

Curso Prático de Terraform - Seção 1: Introdução

🚀 Objetivo da Seção

A primeira seção tem como foco apresentar os fundamentos do Terraform, seus conceitos principais e preparar o aluno para entender o ciclo de vida da infraestrutura como código (IaC). É uma base essencial para as próximas seções.

🔍 O que é Terraform?

Terraform é uma ferramenta de **Infrastructure as Code (IaC)** desenvolvida pela HashiCorp. Com ele, você consegue descrever toda a infraestrutura necessária para uma aplicação em arquivos de texto (com extensão `.tf`) e versioná-la usando ferramentas como o Git.

Vantagens:

- Reprodutibilidade: mesma infraestrutura em diferentes ambientes.
- Versionamento: track de mudanças no Git.
- Automatização de provisionamento.
- Idempotente: aplicações repetidas produzem o mesmo resultado.

⚡ Provedores

Um **provider** é o plugin que permite o Terraform interagir com a API de um serviço. Ex: ``aws``, ``azurerm``, ``google``, ``kubernetes``, ``datadog``, ``vault``, ``cloudflare``...

Exemplo de uso:

```
```.hcl
terraform {
 required_providers {
 aws = {
 source = "hashicorp/aws"
 version = "~> 5.0"
 }
 }
}
...

```

## ## 🎓 Blocos Principais

### ### 1. `terraform {}`

Define as configurações globais do projeto: provedores, backend, etc.

### ### 2. `provider {}`

Configura os detalhes de acesso ao provedor (ex: credenciais da AWS).

### ### 3. `resource {}`

Elemento que representa o que será criado (VM, bucket S3, banco de dados, etc).

### 4. `variable {}`

Define variáveis reutilizáveis no código.

### 5. `output {}`

Exibe resultados da execução, como URLs, IPs, ARNs etc.

## 🔄 Ciclo de Vida do Terraform

### 1. `terraform init`

Inicializa o diretório, instala os provedores.

### 2. `terraform plan`

Mostra o que vai ser criado/modificado/removido. Ideal para revisão.

### 3. `terraform apply`

Aplica as mudanças reais na infraestrutura.

### 4. `terraform destroy`

Remove toda a infraestrutura criada pelo código.

## 📋 Comandos adicionais do Terraform

| Comando | Função |

|-----|-----|

| `terraform validate` | Valida a sintaxe dos arquivos Terraform |

| `terraform fmt` | Formata os arquivos `.tf` conforme o padrão oficial |

| `terraform show` | Mostra os recursos aplicados atualmente |

| `terraform output` | Exibe os valores definidos no bloco `output {}` |

| `terraform refresh` | Atualiza o estado atual consultando o provedor |

| `terraform taint` | Marca um recurso para ser recriado no próximo apply |

| `terraform state list` | Lista todos os recursos gerenciados no state file |

| `terraform state show <recurso>` | Detalha atributos do recurso |

| `terraform import` | Importa um recurso externo para o controle do Terraform |

### ## 📖 Explicação do Exemplo Prático

```
```:hcl
```

```
terraform {
```

```
  required_providers {
```

```
    null = {
```

```
      source = "hashicorp/null"
```

```
      version = "~> 3.0"
```

```
    }
```

```
  }
```

```
}
```

```
provider "null" {}
```

```
resource "null_resource" "exemplo" {  
  provisioner "local-exec" {  
    command = "echo 'Hello Terraform! Teste concluido com sucesso'"  
  }  
}  
...
```

Análise do bloco

- ``terraform {}``: Define o uso do provider ``null``, que é um plugin da HashiCorp que não provisiona recursos reais.
- ``provider "null" {}``: Configura esse provider, que não exige credenciais nem acesso à nuvem.
- ``resource "null_resource" "exemplo"``: Cria um recurso vazio com nome ``exemplo``. Esse tipo de recurso é ótimo para testes ou lógica condicional.
- ``provisioner "local-exec"``: Executa um comando local. No caso, imprime uma mensagem com ``echo``.

Quando usar `null_resource`:

- Para testes locais.
- Para executar scripts shell ou scripts de provisionamento.
- Para criar dependências artificiais entre recursos.
- Para simular aplicação de infraestrutura sem custo.

Exemplo Prático com ``null_resource``

Estrutura do Projeto:

```
```bash
```

```
mkdir terraform-intro && cd terraform-intro
```

```
```
```

```
### Arquivo: `main.tf`
```

```
```hcl
```

```
terraform {
```

```
 required_providers {
```

```
 null = {
```

```
 source = "hashicorp/null"
```

```
 version = "~> 3.0"
```

```
 }
```

```
 }
```

```
}
```

```
provider "null" {}
```

```
resource "null_resource" "exemplo" {
```

```
 provisioner "local-exec" {
```

```
 command = "echo 'Hello Terraform! Teste concluído com sucesso.'"
```

```
 }
```

```
}
```

```
```
```

```
### Comandos no terminal:
```

```
```bash
terraform init
terraform plan
terraform apply
terraform destroy
```
```

Esse exemplo não cria nada na nuvem, mas é ótimo para testar o ambiente.

Exercício Prático

1. Crie o diretório `terraform-intro`.
2. Salve o `main.tf` acima.
3. Execute os comandos em ordem: `init`, `plan`, `apply`.
4. Observe a saída do `echo` no terminal.
5. Por fim, execute `terraform destroy` para limpar.

Checklist de Aprendizado:

- [x] Entendi o conceito de IaC.
- [x] Sei o que é um provider e um resource.
- [x] Executei meu primeiro `terraform apply`.
- [x] Conheço o ciclo de vida: init -> plan -> apply -> destroy.
- [x] Compreendi o uso do `null_resource`.
- [x] Conheci comandos extras para diagnóstico, validação e formatação.

Quer que eu avance agora para a Seção 2: Tools and Setup com essa mesma abordagem detalhada?

(esqueci de mencionar, terminando, gere um pdf)