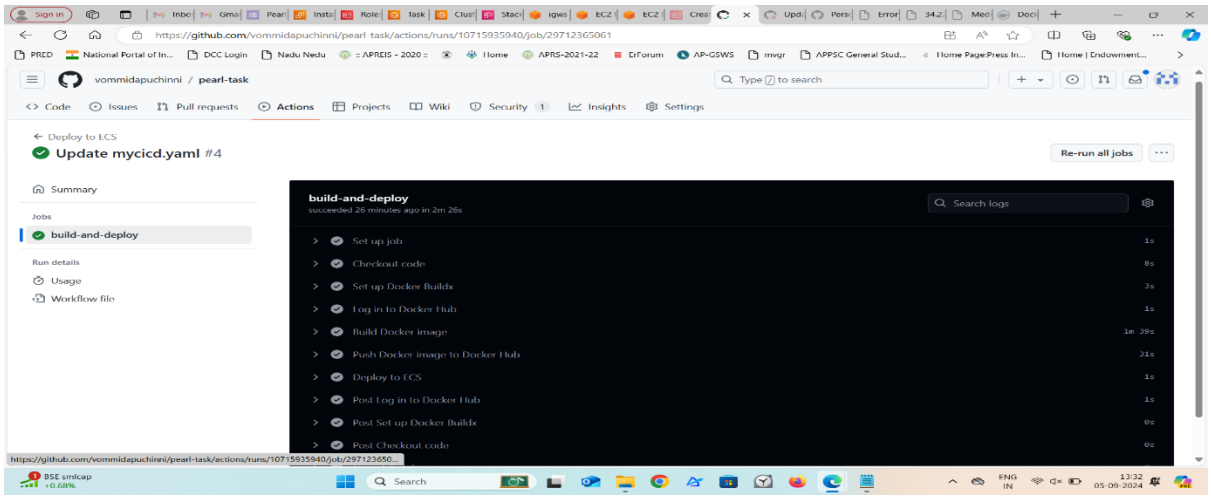

Amazon Elastic Container Service
Clusters
Namespaces
Task definitions
Account settings
Install AWS Copilot
Amazon ECR
Repositories
AWS Batch
Documentation
Discover products
Subscriptions
Amazon Elastic Container Service > Task definitions > medusa-task
medusa-task (1) Info
Deploy
Actions
Create new revision
Filter status
Filter task definition revisions by value
Active
Task definition: revision
Status
medusa-task:3
ACTIVE
CloudShell
Feedback





6. Configure GitHub Actions for CD

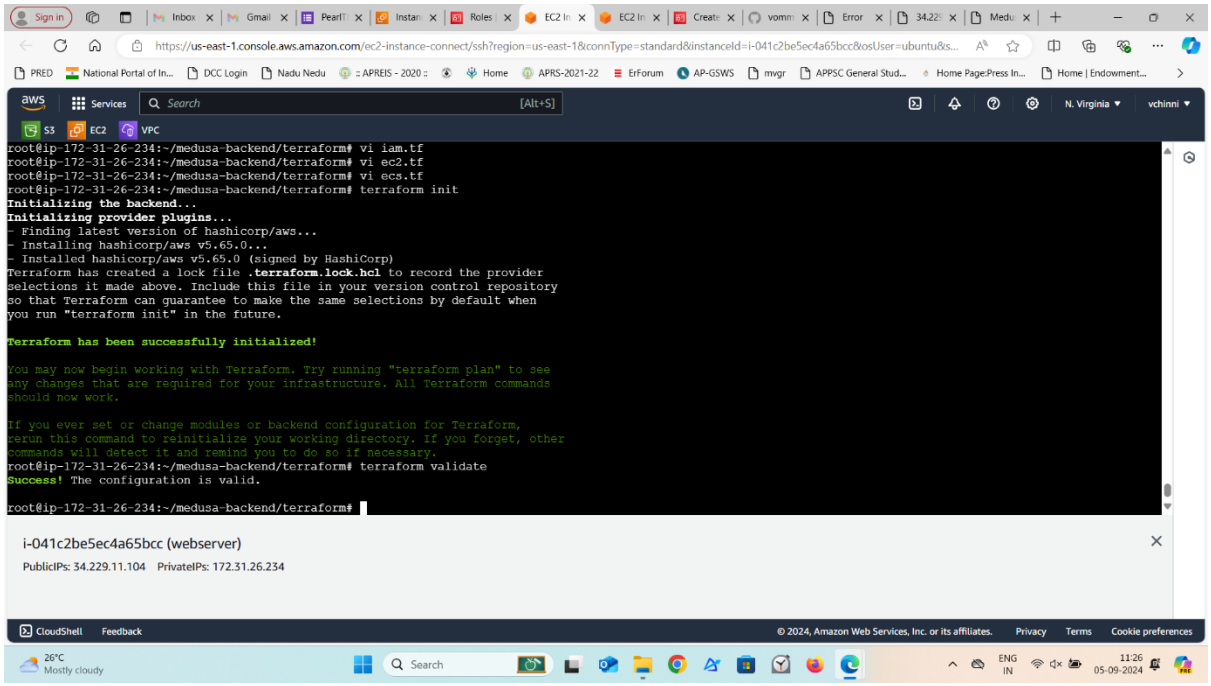
Create a GitHub Actions workflow in .github/workflows/deploy.yml

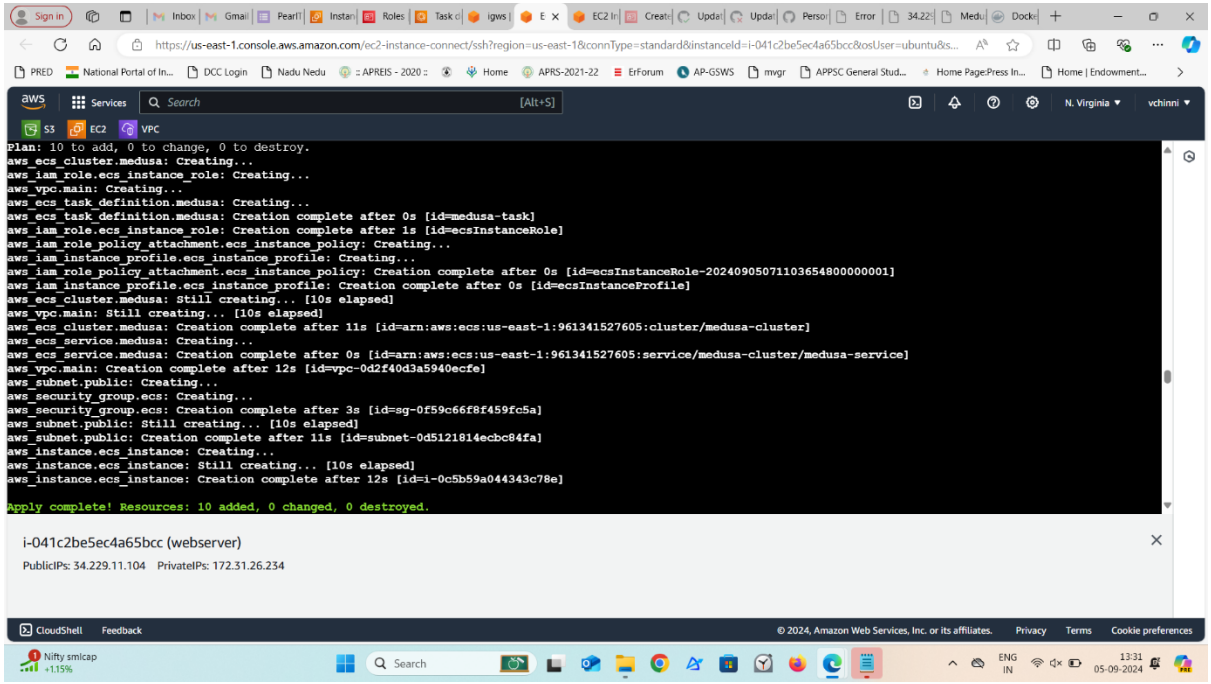


Deploy the Configuration

Initialize and apply the Terraform configuration:

- terraform init
- terraform validate
- terraform apply –auto-approve

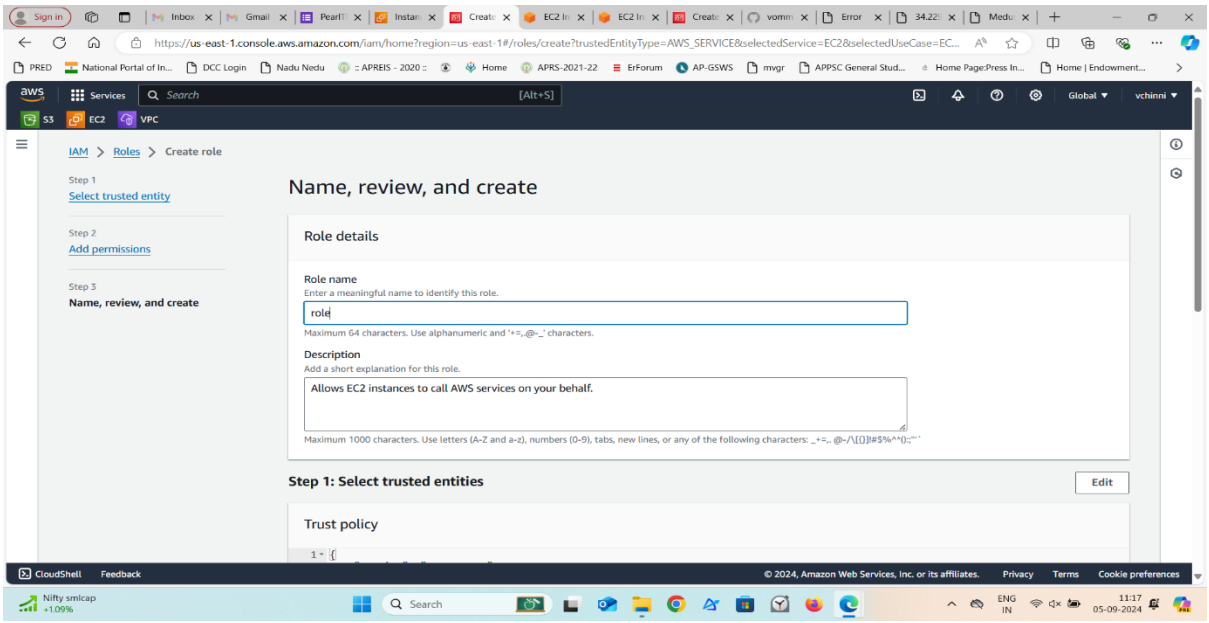


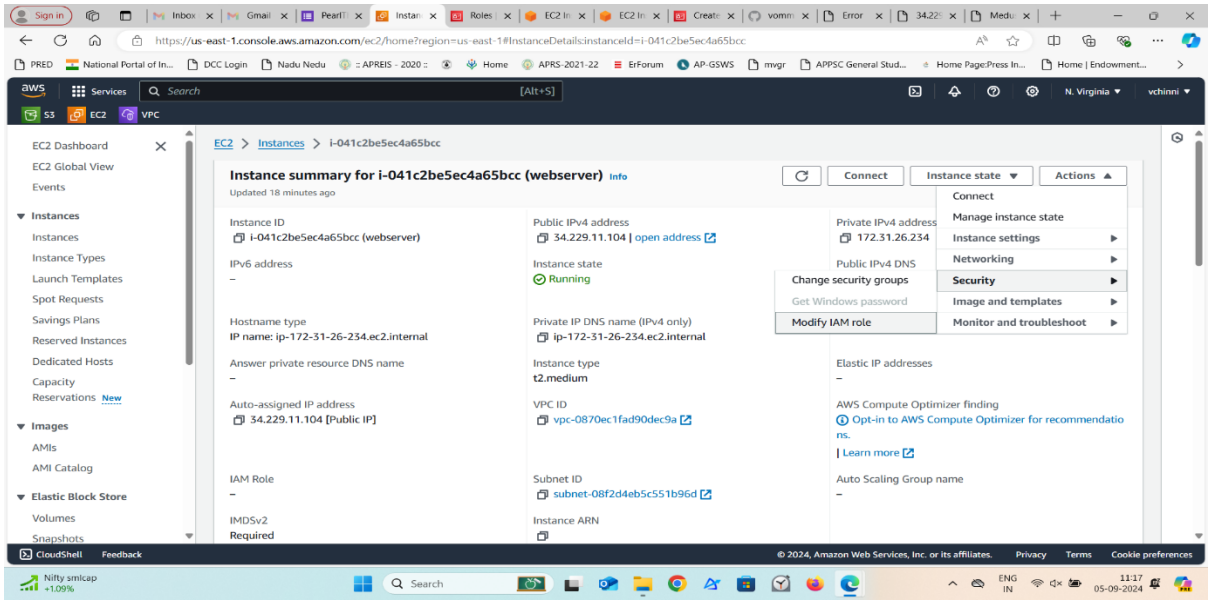


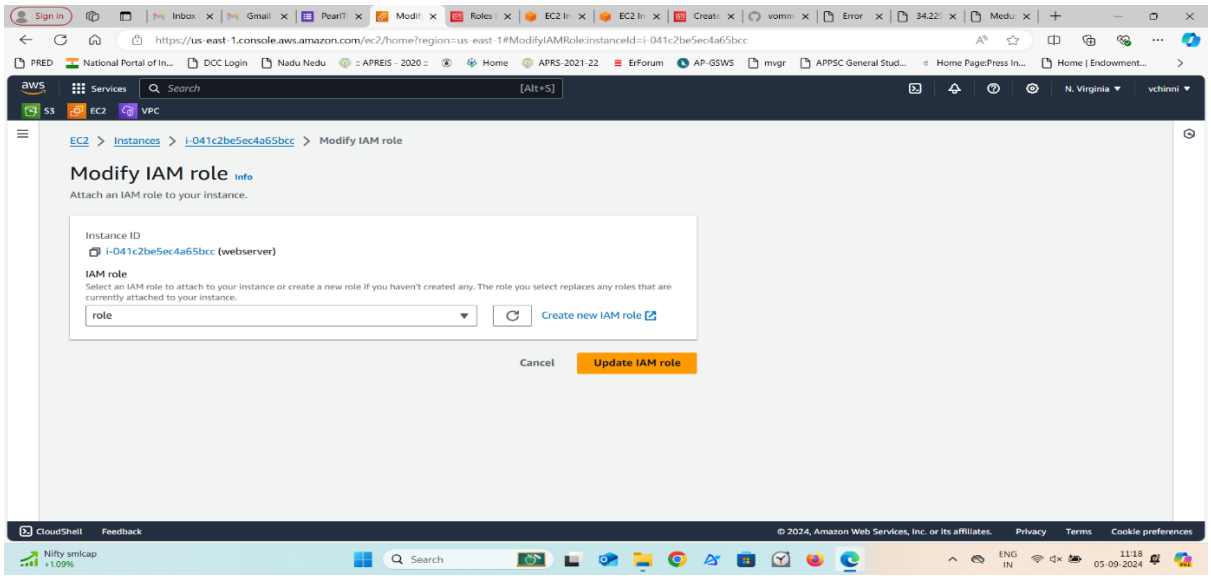
Step 8: Install the ECS Agent

To install the ECS agent on your EC2 instance, follow these steps:

1. Update the Package List: sudo apt-get update
2. Install Docker (if not already installed): sudo apt-get install -y docker.io
3. Start Docker Service: sudo systemctl start docker
4. Install the ECS Agent: sudo apt-get install -y ecs-init
5. Start and Enable the ECS Agent Service: sudo systemctl enable ecs && sudo systemctl start ecs
6. Verify ECS Agent Installation. After installing the ECS agent, verify that it's running: sudo systemctl status ecs
7. Verify ECS Configuration on EC2 Instance: Confirm that the ECS agent is correctly configured on your EC2 instance. The configuration file /etc/ecs/ecs.config should have the correct ECS cluster name: cat /etc/ecs/ecs.config
8. Ensure it contains: ECS_CLUSTER=medusa-cluster
9. Confirm EC2 Instance Role and Permissions: Ensure the IAM role attached to the EC2 instance has the required permissions: IAM Role: Should include the **AmazonEC2ContainerServiceforEC2Role** policy.







10. Policies: Check that the role has appropriate permissions to interact with ECS and other AWS services.

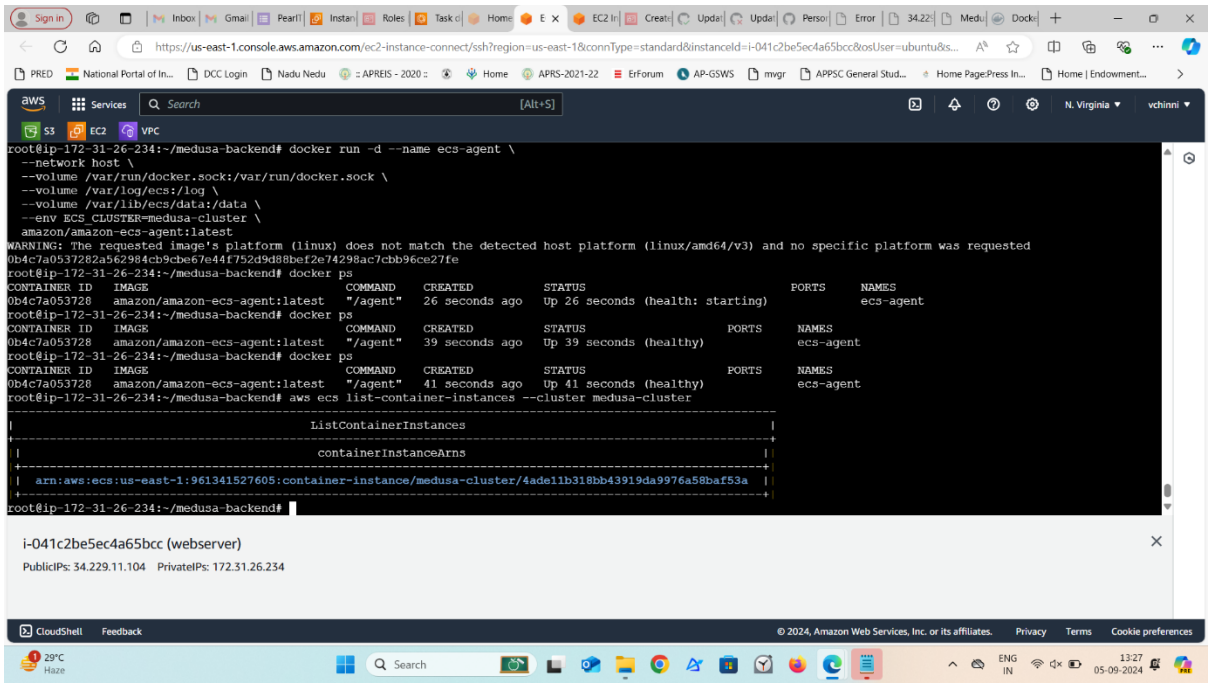
11. Restart ECS Agent**:** Restart the ECS agent after verifying configurations: docker restart ecs-agent
12. Inspect Container Instance Status**:** After ensuring configurations and permissions, check if the instance appears in the ECS console or via CLI: aws ecs list-container-instances --cluster medusa-cluster

Run the ECS agent container with the updated command:

docker run -d --name ecs-agent \ --network host \ --volume /var/run/docker.sock:/var/run/docker.sock \ --volume /var/log/ecs:/log \ --

volume /var/lib/ecs/data:/data \ --env ECS_CLUSTER=medusa-cluster \
amazon/amazon-ecs-agent:latest

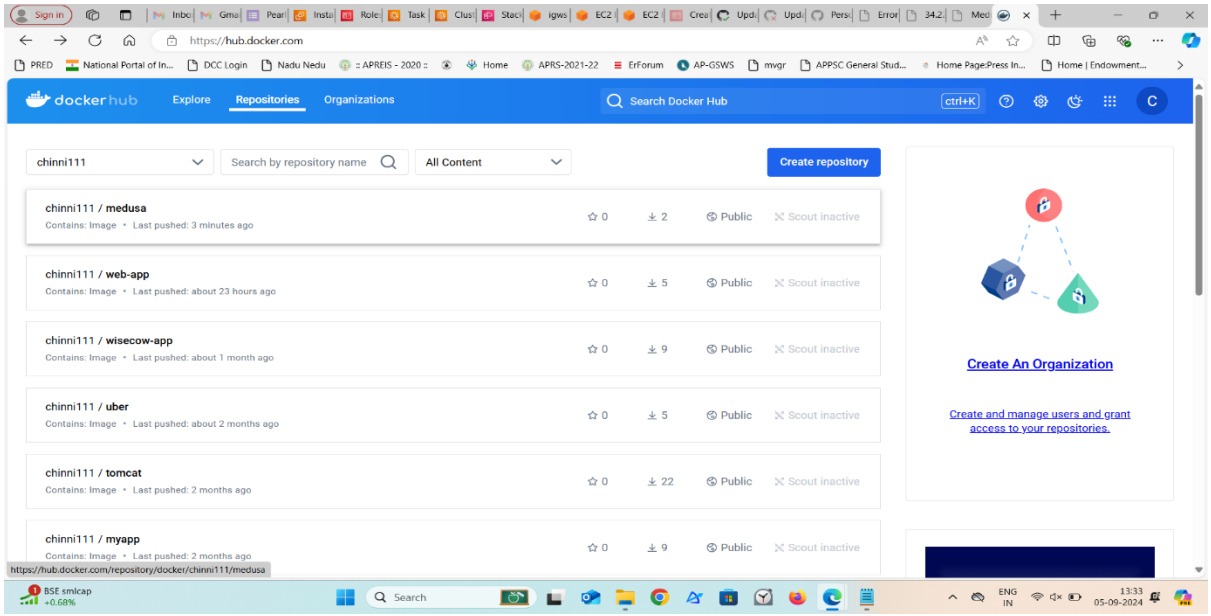
Check Cluster Registration: Verify if the ECS instance is now registered with the cluster: aws ecs list-container-instances --cluster medusa-cluster



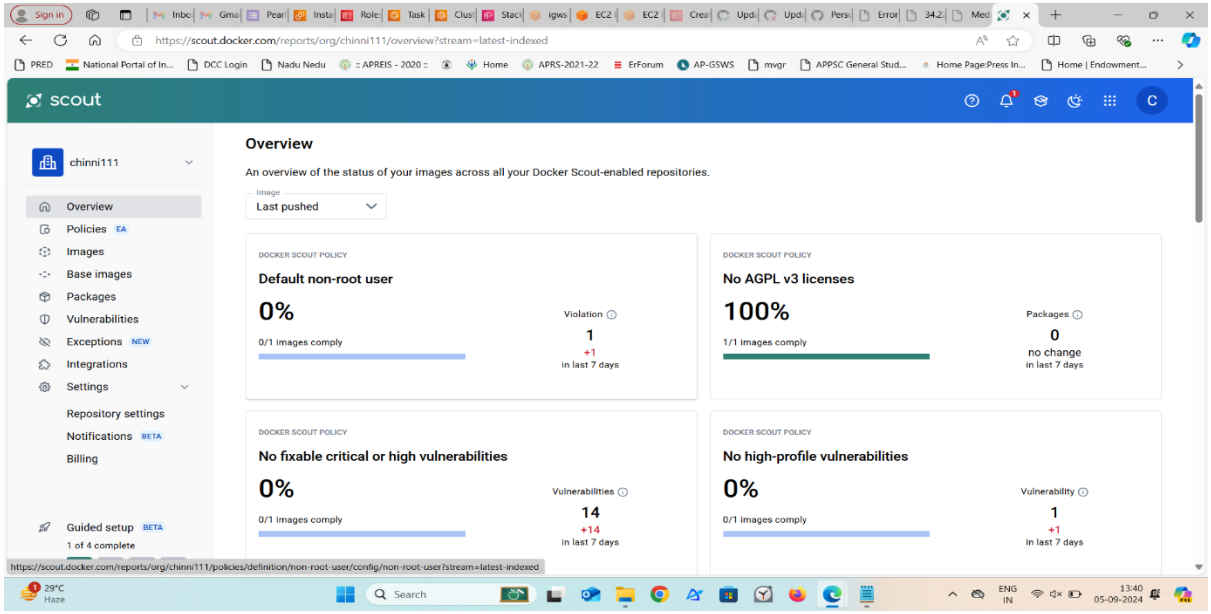
Verify Container Status: Check the status of the ECS agent container: docker ps

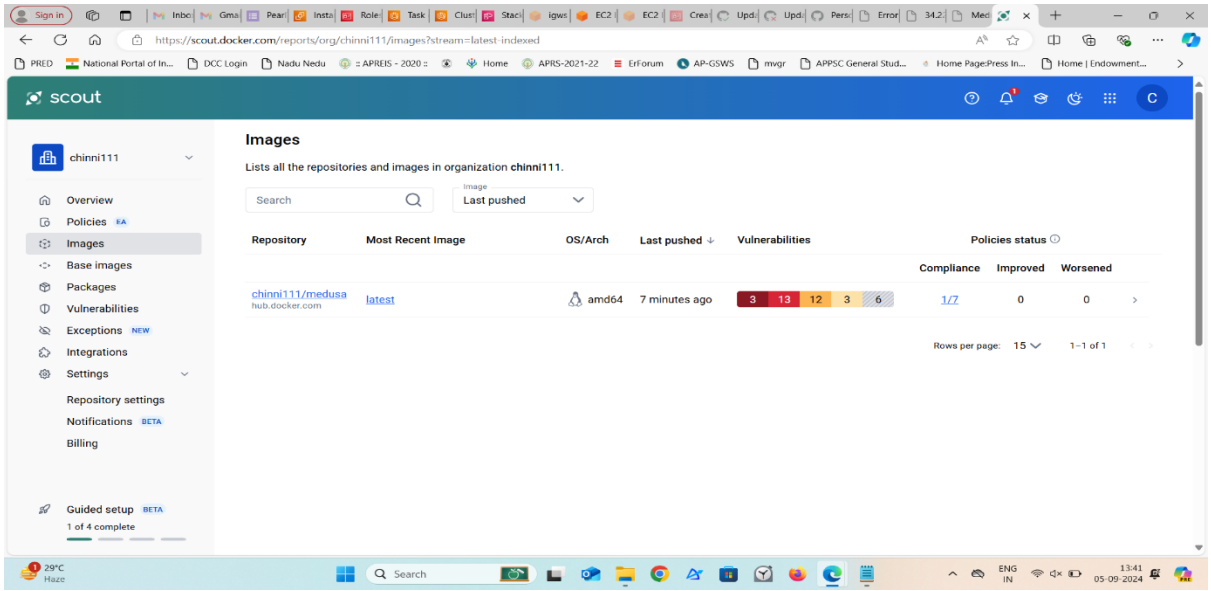
We see our image is deployed to ECS service.

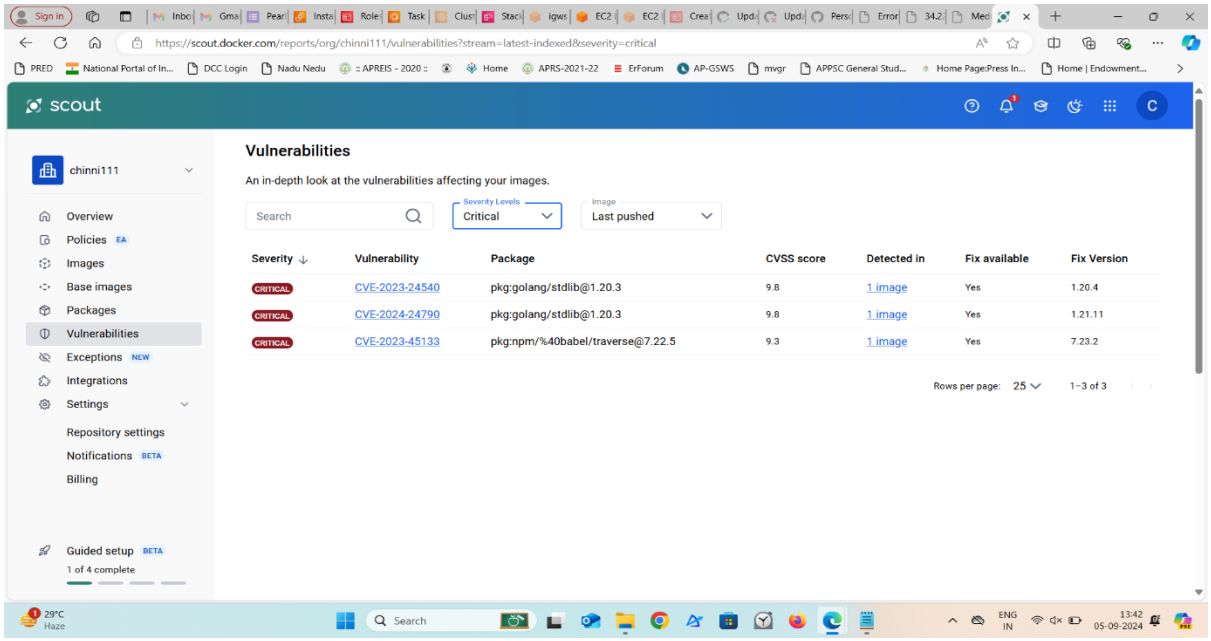
Our image is pushed to docker hub

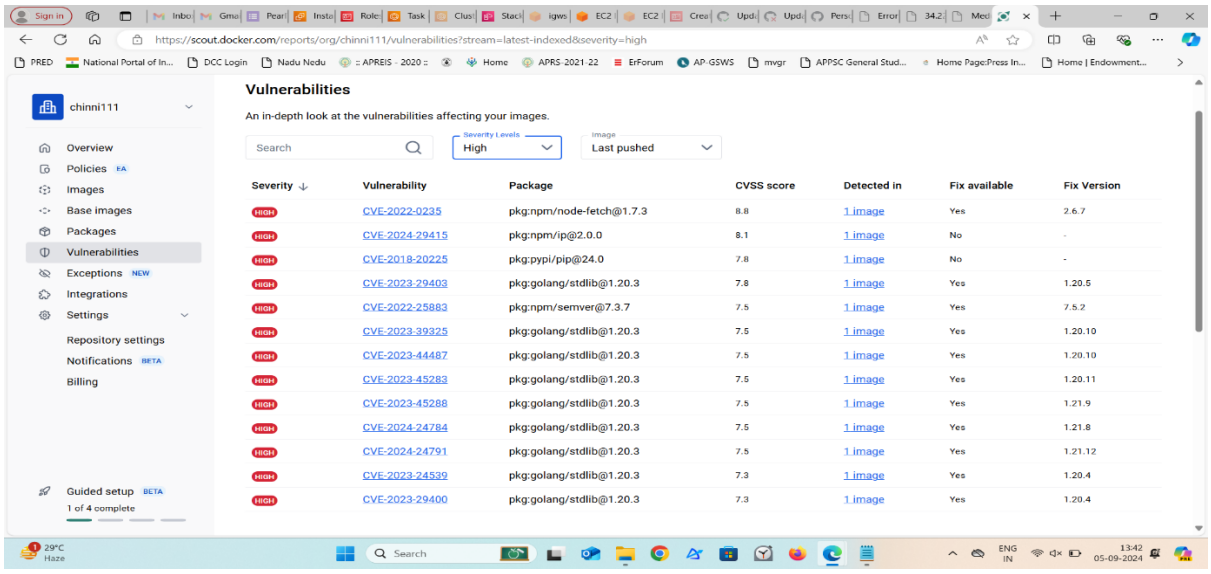


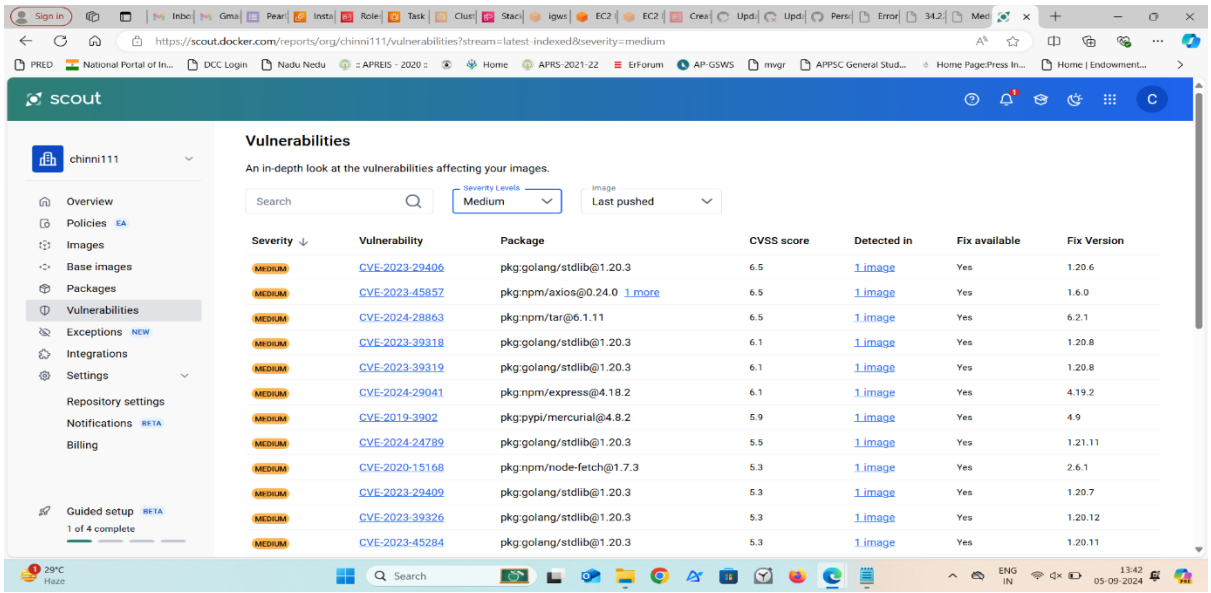
Docker Scout is a tool for scanning container images for vulnerabilities.

