FULLSTACK APPLICATION DEPLOYMENT ON ECS-FARGATE THROUGH TERRAFORM

1. Overview of Three-Tier Architecture : Deploying an ECS task on private subnets ensures that your containers remain isolated from direct internet access while still being able to communicate internally within your VPC. Here's a step-by-step guide to deploying an ECS task in private subnets.

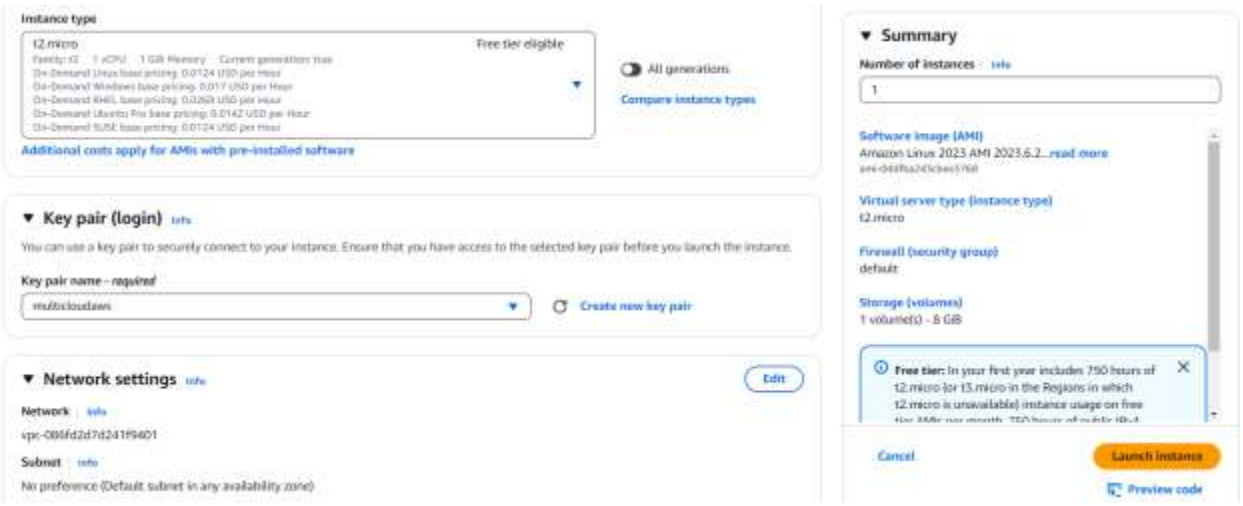
A three-tier architecture consists of:

1. **Presentation Layer (Frontend)**
   - This is the user interface (UI) of the application, such as a web or mobile frontend.
   - Deployed on **ECS (Fargate or EC2) running containers** with Nginx, Apache, or a React/Angular application.
2. **Application Layer (Backend)**
   - This layer handles business logic and processing.
   - Deployed on **ECS (Fargate or EC2) running containers** with Node.js, Java, Python, or another backend service.
3. **Data Layer (Database)**
   - A managed relational database, such as **Amazon RDS (MySQL, PostgreSQL, or Aurora).**
   - Stores and manages application data.

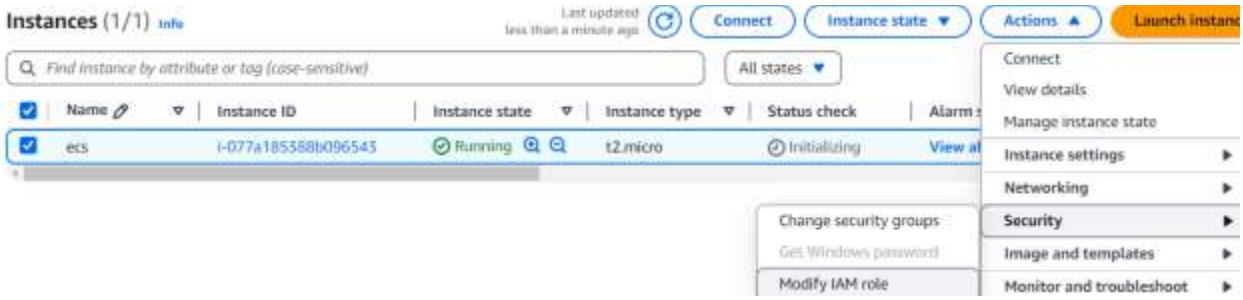
Key AWS Services Used

- **Amazon ECS (Fargate or EC2):** Manages containerized workloads.
- **Elastic Load Balancer (ALB):** Routes traffic to frontend/backend services.
- **AWS RDS:** Managed relational database service.
- **Amazon VPC:** Segregates frontend, backend, and database tiers into subnets.
- **IAM & Security Groups:** Controls access between different layers.

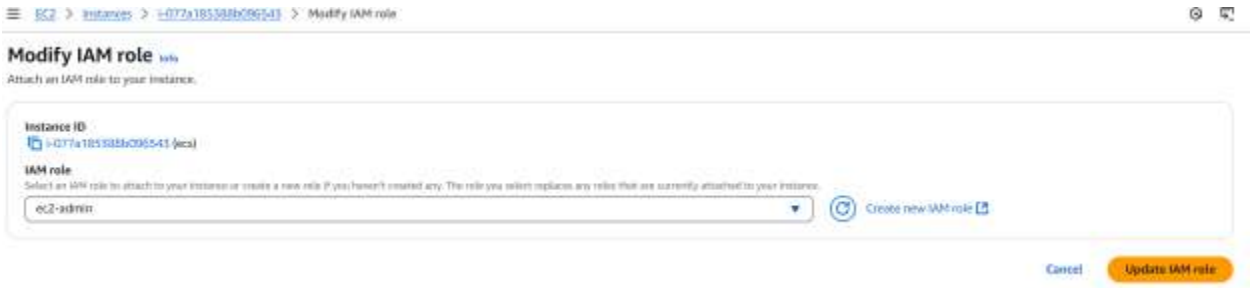
- Create a ec2



- Attach Iam role to ec2



- Update the role



- Connect to the ec2

```
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-172-31-7-58 ~]$
```

- Install the git and docker by following commands

sudo yum install git -y

sudo yum install docker -y #linux 2023

sudo usermod -aG docker ec2-user

newgrp docker

sudo service docker start

sudo chmod 777 /var/run/docker.sock

```
[ec2-user@ip-172-31-7-58 ~]$ sudo yum install git -y
sudo yum install docker -y #linux 2023
sudo usermod -aG docker ec2-user
newgrp docker
sudo service docker start
sudo chmod 777 /var/run/docker.sock
```

- Clone the git repo for docker image build purpose

git clone https://github.com/CloudTechDevOps/2nd10WeeksofCloudOps-main.git

```
[ec2-user@ip-172-31-7-58 ~]$ git clone https://github.com/CloudTechDevOps/2nd10WeeksofCloudOps-main.git
Cloning into '2nd10WeeksofCloudOps-main'...
remote: Enumerating objects: 405, done.
remote: Counting objects: 100% (250/250), done.
remote: Compressing objects: 100% (109/109), done.
remote: Total 405 (delta 213), reused 141 (delta 141), pack-reused 155 (from 1)
Receiving objects: 100% (405/405), 3.66 MiB | 4.76 MiB/s, done.
Resolving deltas: 100% (239/239), done.
[ec2-user@ip-172-31-7-58 ~]$ ls
2nd10WeeksofCloudOps-main
[ec2-user@ip-172-31-7-58 ~]$
```

- Switch to client directory

cd 2nd10WeeksofCloudOps-main/client

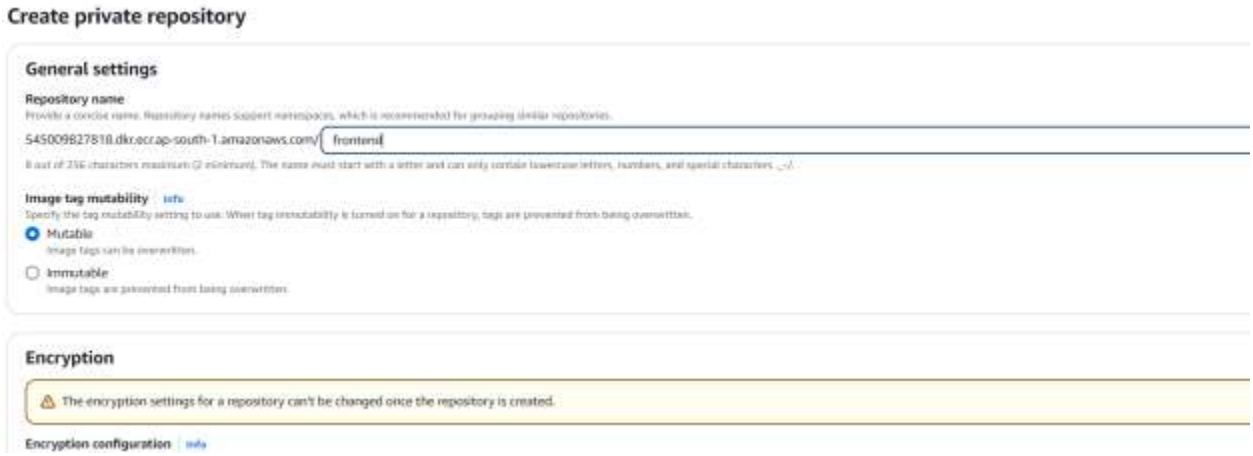
```
[ec2-user@ip-172-31-7-58 ~]$ cd 2nd10WeeksofCloudOps-main/client/
[ec2-user@ip-172-31-7-58 client]$ ls
Dockerfile  package-lock.json  package.json  public  src
[ec2-user@ip-172-31-7-58 client]$
```

- Change the api base url according to the your requirement

vi src/pages/config.js

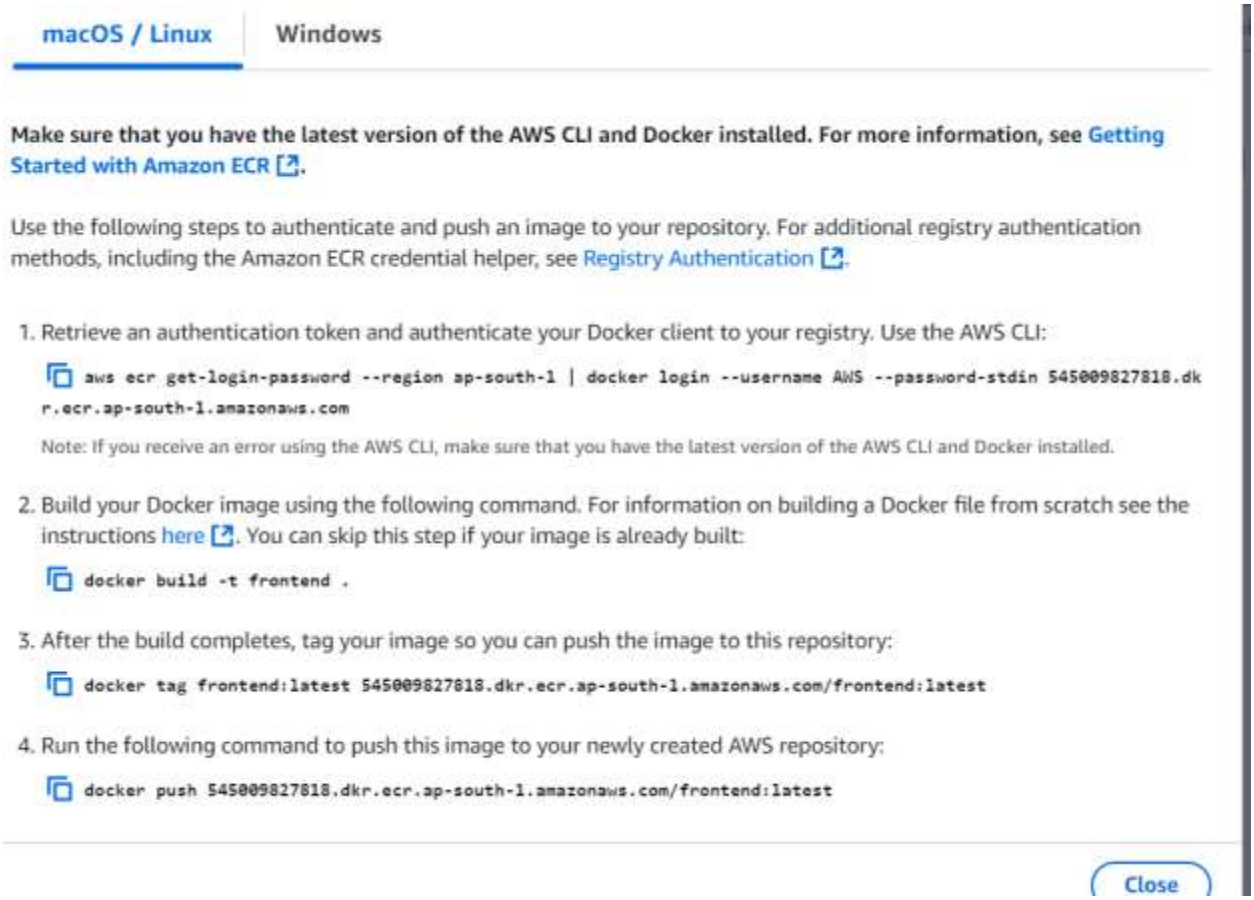
```
[ec2-user@ip-172-31-7-58 client]$ cat src/pages/config.js
// const API_BASE_URL = "http://localhost:8800";
const API_BASE_URL = "http://veera.narni.co.in";
export default API_BASE_URL;
[ec2-user@ip-172-31-7-58 client]$
```

- Create a ecr repository named as frontend



- Open the frontend ecr and click on **view push commands**

- Copy all commands paste it on ec2 one by one in client directory
- Copy the 1st login command and paste it on ec2 client directory



- Copy the 1st login command and paste it on ec2 client dirictory

- Copy the 2nd build command and paste it on ec2 client directory

```
[ec2-user@ip-172-31-7-58 client]$ docker build -t frontend .
[+] Building 0.9s (1/3)
```

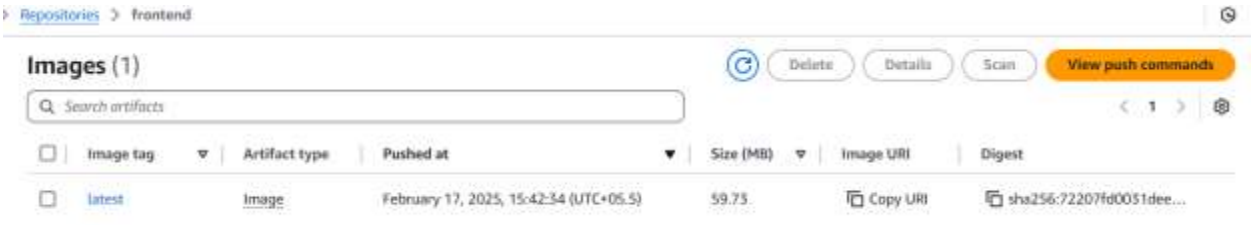
- Copy the 3rd tag command and paste it on ec2 client directory

```
[ec2-user@ip-172-31-7-58 client]$ docker tag frontend:latest 545009827818.dkr.ecr.ap-south-1.amazonaws.com/frontend:latest
[ec2-user@ip-172-31-7-58 client]$
```

- Copy the 4th push command and paste it on ec2 client directory

```
[ec2-user@ip-172-31-7-58 client]$ docker push 545009827818.dkr.ecr.ap-south-1.amazonaws.com/frontend:latest
The push refers to repository [545009827818.dkr.ecr.ap-south-1.amazonaws.com/frontend]
afa9a0ec131e: Pushed
e48683950315: Pushed
1b533b3f600d: Pushed
45d2a6f2a0b1: Pushed
5f70bf18a086: Pushed
d465f9c6793b: Pushed
7914c8f600f5: Pushed
latest: digest: sha256:72207fd0031deee427ffa317efb07d74dcaa1473a8efbfd82eb7d09f3733d990 size: 1782
[ec2-user@ip-172-31-7-58 client]$
```

- Frontend image is pushed into ecr successfully



- Switch to backend directory

```
[ec2-user@ip-172-31-7-58 client]$ cd ..
[ec2-user@ip-172-31-7-58 2nd10WeeksofCloudOps-main]$ cd backend/
[ec2-user@ip-172-31-7-58 backend]$ ls
Dockerfile  index.js  package-lock.json  package.json  test.sql
[ec2-user@ip-172-31-7-58 backend]$
```

- Create ecr repo for backend named as backend

- Copy all commands paste it on ec2 one by one in backend directory
- Copy the 1st login command and paste it on ec2 backend directory

Make sure that you have the latest version of the AWS CLI and Docker installed. For more information, see Getting Started with Amazon ECR.

Use the following steps to authenticate and push an image to your repository. For additional registry authentication methods, including the Amazon ECR credential helper, see Registry Authentication.

1. Retrieve an authentication token and authenticate your Docker client to your registry. Use the AWS CLI:

```
aws ecr get-login-password --region ap-south-1 | docker login --username AWS --password-stdin 545009827818.dkr.ecr.ap-south-1.amazonaws.com
```

   Note: If you receive an error using the AWS CLI, make sure that you have the latest version of the AWS CLI and Docker installed.

2. Build your Docker image using the following command. For information on building a Docker file from scratch see the instructions here. You can skip this step if your image is already built:

```
docker build -t backend .
```

3. After the build completes, tag your image so you can push the image to this repository:

```
docker tag backend:latest 545009827818.dkr.ecr.ap-south-1.amazonaws.com/backend:latest
```

4. Run the following command to push this image to your newly created AWS repository:

```
docker push 545009827818.dkr.ecr.ap-south-1.amazonaws.com/backend:latest
```

- Copy the 2nd build command and paste it on ec2 backend directory

```
[ec2-user@ip-172-31-7-58 backend]$ docker build -t backend .
[+] Building 0.3s (1/2)
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 738B
 => [internal] load metadata for docker.io/library/node:18
```

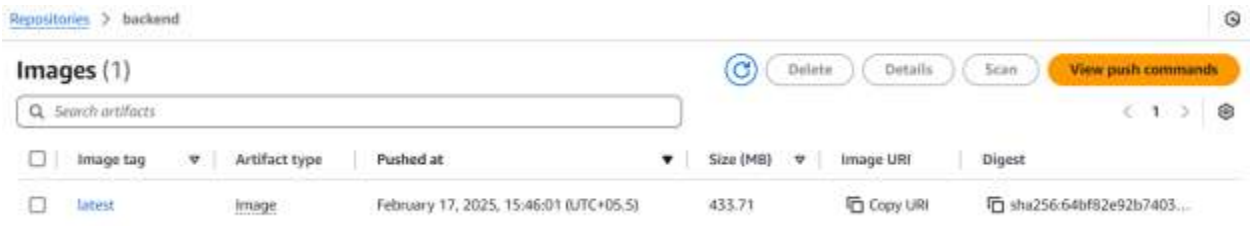
- Copy the 3rd tag command and paste it on ec2 backend directory

```
[ec2-user@ip-172-31-7-58 backend]$ docker tag backend:latest 545009827818.dkr.ecr.ap-south-1.amazonaws.com/backend:latest
[ec2-user@ip-172-31-7-58 backend]$
```

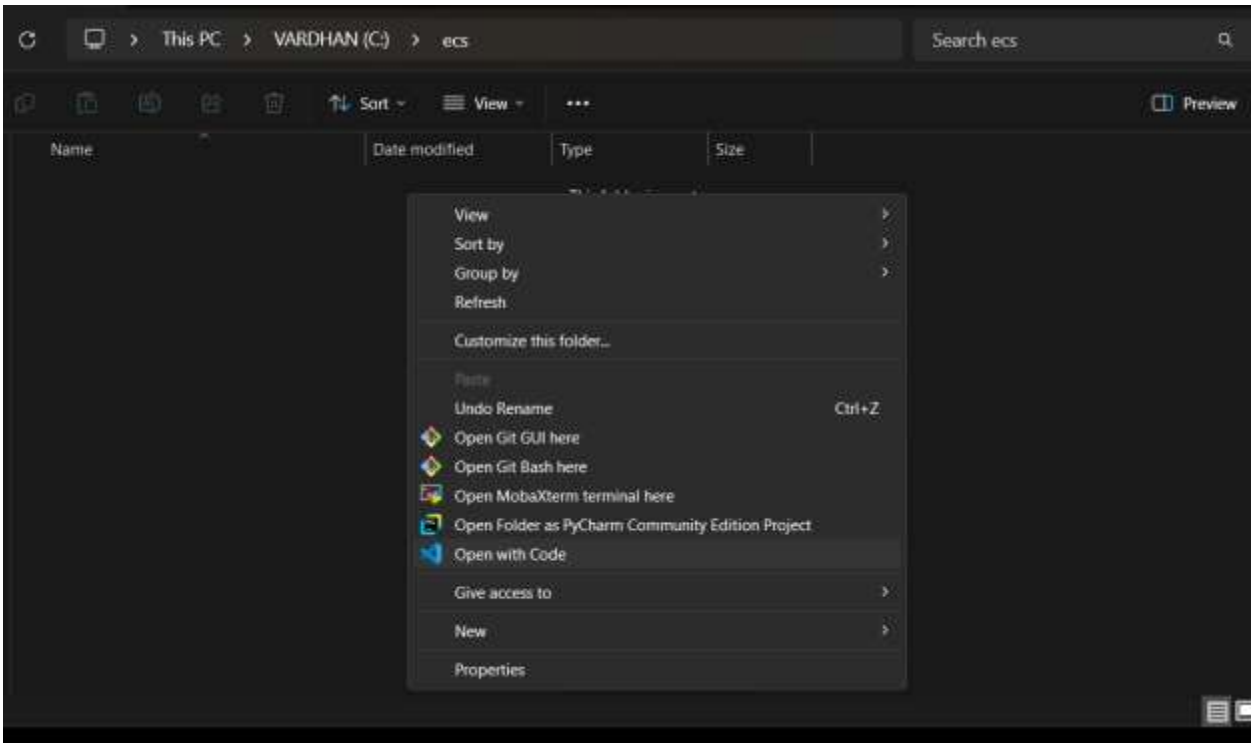
- Copy the 4th push command and paste it on ec2 backend directory

```
[ec2-user@ip-172-31-7-58 backend]$ docker push 545009827818.dkr.ecr.ap-south-1.amazonaws.com/backend:latest
The push refers to repository [545009827818.dkr.ecr.ap-south-1.amazonaws.com/backend]
7626fab5e78f: Pushed
b65677ed5817: Pushed
4b1963b1f5c4: Pushed
b54ab579bea8: Pushed
8d3309836362: Pushed
bcfce6d69a55: Pushed
f1f62a16530c: Pushed
48b86cff2099: Pushed
48cc25dc4131: Pushed
dafb8aed9f7f: Pushed
41d4dc7516bb: Pushed
c0f51bbdc37d: Pushed
91b542912d12: Pushed
latest: digest: sha256:64bf82e92b7403d2c66c1d3cfa2aeadd7523f2f47124e103db4817c9ebb9c551 size: 3052
[ec2-user@ip-172-31-7-58 backend]$
```

- Pushed the backend image to ecr successfully



- In your local drive create a folder
- Open that folder In vs code



- Configure the aws credentials in local macine by using aws configure
- Prerequisites for this process aws cli,terraform,vs code must be installed on your local machine
- After that Just clone the ecs terraform repo into your local

git clone https://github.com/CloudTechDevOps/FULLSTACK-ECS-FARGATE-THROUGH-TERRAFORMM.git

```
PS C:\ecs> git clone https://github.com/CloudTechDevOps/FULLSTACK-ECS-FARGATE-THROUGH-TERRAFORM.git
```

- Cloned the repo successfully

```
PS C:\ecs> git clone https://github.com/CloudTechDevOps/FULLSTACK-ECS-FARGATE-THROUGH-TERRAFORM.git
Cloning into 'FULLSTACK-ECS-FARGATE-THROUGH-TERRAFORM'...
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 8 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (8/8), done.
Resolving deltas: 100% (1/1), done.
PS C:\ecs>
```

- Switch to repo

cd FULLSTACK-ECS-FARGATE-THROUGH-TERRAFORMM

```
PS C:\ecs> cd .\FULLSTACK-ECS-FARGATE-THROUGH-TERRAFORM\
```

```
PROBLEMS   DEBUG CONSOLE   OUTPUT   TERMINAL   PORTS   CODE REFERENCE LOG
FULLSTACK-ECS-FARGATE-THROUGH-TERRAFORMM/

HONEY@honey MINGW64 ~/Documents/ECR-REPO
$ cd FULLSTACK-ECS-FARGATE-THROUGH-TERRAFORMM/

HONEY@honey MINGW64 ~/Documents/ECR-REPO/FULLSTACK-ECS-FARGATE-THROUGH-TERRAFORMM (main)
$ ls
ecs-task/  rds/  vpc-network/

HONEY@honey MINGW64 ~/Documents/ECR-REPO/FULLSTACK-ECS-FARGATE-THROUGH-TERRAFORMM (main)
$ cd vpc-network/

HONEY@honey MINGW64 ~/Documents/ECR-REPO/FULLSTACK-ECS-FARGATE-THROUGH-TERRAFORMM/vpc-network (main)
$
```

- **cd vpc-network**

  Run terraform init, terraform plan and terraform apply

  Then give cd ..

- **After change the "rds" Directory.**

  Run terraform init, terraform plan and terraform apply – **after creating a rds you should launch one manual server**

*install these commands on the server

sudo yum install git -y

sudo yum install mariadb105-server -y

```
[root@ip-172-31-2-231 ~]# yum install git -y
Amazon Linux 2023 Kernel Livepatch repository                                    136 kB/s |  14 kB     00:00
Dependencies resolved.
================================================================================================================
 Package                    Architecture        Version                          Repository              Size
================================================================================================================
Installing:
 git                        x86_64              2.47.1-1.amzn2023.0.2            amazonlinux             54 k
Installing dependencies:
 git-core                   x86_64              2.47.1-1.amzn2023.0.2            amazonlinux            4.7 M
 git-core-doc               noarch              2.47.1-1.amzn2023.0.2            amazonlinux            2.8 M
```

```
[ec2-user@ip-172-31-7-58 backend]$ sudo yum install mariadb105-server -y
```

After installilng the commands, you need to **git clone** the repository.

Git clone - https://github.com/CloudTechDevOps/2nd10WeeksofCloudOps-main.git

```
[root@ip-172-31-2-231 ~]# git clone https://github.com/CloudTechDevOps/2nd10WeeksofCloudOps-main.git
Cloning into '2nd10WeeksofCloudOps-main'...
remote: Enumerating objects: 405, done.
remote: Counting objects: 100% (250/250), done.
remote: Compressing objects: 100% (109/109), done.
remote: Total 405 (delta 213), reused 141 (delta 141), pack-reused 155 (from 1)
Receiving objects: 100% (405/405), 3.66 MiB | 16.28 MiB/s, done.
Resolving deltas: 100% (239/239), done.
[root@ip-172-31-2-231 ~]# ls
2nd10WeeksofCloudOps-main
[root@ip-172-31-2-231 ~]# cd 2nd10WeeksofCloudOps-main/
```

After that switch to the backend directory using the cd command.

```
[root@ip-172-31-2-231 ~]# cd 2nd10WeeksofCloudOps-main/
[root@ip-172-31-2-231 2nd10WeeksofCloudOps-main]# ls
Jenkins-Pipeline-Code  architecture.gif  client                          docker-compose.yaml  kubernetes-files  terraform_main_ec2
README.md              backend           docker-compose-process-note.md  eks-terraform        mysql
[root@ip-172-31-2-231 2nd10WeeksofCloudOps-main]# cd backend/
[root@ip-172-31-2-231 backend]#
```

After switching run the command mentioned below

mysql -h rds-endpoint -u admin -pveeranarni < test.sql

- Intilize database in backend directory. Before running the command check your project directory must be in backend directory

Come back to VS code then change the **ecs-task** directory. And add **rds endpoint** near the **db host**.

- Change the ecr image urls in the frontend,backend task files

```
FULLSTACK-ECS-FARGATE-THROUGH-TERRAFORM > backend-task-service.tf > resource "aws_ecs_task_definition" "back-task" > container_definitions
32  resource "aws_ecs_task_definition" "back-task" {
40    container_definitions = jsonencode([
42      name      = "backend"
43      image     = "545009827818.dkr.ecr.us-east-1.amazonaws.com/backend:latest"
44      cpu       = 256
45      memory    = 512
46      essential = true
47
48      portMappings = [
49        {
50          containerPort = 80
51          protocol      = "tcp"
52        }
53      ]
54
55      environment = [
56        { name = "DB_HOST", value = aws_db_instance.rds.address },
57        { name = "PORT", value = "3306" },
58        { name = "DB_USERNAME", value = "admin" },
59        { name = "DB_PASSWORD", value = "veeranarni" }
60      ]
61    }
62  ])

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS
PS C:\ecs\FULLSTACK-ECS-FARGATE-THROUGH-TERRAFORM>
```

- Change the image urls

```
FULLSTACK-ECS-FARGATE-THROUGH-TERRAFORM > frontend-task-service.tf > resource "aws_ecs_task_definition" "f
}

# ECS Task Definition
resource "aws_ecs_task_definition" "front_task" {
  family                   = "front-ecs-task"
  requires_compatibilities = ["FARGATE"]
  network_mode             = "awsvpc"
  memory                   = "512"
  cpu                      = "256"
  execution_role_arn       = aws_iam_role.ecs_task_execution_role.arn

  container_definitions = jsonencode([
    {
      name      = "frontend"
      image     = "545009827818.dkr.ecr.us-east-1.amazonaws.com/frontend:latest"
      cpu       = 256
      memory    = 512
      essential = true
      portMappings = [
        {
          containerPort = 80
          protocol      = "tcp"
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS
PS C:\ecs\FULLSTACK-ECS-FARGATE-THROUGH-TERRAFORM>
```

- Give the terraform init

```
PS C:\ecs\FULLSTACK-ECS-FARGATE-THROUGH-TERRAFORM> terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.87.0...
```

- Give the terraform paln

```
PS C:\ecs\FULLSTACK-ECS-FARGATE-THROUGH-TERRAFORM> terraform plan

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:
```

- Give the terraform apply

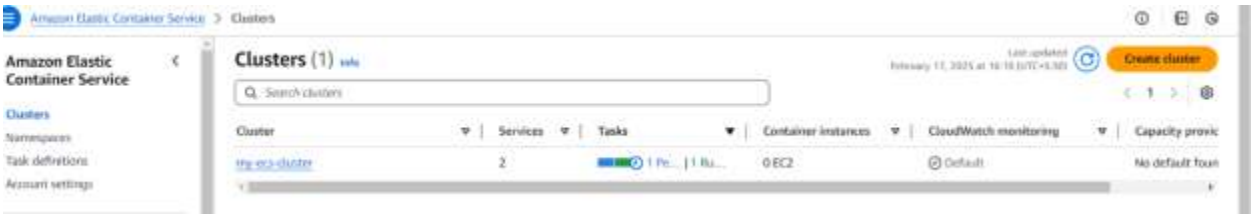
```
PS C:\ecs\FULLSTACK-ECS-FARGATE-THROUGH-TERRAFORM> terraform apply -auto-approve
```

- Resources created

```
aws_db_instance.rds: Still creating... [9m30s elapsed]
aws_db_instance.rds: Still creating... [9m40s elapsed]
aws_db_instance.rds: Still creating... [9m50s elapsed]
aws_db_instance.rds: Still creating... [10m0s elapsed]
aws_db_instance.rds: Still creating... [10m10s elapsed]
aws_db_instance.rds: Still creating... [10m20s elapsed]
aws_db_instance.rds: Still creating... [10m30s elapsed]
aws_db_instance.rds: Still creating... [10m40s elapsed]
aws_db_instance.rds: Still creating... [10m50s elapsed]
aws_db_instance.rds: Still creating... [11m0s elapsed]
aws_db_instance.rds: Still creating... [11m10s elapsed]
aws_db_instance.rds: Still creating... [11m20s elapsed]
aws_db_instance.rds: Still creating... [11m30s elapsed]
aws_db_instance.rds: Still creating... [11m40s elapsed]
aws_db_instance.rds: Still creating... [11m50s elapsed]
aws_db_instance.rds: Still creating... [12m0s elapsed]
aws_db_instance.rds: Still creating... [12m10s elapsed]
aws_db_instance.rds: Still creating... [12m20s elapsed]
aws_db_instance.rds: Still creating... [12m30s elapsed]
aws_db_instance.rds: Still creating... [12m40s elapsed]
aws_db_instance.rds: Still creating... [12m50s elapsed]
aws_db_instance.rds: Creation complete after 12m52s [id=db-X4CP3GGOSPARISHQDGXDWVHSEI]
aws_ecs_task_definition.back-task: Creating...
aws_ecs_task_definition.back-task: Creation complete after 0s [id=backend]
aws_ecs_service.back-ecs_service: Creating...
aws_ecs_service.back-ecs_service: Creation complete after 1s [id=arn:aws:ecs:ap-south-1:545009827818:service/my-ecs-cluster/backend-ecs-service]

Apply complete! Resources: 25 added, 0 changed, 0 destroyed.

Outputs:

rds_address = "book-rds.c1u4kewc6r37.ap-south-1.rds.amazonaws.com"
```

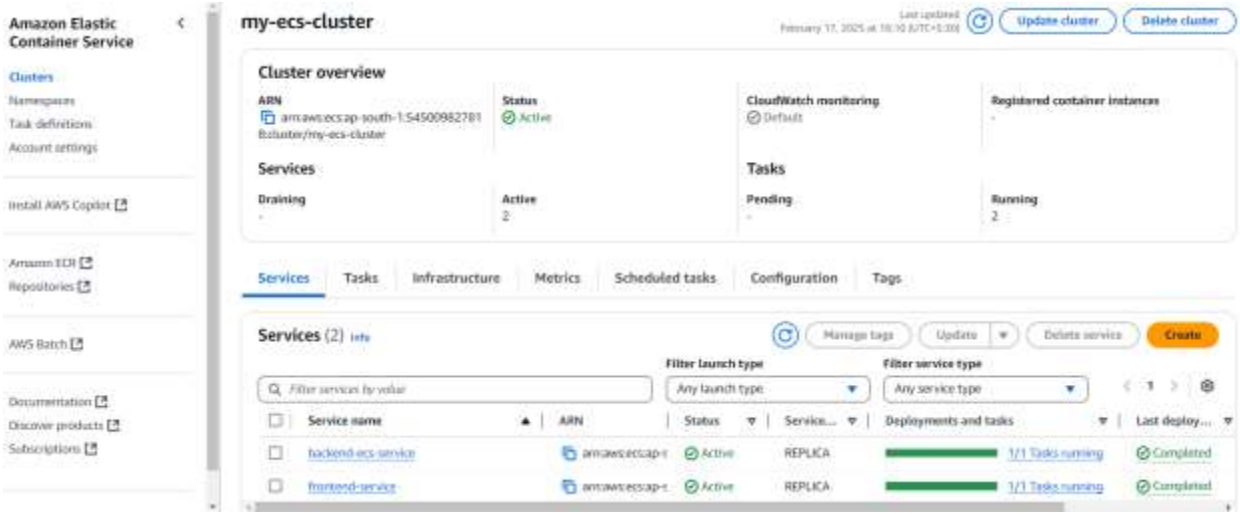
- Open the console and check the load balancers

- Open the ecs check the cluster

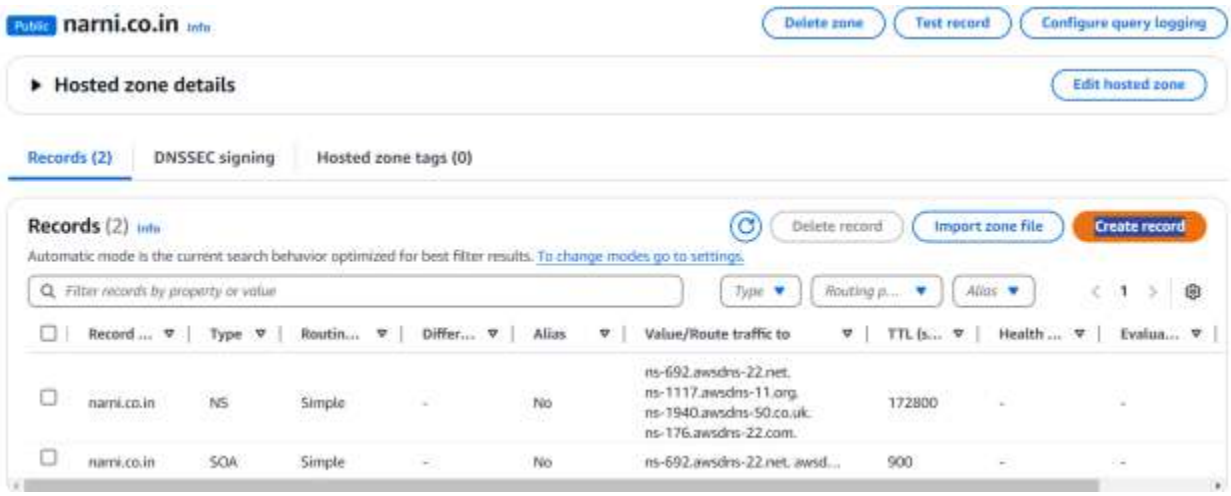


- check the task definitions

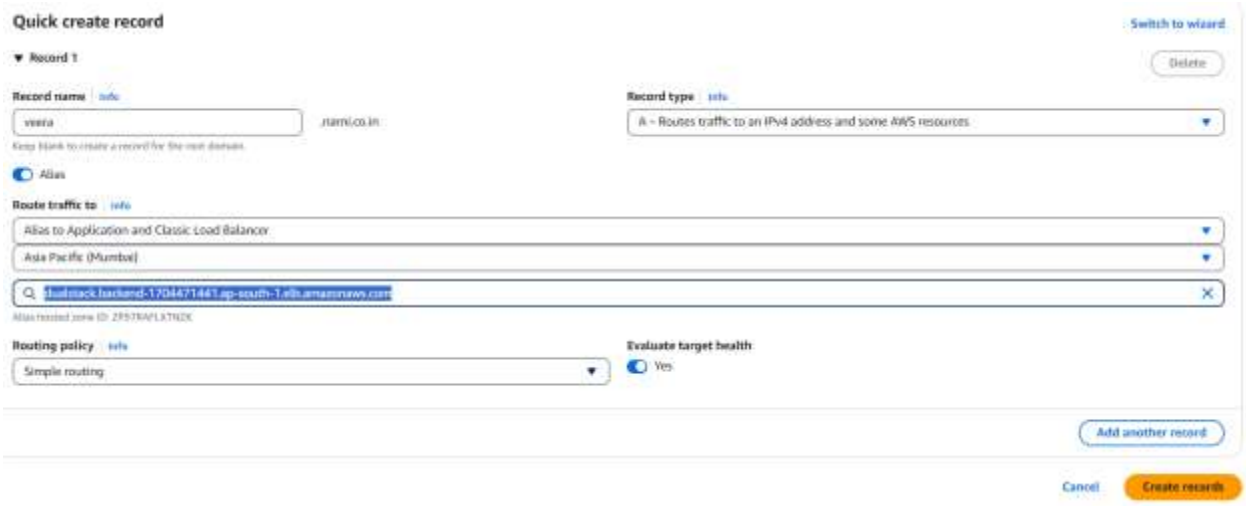
- Open your cluster and check the task status



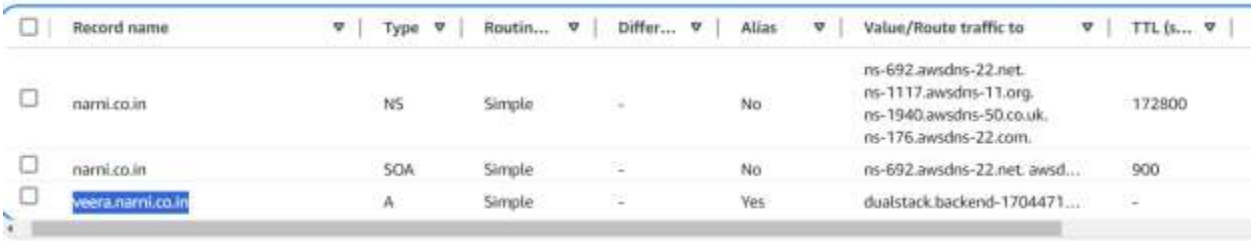
- Open your domain name
- Click on create record



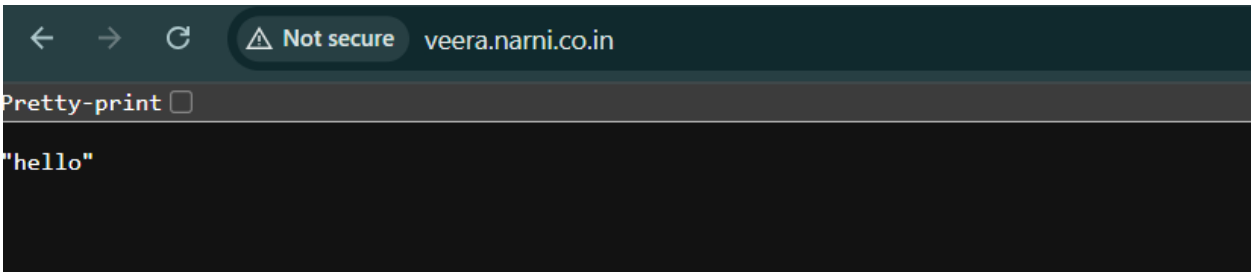
- Give sub domain as per your config.js value
- Select alias
- Select application and classic loadbalancer
- Select load balancer region
- Select your backend load balancer
- Create the record



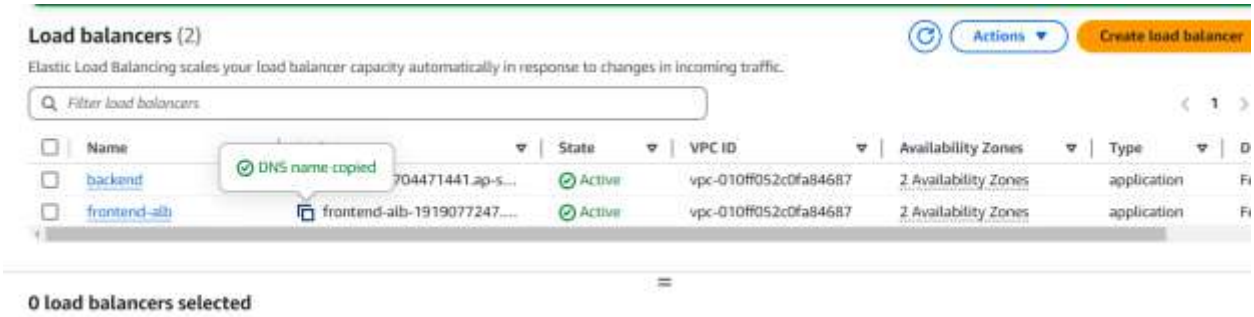
- Copy the record name



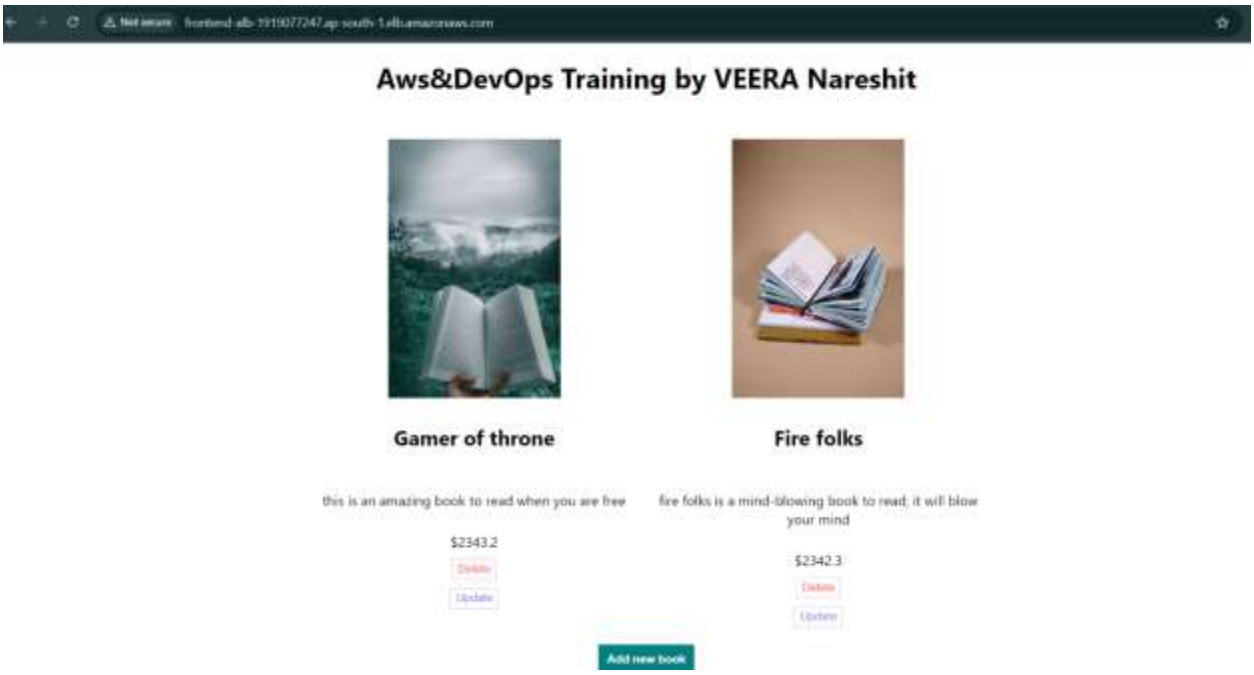
- Search on browser you will get hello response



- Copy the frontend loadblancer url



- Search on web you will get final out put
- Click on add book



----------THANK YOU----------