**Etapa 1**

**Instalação do Docker**

Inicialmente, você precisa instalar o Docker no seu servidor. Conforme até mesmo a documentação oficial do Docker, esses são os passos:

# Atualize a lista de pacotes

Sudo apt-get update

# Instale os pacotes necessários para usar o repositório do Docker

Sudo apt-get install apt-transport-https ca-certificates curl software-properties-common

# Adicione a chave GPG oficial do Docker

Curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

# Adicione o repositório oficial do Docker ao APT

Sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"

# Atualize a lista de pacotes novamente

Sudo apt-get update

# Instale o Docker

Sudo apt-get install docker-ce

Verifique se o Docker foi instalado corretamente:

Sudo docker --version

**2. Obtendo a Imagem Docker**

Depois de instalar o Docker, você pode obter a imagem Docker que deseja executar. Isso geralmente é feito usando o comando docker pull. Por exemplo, para baixar a imagem do Nginx:

Sudo docker pull nginx

Caso queira alguma versão em específico pode usar a tag ao final do comando:

Sudo docker pull nginx: alpine3.20

**3. Executando o Contêiner**

Com a imagem já baixada, você pode executar um contêiner a partir dela usando o comando docker run junto com o -it que seria o modo iterativo:

Sudo docker run -it nginx

Se você quiser executar um comando específico dentro do contêiner, pode adicionar o comando /bin/bash para entrar em modo bash dentro do container ao final:

Sudo docker run -it nginx /bin/bash

**4. Configurações Adicionais**

* **Portas:** Se o contêiner precisar expor portas, use -p para mapear as portas do contêiner para as portas do host. Por exemplo, para mapear a porta 80 do contêiner para a porta 8080 do host:

sudo docker run -p 8080:80 -it nginx

* **Execução em Background:** Para executar o contêiner em segundo plano (detached mode), use -d:

sudo docker run -d -p 8080:80 nginx

**5. Verificando Contêineres em Execução**

Para listar os contêineres em execução:

sudo docker ps

Para listar todos os contêineres (incluindo os parados):

sudo docker ps -a

**6. Parando e Removendo Contêineres**

Para parar um contêiner:

sudo docker stop <CONTAINER\_ID>

Para remover um contêiner:

sudo docker rm <CONTAINER\_ID>

**7. Detalhe:**

Para que não haja necessidade de todos esses comandos docker sejam em Sudo, podemos adicionar o nosso usuário ao grupo Docker(somente após a instalação) permitindo assim que utilizemos os comandos Docker sem a necessidade do sudo :

Sudo usermod -aG docker $USER

**Etapa 2**

Para a comunicação entre o ECS Fargate e o RDS é necessário a criação de uma VPC para cada produto e um Security Group para cada um dos produtos:

**1. Configuração da VPC**

Primeiro, ambos os serviços (ECS Fargate e RDS) devem estar na mesma VPC (Virtual Private Cloud) ou em VPCs que estão configuradas para se comunicar entre si (por exemplo, usando VPC Peering).

**2. Criação da Subnet**

Na configuração de subnet é preciso especificar se será uma subnet privada ou pública. Sendo que o que precisar de saída para a internet ficará pública e o que não precisar ficará privada para maior segurança.

**3. Criação do Security Group**

No caso do security group é onde se deve adicionar a comunicação entre os dois dando permissão de inbound e outbound onde no security group do RDS tem que ser liberado no caso por exemplo de estar utilizando o MySQL o inbound na porta 3306 vinda somente do Fargate.

Já no caso do Fargate é o contrário onde deve-se permitir o outbound pela porta 3306 para que envie dados.

No caso para que seja efetivo ao criar o RDS deverá selecionar a VPC e o security group criados anteriormente e no caso do ECS Fargate também.

Depois de configurar tudo, faça testes para garantir que a aplicação no ECS Fargate consegue se conectar ao banco de dados RDS. Você pode verificar logs de erros ou adicionar scripts de teste dentro da aplicação para validar a conectividade.

**Etapa 3**

Segue a URL para o repositório do GitHub onde se encontra o teste: <https://github.com/Felipe0599/Teddy-test>

Nesse caso a pipeline não funcionou pois eu não cheguei a acessar a AWS para criar o ECR. No caso de criar o ECR deve-se apenas pegar o URI gerado dentro da AWS e adicionar como variável dentro do Secrets and variables do repositório Git com o nome de ECR\_URI e valor o URI do ECR.

Existe também a parte de acesso a AWS onde o workflow precisa ter as variáveis para acessar a AWS (também será adicionado no Secrets and variables do repositório Git) e chegar no ECR e são elas:

* AWS\_ACCESS\_KEY\_ID: O Access Key ID da sua conta AWS.
* AWS\_SECRET\_ACCESS\_KEY: O Secret Access Key da sua conta AWS.
* AWS\_REGION: A região onde o seu ECR está localizado (por exemplo, us-west-1).

PS: No caso desse repositório por se tratar de um teste, não foi criado e nem utilizado reusable-workflows de GitHub Actions

**Etapa 4**

Pro caso da utilização do Terraform iremos ter arquivos diferentes para cada etapa, isso é utilizado para questões como reutilização dos scripts, fácil colaboração dos colegas de equipe, modularidade, facilidade na atualização do script e isolamento dos problemas.

* Primeiro o provider.tf que configura o provedor onde no nosso caso é AWS:

# provider.tf

provider "aws" {

region = "us-west-1" # Altere para a região desejada

}

* Segundo temos o vpc.tf que define a VPC, subnets, e tabelas de roteamento:

# vpc.tf

resource "aws\_vpc" "main" {

  cidr\_block = "10.0.0.0/16" #Altere conforme seu range de IP

  enable\_dns\_support = true

  enable\_dns\_hostnames = true

  tags = {

    Name = "my-vpc"

  }

}

resource "aws\_subnet" "public" {

  vpc\_id     = aws\_vpc.main.id

  cidr\_block = "10.0.1.0/24" #Altere conforme seu range de IP

  availability\_zone = "us-west-1a"  # Altere conforme sua região

  map\_public\_ip\_on\_launch = true

  tags = {

    Name = "my-public-subnet"

  }

}

resource "aws\_subnet" "private" {

  vpc\_id     = aws\_vpc.main.id

  cidr\_block = "10.0.2.0/24" #Altere conforme seu range de IP

  availability\_zone = "us-west-1a"  # Altere conforme sua região

  tags = {

    Name = "my-private-subnet"

  }

}

resource "aws\_internet\_gateway" "igw" {

  vpc\_id = aws\_vpc.main.id

  tags = {

    Name = "my-igw"

  }

}

resource "aws\_route\_table" "public" {

  vpc\_id = aws\_vpc.main.id

  route {

    cidr\_block = "0.0.0.0/0"

    gateway\_id = aws\_internet\_gateway.igw.id

  }

  tags = {

    Name = "my-public-route-table"

  }

}

resource "aws\_route\_table\_association" "public" {

  subnet\_id      = aws\_subnet.public.id

  route\_table\_id = aws\_route\_table.public.id

}

* Terceiro temos o security\_groups.tf para definir os grupos de segurança:

# security\_groups.tf

resource "aws\_security\_group" "public\_sg" {

vpc\_id = aws\_vpc.main.id

egress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

ingress {

from\_port = 22

to\_port = 22

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"] # Permitir acesso SSH de qualquer lugar

}

ingress {

from\_port = 80

to\_port = 80

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"] # Permitir acesso HTTP de qualquer lugar

}

tags = {

Name = "my-public-sg"

}

}

resource "aws\_security\_group" "private\_sg" {

vpc\_id = aws\_vpc.main.id

egress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

ingress {

from\_port = 22

to\_port = 22

protocol = "tcp"

cidr\_blocks = ["10.0.0.0/16"] # Permitir comunicação interna

}

tags = {

Name = "my-private-sg"

}

}

* E quarto o instances.tf que define as instâncias EC2 com as subnets públicas e privadas:

# instances.tf

resource "aws\_instance" "public\_instance" {

ami = "" # Altere para uma AMI válida para sua região

instance\_type = "t2.micro" #Altere para o tipe de Instância Desejado

subnet\_id = aws\_subnet.public.id #Nome da subnet criada anteriormente

security\_group = aws\_security\_group.public\_sg.id #Nome do Security Group criado anteriormente

associate\_public\_ip\_address = true

tags = {

Name = "my-public-instance"

}

}

resource "aws\_instance" "private\_instance" {

ami = "" # Altere para uma AMI válida para sua região

instance\_type = "t2.micro" #Altere para o tipe de Instância Desejado

subnet\_id = aws\_subnet.private.id #Nome da subnet criada anteriormente

security\_group = aws\_security\_group.private\_sg.id #Nome do Security Group criado anteriormente

tags = {

Name = "my-private-instance"

}

}

E para utilizar o terraform basta utilizar os seguintes comandos:

Terraform init – Para iniciar o terraform

Terraform plan – Que seria para verificar todas as modificações que serão feitas

Terraform apply – Para aplicar as configurações

1. A comunicação entre os recursos na infraestrutura AWS é garantida pela VPC, que cria uma rede isolada onde todos os recursos, como subnets e instâncias EC2, podem se comunicar usando IPs privados. As subnets (pública e privada) dentro da mesma VPC podem se comunicar diretamente. Grupos de segurança controlam o tráfego de rede: a instância privada permite conexões SSH apenas internamente através da VPC, enquanto a instância publica permite SSH e HTTP de qualquer lugar, garantindo acesso controlado e seguro entre os recursos e a internet.
2. As portas de comunicação liberadas foram:

**Porta 22 (SSH)**:

**Instância Pública**: Liberada para todo o tráfego de entrada, permitindo conexões SSH de qualquer lugar da internet.

**Instância Privada**: Liberada apenas para tráfego vindo da subnet pública, permitindo que somente a instância pública (ou outras na mesma subnet) se conecte via SSH.

**Porta 80 (HTTP)**:

**Instância Pública**: Liberada para todo o tráfego de entrada, permitindo que qualquer usuário na internet acesse serviços HTTP hospedados na instância (Caso tenha).

1. Para realizar o teste de comunicação entre os recursos basta acessar a instância pública via SSH e de dentro da Instância publica validar a comunicação com a instancia privada se conectar a ela também via SSH e para validar a comunicação via porta 80 basta utilizar o comando curl <http://com> ip da instância pública.