MayAkosua25 Class Project: ECS Deployment Guide

📄 Presentation Title:

Deploying a Flask App to AWS using Docker, ECR, and ECS (Fargate)

👤 Introduction

We are team **MayAkosua25**. In this project, we built a simple Flask web app, containerized it using Docker, pushed it to AWS Elastic Container Registry (ECR), and deployed it using ECS (Fargate). This presentation shows our process, the challenges we faced, and what we learned.

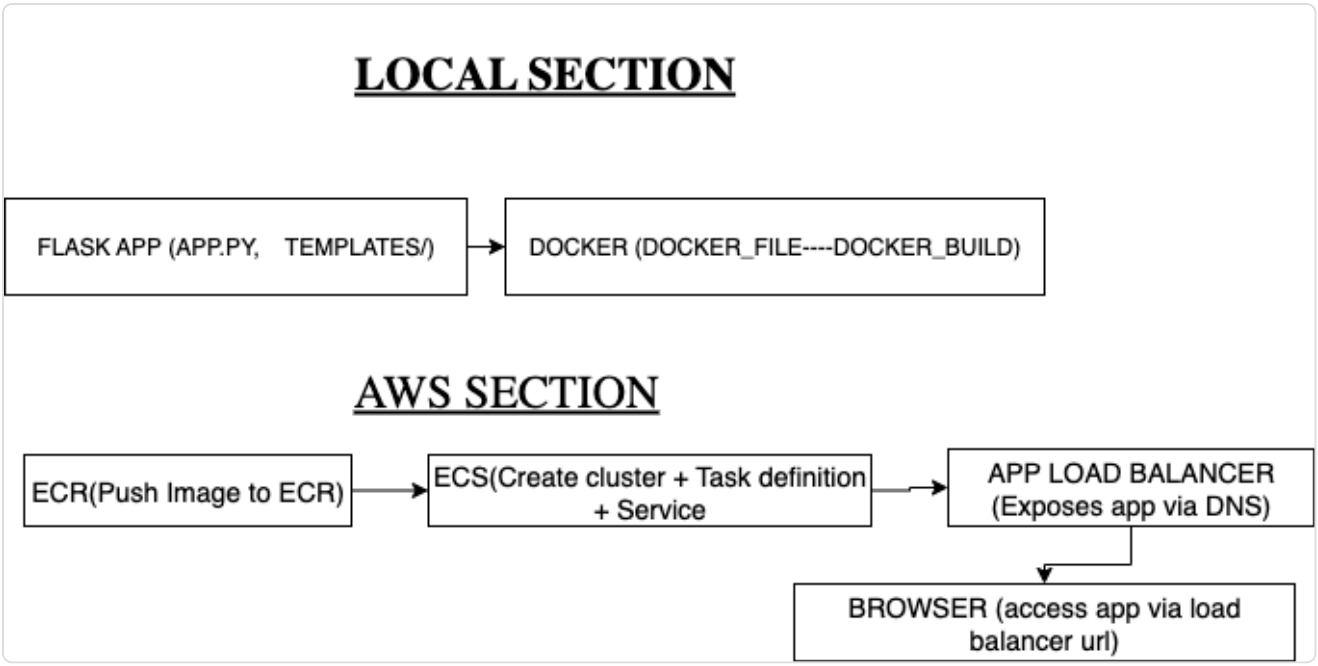
🧰 What We Did Before the Cloud

Before using AWS services, we worked on our project locally. These were the key steps:

- Built a basic **Flask web application**
- Created `index.html` inside a `templates/` folder
- Wrote backend logic using `app.py`
- Tested the app on `localhost:5000` using `flask run`
- Created a `Dockerfile` and tested the Docker container locally

🔄 Workflow Overview

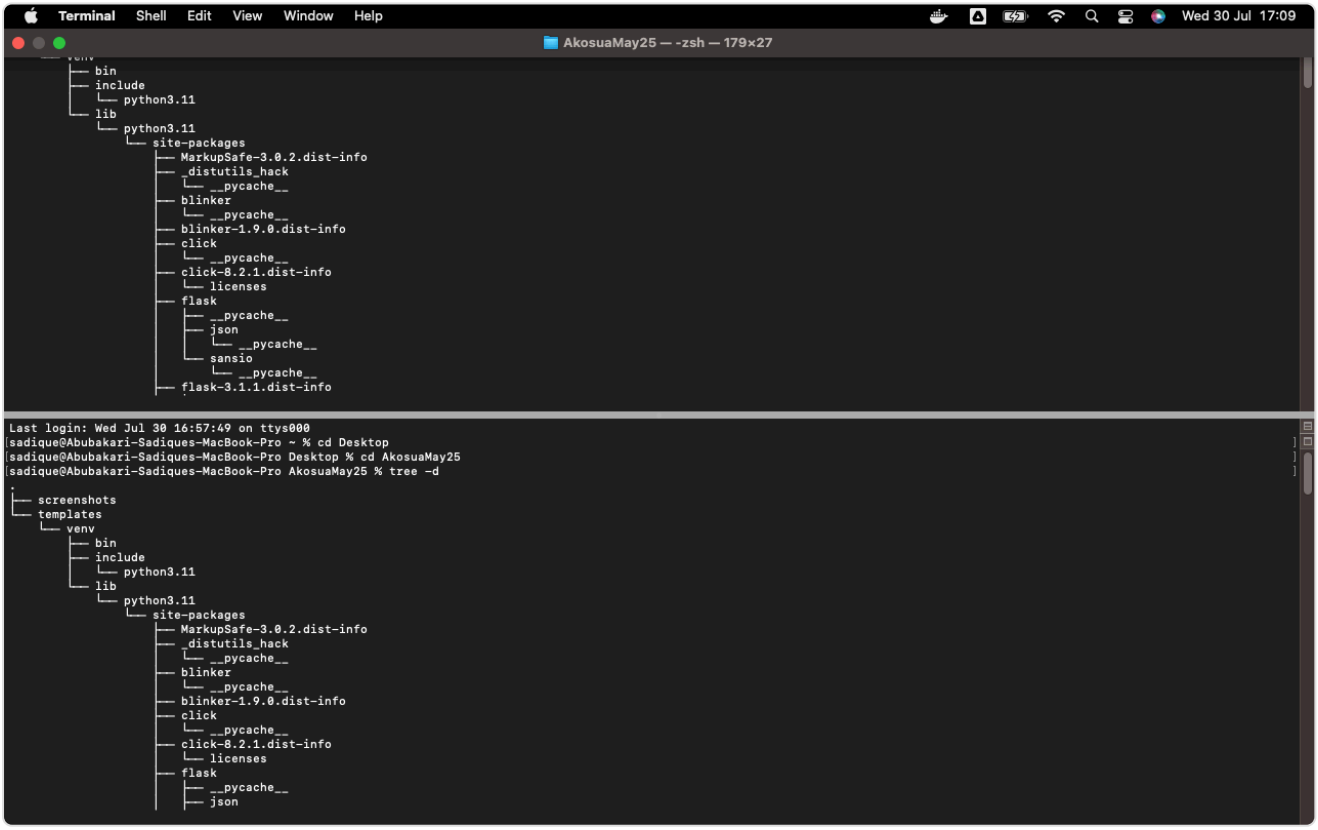
1. Build Flask app locally
2. Create a Docker image
3. Push image to Amazon ECR
4. Create ECS Cluster (Fargate)
5. Define Task
6. Deploy Service with Load Balancer
7. Access app in browser



✅ Step-by-Step Guide

⭐ Step 1: Build the Flask App

- Basic Flask app with `index.html` in a `templates/` folder
- `app.py` serves on port `5000`



⭐ Step 2 & 3: Create Dockerfile, Build Image & Push to ECR

1. Created a Dockerfile

2. Built image:

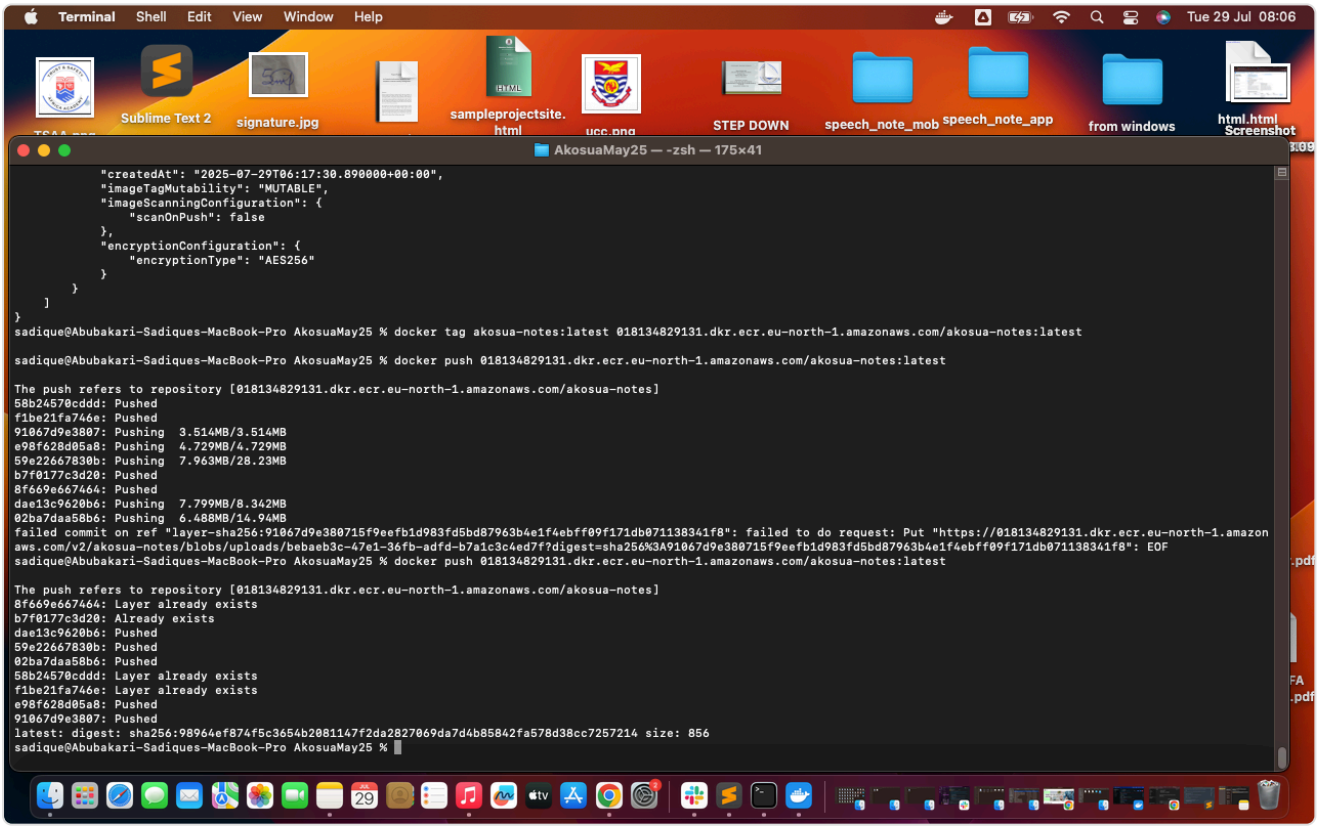
```
docker build -t MayAkosua25-notes .
```

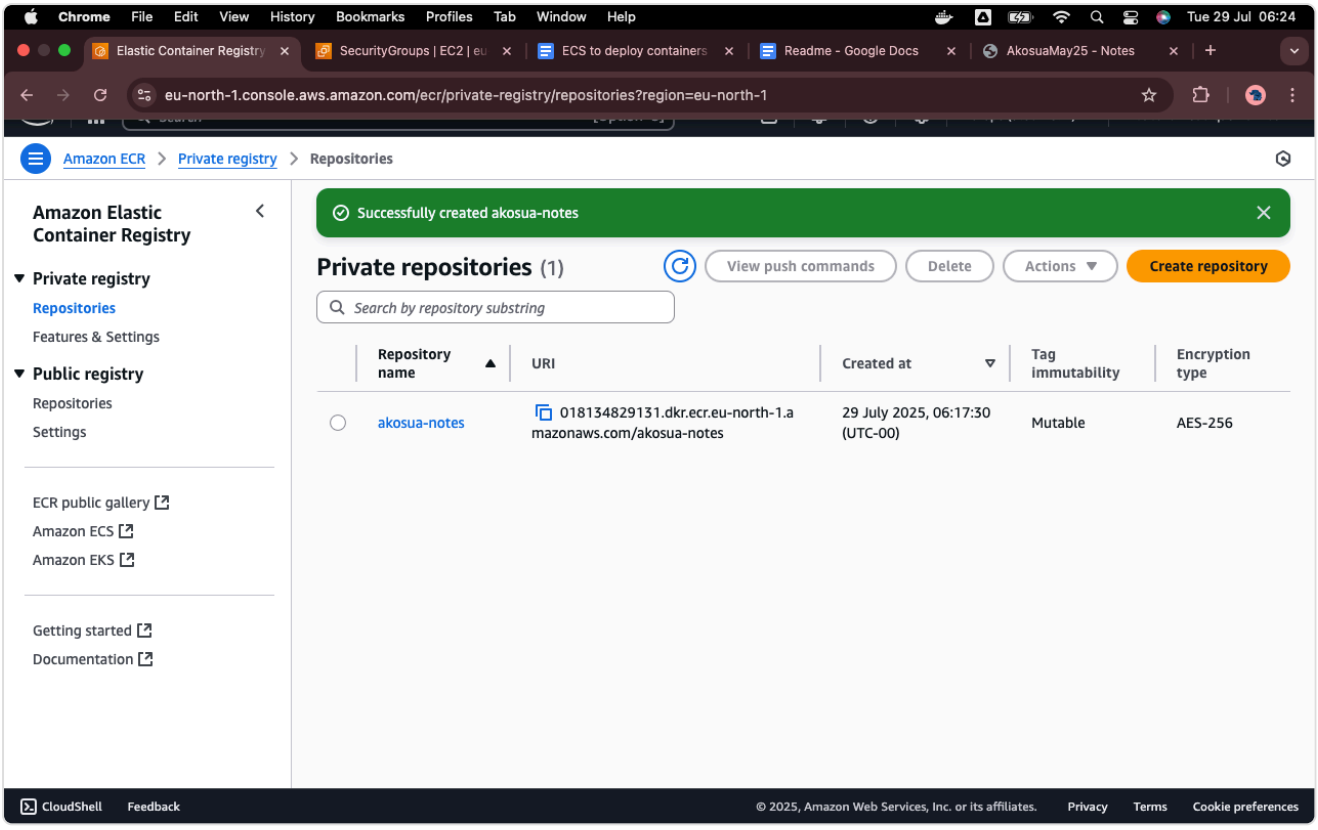
3. Created ECR repo: `MayAkosua25-notes`
4. Logged in to ECR:

```
aws ecr get-login-password | docker login --username AWS --password-stdin ·
```

5. Tagged and pushed the image:

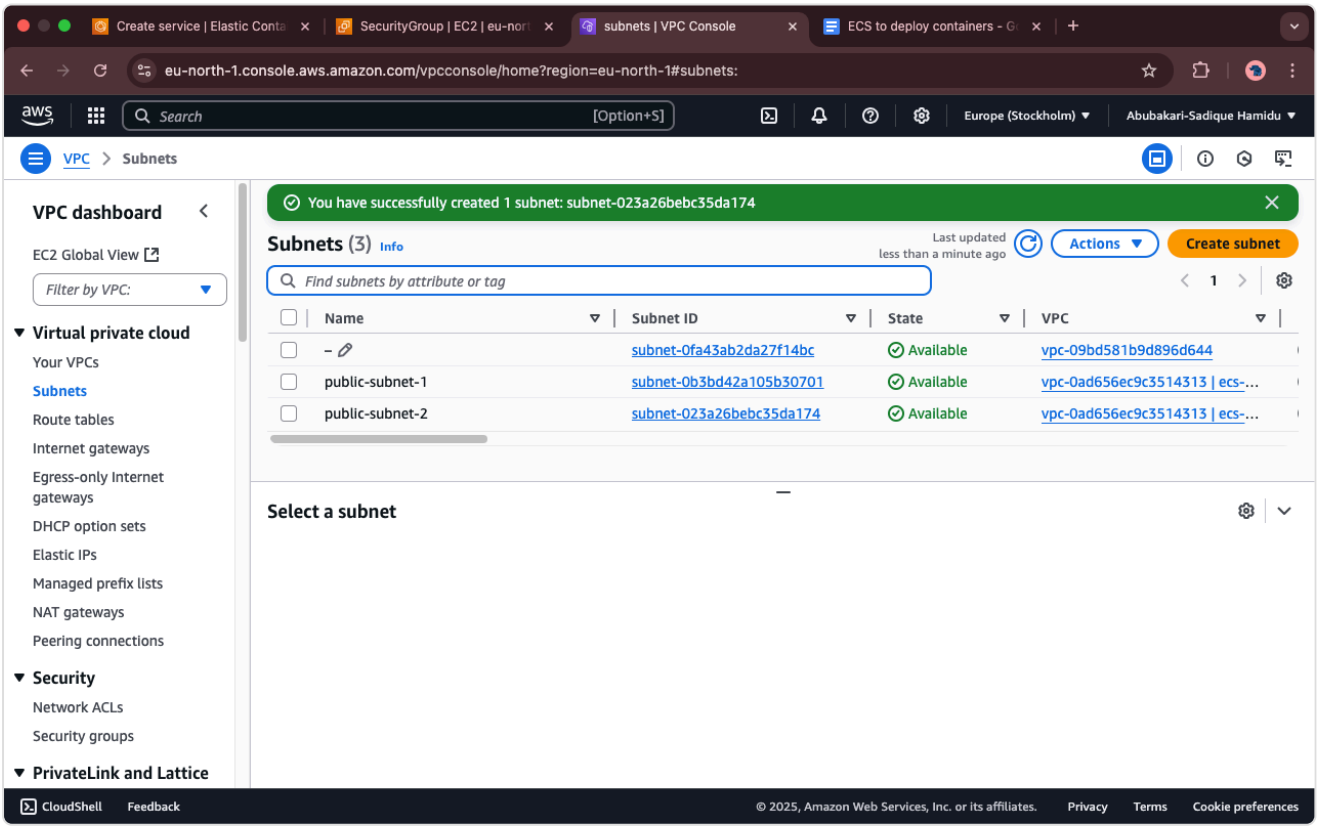
```
docker tag MayAkosua25-notes:latest <your-ecr-url>/MayAkosua25-notes:latest
docker push <your-ecr-url>/MayAkosua25-notes:latest
```





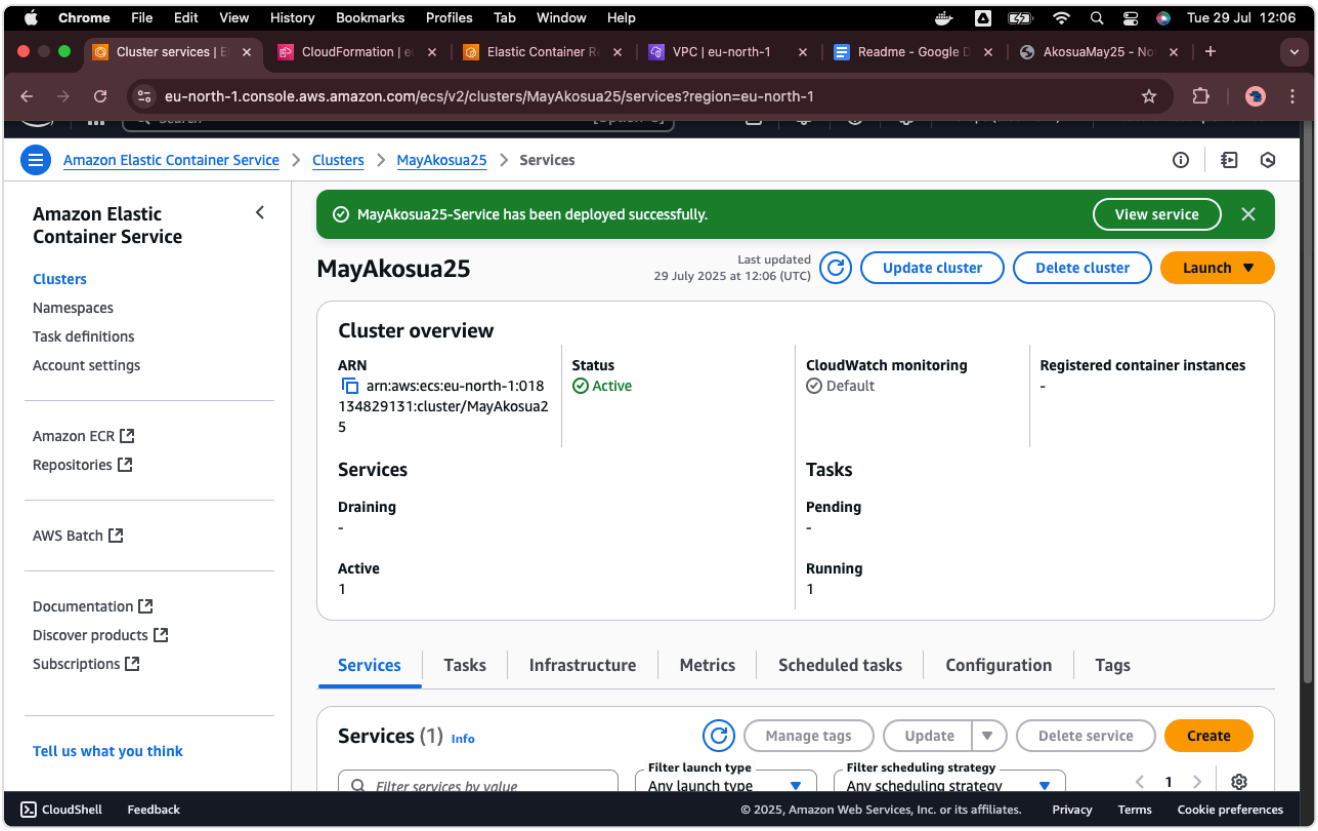
⭐ Step 4: Setup the Network

- Created a second **public subnet** in a different Availability Zone
- Attached Internet Gateway and updated the route table



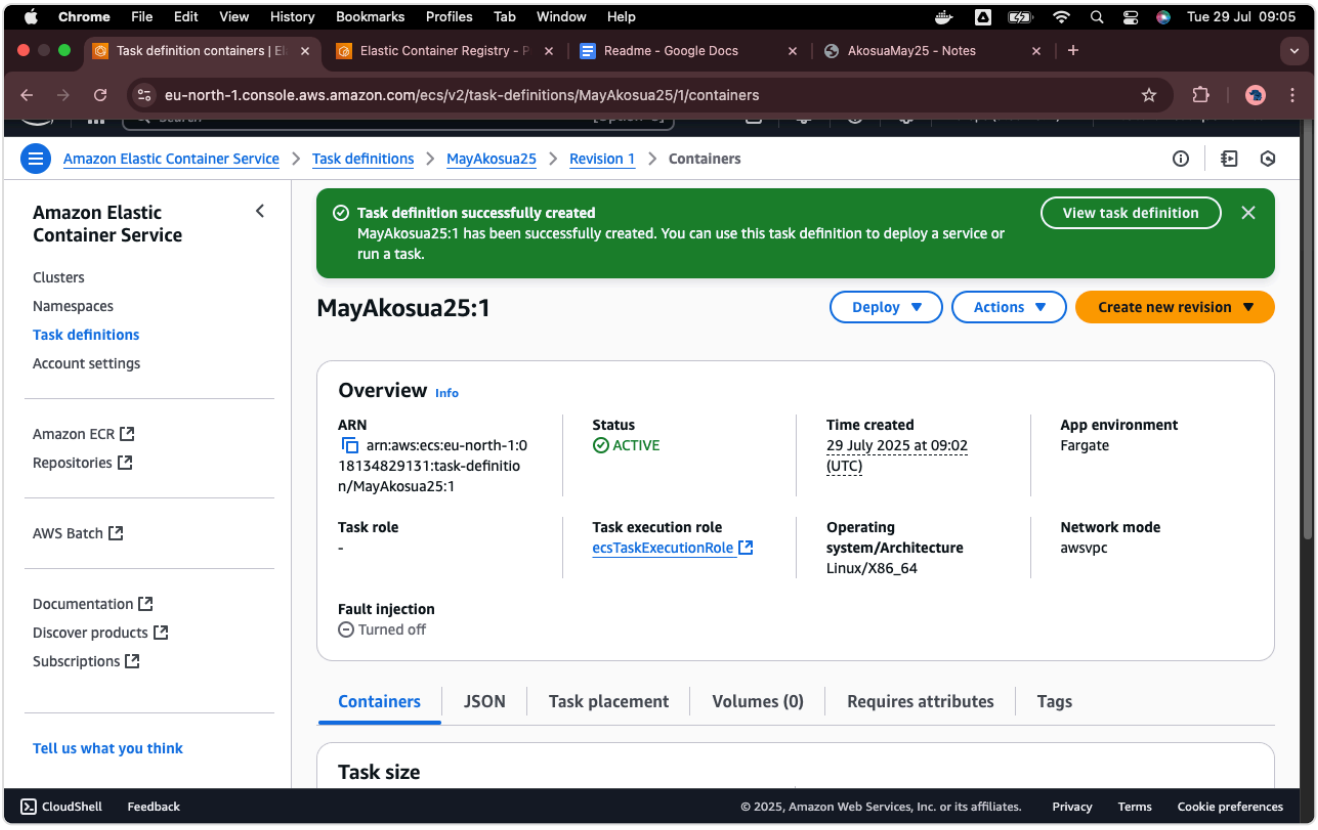
⭐ Step 5: Create ECS Cluster

- Type: **Networking only (Fargate)**
- Cluster name: `MayAkosua25-cluster`



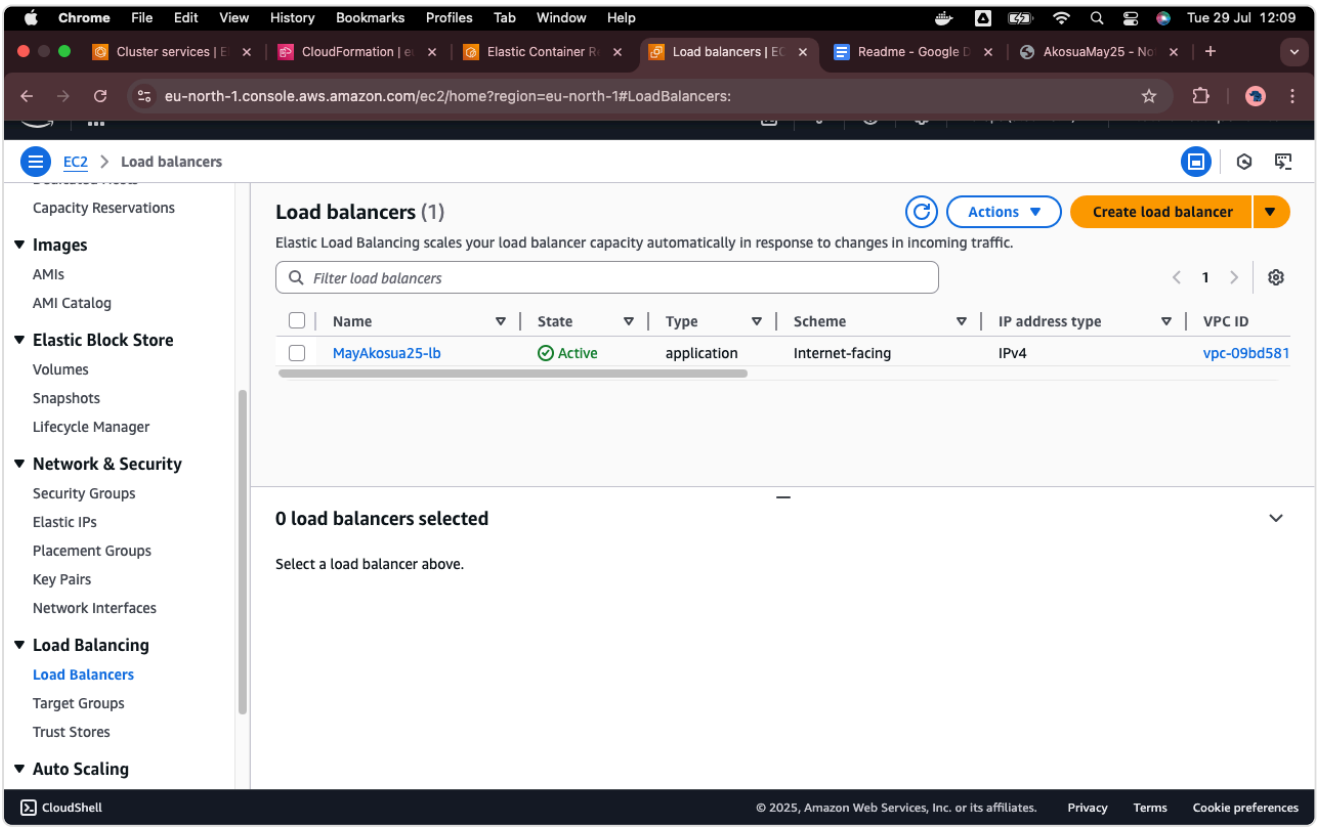
⭐ Step 6: Create Task Definition

- Task name: `MayAkosua25-task`
- Runtime: Fargate
- Container:
  - Image: `<your-ecr-url>/MayAkosua25-notes:latest`
  - Port mapping: `5000`



⭐ Step 7: Deploy Service with Load Balancer

- Deployed service inside ECS Cluster
- Service name: `MayAkosua25-service`
- Used an Application Load Balancer (ALB)
- Selected 2 public subnets in different AZs
- Created a Target Group on port `5000`

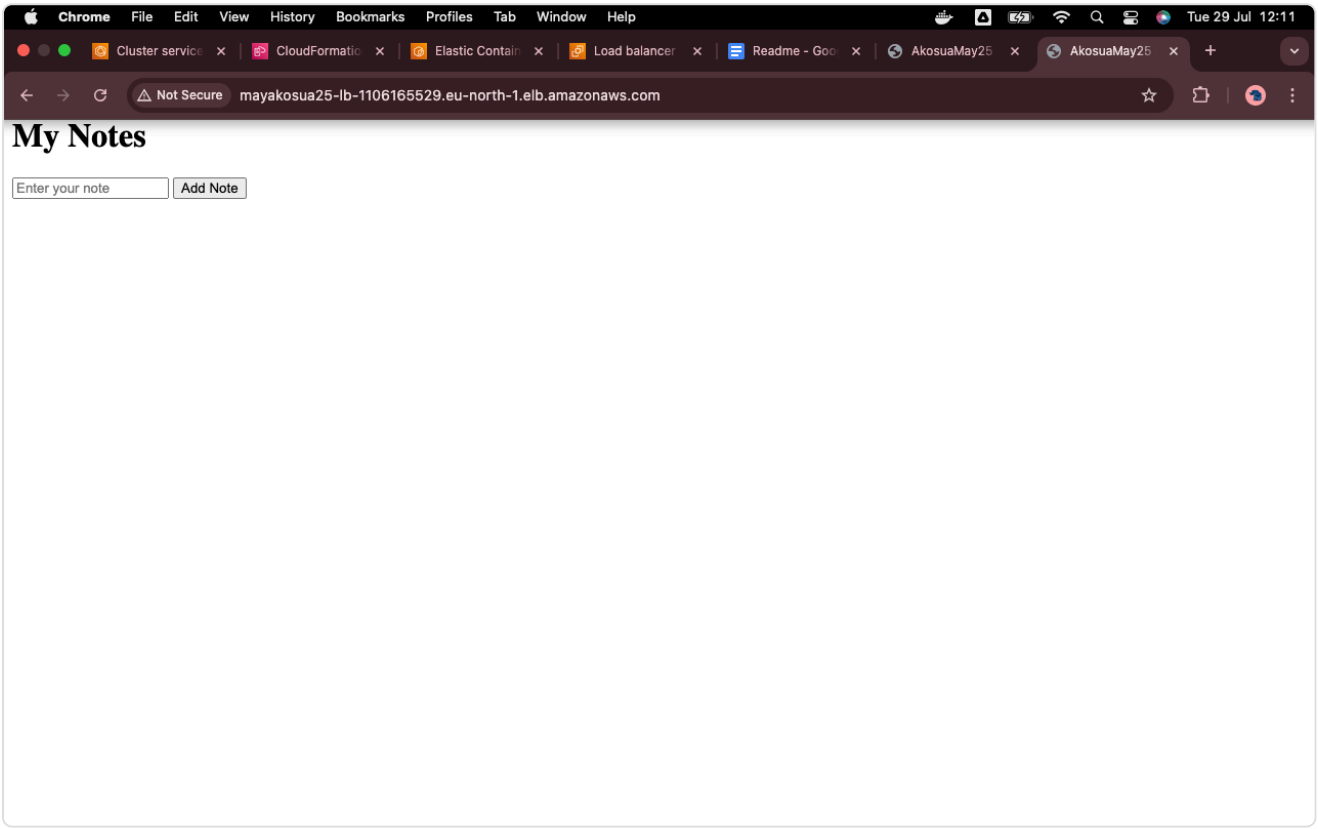


⭐ Step 8: Access the App

Load Balancer DNS:

```
http://mayakosua25-lb-11061#####.eu-north-1.elb.amazonaws.com
```

Open this address in your browser to see the running Flask app!



❌ Errors & Fixes

- **No such file or directory** – Fixed by checking that all required files (like `requirements.txt`) existed and Docker paths were correct
- **Push access denied** – Fixed by logging in to ECR using the correct `aws ecr get-login-password` command
- **Load Balancer creation failed** – Solved by creating two public subnets in different Availability Zones and ensuring they had route access via Internet Gateway
- **Stack already exists** – Fixed by going to CloudFormation and deleting the existing stack, then re-running the deployment

🎯 Final Notes

- Always use 2 public subnets in separate AZs when using Load Balancers
- Remember to assign the correct IAM role to ECS tasks
- Check CloudFormation stacks if the UI throws errors—it controls infrastructure

📊 What We Learned

- AWS networking and IAM can be tricky but manageable with practice
- Containers simplify app deployment across environments
- Debugging is essential to mastering cloud deployments
- We improved our skills in both development and DevOps practices

Thank you! ✨

Presented by Team Amalitech || MayAkosua25