Assignment 2

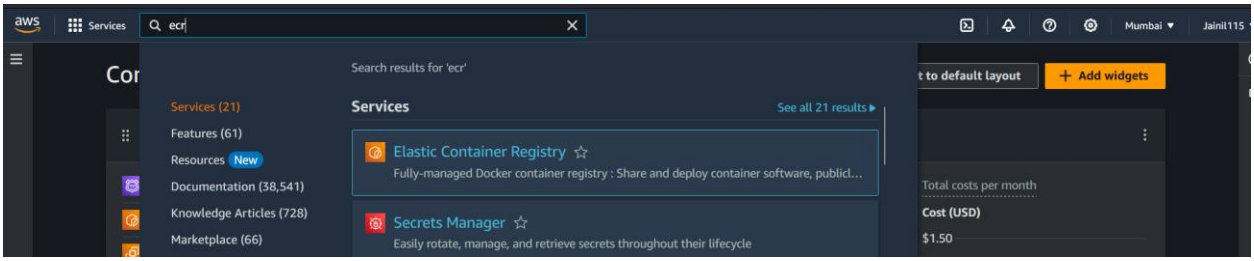
Objective: Create a GitHub Actions workflow that automates the building and deployment of a Node.js application to Amazon ECS using Amazon ECR.

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

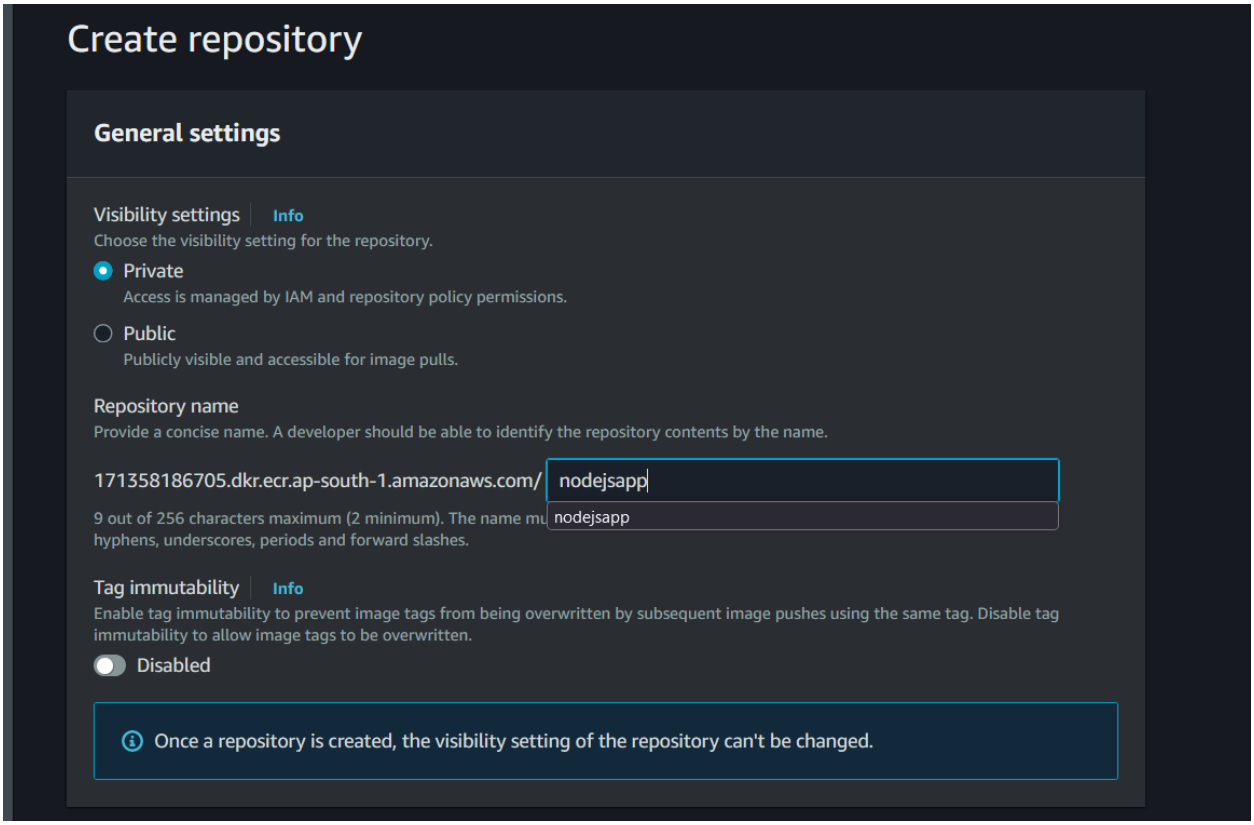
Steps to create ECR (Elastic Container Registry).

1. Go to aws console and search for ecr.

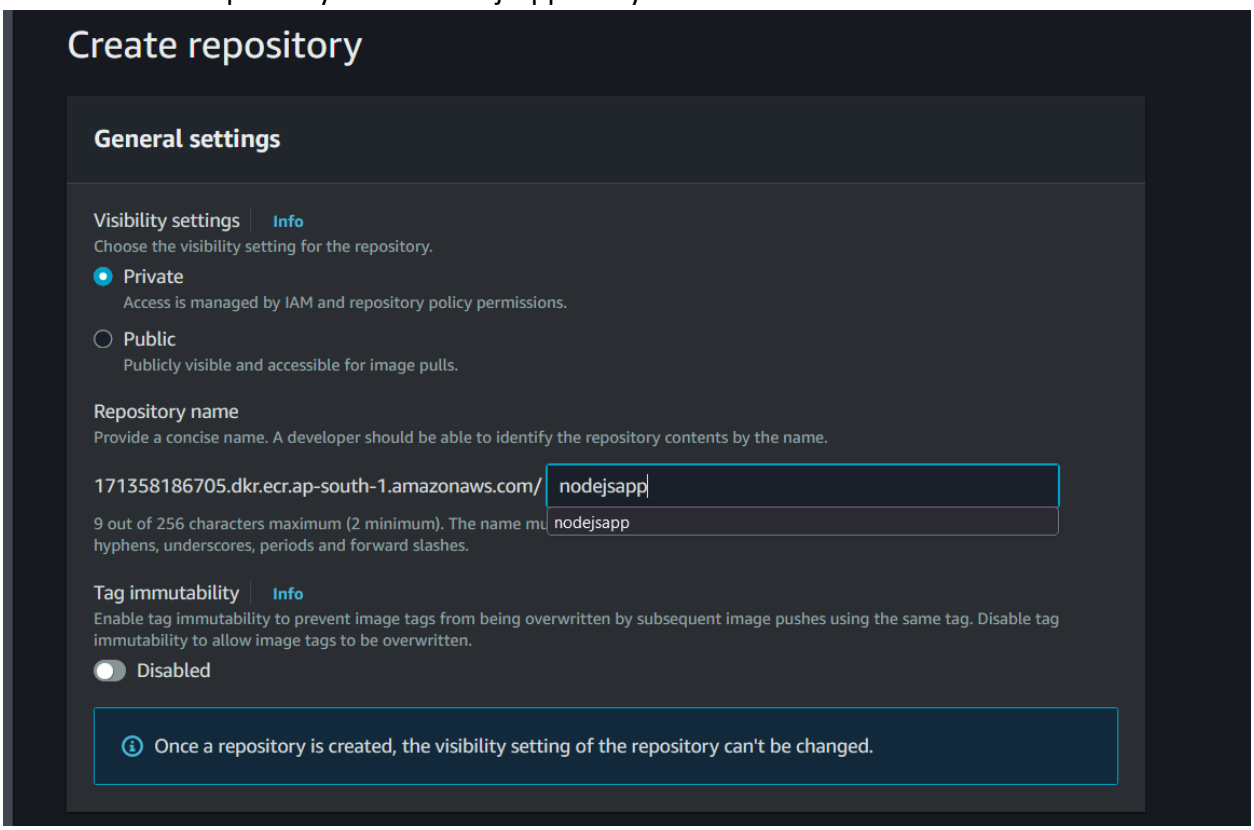


2. Then click on Get Started in create repository.

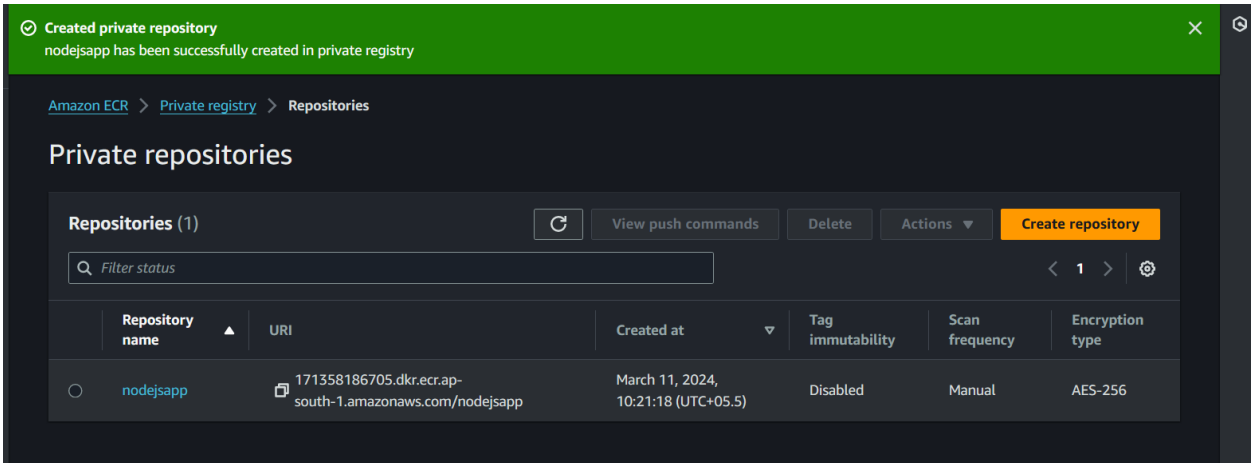
3. Change the visibility setting to private for this assignment.



