

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

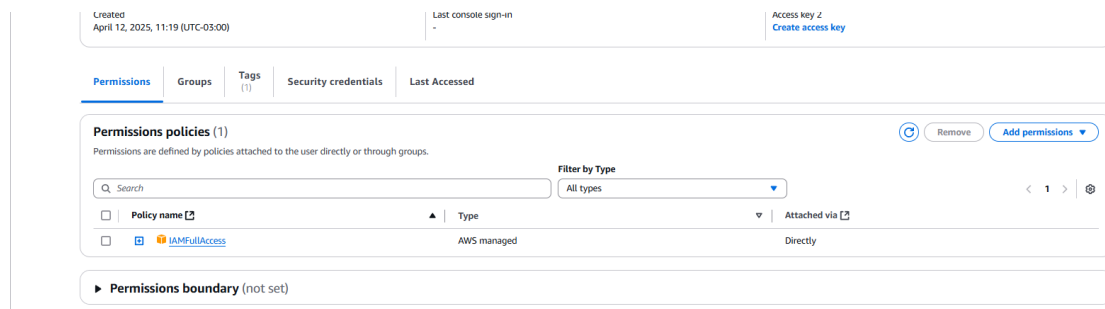
Este projeto é uma analogia de uma cidade construída na AWS, utilizando o Terraform para automatizar a criação de recursos essenciais para nossa cidade. Cada parte da cidade será representada por um recurso na AWS, o que ajuda a entender como diferentes serviços funcionam em conjunto.

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.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
AWS Access Key ID [*****GRV4]:
AWS Secret Access Key [*****n6SA]:
Default region name [us-east-1]:
Default output format [json]:
PS C:\Users\chuck\Documents\PROJETOS\Cidade na Nuvem> aws sts get-caller-identity
{
  "UserId": "AIDAYTNFWT7ADVOW4JAUT",
  "Account": "591442386880",
  "Arn": "arn:aws:iam::591442386880:user/AmazonFullAccess"
}

PS C:\Users\chuck\Documents\PROJETOS\Cidade na Nuvem>
PS C:\Users\chuck\Documents\PROJETOS\Cidade na Nuvem>
```

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

AmazonFullAccess informações

Resumo		
ARN arn:aws:iam::591442386880:user/AmazonFullAccess	Acesso ao console Desabilitado	Chave de acesso 1 AKIAIYTNFWT7AKAJEGRV4 - Active Usada hoje. Criada hoje.
Criado April 12, 2025, 11:19 (UTC-03:00)	Último login no console -	Chave de acesso 2 Criar chave de acesso

Permissões Grupos **Etiquetas (1)** Credenciais de segurança Último acesso

Tags (1)
Tags são pares de chave/valor que você pode adicionar aos recursos da AWS para ajudar a identificar, organizar ou pesquisar recursos.

Chave	Valor
AKIAIYTNFWT7AKAJEGRV4	Terraform CLI local - projeto cidade na nuvem

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 explicaremos no próximo passo.

Crie 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

```
main.tf
1 #####
2 # PREFEITURA (Gerenciamento e Permissões)
3 #####
4 provider "aws" {
5   region     = "us-east-1"
6   access_key = "AKIAIYTNFWT7AK4JEGRV4"
7   secret_key = "wJalrXUdfjKOzYMBp36vW0xuuBjPgkUG8bXn9"
8 }
9
10 #####
11 # SERVIDOR PRINCIPAL (Servidor Central da Cidade)
12 #####
13 resource "aws_instance" "cidade-servidor" {
14   ami           = "ami-00a929b66ed6e0de6" # Amazon Linux 2 (us-east-1)
15   instance_type = "t2.micro"
16   tags = {
17     Name = "ServidorPrincipal"
18   }
19 }
```

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

> Instâncias

←

Instâncias (1) Informações

Última atualização
13 minutos atrás

Conectar Estado da instância Ações Excluir

Q Localizar instância por atributo ou tag (case-sensitive)

Todos os ... ▼

Ào Global do EC2

<input type="checkbox"/>	Name ↗	ID da instância	Estado da inst...	Tipo de inst...	Verificação de star	Status do alarm	Zona de dispon...	DNS IPv4 pública	Endereço IP...	IP elástico	IP
<input type="checkbox"/>	ServidorPrinci...	i-0d6d86c15d0e2ecf0	Executando 🔍	t2.micro	Inicializando	Exibir alarmes +	us-east-1b	ec2-98-84-179-127.co...	98.84.179.127	-	-

←

15

↓

instância

de execução

es spot

lans

reservas

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)

```
main.tf
1 #####
2 # PREFEITURA (Gerenciamento e Permissões)
3 #####
4 provider "aws" {
5     region     = "us-east-1"
6     access_key = "AKIAVTFNFWT7AK4JEGRV4"
7     secret_key = "wJalrXU3WhXdZd7Hs6N5YsPpUv"
8 }
9
10 #####
11 # SERVIDOR PRINCIPAL (Servidor Central da Cidade)
12 #####
13 resource "aws_instance" "cidade-servidor" {
14     ami           = "ami-00a929b66ed6e0de6" # Amazon Linux 2 (us-east-1)
15     instance_type = "t2.micro"
16     tags = {
17         Name = "ServidorPrincipal"
18     }
19 }
20
21 #####
22 # CASA (Residência da cidade)
23 #####
24 resource "aws_instance" "casa1" {
25     ami           = "ami-00a929b66ed6e0de6"
26     instance_type = "t2.micro"
27     tags = {
28         Name = "Casa 1"
29     }
30 }
```

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)

Instâncias (2)

Informações

Última atualização

less than a minute atrás

Conectar

Estado da instância ▾

Q

Localizar instância por atributo ou tag (case-sensitive)

Todos os ... ▾

<input type="checkbox"/>	Name 🔗 ▾	ID da instância	Estado da inst... ▾	Tipo de inst... ▾	Verificação de stat	Status do alarm	Zona de dispon... ▾	DNS IPv4 público ▾	Endereço IP... ▾	IP elá
<input type="checkbox"/>	ServidorPrind...	i-0d6d86c15d0e2ecf0	⊖ Interrompido 🔗 🔗	t2.micro	-	Exibir alarmes +	us-east-1b	-	-	-
<input type="checkbox"/>	Casa 1	i-08c6dd841d64c38cd	✅ Executando 🔗 🔗	t2.micro	⊖ Inicializando	Exibir alarmes +	us-east-1a	ec2-52-90-58-133.com...	52.90.58.133	-

⏮

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.

```
24 resource "aws_instance" "casa1" {
25
26     instance_type = "t2.micro"
27     tags = {
28         Name = "Casa 1"
29     }
30 }
31
32 #####
33 # ESTRADAS DA CIDADE (VPC)
34 #####
35 resource "aws_vpc" "cidade_vpc" {
36     cidr_block = "10.0.0.0/16"
37     tags = {
38         Name = "CidadeVPC"
39     }
40 }
41
42 #####
43 # BAIRRO (Subnet)
44 #####
45 resource "aws_subnet" "bairro_subnet" {
46     vpc_id       = aws_vpc.cidade_vpc.id
47     cidr_block   = "10.0.1.0/24"
48     availability_zone = "us-east-1a"
49     tags = {
50         Name = "BairroSubnet"
51     }
52 }
```

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

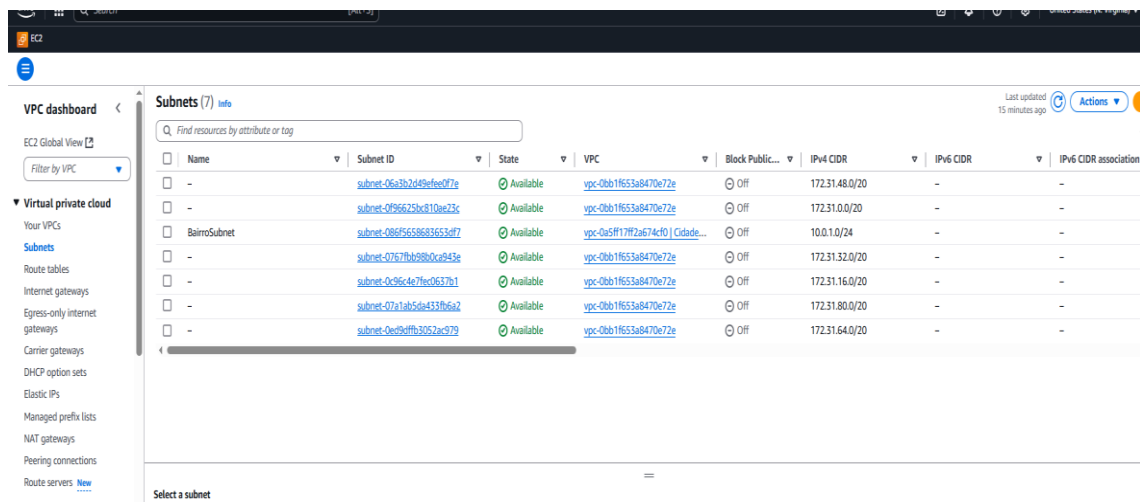
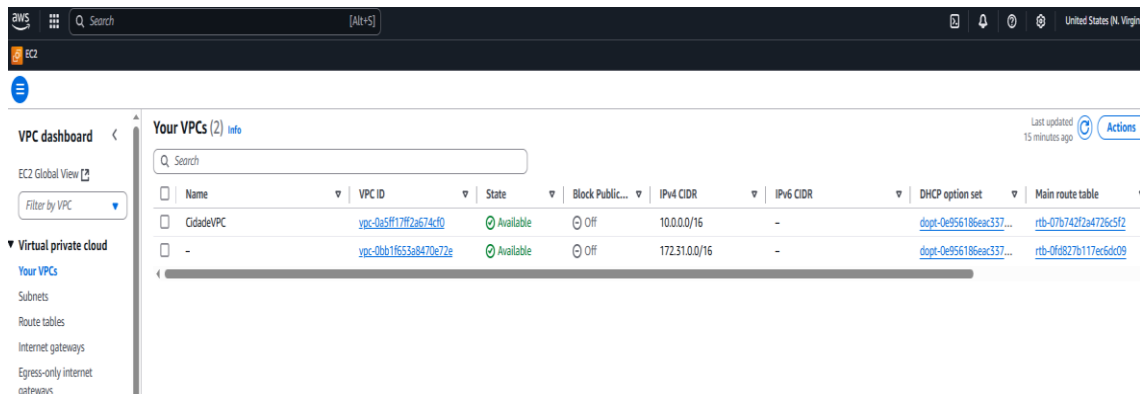
Dentro da sua cidade, você define bairros com endereços IP menores (ex: 10.0.1.0/24).

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!**

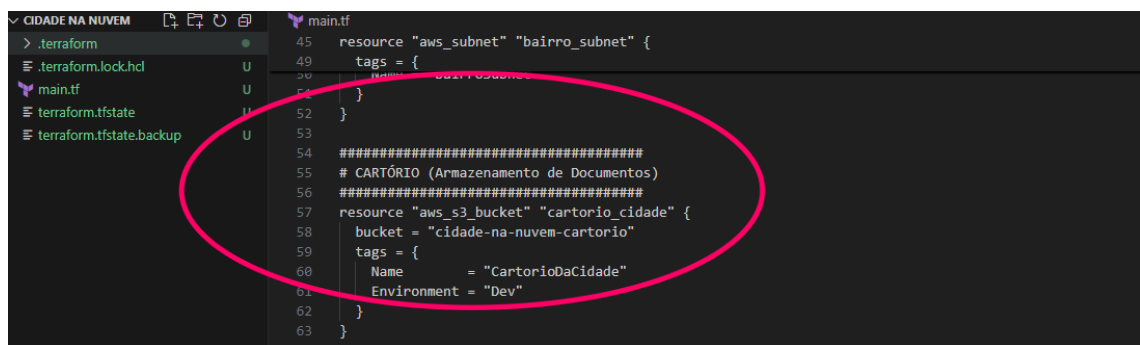
- Dentro dela, criei uma subnet: 10.0.1.0/24 **capacidade para 256 IPs**, em uma zona específica da AWS



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

The screenshot shows the AWS Management Console interface for an S3 bucket named 'cidade-na-nuvem-cartorio'. The left sidebar contains navigation links for Amazon S3, General purpose buckets, Directory buckets, Table buckets, Access Grants, Access Points, Object Lambda Access Points, Multi-Region Access Points, Batch Operations, IAM Access Analyzer for S3, Block Public Access settings, Storage Lens, Dashboards, Storage Lens groups, and AWS Organizations settings. The main content area displays an 'Account snapshot' and a list of 'General purpose buckets'. The bucket 'cidade-na-nuvem-cartorio' is listed with details: Name, AWS Region (US East (N. Virginia) us-east-1), IAM Access Analyzer, and Creation date (April 14, 2025, 10:57:37 (UTC-05:00)). Below the bucket list, the 'Objects' tab is selected, showing 'Objects (0)'. A message states: 'No objects. You don't have any objects in this bucket.' with an 'Upload' button.

Exemplo prático de uso: O **bucket S3** chamado **cartoriocidade** 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”.

```
54 }
55
56 # 5. Firewall da Cidade (Security Group) - Proteção e Segurança
57 resource "aws_security_group" "firewall_cidade" {
58   name           = "FirewallCidade"
59   description    = "Firewall para proteger a cidade"
60
61   ingress {
62     from_port     = 80
63     to_port       = 80
64     protocol      = "tcp"
65     cidr_blocks   = ["0.0.0.0/0"]
66   }
67
68   egress {
69     from_port     = 0
70     to_port       = 0
71     protocol      = "-1"
72     cidr_blocks   = ["0.0.0.0/0"]
73   }
74 }
75 }
```

Security Groups (3) [Info](#)

<input type="checkbox"/>	Name	Security group ID	Security group name	VPC ID	Description	Owner
<input type="checkbox"/>	-	sg-06fbf9ae43c91f406	default	vpc-9a5ff17ff2a674cd0	default VPC security group	591442386880
<input type="checkbox"/>	-	sg-0c22cf8e8a9b80012	FirewallCidade	vpc-0bb1f653a8470e72e	Firewall para proteger a cidade	591442386880

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.

General purpose buckets (2) [Info](#) All AWS Regions

Buckets are containers for data stored in S3.

	Name	AWS Region	IAM Access Analyzer	Creation date
<input checked="" type="radio"/>	<u>cidade-na-nuvem-banco</u>	US East (N. Virginia) us-east-1	View analyzer for us-east-1	April 14, 2025, 12:54:49 (UTC-03:00)
<input type="radio"/>	<u>cidade-na-nuvem-cartorio</u>	US East (N. Virginia) us-east-1	View analyzer for us-east-1	April 14, 2025, 10:57:37 (UTC-03:00)


```
75
76 # Banco da Cidade (S3 com segurança) - Armazenamento Seguro
77 resource "aws_s3_bucket" "banco_cidade" {
78   bucket = "cidade-na-nuvem-banco"
79
80   # Habilitando a criptografia no S3 para aumentar a segurança
81   server_side_encryption_configuration {
82     rule {
83       apply_server_side_encryption_by_default {
84         sse_algorithm = "AES256"
85       }
86     }
87   }
```

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 círculo 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.

```
.terraform
.terraform.lock.hcl
main.tf
terraform.tfstate
terraform.tfstate.backup

119
120 # Criando o Cliente do Banco (usuário IAM)
121
122 resource "aws_iam_user" "usuario_banco" {
123   name = "clienteBanco"
124 }
125
126 #Ligando o Cliente às Permissões
127
128 resource "aws_iam_user_policy_attachment" "usuario_banco_policy" {
129   user          = aws_iam_user.usuario_banco.name
130   policy_arn    = aws_iam_policy.acesso_banco.arn
131 }
132
133 #Gerando as Chaves de Acesso do Cliente
134 # Isso representa o cartão do banco que permite o cliente movimentar o dinheiro.
135 resource "aws_iam_access_key" "usuario_banco_key" {
136   user = aws_iam_user.usuario_banco.name
137 }
138
139 # Saída das credenciais
140
141 output "cliente_banco_access_key_id" {
142   value     = aws_iam_access_key.usuario_banco_key.id
143   sensitive = true
144 }
145
146 output "cliente_banco_secret_access_key" {
147   value     = aws_iam_access_key.usuario_banco_key.secret
148   sensitive = true
149 }
150
```

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.

< Users (3) info

An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.

Q Search

<input type="checkbox"/>	User name	Path	Group	Last activity	MFA	Password age	Console last sign-in	Access key ID	Active key age
<input type="checkbox"/>	AmazonFullAccess	/	0	1 hour ago	-	-	-	Active - AKIAYTNFWT7...	2 days
<input type="checkbox"/>	clienteBanco	/	0	-	-	-	-	Active - AKIAYTNFWT7...	Now
<input type="checkbox"/>	usuario-cidade	/	0	-	-	-	-	-	-

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.

```
149 # Correios da cidade SNS + CloudWatch
150
151
152
153 # Caixa Postal dos Correios (SNS Topic)
154 resource "aws_sns_topic" "caixa_postal_cidade" {
155   name = "caixa-postal-cidade"
156 }
157
158 # Endereço de entrega da carta (Assinatura de email)
159 resource "aws_sns_topic_subscription" "endereco_entrega_email" {
160   topic_arn = aws_sns_topic.caixa_postal_cidade.arn
161   protocol = "email"
162   endpoint = "seuemail@exemplo.com" # Altere para seu e-mail real
163 }
164
165 # Funcionário dos Correios observando o Servidor Principal (Alarme de CPU)
166 resource "aws_cloudwatch_metric_alarm" "correios_olheiro_servidor" {
167   alarm_name = "CorreioAvisoCPUAlta-ServidorPrincipal"
168   comparison_operator = "GreaterThanThreshold"
169   evaluation_periods = "1"
170   metric_name = "CPUUtilization"
171   namespace = "AWS/EC2"
172   period = "300"
173   statistic = "Average"
174   threshold = "70"
175   alarm_description = "O funcionário dos Correios vai enviar uma carta se a CPU do servidor passar de 70%."
176   alarm_actions = [aws_sns_topic.caixa_postal_cidade.arn]
177   dimensions = {
178     InstanceId = aws_instance.cidade-servidor.id
179   }
180 }
```

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.

Amazon SNS > Topics

Amazon SNS

Dashboard

Topics

Subscriptions

▼ Mobile

Push notifications

Text messaging (SMS)

New Feature
Amazon SNS now supports High Throughput FIFO topics. [Learn more](#)

Topics (1)

Search

Name	Type	ARN
caixa-postal-cidade	Standard	arn:aws:sns:us-east-1:591442396880:caixa-postal-cidade

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

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

```

181
182 # FÁBRICA DE CONTÊINERES (EC2 + Docker)
183
184 # Criando a Fábrica de Contêineres (Instância EC2)
185 resource "aws_instance" "fabrica_containeres" {
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 }
204

```

- 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.

- `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.

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	ELB
ServidorPrincipal	i-0d6d86c15d0e2ecf0	Stopped	t2.micro	-	View alarms +	us-east-1b	-	-	-
Casa 1	i-08c6d841d54c38cd	Stopped	t2.micro	-	View alarms +	us-east-1a	-	-	-
FabricaDeConteineres	i-07ea702717ca272d9	Stopping	t2.micro	2/2 checks passes	View alarms +	us-east-1a	-	-	-

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.



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. Destruímos tudo para evitar cobranças