APRENDENDO TERRAFORM COM AWS

Sobre o autor: Walker Viana
DevOps Engineer | SRE | Cloud Infrastructure Specialist
Profissional com 25 anos de experiência em TI, especializado em automação de infraestrutura, CI/CD e arquiteturas cloud-native. Apaixonado por transformar processos manuais em soluções automatizadas e escaláveis.

https://github.com/skywalker2077

1: Introdução ao Terraform:

1. O Terraform é uma ferramenta de infraestrutura como código (IaC) que permite construir, modificar e versionar a infraestrutura de forma segura e eficiente.
    - Terraform é Infraestrutura como Código.
    - Automatize sua infraestrutura.
    - Mantenha sua infraestrutura em um estado específico (em conformidade).
    - Torne sua infraestrutura auditável.
        - Você pode manter o histórico de alterações da sua infraestrutura em um sistema de controle de versão como o GIT.
    - Ansible, Chef, Puppet e Saltstack têm como foco a automação da instalação
      Its Ok for me.
        - Manter as máquinas em conformidade, em um determinado estado.
    - Terraform pode automatizar o provisionamento da própria infraestrutura.
        - Funciona bem com softwares de automação como o Ansible para instalação de softwares após o provisionamento da infraestrutura.

1.2 Instalação do Terraform no Linux:

1. **Método 1**: Instalação via repositório oficial (recomendado)

```
# Instalar dependências
sudo apt-get update && sudo apt-get install -y gnupg
software-properties-common

# Adicionar a chave GPG da HashiCorp
wget -O- https://apt.releases.hashicorp.com/gpg | \
gpg --dearmor | \
sudo tee /usr/share/keyrings/hashicorp-archive-keyring.gpg

# Adicionar o repositório oficial
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] \
https://apt.releases.hashicorp.com $(lsb_release -cs) main" | \
sudo tee /etc/apt/sources.list.d/hashicorp.list

# Atualizar e instalar
sudo apt update
sudo apt install terraform
```

2. **Método 2**: Download direto (funciona em qualquer distribuição Linux)

```
# Baixar a versão mais recente (verifique a versão atual no site)
wget https://releases.hashicorp.com/terraform/1.9.0/terraform_1.9.0_linux_amd64.zip

# Descompactar
unzip terraform_1.9.0_linux_amd64.zip

# Mover para o PATH
sudo mv terraform /usr/local/bin/

# Dar permissão de execução
sudo chmod +x /usr/local/bin/terraform
```

```
container1:/app# wget https://releases.hashicorp.com/terraform/1.9.0/terraform_1.9.0_linux_amd64.zip
--2025-10-30 19:17:50--  https://releases.hashicorp.com/terraform/1.9.0/terraform_1.9.0_linux_amd64.zip
Resolving releases.hashicorp.com (releases.hashicorp.com)... 18.161.205.115, 18.161.205.57, 18.161.205.73, ...
Connecting to releases.hashicorp.com (releases.hashicorp.com)|18.161.205.115|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 27010938 (26M) [application/zip]
Saving to: 'terraform_1.9.0_linux_amd64.zip'

terraform_1.9.0_linux_amd64.zip      100%[=====================================================================>]  25.76M  22.0MB/s    in 1.2s

2025-10-30 19:17:52 (22.0 MB/s) - 'terraform_1.9.0_linux_amd64.zip' saved [27010938/27010938]

container1:/app#
```

3. Verifique o Terraform usando o seguinte comando.
   terraform --version

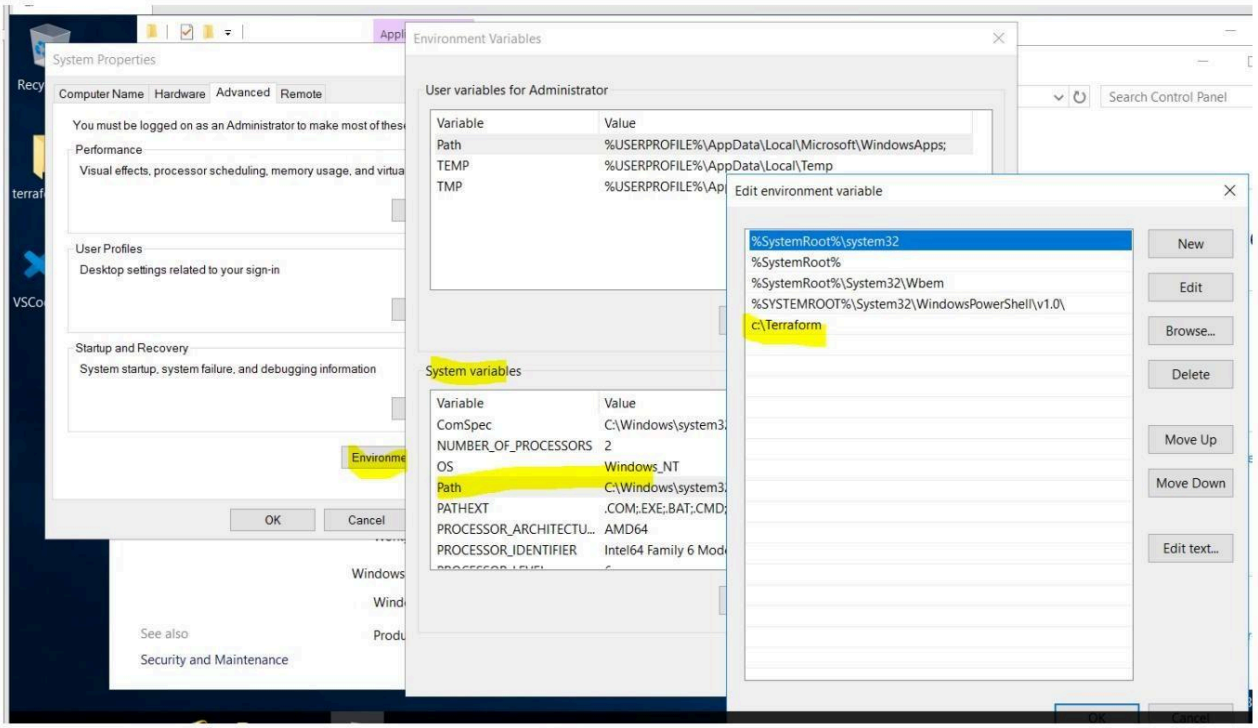
```
container1:/app# terraform --version
Terraform v1.9.0
on linux_amd64

Your version of Terraform is out of date! The latest version
is 1.13.4. You can update by downloading from https://www.terraform.io/downloads.html
container1:/app#
```

1.3 Instalação do Terraform no Windows:

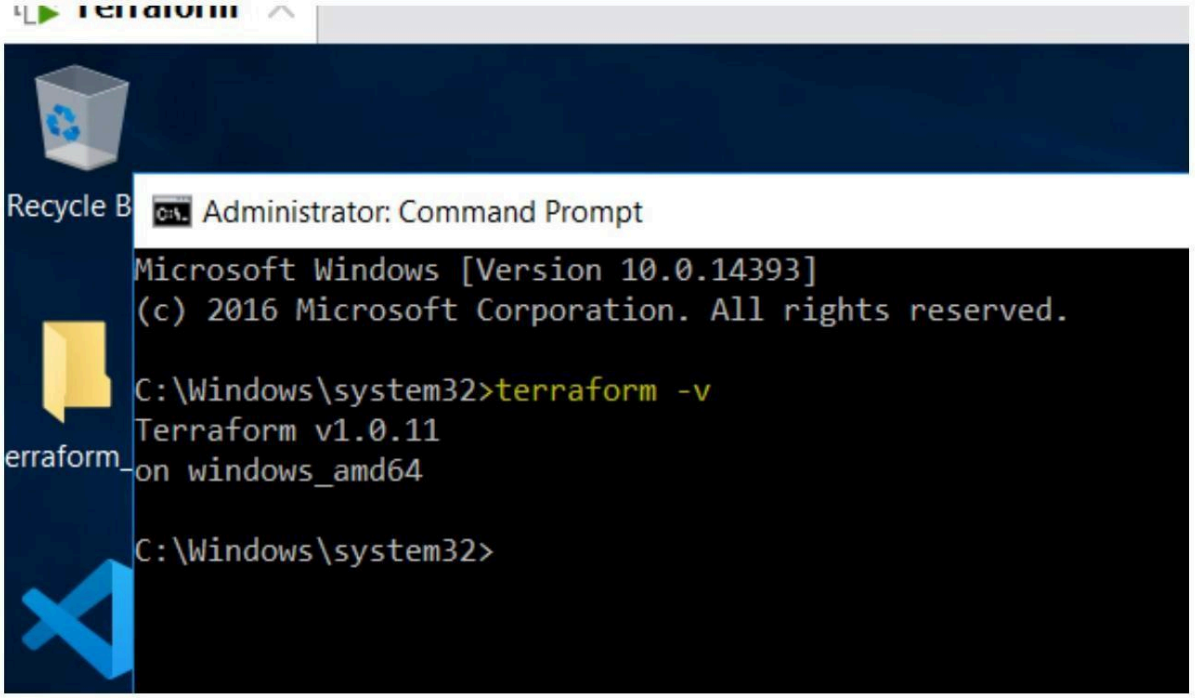
1) Baixe o Terraform do site oficial do Terraform, de acordo com o seu sistema operacional.
https://www.terraform.io/downloads.html

2) Copie o arquivo da pasta Downloads para a pasta C:\Terraform.

3) Crie a variável de ambiente conforme mostrado abaixo.



4) Verifique a versão com **terraform -v**



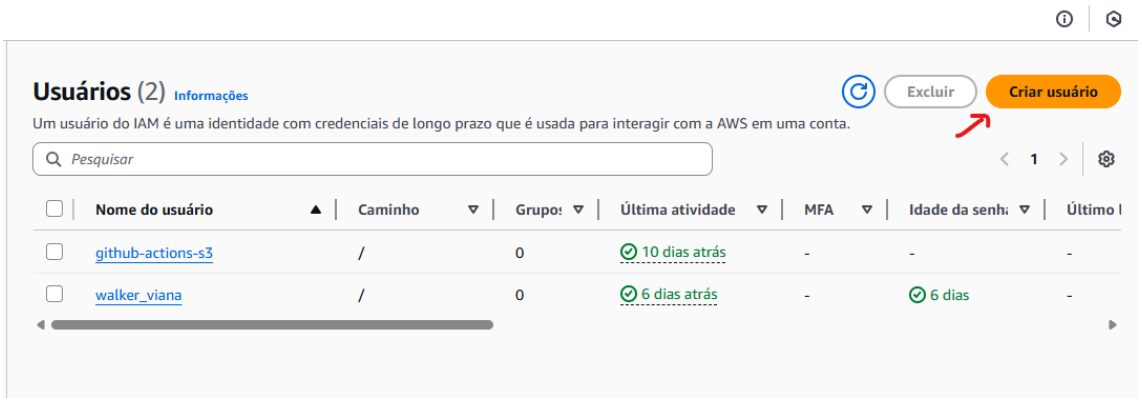
2: Noções básicas de Terraform:

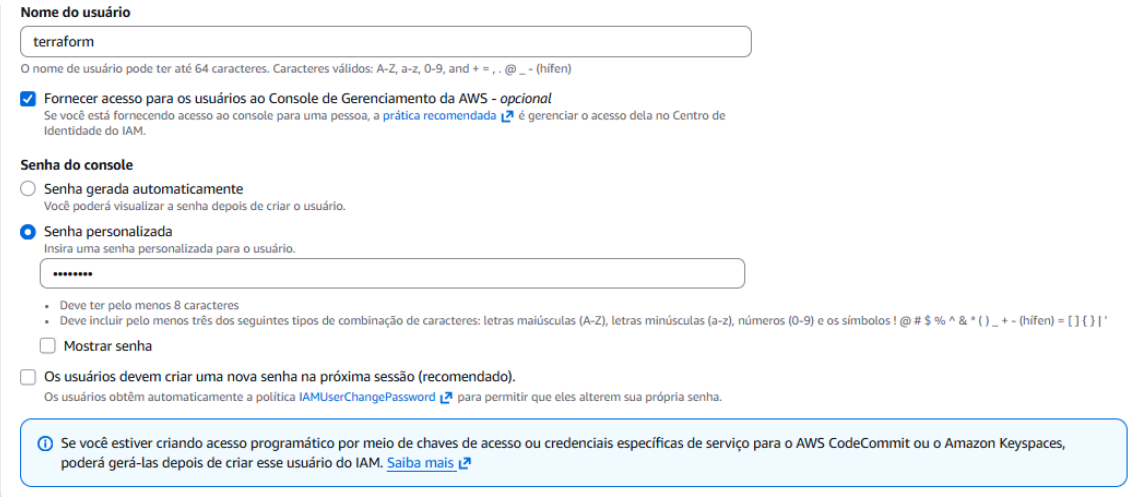
2.1: Compreendendo a HCL (Hashicorp Configuration Language) do Terraform.

1) Criando uma instância na AWS:
   - Abra uma conta da AWS.
   - Crie um usuário administrador do IAM.
   - Crie um arquivo Terraform para criar a instância t2.micro •
   - Execute o comando `terraform apply`.

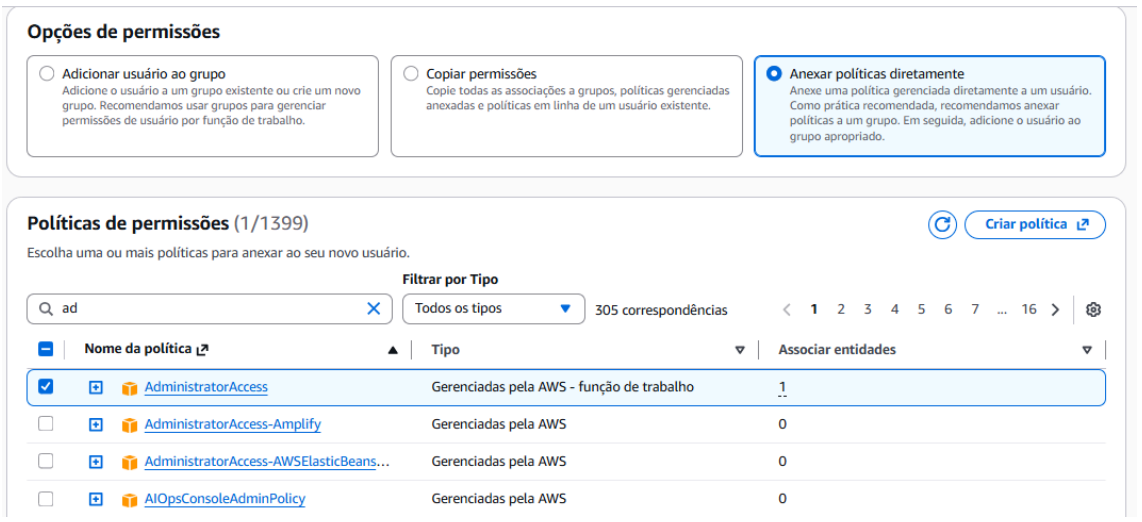
2) Faça login no console de gerenciamento da AWS usando seu ID e senha.
3) Acesse o serviço chamado IAM

   ➡️ Crie um novo usuário chamado “terraform” ;

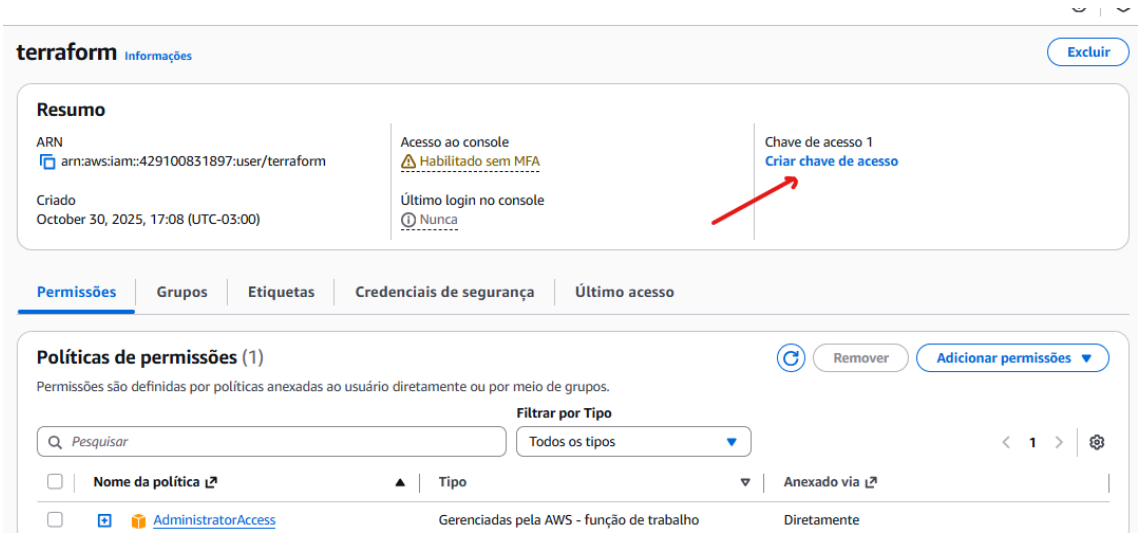


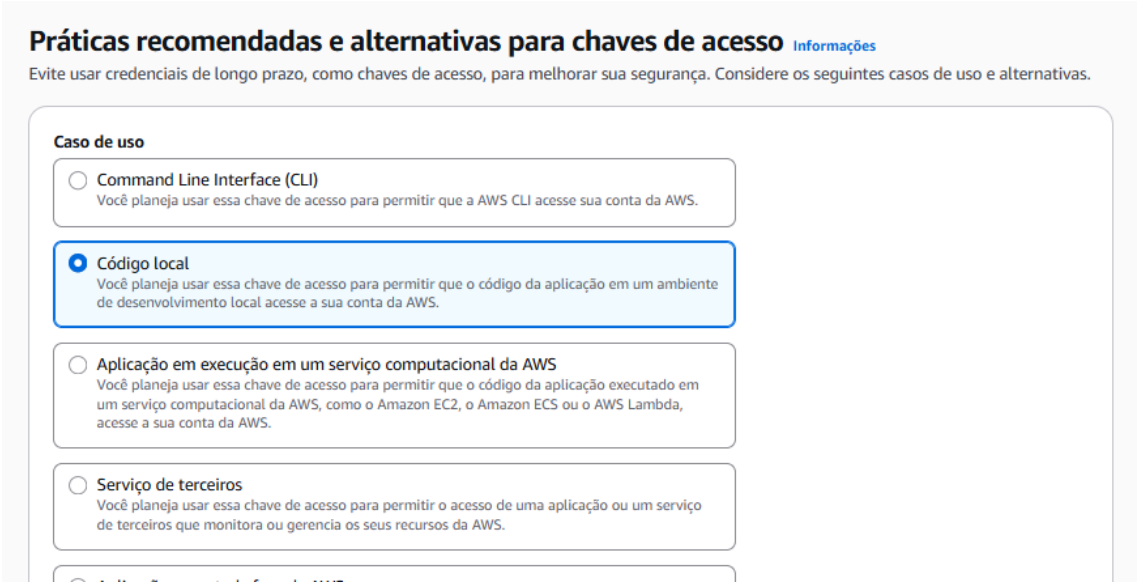


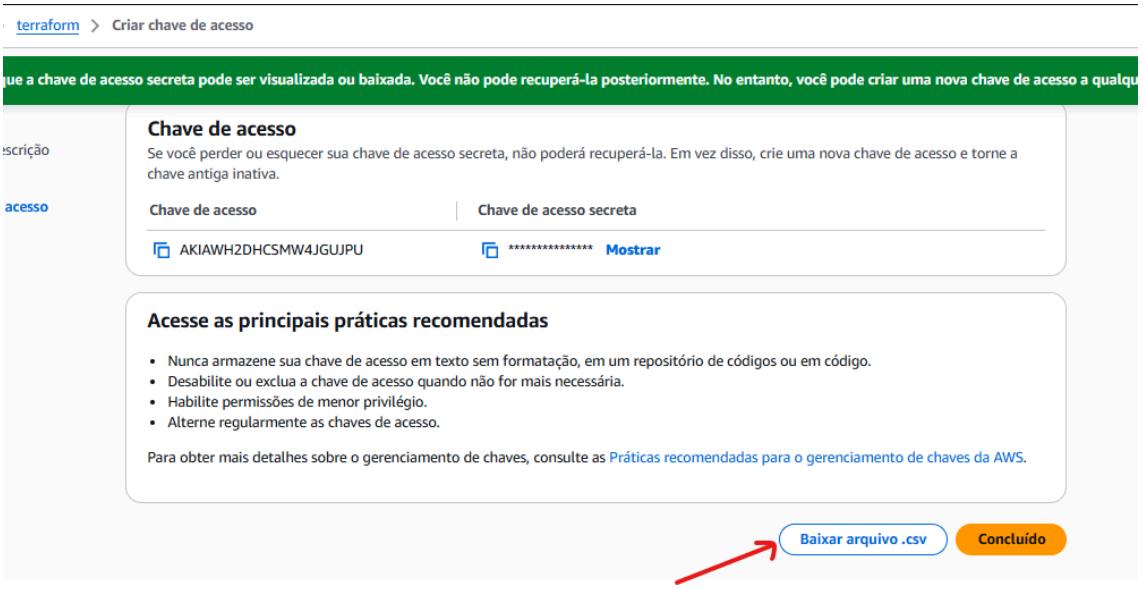
➡️ Conceda o acesso apropriado ao usuário terraform;



➡️ Criar chave de acesso;



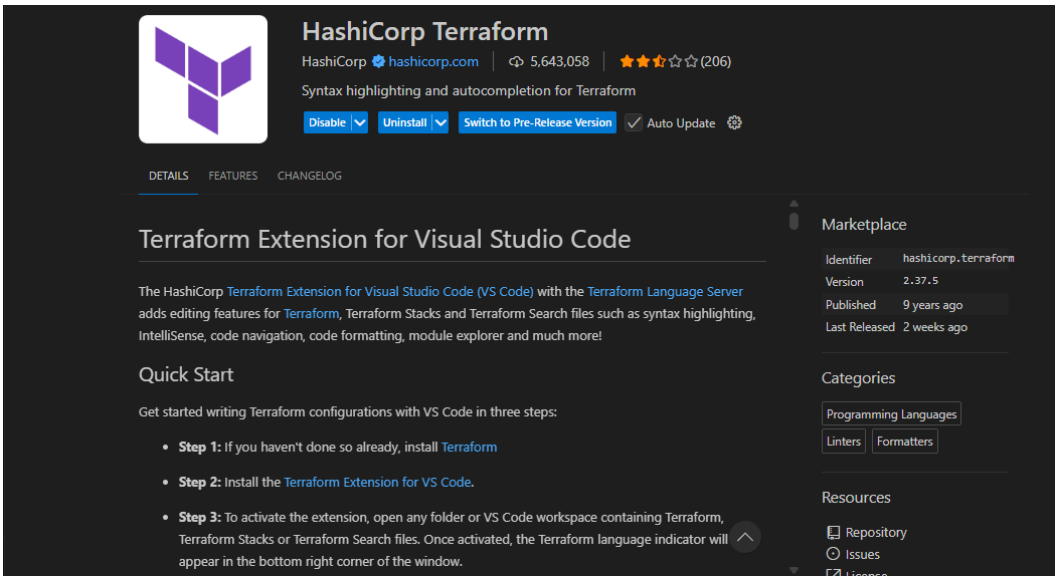




➡️ Memorize ou baixe o arquivo com as chaves de acesso(obs.: a chave só será mostrada uma vez);

2.2. Criando a instância usando o Terraform na AWS:

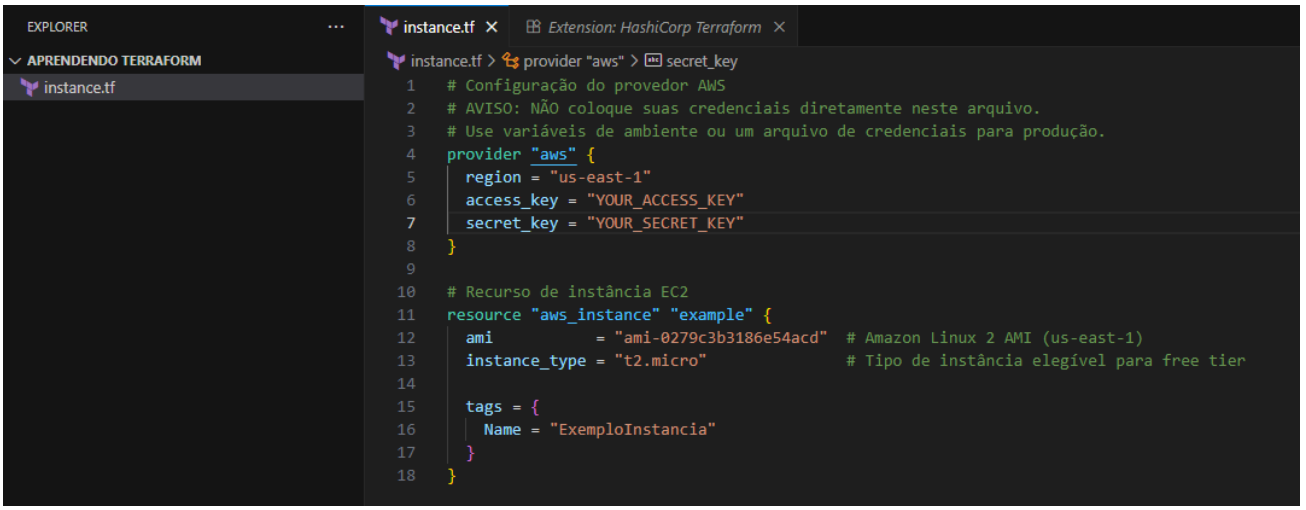
1) Instale o MS Visual Studio em uma máquina Windows e instale o plugin Terraform no VS Code.



2) Crie um arquivo chamado instance.tf no VS Code, conforme mostrado abaixo.

```
# Configuração do provedor AWS
# AVISO: NÃO coloque suas credenciais diretamente neste arquivo.
# Use variáveis de ambiente ou um arquivo de credenciais para produção.

provider "aws" {
  region = "us-east-1"
  access_key = "YOUR_ACCESS_KEY"
  secret_key = "YOUR_SECRET_KEY"
}

# Recurso de instância EC2
resource "aws_instance" "example" {
  ami           = "ami-0279c3b3186e54acd"          # Amazon Linux 2 AMI (us-east-1)
  instance_type = "t2.micro"                       # Tipo de instância elegível para free tier

  tags = {
    Name = "ExemploInstancia"
  }
}
```

3) Deve ficar desta forma:



4) Inicialize o Terraform usando o seguinte comando.

terraform init

```
PS C:\Users\Usuário\VS\Aprendendo Terraform> terraform init
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v6.18.0

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

5) Valide as configurações usando o seguinte comando.

terraform validate

```
PS C:\Users\Usuário\VS\terraform> terraform validate
Success! The configuration is valid.
```

6) Planejamento das Mudanças

Visualize quais recursos seriam criados/modificados sem aplicar as mudanças:

terraform plan

```
      + private_dns_name_options (known after apply)

      + root_block_device (known after apply)
    }

Plan: 1 to add, 0 to change, 0 to destroy.
```

- ***Salvando o Plano em Arquivo***

Para salvar o plano em um arquivo e aplicá-lo posteriormente:

terraform plan -out out.terraform

7) Agora execute o comando ´terraform apply` usando o seguinte comando:

terraform apply ou **terraform apply -auto-approve**

**Atenção: É considerado arriscado usar -auto-approve em ambientes de produção sem revisar o plano primeiro, pois pode executar mudanças destrutivas sem aviso.*

```
  Enter a value: yes

aws_instance.example: Creating...
aws_instance.example: Still creating... [00m10s elapsed]
aws_instance.example: Creation complete after 15s [id=i-0a982b716ccffc65c]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
PS C:\Users\Usuário\VS\Aprendendo Terraform>
```

- ***Aplicação de Plano Salvo***

Aplicar um plano previamente salvo (não solicita confirmação):

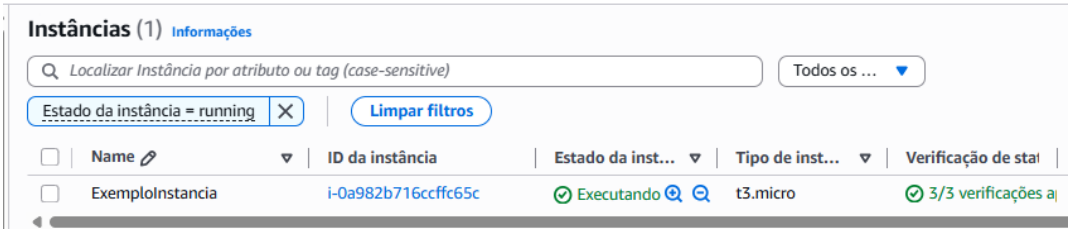
terraform apply "out.terraform"

8) Agora podemos verificar se a instância foi criada no console da AWS.

terraform show

```
PS C:\Users\Usuário\VS\Aprendendo Terraform> terraform show
# aws_instance.example:
resource "aws_instance" "example" {
    ami                                  = "ami-0c02fb55956c7d316"
    arn                                  = "arn:aws:ec2:us-east-1:429100831897:instance/i-0a982b716ccffc65c"
    associate_public_ip_address          = true
    availability_zone                    = "us-east-1c"
    disable_api_stop                     = false
    disable_api_termination              = false
    ebs_optimized                        = false
    force_destroy                        = false
    get_password_data                    = false
    hibernation                          = false
    host_id                              = null
    iam_instance_profile                 = null
    id                                   = "i-0a982b716ccffc65c"
    instance_initiated_shutdown_behavior = "stop"
    instance_lifecycle                   = null
    instance_state                       = "running"
    instance_type                        = "t3.micro"
```

Na console da AWS vai ficar assim:

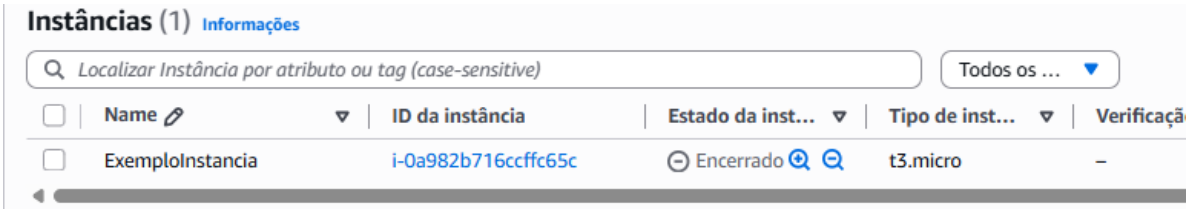


9) Também podemos excluir esta instância usando o comando abaixo.

terraform destroy

```
  Enter a value: yes

aws_instance.example: Destroying... [id=i-0a982b716ccffc65c]
aws_instance.example: Still destroying... [id=i-0a982b716ccffc65c, 00m10s elapsed]
aws_instance.example: Destruction complete after 12s

Destroy complete! Resources: 1 destroyed.
```

10) Valide o mesmo no console da AWS.



11) Fluxo Completo com Arquivo de Mudanças (quando você quer revisar e aplicar mudanças específicas):

1. Criar e salvar o plano

   terraform plan -out plano-mudancas.terraform

2. Aplicar apenas essas mudanças

   terraform apply plano-mudancas.terraform

3. Remover o arquivo de plano (boa prática)

   rm plano-mudancas.terraform

12) Resumo do Fluxo de Trabalho

1. **terraform init** → Inicializa o projeto
2. **terraform validate** → Valida as configurações
3. **terraform plan** → Visualiza mudanças
4. **terraform apply** → Aplica mudanças
5. **terraform destroy** → Remove infraestrutura

13) Boas Práticas

1. Sempre execute **terraform plan** antes de **apply**.
2. Use **-out** para salvar planos em ambientes de produção.
3. Remova arquivos de plano após aplicá-los.
4. Revise cuidadosamente antes de executar **destroy**.

2.3: Terraform Tipos de variáveis:

1. As variáveis do Terraform foram completamente reformuladas para a versão 0.12 do Terraform.
2. Agora você pode ter mais controle sobre as variáveis e usar loops for e for-each o que não era possível com versões anteriores.
3. Não é necessário especificar o tipo nas variáveis, mas é recomendável.
4. Tipos de variáveis simples do Terraform.

• **Number** - para números (inteiros ou decimais)

• **String** - para texto

• **Bool** - para verdadeiro/falso

```
# Variáveis com tipos básicos no Terraform

# String - texto
variable "a-string" {
  type        = string
  description = "Uma variável do tipo texto"
}

# Number - número (inteiro ou decimal)
variable "this-is-a-number" {
  type        = number
  description = "Uma variável do tipo numérico"
}

# Boolean - verdadeiro ou falso
variable "true-or-false" {
  type        = bool
  description = "Uma variável do tipo booleano"
}

# Exemplos com valores padrão
variable "region" {
  type        = string
  default     = "us-east-1"
  description = "Região AWS padrão"
}

variable "instance-count" {
  type        = number
  default     = 2
  description = "Número de instâncias a criar"
}

variable "enable-monitoring" {
  type        = bool
  default     = true
  description = "Habilita ou desabilita monitoramento"
}
```

5. Tipos complexos do Terraform:
    1. List(type)
    2. set(type)
    3. Map(type)
    4. Object({<ATTR NAME> = <TYPE>,..})
    5. Tuple([<TYPE>,...])

6. Lista e Mapa, que já abordamos em uma demonstração anterior.
    1. Lista: [0, 1, 5, 2]
    2. Mapa: {"chave" = "valor"}

7. Uma lista é sempre ordenada, ela sempre retornará 0,1,5,2 e não 5,1,2,0.
8. Um "conjunto" é como uma lista, mas não mantém a ordem em que você o inseriu e só pode conter valores únicos.

   Uma lista que tem [5,1,1,2] torna-se [1,2,5] em um conjunto.
9. Um objeto é como um mapa, mas cada elemento pode ter um tipo diferente.
10. Uma tupla é como uma lista, mas cada elemento pode ter um tipo diferente.
11. Os tipos mais comuns são lista e mapa; os outros são usados apenas esporadicamente.
12. Os que você deve lembrar são os tipos de variáveis simples: **string**, **number**, **bool**, **list** e **map**.
13. Você também pode deixar o Terraform decidir a tipologia. * basta omitir o atributo **type**

2.4 Variáveis no Terraform:

1. Estrutura de Projeto Terraform – AWS

```
projeto-terraform/
├---- providers.tf
├---- variable.tf
├---- terraform.tfvars
└--- instancia.tf
```

2. providers.tf

- Configuração do provider AWS com credenciais:

```
provider "aws" {
  access_key      = var.AWS_ACCESS_KEY
  secret_key      = var.AWS_SECRET_KEY
  region          = var.AWS_REGION
}
```

3. variable.tf

- Declaração das variáveis (sem valores sensíveis):

```
variable "AWS_ACCESS_KEY" {
  type            = string
  description     = "AWS Access Key ID"
}

variable "AWS_SECRET_KEY" {
  type            = string
  description     = "AWS Secret Access Key"
  sensitive       = true
}

variable "AWS_REGION" {
  type            = string
  description     = "Região AWS para criar recursos"
  default         = "us-east-1"
}
```

4. terraform.tfvars

- Arquivo com os valores das variáveis: (⚠️ ATENÇÃO: Não commitar no Git!).

```
AWS_ACCESS_KEY      = "sua-access-key-aqui"
AWS_SECRET_KEY      = "sua-secret-key-aqui"
AWS_REGION          = "us-east-1"
```

- Adicione ao .gitignore:

```
terraform.tfvars
*.tfvars
```

5. instancia.tf

- Recursos da infraestrutura (exemplo de instância EC2):

```
resource "aws_instance" "exemplo" {
  ami                         = "ami-0c55b159cbfafe1f0"
  instance_type          = "t2.micro"

  tags = {
    Name = "Minha-Instancia-Terraform"
  }
}
```

2.5 Provisionamento de Software:

1. Você pode criar sua própria AMI personalizada e incluir seu software na imagem.
   - Usar Ansible como provisioner do Packer para criar imagens:
   - Usando ferramentas de automação como Chef, Puppet e Ansible.

2. Estado atual do Terraform com automação:

- O Chef está integrado ao Terraform; você pode adicionar instruções Chef.
- Você pode executar o agente Puppet usando execução remota.
- Para o Ansible, você pode primeiro executar o Terraform e exibir os endereços IP.
- Execute Ansible-Playbooks nesses hosts.
- Isso pode ser automatizado em um script de fluxo de trabalho.
- Existem iniciativas de terceiros integrando o Ansible com o Terraform.

2.6 Atributos de saída:

1. O Terraform armazena os atributos de todos os recursos que você cria. Exemplo: O recurso aws_instance possui o atributo public_ip.

2. Esses atributos podem ser consultados e exibidos.
3. Isso pode ser útil simplesmente para gerar informações valiosas ou para fornecer informações a fontes externas. software.
4. Use “output” para exibir o endereço IP público de um recurso da AWS:

```
resource "aws_instance" "example" {
  ami                = lookup(var.AMIS, var.AWS_REGION)
  instance_type      = "t2.micro"

  tags = {
    Name = "ExampleInstance"
  }
}

output "ip" {
value = "${aws_instance.example.public_ip}"
}
```

5. Você pode se referir a qualquer atributo especificando os seguintes elementos em suas variáveis:

   - Tipo de recurso:
     aws_instance
   - Nome do recurso:
     example
   - Nome do atributo:
     public_ip

6. Você também pode usar os atributos em um script:

```
resource "aws_instance" "example" {
  ami                    = lookup(var.AMIS, var.AWS_REGION)
  instance_type       = "t2.micro"

  provisioner "local-exec" {
    command = "echo ${self.private_ip} >> private_ips.txt"
  }

  tags = {
    Name = "ExampleInstance"
  }
}

# Arquivo de variáveis necessário
variable "AWS_REGION" {
  description          = "Região AWS"
  type                    = string
  default                = "us-east-1"
}

variable "AMIS" {
  description          = "AMIs por região"
  type                    = map(string)
  default  = {
    us-east-1 = "ami-0c55b159cbfafe1f0"
    us-west-2 = "ami-0d70546e43a941d70"
    eu-west-1 = "ami-0d71ea30463e0ff8d"
  }
}

# Output opcional para ver o IP
output "instance_private_ip" {
  description          = "IP privado da instância"
  value                   = aws_instance.example.private_ip
}
```

7. Como usar:

   1. **Inicializar:**

      Comando: `terraform init`

   2. **Planejar:**

      Comando: `terraform plan`

   3. **Aplicar:**

      Comando: `terraform apply`

   4. **Verificar Saída:**

      Comando: `cat private\_ips.txt`

8. Útil, por exemplo, para iniciar scripts de automação após o provisionamento da infraestrutura.

9. Você pode inserir os endereços IP em um arquivo de hosts do Ansible.

10. Ou outra possibilidade: executar um script (com atributos como argumento) que cuidará disso. de um mapeamento entre o nome do recurso e o endereço IP.

2.7 Estado Remoto:

1. O Terraform mantém o estado remoto da infraestrutura.
2. Ele armazena isso em um arquivo chamado **terraform.tfstate**.
3. Existe também um backup do estado anterior em **terraform.tfstate.backup**.
4. Ao executar o comando **terraform apply,** um novo arquivo **terraform.tfstate** e um backup são criados.
5. É assim que o Terraform controla o estado remoto.
   • Se o estado remoto mudar e você executar **terraform apply** novamente, o Terraform fará as alterações necessárias para corresponder ao estado remoto correto.
   • Por exemplo: você encerra uma instância gerenciada pelo Terraform, após o comando **terraform apply** será iniciado novamente.
6. Você pode manter o arquivo terraform.tfstate no controle de versão.

   Exemplo: GIT

7. Ele fornece um histórico do seu arquivo terraform.tfstate.
8. Ele permite que você colabore com outros membros da equipe.
   - Infelizmente, podem ocorrer conflitos quando duas pessoas trabalham ao mesmo tempo.
9. O modelo local funciona bem no início, mas quando seu projeto cresce, você pode armazenar seu estado remotamente?
10. O estado do Terraform pode ser salvo remotamente, usando a funcionalidade de backend do Terraform.
11. O padrão é um backend local (o arquivo de estado local do Terraform).
12. Outros sistemas de backend incluem:
    - S3 (com mecanismo de bloqueio usando DynamoDB)
    - Consul (com bloqueio)
    - Terraform enterprise (a solução comercial)

13. Utilizar a funcionalidade de backend traz benefícios:
    - Trabalho em equipe: permite a colaboração; o trabalho remoto estará sempre disponível para toda a equipe.
    - O arquivo de estado não é armazenado localmente. Informações potencialmente sensíveis agora são armazenadas apenas no estado remoto.
    - Alguns backends permitem operações remotas. O comando **terraform apply** será executado completamente de forma remota. Esses são chamados de backends aprimorados. https://www.terraform.io/docs/backends/types/index.html

14. tem 2 etapas para configurar um estado remoto:
    - Adicione o código de backend a um arquivo .tf.
    - Execute o processo de inicialização 15.
15. Para configurar um armazenamento remoto do Consul, você pode adicionar um arquivo backend.tf com o seguinte conteúdo:

```
terraform {
  backend "consul" {
    address = "demo.consul.io"      # nome do host do cluster Consul
    path    = "terraform/myproject"  # caminho para armazenar o estado
  }
}
```

16. Você também pode armazenar seu estado no S3:

```
terraform { backend
"s3"{ bucket = "mybucket"
key = "terraform/myproject" region = "eu-west-1" }
}
```

17. Ao usar um estado remoto S3, é melhor configurar as credenciais da AWS:

```
Bash

aws configure
```

Depois:

```
Código

AWS Access Key ID [None]: AWS-key
AWS Secret Access Key [None]: AWS_secret_key
Default region name [None]: eu-west-1
Default output format [None]: json
```

18. Próximo passo: **terraform init**
19. Usar um armazenamento remoto para o estado do Terraform garantirá que você sempre tenha a versão mais recente do estado.
20. Isso evita ter que confirmar e enviar o arquivo terraform.tfstate para o controle de versão.
21. Os armazenamentos remotos do Terraform nem sempre suportam bloqueio
    - A documentação sempre menciona se o bloqueio está disponível para um armazenamento remoto.
    - S3 e Consul oferecem suporte a isso.
22. Você também pode especificar um armazenamento remoto (somente leitura) diretamente no **arquivo .tf.** data "terraform_remote_state"

```
terraform {
  backend "s3" {
    bucket     = "terraform-state"
    key        = "terraform.tfstate"
    region     = var.AWS_REGION
    access_key = var.AWS_ACCESS_KEY
    secret_key = var.AWS_SECRET_KEY
  }
} backend "s3" {
    bucket     = "mybucket"              # nome do bucket S3
    key        = "terraform/myproject"   # caminho do arquivo de estado
    region     = "eu-west-1"             # região da AWS
  }
}
```

23. Isso só é útil como um feed somente leitura do seu arquivo remoto.
24. É uma fonte de dados.
25. É muito útil para gerar resultados.

2.8 Fontes de dados:

1. Para determinados provedores (como a AWS), o Terraform fornece fontes de dados.
2. As fontes de dados fornecem informações dinâmicas.
    - A AWS disponibiliza uma grande quantidade de dados em formato estruturado por meio de sua API.
    - O Terraform também expõe essas informações usando fontes de dados.
    - Exemplo: Lista de AMIs ou Lista de Zonas de Disponibilidade.

3. Outro ótimo exemplo é a fonte de dados que fornece todos os endereços IP em uso pela AWS.
4. Isso é ótimo se você quiser filtrar o tráfego com base em uma região da AWS.

   - Exemplo: Permitir todo o tráfego de instâncias da Amazon na Europa.

5. O filtro de tráfego na AWS pode ser feito usando grupos de segurança.
   - O tráfego de entrada e saída pode ser filtrado por protocolo, intervalo de IP e porta.
   - Semelhante ao Iptables (Linux) ou a um dispositivo de firewall.

```
data "aws_ip_ranges" "european_ec2" {
  regions  = ["eu-west-1", "eu-central-1"]
  services = ["ec2"]
}

resource "aws_security_group" "from_europe" {
  name = "from_europe"

  ingress {
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = slice(data.aws_ip_ranges.european_ec2.cidr_blocks, 0, 50)
  }

  tags = {
    CreateDate = data.aws_ip_ranges.european_ec2.create_date
    SyncToken  = data.aws_ip_ranges.european_ec2.sync_token
  }
}
```

2.9) Modelos (Templates):

1. O fornecedor de modelos pode ajudar na criação de arquivos de configuração personalizados.
2. Você pode criar modelos com base em variáveis de atributos de recursos do Terraform (por exemplo, um endereço de IP público).
3. O resultado é uma string que pode ser usada como variável no Terraform.
   - A string contém um modelo. Ex.: um arquivo de configuração.
   - Pode ser usada para criar modelos genéricos ou configurações do Cloud-Init.
   - **Cloud-Init** é um padrão da indústria para inicializar e configurar instâncias de máquinas virtuais na nuvem durante o primeiro boot (inicialização).
5. Na AWS, você pode passar comandos que precisam ser executados quando a instância for iniciada pela primeira vez.
6. Na AWS, isso é chamado de "user-data".
7. Se você quiser passar dados do usuário que dependem de outras informações no Terraform (por exemplo, IP). endereço), você pode usar o modelo fornecido.

8. Primeiro, você pode criar um arquivo de modelo: #!/bin/bash

   echo "database-ip" = ${myip} >>/etc/myapp.config

9. Em seguida, você cria um recurso template_file que lerá o arquivo de modelo e substituirá ${myip} pelo endereço IP de uma instância da AWS criada pelo Terraform.

   dados "template_file" "my-template" { template
   = "${file("templates/init.tpl")}"

   vars{ myip = "${aws_instance.database1.private_ip} }
   }

10. Em seguida, você pode usar o recurso my-template ao criar uma nova instância.

```
# Criar um recurso de servidor web
resource "aws_instance" "web" {
  ami           = "ami-xxxxxxxxx"    # Especifique a AMI desejada
  instance_type = "t2.micro"         # Especifique o tipo de instância

  user_data = data.template_file.my-template.rendered

  tags = {
    Name = "web-server"
  }
}

# Exemplo de template_file (se ainda não existir)
data "template_file" "my-template" {
  template = file("${path.module}/user-data.sh")

  vars = {
    # Adicione suas variáveis aqui
  }
}
```

11. Quando o Terraform for executado, ele perceberá que primeiro precisa iniciar a instância database1 , em seguida, gerar o modelo e só então inicie a instância web.
12. A instância web terá o modelo injetado no user_data e, quando for iniciada, o user_data criará um arquivo /etc/ myapp.config com o endereço IP do banco de dados.
13. Primeiro, crie um arquivo de modelo:
    #!/ bin/bash echo
    "database-ip = ${myip}" >>/etc/myapp.config

2.10) Outros Fornecedores:

1. O Terraform é uma ferramenta para criar e gerenciar recursos de infraestrutura.
2. O Terraform oferece diversas opções de provedores.
3. A AWS é a mais popular e será a que abordaremos neste guia.
4. Potencialmente, qualquer empresa que disponibilize uma API pode ser usada como fornecedora do Terraform.
5. Alguns outros exemplos de provedores de nuvem compatíveis com o Terraform são:
   - Google Cloud
   - Azure
   - Heroku
   - Digital Ocean
6. Nuvem privada/local: VMWare vSphere & vCloud/ / OpenStack
7. Não se limita apenas a provedores de nuvem:
   - Datadog – monitoramento
   - GitHub – controle de versão
   - Mailgun – envio de e-mails (SMTP)
   - DNSSimple / DNSMadeEASy / UltraDNA – hospedagem de DNS 8.
     A lista completa pode ser encontrada em
     https://www.terraform.io/docs/providers/index.html
8. Os recursos do Terraform para outros provedores são muito semelhantes.

2.11) Módulos:

1. Você pode usar módulos para tornar seu Terraform mais organizado.

```
module-example/
    ├── variables.tf    # Parâmetros de entrada
    ├── cluster.tf      # Recursos a serem criados
    └── outputs.tf      # Valores de saída
```

2. Utilize módulos de terceiros, por exemplo: Reutilize módulos do GitHub.

3. Reutilize partes do seu código, por exemplo: para configurar a rede na AWS - Rede Virtual Privada (VPC)

4. Use um módulo do Git:

```
module "module-example" {
        source = "github.com/sundara/terraform-module-example"
}
```

5. Use um módulo de uma pasta local:

```
module "module-example" {
        source = "./module-example"
}
```

6. Passe os argumentos para o Módulos no Terraform

```
module "module-example" {
        source = "./module-example"
        region = "us-west-1"
        ip-range = "10.0.0.0/8" cluster-size = "3"
}
```

7. Dentro da pasta do módulo, você encontrará novamente os arquivos do Terraform:

`variables.tf` - Definição de Entradas

```
variable "region" {
  description = "Região AWS onde os recursos serão criados"
  type        = string
}

variable "ip-range" {
  description = "Faixa de IPs CIDR para a rede"
  type        = string
}

variable "cluster-size" {
  description = "Tamanho do cluster (número de instâncias)"
  type        = string
}
```

`cluster.tf` - Criação de Recursos

```
# As variáveis podem ser usadas aqui através de var.nome_variavel

resource "aws_instance" "instance-1" {
  ami           = "ami-123456"
  instance_type = "t2.micro"

  tags = {
    Name = "Cluster Instance 1"
  }
}

resource "aws_instance" "instance-2" {
  ami           = "ami-123456"
  instance_type = "t2.micro"

  tags = {
    Name = "Cluster Instance 2"
  }
}
```

```
resource "aws_instance" "instance-3" {
  ami           = "ami-123456"
  instance_type = "t2.micro"

  tags = {
    Name = "Cluster Instance 3"
  }
}
```

8. Utilize a saída do módulo na parte principal do seu código:

```
output "alguma-saida" {
        description = "IPs do cluster AWS"
        value       = module.module-example.aws_cluster
}
```

.

2.12) Visão geral dos comandos do Terraform:

1. Comandos Principais

Comando Terraform	Função
`terraform init`	Inicializa o diretório de trabalho, baixando *providers* e módulos necessários.
`terraform plan`	Exibe um resumo das mudanças que serão aplicadas à infraestrutura.
`terraform apply`	Executa as alterações planejadas na infraestrutura.
`terraform destroy`	Remove toda a infraestrutura gerenciada pelo Terraform.
`terraform validate`	Verifica a sintaxe e a validade dos arquivos de configuração.

2. Comandos de Formatação e Visualização

Comando	Descrição
`terraform fmt`	Reorganiza os arquivos de configuração para aderir ao formato padrão (canônico).
`terraform show`	Apresenta o estado atual da infraestrutura em um formato de fácil leitura.
`terraform graph`	Gera uma representação gráfica (visual) da configuração, no formato DOT.
`terraform output`	Mostra os valores de saída definidos na configuração do projeto.

3. Comandos de Módulos

Comando	Descrição
`terraform get`	Faz o download e atualiza os módulos referenciados.

4. Comandos de Estado

Comando Terraform	Função Principal
`terraform state`	Gerenciamento avançado do arquivo de estado
`terraform state list`	Exibe uma lista de todos os recursos contidos no estado
`terraform state show`	Apresenta os detalhes de um recurso específico no estado
`terraform refresh`	Sincroniza o estado do Terraform com a infraestrutura real
`terraform import`	Inclui recursos já existentes no estado do Terraform

5. Comandos de Workspace

Comando	Descrição
`terraform workspace`	Gerenciamento de ambientes isolados (workspaces).
`terraform workspace list`	Exibe a lista de workspaces disponíveis.
`terraform workspace new`	Cria um novo workspace.
`terraform workspace select`	Define o workspace ativo (seleciona um).

6. Comandos de Marcação de Recursos

Comando Terraform	Função
`terraform taint`	Marca um recurso como "contaminado", forçando sua recriação na próxima aplicação.
`terraform untaint`	Remove a marcação de "contaminado" de um recurso, impedindo sua recriação forçada.

📝 Notas Importantes

- **terraform remote** e **terraform push** foram descontinuados.
  Use backends configurados no bloco `terraform {}`
- Sempre execute `terraform plan` antes de `terraform apply` para revisar mudanças
- Use `terraform fmt` regularmente para manter código padronizado
- O comando `terraform taint` força a recriação de recursos no próximo apply.

Seção 3: O Packer

O Packer é **uma ferramenta DevOps de código aberto criada pela Hashicorp para gerar imagens a partir de um único arquivo de configuração JSON,** o que facilita o rastreamento de alterações a longo prazo. Este software é compatível com diversas plataformas e pode criar várias imagens em paralelo.

- O Packer é uma ferramenta de linha de comando que pode criar AMIs da AWS com base em modelos.
- Em vez de instalar o software após inicializar uma instância, você pode criar uma AMI.

com todo o software necessário já instalado.

- Isso pode acelerar o tempo de inicialização das instâncias.
- É uma abordagem comum quando você executa uma camada de aplicativo com escala horizontal ou um cluster de algo.

3.1 : Terraform com Packer e Jenkins:

1. O Terraform se encaixa perfeitamente em uma organização com mentalidade DevOps.
2. Ferramentas como Terraform e Packer podem ser usadas no Ciclo de Vida de Desenvolvimento de Software:

- A liberação, o provisionamento e a implantação podem ser feitos usando:
- Git + Jenkins + Terraform + Packer (imagens da Amazon)
- Git + Jenkins + Terraform + Orquestração Docker (imagens Docker)

Seção 4: Terraform com AWS:

4.1 : Nuvem Privada Virtual (VPC):

1. Na Amazon AWS, você tem uma VPC (Rede Virtual Privada) padrão criada para você por AWS lançará instâncias em
2. Até agora, usamos esta VPC padrão.
3. A VPC isola as instâncias em nível de rede.
4. É como sua própria rede privada na nuvem.
5. A melhor prática é sempre iniciar suas instâncias em uma VPC.
6. A VPC padrão
7. ou uma que você mesmo crie (gerenciada pelo Terraform)
8. Existe também o EC2-Classic, que é basicamente uma grande rede onde todos os clientes da AWS estão conectados. poderiam lançar suas instâncias em
9. Para configurações de pequeno a médio porte, uma VPC (por região) será adequada às suas necessidades.
10. Uma instância iniciada em uma VPC nunca poderá se comunicar com uma instância em outra VPC usando seus endereços IP privados.
11. Eles ainda podiam se comunicar, mas usando seu IP público (não recomendado)
12. Você também pode conectar duas VPCs, o que é chamado de peering.
13. Na Amazon AWS, você começa criando sua própria Rede Virtual Privada (VPN) para implantar suas instâncias (servidores) / bancos de dados em
14. Esta VPC utiliza o espaço de endereçamento 10.0.0.0/16, permitindo que você utilize os endereços IP que Comece com "10.0", assim: 10.0.xx
15. Esta VPC abrange a região eu-west-1, que é uma região da Amazon AWS na Irlanda.

Faixas de Endereços Privados (RFC 1918)

Classe/Faixa	Máscara CIDR	Início da Faixa	Fim da Faixa
Classe A	10.0.0.0/8	10.0.0.0	10.255.255.255
Classe B	172.16.0.0/12	172.16.0.0	172.31.255.255
Classe C	192.168.0.0/16	192.168.0.0	192.168.255.255

Exemplos de Máscaras e Endereços de Sub-redes

Máscara CIDR	Máscara de Sub-rede	Total de Endereços	Exemplo de IP Válido
10.0.0.0/8	255.0.0.0	16.777.216	**10.0.0.1**
10.0.0.0/16	255.255.0.0	65.536	**10.0.5.1**
10.1.0.0/16	255.255.0.0	65.536	**10.1.5.1**
10.0.0.0/24	255.255.255.0	256	**10.0.0.1**
10.0.1.0/24	255.255.255.0	256	**10.0.1.5**
10.0.0.5/32	255.255.255.255	1	**10.0.0.5**

1. Cada zona de disponibilidade possui sua própria sub-rede pública e privada.
2. As instâncias iniciadas na sub-rede maon-public-3 terão o endereço IP 10.0.3.x e serão Lançado na zona de disponibilidade eu-west-1c (a AWS considera 1 datacenter como uma zona de disponibilidade).
3. Uma instância iniciada em main-private-1 terá um endereço IP 10.0.4.x e residirá na Zona de Disponibilidade (AZ) eu-west-1a da AWS.
4. Todas as sub-redes públicas estão conectadas a um gateway da Internet. Essas instâncias também terão um endereço IP público, permitindo que sejam acessíveis pela internet.
5. As instâncias iniciadas em sub-redes privadas não recebem um endereço IP público, portanto, não serão acessíveis pela internet.
6. Instâncias do servidor público principal podem acessar as instâncias do servidor privado principal, pois todas estão na mesma VPC. Isso, é claro, se você configurar o firewall para permitir o tráfego de uma para a outra.
7. Normalmente, você usa as sub-redes públicas para serviços/aplicativos voltados para a internet.
8. Bancos de dados, serviços de cache e back-ends ficam todos em sub-redes privadas.
9. Se você usar um balanceador de carga (LB), normalmente colocará o LB nas sub-redes públicas e as instâncias que servem um aplicativo nas sub-redes privadas.

4.2: Iniciando uma instância EC2 em uma VPC:

1. Criar uma instância EC2 é muito simples.
2. Agora queremos iniciar a instância em nossa VPC recém-criada.
    - Com grupos de segurança
    - Usando um par de chaves que será carregado pelo Terraform.
3. Precisamos de um novo grupo de segurança para esta instância EC2.
4. Um grupo de segurança é como um firewall, gerenciado pela AWS.
5. Você especifica as regras de tráfego de entrada (entrada) e de saída (saída).
6. Se você deseja acessar apenas o SSH (porta 22), pode criar um grupo de segurança que:
    - Permite a porta de entrada 22 no intervalo de endereços IP 0.0.0.0/0 (todos os IPs).
    - É uma boa prática permitir apenas o IP do seu trabalho/casa/escritório.
    - Permite todo o tráfego de saída da instância para 0.0.0.0/0 (todos os IPs).
7. Para poder fazer login, o último passo é garantir que a AWS instale nosso par de chaves públicas no servidor. exemplo
8. Nossa instância EC2 já faz referência a um Keypairs; você só precisa declará-lo no Terraform:

```
# keypairs.tf
```

```
# Declaração do par de chaves SSH para acesso às instâncias EC2

resource "aws_key_pair" "mykeypair" {
  key_name   = "mykeypair"
  public_key = file("${path.module}/keys/mykeypair.pub")
}

# Uso em uma instância EC2 (exemplo)
resource "aws_instance" "example" {
  ami           = "ami-0c55b159cbfafe1f0"
  instance_type = "t2.micro"

# Referência ao key pair criado acima
  key_name = aws_key_pair.mykeypair.key_name

  tags = {
    Name = "MyInstance"
  }
}
```

9. O arquivo Keys/mykepair.pub será carregado na AWS e permitirá a criação de uma instância. lançado com esta chave pública instalada.
10. Você nunca deve enviar sua chave privada! Você usa sua chave privada para fazer login na instância.

4.3: EBS (Elastic Block Storage):

1. A instância t2.micro com esta AMI específica adiciona automaticamente 8 GB de armazenamento EBS.
2. Alguns tipos de instância possuem armazenamento local na própria instância.
   - Isso é chamado de armazenamento efêmero (ephemeral storage).
   - Esse tipo de armazenamento é sempre perdido quando a instância é encerrada.
3. O armazenamento de volume raiz EBS de 8 GB que acompanha a instância também está configurado para ser removido automaticamente quando a instância é encerrada.
4. Você ainda poderia instruir a AWS a não fazer isso, mas isso seria contrário à lei do intuitivo (antipadrão)
5. Na maioria dos casos, 8 GB para o sistema operacional (dispositivos de bloco raiz) são suficientes.
6. No próximo exemplo, estou adicionando um volume de armazenamento EBS extra.
   - Volumes extras podem ser usados para arquivos de log e quaisquer dados reais que sejam colocados no sistema.
   - Esses dados serão mantidos até que você instrua a AWS a removê-los.
7. O armazenamento EBS pode ser adicionado usando um recurso do Terraform e, em seguida, anexado à nossa instância.
8. No exemplo anterior, adicionamos um volume extra. 8. O volume raiz de 8 GB ainda existe
9. Se você quiser aumentar o armazenamento ou o tipo do volume raiz, você pode usar root_block_device dentro do recurso aws_instance.

4.4 Dados do Usuário:

1. Os dados do usuário na AWS podem ser usados para realizar qualquer personalização no momento do lançamento:
   - ◆ Instalar software adicional
   - ◆ Preparar a instância para ingressar em um cluster
     - ➔ Exemplo: cluster Consul
     - ➔ Exemplo: cluster ECS

- Executar comandos/scripts
- Montar volumes

2. Os dados do usuário são executados apenas na criação da instância, não quando a instância é reiniciada.
3. O Terraform permite adicionar dados de usuário ao recurso aws_instance.
    - Assim como uma string.
    - Usando modelos .
4. Instalará um servidor de aplicativos OpenVPN na inicialização.

4.5: IPs estáticos, EIPs e Route53:

1. Endereços IP privados serão atribuídos automaticamente às instâncias EC2.
2. Cada sub-rede dentro do VPS tem seu próprio intervalo (ex: 10.0.1.0 - 10.0.1.255)
3. Ao especificar o IP privado, você garante que a instância EC2 sempre use o mesmo Endereço IP.
4. Para usar um endereço IP público, você pode usar EIPs (endereços IP elásticos).
5. Este é um endereço IP público estático que você pode associar à sua instância.
6. Normalmente, você não usará endereços IP, mas sim nomes de host.
7. É aqui que entra no **AWS Route53**.
8. Você pode hospedar um nome de domínio na AWS usando o Route53. 9.

Primeiro, você precisa registrar um nome de domínio usando a AWS ou qualquer registrador credenciado.

10. Em seguida, você pode criar uma zona no Route53 (por exemplo, example.com) e adicionar registros DNS (por exemplo, servidor1.exemplo.com)
11. Adicionar uma zona e registros pode ser feito no Terraform.
12. O Route53 possui muitos servidores de nomes. Para saber quais são os servidores de nomes do seu domínio específico, você pode usar o recurso de saída para exibir a propriedade aws_route53_zone.example-com.name_servers.

4.6: RDS (Serviços de Banco de Dados Relacionais):

1. RDS significa Serviços de Banco de Dados Relacional.
2. É uma solução de banco de dados gerenciada:
    - Você pode configurar facilmente a replicação (alta disponibilidade) .
    - Snapshots automatizados (para backups) .
    - Atualizações de segurança automatizadas .
    - Substituição fácil de instâncias (para escalonamento vertical).
3. Os bancos de dados suportados são:
    - MySQL
    - MariaDB
    - PostgreSQL
    - Microsoft SQL Server
    - Oracle
4. Etapas para criar uma instância RDS:
    - Criar um **grupo de sub-redes**

    Permite especificar em quais sub-redes o banco de dados estará localizado.

- Criar um **Grupo de Parâmetros**

 Permite especificar parâmetros para alterar as configurações no banco de dados.

- Crie um **grupo de segurança** que permita o tráfego de entrada para a instância RDS.
- Crie uma **instância RDS .**

4.7 : IAM (Gestão de Identidade e Acesso):

1. IAM na AWS significa Gerenciamento de Identidade e Acesso.
2. É um serviço que ajuda você a controlar o acesso aos seus recursos da AWS.
3. Na AWS, você pode criar:
   - Grupos
   - Usuários
   - Funções
4. Os usuários podem formar grupos.
   - Por exemplo: um grupo de "Administradores" pode conceder privilégios de administrador a Usuários.
4. Os usuários podem se autenticar.
   - Utilizando um login/senha
   - Opcionalmente, usando um token: MFA / Google Authenticator
   - Uma chave de acesso e uma chave secreta (as chaves da API)
5. As funções podem conceder aos usuários/serviços acesso (temporário) que eles normalmente não teriam.
6. As funções podem, por exemplo, ser associadas a instâncias EC2.
7. Um exemplo:
   - A partir dessa instância, um usuário ou serviço pode obter credenciais de acesso.
   - Usando essas credenciais de acesso, o usuário ou serviço pode assumir a função, o que lhes dá permissão para fazer algo.
   - Você cria uma função chamada mybucket-access e a atribui a uma instância do EC2 durante a inicialização.
   - Você concede à função as permissões para ler e gravar itens em "mybucket"
   - Ao fazer login, você agora pode assumir essa função mybucket-access sem usar suas próprias credenciais
   - Você receberá credenciais de acesso temporárias que se parecem com credenciais de usuário normais.
     - Agora você pode ler e escrever itens em "mybucket".
8. Em vez de um usuário usar o aws-cli, um serviço também assume uma função.
9. O serviço precisa implementar o SDK da AWS.
10. Ao tentar acessar o bucket S3, será feita uma chamada de API para a AWS.
11. Se as funções estiverem configuradas para esta instância EC2, a API da AWS concederá acesso temporário. Chaves que podem ser usadas para assumir essa função.
12. Depois disso, o SDK pode ser usado da mesma forma que você usaria com credenciais normais.
13. Isso acontece mesmo em segundo plano, e você não vê muito disso.
14. As funções do IAM funcionam apenas em instâncias EC2 e não, por exemplo, fora da AWS.
15. As credenciais de acesso temporário também precisam ser renovadas, pois são válidas apenas por um período determinado.

    período de tempo predefinido.
    - Isso também é algo que o SDK da AWS cuidará.

16. Para criar um grupo de administradores do IAM na AWS, você pode criar o grupo e anexar a ele a política de administrador gerenciada pela AWS.
17. Você também pode criar sua própria política personalizada.

4.8: Autoscaling:

1. Na AWS, grupos de escalonamento automático podem ser criados para adicionar/remover instâncias automaticamente quando necessário.

   Certos limites são atingidos.

➢ Por exemplo, a sua camada de aplicação pode ser escalada horizontalmente quando tiver mais visitantes.

2. Para configurar o escalonamento automático na AWS, você precisa configurar pelo menos 2 recursos:

   Uma configuração de lançamento da AWS

   Um grupo de escalonamento automático.

3. Depois de configurar o grupo de escalonamento automático, você pode criar políticas de escalonamento automático.
4. Uma política é acionada com base em um limite (Alarme do CloudWatch) ;
5. Um ajuste será executado.
6. Primeiro, é necessário criar a configuração de inicialização e o grupo de escalonamento automático.
7. Para criar uma política, você precisa de uma aws_autoscaling_policy.
8. Em seguida, você pode criar um alarme do CloudWatch que acionará a política de escalonamento automático.
9. Se você deseja receber um alerta (por exemplo, por e-mail) quando o dimensionamento automático for acionado, é necessário criar um tópico do SNS (Simple Notification Service).
10. Esse tópico do SNS precisa ser anexado ao grupo de dimensionamento automático.

4.9: ELB (Balanceadores de Carga Elásticos)

1. Agora que você tem instâncias com escalonamento automático, talvez queira colocar um balanceador de carga na frente do servidor.
2. O AWS Elastic Load Balancer (ELB) distribui automaticamente o tráfego de entrada entre os servidores. várias instâncias EC2.
    - O próprio ELB escala quando você recebe mais tráfego.
    - O ELB verifica a integridade das suas instâncias.
    - Se uma instância falhar na verificação de integridade, nenhum tráfego será enviado para ela.
    - Se uma nova instância for adicionada pelo grupo de escalonamento automático, o ELB irá adicionar automaticamente as novas instâncias e iniciará a verificação de integridade.
3. O ELB também pode ser usado como terminador SSL.
    - É possível descarregar a criptografia das instâncias EC2.
    - A AWS pode até gerenciar os certificados SSL para você.
4. Os ELBs podem ser distribuídos por várias Zonas de Disponibilidade para maior tolerância a falhas.
5. Em geral, você obterá níveis mais altos de tolerância a falhas com um roteamento ELB. tráfego para sua aplicação.
6. O ELB é comparável a um nginx/haproxy, mas é fornecido como um serviço.

A AWS oferece 2 tipos diferentes de balanceadores de carga:

Balanceador de Carga Clássico (ELB):

- Direciona o tráfego com base em informações de rede.
- Exemplo: Encaminha todo o tráfego da porta 80 (HTTP) para a porta 8080 (aplicação).

O balanceador de carga de aplicativos (ALB):

- Direciona o tráfego com base em informações do nível da aplicação.
- Exemplo, pode direcionar /api e /website para instâncias EC2 diferentes.

4.10: Elastic Beanstalk:

1. O Elastic Beanstalk é a solução de Plataforma como Serviço (PaaS) da AWS.
2. É uma plataforma onde você lança seu aplicativo sem precisar se preocupar com a manutenção da infraestrutura subjacente.
3. Você ainda é responsável pelas instâncias EC2, mas a AWS fornece as atualizações que você pode instalar.
4. As atualizações podem ser aplicadas manualmente ou automaticamente.
5. As instâncias EC2 executam o Amazon Linux.
6. O Elastic Beanstalk pode lidar com o escalonamento da aplicação para você.
7. Subjacente a isso, utilize um balanceador de carga e um grupo de escalonamento automático para atingir esse objetivo.
8. Você pode agendar eventos de escalonamento ou ativar o escalonamento automático com base em uma métrica.
9. É semelhante ao Heroku (outra solução PaaS)
10. É semelhante ao Heroku (outra solução PaaS)
11. Você pode ter um aplicativo em execução com apenas alguns cliques usando o console da AWS.
12. Ou utilizando os recursos do Elastic Beanstalk no Terraform.
13. As plataformas suportadas são:
    - ➢ PHP
    - ➢ Java SE, Java com Tomcat
    - ➢ .NET no Windows com IIS
    - ➢ Node.js
    - ➢ Python
    - ➢ Ruby
    - ➢ Go
    - ➢ Docker

1. Ao implantar um ambiente Elastic Beanstalk, você receberá um CNAME (nome do host) que você pode usar como endpoint;
2. Você pode usar o Route53 para apontar seu domínio para esse CNAME;
3. Uma vez que Elastic Beanstalk estiver em execução, você poderá implantar seu aplicativo nele usando o EB;
4. O utilitário de linha de comando. http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/eb-cli3.html