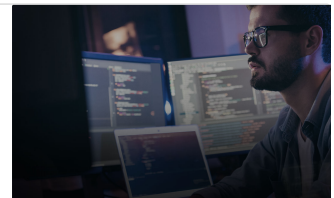


Terraform

Criando infraestrutura como código utilizando Terragrunt e Atlantis na AWS - Blog da Compass

Este artigo será dividido em duas partes, no primeiro artigo vamos falar mais sobre o Terragrunt e Atlantis, sobre como estas ferramentas podem auxiliar a adoção de uma cultura DevOps na sua empresa, vamos mostrar os passos necessários para a sua instalação e integração com um repositório do Github, e na

<https://blog.compass.uol/autores/criando-infraestrutura-como-codigo-utilizando-terragrunt-e-atlantis-na-aws/>



▼ **TERRAFORM - Alura**

- <https://www.terraform.io/>
- Ferramenta multiplataforma
- Serve para administrar a infraestrutura cloud, tanto para azure, aws, gcp, etc.
- Irá provisionar o provedor de cloud

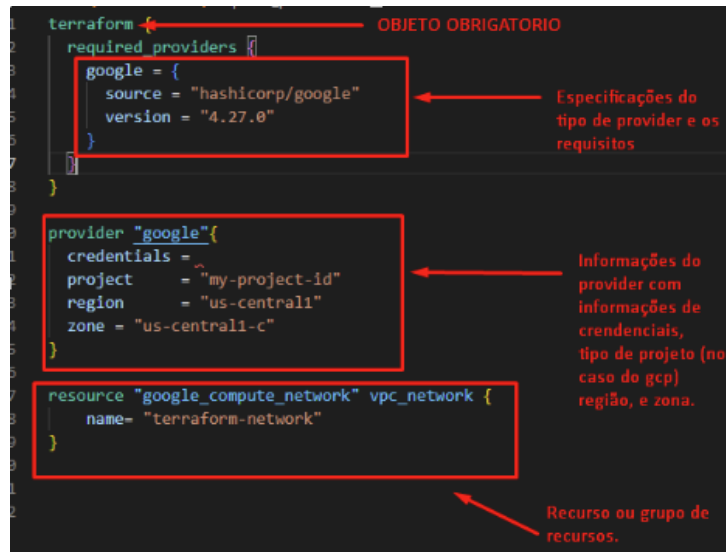
INSTALAÇÃO

- LINUX-UBUNTU
 - Rodar os seguintes comandos :
 - `curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -`
 - `sudo apt-add-repository "deb [arch=amd64] https://apt.releases.hashicorp.com $(lsb_release -cs) main"`
 - `sudo apt-get update && sudo apt-get install terraform`
 - Se necessario instalar "curl"
- WINDOWS
 - Baixar o arquivo em zip
 - Criar um pasta no diretorio C: chamada terraform

- Crie a variável de ambiente chamada Path – C:\terraform

CONFIGURANDO A INFRAESTRUTURA

- Os arquivos de configuração podem ter quaisquer nomes, desde que estejam no diretório correto e possuam a extensão .tf.



- Para buscas de templates google_projects
- TAGS – servem para dar nome ao recurso ou grupo de recursos
- Podemos criar vários recursos de uma vez usando um comando
 - No caso "count"
- "apply" acionado ao criar os recursos
- **Na primeira vez utilizamos o init, depois o plan na sequência (mostra todas as alterações, mas sem aplicá-las) e apply para efetivar as mudanças.**
- Mesmo não sendo recomendável, o Terraform permite que você utilize diretamente o apply.
- O comando show lê o arquivo terraform.tfstate e exibe as informações associadas (IP, rede, etc).

```

ami = "ami-052efd3df9dad4825"
instance_type = "t2.micro"
key_name = "terraform-aws"

tags = {
  Name = "dev${count.index}"
}
vpc_security_group_ids = [ "sg-00f7d58cab2f5f331" ]

#Codigo para configuração do VPC (Virtual private cloud).
#Para que possamos acessar as instancias criadas.
resource "aws_security_group" "acesso-ssh" {
  name = "acesso-ssh"
  description = "acesso-ssh"

  ingress {
    from_port = 22
    to_port = 22
    protocol = "SSH"
    cidr_blocks = ["201.49.103.129/32"]
  }

  tags = {
    Name = "ssh"
  }
}

```

Colocamos o id do VPC criado

Fazemos uma referência do VPC, através deste comando, para que possa ser atraído as instancias criadas

Criamos o recurso security group, colocamos um nome+descrição, adicionamos a porta protocolo, e o IP ao qual terá acesso ao VPC

```

1 terraform {
2   required_providers {
3     google = {
4       source = "hashicorp/google"
5       version = "4.27.0"
6     }
7   }
8 }
9
10 provider "google" {
11   credentials = "~/.google/credentials.json"
12   project = "my-project-id"
13   region = "us-central1"
14   zone = "us-central1-c"
15 }
16
17 resource "google_compute_network" "vpc_network" {
18   name = "terraform-network"
19 }
20
21
22

```

OBJETO OBRIGATORIO

Especificações do tipo de provider e os requisitos

Informações do provider com informações de credenciais, tipo de projeto (no caso do gcp) região, e zona.

Recurso ou grupo de recursos.

```

instance_type = t2.micro
key_name = "terraform-aws"

tags = {
  Name = "dev${count.index}"
}
vpc_security_group_ids = [ "${aws_security_group.acesso-ssh.id}" ]
}

#Codigo para configuração do VPC (Virtual private cloud).
#Para que possamos acessar as instancias criadas.
resource "aws_security_group" "acesso-ssh" {
  name = "acesso-ssh"
  description = "acesso-ssh"

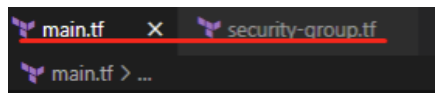
  ingress {
    from_port = 22
    to_port = 22
    protocol = "tcp"
    cidr_blocks = ["201.49.103.129/32"]
  }

  tags = {
    Name = "ssh"
  }
}

```

Para que não precisemos escrever o id completo podemos fazer uma referência ao resource utilizando \$

- Podemos separar os arquivos dentro do terraform, contanto que os arquivos estejam na mesma pasta e com o .tf



- Podemos deixar as instancias principais como VMs Ec2, Buckets S3, DataBase colocamos no main
- Security groups, e variaveis colocamos separados
- A configuração pode ser "quebrada" em mais de um arquivo, desde que esteja na mesma estrutura de diretório.
- O Terraform permite que você concentre suas configurações em um único arquivo.

```

resource "aws_instance" "dev4" {
  ami = "ami-052efd3df9dad4825"
  instance_type = "t2.micro"
  key_name = "terraform-aws"

  tags = {
    Name = "dev4"
  }
  vpc_security_group_ids = [ "${aws_security_group.acesso-ssh.id}" ]
  depends_on = [ aws_s3_bucket.dev4 ]
}

resource "aws_instance" "dev5" {
  ami = "ami-052efd3df9dad4825"
  instance_type = "t2.micro"
  key_name = "terraform-aws"

  tags = {
    Name = "dev5"
  }
  vpc_security_group_ids = [ "${aws_security_group.acesso-ssh.id}" ]
}

resource "aws_s3_bucket" "dev4" {
  bucket = "lbrs-dev4"

  tags = {
    Name = "lbrs-dev4"
  }
}

resource "aws_s3_bucket_acl" "dev4" {
  bucket = aws_s3_bucket.dev4.id
  acl = "private"
}

```

Criamos um bucket S3 e criamos uma VM (dev4) para podermos deixar o bucket como dependente do recurso

- Para um mesmo provedor, utilizamos o **alias** para referenciar os recursos de outra região.

```

# Configure the AWS Provider
provider "aws" {
  region = "us-east-1"
}

provider "aws" {
  alias = "us-east-2"
  region = "us-east-2"
}

```

- E quando criarmos instancias/recursos de outras regiões, precisamos adicionar o provider, e em caso de security groups precisamos criar um novo para aquela região

```

resource "aws_instance" "dev6" {
  provider = aws.us-east-2
  ami = "ami-02d1e544b84bf7502"
  instance_type = "t2.micro"
  key_name = "terraform-aws"

  tags = {
    Name = "dev6"
  }
  vpc_security_group_ids = [ "${aws_security_group.acesso-ssh-us-east-2.id}" ]
}

```

```

resource "aws_security_group" "acesso-ssh" {
  name      = "acesso-ssh"
  description = "acesso-ssh"

  ingress {
    from_port      = 22
    to_port        = 22
    protocol       = "tcp"
    cidr_blocks    = ["201.49.103.129/32"]
  }

  tags = {
    Name = "ssh"
  }
}

resource "aws_security_group" "acesso-ssh-us-east-2" {
  provider = aws.us-east-2
  name      = "acesso-ssh"
  description = "acesso-ssh"

  ingress {
    from_port      = 22
    to_port        = 22
    protocol       = "tcp"
    cidr_blocks    = ["201.49.103.129/32"]
  }

  tags = {
    Name = "ssh"
  }
}

```

- Utilizamos a tag provider, seguida do alias que desejamos utilizar.
- Com as variáveis podemos puxar um arquivo através do deixar o código mais limpo

```

vars.tf > ...
1  variable "amis" {
2    type = map
3
4    default = {
5      "us-east-1" = "ami-052efd3df9dad4825"
6      "us-east-2" = "ami-02d1e544b84bf7502"
7    }
8  }
9

```

Criamos uma variável chamada de "amis" do tipo "map". Acrescentamos as amis de cada região + um apelido que será usado para o código ser puxado.

```
#us-east-1, criação de 3 vms dev
resource "aws_instance" "dev" {
  count = 3
  ami = var.amis["us-east-1"]
  instance_type = "t2.micro"
  key_name = "terraform-aws"

  tags = {
    Name = "dev${count.index}"
  }
  vpc_security_group_ids = [ "${aws_security_group.acesso-ssh.id}" ]
}
```

No código ficará da seguinte forma

- Para a remoção dos recursos podemos retirar no próprio texto do código e dar um **terraform plan** e **apply** ou então podemos colocar **terraform destroy -target RECURSO**
- Para remover os recursos co-dependentes remove o recurso dependente para que seja excluído os dois ao mesmo tempo
- Caso formos remover tudo utilizamos o comando **terraform destroy**
- Tanto removendo as configurações dos recursos no arquivo de configuração, como também utilizando o comando **terraform destroy**.
- Os outputs trazem uma informação devolta, como no caso

```
outputs.tf ▸ ...
0 references
1 output "ips" {
2   value = "${aws_instance.dev5.public_ip}"
3 }
4

PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL 1: bash

aws_security_group.acesso-ssh-us-east-2: Refreshing state... [id=sg-0ce9bfe555]
aws_dynamodb_table.dynamodb-homologacao: Refreshing state... [id=GameScores]
aws_instance.dev5: Refreshing state... [id=i-022784930c11f25c4]
aws_instance.dev[0]: Refreshing state... [id=i-0e16fadble3471a2b]
aws_instance.dev[2]: Refreshing state... [id=i-0ad9e0edabaf7267a]
aws_instance.dev[1]: Refreshing state... [id=i-0ea88ec9924c3941f]
aws_instance.dev7: Refreshing state... [id=i-0de9704b5c95d1f61]
aws_instance.dev6: Refreshing state... [id=i-0b2b8032650172592]

Outputs:
ips = 3.92.196.246
HAL:terraform rmo$
```

- São declaradas com a sintaxe:
- `output "nome" { value = "${referência}" }`
- a principal vantagem de utilizar o backend remoto, o que permite inclusive a administração da infraestrutura por mais de uma pessoa.

- É necessário criar alguns arquivos adicionais para utilizar o backend remoto.