


# HashiCorp Certified Terraform Associate 003



## 5. Interact with Terraform modules



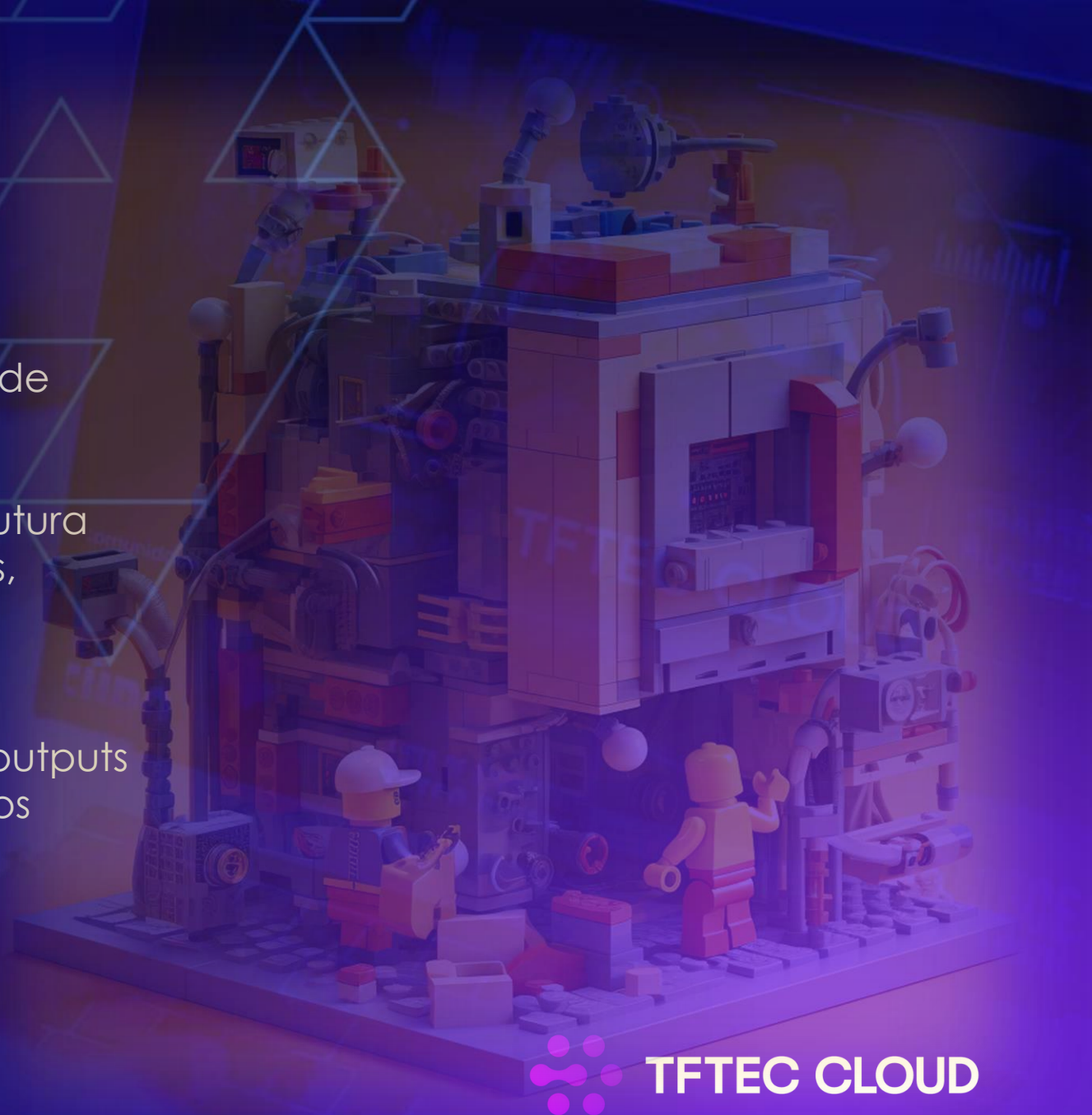
The background features abstract, glowing blue and white geometric shapes, including a large cube on the left and various lines and arcs extending across the frame, set against a black background with subtle white speckles.

**Contrast and use different  
module source options  
including the public  
Terraform Module Registry**



# Módulos no Terraform

- Módulos no Terraform são uma abstração poderosa para organizar e reutilizar código de infraestrutura.
- Eles permitem que você divida sua infraestrutura em componentes modulares e gerenciáveis, tornando o código mais legível e de fácil manutenção.
- Módulos podem conter recursos, variáveis, outputs e outros elementos do Terraform, tornando-os versáteis.



TFTEC CLOUD

# Benefícios dos módulos

- **Reutilização de Código:** Os módulos permitem compartilhar configurações de infraestrutura comuns entre vários projetos, economizando tempo e esforço.
- **Abstração:** Eles fornecem uma camada de abstração que oculta detalhes de implementação complexos, permitindo que os usuários interajam com os módulos de forma simples.
- **Organização:** Os módulos ajudam a organizar e estruturar seu código, facilitando a localização e a manutenção de recursos específicos.
- **Compartilhamento na Comunidade:** A comunidade Terraform oferece um repositório público chamado Terraform Module Registry, onde você pode compartilhar e encontrar módulos prontos para uso.



# Estrutura de um módulo

Um módulo Terraform é composto por uma estrutura de diretórios e arquivos que inclui:

- **main.tf:** Este arquivo contém a configuração dos recursos específicos do módulo.
- **variables.tf:** Aqui, você define as variáveis que podem ser passadas para o módulo para personalização.
- **outputs.tf:** Você pode especificar quais valores o módulo deve retornar como outputs.
- Outros arquivos e diretórios para organizar e documentar seu módulo.

# Fontes de módulos no Terraform

O Terraform oferece várias opções de origem para módulos, incluindo:

- **Local:** Você pode especificar o caminho local para um módulo em seu sistema de arquivos.
- **GitHub:** É possível referenciar um repositório Git hospedado no GitHub como fonte do módulo.
- **Bitbucket:** Similar ao GitHub, você pode usar um repositório Git hospedado no Bitbucket.
- **Git:** Referencie um repositório Git remoto, seja público ou privado.
- **Terraform Module Registry:** Este é um repositório público que contém uma ampla variedade de módulos prontos para uso.



# Utilizando o Terraform Module Registry

O Terraform Module Registry é uma fonte confiável e centralizada de módulos Terraform.

- Pode ser acessado em [registry.terraform.io](https://registry.terraform.io), onde você pode explorar e buscar módulos.
- Essa é uma maneira eficiente de encontrar módulos desenvolvidos pela comunidade que atendem às suas necessidades.

```
1 module "azure_vnet" {  
2   source = "Azure/network/azurerm"  
3   version = "2.0.0"  
4  
5   name = "my-vnet"  
6   resource_group_name = "my-resource-group"  
7   location = "East US"  
8   address_space = ["10.0.0.0/16"]  
9 }
```



# Publicando módulos no Terraform Module Registry

Passos para publicar um módulo:

1. **Crie um Módulo:** Desenvolva seu próprio módulo reutilizável do Terraform. Certifique-se de que ele siga as melhores práticas e seja bem documentado.
2. **Versionamento:** Escolha uma versão para o seu módulo. Siga práticas de versionamento semântico para garantir que outros usuários possam confiar nas atualizações.
3. **Empacote seu Módulo:** Prepare seu módulo para ser distribuído. Certifique-se de incluir todos os arquivos necessários e defina metadados claros.
4. **Publicação:** Use a interface do Terraform Module Registry para publicar seu módulo. Você precisará de uma conta no HashiCorp para fazer isso.
5. **Documentação:** Forneça uma documentação clara e informativa para que outros usuários saibam como usar seu módulo.

Ao compartilhar módulos, é importante seguir as práticas recomendadas para documentação e versionamento.





The background features several glowing purple geometric shapes, including a large cube on the left and a triangular prism at the bottom. Faint, curved lines and small dots are scattered across the dark purple background, creating a futuristic or digital aesthetic.

**Interact with module inputs  
and outputs**



# Entendendo as entradas de módulos

No Terraform, os módulos são componentes reutilizáveis que permitem organizar e compartilhar seu código de infraestrutura.

Ao usar módulos, você precisará entender como interagir com as entradas (inputs) e saídas (outputs) desses módulos.

- As entradas de módulos são os parâmetros que você fornece quando instancia um módulo em seu código Terraform.
- As entradas permitem que você personalize o comportamento do módulo de acordo com suas necessidades.
- Vamos considerar um exemplo de uso de um módulo para criar uma instância de máquina virtual (VM) no Azure.

```
1  module "azure_vm" {  
2      source = "Azure/compute/azurerm"  
3  
4      vm_name           = "my-vm"  
5      resource_group    = "my-resource-group"  
6      location          = "East US"  
7      vm_size           = "Standard_DS2_v2"  
8      admin_username    = "adminuser"  
9      admin_password    = "password123"  
10 }
```



# Entendendo as saídas de módulos

As saídas de módulos são os valores que um módulo pode retornar para serem utilizados em outras partes do seu código Terraform.

As saídas permitem que você acesse informações ou recursos criados pelo módulo.

- Neste exemplo, estamos definindo uma saída chamada **vm\_ip\_address**, que retorna o endereço IP público da VM criada pelo módulo.
- Podemos usar essa saída em outras partes do código Terraform.

```
1 output "vm_ip_address" {  
2   value = module.azure_vm.public_ip_address  
3 }
```



# Interagindo com entradas (input) e saídas (outputs)

Para interagir com as entradas e saídas de módulos, você pode usá-las em outros recursos ou módulos.

As saídas de um módulo podem ser referenciadas usando **module.<module\_name>.<output\_name>**.

```
1 resource "azurerm_network_security_group" "example" {
2   name           = "example-nsg"
3   location       = "East US"
4   resource_group_name = "my-resource-group"
5
6   security_rule {
7     name           = "allow_ssh"
8     priority       = 1001
9     direction      = "Inbound"
10    access         = "Allow"
11    protocol        = "Tcp"
12    source_port_range = "*"
13    destination_port_range = "22"
14    source_address_prefix = "*"
15    destination_address_prefix = module.azure_vm.vm_ip_address
16  }
17 }
```

- Neste exemplo, estamos usando a saída **vm\_ip\_address** do módulo da VM no Azure como o valor de destino para uma regra de segurança de rede (Network Security Group - NSG).
- Isso permite que apenas o endereço IP público da VM acesse a porta SSH na NSG.



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 variable scope  
within modules/child  
modules**



# O que são Child Modules?

- Às vezes, você precisa entender como as variáveis são escopadas quando se trabalha com módulos aninhados, também conhecidos como child modules.
- Child modules, ou módulos aninhados, são módulos que são usados dentro de outros módulos.
- Eles permitem a composição de recursos e lógica de infraestrutura em níveis mais granulares.



**TFTEC CLOUD**



# Escopo de variáveis em módulos

As variáveis em Terraform podem ser definidas em três níveis: módulo pai (root module), módulo filho (child module) e variáveis de entrada.

A ordem de precedência das variáveis é importante e afeta como elas são resolvidas.

1. **Variáveis de entrada (Input Variables):** São passadas para o módulo como parâmetros.
2. **Variáveis de módulo pai (Variables in Parent Module):** Definidas no módulo que chama o módulo filho.
3. **Variáveis de módulo filho (Variables in Child Module):** Definidas no módulo filho (child module) em si.



# Exemplo de uso das variáveis

Vamos considerar um exemplo de módulo pai que chama um **módulo filho** para criar uma VM no Azure.

```
1 module "azure_vm" {
2   source = "../modules/azure_vm"
3
4   vm_name = var.vm_name
5 }
```

Neste exemplo, **var.vm\_name** é uma variável de entrada que é passada para o módulo filho.

Agora, vamos dar uma olhada em como as variáveis são definidas no módulo filho.

```
1 variable "vm_name" {
2   description = "Name of the VM"
3 }
4
5 resource "azurerm_virtual_machine" "example" {
6   name                = var.vm_name
7   # outras configurações aqui
8 }
```

Neste exemplo, **var.vm\_name** é uma variável de entrada que é passada para o módulo filho.



The background features several glowing purple geometric shapes, including a large cube on the left and a triangular prism at the bottom. A faint circular arc is also visible. The overall aesthetic is futuristic and digital.

# Set module version



# Por que definir versões de módulos?

Ao usar módulos no Terraform, é importante entender como definir e especificar a versão de um módulo.

As versões garantem a consistência e a estabilidade do código de infraestrutura.

- Definir versões de módulos é crucial para evitar surpresas e garantir que a infraestrutura seja criada conforme o esperado.
- As atualizações automáticas de módulos podem causar problemas se não forem gerenciadas.



# Como definir a versão de um módulo

- Para definir a versão de um módulo, você usa a diretiva `version`.
- A versão pode ser uma string que representa um número de versão ou uma faixa de versão.
- Considere o seguinte exemplo de como definir a versão de um módulo em seu código Terraform:
- Neste exemplo, estamos usando o módulo "virtual-machine/azurerm" da organização "hashicorp" com a versão específica "2.1.0".

```
1 module "example" {  
2   source = "hashicorp/virtual-machine/azurerm"  
3   version = "2.1.0"  
4   # outras configurações aqui  
5 }
```



# Usando faixas de versão

Além de especificar versões específicas, você pode usar faixas de versão para flexibilidade.

Neste caso, "~> 2.0" permite qualquer versão a partir de "2.0" até, mas não incluindo "3.0".



```
1  module "example" {  
2    source  = "hashicorp/virtual-machine/azurerm"  
3    version = "~> 2.0"  
4    # outras configurações aqui  
5  }
```



# Resolvendo a versão do Módulo

- Quando você executa o comando terraform init, o Terraform resolve a versão do módulo com base nas especificações no arquivo de configuração.
- Ele baixa a versão especificada ou a versão mais recente dentro da faixa especificada.

## Atualizando a versão do módulo

- Para atualizar a versão de um módulo, basta modificar a diretiva version em seu arquivo de configuração Terraform.
- Certifique-se de testar as atualizações em um ambiente seguro.

## Bloqueio de Versão

- É uma prática recomendada adicionar um arquivo de bloqueio de versão (version lock) ao controle de origem para garantir que todos os colaboradores usem a mesma versão do módulo.
- Isso evita atualizações acidentais.

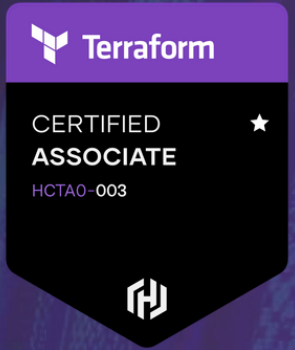


The background features abstract, glowing purple geometric shapes, including a large cube-like structure on the left and various intersecting lines and arcs, set against a dark, textured purple background.

# Hands-On

## *Terraform Modules*





# Obrigado