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

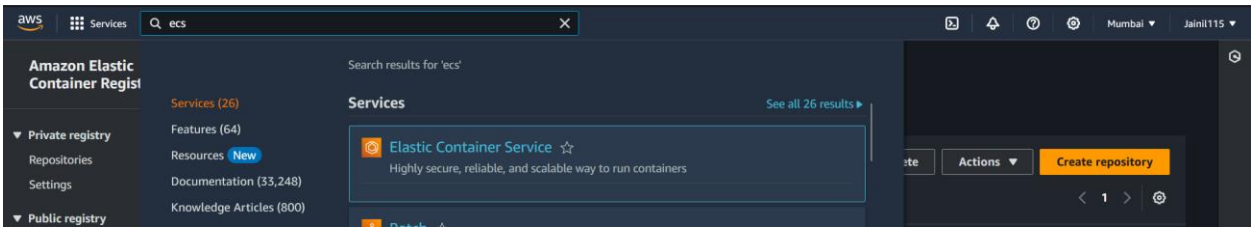


5. Then click on create repository.

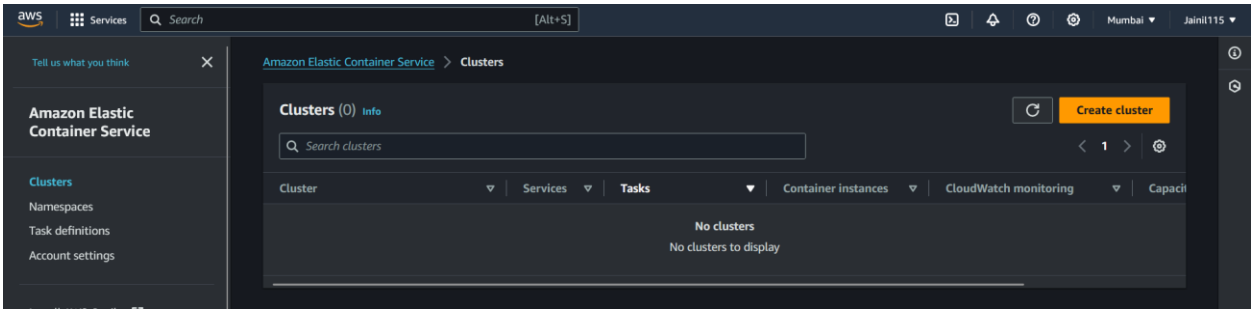


Let's now setup ecs.

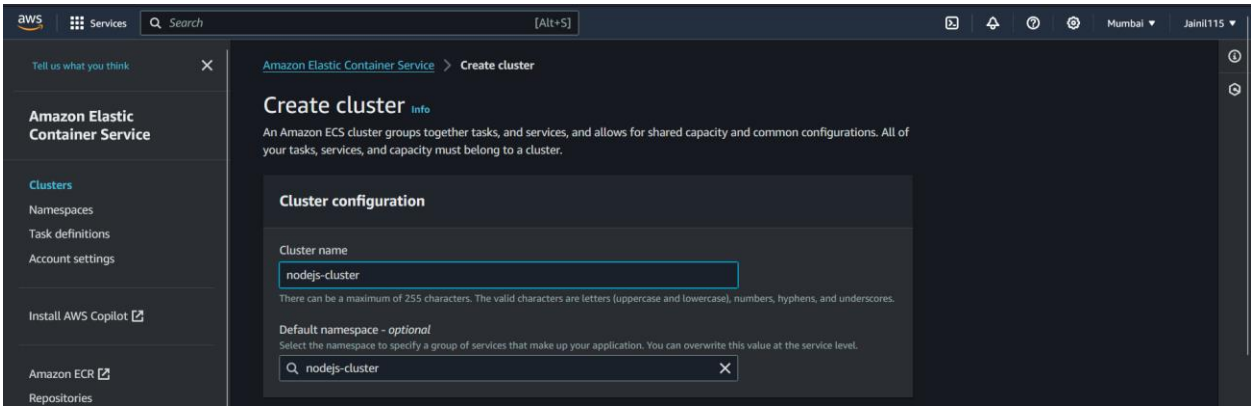
1. Search for ecs in aws console.



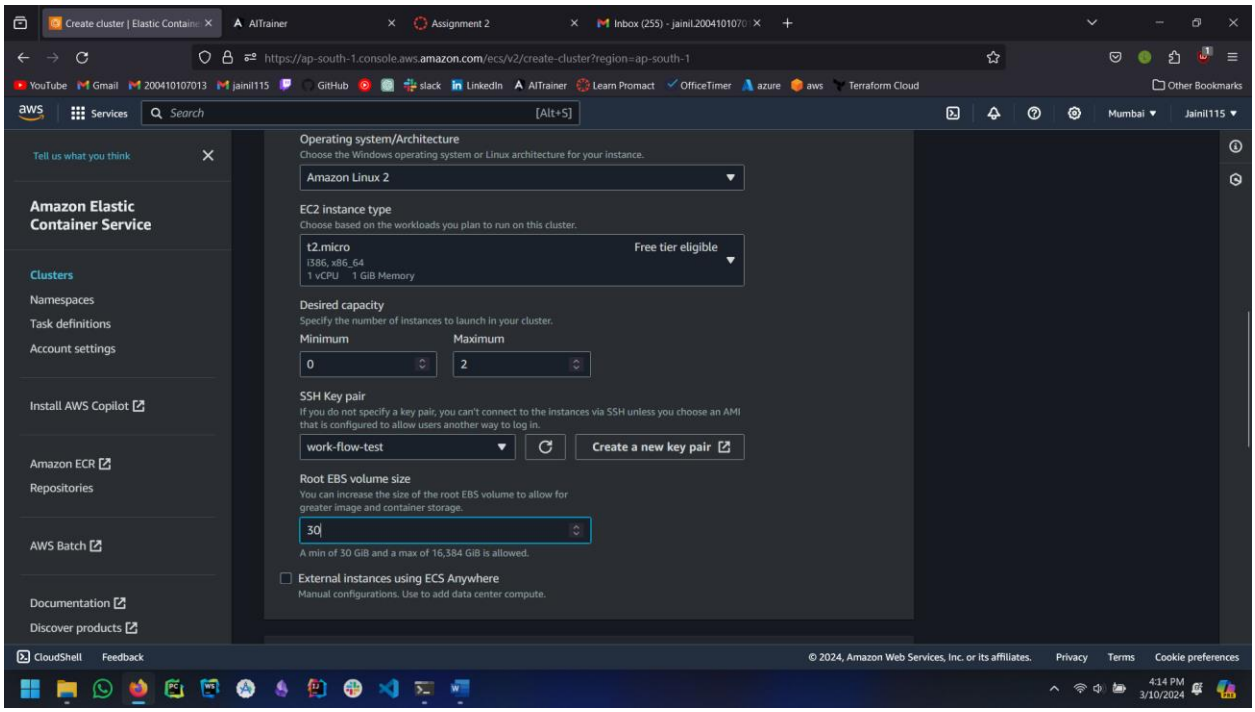
2. Now click on create cluster.



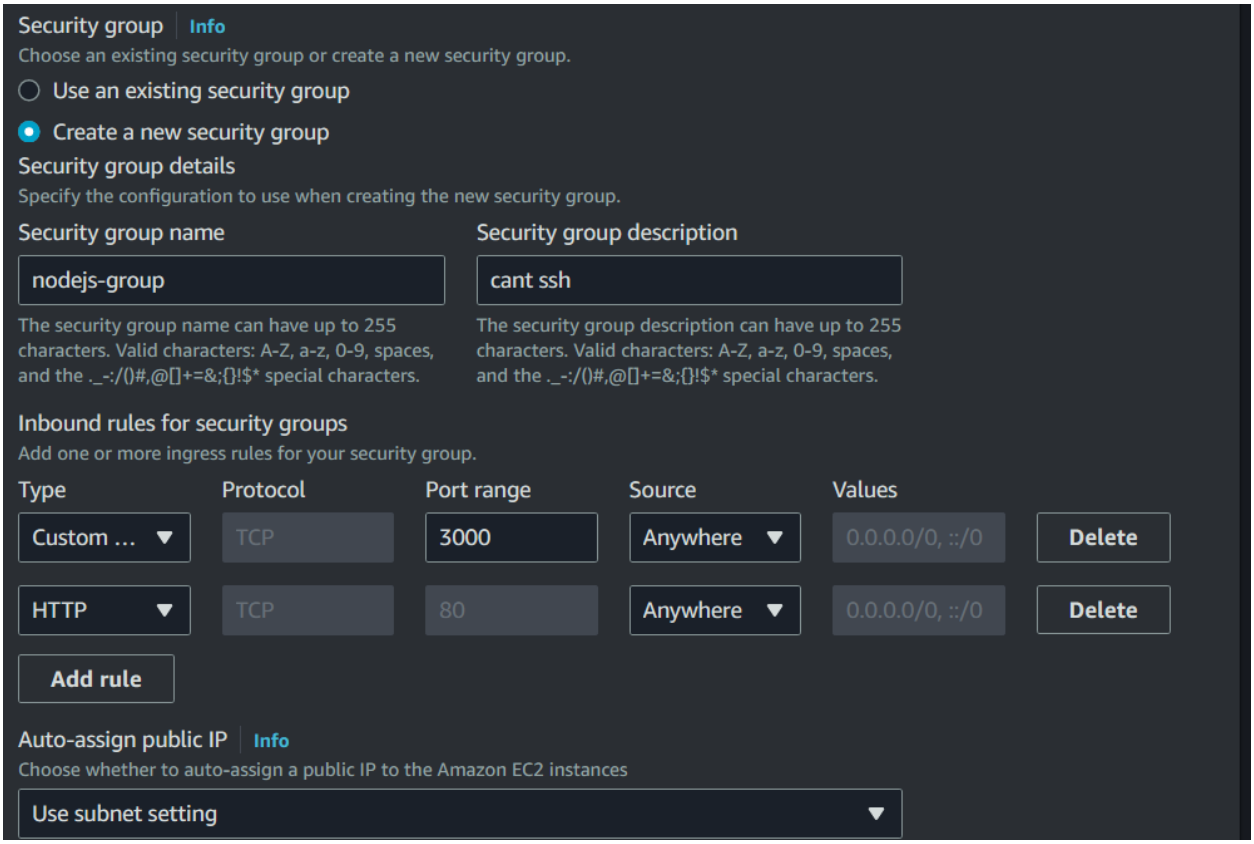
3. Now enter the name of the cluster. In my case it is nodejs-cluster.



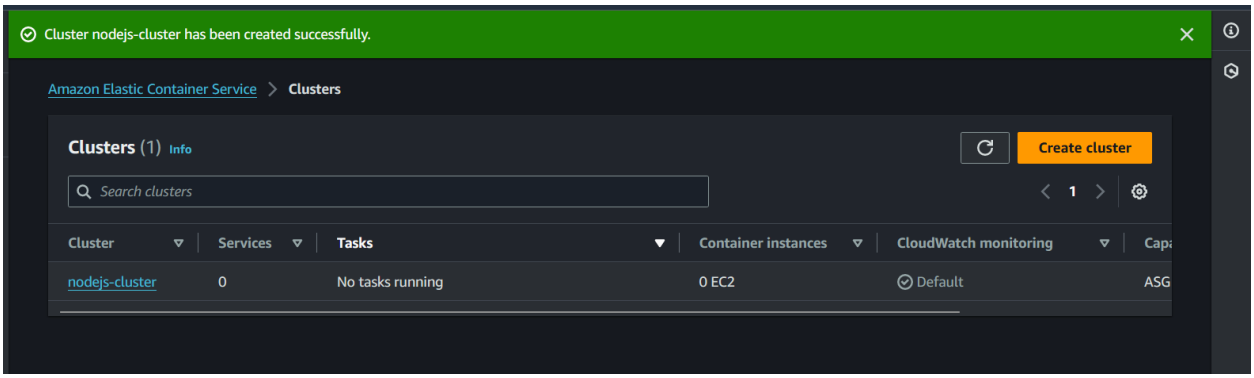
4. Now fill the following details as follows:
    a. Select EC2 Infrastructure
    b. Select Auto Scaling Group
    c. Provisioning model: On demand
    d. Operating system: Amazon Linux 2
    e. EC2 instance type: t2.micro
    f. Desired capacity: Min: 0, Max:2
    g. SSH Key Pair: select your ssh key 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:



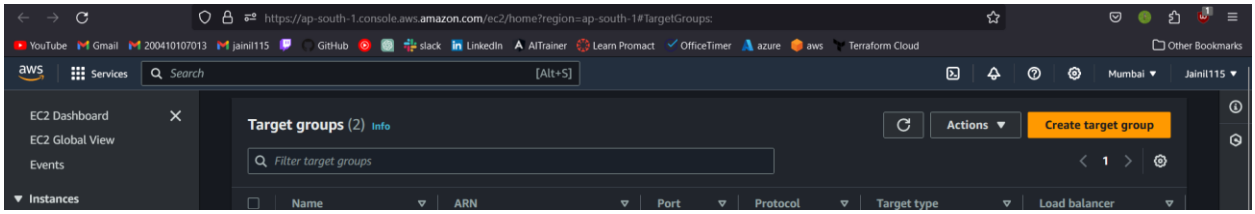
6. Leave everything as default and click on create ECS.



Now lets create load balancer for the ASG.

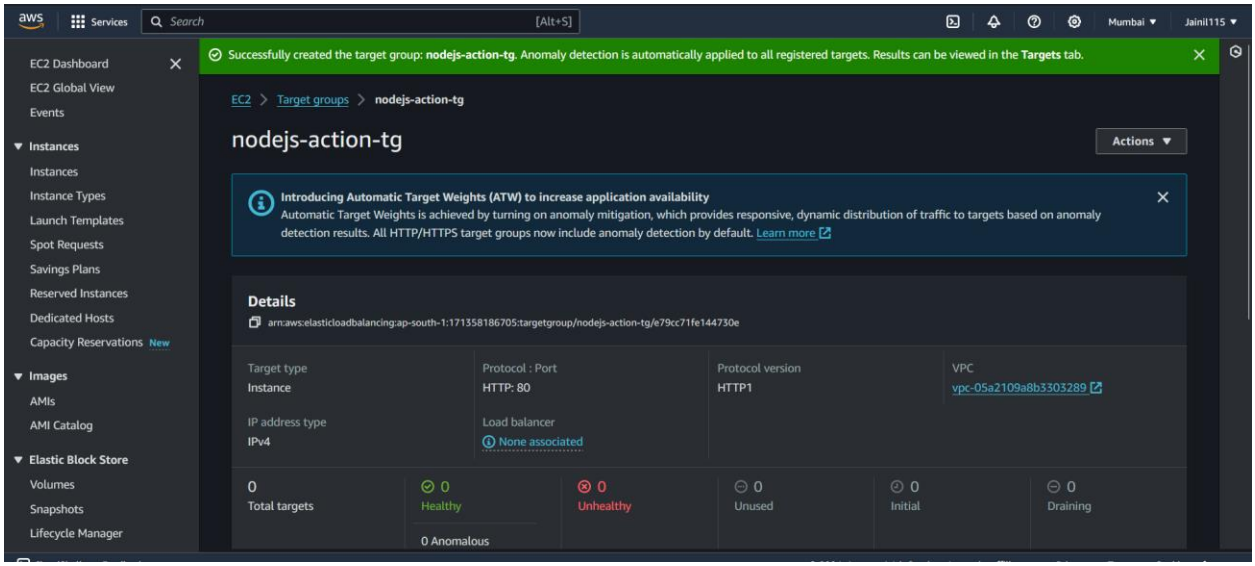
For that lets first create target group:

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



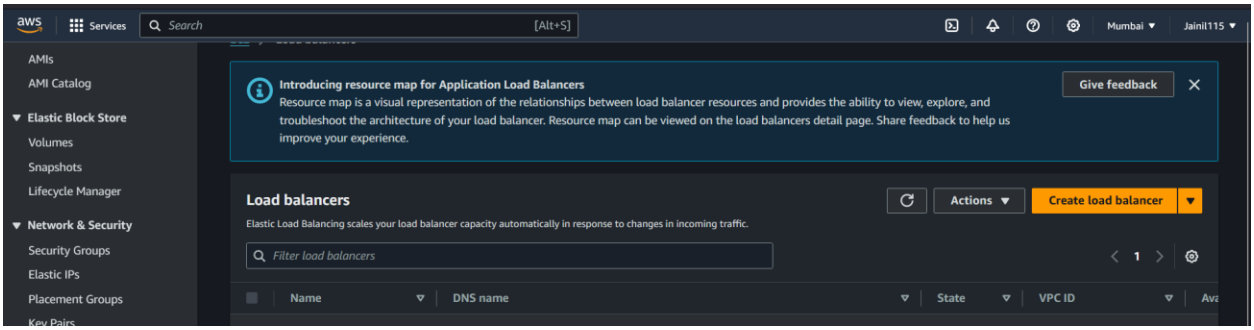
2. Select target type as application load balancer

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

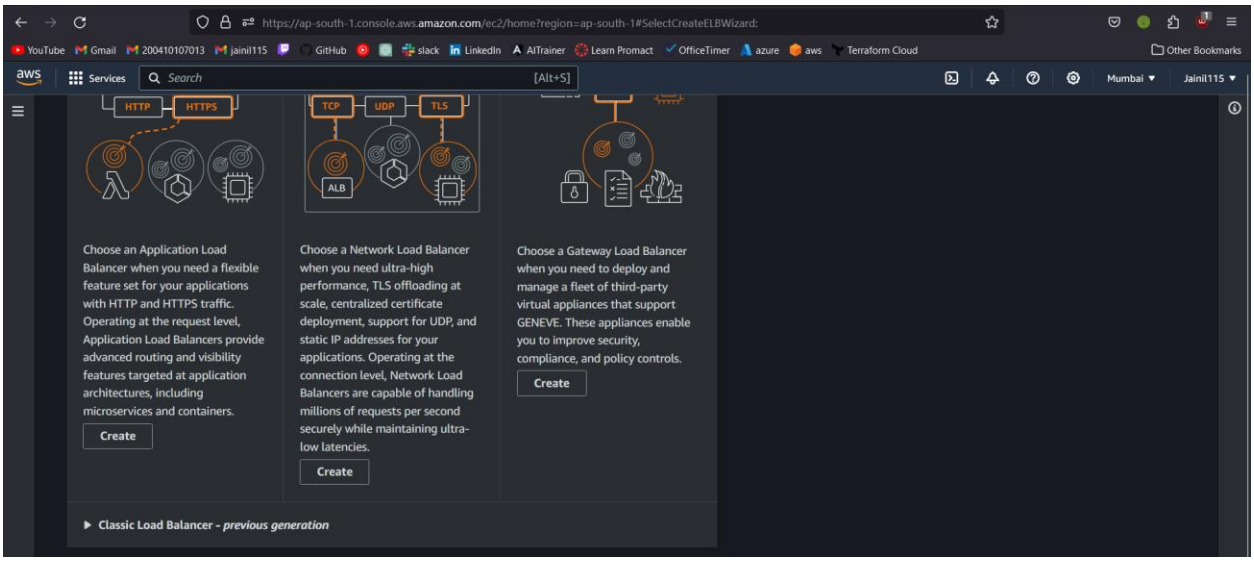


Now lets create that load balancer.

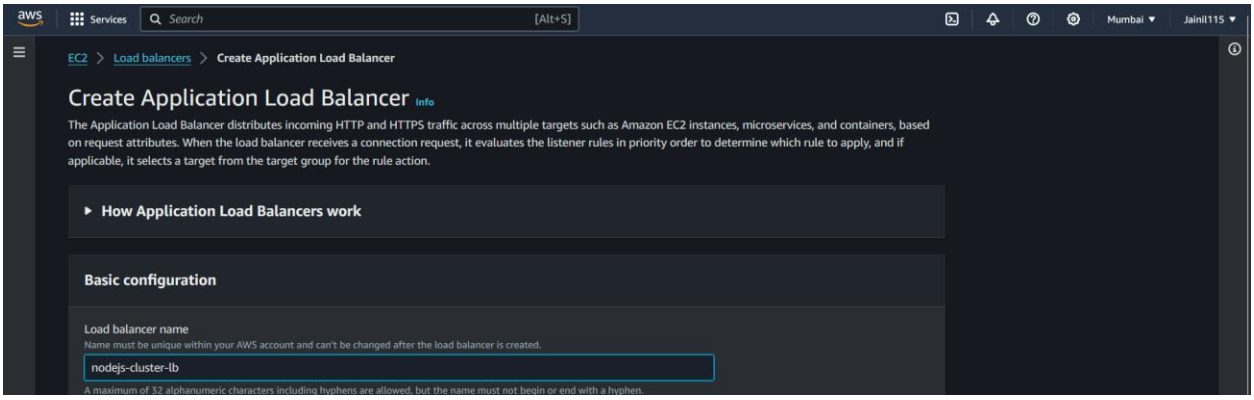
1. Go to load balancer tab and click on create load balancer



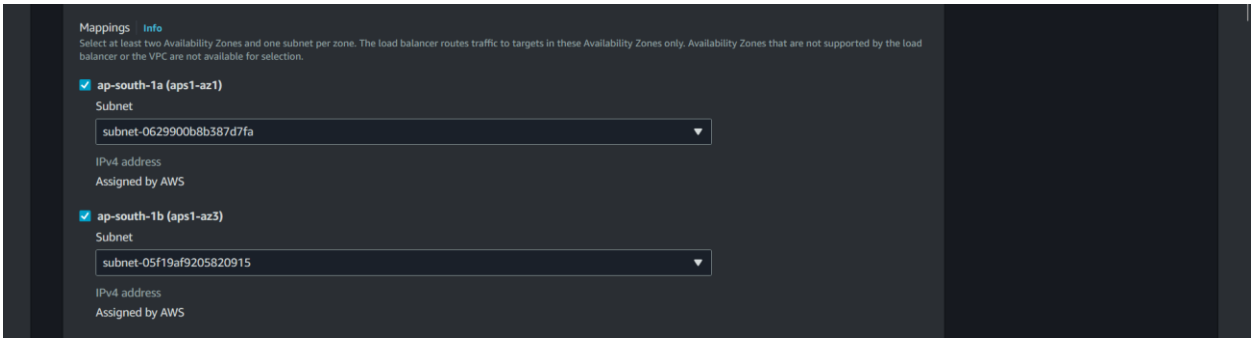
2. Select application load balancer



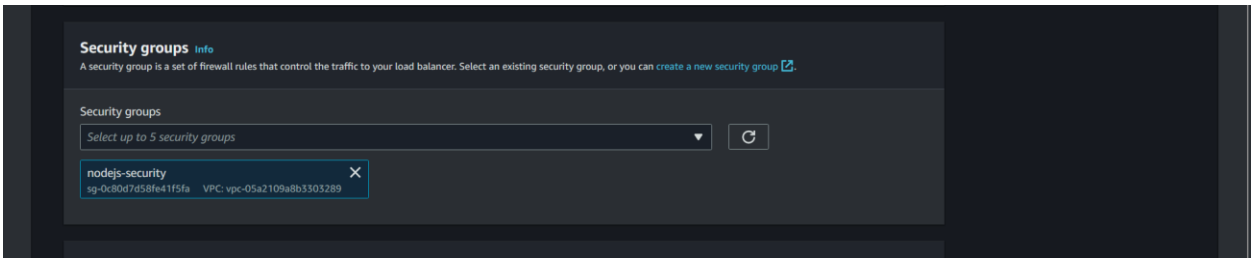
3. Enter the name of the load balancer as nodejs-cluster-lb



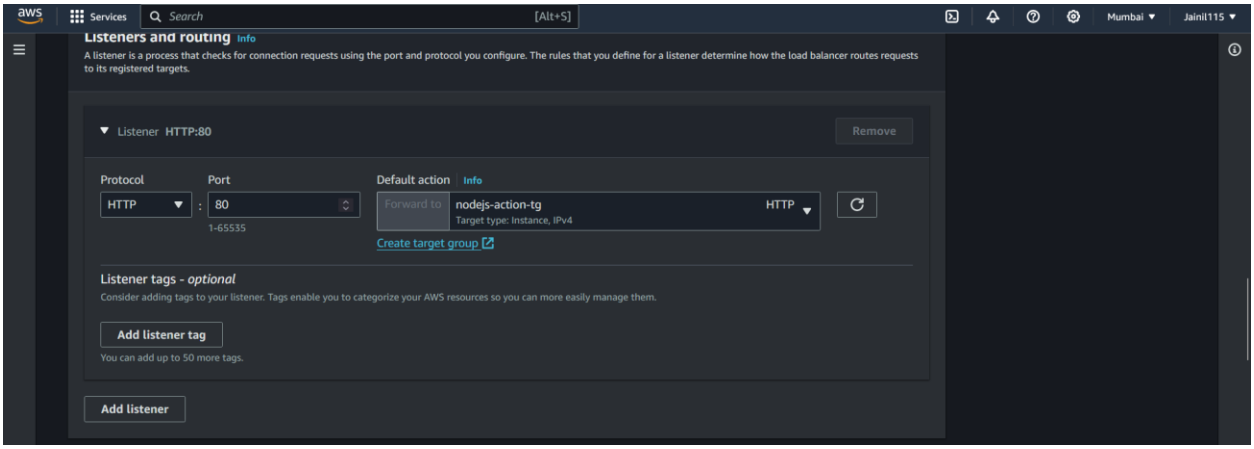
4. In network mapping select ap-south-1a and ap-south-1b in mapping and keep everything in network default



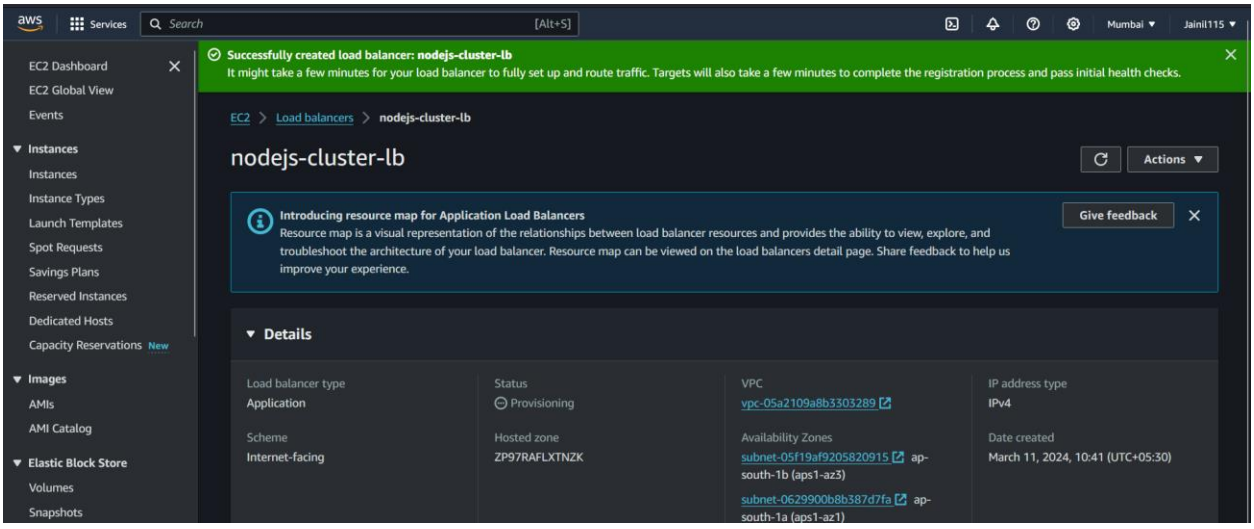
5. In security group select nodejs-security



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

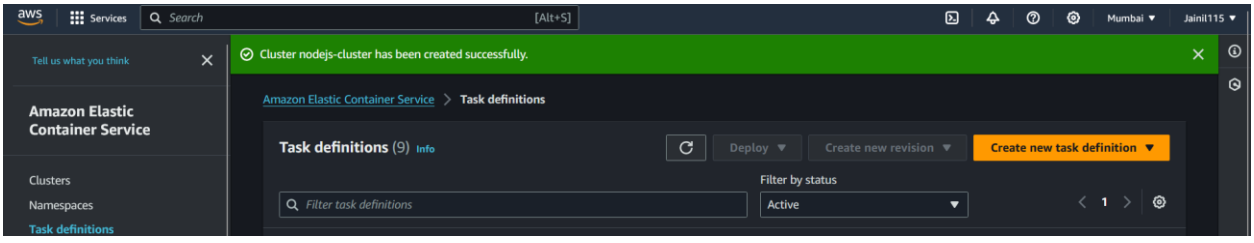


7. Then click on create load balancer.

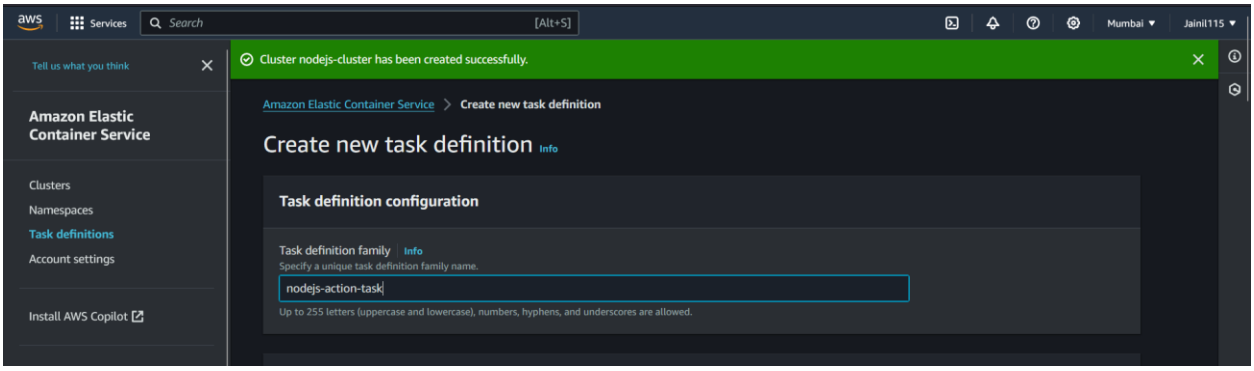


Now lets create task definition:

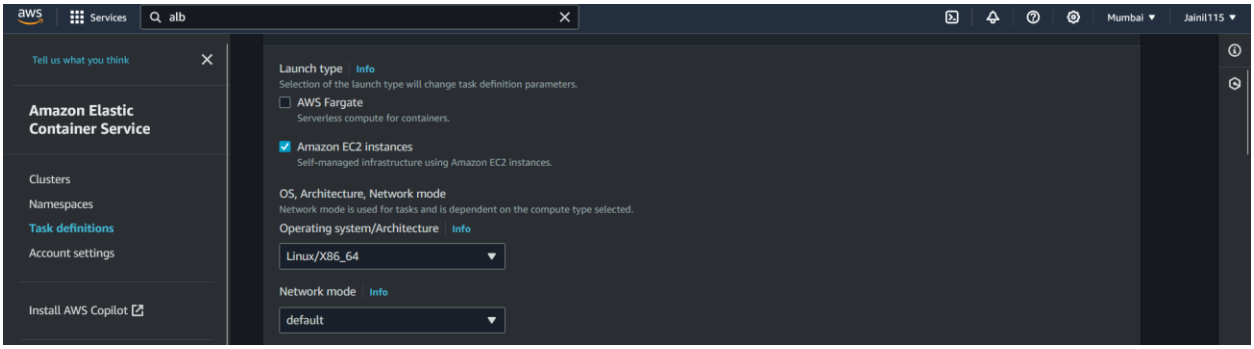
1. Go to task definition and click on create new task definition

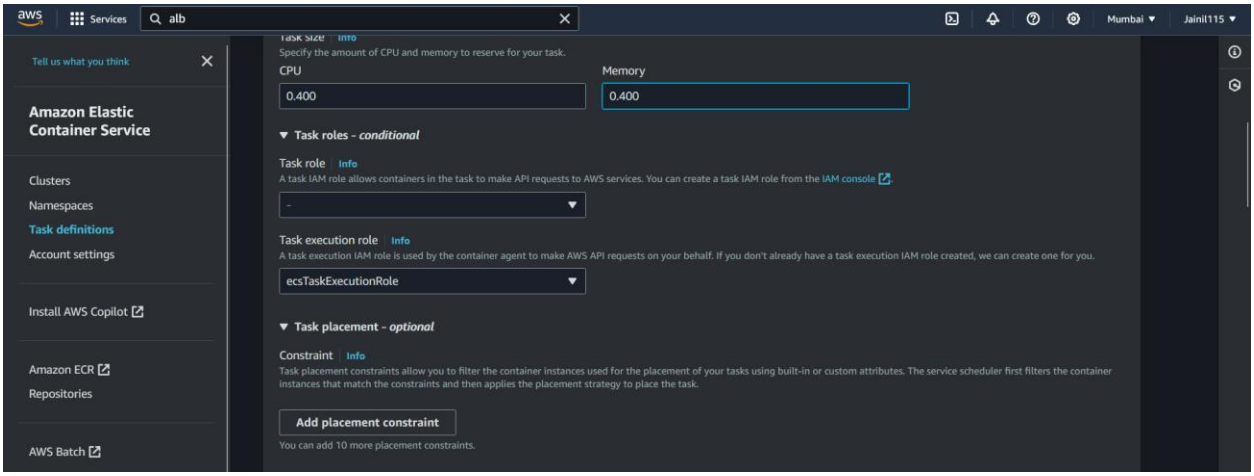


2. Give it name nodejs-action-task



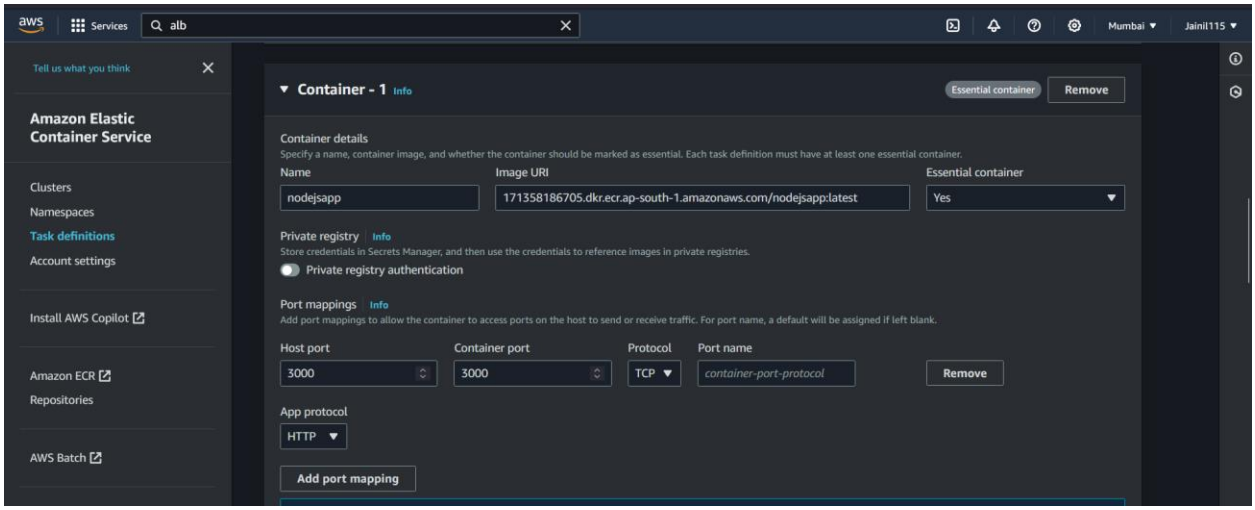
3. Select the following infrastructure requirements
   Launch type: EC2
   OS: Linux_X86_64
   Network mode: default
   Task Size: 0.400 vcpu and 0.400 MB memory
   Task Role: -
   Task execution Role: ecsTaskExecutionRole



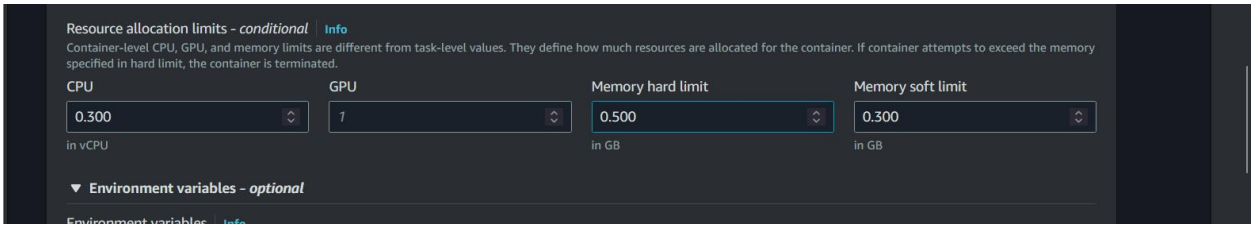


4. Now in container details enter the following details:
   a. Name: nodejsapp
   b. Image URI: 171358186705.dkr.ecr.ap-south-1.amazonaws.com/nodejsapp:latest

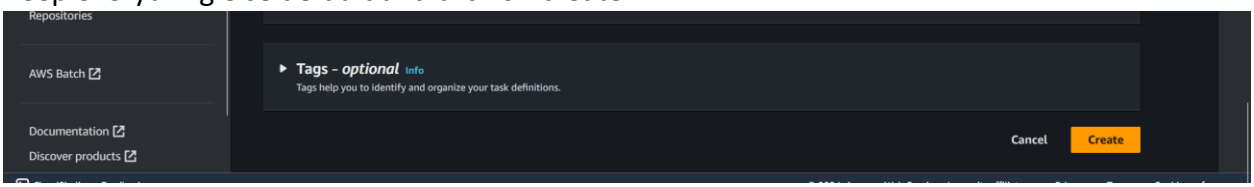
c. Essential container: yes
d. Host port: 3000
e. Container port: 3000
f. Protocol: TCP
g. portname: empty
h. App protocol: HTTP

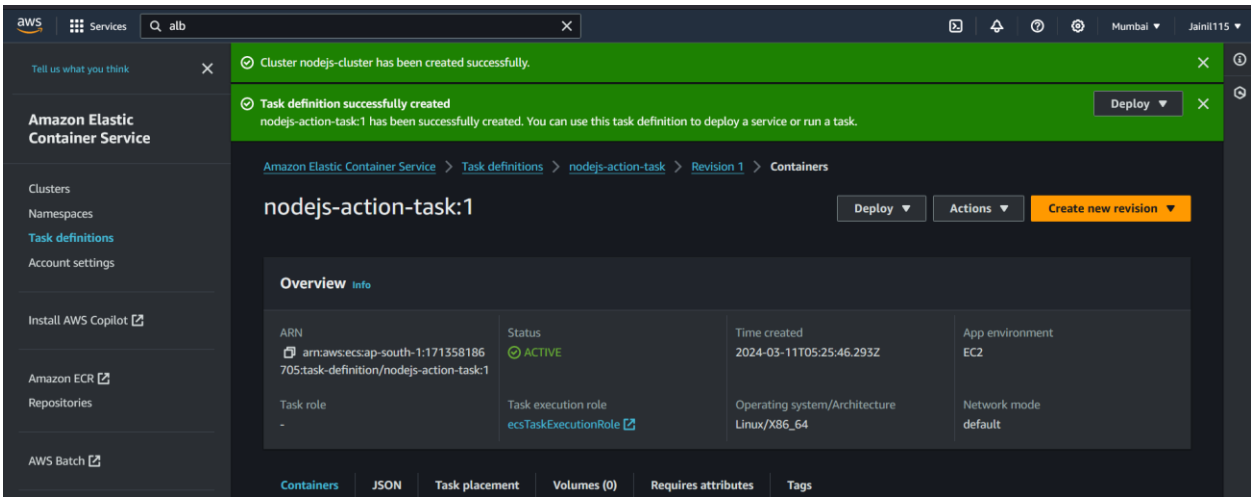


5. Set resource limits as following:
    a. CPU: 0.300
    b. GPU: empty
    c. Memory hard limit: 0.400
    d. Memory soft limit: 0.300



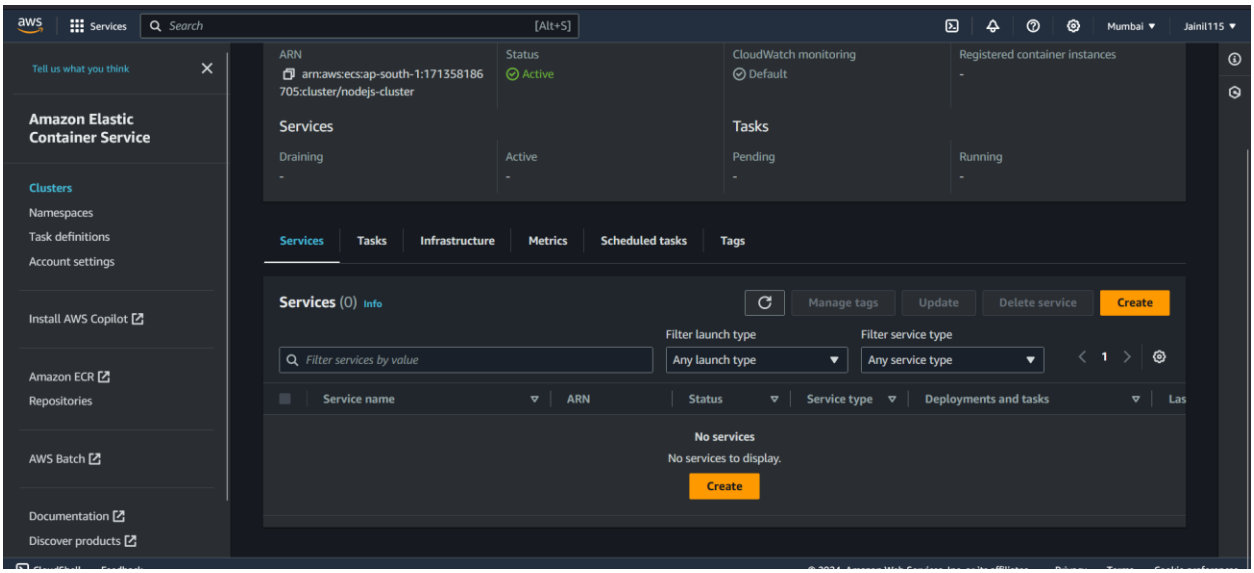
6. Keep everything else default and click on create.



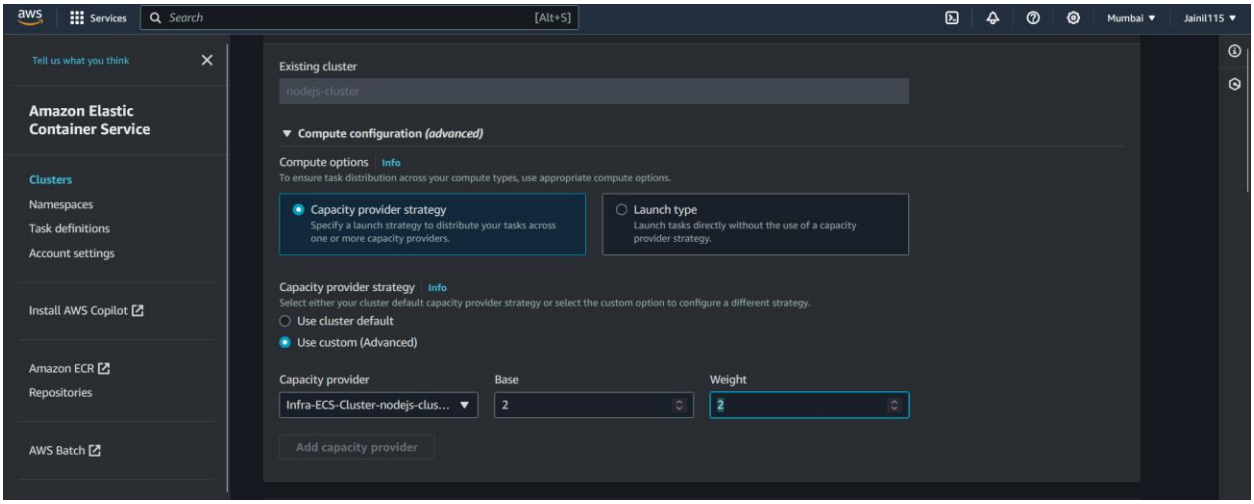


Now lets create service in ecs cluster

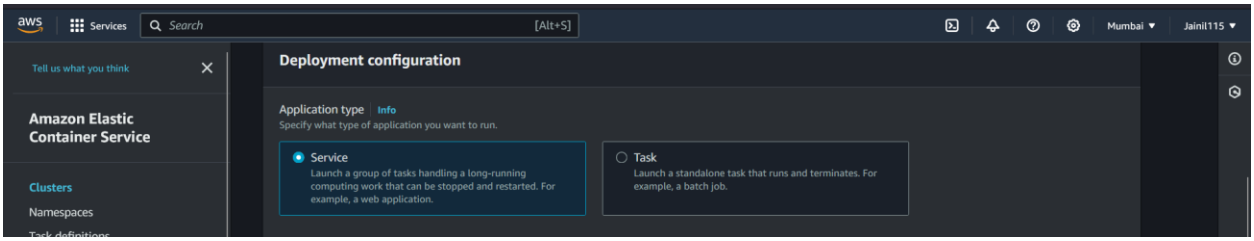
1. Go to nodejs-cluster and click on create in services tab

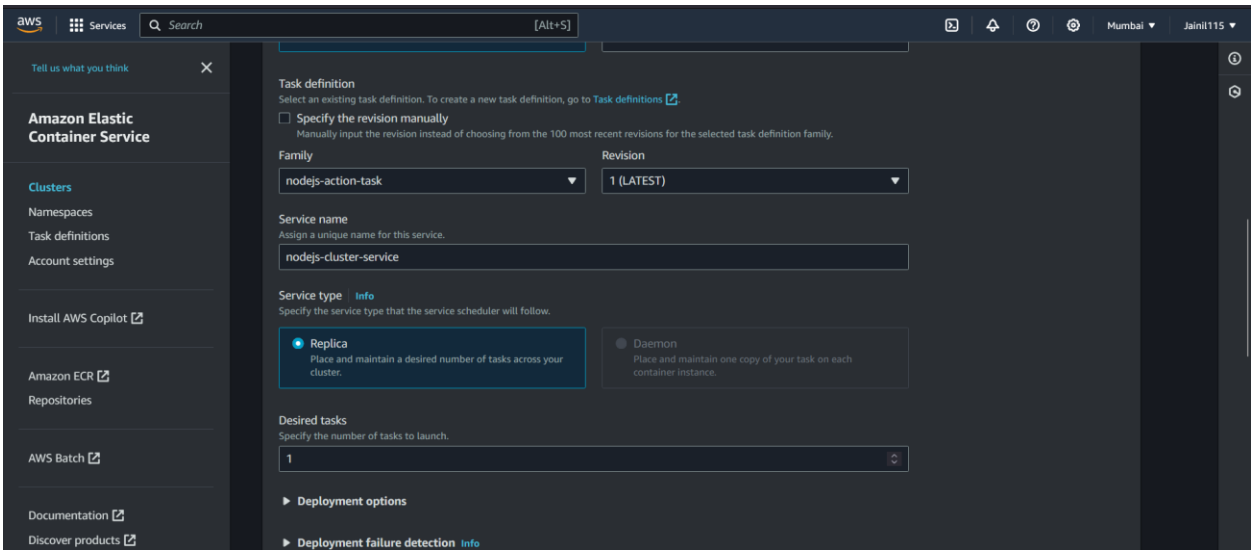


2. Select capacity provider strategy as Base: 2 and Weight: 2

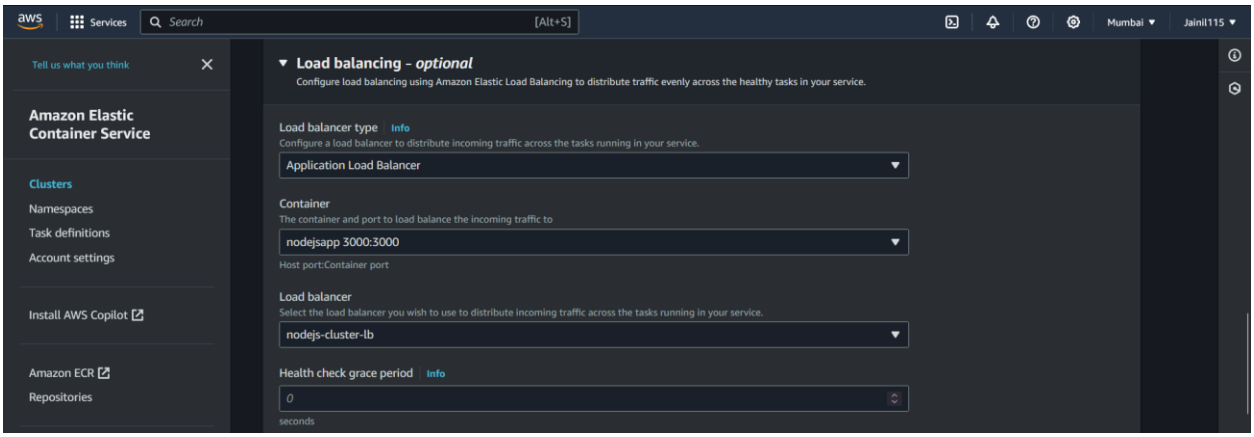


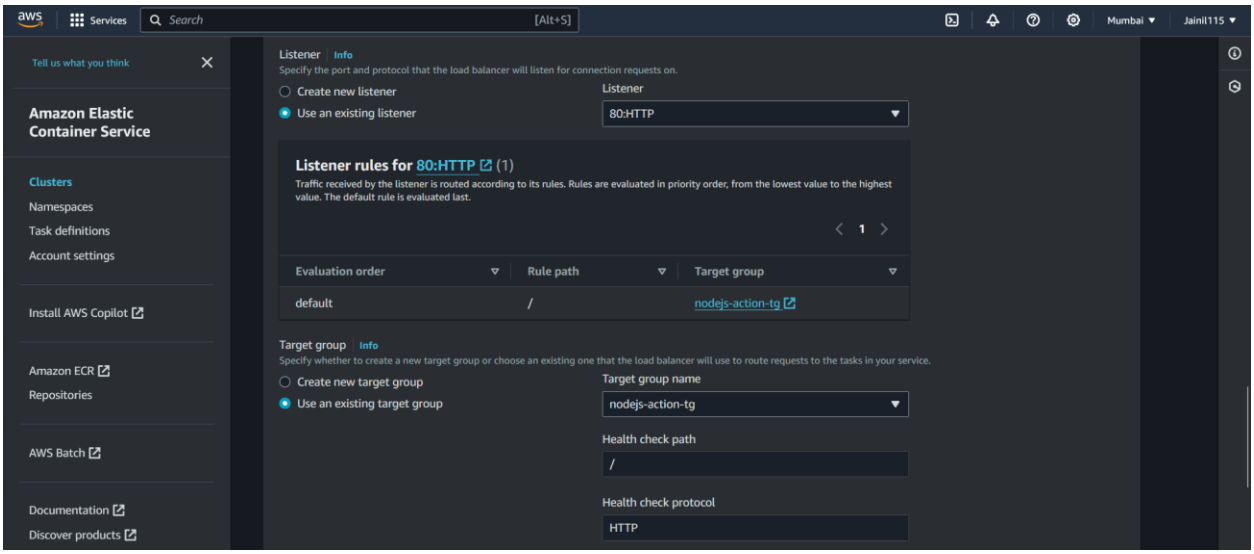
3. Now in deployment configuration select the following:
    a. Application Type: Service
    b. Family: nodejs-action-task
    c. Revision: Auto
    d. Service Name: nodejs-cluster-service
    e. Desired task: 1



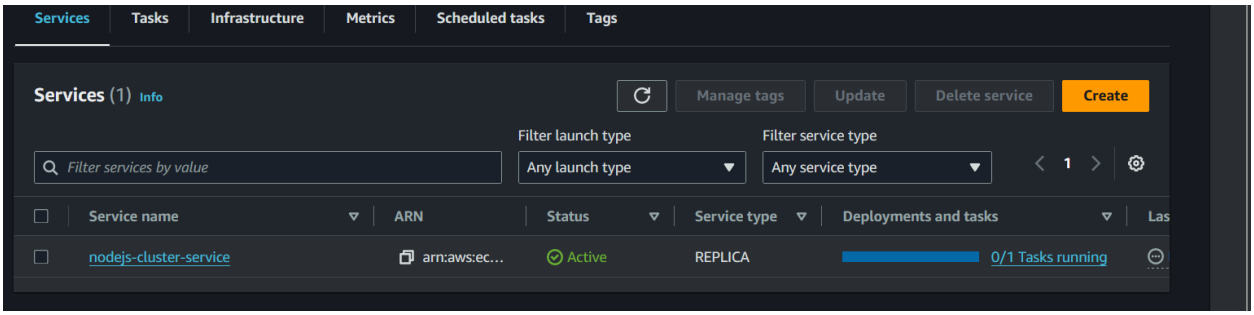


4. Now In load balancing select the following:
    a. Type: Application load balancer
    b. Container: nodejsapp 3000:3000
    c. Load balancer: nodejs-cluster-lb
    d. Listener: Use existing: 80:HTTP
    e. Target group: Use existing: Target group name: nodejs-action-tg





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

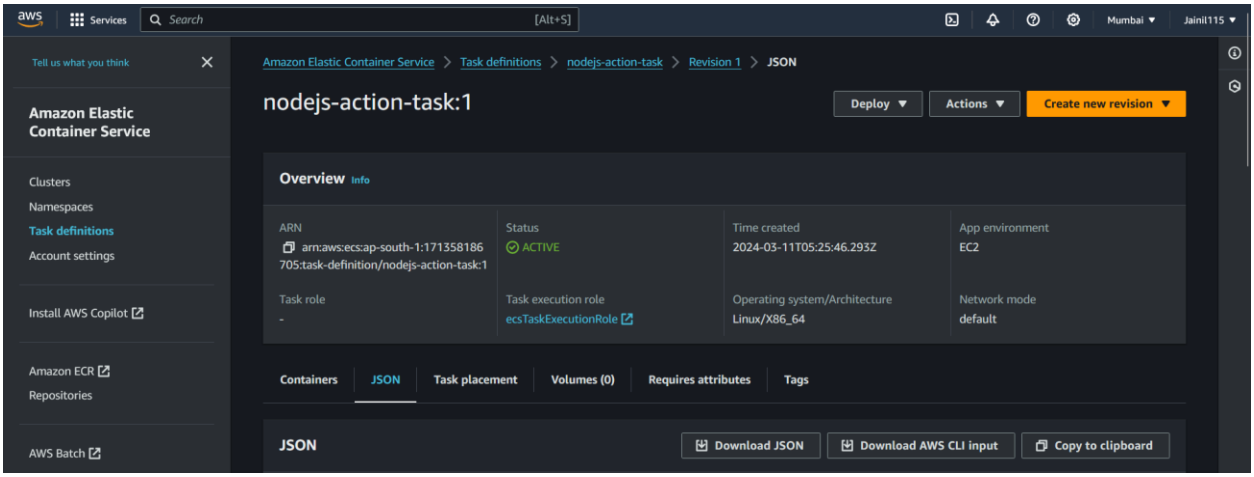
```
FROM node:21-alpine
WORKDIR /app
COPY ./app/package.json ./
RUN npm install
COPY ./app/* .
EXPOSE 3000
CMD [ "node", "index.js" ]
```

Now lets create .dockerignore:

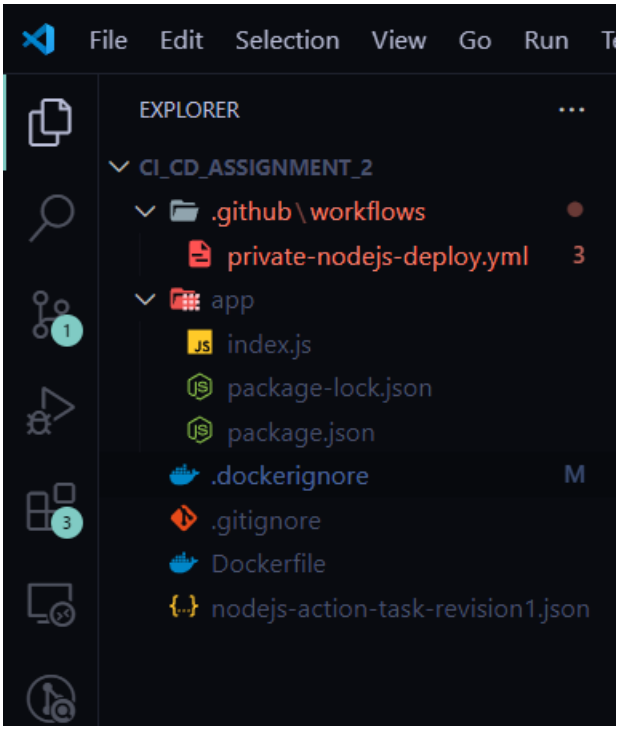
```
node_modules/
Assignment_1.pdf
/.git
/.github
.gitignore
nodejs-action-task-revision1.json
```

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

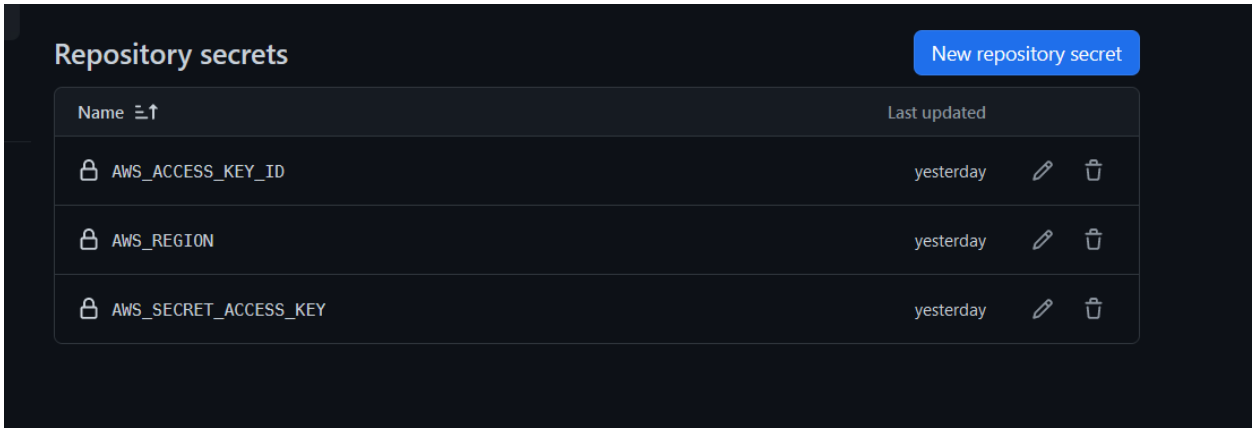
```yaml
name: Deploy to Amazon ECS

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



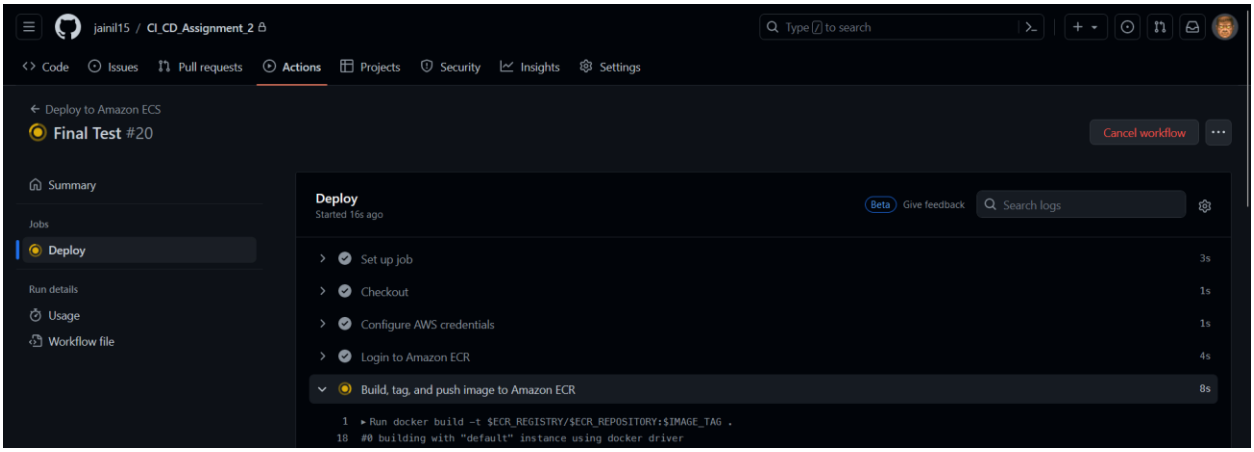
Now lets push this to github.

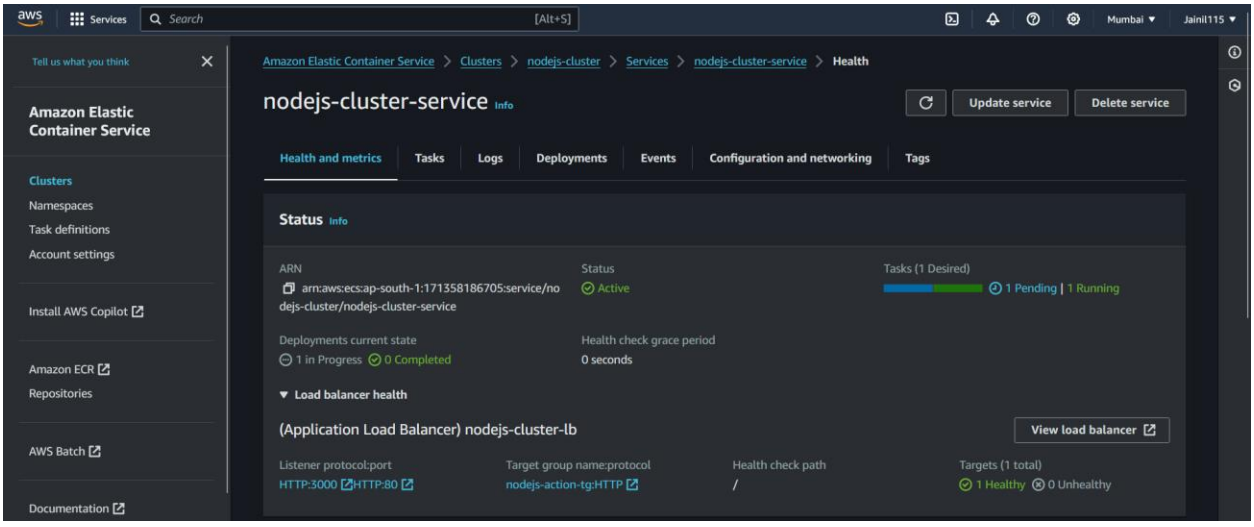
Push using command:

```
git push origin master
```

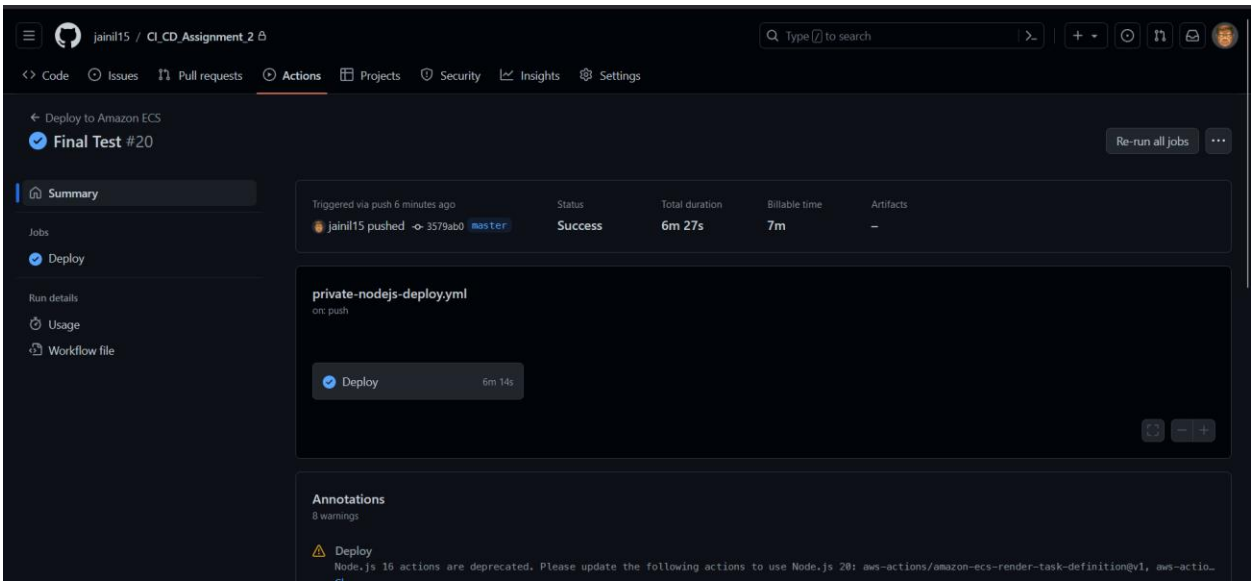
```
CI_CD_Assignment_2  master ≠  57ms  pwsh  67%  default@ap-south-1  AzureCloud
> git add .
CI_CD_Assignment_2  master ≠ ~1  48ms  pwsh  68%  default@ap-south-1  AzureCloud
> git commit -am "Final Test"
[master 3579ab0] Final Test
 1 file changed, 1 insertion(+), 1 deletion(-)
CI_CD_Assignment_2  master ≠  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
```

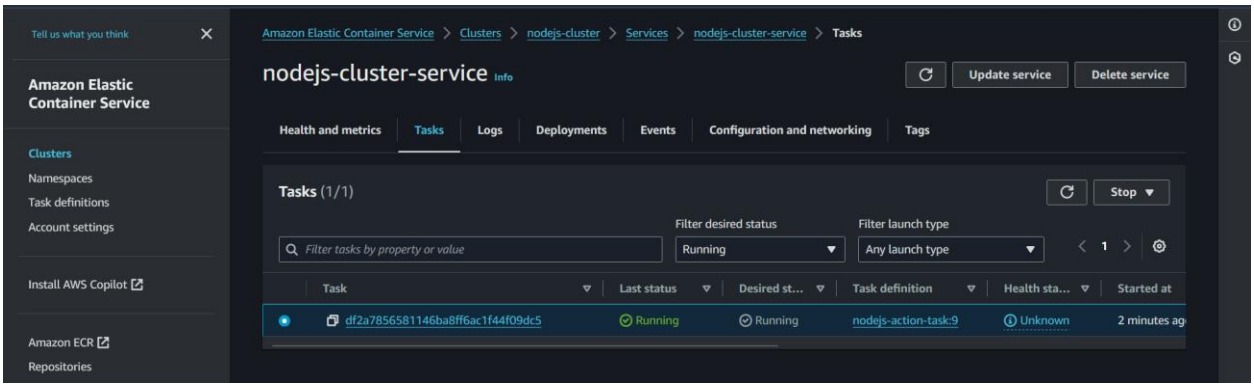
After pushing:



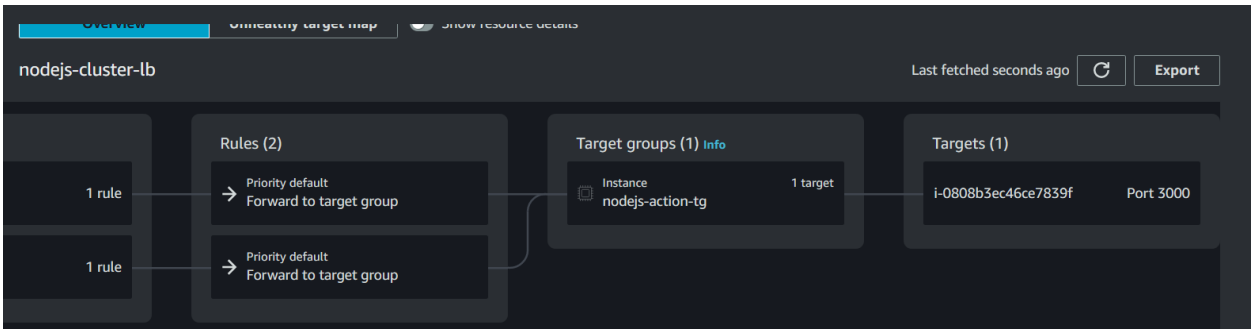


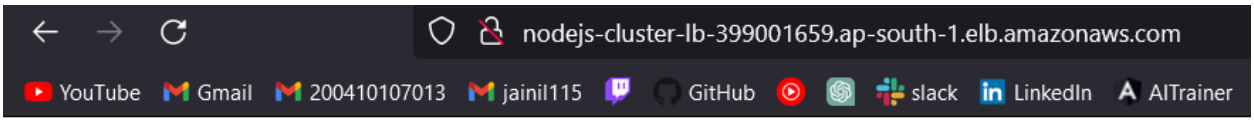
After workflow complete:





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: