ENTREGA FINAL

1. Escolha da Ferramenta de IaC

Terraform foi escolhido, em vez de **CloudFormation**, por várias razões:

- **Multi-Provider Support**: O Terraform é perfeito em relação ao provedor, permitindo que você gerencie recursos na AWS, Azure, Google Cloud, entre outros, usando a mesma linguagem.
- **Estado Declarativo**: O Terraform mantém um estado do ambiente, permitindo visualizações claras e atualizações incrementais.
- **Módulos Reutilizáveis**: Os módulos do Terraform permitem uma organização e reutilização facilitadas do código.

2. Recursos Criados na Configuração

a. Cluster ECS

Um cluster ECS (`aws_ecs_cluster`), que é o agrupamento lógico de serviços e tarefas que você executa usando o Amazon ECS (Elastic Container Service). Este cluster é o ponto de partida para executar contêineres.

b. Task Definition

A definição da tarefa (`aws_ecs_task_definition`) especifica o contêiner que será executado, incluindo a imagem Docker, as variáveis de ambiente e os recursos (CPU e memória) necessários. Aqui eu indiquei a imagem do meu aplicativo.

```
resource "aws_ecs_task_definition" "task_definition" {
  family                   = "meu-app"
  requires_compatibilities = ["FARGATE"]  # Indica que a tare
fa usa Fargate
  network_mode             = "awsvpc"      # Necessário para
```

```
Fargate

  container_definitions = jsonencode([{
    name      = "meu-container"
    image     = "minha-imagem"  # URL da imagem Docker
    memory    = "512"  # Memória alocada
    cpu       = "256"  # CPU alocada
    essential = true

    portMappings = [{
      containerPort = 80  # Porta do contêiner
      hostPort      = 80  # Para Fargate, o hostPort é sempre
igual ao containerPort
      protocol      = "tcp"
    }]
  }])
}
```

c. Load Balancer

Eu configurei um Load Balancer (`aws_lb`), que serve para distribuir o tráfego entre instâncias ou serviços (neste caso, contêineres do ECS), garantindo alta disponibilidade e distribuição de carga.

- O Load Balancer também pode ser usado para monitorar a saúde dos contêineres, retirando do tráfego aqueles que não respondem conforme esperado.

```
resource "aws_lb" "application_load_balancer" {
  name               = "meu-load-balancer"
  internal           = false
  load_balancer_type = "application"
  security_groups    = [aws_security_group.lb_security_group.
id]
  subnets            = [
    "subnet-XXXXXXXX",
```

```
    "subnet-YYYYYYYY"
  ]
}
```

d. Serviço ECS

O serviço ECS (`aws_ecs_service`) é responsável por garantir que uma quantidade especificada de tarefas esteja sempre em execução. Isso significa que se algum contêiner falhar, o serviço ECS iniciará automaticamente novas instâncias.

- Foi configurado o serviço para utilizar o Load Balancer recém-criado:

```
resource "aws_ecs_service" "service" {
  name            = "meu-servico-ecs"
  cluster         = aws_ecs_cluster.cluster.id
  task_definition = aws_ecs_task_definition.task_definition.a
rn
  desired_count   = 1
  launch_type     = "FARGATE"  # Usa Fargate para gerenciamen
to serverless

  network_configuration {
    subnets          = ["subnet-VALIDO1", "subnet-VALIDO2"]
    security_groups  = [aws_security_group.lb_security_group.
id]
    assign_public_ip = "ENABLED"  # Atribui IPs públicos para
acessar o serviço
  }

  load_balancer {
    target_group_arn = aws_lb_target_group.target_group.arn
    container_name   = "meu-container"
    container_port   = 80
  }
}
```

3. Por que usar Amazon ECS em vez de EC2?

Aqui estão algumas razões pelas quais eu optei pelo Amazon ECS (especificamente Fargate) em vez de instâncias EC2 tradicionais:

- **Gerenciamento de Contêineres**: O ECS é otimizado para executar e gerenciar contêineres. Ele permite que você empacote seu aplicativo em contêineres e escale facilmente com base na demanda.
- **Sem Gerenciamento de Infraestrutura**: Ao optar pelo Amazon ECS, especialmente ao utilizar o Fargate, você não precisa se preocupar com o gerenciamento das instâncias subjacentes. O Fargate cuida disso para você, permitindo que você se concentre no código e na lógica de negócios, em vez de na infraestrutura. Isso significa não ter que provisionar, monitorar ou gerenciar servidores EC2 manualmente.
- **Escalabilidade Automática**: O ECS permite escalabilidade automatizada com base na demanda da aplicação. Você pode definir políticas para que o ECS inicie automaticamente mais contêineres ou interrompa contêineres com menos tráfego, permitindo que você otimize custos e recursos.
- **Erros e Recuperação**: O ECS proporciona um gerenciamento mais robusto de contêineres. Se um contêiner falhar, o ECS pode reiniciá-lo automaticamente, garantindo que a quantidade desejada de instâncias esteja sempre ativa e funcional.
- **Integração com Outros Serviços da AWS**: O ECS se integra facilmente a outros serviços da AWS, como o CloudWatch para monitoramento, o IAM para controle de acesso e o Route 53 para gerenciamento de DNS. Essa integração facilita a criação de uma arquitetura de microserviços altamente confiável e escalável.
- **Implantação Rápida e consistente**: Com a definição do serviço e da tarefa no ECS, você pode implantar atualizações de contêiner rapidamente e de maneira consistente. Isso reduz a janela de tempo para colocar novas versões de software em produção.
- **Custo Eficiente**: Quando você utiliza o Fargate com ECS, você paga apenas pelos recursos que usa durante a execução dos contêineres, sem necessidade de pagar por instâncias EC2 o tempo todo. Isso pode reduzir

significativamente os custos, pois você evita a cobrança de capacidade subutilizada.

4. Resumo do Código Criado

- **Estrutura Modularizada**: O código foi estruturado para separar diferentes componentes da infraestrutura, como a definição da rede, o cluster ECS, a task definition e o equilíbrio de carga. Isso melhora a legibilidade e a manutenção do código.
- **Configurações de Rede**: Configurações de VPC, sub-redes e grupos de segurança foram criadas para isolar e proteger os recursos do ECS, garantindo que suas instâncias apenas aceitem tráfego de fontes autorizadas.
- **Deploy Visualizável e Reversível**: O Terraform fornece o comando `terraform plan`, que mostra quais mudanças serão aplicadas antes de fazer qualquer modificação no ambiente. Isso reduz o risco de erros durante o provisionamento e facilita a reversibilidade, caso você precise voltar a uma configuração anterior.
- **Configurabilidade**: O código é altamente configurável, permitindo que você ajuste rapidamente parâmetros como o número de instâncias, a imagem do contêiner, a configuração de CPU e memória e as portas expostas.

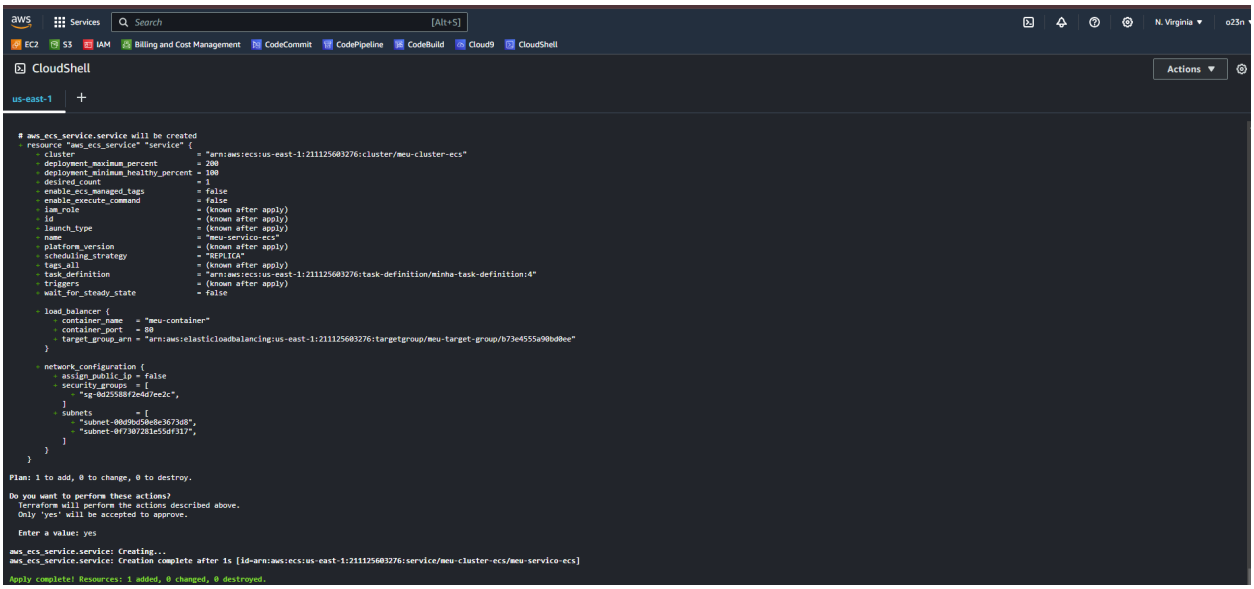
5. Considerações Finais

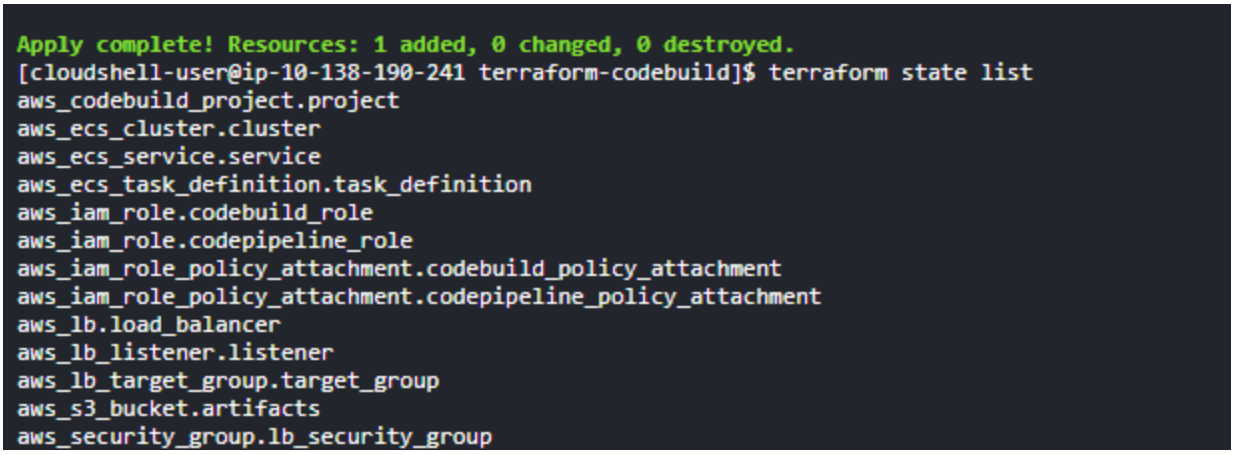
Ao optar pela implementação de microserviços com o Amazon ECS, você está garantindo uma estrutura mais moderna, escalável e menos dependente de hardware específico. O ECS fornece um ambiente capaz de lidar com aplicações que exigem flexibilidade e resiliência, ajudando a manter a eficiência operacional.

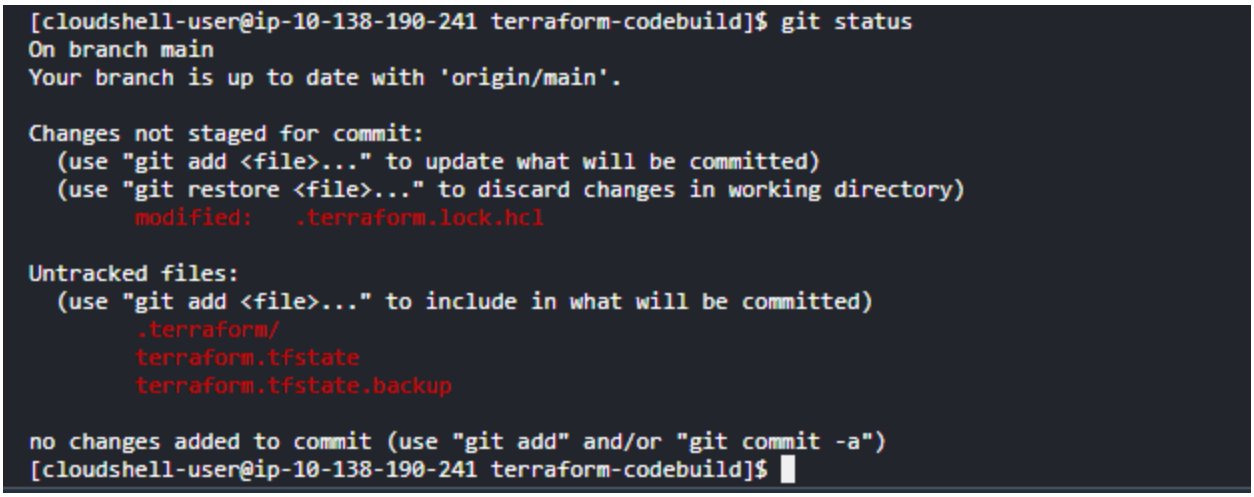
Uma vez que seu código esteja em execução, monitore continuamente utilizando o AWS CloudWatch e adapte sua infraestrutura conforme necessário para otimizar a performance e os custos.

Essas práticas garantirão que você utilize ao máximo o potencial do Amazon ECS e da AWS, permitindo que seus aplicativos sejam altamente disponíveis, escaláveis e de fácil gestão.

Retorno do terraform apply e exibindo que o codigo está versionado no cloudshell;







1. Criação do Bucket S3 para Armazenar Artefatos

O bucket S3 (`aws_s3_bucket`) é um repositório de armazenamento na nuvem que você está criando para armazenar os artefatos gerados pelo processo de build no AWS CodeBuild. Aqui está a parte do código relevante:

hcl

```
# Criação do bucket do S3 para armazenar os artefatos
resource "aws_s3_bucket" "artifacts" {
  bucket = "meu-bucket-de-artefatos"
}
```

- **Nome do Bucket**: O nome do bucket precisa ser único globalmente; portanto, certifique-se de que "meu-bucket-de-artefatos" não seja utilizado por outra conta AWS.
- **Objetivo**: Este bucket será utilizado para armazenar artefatos gerados durante a execução do build, como binários, pacotes ou qualquer outro arquivo que você deseja que esteja disponível após a execução do CodeBuild.

2. Criação do Projeto do CodeBuild

O projeto do CodeBuild (`aws_codebuild_project`) é definido para compilar seu código e executar testes. Aqui está a parte do código que gerencia isso:

hcl

```
# Criação do projeto do CodeBuild
resource "aws_codebuild_project" "project" {
  name         = "meu-projeto-codebuild"
  description  = "Projeto do CodeBuild para o meu aplicativo"
  service_role = aws_iam_role.codebuild_role.arn

  artifacts {
    type = "NO_ARTIFACTS"
  }

  environment {
    compute_type = "BUILD_GENERAL1_SMALL"
    image        = "aws/codebuild/standard:4.0"
    type         = "LINUX_CONTAINER"
  }

  source {
    type            = "GITHUB"
    location        = "https://github.com/23Ant/terraform-cod
ebuild.git"
    git_clone_depth = 1
  }
}
```

Explicação dos Componentes:

- **name**: Nome do projeto do CodeBuild, neste caso, `"meu-projeto-codebuild"`.
- **description**: Uma breve descrição sobre o que este projeto faz.
- **service_role**: O ARN da role do IAM que permissões o CodeBuild pode usar para acessar outros serviços da AWS, como o S3, por exemplo.
- **artifacts**: Especifica o tipo de artefatos gerados pelo build. Aqui, está definido como `"NO_ARTIFACTS"` porque, neste caso, não há artefatos sendo gerados, mas

você pode alterar isso no futuro para incluir artefatos que deseja armazenar no bucket S3. Se você quiser armazenar saída de build, você mudaria para algo como:hcl

```
artifacts {
  type = "S3"
  location = aws_s3_bucket.artifacts.bucket
  packaging = "ZIP"
  path = "artefatos/"
}
```

- **environment**: Define o ambiente onde o build ocorrerá.
  - **compute_type**: O tipo de computação que você está utilizando. `"BUILD_GENERAL1_SMALL"` é uma especificação menor.
  - **image**: A imagem do Docker a ser utilizada para o build. `"aws/codebuild/standard:4.0"` corresponde a uma versão padronizada da ferramenta do build.
  - **type**: O tipo de ambiente; neste caso, estamos usando um contêiner Linux.
- **source**: Especifica a fonte do código que será construída.
  - **type**: O tipo de repositório de origem; aqui, você está usando o GitHub.
  - **location**: A URL do repositório GitHub que contém o código fonte.
  - **git_clone_depth**: Especifica a profundidade do clone. Um valor de `1` indica que você deseja apenas a última versão do repositório.

3. Criação da Role IAM para o CodeBuild

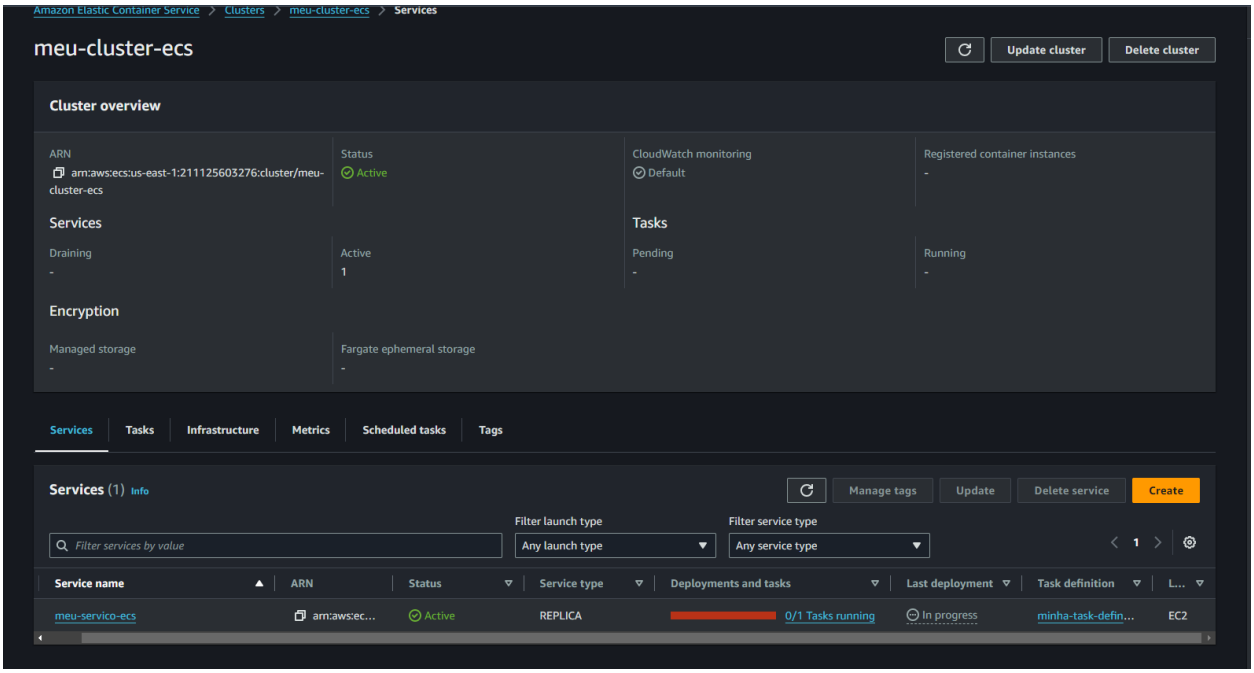
Além do projeto do CodeBuild e do bucket S3, você também está criando uma role IAM que o CodeBuild usará para obter as permissões necessárias para operar. Aqui está a parte do código:

hcl

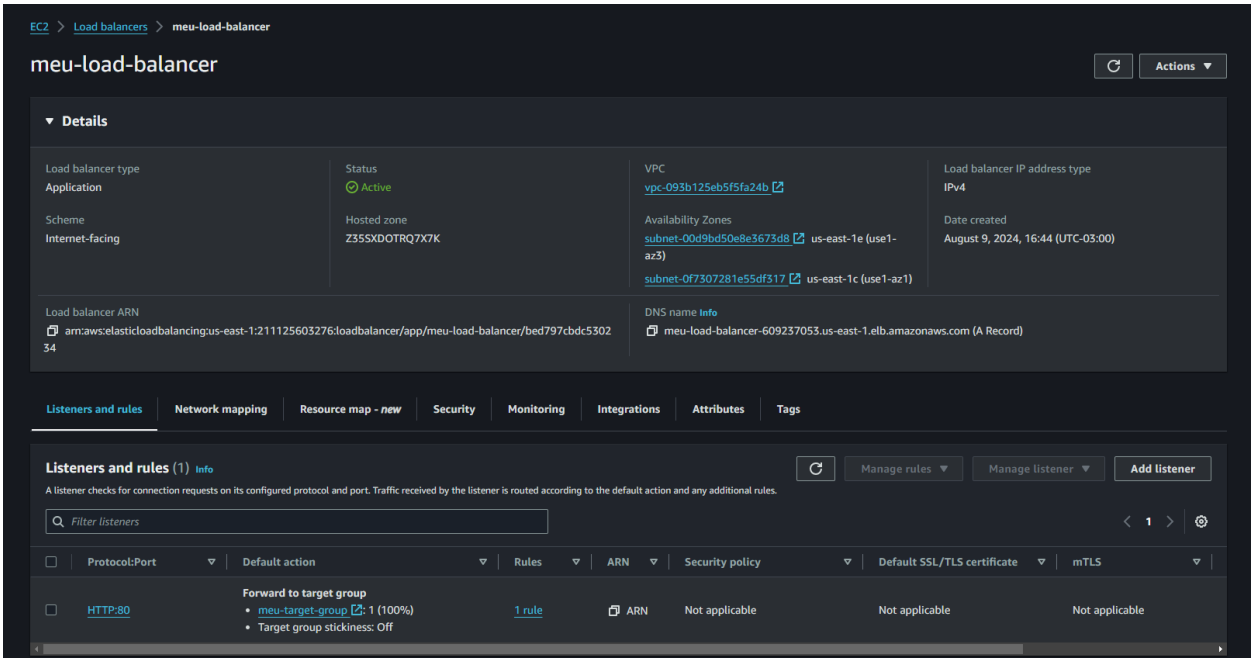
```
# Criação da role do IAM para o CodeBuild
resource "aws_iam_role" "codebuild_role" {
  name = "codebuild-role"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Action = "sts:AssumeRole"
        Effect = "Allow"
        Principal = {
          Service = "codebuild.amazonaws.com"
        }
      }
    ]
  })
}
```

Etapas criadas dentro da AWS

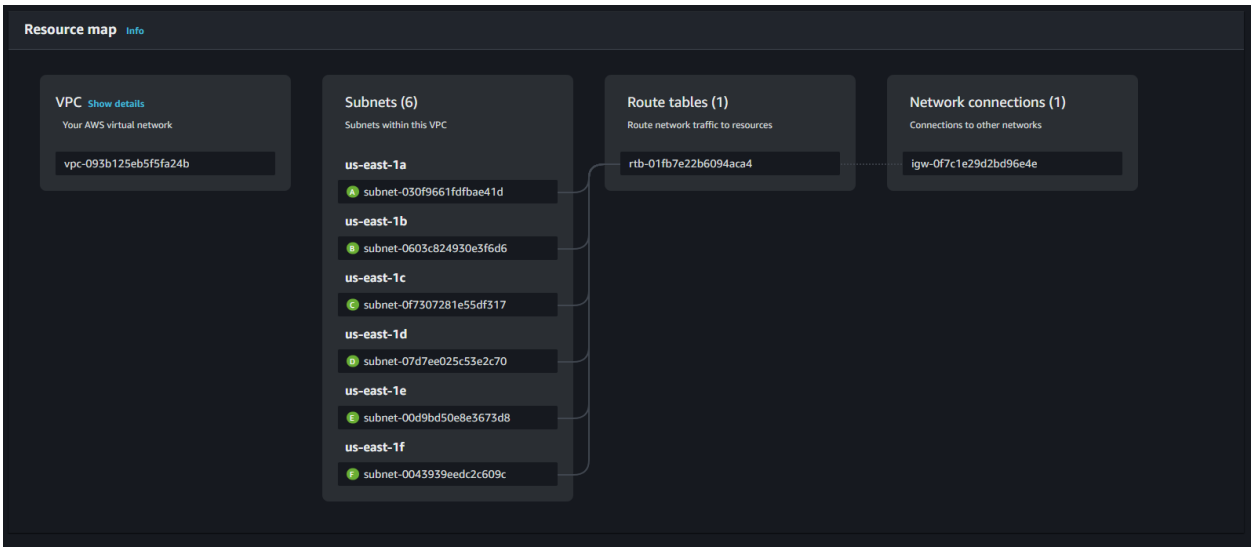
Feito a estrutura, estarei mostrando agora o ECS criado.



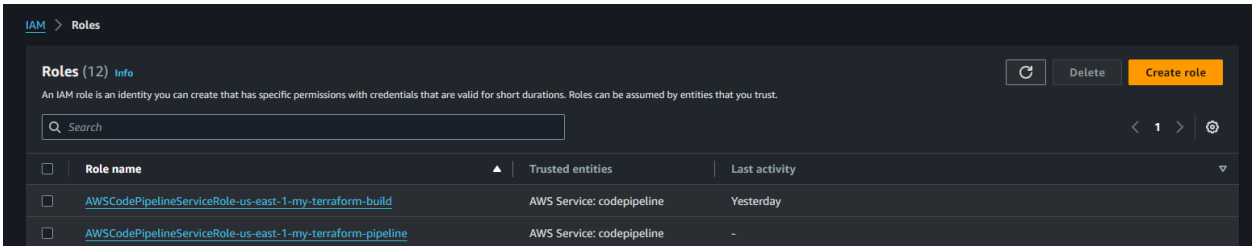
No LOADBALANCE a vpc, subnet definidas e criadas.



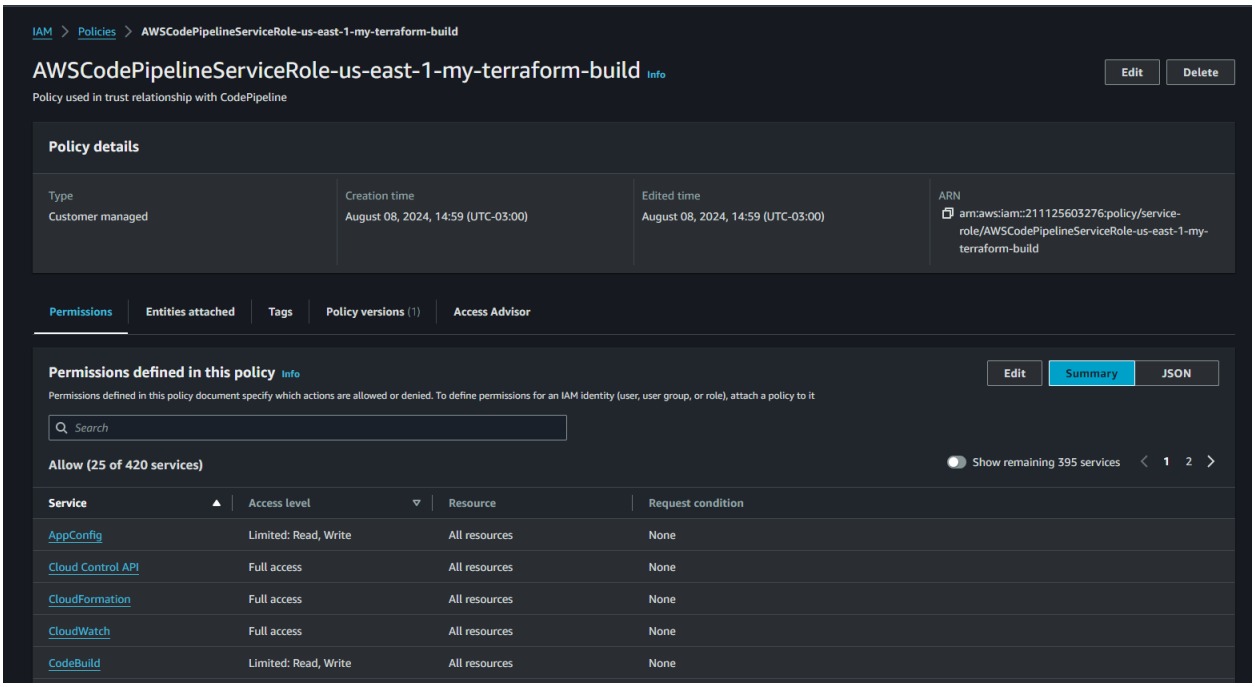
Resource Map da nossa rede;

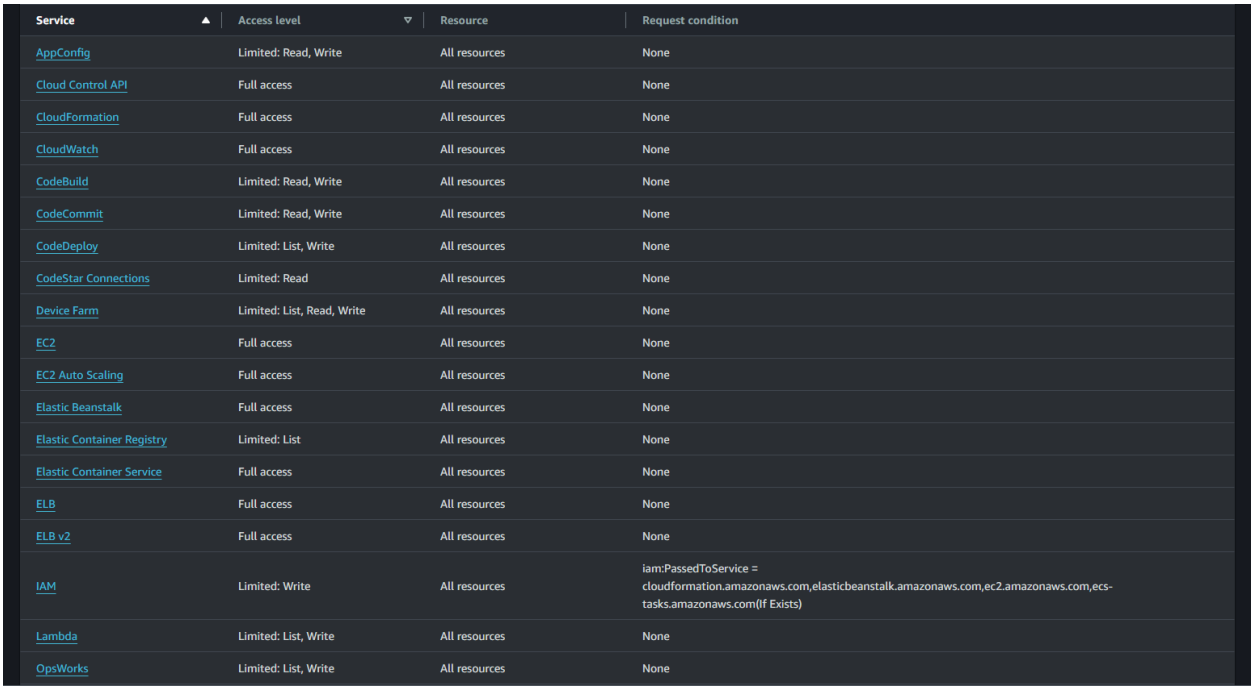


Dentro do IAM, Roles geradas.

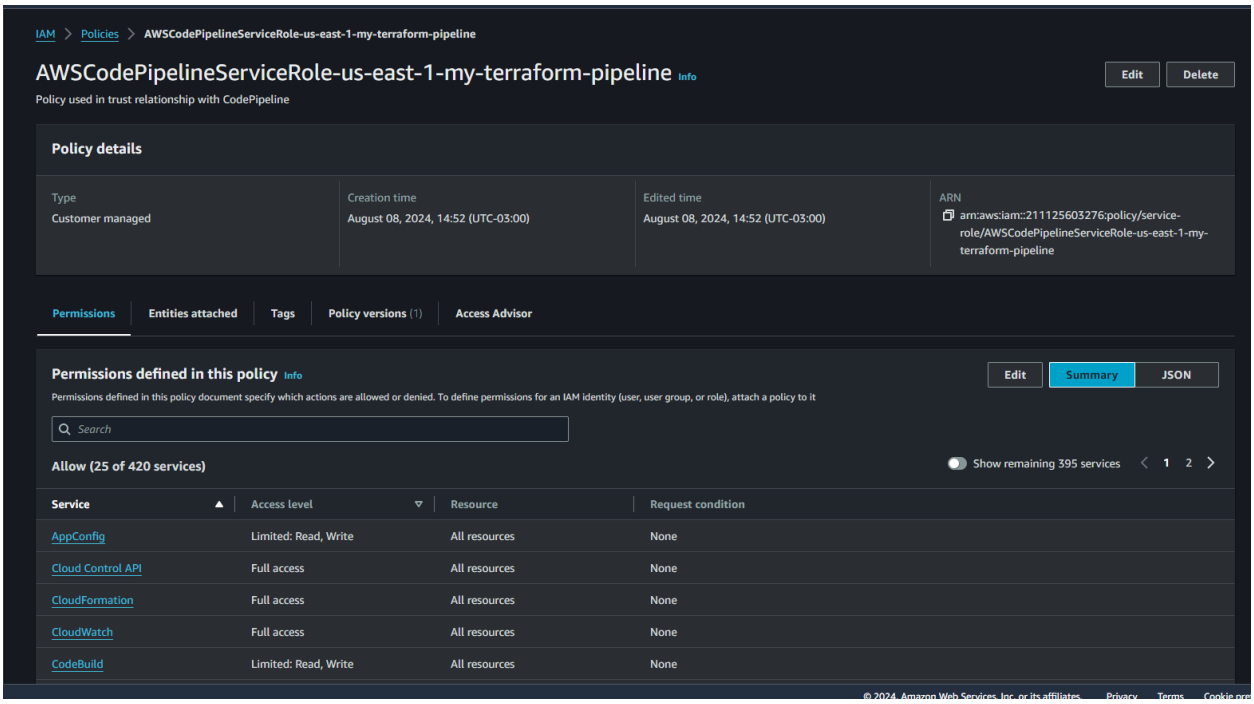


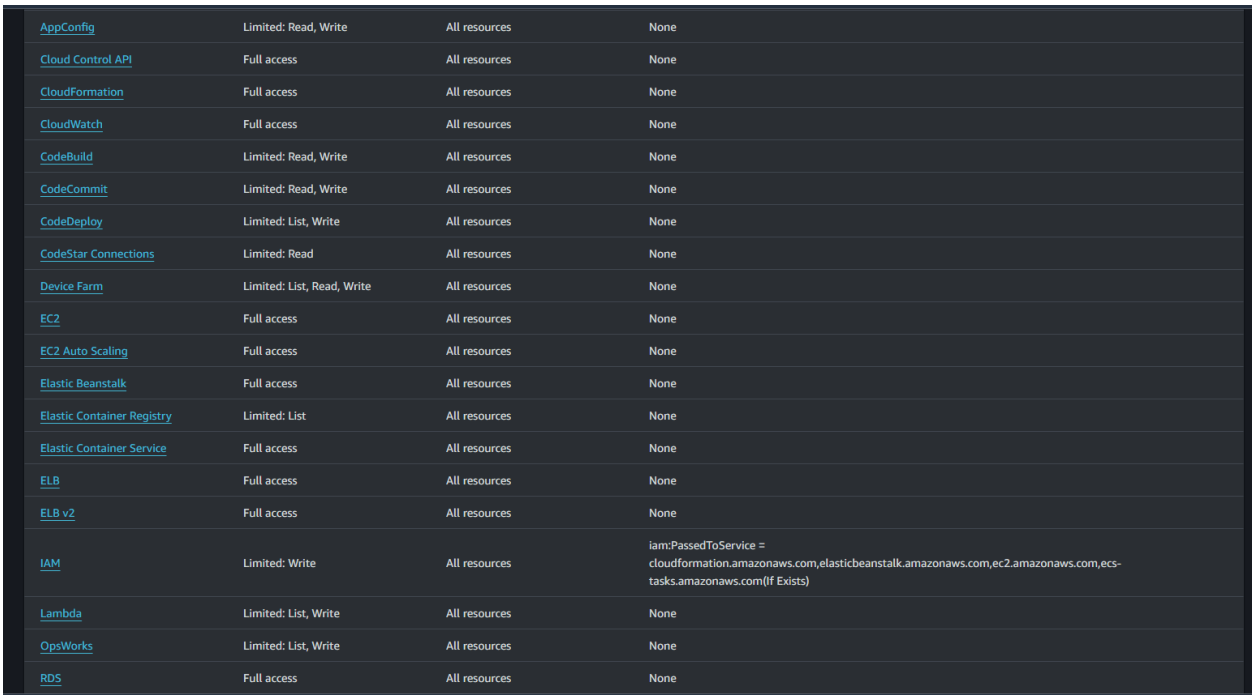
Policies de permissões codebuild, geradas.



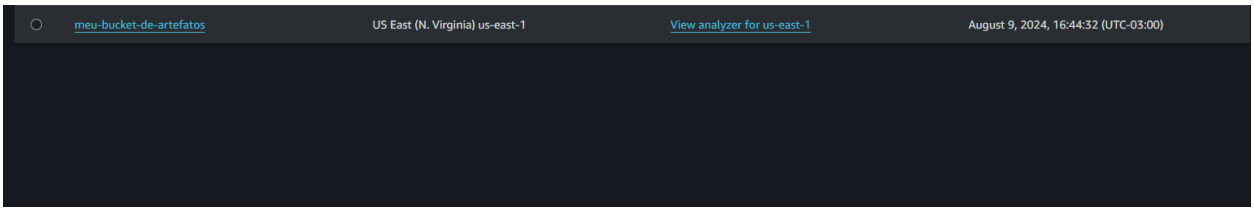


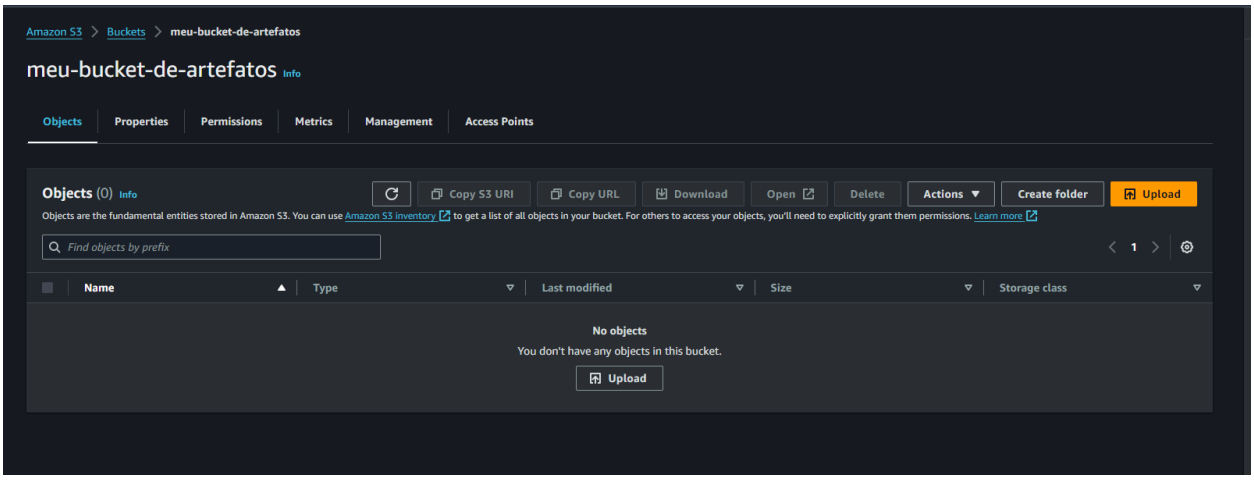
Policies de permissoes geradas codepipeline.



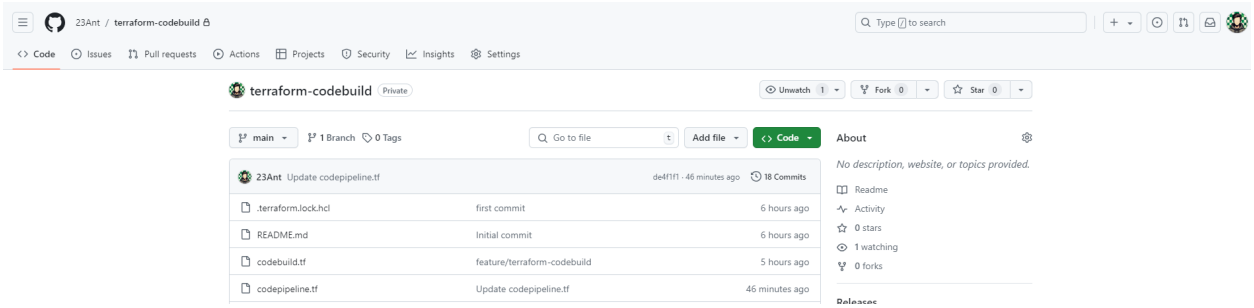


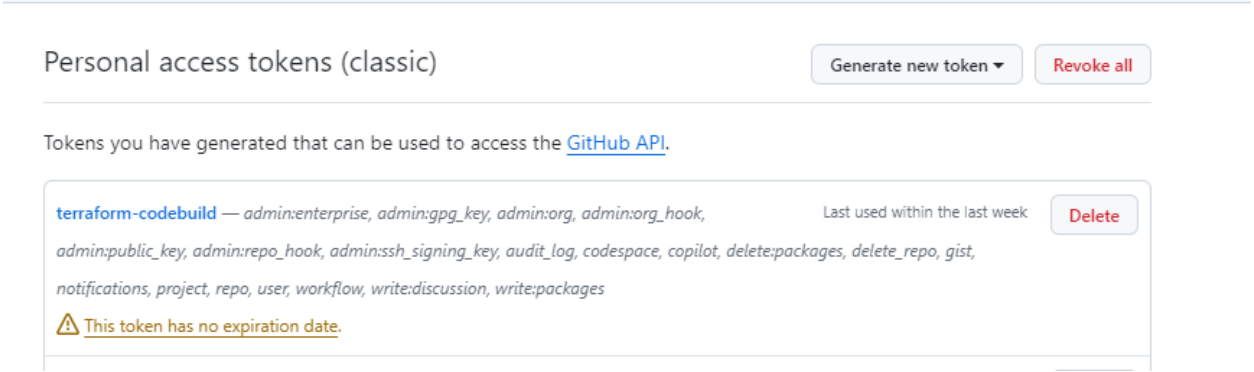
Bucket criado no s3 pelo codedeploy.





Plataforma versionada dentro do github, dando git clone com url de desenvolvedor setada no development settings.





CODIGO CRIADOS

codebuild.tf

Criação do projeto do CodeBuild

```
resource "aws_codebuild_project" "project" {
name        = "meu-projeto-codebuild"
description  = "Projeto do CodeBuild para o meu aplicativo"
service_role = aws_iam_role.codebuild_role.arn

artifacts {
type = "NO_ARTIFACTS"
}

environment {
compute_type = "BUILD_GENERAL1_SMALL"
image        = "aws/codebuild/standard:4.0"
type         = "LINUX_CONTAINER"
}

source {
type           = "GITHUB"
location       = "
https://github.com/23Ant/terraform-codebuild.git"
git_clone_depth = 1
}
}
```

Criação da role do IAM para o CodeBuild

```
resource "aws_iam_role" "codebuild_role" {
name = "codebuild-role"

assume_role_policy = jsonencode({
Version = "2012-10-17"
Statement = [
```

```
{
Action = "sts:AssumeRole"
Effect = "Allow"
Principal = {
Service = "
codebuild.amazonaws.com"
}
}
]
})
}
```

Anexação da política do IAM à role do CodeBuild

```
resource "aws_iam_role_policy_attachment" "codebuild_policy_attachment" {
policy_arn = "arn:aws:iam::aws:policy/AWSCodeBuildAdminAccess"
role       = aws_iam_role.codebuild_role.name
}
```

Criação do bucket do S3 para armazenar os artefatos

```
resource "aws_s3_bucket" "artifacts" {
bucket = "meu-bucket-de-artefatos"
}
```

---

codepipeline.tf

```
provider "aws" {
region = "us-east-1"
}
```

Criação da role do IAM para o CodePipeline

```
resource "aws_iam_role" "codepipeline_role" {
name = "codepipeline-role"

assume_role_policy = jsonencode({
Version = "2012-10-17"
Statement = [
{
Action = "sts:AssumeRole"
Effect = "Allow"
Principal = {
Service = "
codepipeline.amazonaws.com"
}
}
]
})
}
```

Anexação da política do IAM à role do CodePipeline

```
resource "aws_iam_role_policy_attachment" "codepipeline_policy_attachment" {
role       = aws_iam_role.codepipeline_role.name
policy_arn = "arn:aws:iam::aws:policy/AWSCodePipeline_FullAccess"
}
```

Criação do cluster do ECS

```
resource "aws_ecs_cluster" "cluster" {
name = "meu-cluster-ecs"
}
```

Criação da definição da tarefa do ECS

```
resource "aws_ecs_task_definition" "task_definition" {
family                = "minha-task-definition"
network_mode          = "awsvpc"
container_definitions = jsonencode([
{
name      = "meu-container"
image     = "minha-imagem"
cpu       = 256
memory    = 512
essential = true
portMappings = [
{
containerPort = 80
hostPort      = 80
protocol      = "tcp"
}
]
}
])
}
```

Criação do target group do load balancer

```
resource "aws_lb_target_group" "target_group" {
name        = "meu-target-group"
port        = 80
protocol    = "HTTP"
target_type = "ip"
vpc_id      = "vpc-093b125eb5f5fa24b"  # ID  DA MINHA VPC

health_check {
path                = "/"
interval            = 30
timeout             = 5
healthy_threshold   = 2
unhealthy_threshold = 2
```

```
protocol          = "HTTP"
}
}
```

Criação do load balancer

```
resource "aws_lb" "load_balancer" {
name               = "meu-load-balancer"
internal           = false
load_balancer_type = "application"
security_groups    = [aws_security_group.lb_security_group.id]

subnets            = [
"subnet-00d9bd50e8e3673d8",
"subnet-0f7307281e55df317"
]  # IDs das minhas subnets
}
```

Criação do listener do load balancer

```
resource "aws_lb_listener" "listener" {
load_balancer_arn = aws_lb.load_balancer.arn
port              = 80
protocol          = "HTTP"

default_action {
type             = "forward"
target_group_arn = aws_lb_target_group.target_group.arn
}
}
```

Criação do serviço do ECS

```
resource "aws_ecs_service" "service" {
name            = "meu-servico-ecs"
cluster         = aws_ecs_cluster.cluster.id
```

task_definition = aws_ecs_task_definition.task_definition.arn
desired_count   = 1

Configuração do load balancer

load_balancer {
target_group_arn = aws_lb_target_group.target_group.arn
container_name   = "meu-container"
container_port   = 80
}

Configuração da rede

network_configuration {
subnets   = [
"subnet-00d9bd50e8e3673d8",
"subnet-0f7307281e55df317"
] # IDs das minhas subnets
security_groups = [aws_security_group.lb_security_group.id]
assign_public_ip = false # "ENABLED" para habilitar IPs públicos
}

Outras configurações do serviço do ECS...

} #iria fazer a instalação do sql mas nao deu tempo

Criação do grupo de segurança para o load balancer

resource "aws_security_group" "lb_security_group" {
name_prefix   = "lb-sg"
vpc_id   = "vpc-093b125eb5f5fa24b" # ID da minha VPC

ingress {
from_port   = 80

```
to_port     = 80
protocol    = "tcp"
cidr_blocks = ["0.0.0.0/0"]  # Permitindo acessos públicos na porta 80
}

egress {
from_port   = 0
to_port     = 0
protocol    = "-1"  # Permitindo todo o tráfego de saída
cidr_blocks = ["0.0.0.0/0"]  # Permitindo a saída para qualquer IP
}
}
```

resultado: