

# HashiCorp Certified Terraform Associate 003



The background features abstract, glowing purple geometric shapes, including a large cube on the left and various lines and arcs extending across the frame, set against a dark, textured purple background.

## 8. Read, generate, and modify configuration



The background features several glowing purple geometric shapes, including a large cube on the left and a triangular prism at the bottom, set against a dark, starry space-like background.

**Demonstrate use of  
variables and outputs**



# Benefícios em usar variáveis no Terraform

O uso eficiente de variáveis, variáveis locais e outputs no Terraform é essencial para modularizar e reutilizar código, facilitando a manutenção e escalabilidade de infraestruturas na nuvem.

Os dois benefícios principais em usar variáveis Terraform:

1. **Reutilização:** Ao empregar variáveis na configuração de sua infraestrutura como código (IaC), é possível evitar a codificação direta de valores, como tipos de instância ou nomes de região. Essa abordagem flexível permite a utilização dos mesmos arquivos de configuração em diversos ambientes (por exemplo, desenvolvimento, teste e produção) ou com diferentes configurações, simplesmente modificando os valores atribuídos às variáveis. Isso promove a reutilização dos arquivos de configuração e simplifica a adaptação para ambientes distintos.
2. **Segurança:** Para reduzir o risco de expor dados confidenciais, é recomendável evitar a codificação direta de valores específicos, como chaves de API e senhas, nas configurações. Em vez disso, é mais seguro tratá-los como variáveis, permitindo sua inserção dinâmica quando necessário. Isso protege informações sensíveis ao evitar sua exposição acidental ou não autorizada.

# Terraform Input and Output Variables

**Input variables:** Variáveis de entrada, criadas como blocos variáveis, podem ser usadas para parametrizar configurações do Terraform. Você pode usá-los para especificar valores para comandos do Terraform, como terraform plan ou terraform apply.

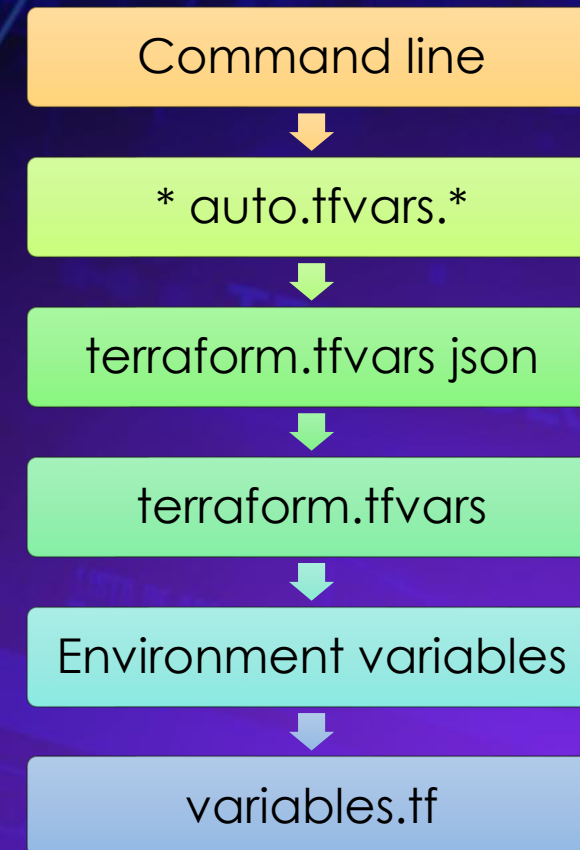
```
1 # Definindo uma variável de instância
2 variable "instance_type" {
3     description = "Tipo de instância da máquina virtual"
4     type        = string
5     default     = "Standard_DS2_v2"
6 }
7
8 # Definindo uma variável local
9 locals {
10     region = "East US"
11 }
```

**Output variables:** Usamos variáveis de saída para exportar valores de um arquivo de configuração do Terraform. Eles são definidos no bloco de saída na configuração do Terraform.

```
1 # Expondo o IP público da máquina virtual
2 output "public_ip" {
3     value = azurerm_virtual_machine.example.network_interface_ids[0]
4 }
```

# Diferentes maneiras de usar variáveis Terraform

1. **Interactive Input:** Apenas não forneça valor padrão. O Terraform pedirá valor no tempo de execução da criação dos recursos.
2. **Command-line variables:** Você pode passar variáveis de linha de comando no momento da ação de aplicação.
3. **Variable file .tfvars:** O Terraform procura por um terraform.tfvars. Se você criou vários arquivos com nomes diferentes. Você pode especificar o nome do arquivo.
4. **Environment variables:** Definir variável de ambiente com prefixo "TF\_VAR".





# Parâmetros de Input Variables no Terraform

As variáveis do Terraform possuem vários parâmetros que definem como elas se comportam, incluindo: **string, number, bool, list, map, set, object, tuple, and any.**

- **type:** Indica o tipo de dado que será armazenado na variável. Este parâmetro deve ser fornecido. Os tipos de variáveis válidos incluem string, número, bool, lista, etc.
- **default:** Define o valor padrão das variáveis. Se nenhum valor for especificado, esse valor padrão será aplicado quando os comandos do Terraform forem executados.
- **description:** Esta é uma visão geral da variável. Isso pode ser usado para explicar o propósito de uma variável ou seu uso pretendido.
- **validation:** você pode criar regras para validar variáveis. Informações sensíveis ou confidenciais (como senhas e chaves de acesso) podem ser encontradas nesta seção.
- **nullable** - Indica se uma variável pode ser nula dentro de um módulo.



# Describe secure secret injection best practice



TFTEC CLOUD



# Boas práticas para injeção segura de segredos

A gestão segura de segredos é crucial ao trabalhar com infraestrutura na nuvem.

Nesta sessão, vamos explorar as melhores práticas de injeção segura de segredos ao utilizar o Terraform, especialmente considerando o ambiente Azure, garantindo a proteção dos dados sensíveis e a conformidade com as políticas de segurança.



TFTEC CLOUD



# Utilização de variáveis sensíveis

Ao declarar variáveis sensíveis, o Terraform mantém esses valores ocultos nos outputs e no estado do Terraform.



```
1 variable "azure_client_secret" {  
2     type      = string  
3     sensitive = true  
4     description = "Azure client secret"  
5 }
```



# Integração com serviços de gerenciamento de Senhas

O Azure Key Vault oferece uma maneira segura de armazenar e recuperar segredos, garantindo a proteção dos dados sensíveis utilizados em suas configurações de infraestrutura no Terraform.

```
1  data "azurerm_key_vault_secret" "example" {
2    name          = "mySecret"
3    key_vault_id = azurerm_key_vault.example.id
4  }
5
6  resource "azurerm_virtual_machine" "example" {
7    # ...
8    admin_password = data.azurerm_key_vault_secret.example.value
9    # ...
10 }
```



The background features abstract, glowing purple geometric shapes, including a large cube-like structure on the left and various lines and arcs extending across the frame, set against a dark, textured purple background.

# Hands-On

*Terraform Secure  
Variables and Secrets*



# Understand the use of collection and structural types



TFTEC CLOUD



# Tipos de Input Variables no Terraform

Os tipos de variáveis de entrada do terraform são divididos em tipos de variáveis **primitivas** e **complexas**.

## Primitive Variables:

- String
- Number
- Bool

## Complex Variables:

- List
- Map
- Object
- Set
- Tuple



# String

- No código, estamos definindo uma variável chamada "region" usando o Terraform para trabalhar com recursos na Microsoft Azure.
- A linha **type** = string declara o tipo da variável como uma string, o que significa que essa variável pode conter qualquer valor de texto, como nomes de regiões ou outras strings.
- A linha **default** = "eastus" atribui um valor padrão à variável "region" caso nenhum valor seja especificado. Neste exemplo, o valor padrão é "eastus", que pode ser uma região geográfica na Azure.

```
1  # Exemplo de variável "region" como tipo string para Azure
2  variable "region" {
3    type    = string
4    default = "eastus"
5  }
```



# Number

- O exemplo define uma variável chamada "instance\_count" que é usada para representar um número, seja um número inteiro ou um valor de ponto flutuante (como 2.5, por exemplo) no Terraform para trabalhar com recursos na Microsoft Azure.
- A linha **type** = number declara o tipo da variável como um número, permitindo valores numéricos, como inteiros ou decimais.
- A linha **default** = 3 define o valor padrão da variável "instance\_count" como 3. Se nenhum valor for especificado, esse valor será usado como padrão.



```
1  # Exemplo de variável "instance_count" como tipo number para Azure
2  variable "instance_count" {
3      type    = number
4      default = 3
5  }
```



# Bool

- No exemplo acima, estamos definindo uma variável chamada "enable\_logging" no Terraform, para representar um valor booleano, ou seja, um valor que pode ser verdadeiro (true) ou falso (false).
- A linha **type** = bool declara o tipo da variável como booleano, permitindo apenas os valores true ou false.
- A linha **default** = true define o valor padrão da variável "enable\_logging" como true. Isso significa que se nenhum valor for especificado, o valor padrão será true, habilitando a função de log.
- Como mencionado no exemplo, você pode alterar o valor padrão de true para false para desabilitar a função de log.

```
1 # Exemplo de variável "enable_logging" como tipo bool para Azure
2 variable "enable_logging" {
3     type    = bool
4     default = true
5 }
```



# List

- O exemplo define uma variável chamada "instance\_types" para representar uma lista de elementos no Terraform para trabalhar com recursos na Microsoft Azure.
- A linha **type** = list(string) declara o tipo da variável como uma lista que contém elementos do tipo string. Neste exemplo, os elementos da lista são strings, representando diferentes tipos de instâncias na Azure.
- A linha **default** = ["Standard\_B1s", "Standard\_D2s\_v3", "Standard\_F2s\_v2"] define os valores padrão para a variável "instance\_types". Estes são exemplos de tipos de instância, e você pode adicionar ou modificar os valores da lista conforme necessário.



```
1 # Exemplo de variável "instance_types" como tipo list para Azure
2 variable "instance_types" {
3     type      = list(string)
4     default   = ["Standard_B1s", "Standard_D2s_v3", "Standard_F2s_v2"]
5 }
```



# Map

- O exemplo define uma variável chamada "tags" para representar um mapa de pares chave-valor no Terraform para trabalhar com recursos na Microsoft Azure.
- A linha **type** = map(string) declara o tipo da variável como um mapa que contém valores do tipo string. Isso significa que tanto as chaves quanto os valores no mapa devem ser do tipo string.
- A linha **default** = { Name = "demo-instance", Environment = "production" } define os valores padrão para a variável "tags". Neste caso, são fornecidos dois pares chave-valor, onde "Name" está associado a "demo-instance" e "Environment" está associado a "production".
- Quando você usar essa variável, poderá fornecer seus próprios conjuntos de pares chave-valor para substituir ou adicionar aos valores padrão.

```
1 # Exemplo de variável "tags" como tipo map para Azure
2 variable "tags" {
3     type      = map(string)
4     default = {
5         Name      = "demo-instance"
6         Environment = "production"
7     }
8 }
```



# Object

- O exemplo define uma variável chamada "user" como um objeto no Terraform para representar dados estruturados na Microsoft Azure.
- A linha **type** = object({ name = string, email = string, age = number }) declara o tipo da variável como um objeto com atributos fixos. Este objeto possui três atributos: "name", "email" e "age", onde "name" e "email" são strings e "age" é um número.
- A linha **default** = { name = "Sam Doe", email = "sam@example.com", age = 30 } define os valores padrão para a variável "user". Neste caso, representa um exemplo de um usuário com nome "Sam Doe", email ["sam@example.com"](mailto:sam@example.com) e idade 30.
- Ao usar essa variável, você pode fornecer seus próprios valores para substituir ou adicionar aos valores padrão para os atributos do objeto.

```
1  # Exemplo de variável "user" como tipo object para Azure
2  variable "user" {
3    type = object({
4      name = string
5      email = string
6      age = number
7    })
8    default = {
9      name = "Sam Doe"
10     email = "sam@example.com"
11     age = 30
12   }
13 }
```



# Set

- O exemplo define uma variável chamada "nsgs" como um conjunto (set) no Terraform para representar um conjunto de valores únicos na Microsoft Azure.
- A linha **type** = set(string) declara o tipo da variável como um conjunto que contém valores do tipo string. Isso garante que os valores dentro do conjunto sejam únicos, sem repetições, e a ordem dos elementos não é considerada.
- A linha **default** = ["nsg-12345678", "nsg-abcdefgh"] define os valores padrão para a variável "nsgs". Neste caso, são fornecidos dois valores de exemplo para o conjunto. Lembre-se de que em um conjunto, os valores são únicos, então se você tentar adicionar um valor duplicado, ele será considerado apenas uma vez no conjunto.



```
1 # Exemplo de variável "nsgs" como tipo set para Azure
2 variable "nsgs" {
3     type      = set(string)
4     default   = ["nsg-12345678", "nsg-abcdefgh"]
5 }
```



# Tuple

- O exemplo define uma variável chamada "network\_addresses" como uma tupla no Terraform para representar uma coleção ordenada de elementos na Microsoft Azure.
- A linha **type** = tuple([string, string]) declara o tipo da variável como uma tupla que contém dois elementos, ambos do tipo string. As tuplas mantêm a ordem dos elementos e são imutáveis, ou seja, uma vez criadas, não podem ser alteradas.
- A linha **default** = ["192.168.1.1", "192.168.1.2"] define os valores padrão para a variável "network\_addresses". Neste caso, são fornecidos dois endereços IP como exemplo na tupla.



```
1 # Exemplo de variável "network_addresses" como tipo tuple para Azure
2 variable "network_addresses" {
3     type      = tuple([string, string])
4     default   = ["192.168.1.1", "192.168.1.2"]
5 }
```



The background features abstract, glowing geometric shapes in blue and orange. On the left, there is a large, complex blue wireframe structure that resembles a 3D cube or a series of interconnected planes. To its right and slightly below, there is a smaller orange wireframe structure. A large, faint orange circle is also visible, partially overlapping the blue structure. The overall aesthetic is futuristic and technical.

# Hands-On

*Terraform collection  
and structural types*



**TFTEC CLOUD**



The background features several glowing purple geometric shapes, including a large cube on the left and various intersecting lines and arcs, creating a futuristic, digital aesthetic.

**Create and differentiate  
resource and data  
configuration**



# Trabalhando com Data Blocks no Terraform

Blocos de Dados são elementos-chave no Terraform que permitem acessar informações de fontes externas, como APIs de provedores de nuvem.

Eles são úteis para consultar dados existentes e exportar atributos importantes para a configuração de infraestrutura.

Os benefícios principais em usar data blocks no Terraform:

- Acesso rápido a informações específicas de recursos provisionados no Azure.
- Utilização desses dados para configurar novos recursos ou ajustar a infraestrutura existente na nuvem da Microsoft.
- Reutilização eficaz de informações provenientes de consultas em diferentes partes da configuração do Terraform no ambiente Azure.
- Simplificação do processo de utilização de dados específicos em várias configurações de infraestrutura.



# Consultar ou exportar dados coletados

## Consultar recursos existentes com bloco de dados

Utilizar um bloco de dados para consultar informações de recursos já existentes no Microsoft Azure.


```
1 data "azurerm_virtual_network" "example" {  
2   name           = "myVirtualNetwork"  
3   resource_group_name = azurerm_resource_group.example.name  
4 }
```

## Exportar atributos de uma consulta de dados

Exportar atributos obtidos através de uma consulta de dados para uso em outras configurações do Terraform.

```
1 output "virtual_network_id" {  
2   value = data.azurerm_virtual_network.example.id  
3 }
```



The background features a dark, textured purple field. On the left, there are several glowing purple wireframe shapes, including a large cube and a smaller rectangular prism. A curved purple line with an arrowhead points from the bottom left towards the center. Another curved purple line with an arrowhead points from the bottom right towards the center. The text is centered in the middle-right area.

**Use resource addressing  
and resource parameters  
to connect resources  
together**



# Conectando recursos no Terraform

A conexão de recursos é fundamental para estabelecer relacionamentos entre diferentes elementos na infraestrutura. O Terraform oferece recursos para endereçamento, parâmetros e gerenciamento de dependências para facilitar essa conexão.

## Definição de conexões diretas:

```
1 resource "azurerm_virtual_network" "example" {
2   name                       = "myVirtualNetwork"
3   resource_group_name = azurerm_resource_group.example.name
4 }
5
6 resource "azurerm_subnet" "example" {
7   name                       = "mySubnet"
8   resource_group_name = azurerm_resource_group.example.name
9   virtual_network_name = azurerm_virtual_network.example.name
10  address_prefixes         = ["10.0.1.0/24"]
11 }
```

# Gerenciando a ordem de provisionamento

No Terraform, **depends\_on** é um parâmetro utilizado para estabelecer dependências explícitas entre recursos. Ele permite controlar a ordem de criação ou atualização dos recursos, garantindo que um recurso dependente só será provisionado após a conclusão da criação ou atualização dos recursos nos quais ele depende.

**Controle de Dependências:** O parâmetro `depends_on` é usado para indicar que um recurso no Terraform depende de outros recursos para ser provisionado corretamente.

**Definição de Ordem de Provisionamento:** Ele define a ordem em que os recursos devem ser criados ou atualizados. Isso é particularmente útil quando um recurso precisa referenciar outro recurso já existente no momento da criação.

**Garantia de Ordem de Execução:** Garante que o Terraform crie ou atualize os recursos na ordem especificada, evitando erros relacionados à falta de recursos necessários para a configuração correta de outros recursos.

```
1 resource "azurerm_resource_group" "example" {
2   name      = "myResourceGroup"
3   location  = "West Europe"
4 }
5
6 resource "azurerm_virtual_machine" "example_vm" {
7   name                = "myVirtualMachine"
8   location             = azurerm_resource_group.example.location
9   resource_group_name = azurerm_resource_group.example.name
10
11   # Estabelecendo Dependência
12   depends_on = [
13     azurerm_resource_group.example
14   ]
15
16   # Outras configurações da máquina virtual...
17 }
18
```



The background features several glowing purple geometric shapes, including a large cube on the left and various lines and arcs extending across the frame, creating a technical or architectural feel.

**Use HCL and Terraform  
functions to write  
configuration**



# Utilizando funções no Terraform

"HCL (HashiCorp Configuration Language) é a linguagem usada para configurar o Terraform, permitindo o uso de suas funções para operações específicas.

O Terraform oferece recursos dinâmicos poderosos, como **count** e **for\_each**, que possibilitam a criação flexível e repetitiva de recursos, como por exemplo: múltiplas VMs com nomes únicos.

A utilização inteligente dessas funcionalidades dinâmicas no Terraform proporciona eficiência e automação na criação e gerenciamento de recursos, trazendo flexibilidade à infraestrutura na nuvem."



# COUNT - Criando recursos repetidos

- **count:** Permite a repetição de recursos de forma direta e eficiente, ideal para a criação de múltiplas instâncias idênticas de um recurso.



```
1  resource "azurerm_virtual_machine" "example" {
2    count                = 3
3    name                 = "myVM-${count.index}"
4    location             = "West Europe"
5    resource_group_name = "myResourceGroup"
6    vm_size              = "Standard_DS1_v2"
7
8    # Outras configurações da máquina virtual...
9  }
```



# FOR\_EACH - Criando recursos a partir de uma Lista de valores

- **for\_each**: Facilita a criação de recursos baseados em uma lista de valores, permitindo nomes ou configurações personalizadas para cada instância..

```
1  variable "instance_names" {
2    type      = set(string)
3    default   = ["instance1", "instance2", "instance3"]
4  }
5
6  resource "azurerm_virtual_machine" "example" {
7    for_each   = toset(var.instance_names)
8    name       = "myVM-${each.value}"
9    location   = "West Europe"
10   resource_group
```



The background features abstract, glowing purple geometric shapes, including a large cube-like structure on the left and various intersecting lines and arcs, set against a dark, textured purple background.

# Hands-On

## *Terraform Functions*



**TFTEC CLOUD**



The background features several glowing purple geometric shapes, including a large cube on the left and various intersecting lines and arcs on the right, creating a technical or architectural feel.

**Describe built-in  
dependency management  
(order of execution based)**



**TFTEC CLOUD**



# Gestão de dependências incorporada

O Terraform automatiza a gestão de dependências entre recursos. Ele determina a ordem de execução com base nas relações definidas entre eles.

Por exemplo, se você criar um servidor de aplicativos no Azure que depende de um plano de serviço, o Terraform garantirá que o plano de serviço seja criado primeiro, para que o servidor de aplicativos tenha uma infraestrutura adequada para ser implantado.

Quando fazemos modificações em recursos existentes no Azure usando o Terraform, pode ocorrer um problema onde a destruição de um recurso original impede a criação do novo. Isso é especialmente notável em casos onde a destruição do recurso afeta a existência de outros recursos dependentes.

Por padrão, o Terraform executa a substituição de recursos destruindo e recriando-os. No entanto, é possível modificar essa ordem de operações com a diretiva **create\_before\_destroy**.

# create\_before\_destroy

```
1 resource "azurerm_resource_group" "example" {
2   name      = "myResourceGroup"
3   location = "West Europe"
4 }
5
6 resource "azurerm_network_security_group" "example_nsg" {
7   name                = "myNSG"
8   location             = azurerm_resource_group.example.location
9   resource_group_name = azurerm_resource_group.example.name
10
11   # Defina suas regras de segurança aqui...
12 }
13
14 resource "azurerm_network_interface" "example_nic" {
15   name                = "myNIC"
16   location             = azurerm_resource_group.example.location
17   resource_group_name = azurerm_resource_group.example.name
18
19   # Defina a configuração da interface de rede aqui...
20 }
21
22 resource "azurerm_virtual_machine" "example_vm" {
23   name                = "myVM"
24   location             = azurerm_resource_group.example.location
25   resource_group_name = azurerm_resource_group.example.name
26   network_interface_ids = [azurerm_network_interface.example_nic.id]
27   # Outras configurações da máquina virtual...
28
29   lifecycle {
30     # Utilizando create_before_destroy para permitir a coexistência de versões
31     create_before_destroy = true
32   }
33 }
```

Quando o Terraform determina que precisa destruir um objeto e recriá-lo, o comportamento normal criará o novo objeto após a destruição do existente.

Usar este atributo criará primeiro o novo objeto e depois destruirá o antigo.

Isso pode ajudar a reduzir o tempo de inatividade.

Alguns objetos têm restrições com as quais o uso desta configuração pode causar problemas, impedindo que objetos existam simultaneamente.

Portanto, é importante compreender quaisquer restrições de recursos antes de usar esta opção.



# prevent\_destroy

Outra diretiva importante é a **prevent\_destroy**, que alerta quando uma alteração resulta na destruição de um recurso no Azure. Essa diretiva é fundamental para proteger recursos críticos de serem excluídos acidentalmente.

Suponha que você queira evitar a remoção acidental de uma Máquina Virtual crítica no Azure.

Neste caso, ao definir **prevent\_destroy** como verdadeiro para a Máquina Virtual crítica, o Terraform avisará se qualquer mudança resultar na tentativa de remoção da máquina. Isso ajuda a evitar a exclusão acidental de um recurso crítico no Azure.

```
1 resource "azurerm_resource_group" "example" {
2   name      = "myResourceGroup"
3   location  = "West Europe"
4 }
5
6 resource "azurerm_virtual_machine" "critical_vm" {
7   name                = "criticalVM"
8   location             = azurerm_resource_group.example.location
9   resource_group_name = azurerm_resource_group.example.name
10  # Outras configurações da máquina virtual...
11
12  lifecycle {
13    prevent_destroy = true
14  }
15 }
```

# Ignore\_changes

O Terraform permite um controle refinado sobre quais mudanças em recursos devem ser consideradas durante a execução de terraform apply.

A diretiva **ignore\_changes** é útil para especificar quais atributos de um recurso devem ser ignorados pelo Terraform, evitando a aplicação de certas alterações.

```
1 # Recurso: Conta de Armazenamento (Storage Account)
2 resource "azurerm_storage_account" "example_storage" {
3   name                        = "examplestorage"
4   resource_group_name        = azurerm_resource_group.example.name
5   location                   = "East US"
6   account_tier                = "Standard"
7   account_replication_type   = "LRS"
8
9   # Configurações adicionais...
10 }
11
12 # Diretiva 'ignore_changes' aplicada à Conta de Armazenamento
13 lifecycle {
14   ignore_changes = [
15     # Ignora mudanças no atributo 'account_replication_type'
16     azurerm_storage_account.example_storage.account_replication_type
17   ]
18 }
```

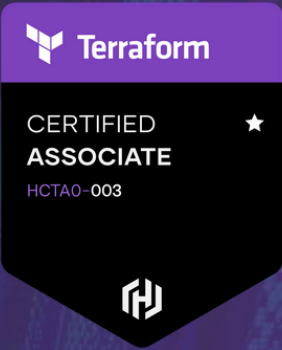
Neste exemplo, temos uma Conta de Armazenamento (**azurerm\_storage\_account**) no Azure. A diretiva **ignore\_changes** é aplicada ao atributo **account\_replication\_type** da Conta de Armazenamento.

Isso significa que, ao realizar alterações nesse atributo, o Terraform não aplicará as mudanças durante a execução do terraform apply, ignorando especificamente as alterações neste campo.

## Benefícios do ignore\_changes:

- Evita a aplicação de mudanças específicas em atributos de recursos durante o processo de aplicação (apply) do Terraform.
- Permite controlar quais alterações devem ser consideradas e aplicadas pelo Terraform em recursos específicos no Azure.





# Obrigado