DigitalHouse>

Certified Tech Developer
The Ultimate Degree

Infraestrutura II

# Terraform HCL - Hashicorp Configuration Language

Bem-vindo! Neste espaço vamos aprender com mais detalhes sobre a linguagem HCL no Terraform.

Vamos lembrar que o principal objetivo do HCL no Terraform é declarar recursos. Isso é extremamente importante, pois será a essência do nosso código. Usando essa linguagem, vamos dizer ao Terraform como gerenciar uma coleção específica de infraestrutura. Vamos lá!

## Semântica e estrutura

A sintaxe e estrutura da linguagem Terraform HCL consiste em alguns elementos básicos:

```
resource "aws_vpc" "mi_vpc" {

 cidr_block = 10.0.0.0 /16

}
```

Analisando a sintaxe geral, podemos descobrir duas sub linguagens integradas:

- A linguagem estrutural, que define a estrutura de configuração hierárquica geral e é uma serialização de corpos, blocos e atributos HCL.
- A linguagem de expressão, usada para expressar valores de atributos, como literais ou como derivações de outros valores.

Em linhas gerais, essas sub linguagens são usadas junto com os arquivos de configuração para descrever uma configuração geral, com a linguagem estrutural.

## Tipo de bloco

Vamos pegar o snippet anterior e dar uma olhada mais de perto em cada um de seus componentes:

```
<BLOCK TYPE> "<PROVIDER_ELEMENT>" "<BLOCK LABEL>" {
```

A primeira coisa que precisamos informar é o tipo de bloco que usaremos. Os tipos de blocos mais comuns usados são: RECURSOS, VARIÁVEIS e DADOS, embora existam outros também.

Esses blocos "declaram" qual recurso de nosso provedor de nuvem vamos usar. Por exemplo, digitando:

```
resource "aws_vpc" "my_vpc"
```

... estamos dizendo que:

1. Precisamos trabalhar com um bloco do tipo “recurso”, “RESOURCE”.
2. Que o "recurso" a ser instanciado será "aws_vpc", "PROVIDER_ELEMENT". [1]
3. E queremos chamá-lo de "my_vpc", "BLOCK_LABEL"

# Argumentos

A seguir encontramos os argumentos que atribuem um valor a um nome e aparecem dentro de blocos:

```
 # Tipo de bloco

 <IDENTIFIER> = <EXPRESSION> # Argumento / s

}
```

É o "conteúdo" do "block body". São os valores e / ou funções que são lidos durante o tempo de execução do código e é onde atribuímos os valores que queremos para os nossos elementos de infraestrutura.

Esses argumentos podem ser divididos em "Identificador" e "Expressão". Ambos são estruturados como uma chave, um valor e são separados por um sinal de igual.

A chave representa um identificador e o valor é o que esse identificador armazena.

Um exemplo simples e claro é a região dentro do provedor:

```
region = "Us-east-1"
```

Referindo-se ao nosso código inicial:

```
cidr_block = 10.0.0.0 /16
```

... estamos dizendo que meus argumentos serão:

1. Uma chave chamada "cidr_block" [2]
2. Que eu quero que a sub-rede base do VPC que estou construindo seja 10.0.0.0/16.

É importante observar que os argumentos que o Terraform aceita não são arbitrários, mas estão programados para receber certas palavras específicas. [2]

Existem também argumentos que são obrigatórios, ou seja, devem existir em nosso código e outros não. Por exemplo, "cidr_block" é um argumento obrigatório porque nosso VPC exige que informemos em qual rede iremos criá-lo.

## Variáveis

HCL usa variáveis de entrada que servem como parâmetros para um módulo Terraform.

Declaramos uma variável da seguinte maneira:

```
variable "image_id" {

 type = string

}
```

A palavra "variável" é um bloco que então suporta apenas um BLOCK_LABEL.

No nosso caso, “image_id” e que deve ser "único" entre todas as variáveis já definidas.

O nome de uma variável pode ser qualquer identificador válido, exceto o seguinte: fonte, versão, provedores, contagem, for_each, ciclo de vida, depends_on, locais que são os chamados meta-argumentos, que não serão abordados neste curso, mas recomendamos navegá-los.

A lista completa pode ser vista em Resources Overview - Configuration Language.
Dentro de sua estrutura, também encontraremos argumentos.

Agora, o que estamos dizendo quando introduzimos a palavra "tipo"? Basicamente, esse conteúdo será do tipo "string", pois pode haver casos em que, em vez de dados do tipo string, podemos ter: número, bool, lista, mapa, tupla, etc. A lista completa pode ser vista no seguinte link: Input Variables - Configuration Language

## Variáveis: um caso prático

Suponha que não queremos que o valor líquido de nossa sub-rede apareça “codificado”. O que podemos fazer é criar uma variável com os próprios dados e, a partir desse código, referir-se a essa variável.

Em nosso exemplo inicial:

```
recurso "aws_vpc" "my_vpc" {

 cidr_block = 10.0.0.0 /16

}
```

... poderíamos substituí-lo, por exemplo, pelos seguintes argumentos:

```
resource "aws_vpc" "my_vpc" {

cidr_block = var.base_cidr_block

}
```

Para fazer isso, basta declarar a variável da seguinte maneira:

```
variable "base_cidr_block" {

 default = 10.0.0.0 /16

}
```

Desta forma, estamos personalizando aspectos do módulo sem alterar o código-fonte do próprio módulo.

Outra forma mais prática é definir um arquivo com o nome: variables.tf onde declaramos uma ou mais variáveis, conseguindo modularizar nosso código.

Nota: é importante que este arquivo de variáveis esteja localizado no mesmo diretório que nosso módulo principal.

# sintaxe

## Palavras reservadas

A maioria das linguagens de programação contém palavras que não podem ser usadas como variáveis de atribuição. Essas palavras são chamadas de "reservadas" porque são usadas pelo idioma para funções específicas. No caso da HCL, o Terraform não possui palavras reservadas globais.

## Comentários

Existem três sintaxes diferentes para comentários:

1. # começa um comentário de uma linha, terminando no final da linha.
2. // também inicia um comentário de uma única linha, como uma alternativa para #.
3. / * e * / são os delimitadores de início e fim de um comentário que pode abranger várias linhas.

O estilo de comentário de linha única # é o estilo de comentário padrão e deve ser usado na maioria dos casos.

# Concluindo

Não é necessário conhecer todos os detalhes da sintaxe HCL para usar o Terraform. Na verdade, apenas por entender sua estrutura e o que você deseja alcançar, já estamos em condições de criar módulos mais complexos. O resto é consultar a documentação oficial para descobrir que tipo de recursos você precisa, quais argumentos você tem e um pouco de imaginação!

**[1]**: Podemos encontrar a lista completa de recursos na Docs overview |hashicorp/aws, embora sempre navegando no site terraform.io diretamente, pois a equipe Terraform atualiza constantemente seus recursos online.

**[2]**: Neste link você pode encontrar as referências aos argumentos que o Terraform espera: aws_vpc | Recursos hashicorp / aws