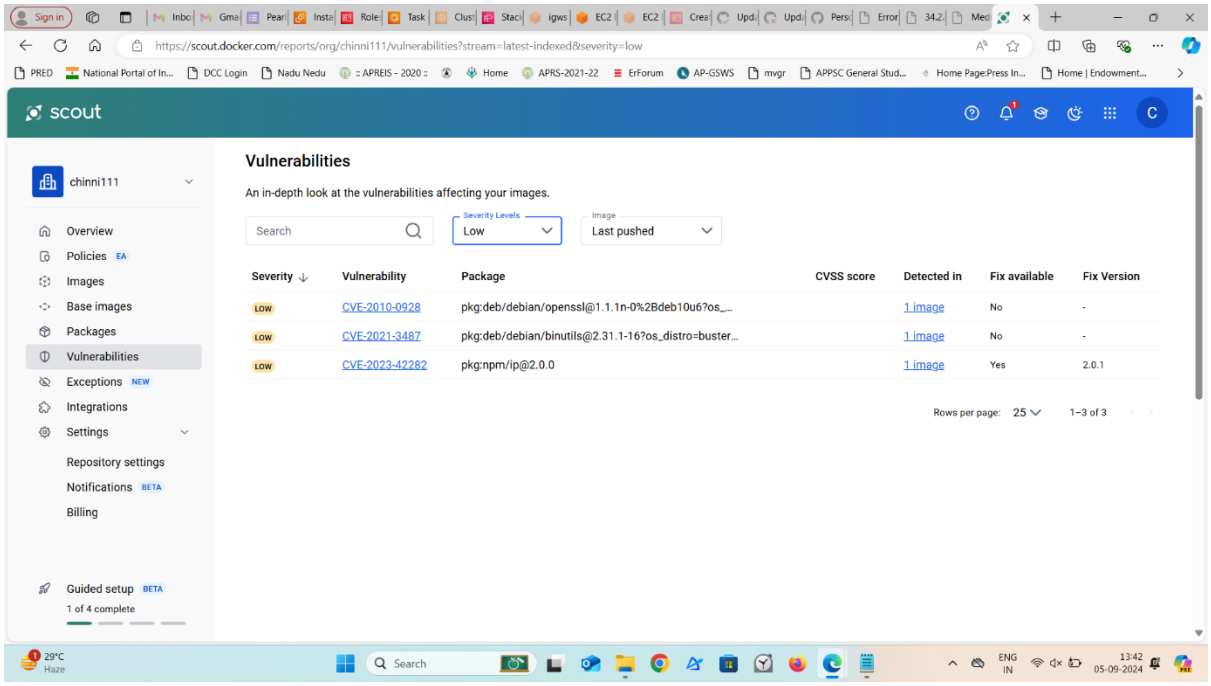





https://scout.docker.com/reports/org/chinni111/vulnerabilities?stream=latest-indexed&severity=critical
scout
chinni111
Overview
Policies EA
Images
Base images
Packages
Vulnerabilities
Exceptions NEW
Integrations
Settings
Repository settings
Notifications BETA
Billing
Guided setup BETA
1 of 4 complete
Vulnerabilities
An in-depth look at the vulnerabilities affecting your images.
Search
Severity Levels
Critical
Image
Last pushed
Severity
Vulnerability
Package
CVSS score
Detected in
Fix available
Fix Version
CRITICAL
CVE-2023-24540
pkg:golang/stdlib@1.20.3
9.8
1 image
Yes
1.20.4
CRITICAL
CVE-2024-24790
pkg:golang/stdlib@1.20.3
9.8
1 image
Yes
1.21.11
CRITICAL
CVE-2023-45133
pkg:npm/%40babel/traverse@7.22.5
9.3
1 image
Yes
7.23.2
Rows per page: 25
1–3 of 3

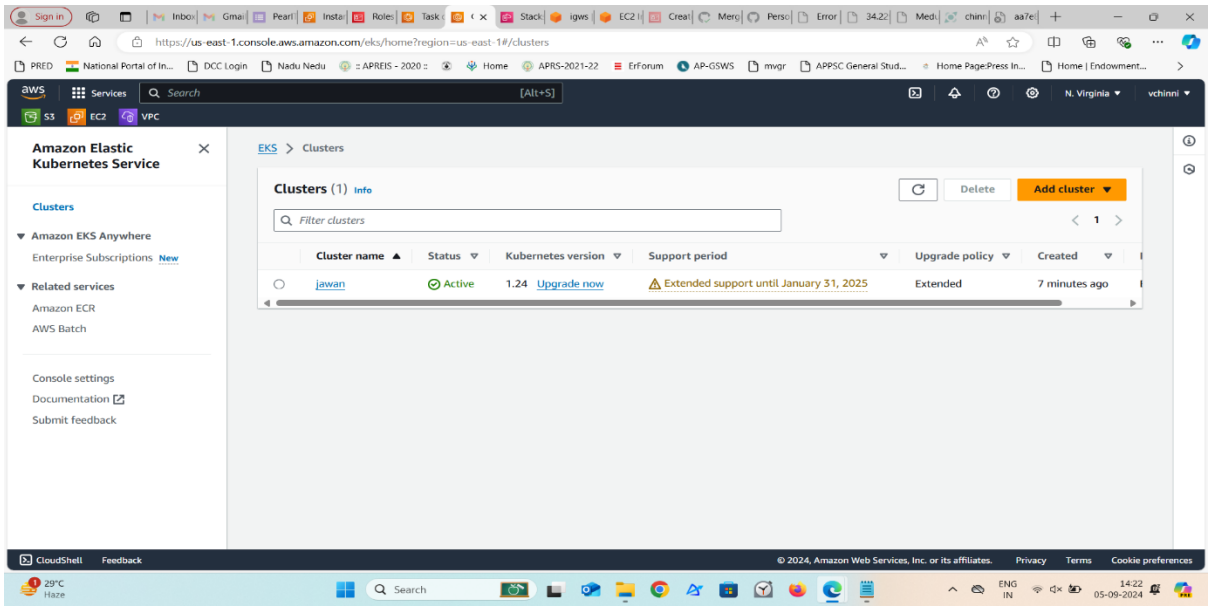

https://scout.docker.com/reports/org/chinni111/vulnerabilities?stream=latest-indexed&severity=high
chinni111
Vulnerabilities
An in-depth look at the vulnerabilities affecting your images.
Search
Severity Levels
High
Image
Last pushed
Severity
Vulnerability
Package
CVSS score
Detected in
Fix available
Fix Version
HIGH
CVE-2022-0235
pkg:npm/node-fetch@1.7.3
8.8
1 image
Yes
2.6.7
HIGH
CVE-2024-29415
pkg:npm/ip@2.0.0
8.1
1 image
No
HIGH
CVE-2018-20225
pkg:pypi/pip@24.0
7.8
1 image
No
HIGH
CVE-2023-29403
pkg:golang/stdlib@1.20.3
7.8
1 image
Yes
1.20.5
HIGH
CVE-2022-25883
pkg:npm/semver@7.3.7
7.5
1 image
Yes
7.5.2
HIGH
CVE-2023-39325
pkg:golang/stdlib@1.20.3
7.5
1 image
Yes
1.20.10
HIGH
CVE-2023-44487
pkg:golang/stdlib@1.20.3
7.5
1 image
Yes
1.20.10
HIGH
CVE-2023-45283
pkg:golang/stdlib@1.20.3
7.5
1 image
Yes
1.20.11
HIGH
CVE-2023-45288
pkg:golang/stdlib@1.20.3
7.5
1 image
Yes
1.21.9
HIGH
CVE-2024-24784
pkg:golang/stdlib@1.20.3
7.5
1 image
Yes
1.21.8
HIGH
CVE-2024-24791
pkg:golang/stdlib@1.20.3
7.5
1 image
Yes
1.21.12
HIGH
CVE-2023-24539
pkg:golang/stdlib@1.20.3
7.3
1 image
Yes
1.20.4
HIGH
CVE-2023-29400
pkg:golang/stdlib@1.20.3
7.3
1 image
Yes
1.20.4


https://scout.docker.com/reports/org/chinni111/vulnerabilities?stream=latest-indexed&severity=medium
scout
chinni111
Vulnerabilities
An in-depth look at the vulnerabilities affecting your images.
Search
Severity Levels
Medium
Image
Last pushed
Severity
Vulnerability
Package
CVSS score
Detected in
Fix available
Fix Version
MEDIUM
CVE-2023-29406
pkg:golang/stdlib@1.20.3
6.5
1 image
Yes
1.20.6
MEDIUM
CVE-2023-45857
pkg:npm/axios@0.24.0 1 more
6.5
1 image
Yes
1.6.0
MEDIUM
CVE-2024-28863
pkg:npm/tar@6.1.11
6.5
1 image
Yes
6.2.1
MEDIUM
CVE-2023-39318
pkg:golang/stdlib@1.20.3
6.1
1 image
Yes
1.20.8
MEDIUM
CVE-2023-39319
pkg:golang/stdlib@1.20.3
6.1
1 image
Yes
1.20.8
MEDIUM
CVE-2024-29041
pkg:npm/express@4.18.2
6.1
1 image
Yes
4.19.2
MEDIUM
CVE-2019-3902
pkg:pypi/mercurial@4.8.2
5.9
1 image
Yes
4.9
MEDIUM
CVE-2024-24789
pkg:golang/stdlib@1.20.3
5.5
1 image
Yes
1.21.11
MEDIUM
CVE-2020-15168
pkg:npm/node-fetch@1.7.3
5.3
1 image
Yes
2.6.1
MEDIUM
CVE-2023-29409
pkg:golang/stdlib@1.20.3
5.3
1 image
Yes
1.20.7
MEDIUM
CVE-2023-39326
pkg:golang/stdlib@1.20.3
5.3
1 image
Yes
1.20.12
MEDIUM
CVE-2023-45284
pkg:golang/stdlib@1.20.3
5.3
1 image
Yes
1.20.11

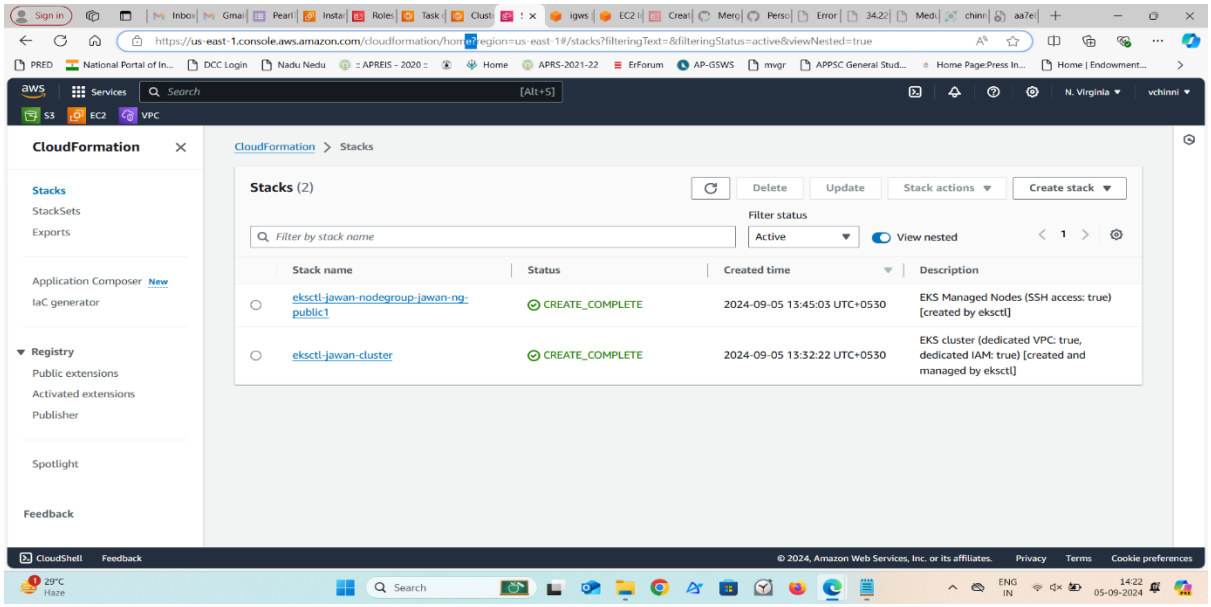


To access your Docker image using a Kubernetes Load Balancer, you need to follow these steps:

First create EKS cluster



Create node group



Config kubectl

1. Deploy Your Docker Image to a Kubernetes Cluster

Assuming you already have a Kubernetes cluster running, you need to deploy your Docker image to it.

1. Create a Kubernetes Deployment: Create a deployment.yaml file
2. Apply the Deployment: kubectl apply -f deployment.yaml
3. Verify Deployment: kubectl get deployments && kubectl get pods

2. Expose the Deployment with a Service

Create a service.yaml file to expose your deployment:

This configuration will create a LoadBalancer service that routes traffic to your pods.

3. Apply the Service: kubectl apply -f service.yaml

4. Check the Load Balancer

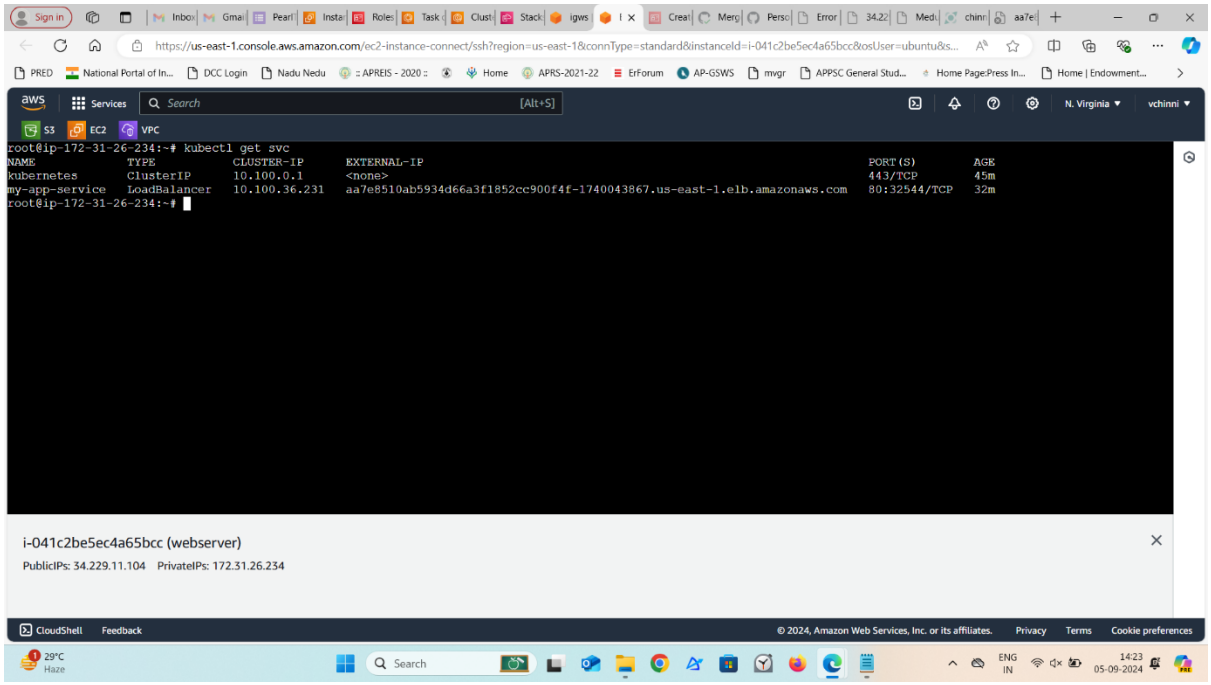
After applying the service, Kubernetes will provision a cloud load balancer and associate it with your service.

1. Get the Service Details: kubectl get services

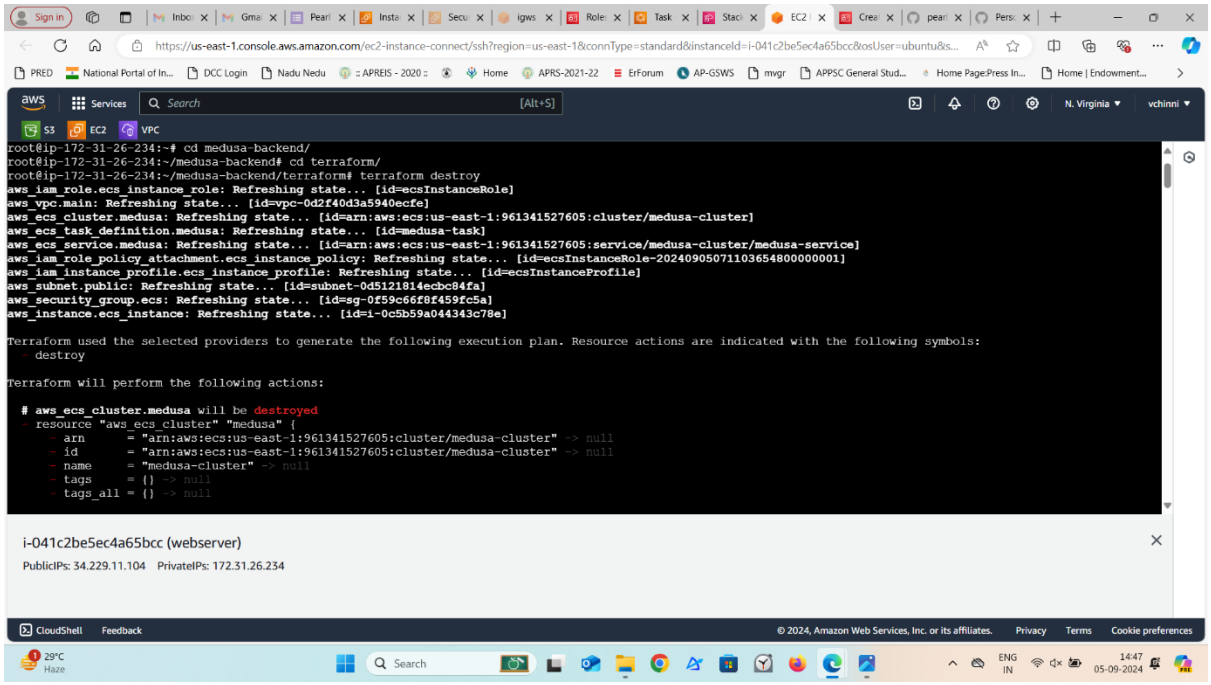
Look for the EXTERNAL-IP of my-app-service. This is the public IP or DNS name of your Load Balancer.

2. **Access Your Application**

Open your web browser and navigate to the EXTERNAL-IP or DNS name.



Destroy the configuration



Terraform and ECS Deployment

1. ECS Configuration:
    - The ecs.tf file defines ECS resources with the following configurations:
        - network_mode set to bridge
        - launch_type set to EC2
    - The ECS service does not include network configuration as it is not required for EC2 launch type with bridge mode.
2. Disk Space Issues:
    - You encountered pod eviction issues in your Kubernetes cluster due to disk space. Here's how disk space was checked:
    - df -h /var/lib/docker
3. Error Handling in ECS Cluster Deletion:
    - You encountered the error: ClusterContainsContainerInstancesException: The Cluster cannot be deleted while Container Instances are active or draining.
    - To resolve this, you deregistered the ECS container instances:
    - aws ecs deregister-container-instance --cluster medusa-cluster --container-instance <container-instance-arn> --force
        - However, the correct container-instance-arn had to be fetched using:
        - aws ecs list-container-instances --cluster medusa-cluster

Conclusion: In this project, you set up and deployed the Medusa backend on an AWS Ubuntu server using PostgreSQL, Redis, and ECS. You automated infrastructure provisioning and deployment using Terraform and GitHub Actions, ensuring a streamlined CI/CD pipeline. Despite facing challenges like ECS cluster deletion and resource management, you resolved issues efficiently, demonstrating your cloud expertise and troubleshooting skills.