Projeto "Cidade na Nuvem" - Infraestrutura na AWS com Terraform

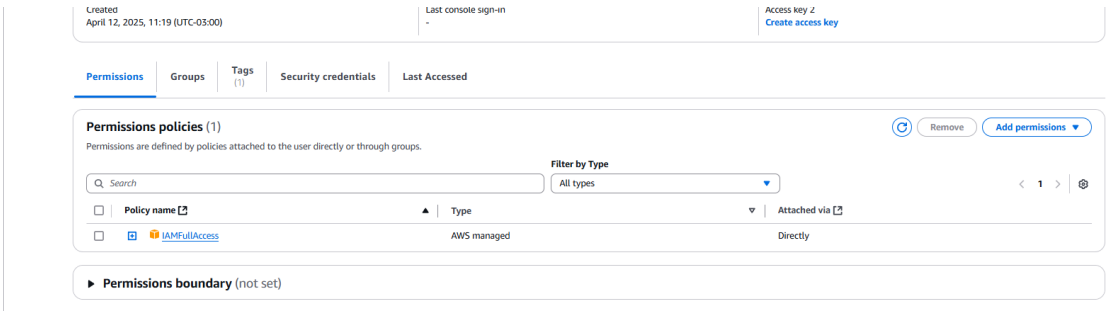
Este projeto é uma analogia de uma cidade construída na AWS, utilizando o Terraform para automatizar a criação de recursos essenciais para nossa cidade. Cada parte da cidade será representada por um recurso na AWS, o que ajuda a entender como diferentes serviços funcionam em conjunto.

Prefeitura (IAM) – Controle e Permissões

A prefeitura da cidade tem a responsabilidade de definir quem pode fazer o quê dentro da cidade. Em termos da AWS, isso é feito usando o **IAM (Identity and Access Management)**. O IAM nos permite controlar quem pode acessar os recursos da AWS e o que cada pessoa pode fazer.

PASSO 1

Criação do IAM (usuário com permissões) com políticas de acesso a modo de estudo, são:



Após criar o IAM e conseguir as chaves de acesso configurei ambiente para usar Terraform com AWS. Feito pelo o VS CODE usando os comandos

AWS Access Key ID [None]:

COLE SUA PRIMEIRA CHAVE

AWS Secret Access Key [None]:

COLE SUA SEGUNDA CHAVE

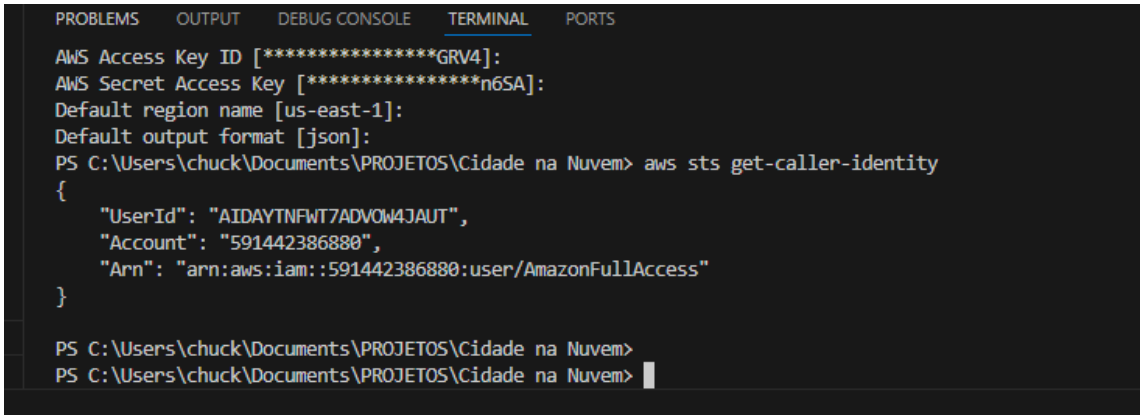
Default region name [None]:

COLOQUE A REGIÃO A QUAL VOCÊ DESEJA

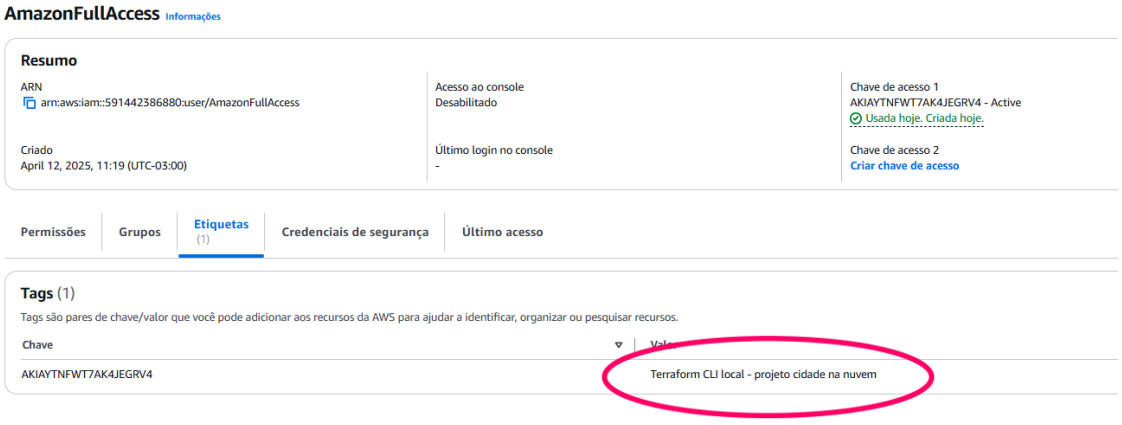
Default output format [None]:

DIGITE O FORMATO, NO MEU CASO FOI JSON

Print abaixo após a criação.



Vendo pelo site do AWS já conseguimos ver a tag criada e assim finalizando o passo 1



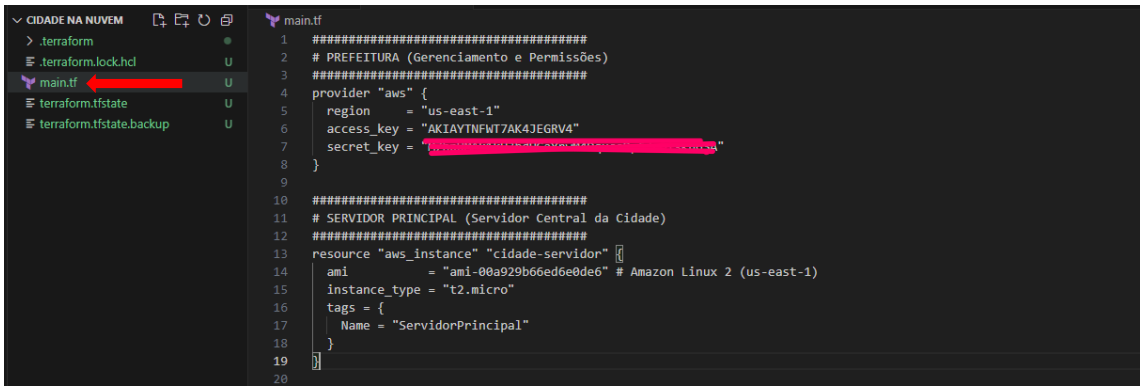
PASSO 2

Casas (EC2) – Instâncias de Máquinas Virtuais

As casas da cidade são representadas pelas **instâncias EC2**. As EC2 são máquinas virtuais onde podemos rodar nossos aplicativos e serviços. Cada casa tem suas próprias especificações que logo explicaremosno proximo passo.

Criei um arquivo dentro da pasta CIDADE NA NUVEM com o nome main.tf, esse arquivo é o coração do projeto por ele vamos começar a criar a cidade.

Conforme a print abaixo coloquei as KEYS que gerei no AWS



Após rodar os comandos do terraform:

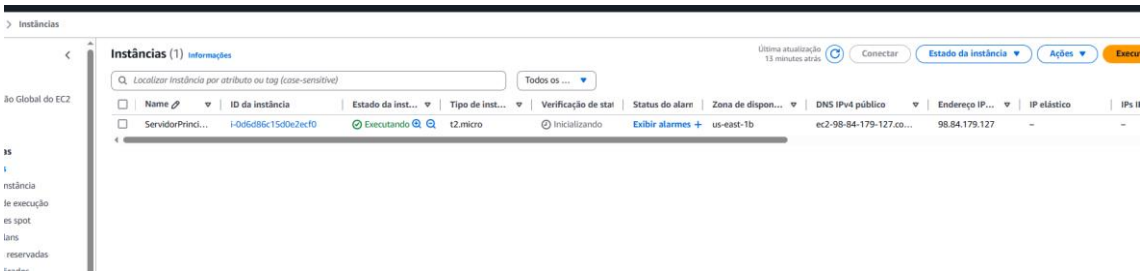
Terraform init

Terraform plan

Terraform apply (yes) A Instancia foi criada no AWS

Caso der erro na ultima etapa pode usar o comando >terraform apply -auto-approve< para aprovar diretamente

PRONTO O SERVIDOR PRINCIPAL(CASA 0) ESTÁ CRIADO finalizando o PASSO 2



PASSO 3 CRIAR MAIS INSTANCIA (ILUSTRAR E ENTENDER A DOCUMENTAÇÃO e o AMI)

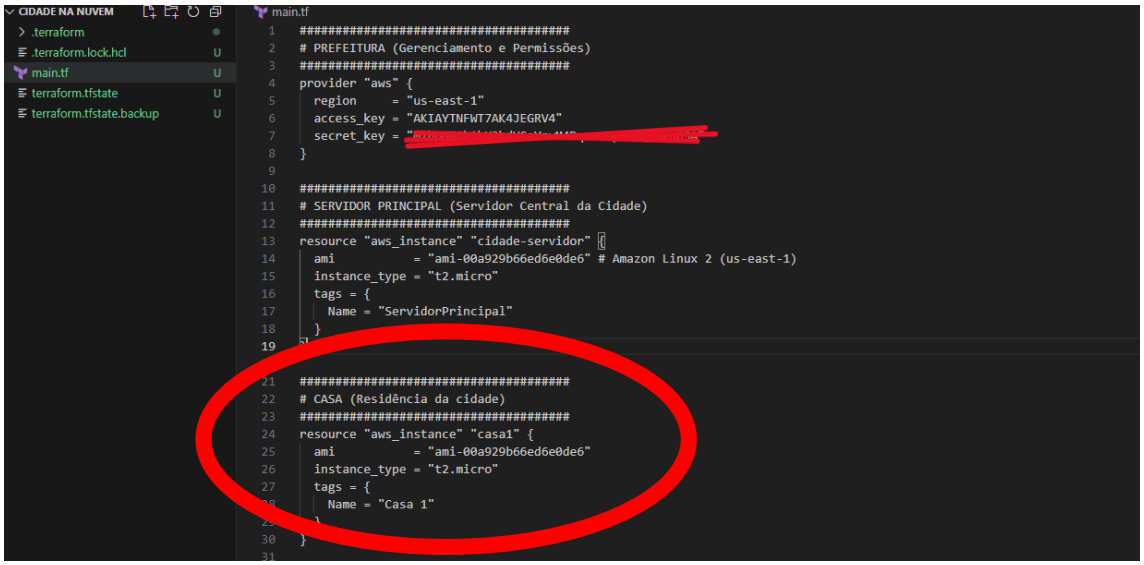
Agora vamos fazer uma casa a modo de praticar o que já fizemos replicando o exemplo do passo 2, lembrando que podemos criar varias casas(instancias) só com linha de código facilitando a velocidade da criação dos processos, usei a mesma AMI.

Explicando sobre AMI

Uma **AMI** é uma imagem de máquina que contém todas as informações necessárias para **iniciar uma instância EC2** (servidor virtual). Ela funciona como um "modelo" que define:

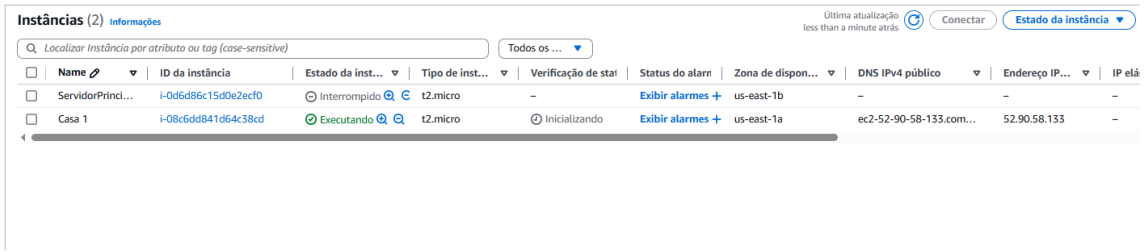
- O **sistema operacional** (ex: Ubuntu, Amazon Linux, Windows)
- Os **softwares e aplicações pré-instalados**
- As **configurações de sistema** e permissões

- As configurações de **armazenamento em disco** (volumes EBS, por exemplo)



Após adicionar mais algumas linhas de comando dentro do documento do main.tf, basta rodar o terraform novamente para subir a instância com a mesma AMI ou diferente conforme a necessidade.(rodar os comandos terraform novamente)

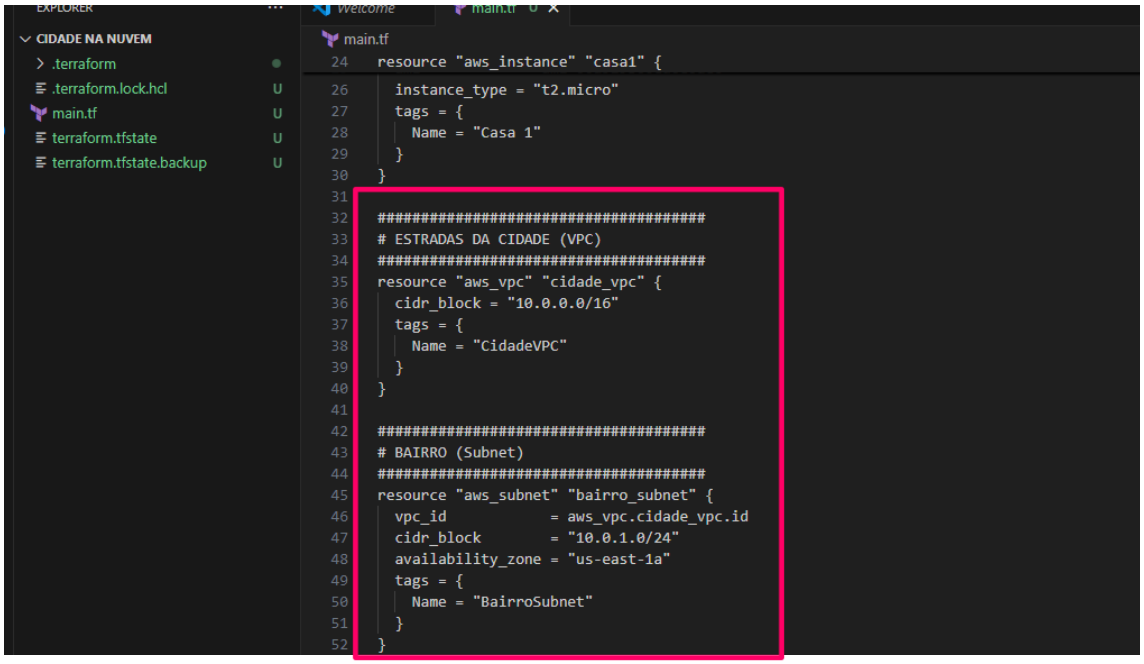
Após a criação da 1 casa em poucos segundos já é criada no painel da AWS. (podemos criar 50 maquinas em poucos minutos através da linha de código se quiser)



Passo 4

Ruas da cidade (VPC e Subnets) – Comunicação entre os Recursos

As ruas da cidade representam a **VPC (Virtual Private Cloud)** e as **Subnets**, que são responsáveis pela comunicação entre os diferentes recursos da cidade. Elas permitem que a cidade tenha uma infraestrutura bem conectada, com separação de áreas (sub-redes) para diferentes tipos de serviços.



Analogia:

Cidade → VPC (Virtual Private Cloud)

Bairro → Subnet

Ruas → Conexões internas entre recursos (dentro da VPC)

Agora vamos por partes:

VPC (Virtual Private Cloud) = a cidade inteira

Imagine que você comprou um terreno gigante (bloco de IPs como 10.0.0.0/16).

É aqui que tudo vai ser construído: casas (EC2), a prefeitura (IAM), biblioteca (S3), etc.

A VPC garante que tudo isso está isolado, seguro e sob seu controle.

Subnet = o bairro da cidade

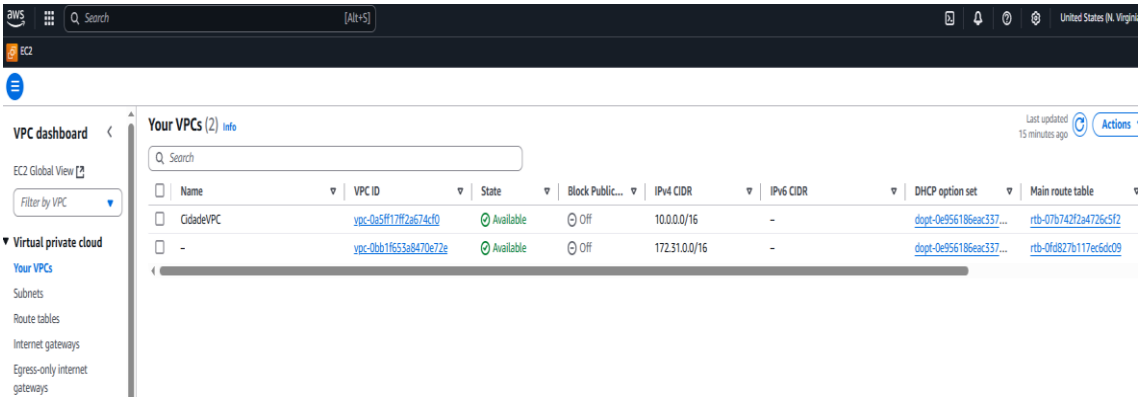
Dentro da sua cidade, você define bairros com endereços IP menores (ex: 10.0.1.0/24).

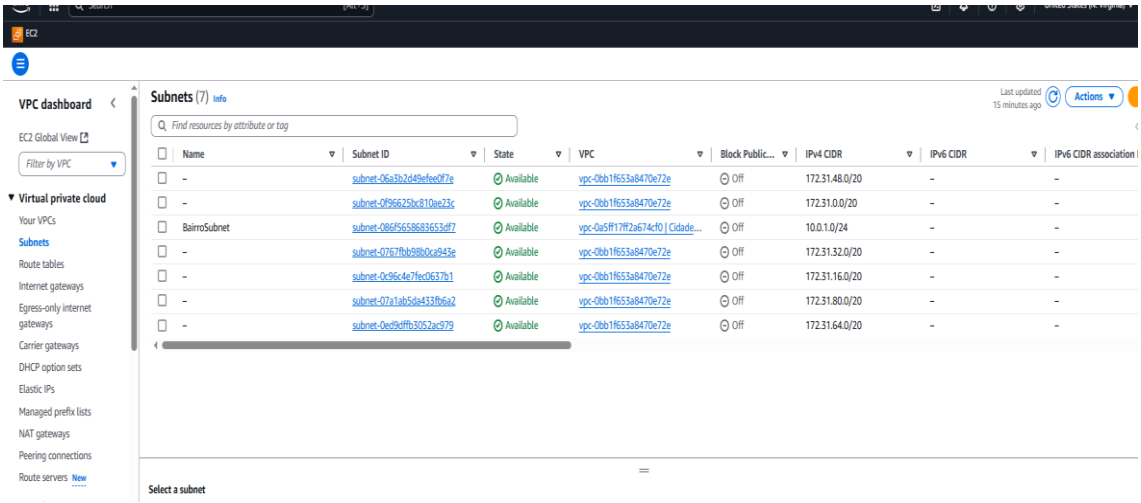
Esses bairros (subnets) organizam os recursos. Por exemplo: ▸ Um bairro só para servidores públicos (com acesso à internet). ▸ Outro só para bancos de dados (sem acesso externo).

Você pode criar várias subnets e decidir se são públicas ou privadas, de acordo com o que será construído nelas.

- Criei uma VPC com um bloco IP: 10.0.0.0/16 capacidade para mais de 65 mil IPs!

- Dentro dela, criei uma subnet: 10.0.1.0/24 capacidade para 256 IPs, em uma zona específica da AWS

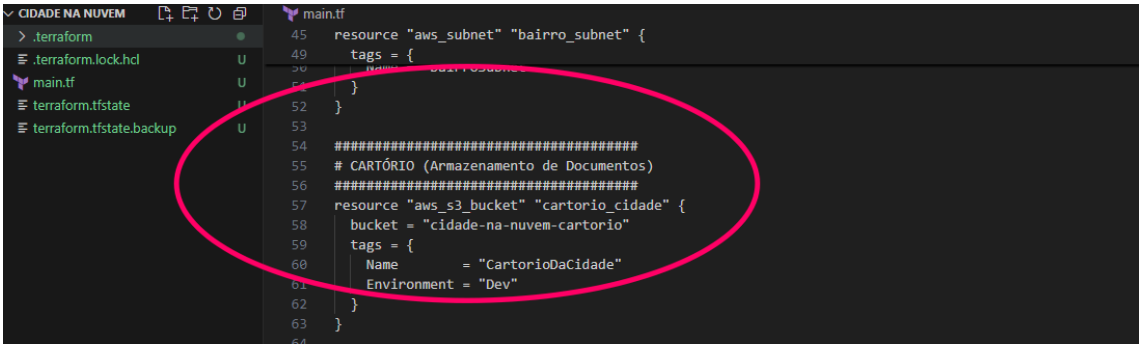




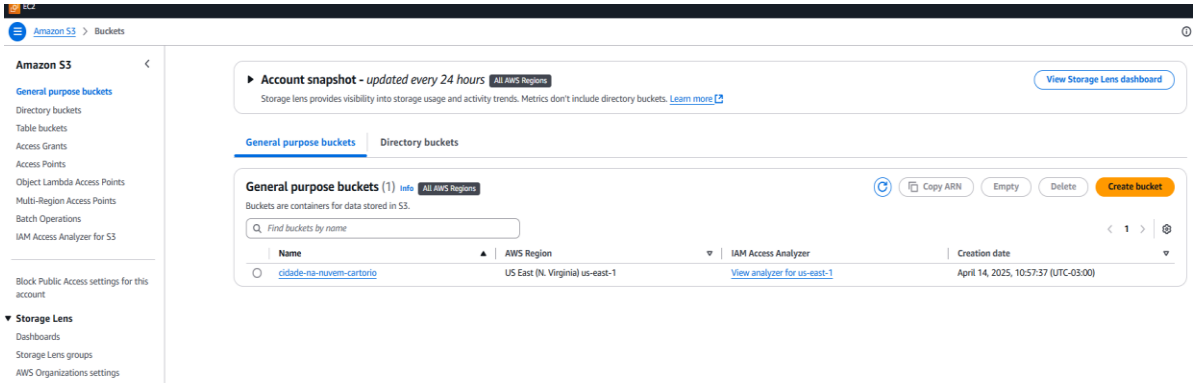
Passo 5 –

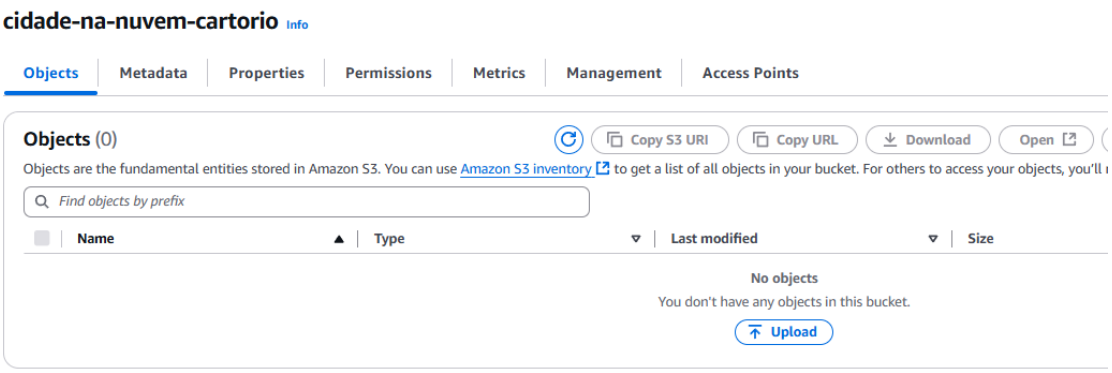
Cartório (S3) – Armazenamento de Documentos

O cartório da cidade é representado pelo **S3 (Simple Storage Service)**, que armazena documentos e arquivos públicos ou privados da cidade. Ele garante que tudo o que for importante, como registros e contratos, esteja seguro e acessível.



Após rodar os comandos do terraform, criei bucket s3 no AWS





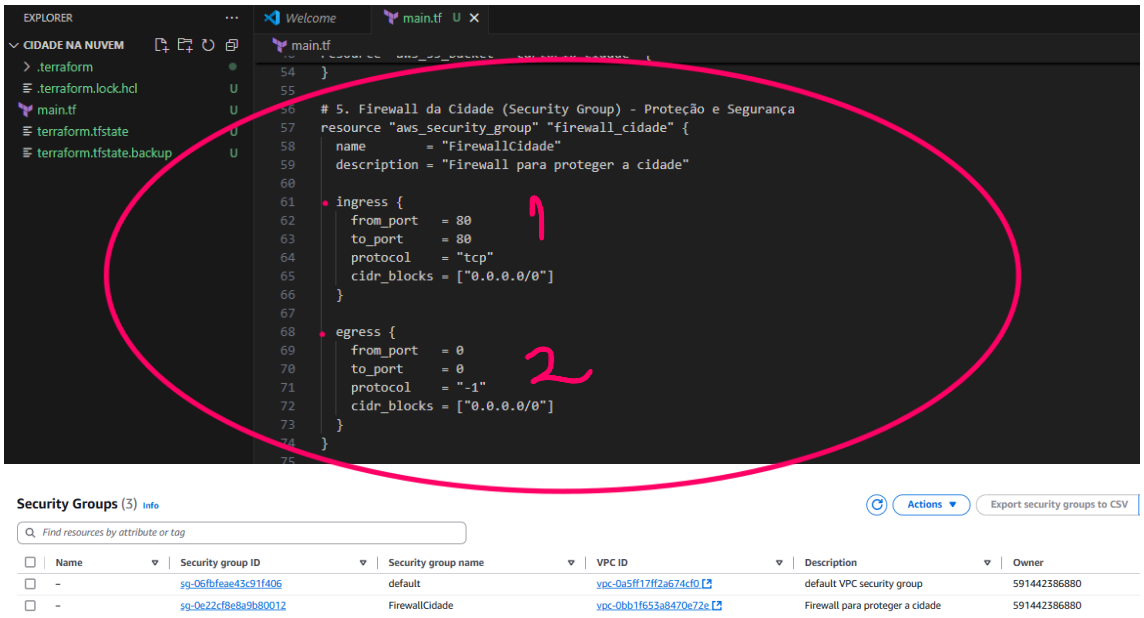
Exemplo prático de uso: O **bucket S3** chamado **cartoriodacidade** armazena os "documentos" importantes da cidade, como registros de nascimento, contratos, e outros arquivos necessários para o funcionamento da cidade.

Passo 6

PORTARIA DA CIDADE (FIREWALL)

Nossa cidade na nuvem está aberta para o mundo. Assim como uma cidade de verdade precisa de muros e portões para controlar quem entra e quem sai, nossa cidade digital também precisa de proteção.

Essa proteção é feita por algo chamado Security Group, que funciona como um “muro e portaria virtual com regras”.



Criei um firewall chamado FirewallCidade

Defini algumas regras de entrada (quem pode entrar): 1

Aqui estamos dizendo: "Qualquer recurso dentro da cidade pode sair para qualquer lugar, por qualquer porta". 2

Em resumo:

> Esse passo cria o muro da cidade com uma entrada liberada para visitas na porta 80 (como se fosse um site aberto ao público).
>
> E permite que quem estiver dentro da cidade possa sair livremente.

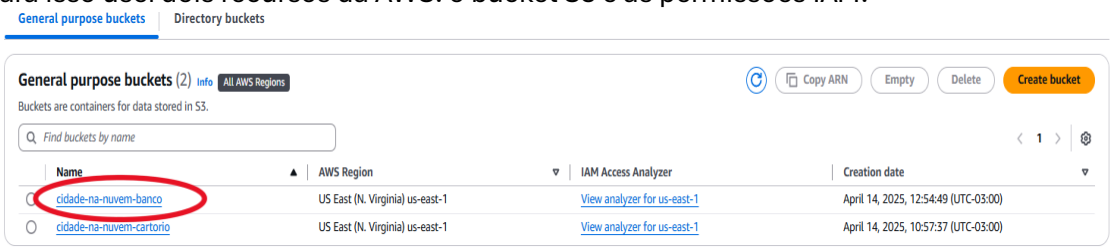
Passo 7

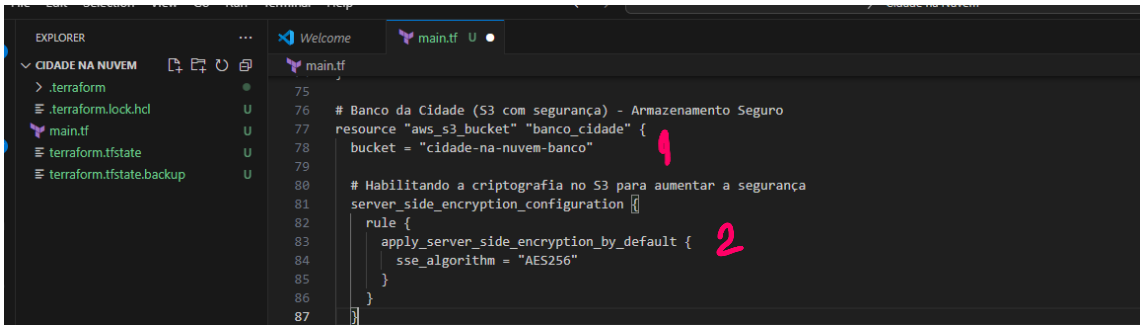
O Banco da Cidade (bucket S3 Seguro + Permissões)

Agora a cidade tem um banco onde as pessoas guardam dinheiro e documentos importantes. Esse banco precisa:

> Ser seguro
>
> Ter regras claras sobre quem pode acessar e o quê.

Para isso usei dois recursos da AWS: o bucket S3 e as permissões IAM.





Isso é como construir o prédio do banco com o nome “cidade-na-nuvem-banco”. 1

Depois, ativamos a segurança criptografia 2

Aqui estamos dizendo: “Tudo que for guardado neste banco será criptografado automaticamente”. Isso é como ter cofres automáticos dentro do banco.

Usei o padrão de criptografia AES256, que é uma forma segura de proteger os arquivos.

Criando as políticas do banco

Agora que já temos o banco, precisamos definir quem pode acessar o banco através de um cartão . Para isso, criei uma política de permissões que define quem pode usar o banco (usuários ou serviços) e o que eles podem fazer dentro. Nesse caso, queremos permitir que alguém consulte o saldo, faça depósitos e realize saques.

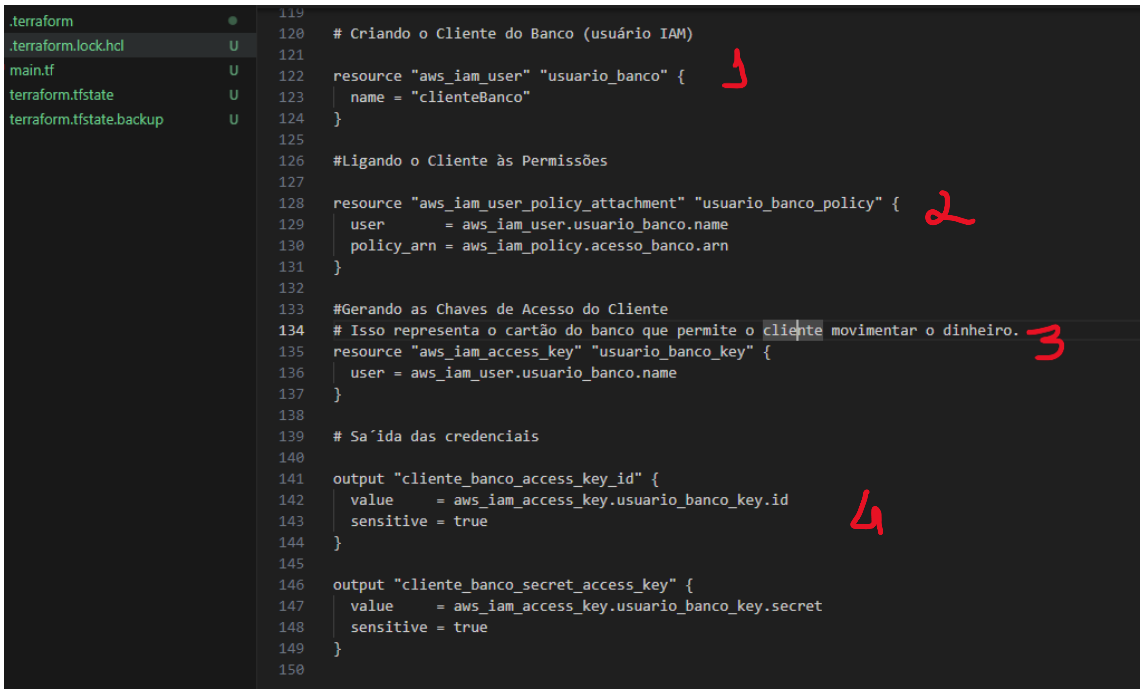
```
# Política de Acesso ao Banco

resource "aws_iam_policy" "acesso_banco" {
  name        = "AcessoBancoPolicy"
  description = "Permissões para o cliente acessar o banco da cidade"

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Action = [
          "s3:GetObject",      # Ver saldo
          "s3:PutObject",      # Depositar
          "s3:DeleteObject"    # Sacar
        ]
        Resource = "arn:aws:s3:::cidade-na-nuvem-banco/*"
      }
    ]
  })
}
```

Onde está o circulo é uma política que diz o que pode ser feito dentro do banco.

Consultar saldo: s3:GetObject é como olhar o saldo na conta, ou seja, ler os arquivos no banco.

Depósito: s3:PutObject é como depositar dinheiro, ou seja, enviar arquivos para o banco.

Saque: s3:DeleteObject é como retirar dinheiro, ou seja, deletar arquivos do banco.

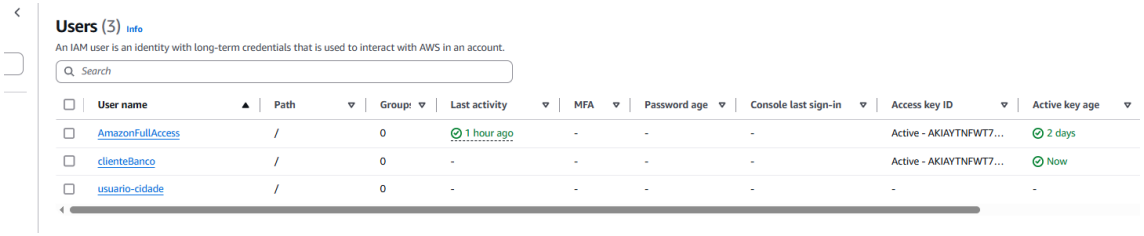


1 Criando o Cliente do Banco

2 Ligando o Cliente à Política, aqui conectamos o cidadão à política do banco. Agora ele tem permissão de movimentar o dinheiro.

3 Gerando as Chaves de Acesso, aqui estamos emitindo o “cartão do banco” — ou seja, as credenciais de acesso que o cliente poderá usar via AWS CLI.

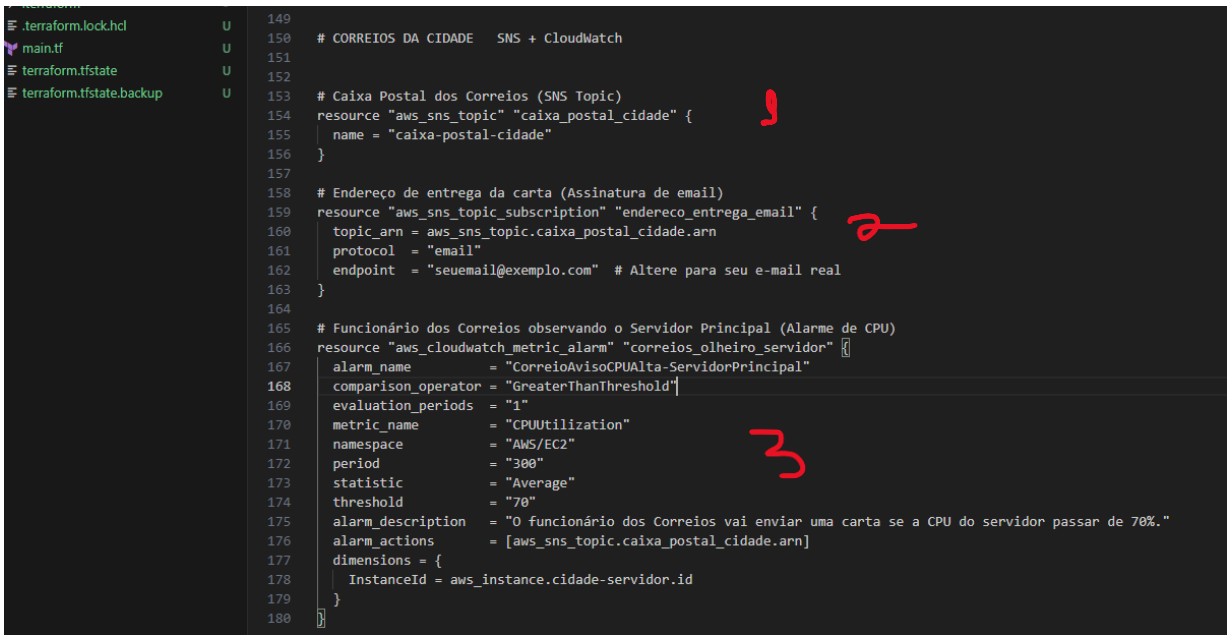
4 Exibindo as Credenciais Esses outputs mostram as credenciais do cliente. Pode ver as senhas dos clientes informações sensíveis.



Passo 8

Correios da cidade (SNS + CloudWatch)

Imagine que toda cidade precisa de um sistema de comunicação oficial . Se algo dá errado, como uma casa pegando fogo ou o servidor ficando sobrecarregado, usei a analogia o Correios que precisam enviar um aviso urgente para as autoridades.



1. Caixa Postal dos Correios (SNS Topic)

- Essa é a central dos Correios.
- Quando o olheiro (CloudWatch) precisa avisar algo, ele envia para essa caixa postal.

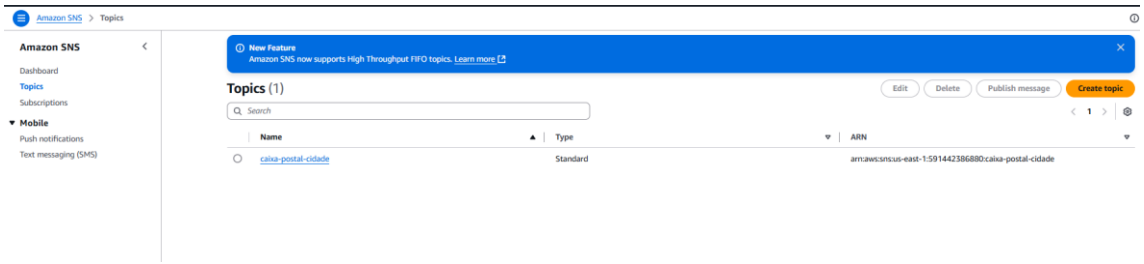
2. Endereço de entrega da carta (Assinatura por E-mail):

- Aqui configuramos quem vai receber a carta dos Correios.
- Pode ser um e-mail da prefeitura, por exemplo.
- Você receberá um e-mail para confirmar a assinatura.

3. Funcionário dos Correios observando a cidade (CloudWatch Alarm)

- Esse é o funcionário dos Correios que está monitorando o “Servidor Principal” da cidade.
- Se a CPU do servidor ficar acima de 70% por mais de 5 minutos, ele envia uma carta para a caixa postal.
- A carta será redirecionada para o e-mail configurado anteriormente.

Essa pratica de adicionar o CloudWatch é essencial para manter a cidade (infraestrutura) sob controle e bem monitorada.



Passo 9

1 Criando a Fábrica de Contêineres (Instância EC2 com Docker)

Criei uma instância EC2 (máquina virtual) que vai ser nossa "fábrica de contêineres". A instância vai rodar o **Amazon Linux 2**, uma distribuição Linux otimizada para a AWS.

O tipo de instância é **t2.micro**, que é adequado para cargas leves, sendo ideal para testes ou pequenas aplicações. Conforme o print abaixo

A instância é conectada a uma **sub-rede** (subnet_id), que foi criada previamente em outro recurso (no caso, aws_subnet.bairro_subnet.id), mantendo tudo organizado

```
181
182  # FÁBRICA DE CONTÊINERES (EC2 + Docker)
183
184  # Criando a Fábrica de Contêineres (Instância EC2)
185  resource "aws_instance" "fabrica_conteineres" {
186    ami           = "ami-00a929b66ed6e0de6"  # Amazon Linux 2
187    instance_type = "t2.micro"
188    subnet_id     = aws_subnet.bairro_subnet.id  # Conectar à rede (bairro)
189
190    # Script para instalar Docker e rodar o contêiner
191    user_data = <<-EOF
192                #!/bin/bash
193                yum update -y
194                amazon-linux-extras install docker -y
195                service docker start
196                usermod -a -G docker ec2-user
197                docker run -d -p 80:80 my_custom_app:latest  # Rodando o contêiner
198                EOF
199
200    tags = {
201      Name = "FabricaDeConteineres"
202    }
203  }
```

2 **Instalação e Execução do Docker:**

- Usei um recurso chamado user_data. Ele serve para passar um script que será executado automaticamente assim que a instância EC2 for criada. É como entregar uma lista de tarefas que a máquina vai seguir sozinha ao ligar pela primeira vez.

O que esse script faz:

- yum update -y: Ele atualiza o sistema da instância para garantir que tudo esteja com as versões mais recentes.

- amazon-linux-extras install docker -y: Instala o Docker, que é a plataforma que vamos usar para rodar nossa aplicação dentro de um contêiner.
- service docker start: Liga o serviço do Docker, como se fosse apertar o botão "Ligar" da ferramenta.
- usermod -a -G docker ec2-user: Dá permissão ao usuário principal da instância (ec2-user) para usar o Docker sem precisar de comandos de administrador.
- docker run -d -p 80:80 my_custom_app:latest: Esse comando inicia um contêiner com a nossa aplicação dentro. Ele também mapeia a porta 80 da instância (porta padrão para acesso web) para a porta 80 do contêiner. Isso significa que quando acessarmos o IP da instância, vamos ver a aplicação rodando.

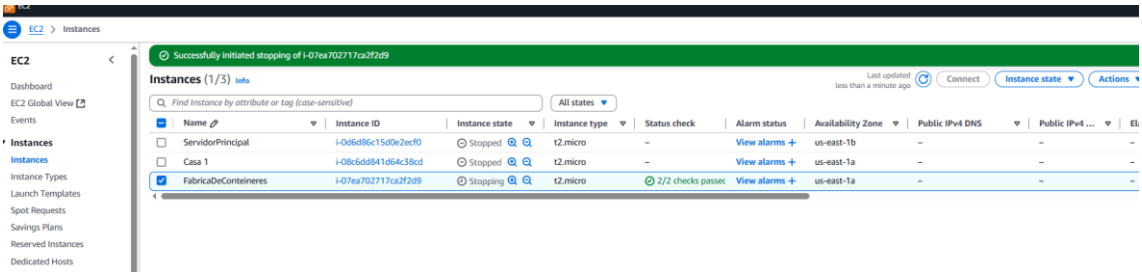
TAG

A instância recebe uma **tag** com o nome "FabricaDeConteineres" para facilitar a identificação na AWS.

```
205  # Exibindo o IP da Fábrica (Instância EC2)
206  output "fabrica_conteineres_ip" {
207    value = aws_instance.fabrica_conteineres.public_ip
208  }
209
```

Exibindo o IP Público da Instância:

- Depois que a instância EC2 (máquina virtual) for criada, o código mostra automaticamente o endereço IP público dela.
- Esse IP funciona como um “endereço de rua” na internet, você consegue acessar o que está sendo executado dentro da instância, como a aplicação rodando no Docker.
- É como se fosse o número da casa da “fábrica de contêineres”. Com ele, você pode abrir o navegador e ver a aplicação funcionando direto de lá, ou até conectar com outras ferramentas.



Passo 10 (destruir tudo para não gerar cobrança)

Usei o comando terraform destroy, para destruir todo o projeto e não gerar cobranças

```
Destroy complete! Resources: 16 destroyed.
```


Docker
Docker
AM
EC2
EC
EC2
CloudWatch
S3
VPC
Subnets

RESUMO DO PROJETO CIDADE NA NUVEM

Este projeto foi desenvolvido para simular a criação de uma cidade inteira utilizando a infraestrutura da AWS, com o Terraform sendo a ferramenta principal para automatizar a criação de todos os recursos. O objetivo é mostrar como diversos serviços da AWS podem ser usados para representar os componentes de uma cidade real, mas de forma virtual e escalável na nuvem.

Componentes do Projeto:

1. **Prefeitura (IAM)**: Criamos usuários e políticas de permissão para controlar o acesso e autorizações dos recursos dentro da cidade. A Prefeitura garante que somente quem tem permissão possa acessar ou modificar certos recursos.
2. **Casas (EC2)**: Usamos instâncias EC2 para simular as casas da cidade. Cada instância pode ser vista como uma residência. Criamos uma instância especial para rodar contêineres Docker, simulando um servidor ou uma empresa.
3. **Ruas e Bairros (VPC/Subnet)**: Criamos uma VPC (Virtual Private Cloud) e uma subnet para garantir que todos os recursos na cidade possam se comunicar entre si de forma segura e organizada.
4. **Cartório e Banco da Cidade (S3)**: Utilizamos buckets S3 para simular um cartório, onde documentos são armazenados, e um banco, onde são guardados os dados financeiros da cidade, com políticas que controlam quem pode acessar cada recurso.
5. **Segurança (Security Group)**: Criamos um grupo de segurança para proteger a cidade, controlando quem pode acessar as casas e servidores através da internet. Isso ajuda a evitar acessos não autorizados.
6. **Correios (CloudWatch + SNS)**: Implementamos uma solução de monitoramento, onde se o servidor principal ultrapassar 70% de uso de CPU, um alerta será enviado por e-mail. Isso é como um funcionário dos Correios que avisa quando algo não está funcionando corretamente.
7. **Fábrica de Contêineres (Docker na EC2)**: Criamos uma instância EC2 que, ao ser configurada, instala o Docker e executa uma aplicação dentro de um contêiner. Isso simula um servidor rodando uma aplicação real.
8. Destruimos tudo para evitar cobranças