

Assignment 2

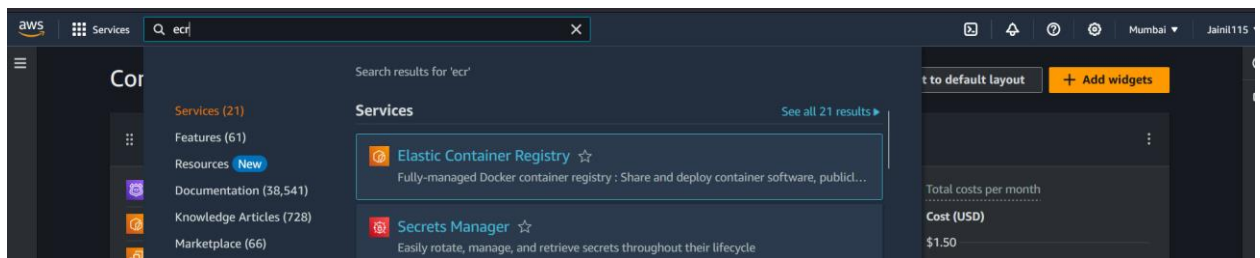
Objective: Create a GitHub Actions workflow that automates the building and deployment of a Node.js application to Amazon ECS using Amazon ECR.

Tasks:

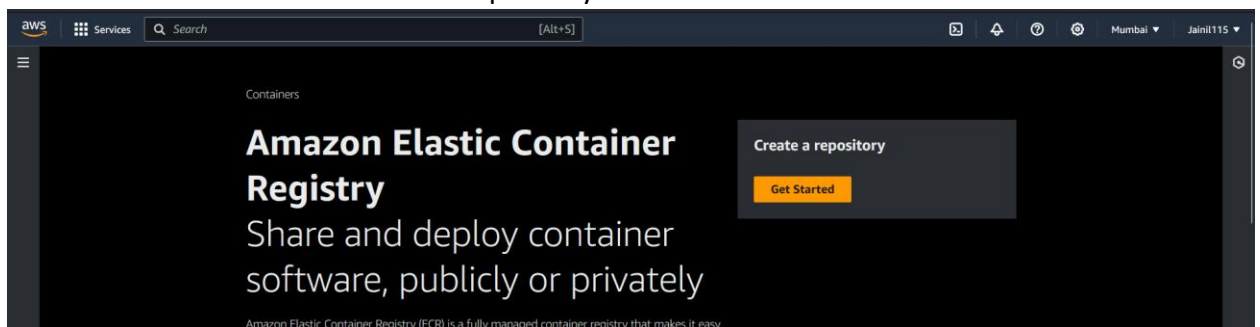
1. Set up GitHub Actions Workflow
2. Define Environment Variables
3. Build and Push Docker Image
4. Deploy to Amazon ECS
5. Complete Workflow

Steps to create ECR (Elastic Container Registry).

1. Go to aws console and search for ecr.



2. Then click on Get Started in create repository.



3. Change the visibility setting to private for this assignment.

Create repository

General settings

Visibility settings | [Info](#)
Choose the visibility setting for the repository.

☒ **Private**
Access is managed by IAM and repository policy permissions.

☐ **Public**
Publicly visible and accessible for image pulls.

Repository name
Provide a concise name. A developer should be able to identify the repository contents by the name.

171358186705.dkr.ecr.ap-south-1.amazonaws.com/

9 out of 256 characters maximum (2 minimum). The name must not contain hyphens, underscores, periods and forward slashes.

Tag immutability | [Info](#)
Enable tag immutability to prevent image tags from being overwritten by subsequent image pushes using the same tag. Disable tag immutability to allow image tags to be overwritten.

☐ **Disabled**

ⓘ Once a repository is created, the visibility setting of the repository can't be changed.

4. Now enter the repository name. nodejsapp in my case.

Create repository

General settings

Visibility settings | [Info](#)
Choose the visibility setting for the repository.

☒ **Private**
Access is managed by IAM and repository policy permissions.

☐ **Public**
Publicly visible and accessible for image pulls.

Repository name
Provide a concise name. A developer should be able to identify the repository contents by the name.

171358186705.dkr.ecr.ap-south-1.amazonaws.com/

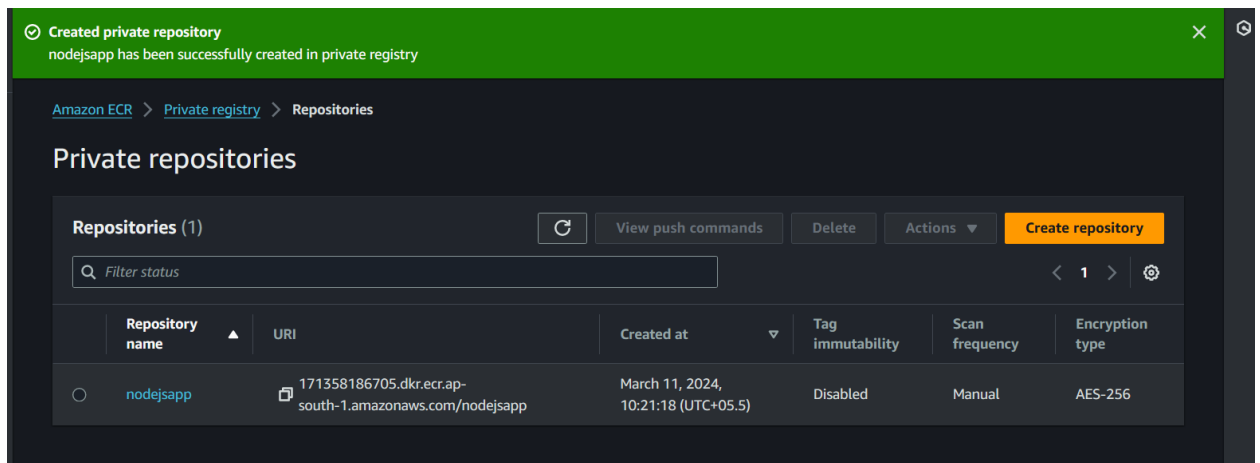
9 out of 256 characters maximum (2 minimum). The name must not contain hyphens, underscores, periods and forward slashes.

Tag immutability | [Info](#)
Enable tag immutability to prevent image tags from being overwritten by subsequent image pushes using the same tag. Disable tag immutability to allow image tags to be overwritten.

☐ **Disabled**

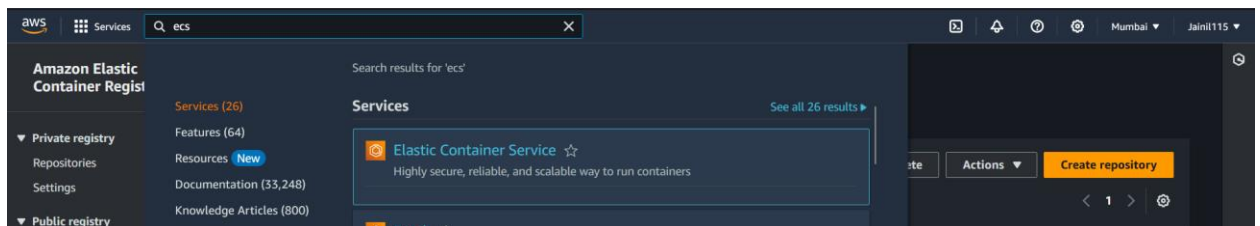
ⓘ Once a repository is created, the visibility setting of the repository can't be changed.

5. Then click on create repository.

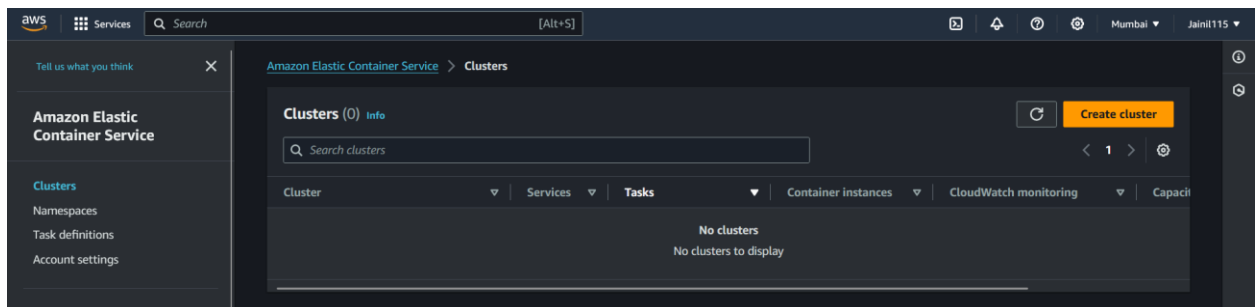


Let's now setup ecs.

