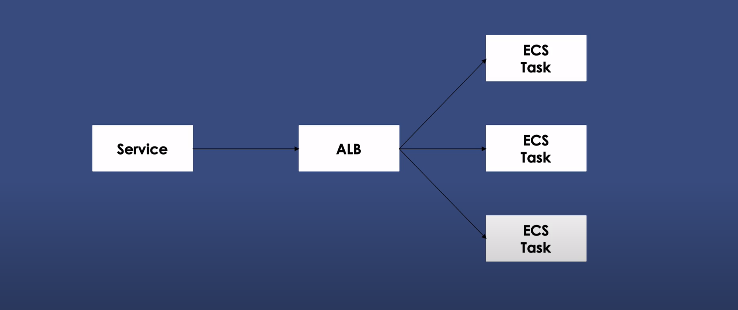
**Deploying NodeJs Application Using ECS Fargate**



**Prerequisites :**

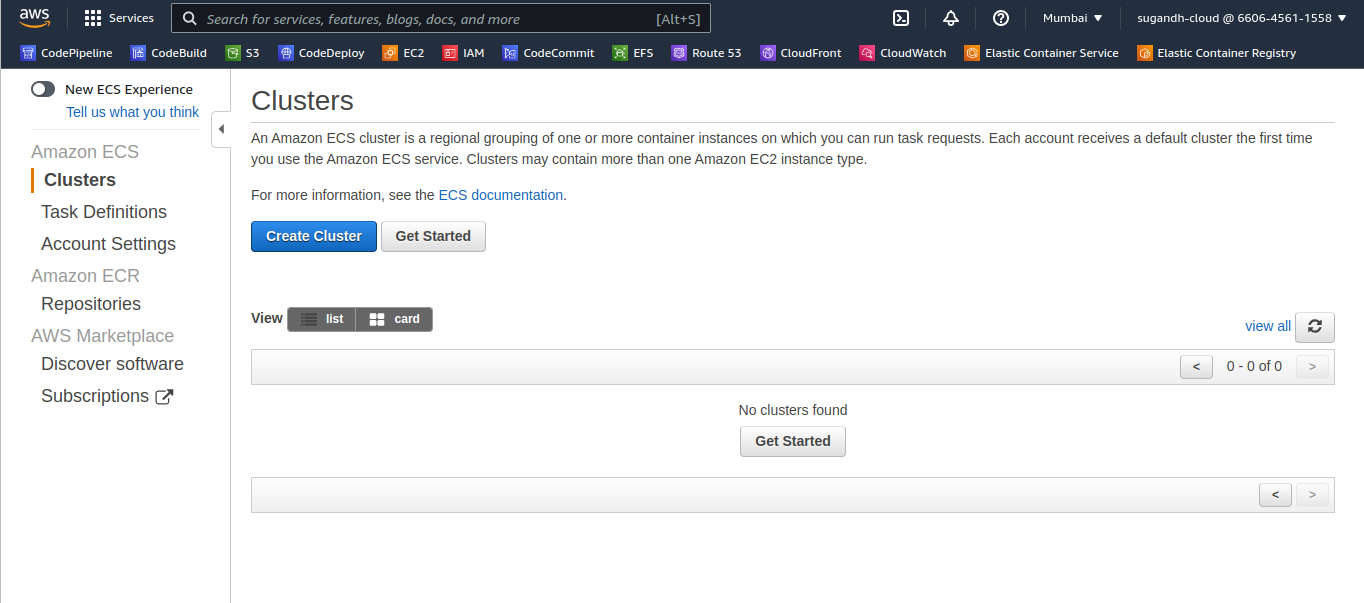
GitHub : For Source

AWS CodeBuild : Build Docker Images & push it to ECR

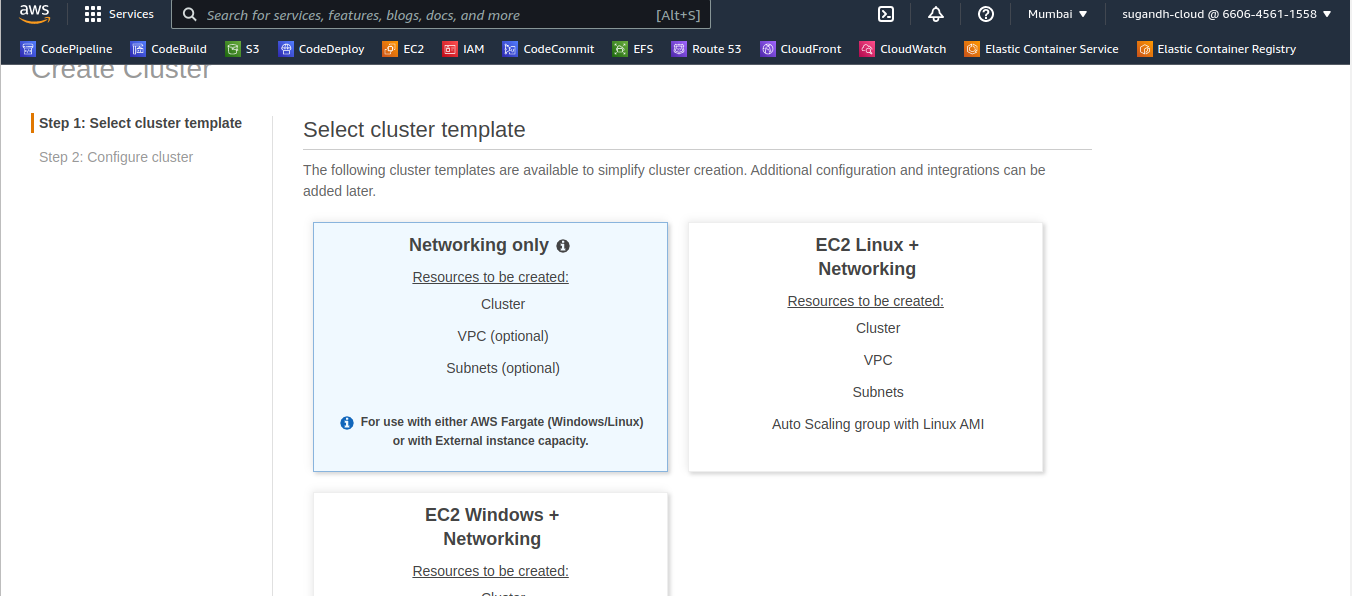
ECS Fargate Cluster : for deploying application to the environments.

**ECS Fargate** is AWS services for orchestrating production grade docker wordkloads so ECS Fargate will take care of all havey lifting & it allow us to just deploy containers into cluster & the rest is automatically taken care first

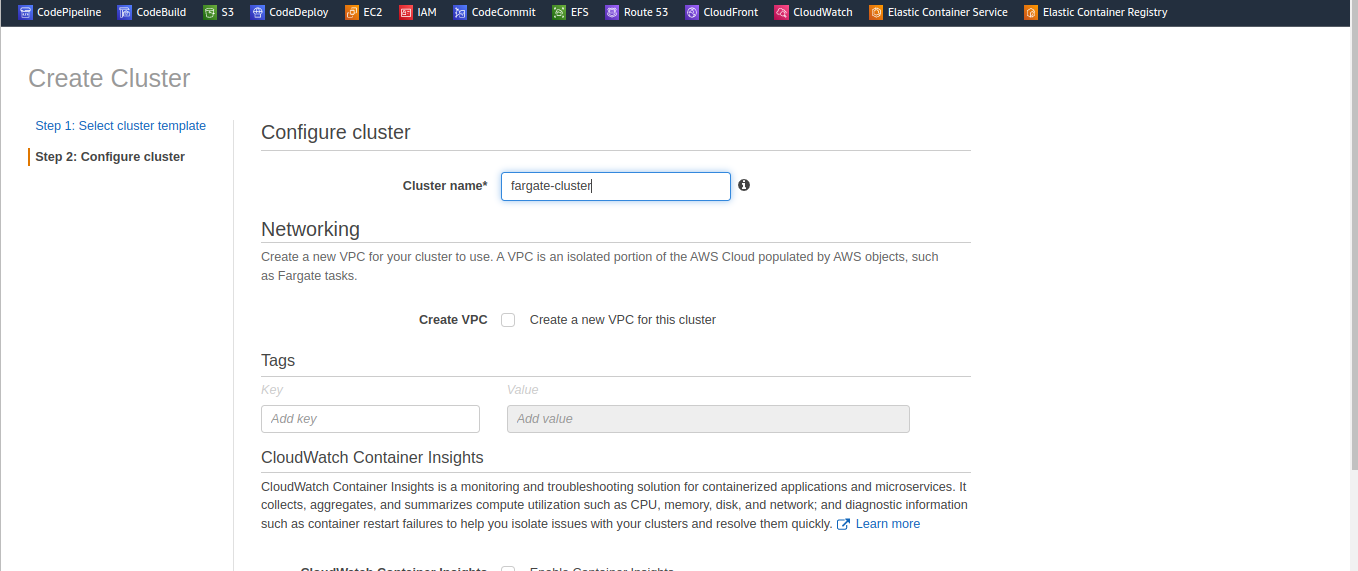
First I want to create **ECS Cluster**

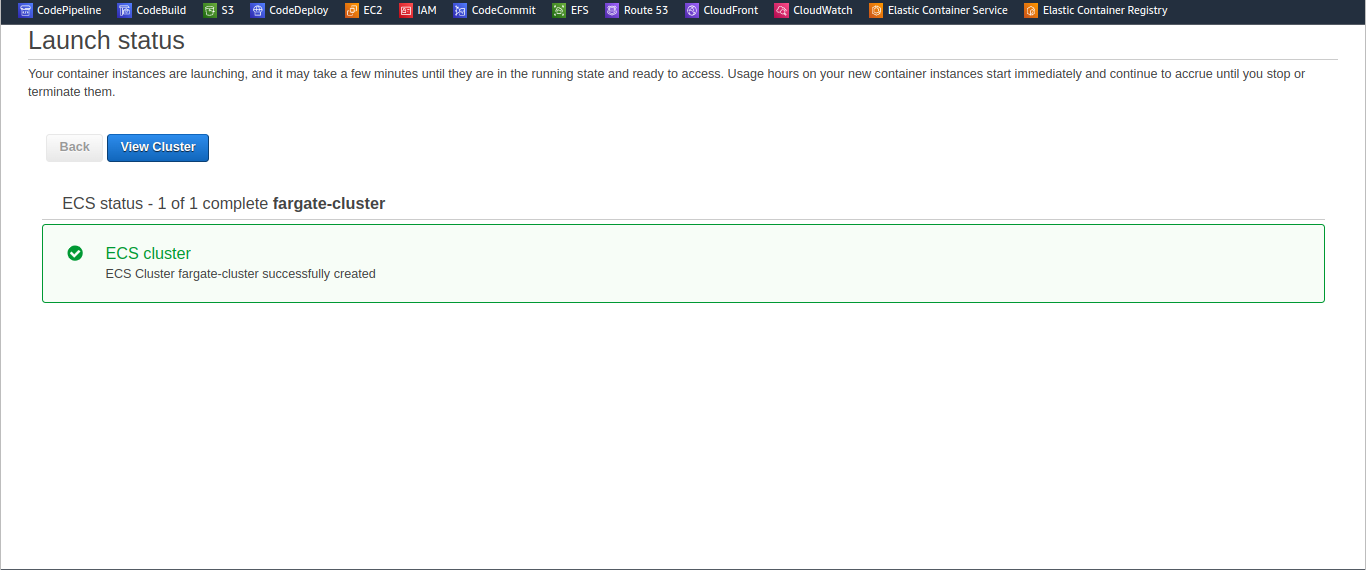
****

Choose Fargate :



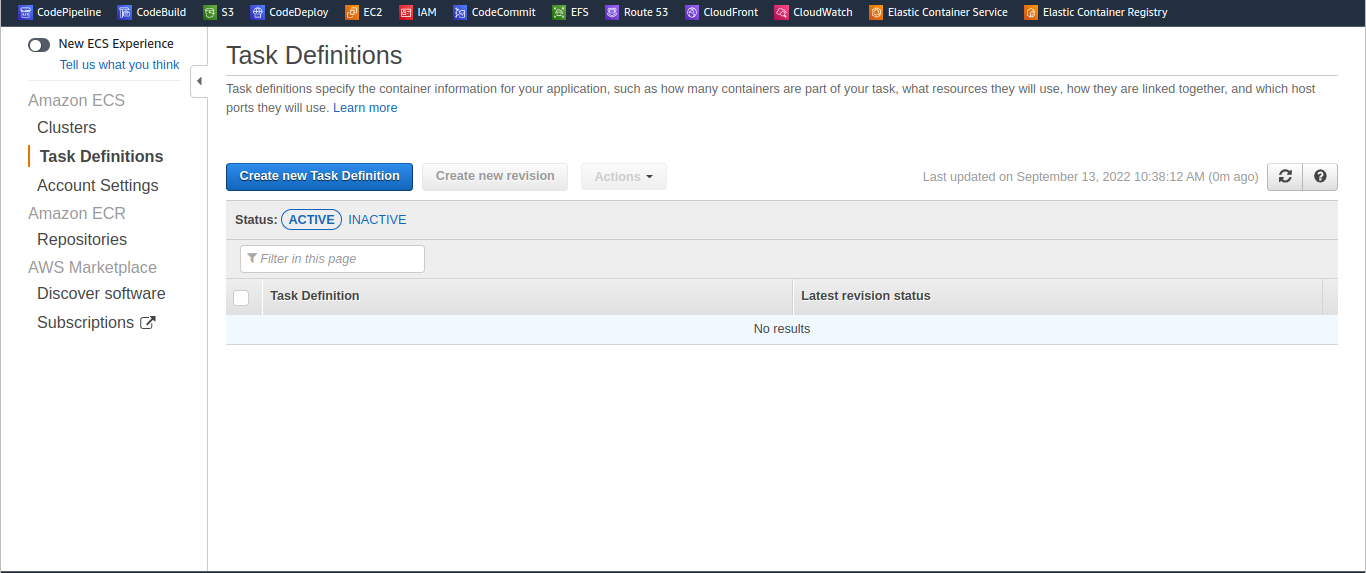
Go to next Name the cluster ‘Fargate-Cluster’ I don’t want to create VPC for the cluster here -----> create it.



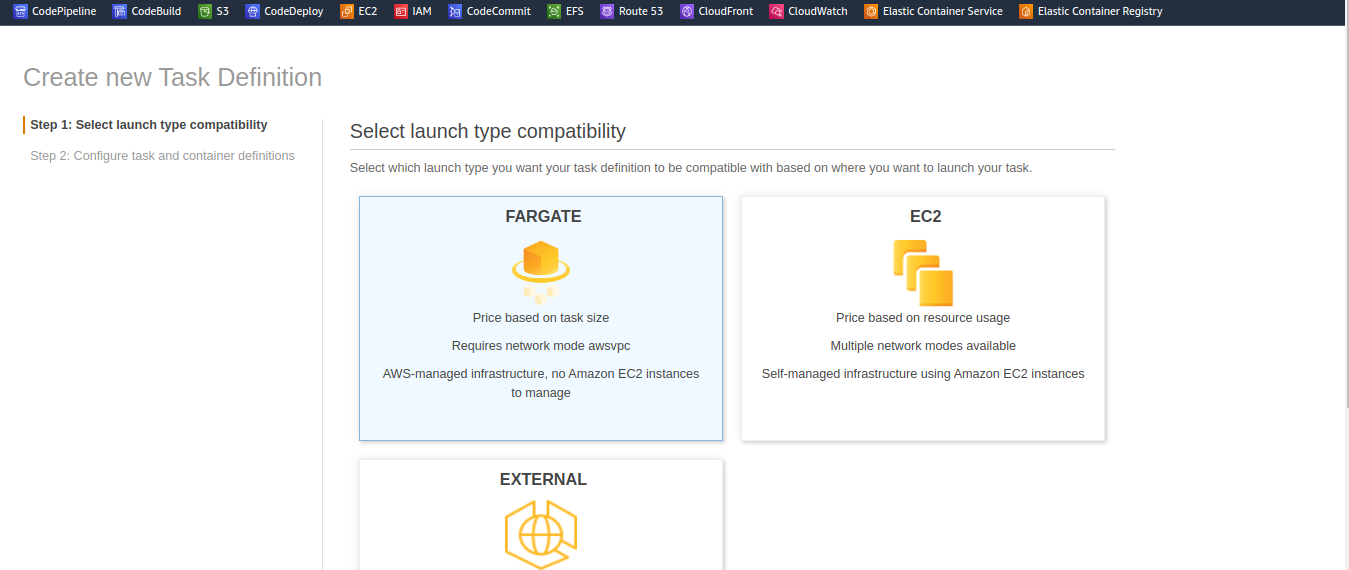


The cluster is ready the next step is to create task definition in oerder to run container inside this cluster first we need to have a **Task Definition** .

Go to Task Definition & hit Create new Task Definition .



Choose the launch type is Fargate.



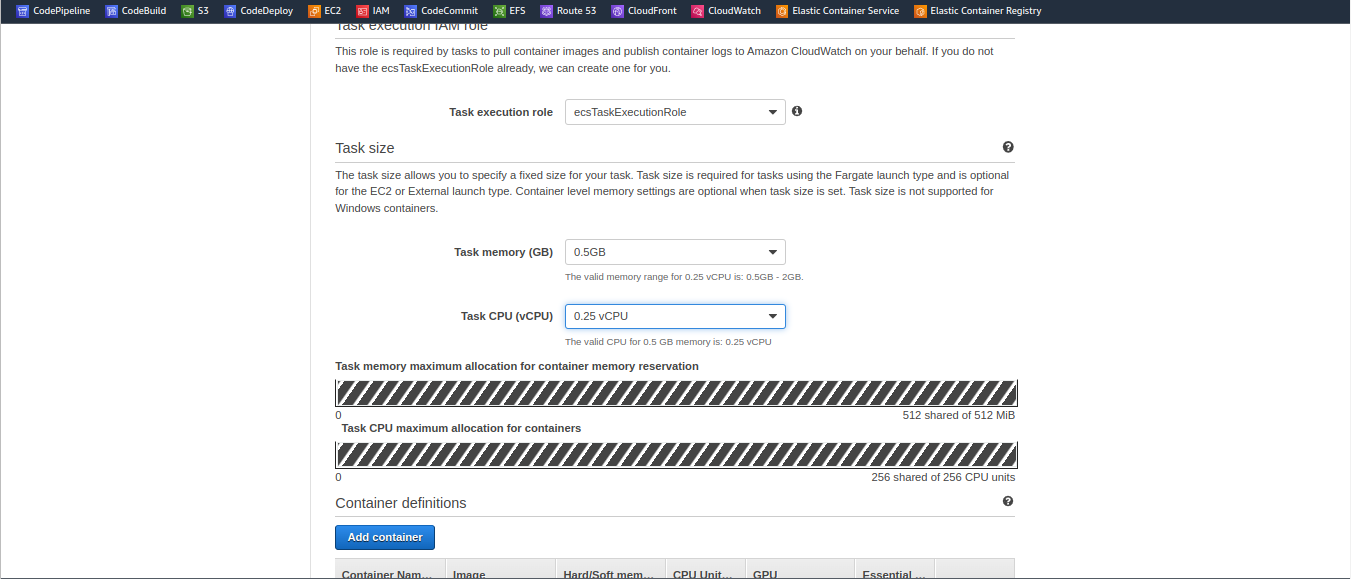
Next step Task Defination Name ‘NodeApp-’ I have one sample Node App in my GitHub repository I want to use that.

**Task Role :** Is Optional

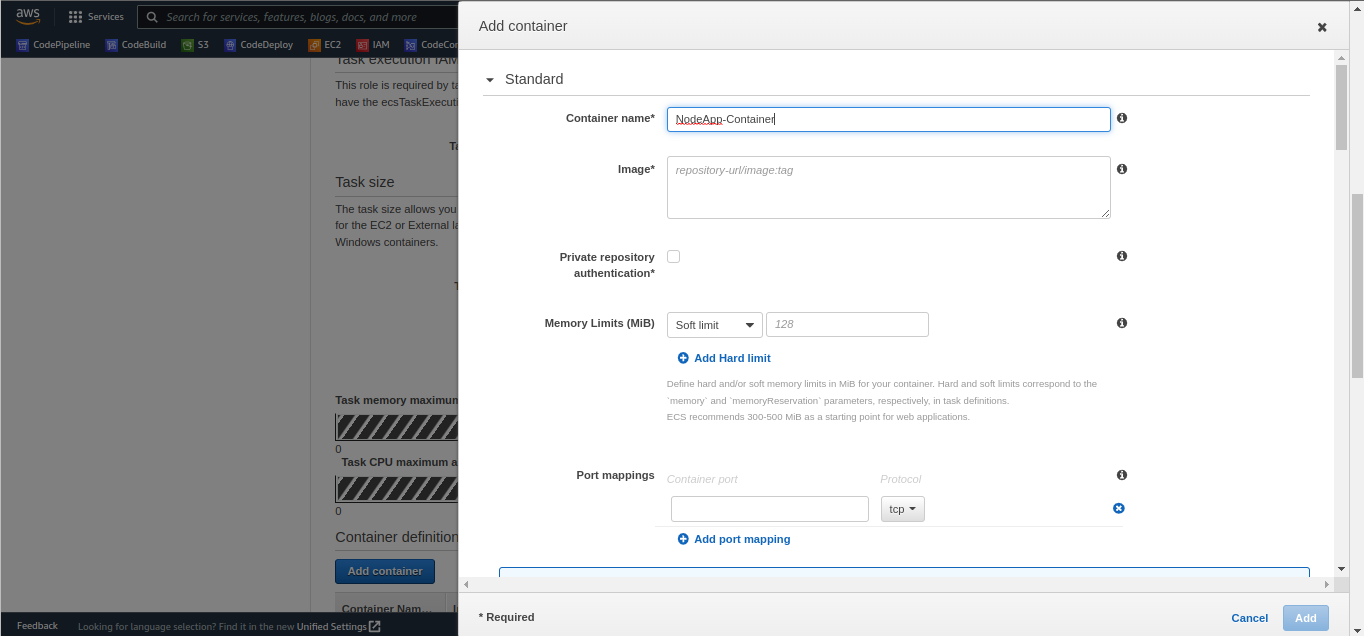
**Network Mode :** Is awsvpc sach as docker networks, for fargate kinds of task definitions the default network mode is awsvpc

**Task Definition IAM role :** this role automatically create when you launch your first ECS cluster.

**Task Size :** you have to allocate the compute & memory for the task definition & iam selecting minimum values, **0.5GB** **0.25vCPU**



So the allocation we did here is a task definition level inside each task definitions we can have multiple containers so to repeat the allocation we made is a task definition level & that we are going to spread across multiple containers inside task definition , In This practice iam going to have a task with only one container so add container.

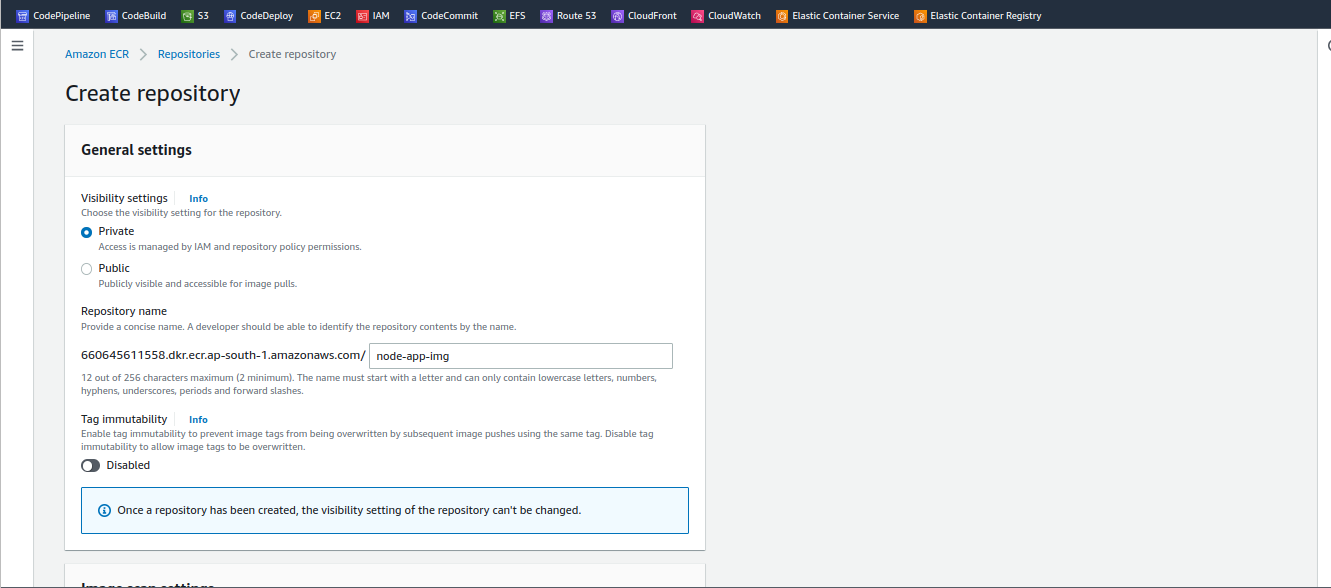


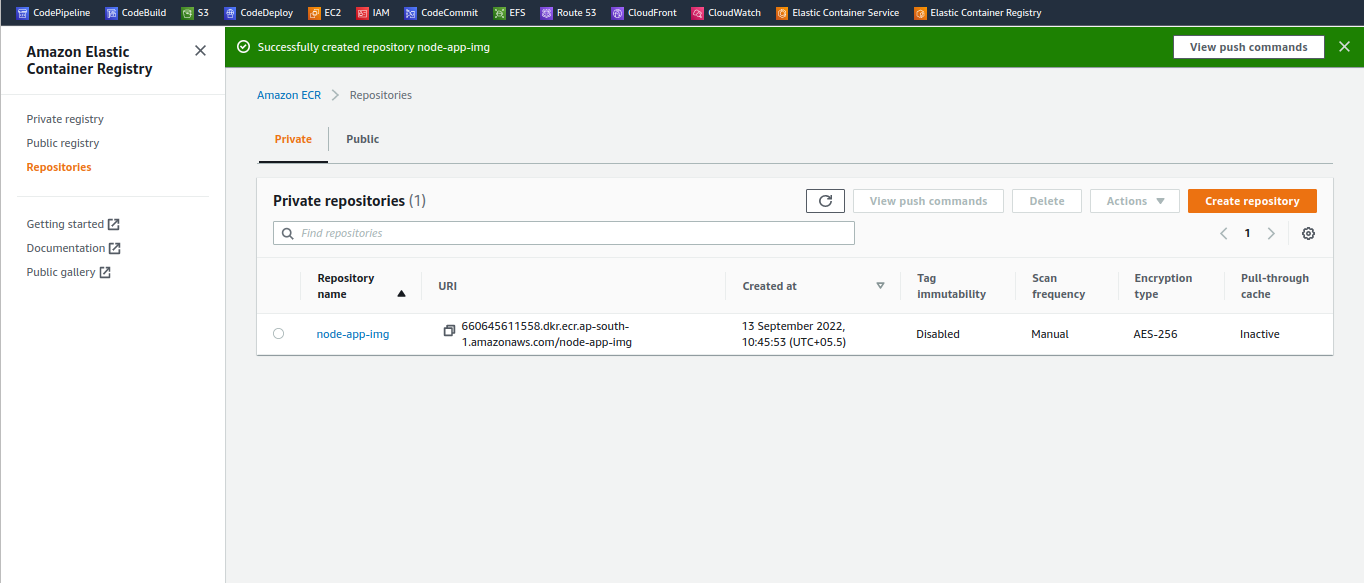
Fill the container name & iam going to pull the images from my ECR ,

Let’s setup a ECR repository , Go to AWS Console ---->

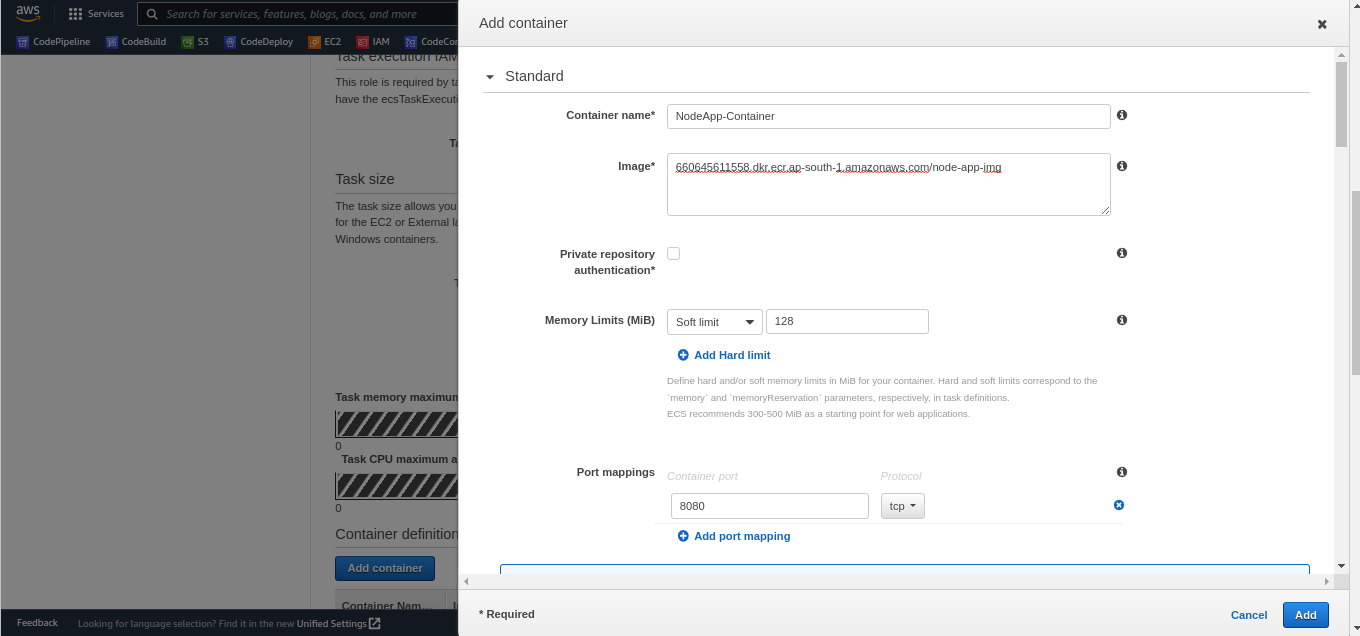


Choose ECR ----> Create repository ----> Get started





As of now we do not have images, Copy the image url & paste it on ECS task definition

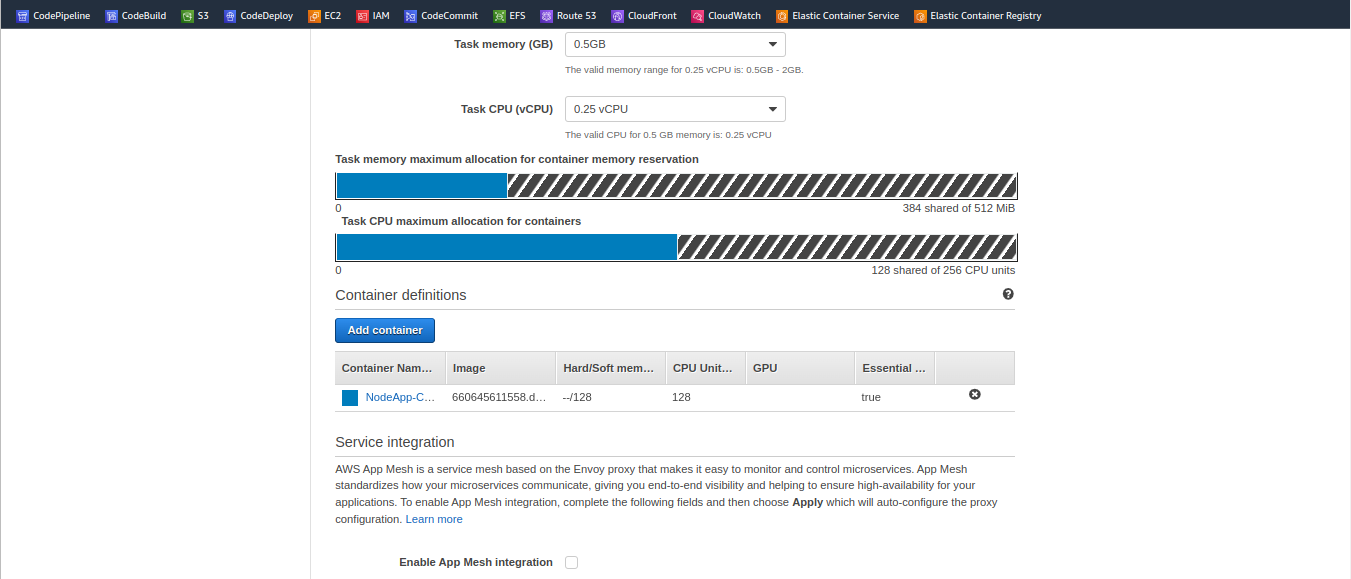


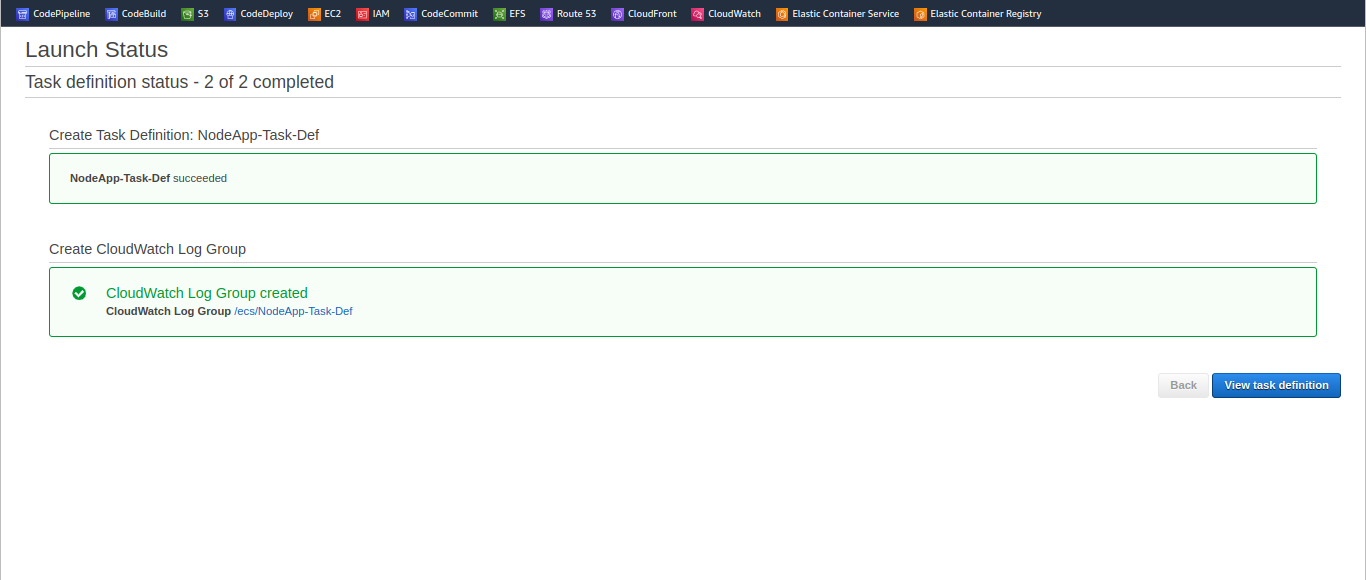
**Soft Limit :** nothing but the minimum allocation is made to the container at launch time So you can set hard limit, when you set hard limit

**Hard Limit :** when you set hard limit it can allocate memory up to the hard limit if you leave it so whatever it available at task definition level that is utilized on demand

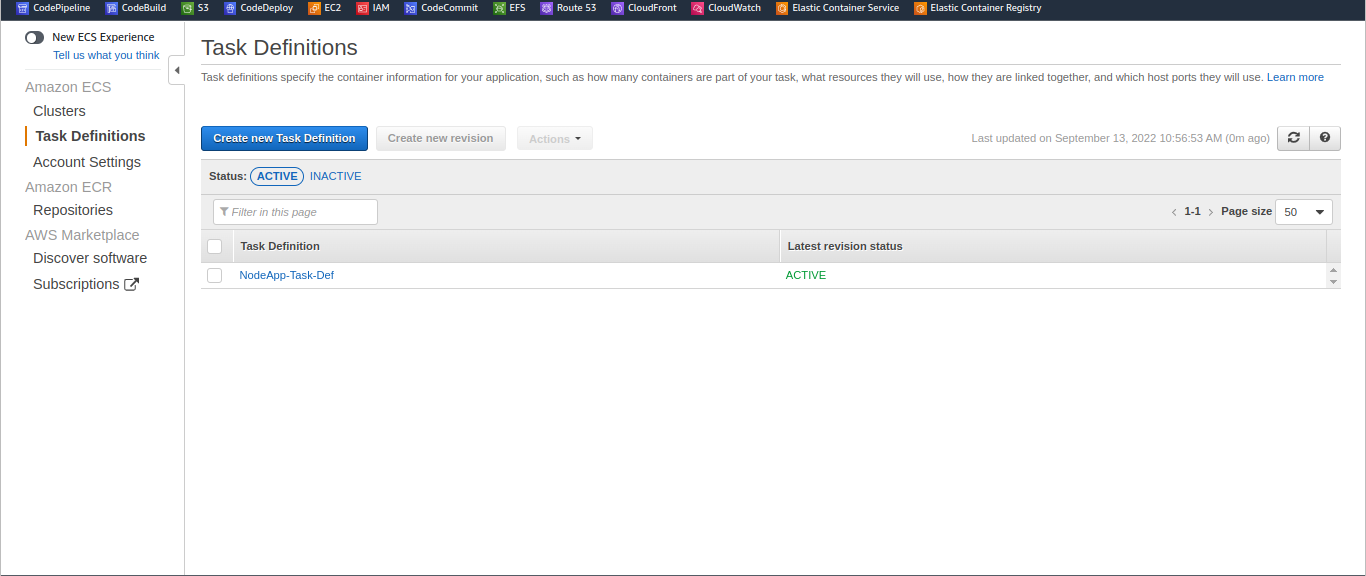
**Port Mapping :** Inside container out application is exposing 8080 let me keep it 8080& we have other details like health check all these details are not required for this demonstration all the stuff is not required as of now.

Task Definition level we have 512 MB for memory & CPU 256 vCPU unites these are the allocation .

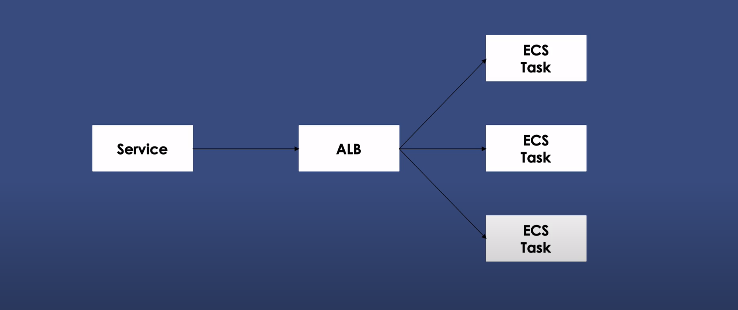




Even our task definition is ready .



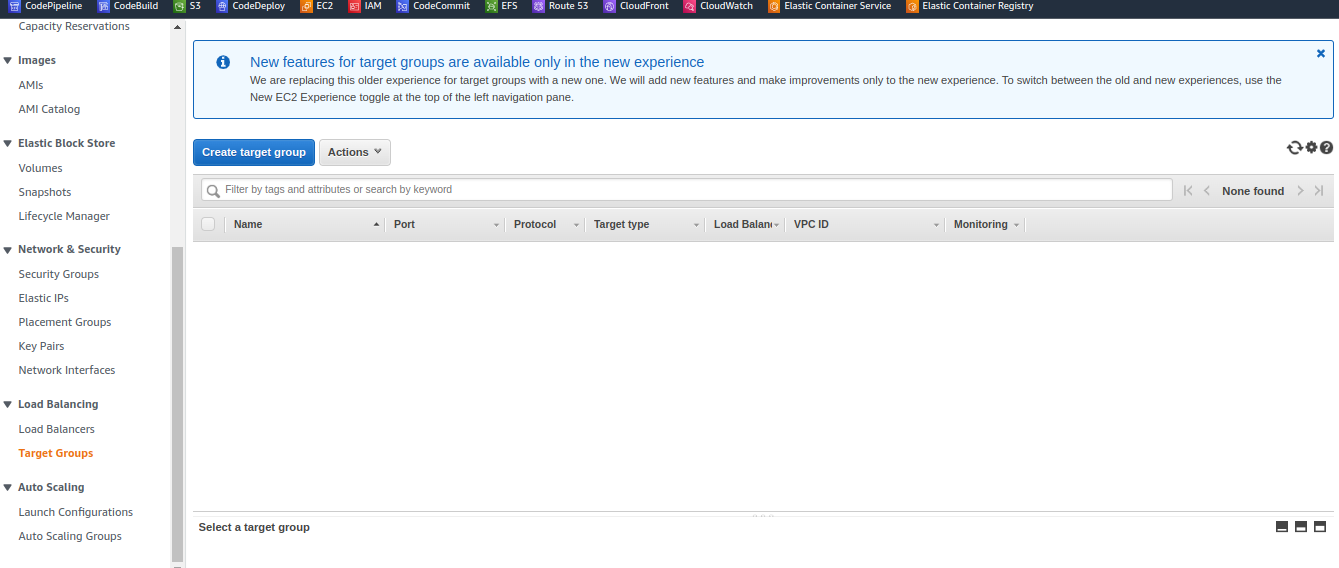
We have task definition is ready now , we have cluster & task definition

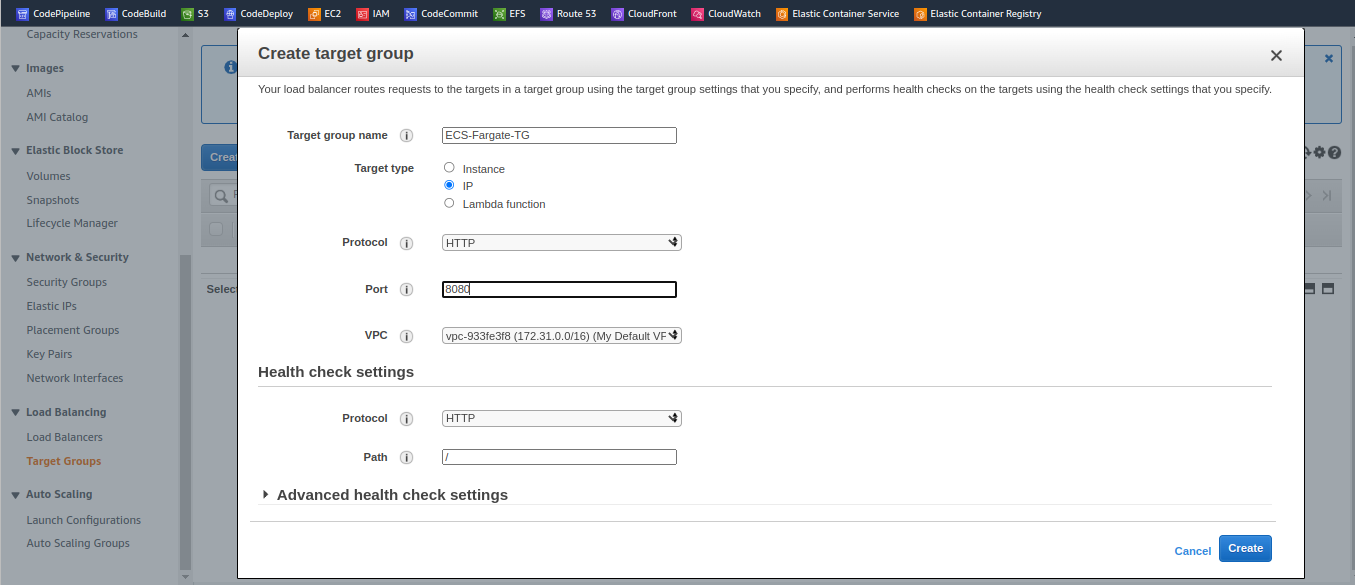


Using service am going to create multiple tasks nothing but replicas & I want to add those behind Application Load Balancer (ALB) sach that in user will hit ALB & ALB disributes traffic across multiple replicas of your tasks,

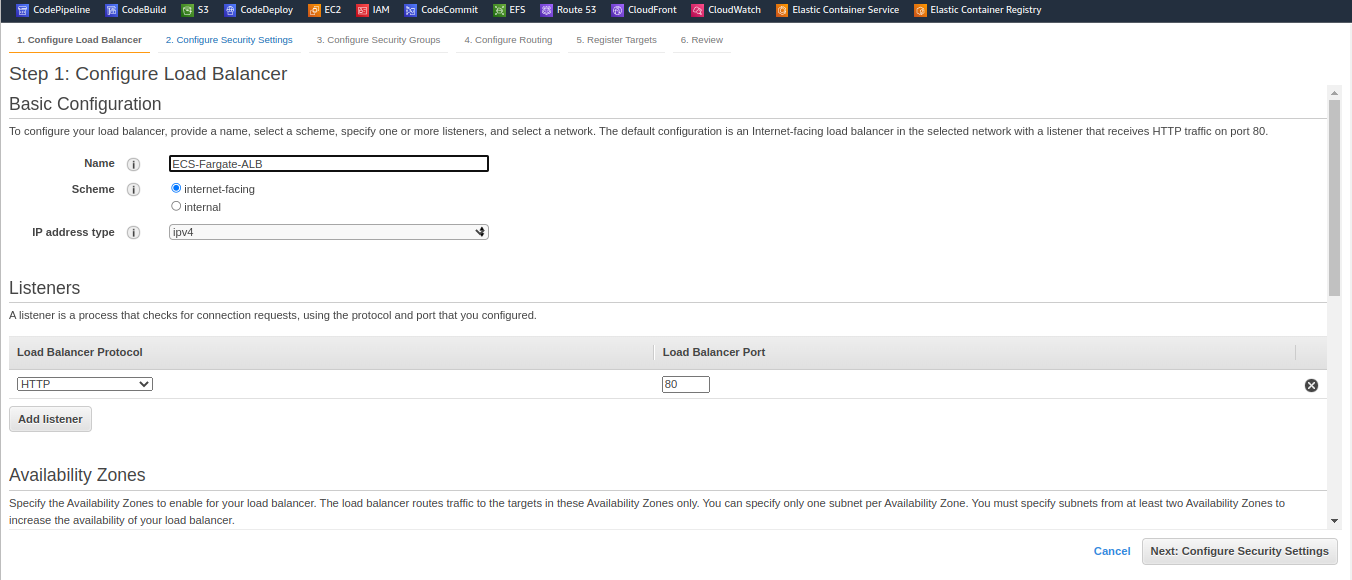
Before i go ahead & create a service I need Application Load Balancer (ALB)

Go to EC2 ---> select under load balancer ‘taget group’.

****

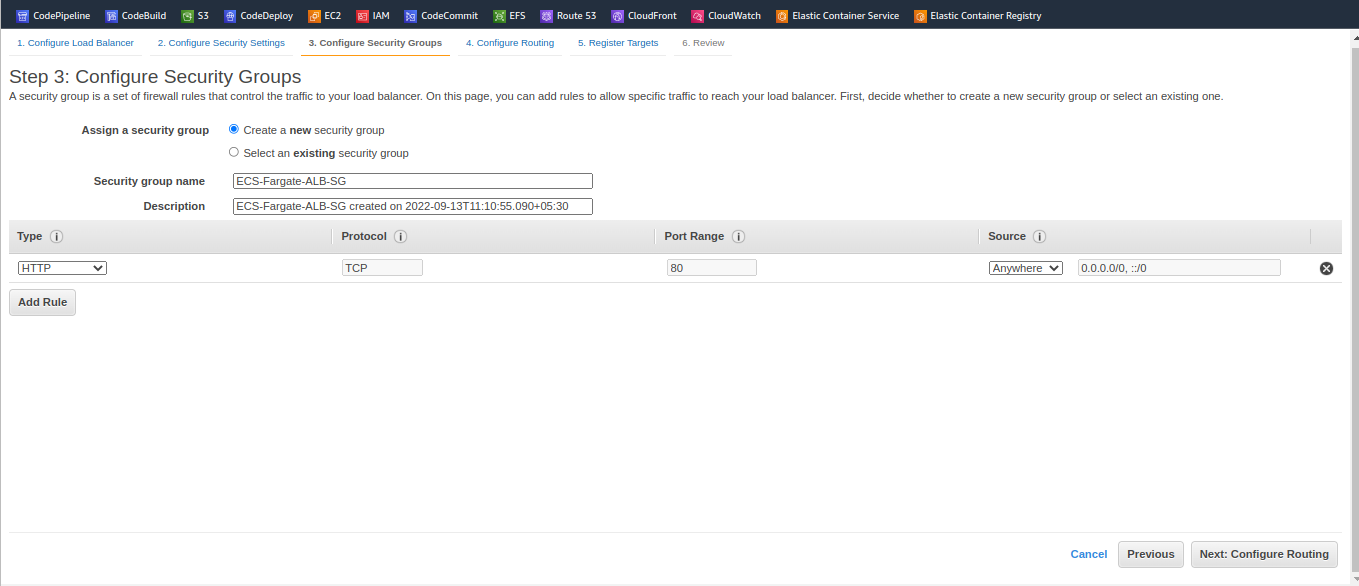


Next step is Load Balancer, create load balancer Application Load Balancer.

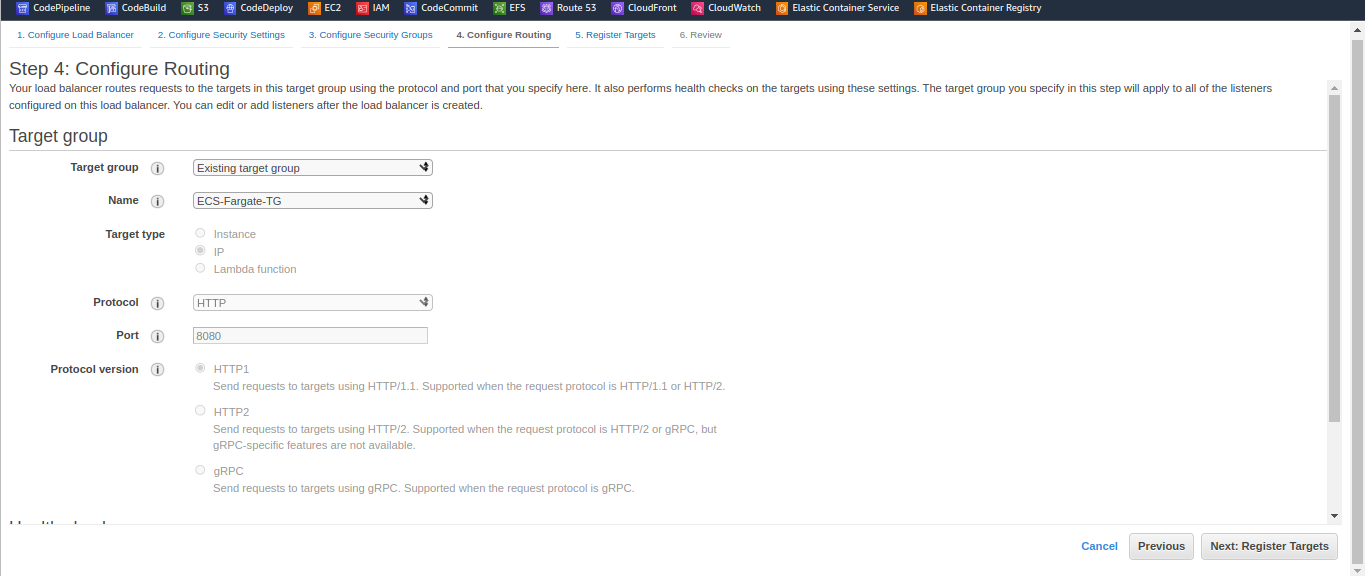
****

We are going to deploy ECS tasks in default VPC, so picking default VPC & Subnets.

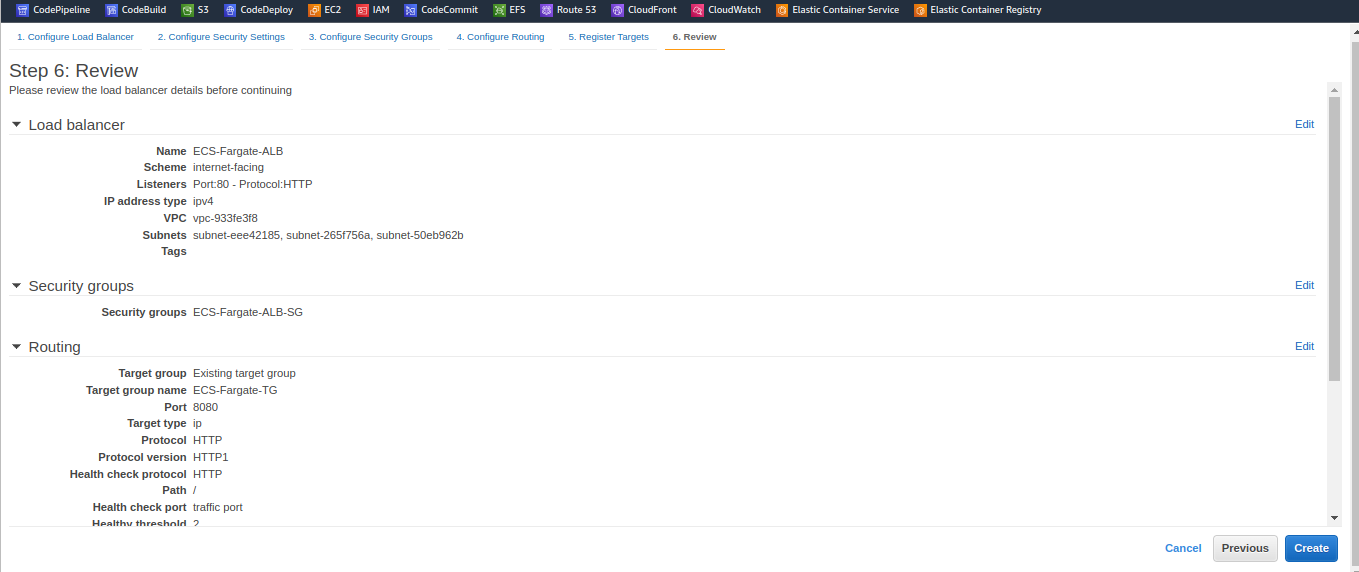
Now configure security group settings ----> Create a Security Group

****

Now Configure the routing part, we have existing target group here, so thses details auto-populated .

****

Review & create it.

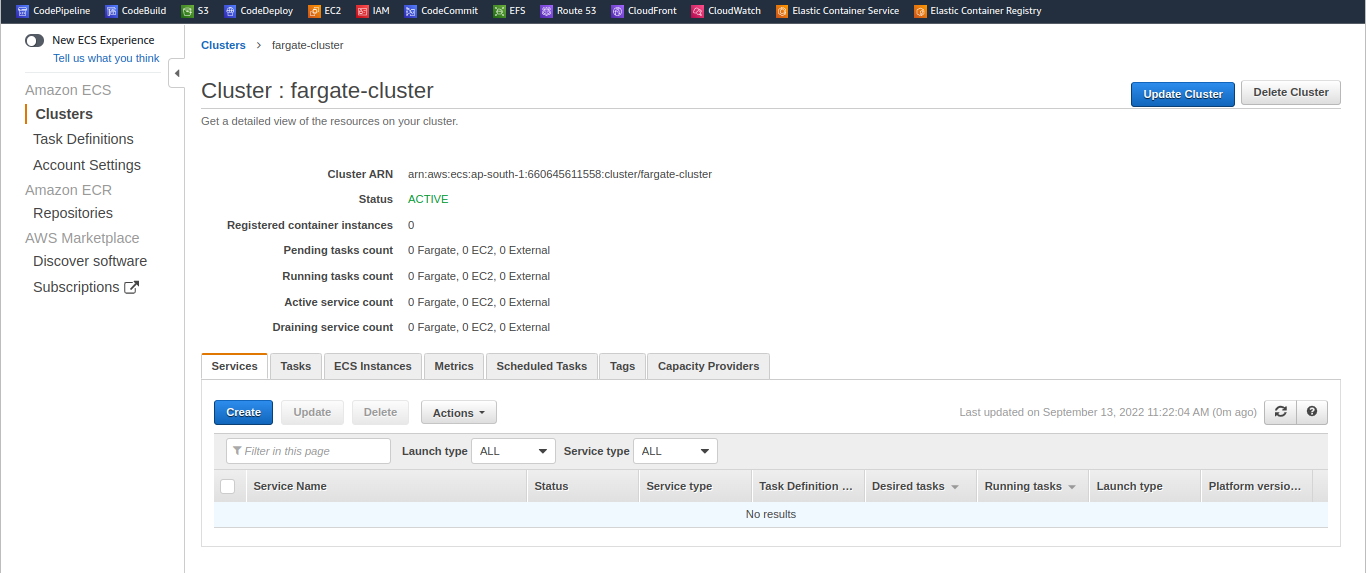


“ALB listens on 80 & the Target Group link they listen on 8080given hralth checks on ECS tasks & our ECS tasks runs on 8080so thats why I made this 8080”

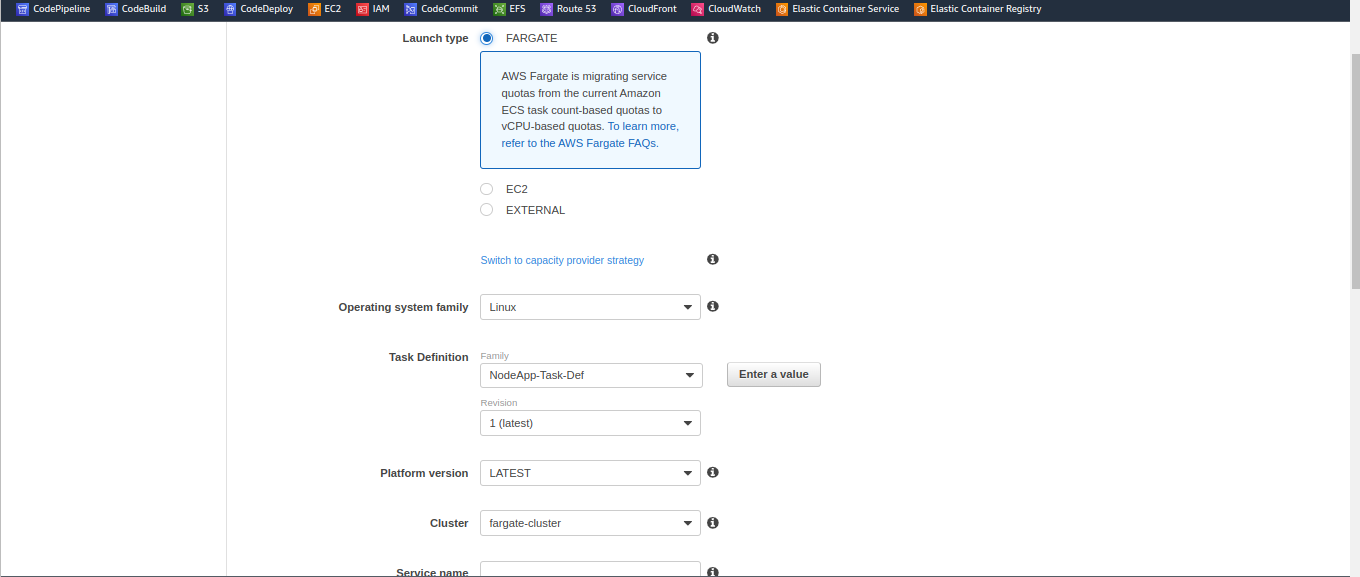
The next step is to create a service we have cluster tasks definition we have ALB for integrating the tasks with the Load Balancer.

The next step is to create a **service** go to **clusters** open your cluster --->

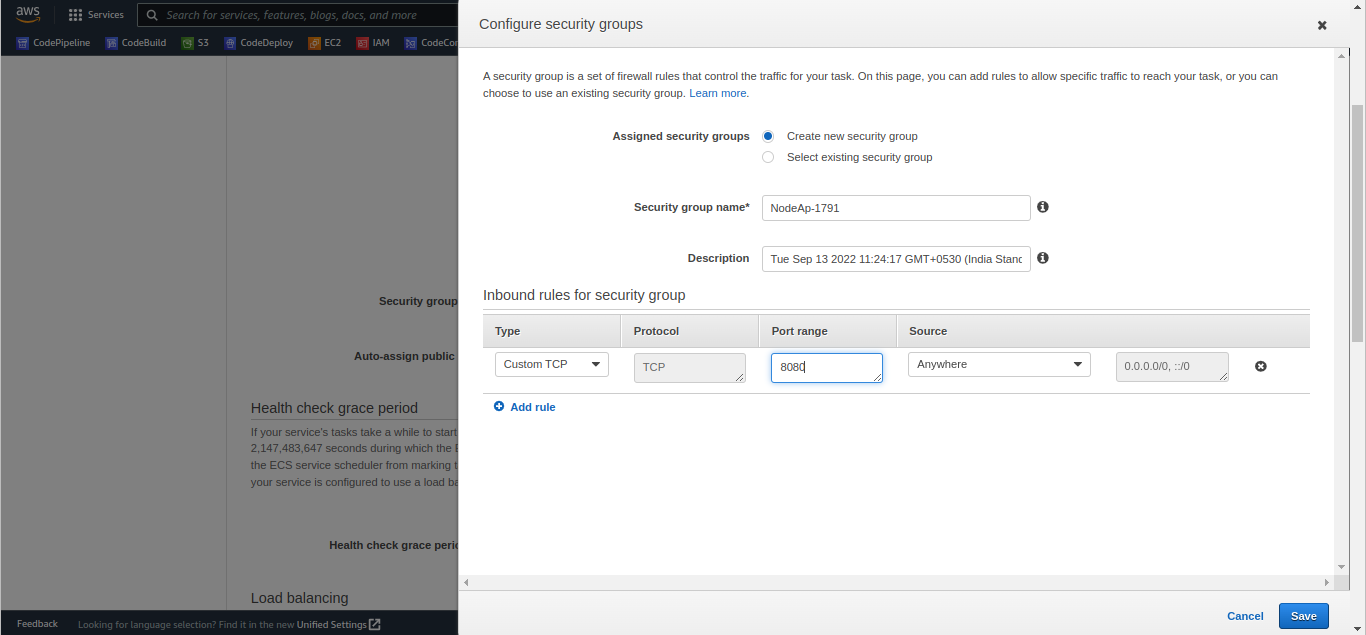
Open your cluster -----> create a **Service .**



Launch type is **FARGATE** latest Definition we have 1 (latest) click next step.

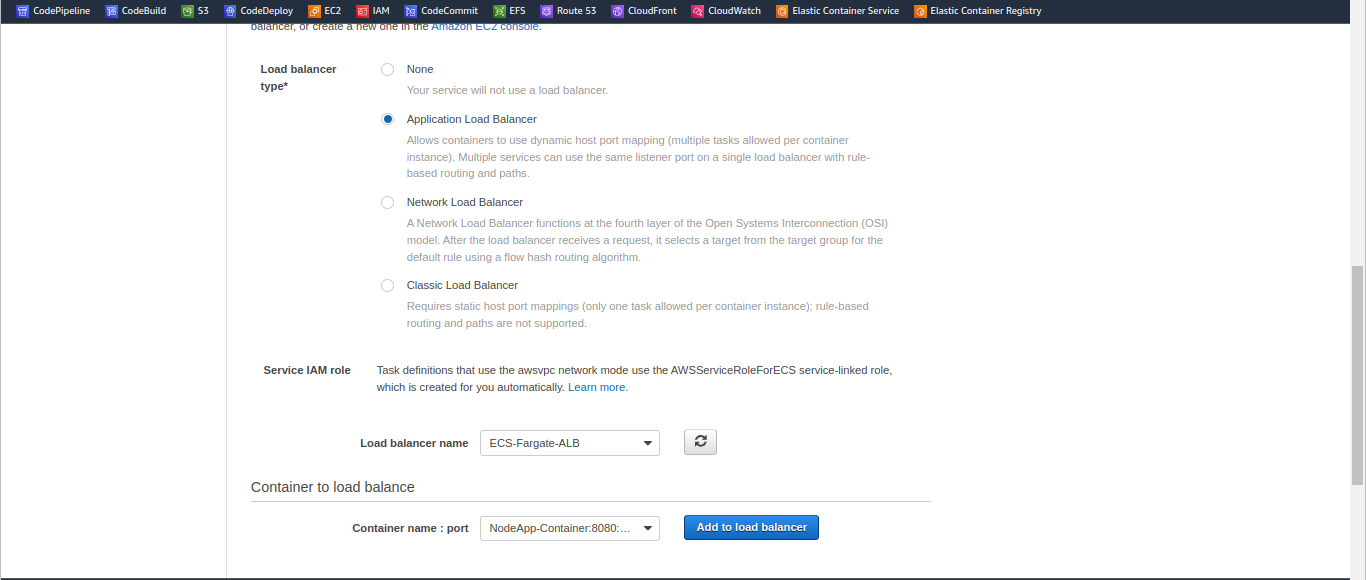


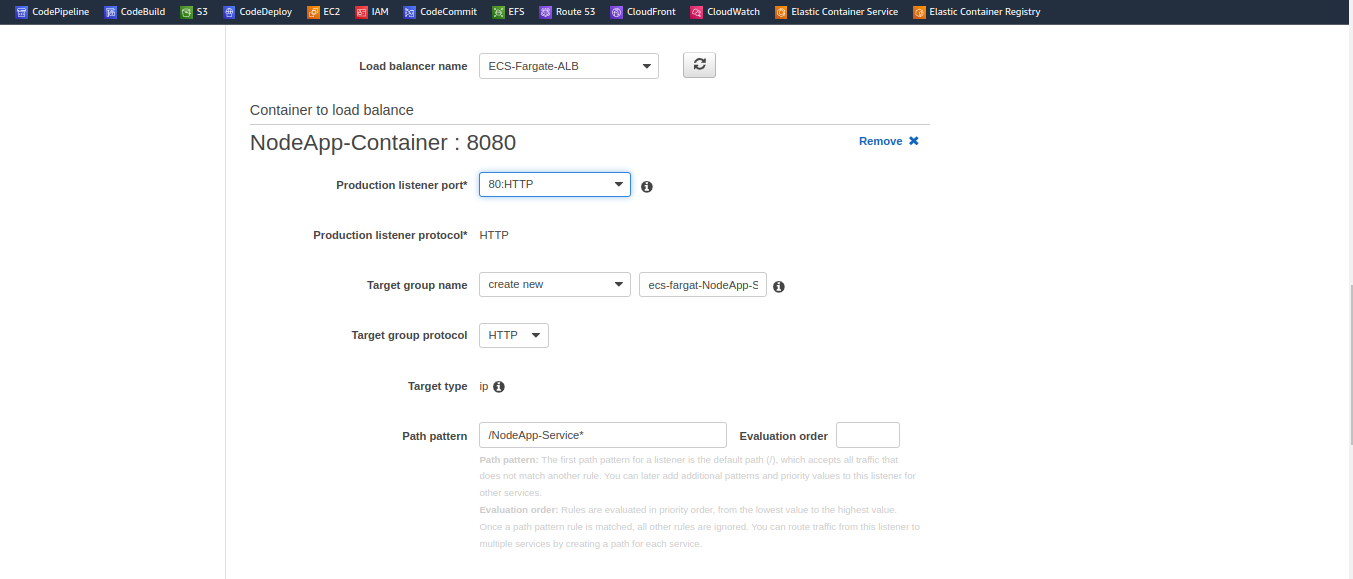
Choose default VPC & for hight availablity, I select more than one Subnet across the diffrent zones, let’s edit **Security Group** & choose custom TCP & listen port on 8080, so this security group is a firewall for your tasks.



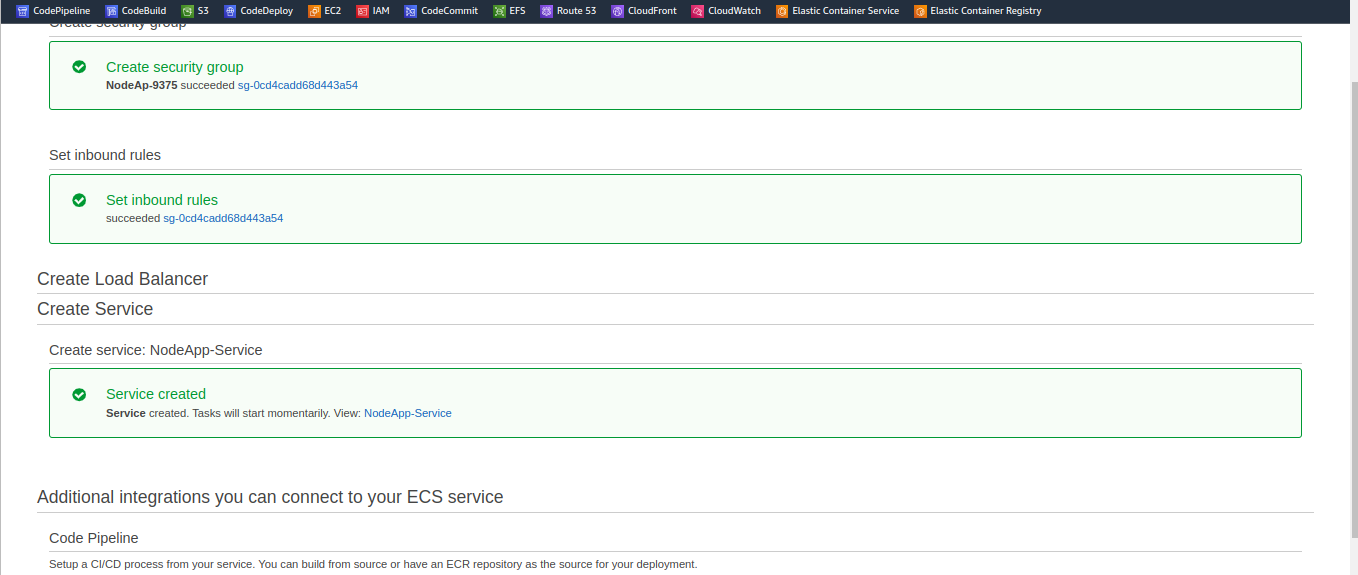
As I said tasks are listening on 8080 that’s why am opening 8080 source anywhere, Save.

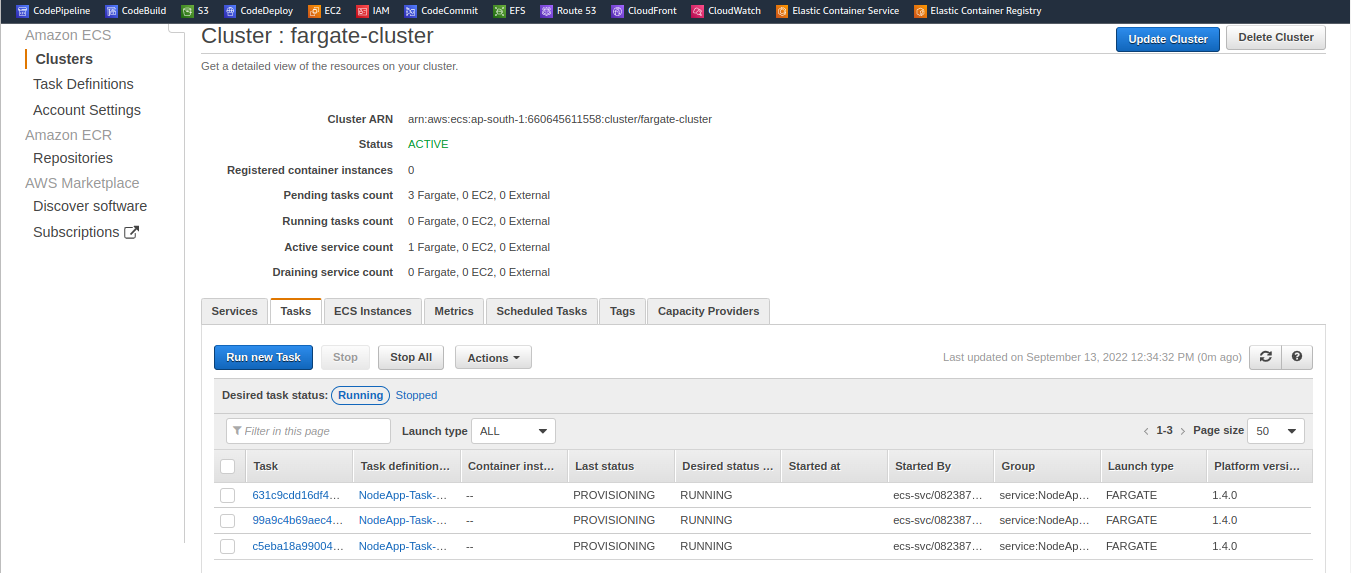
Now I want my service intregrate with ALB , Select that Load Balancer Name , it auto-populated & so container to Load Balancing mapping is auto-populated again so add to Load Balancer





Even autoscalling I do not want Click on create.





This tasks going to fail for sure why beacause we’re pointing to ECR which don’t have any imageits going to fail when it pulls images & obviously the task

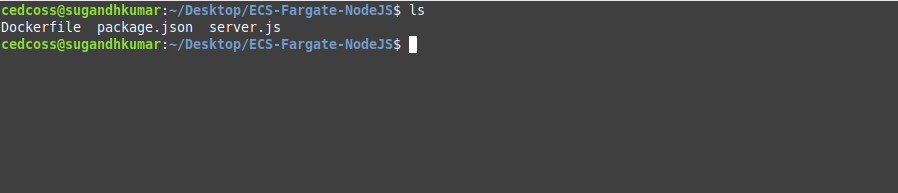
provisioning is also going to fail tempoarrily fixed this problem let me upload my image to ECR

Go to terminal ----> Create 3 files.

1 : Dockerfile

2 : package.json

3 : server.js



**sudo nano Dockerfile**

FROM node:16

WORKDIR /usr/src/app

COPY package\*.json ./

RUN npm install

COPY . .

EXPOSE 8080

CMD [ "npm", "start" ]

**sudo nano server.js**

'use strict';

const express = require('express');

// Constants

const PORT = 8080;

const HOST = '0.0.0.0';

// App

const app = express();

app.get('/', (req, res) => {

res.send('<h1 style="color:green;">CEDCOSS ECS Testing - v1!!</h1> \n');

});

app.listen(PORT, HOST);

console.log(`Running on http://${HOST}:${PORT}`);

**sudo nano package.json**

{

"name": "docker\_web\_app",

"version": "1.0.0",

"description": "Node.js on Docker",

"author": "First Last <first.last@example.com>",

"main": "server.js",

"scripts": {

"start": "node server.js"

},

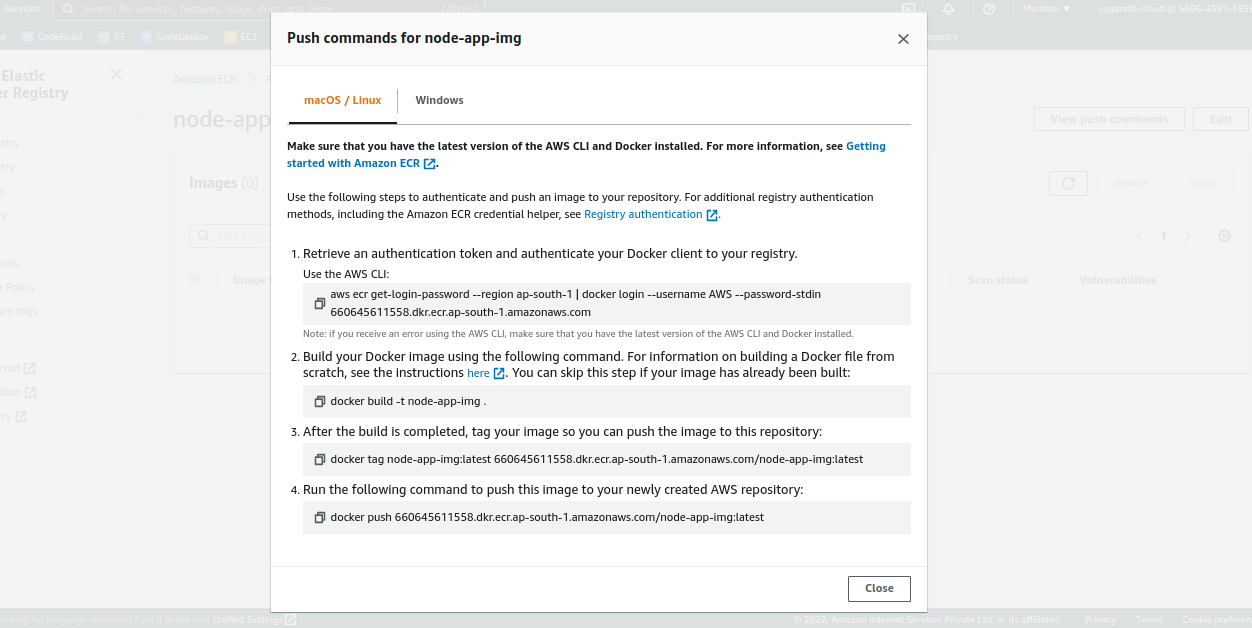
"dependencies": {

"express": "^4.16.1"

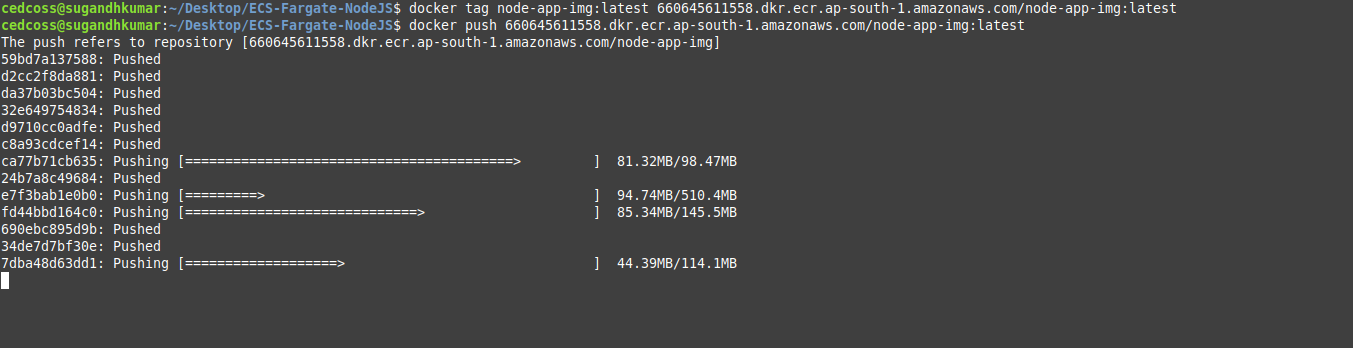
}

}

Now go to ECR ----> Create a repository ----> click on the pop-up

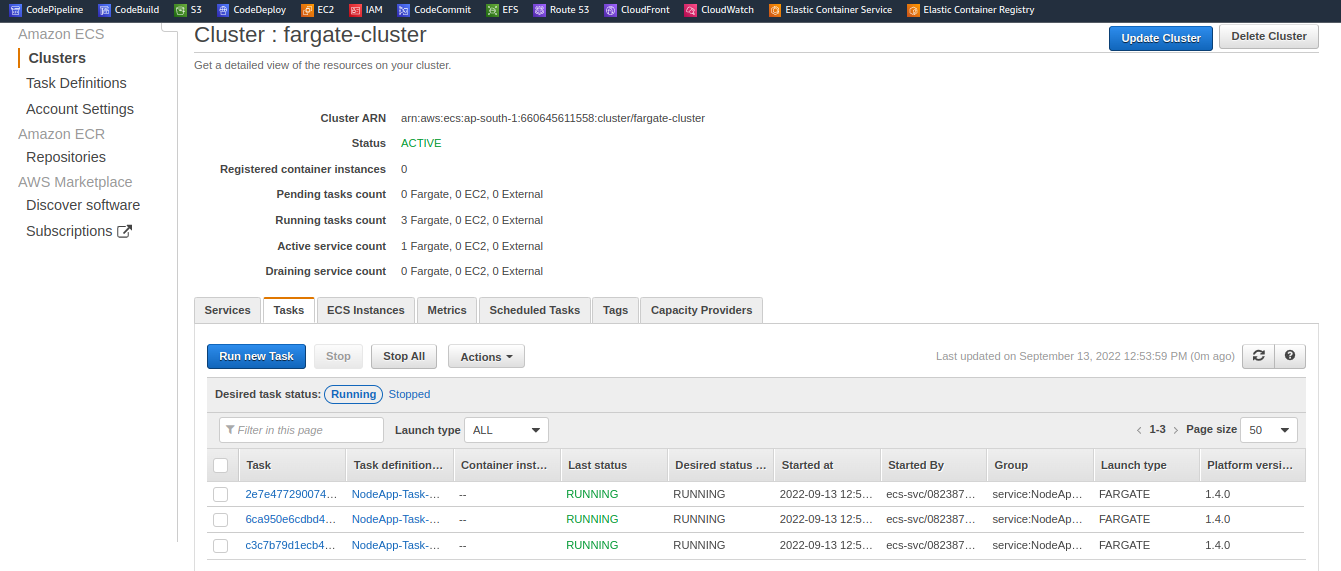


follow these commands ‘termminal’ under source directory,

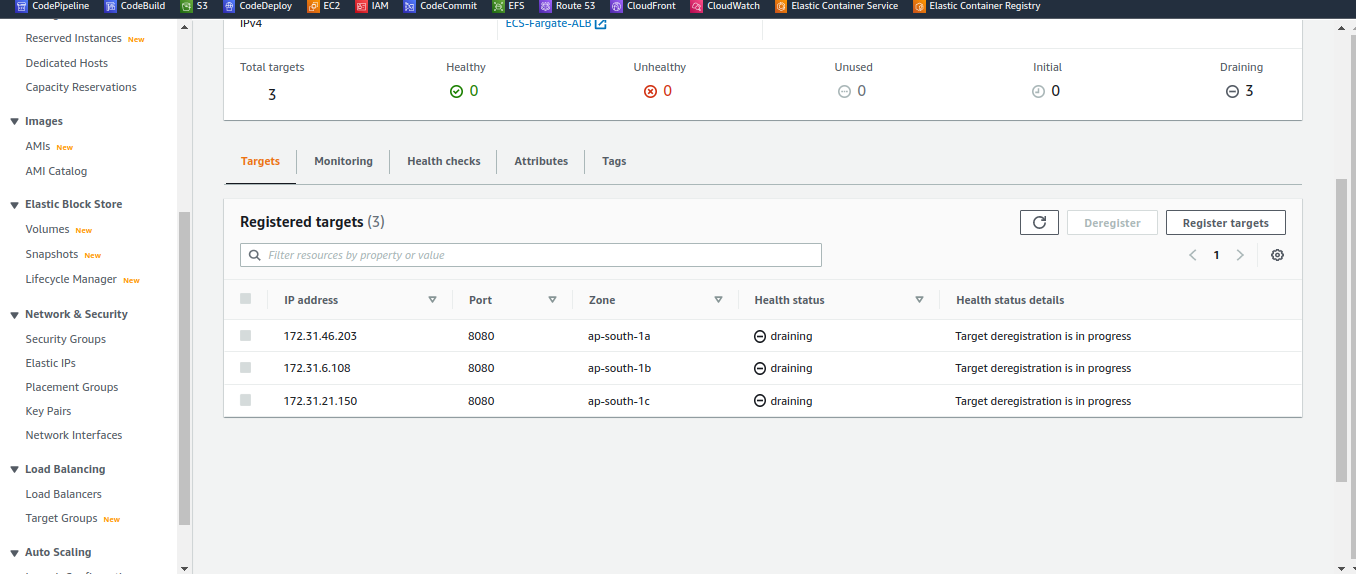


After done these stuff go & reload the ECR page, here we successfully pushed our DockerImages.

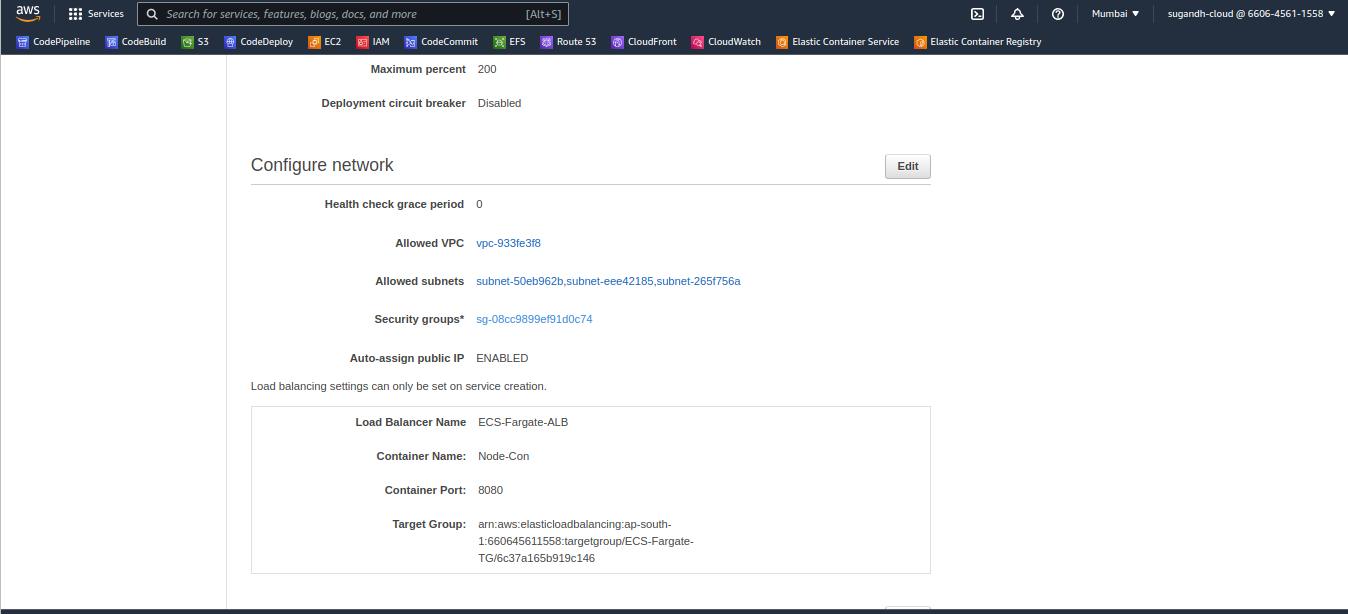
Now get back to ECS and refresh, they are in running state,



Now check one more things go back to EC2 ---> Load Balacer ----> Target Group ,



It create three tasks all those IP Addresses are added to the target group that target group in turn is linked with a Load Balancer so we can see all of them healthy now if you pickup DNS name & paste it on you fav browser, you will see, Application Successfully Depoyed In ECSFargate Cluster.



Done!