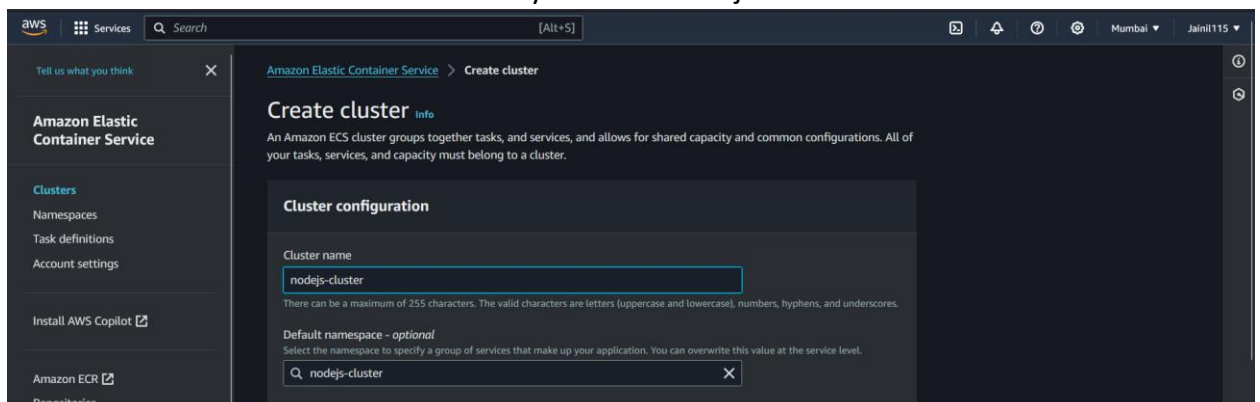
1. Search for ecs in aws console.



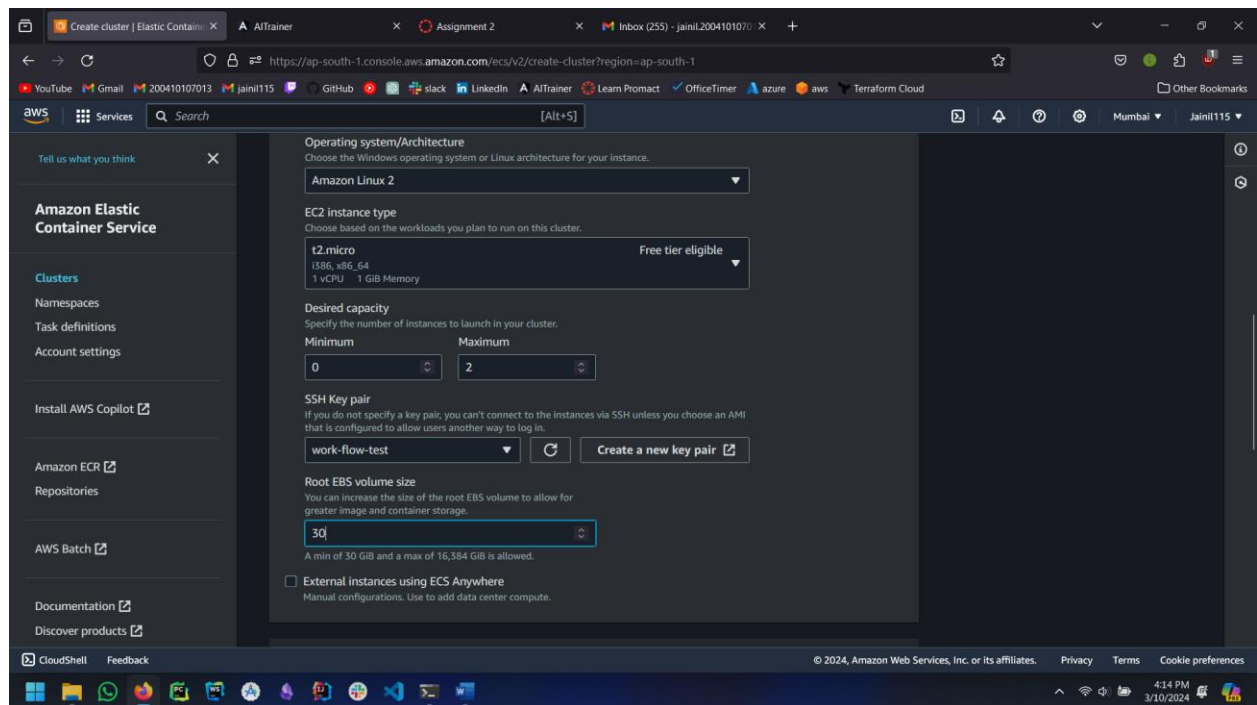
2. Now click on create cluster.



3. Now enter the name of the cluster. In my case it is nodejs-cluster.



4. Now fill the following details as follows:
 - a. Select EC2 Infrastructure
 - b. Select Auto Scaling Group
 - c. Provisioning model: On demand
 - d. Operating system: Amazon Linux 2
 - e. EC2 instance type: t2.micro
 - f. Desired capacity: Min: 0, Max:2
 - g. SSH Key Pair: select your ssh key pair, In my case work-flow-test.
 - h. Root EBS Volume Size: 30



5. Now click on create Security group and select the configuration show in below image:

Security group [Info](#)
Choose an existing security group or create a new security group.

☐ Use an existing security group

☒ Create a new security group

Security group details
Specify the configuration to use when creating the new security group.

Security group name
nodejs-group

The security group name can have up to 255 characters. Valid characters: A-Z, a-z, 0-9, spaces, and the _-:/()#,@[]+=&{}!\$* special characters.

Security group description
cant ssh

The security group description can have up to 255 characters. Valid characters: A-Z, a-z, 0-9, spaces, and the _-:/()#,@[]+=&{}!\$* special characters.

Inbound rules for security groups
Add one or more ingress rules for your security group.

Type	Protocol	Port range	Source	Values	
Custom ... ▼	TCP	3000	Anywhere ▼	0.0.0.0/0, ::/0	Delete
HTTP ▼	TCP	80	Anywhere ▼	0.0.0.0/0, ::/0	Delete

Add rule

Auto-assign public IP [Info](#)
Choose whether to auto-assign a public IP to the Amazon EC2 instances

Use subnet setting ▼

- Leave everything as default and click on create ECS.

Cluster nodejs-cluster has been created successfully.

Amazon Elastic Container Service > Clusters

Clusters (1) [Info](#)

[nodejs-cluster](#) | 0 | No tasks running | 0 EC2 | Default | ASG

Now lets create load balancer for the ASG.

For that lets first create target group:

- For that go to target group in ec2 dashboard and click on create target group

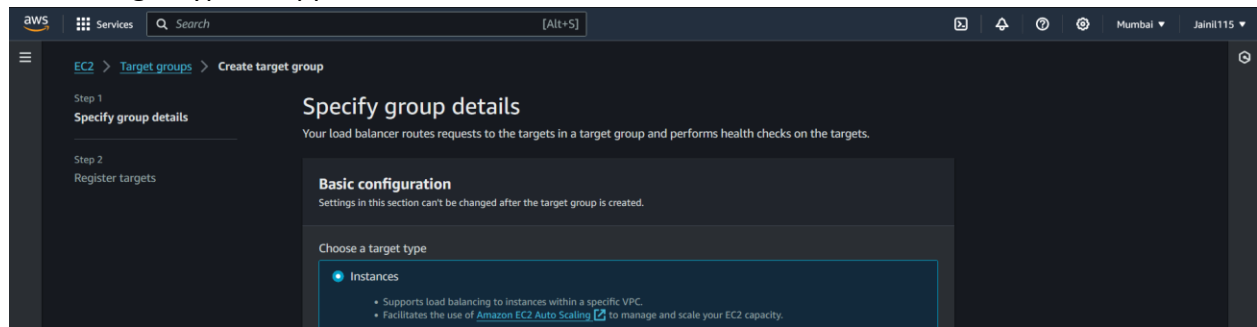
EC2 Dashboard

Target groups (2) [Info](#)

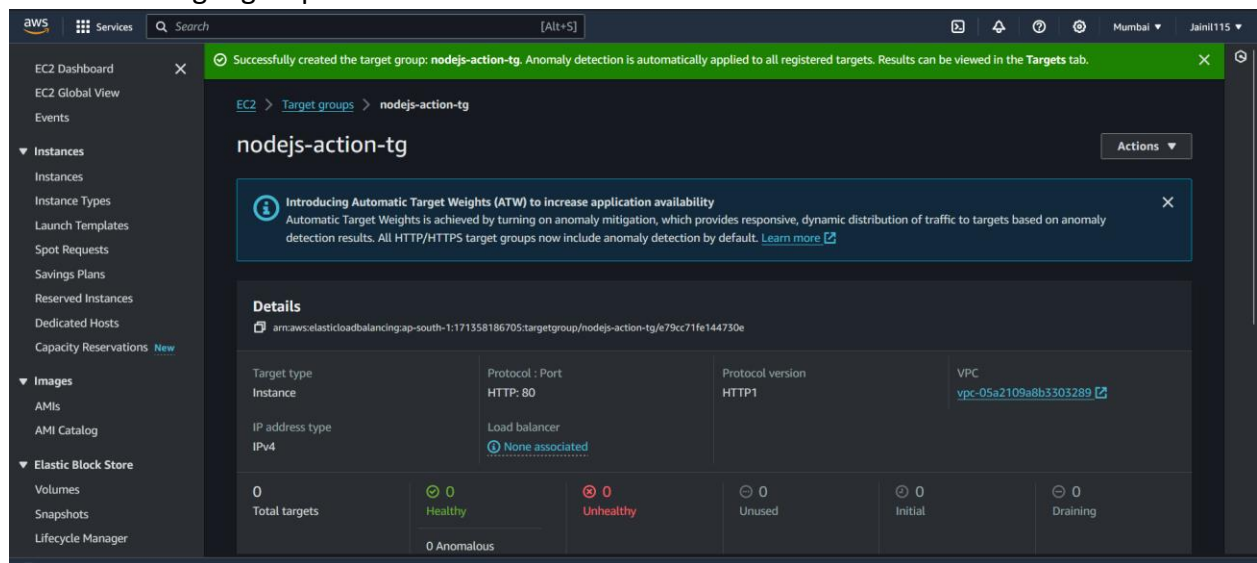
Name	ARN	Port	Protocol	Target type	Load balancer
------	-----	------	----------	-------------	---------------

Create target group

2. Select target type as application load balancer

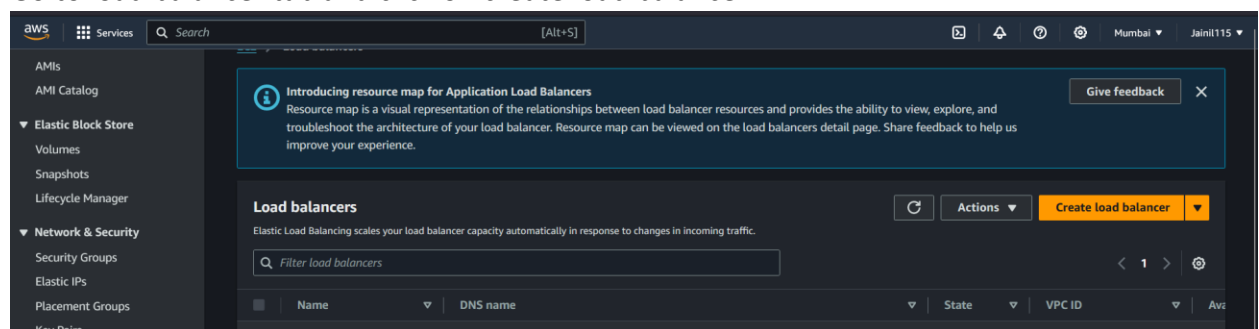


3. Now enter the target group name as nodejs-action-tg, Keep everything else default and create target group.

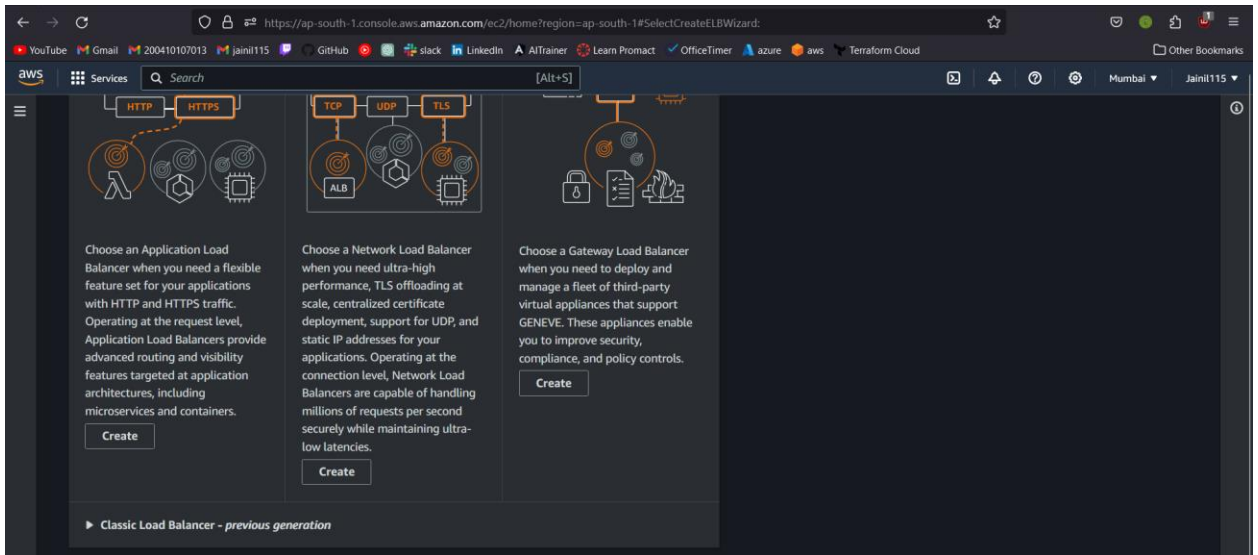


Now lets create that load balancer.

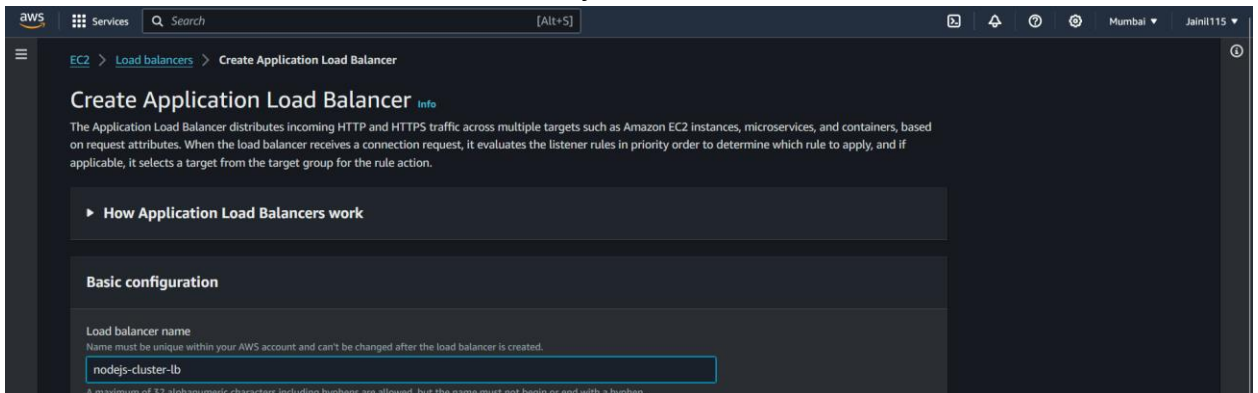
1. Go to load balancer tab and click on create load balancer



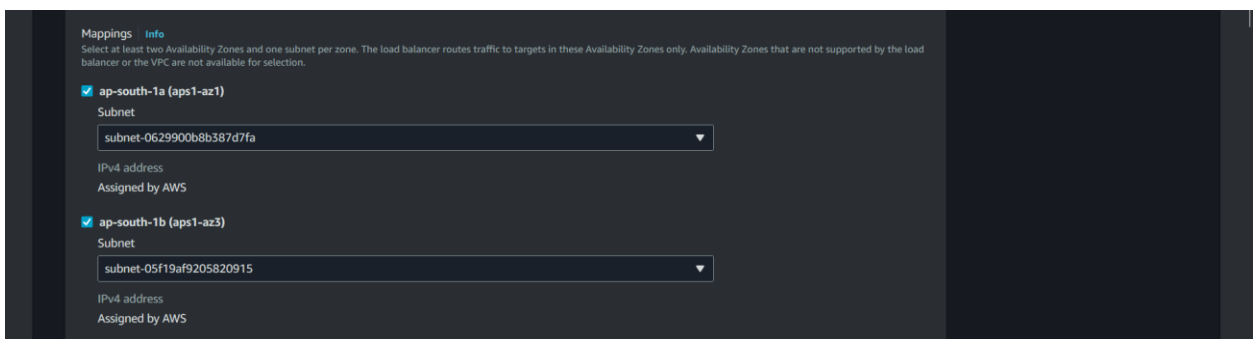
2. Select application load balancer



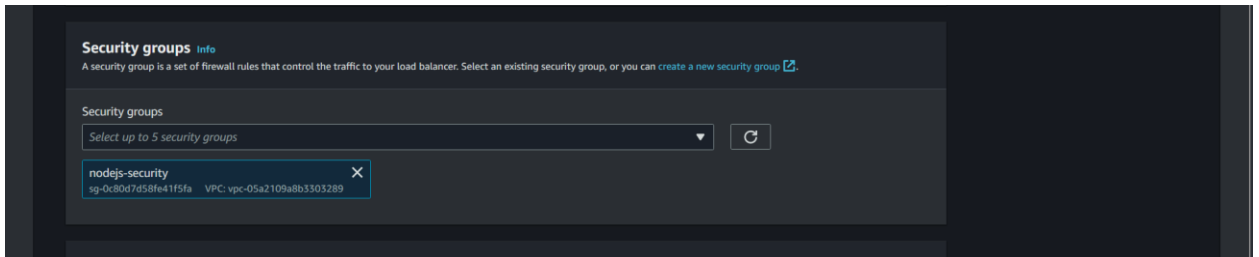
3. Enter the name of the load balancer as `nodejs-cluster-lb`



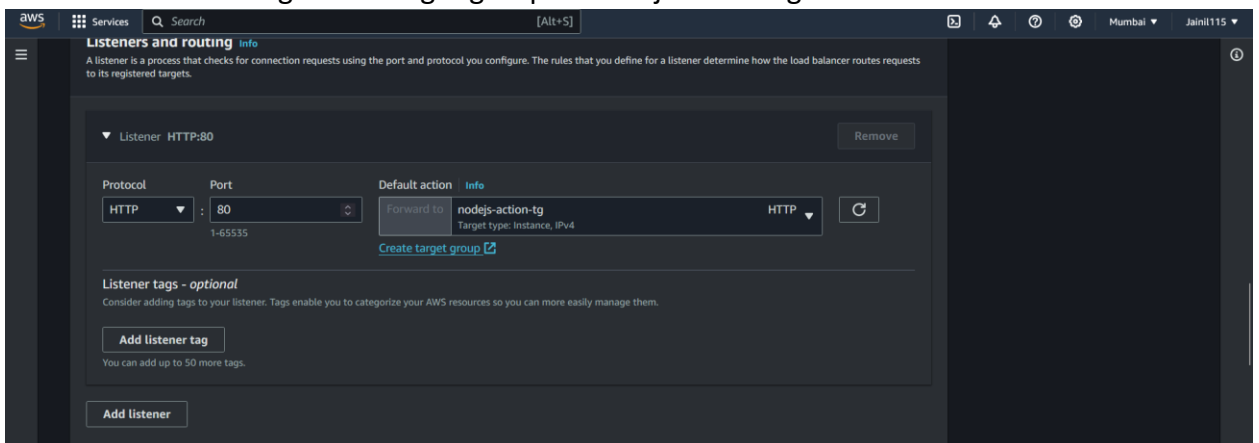
4. In network mapping select `ap-south-1a` and `ap-south-1b` in mapping and keep everything in network default



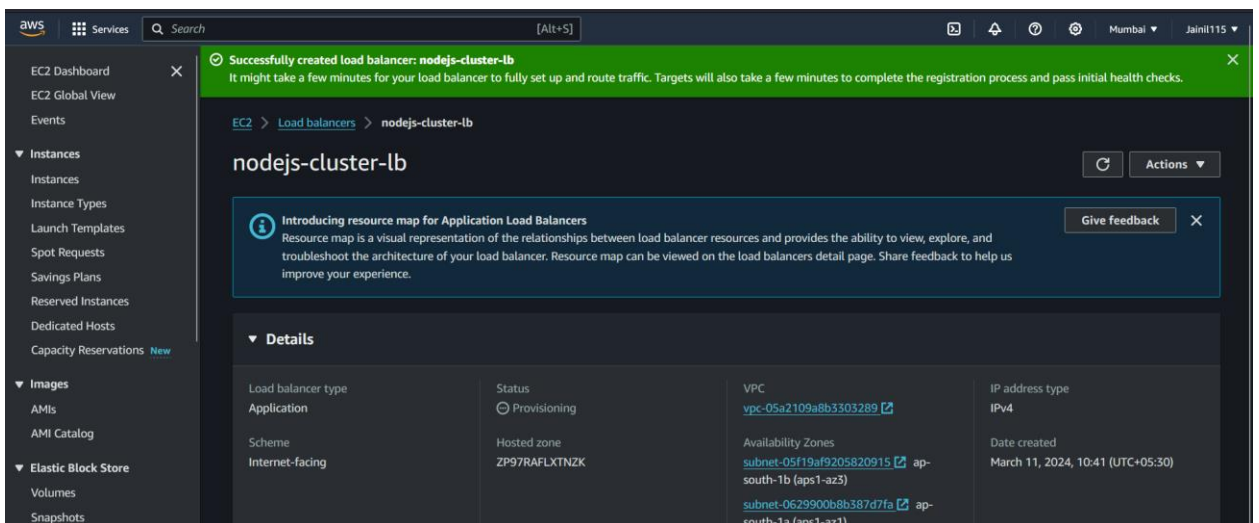
5. In security group select `nodejs-security`



6. In listener and routing select target group as nodejs-action-tg

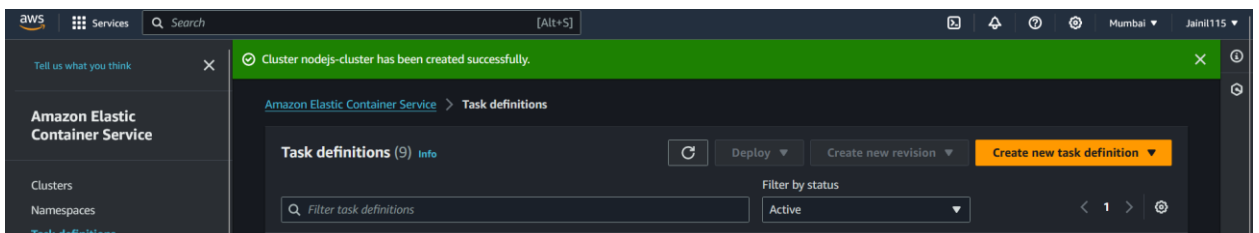


7. Then click on create load balancer.

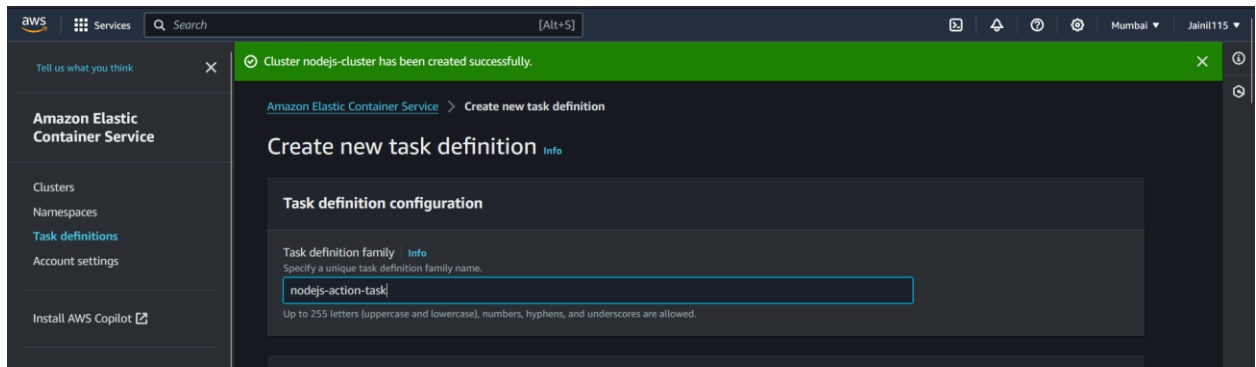


Now lets create task definition:

1. Go to task definition and click on create new task definition



2. Give it name nodejs-action-task



3. Select the following infrastructure requirements

Launch type: EC2

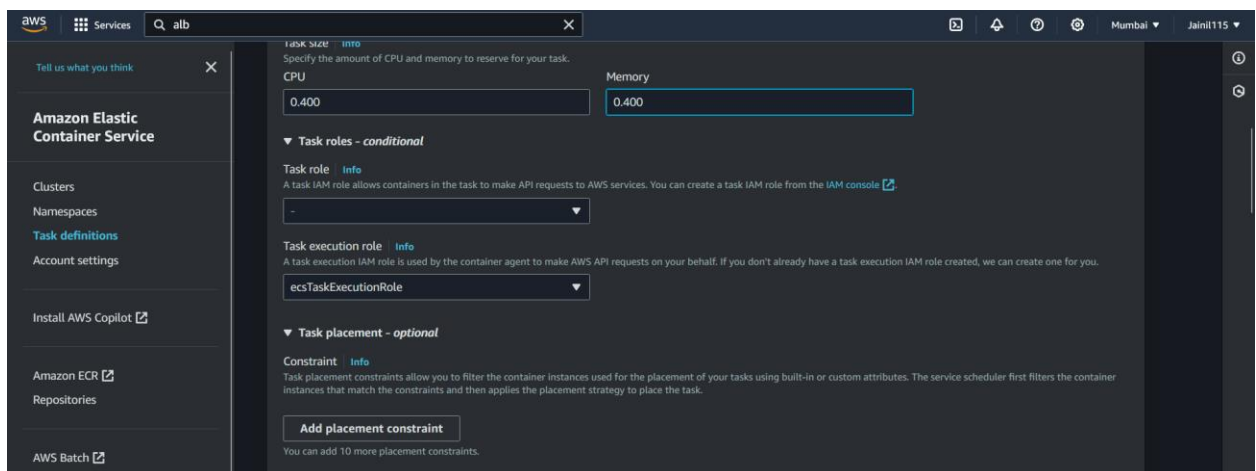
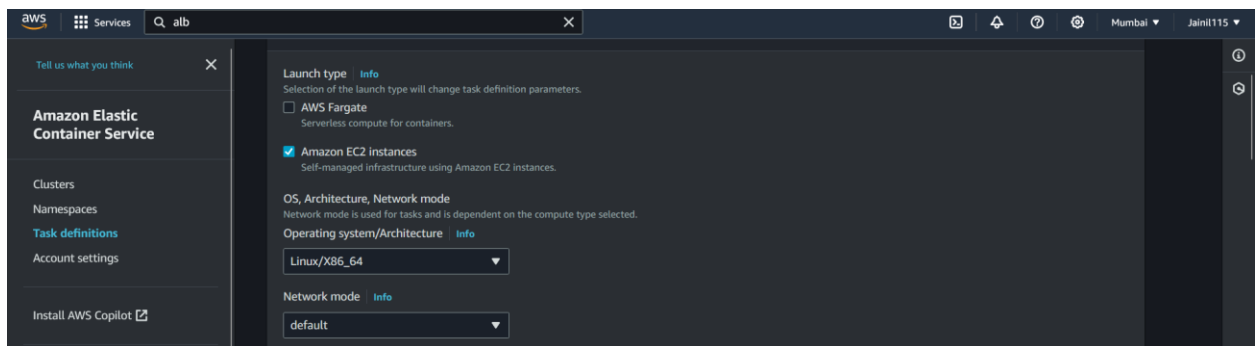
OS: Linux_X86_64

Network mode: default

Task Size: 0.400 vcpu and 0.400 MB memory

Task Role: -

Task execution Role: ecsTaskExecutionRole



4. Now in container details enter the following details:

a. Name: nodejsapp

b. Image URI: 171358186705.dkr.ecr.ap-south-1.amazonaws.com/nodejsapp:latest

- c. Essential container: yes
- d. Host port: 3000
- e. Container port: 3000
- f. Protocol: TCP
- g. portname: empty
- h. App protocol: HTTP

Container - 1 Info Essential container Remove

Container details
Specify a name, container image, and whether the container should be marked as essential. Each task definition must have at least one essential container.

Name	Image URI	Essential container
nodejsapp	171358186705.dkr.ecr.ap-south-1.amazonaws.com/nodejsapp:latest	Yes

Private registry Info
Store credentials in Secrets Manager, and then use the credentials to reference images in private registries.
☐ Private registry authentication

Port mappings Info
Add port mappings to allow the container to access ports on the host to send or receive traffic. For port name, a default will be assigned if left blank.

Host port	Container port	Protocol	Port name
3000	3000	TCP	container-port-protocol

App protocol
HTTP

Add port mapping

5. Set resource limits as following:
 - a. CPU: 0.300
 - b. GPU: empty
 - c. Memory hard limit: 0.400
 - d. Memory soft limit: 0.300

Resource allocation limits - conditional Info
Container-level CPU, GPU, and memory limits are different from task-level values. They define how much resources are allocated for the container. If container attempts to exceed the memory specified in hard limit, the container is terminated.

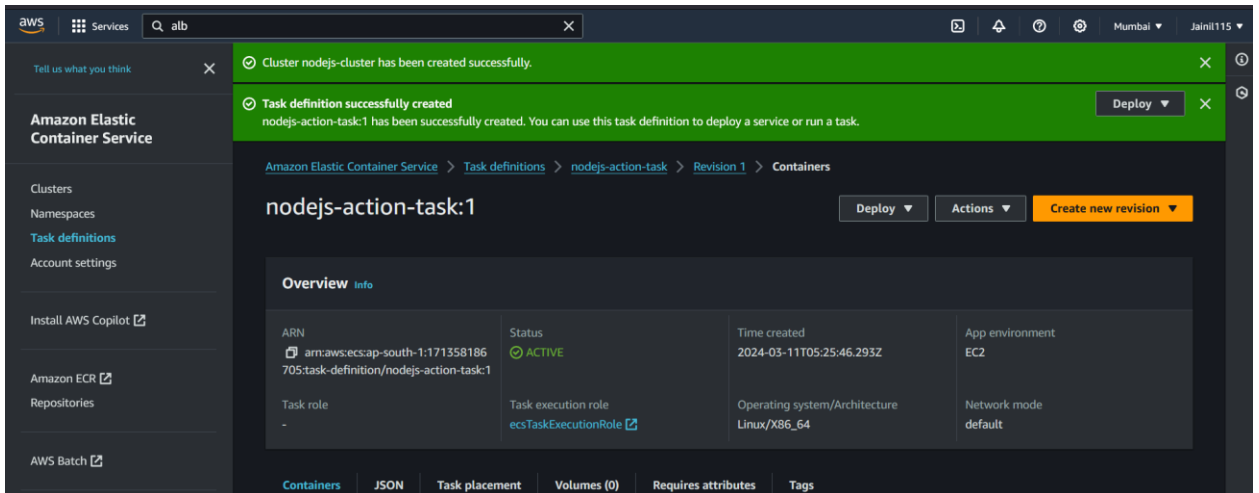
CPU	GPU	Memory hard limit	Memory soft limit
0.300 in vCPU	1	0.500 in GB	0.300 in GB

Environment variables - optional Info

6. Keep everything else default and click on create.

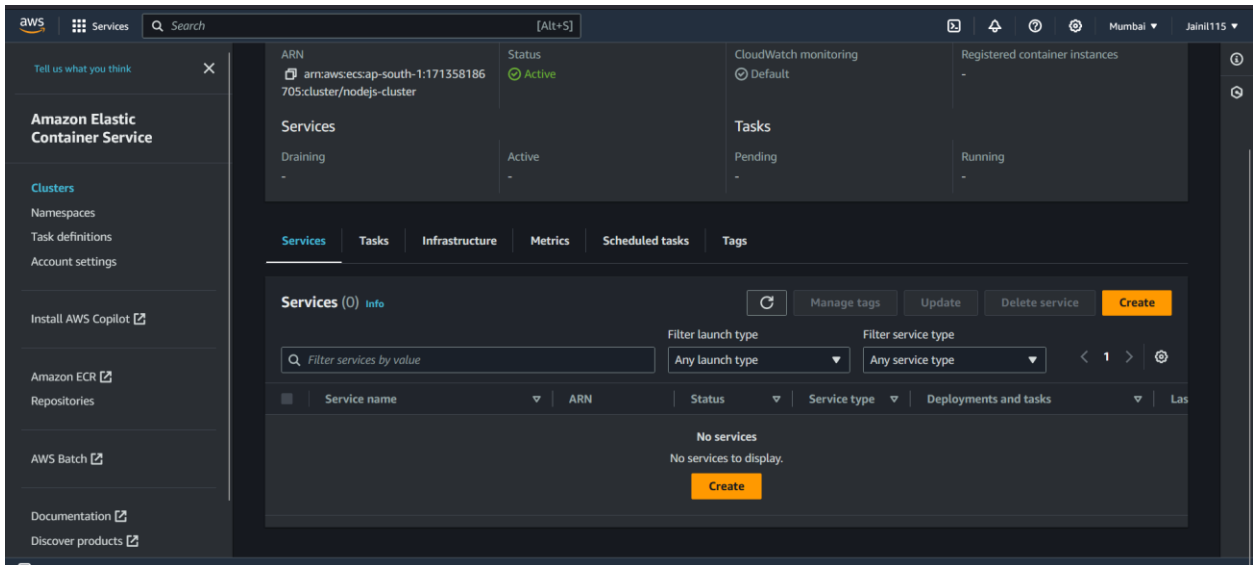
Tags - optional Info
Tags help you to identify and organize your task definitions.

Cancel Create

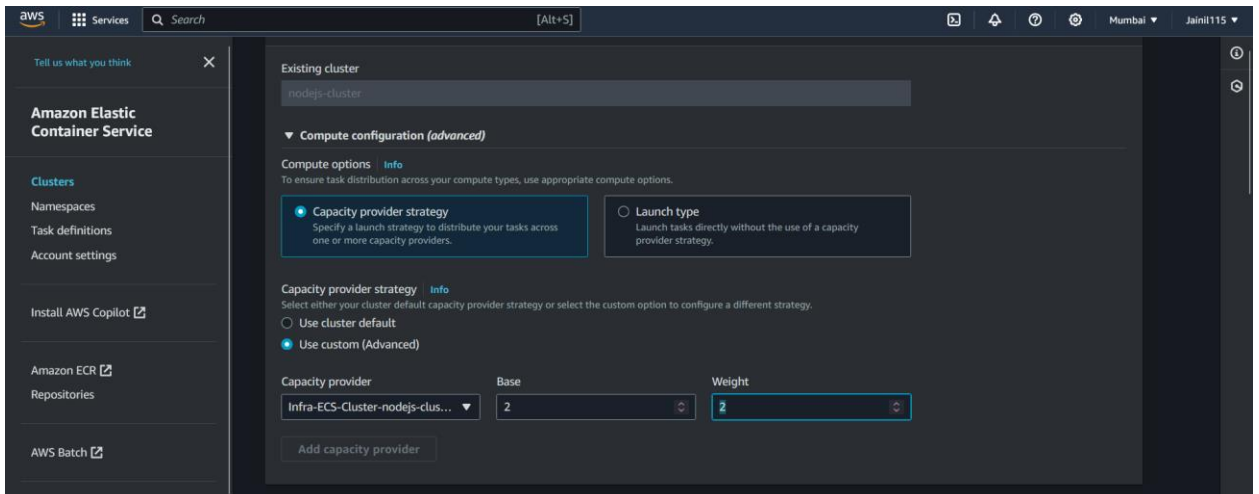


Now lets create service in ecs cluster

1. Go to nodejs-cluster and click on create in services tab

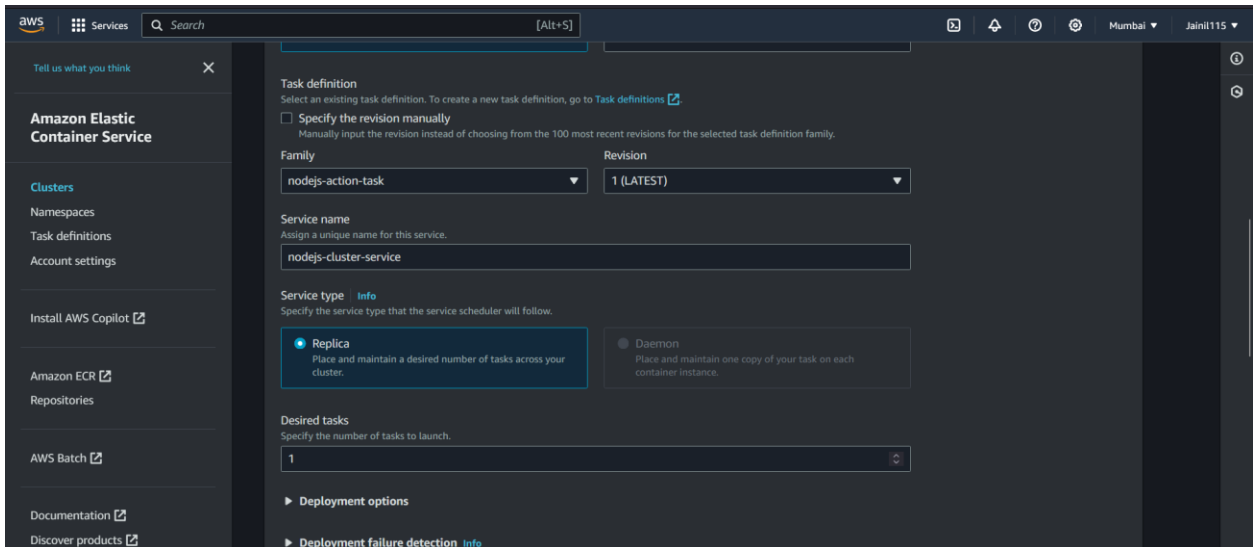
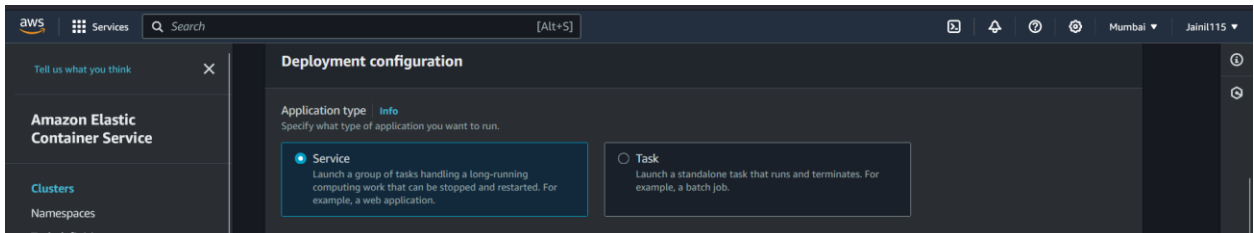


2. Select capacity provider strategy as Base: 2 and Weight: 2



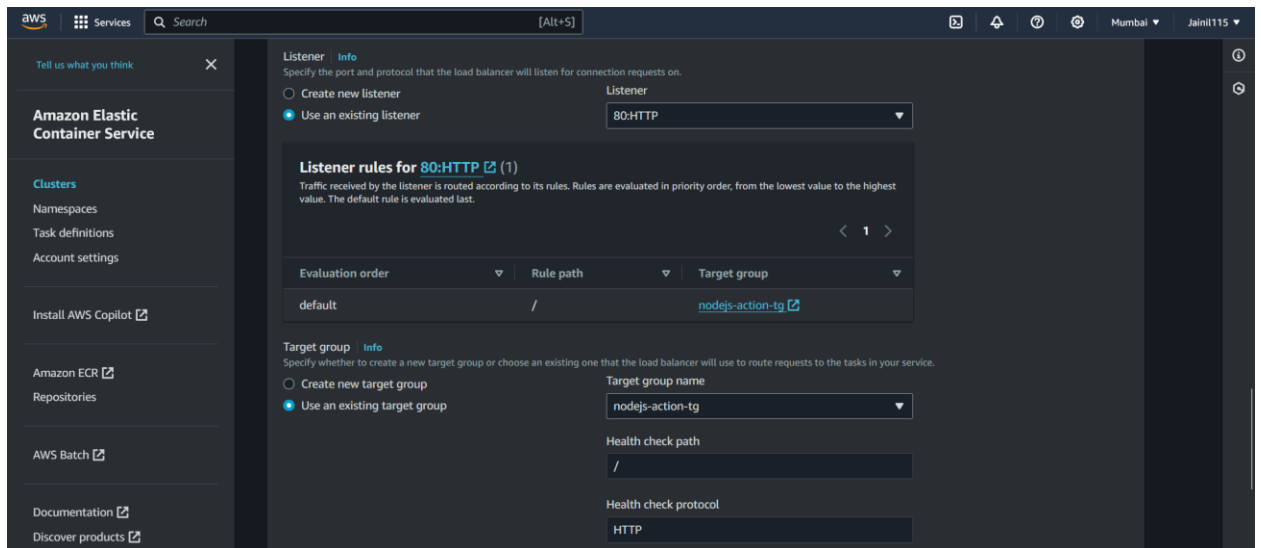
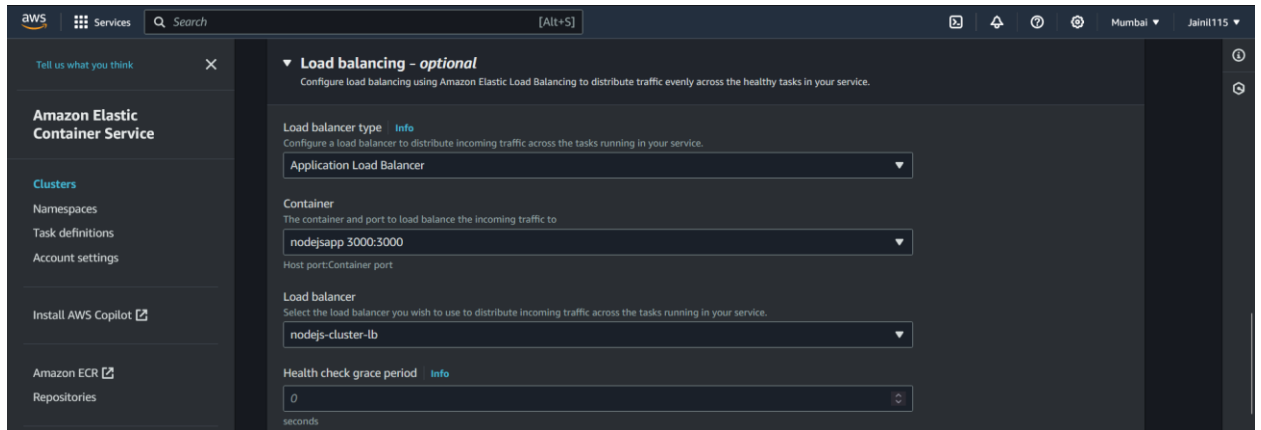
3. Now in deployment configuration select the following:

- a. Application Type: Service
- b. Family: nodejs-action-task
- c. Revision: Auto
- d. Service Name: nodejs-cluster-service
- e. Desired task: 1

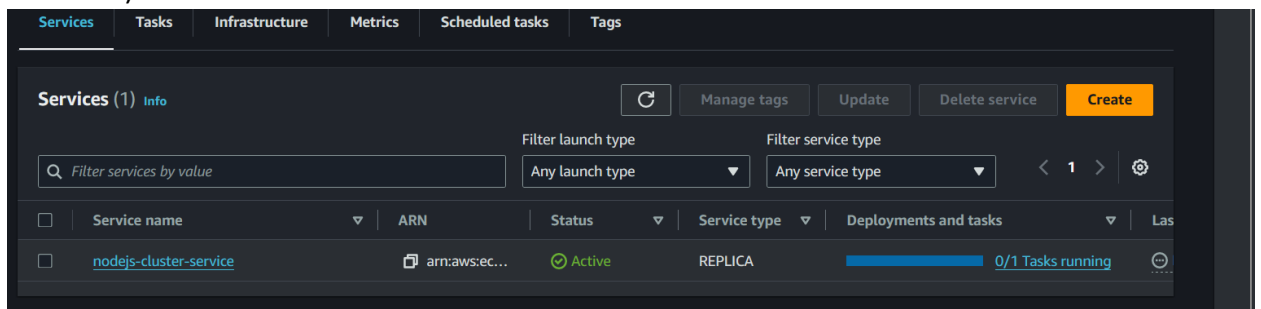


4. Now In load balancing select the following:

- a. Type: Application load balancer
- b. Container: nodejsapp 3000:3000
- c. Load balancer: nodejs-cluster-lb
- d. Listener: Use existing: 80:HTTP
- e. Target group: Use existing: Target group name: nodejs-action-tg



5. Then click on create. (This will fail, we don't have any image at this point, that is intended)



Create nodejs app in app folder:

1. Go inside app folder and do npm init
2. Then create index.js

index.js:

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

app.get("/", (req, res) => {
  res.send("Hello, world! From Jainil, Sahi me chal gaya???");
});

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

Now lets create Dockerfile

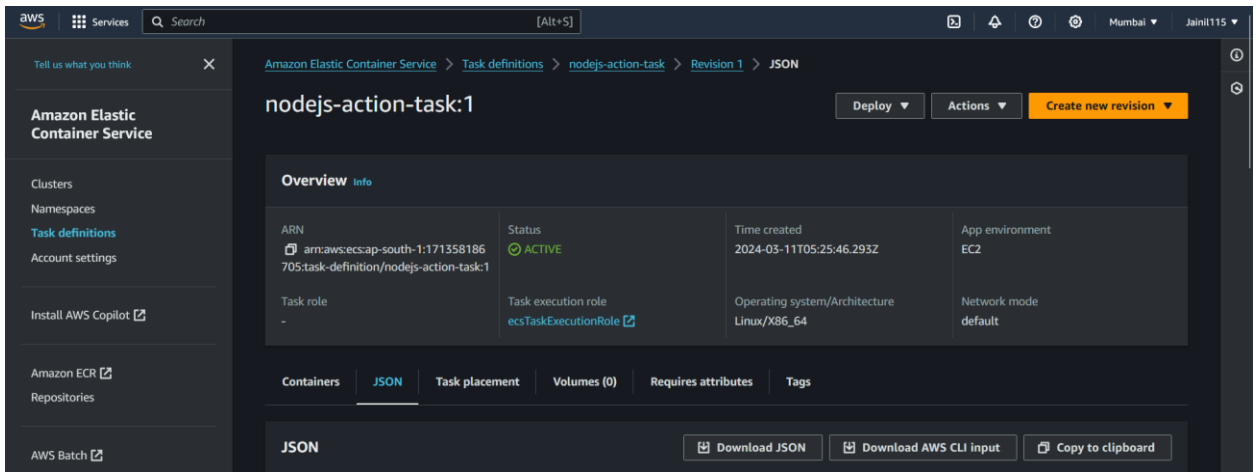
```
FROM node:21-alpine
WORKDIR /app
COPY ./app/package.json ./
RUN npm install
COPY ./app/* .
EXPOSE 3000
CMD [ "node", "index.js" ]
```

Now lets create .dockerignore:

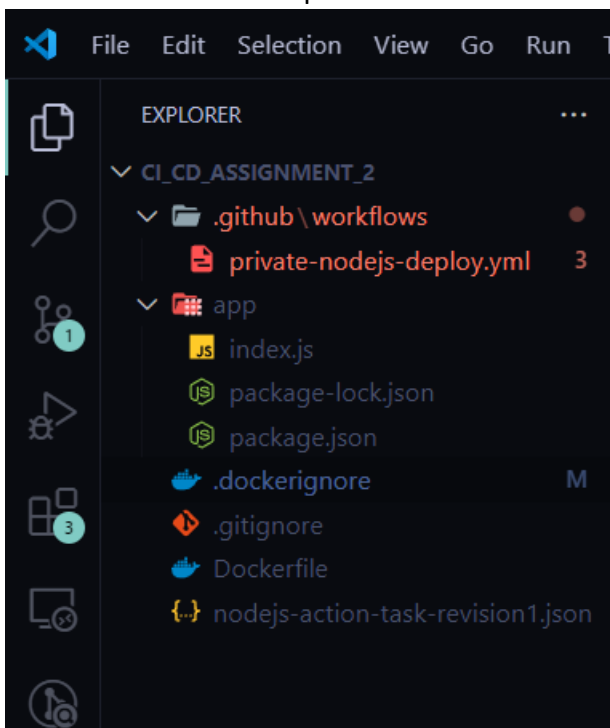
```
node_modules/
Assignment_1.pdf
/.git
/.github
.gitignore
nodejs-action-task-revision1.json
```

Now lets download the nodejs-action-task-revision1.json

1. Go to task definition in the ecs dashboard



2. Download this file and place it in the root folder.



Now lets create github workflow

1. Create folder .github and inside that create folder workflows and then create a file name private-nodejs-deploy.yml

```
name: Deploy to Amazon ECS

on:
  push:
    branches: ["master"]
  pull_request:
```

```

    branches: ["master"]

env:
  AWS_REGION: ap-south-1
  ECR_REPOSITORY: "nodejsapp"
  ECS_SERVICE: "nodejs-cluster-service"
  ECS_CLUSTER: "nodejs-cluster"
  ECS_TASK_DEFINITION: nodejs-action-task-revision1.json
  CONTAINER_NAME: nodejsapp

jobs:
  deploy:
    name: Deploy
    runs-on: ubuntu-latest
    environment: production

    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v4
        with:
          aws-access-key-id: ${ secrets.AWS_ACCESS_KEY_ID }
          aws-secret-access-key: ${ secrets.AWS_SECRET_ACCESS_KEY }
        }}

          aws-region: ${ env.AWS_REGION }}

      - name: Login to Amazon ECR
        id: login-ecr
        uses: aws-actions/amazon-ecr-login@v2

      - name: Build, tag, and push image to Amazon ECR
        id: build-image
        env:
          ECR_REGISTRY: ${ steps.login-ecr.outputs.registry }
          IMAGE_TAG: ${ github.sha }
        run: |
          docker build -t $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG .
          docker push $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG
          echo "image=$ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG" >>
$GITHUB_OUTPUT

      - name: Fill in the new image ID in the Amazon ECS task
definition

```



```

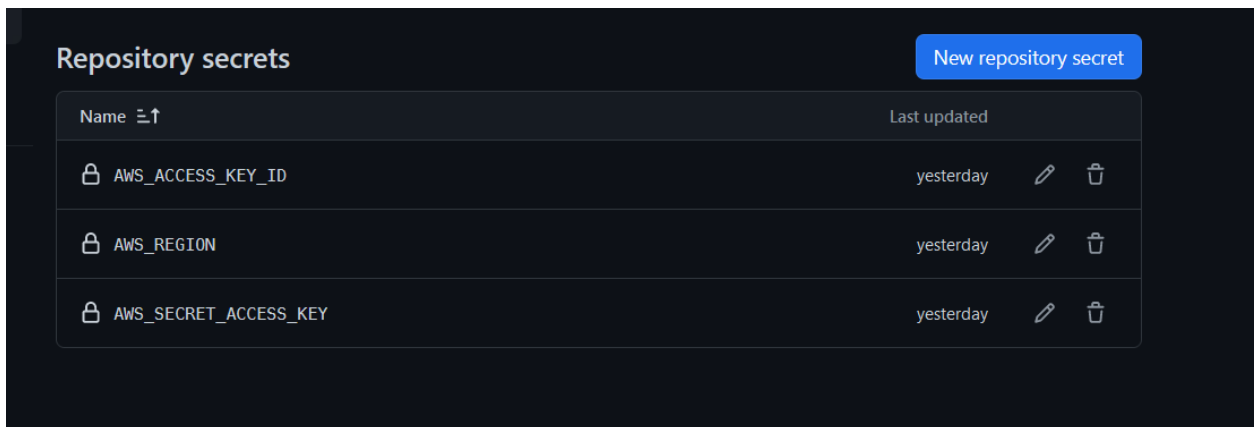
    id: task-def
    uses: aws-actions/amazon-ecs-render-task-definition@v1
    with:
      task-definition: ${{ env.ECS_TASK_DEFINITION }}
      container-name: ${{ env.CONTAINER_NAME }}
      image: ${{ steps.build-image.outputs.image }}

  - name: Deploy Amazon ECS task definition
    uses: aws-actions/amazon-ecs-deploy-task-definition@v1
    with:
      task-definition: ${{ steps.task-def.outputs.task-
definition }}
      service: ${{ env.ECS_SERVICE }}
      cluster: ${{ env.ECS_CLUSTER }}
      wait-for-service-stability: true

```

Create a repository and enter the secrets in the github repository of aws for the IAM user.

1. AWS_SECRET_KEY_ID
2. AWS_SECRET_ACCESS_KEY



Now lets push this to github.

Push using command:

```
git push origin master
```

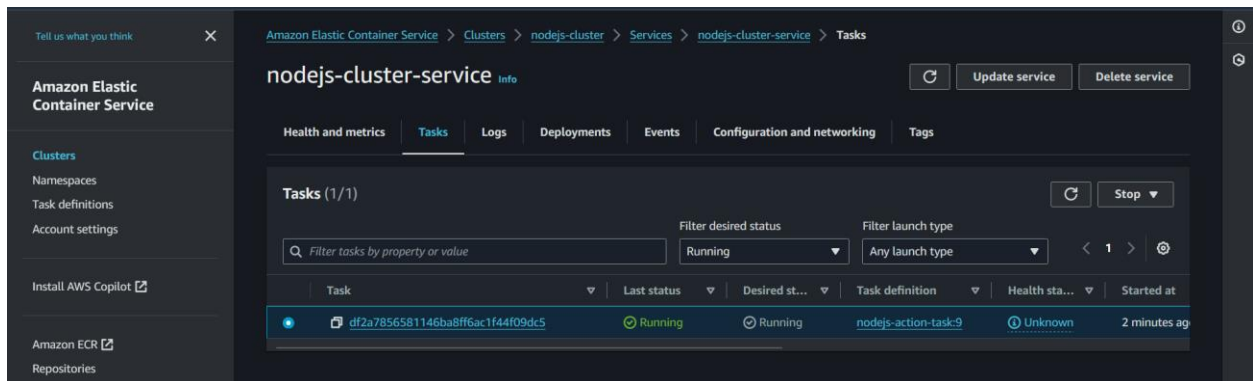
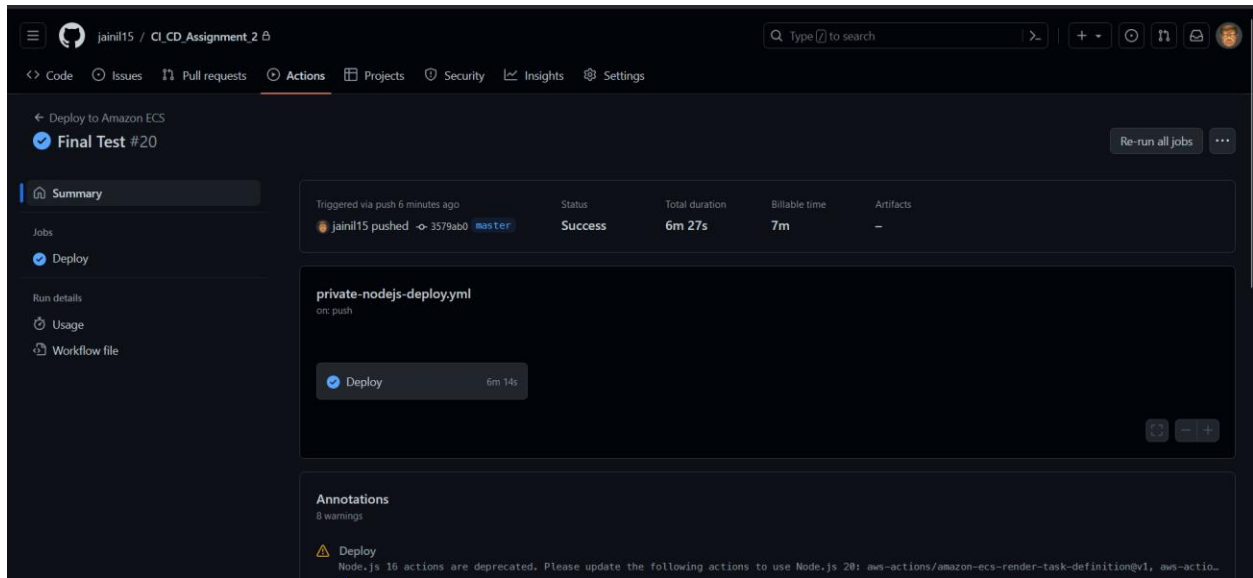
```
> git add .
• CI_CD_Assignment_2 master # 57ms pwsh 67% default@ap-south-1 AzureCloud
• CI_CD_Assignment_2 master # ~1 48ms pwsh 68% default@ap-south-1 AzureCloud
• git commit -am "Final Test"
[master 3579ab0] Final Test
1 file changed, 1 insertion(+), 1 deletion(-)
• CI_CD_Assignment_2 master # 96ms pwsh 68% default@ap-south-1 AzureCloud
• git push origin master
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 12 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 411 bytes | 411.00 KiB/s, done.
Total 4 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/jainil15/CI_CD_Assignment_2.git
73ec451..3579ab0 master -> master
• CI_CD_Assignment_2 master # 6s 205ms pwsh 71% default@ap-south-1 AzureCloud
minikube :: default 15:20:29
```

After pushing:

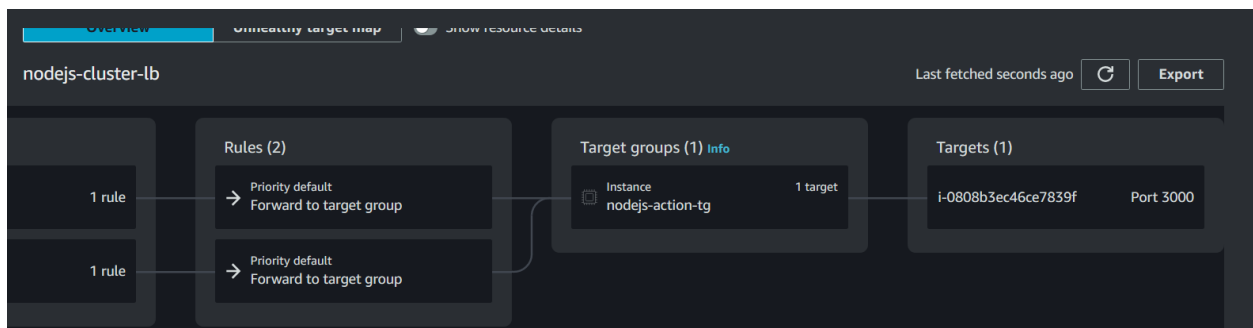
The screenshot shows the GitHub Actions interface for the workflow "Deploy to Amazon ECS". The job "Final Test #20" is currently running. The left sidebar shows the workflow steps: Summary, Jobs, Deploy (selected), Run details, Usage, and Workflow file. The main area displays the "Deploy" job, which started 16s ago. The job steps are: Set up job (3s), Checkout (1s), Configure AWS credentials (1s), Login to Amazon ECR (4s), and Build, tag, and push image to Amazon ECR (8s). The current step is "Build, tag, and push image to Amazon ECR", which is running on a "default" instance using the "docker" driver. The command being executed is "Run docker build -t \$ECR_REGISTRY/\$ECR_REPOSITORY:\$IMAGE_TAG .".

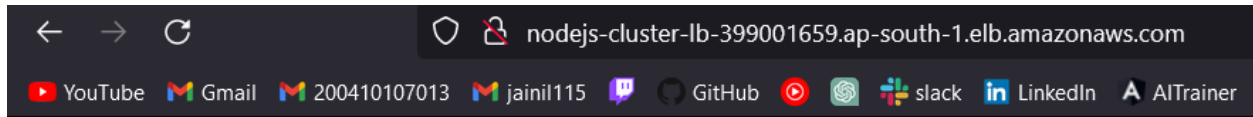
The screenshot shows the AWS Management Console for the "nodes-cluster-service" in the "Amazon Elastic Container Service" (ECS) console. The service is in the "Active" state. The "Status" section shows the ARN "arn:aws:ecs:ap-south-1:171358186705:service/nodes-cluster/nodes-cluster-service" and the status "Active". The "Tasks (1 Desired)" section shows a progress bar with "1 Pending" and "1 Running" tasks. The "Deployments current state" section shows "1 in Progress" and "0 Completed" deployments. The "Load balancer health" section shows the "Application Load Balancer" "nodes-cluster-lb" with "1 Healthy" and "0 Unhealthy" targets. The "Listener protocol/port" is "HTTP-3000" and "HTTP-80". The "Target group name: protocol" is "nodes-action-tg:HTTP". The "Health check path" is "/".

After workflow complete:



Check the application using load balancers url (nodejs-cluster-lb-399001659.ap-south-1.elb.amazonaws.com):





Hello, world! From Jainil, Sahi me chal gaya??? WOOO!!!

Lets check the ecr image:

