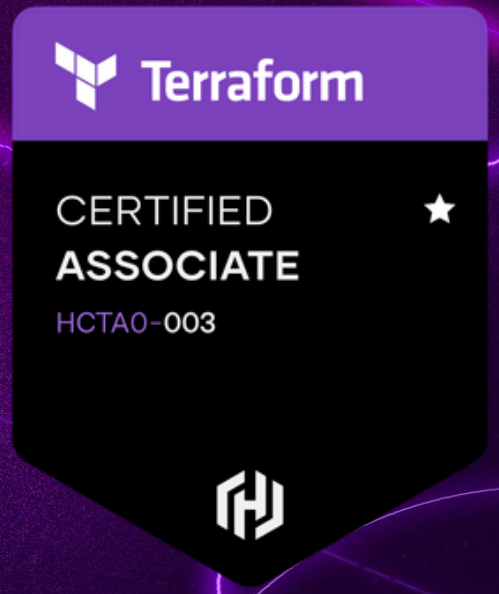


HashiCorp Certified Terraform Associate

003

TFTEC CLOUD


3. Understand Terraform basics
TFTEC CLOUD

Terraform HCL Basics

TFTEC CLOUD

HashiCorp Configuration Language (HCL)

A **HashiCorp Configuration Language (HCL)** é uma linguagem de configuração de autoria da HashiCorp. A **HCL** é usada com ferramentas de automação de infraestrutura em nuvem da HashiCorp, como o Terraform. A linguagem foi criada com o objetivo de ser amigável tanto para humanos quanto para máquinas. 12 de dezembro de 2018

Pontos fortes do HCL:

- **Legibilidade e Expressividade:**
  - O **HCL** é elogiado por sua sintaxe limpa e fácil de entender, o que facilita a criação e manutenção de configurações complexas.
  - Ele permite uma descrição mais humana e próxima do idioma natural da infraestrutura, o que aprimora a colaboração e compreensão.

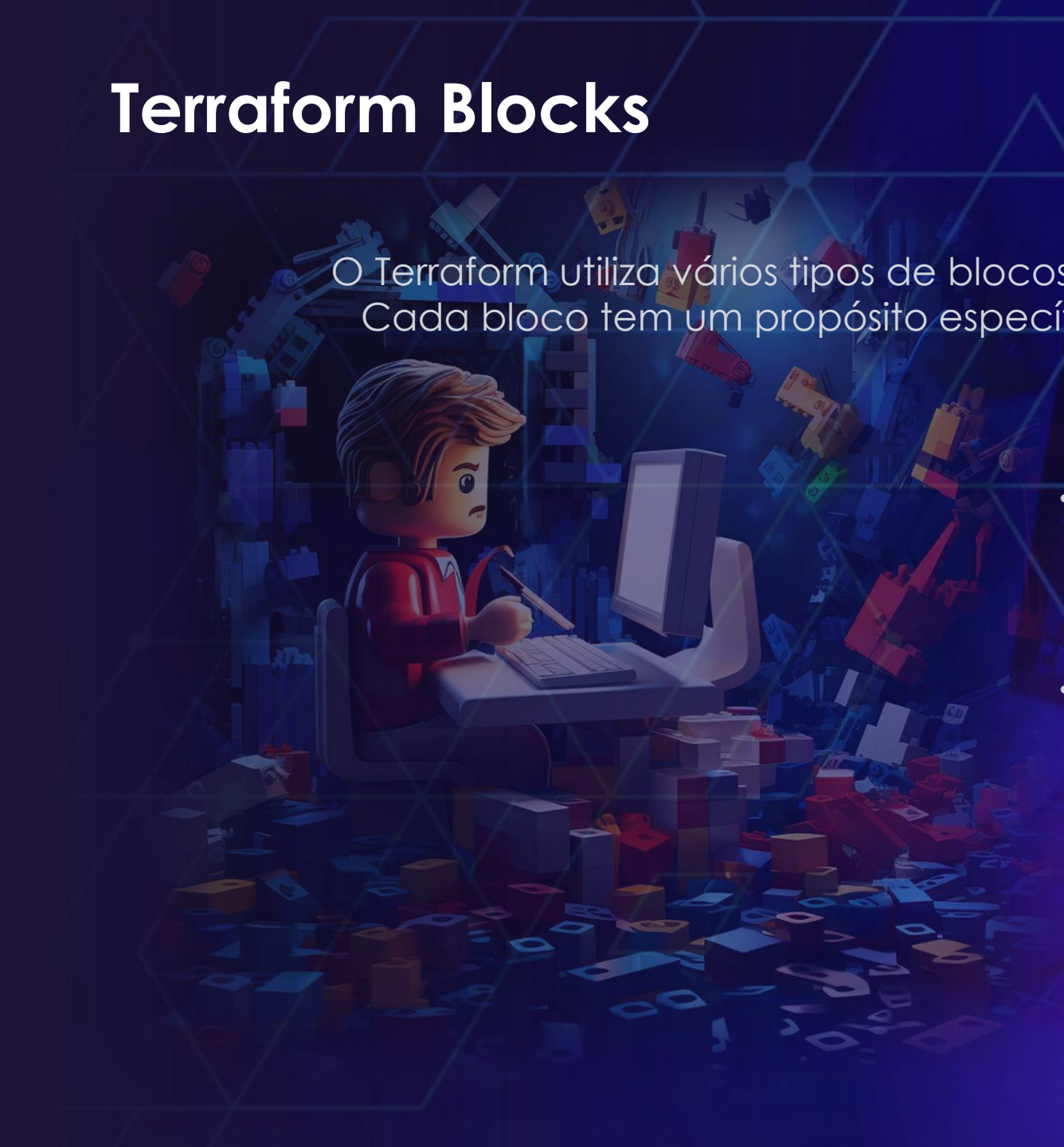
- **Suporte e Ecossistema:**
  - O HCL é amplamente adotado na comunidade de DevOps e gerenciamento de infraestrutura.
  - Além do **Terraform**, outras ferramentas da **HashiCorp**, como **Packer** e **Consul**, também utilizam o HCL para configurações.

```
# Exemplo de configuração em HCL
<BLOCK TYPE> "<RESOURCE TYPE>" "<BLOCK NAME>" {
  # Block body
    <IDENTIFIER> = <EXPRESSION> # Argument
  }

# Exemplo de configuração em HCL para Microsoft Azure
resource "azurerm_resource_group" "example" {
  name     = "example-resources"
  location = "East US"
}
```

TFTEC CLOUD

Terraform Blocks



O Terraform utiliza vários tipos de blocos para descrever a infraestrutura como código. Cada bloco tem um propósito específico e desempenha um papel fundamental na configuração do ambiente de nuvem.

- **Tipos de Blocos**:
  - Provider Block, Resource Block, Data Block, Variable Block, Local Block, Output Block, Module Block, entre outros.

- **Organização e Estrutura**:
  - Os blocos são organizados em arquivos de configuração HCL para definir e gerenciar recursos.
  - Cada bloco possui atributos e configurações específicas para personalizar o comportamento da infraestrutura.

TFTEC CLOUD

Terraform Settings Block

Configurações Globais no Terraform:

- O bloco terraform é usado para configurar opções globais no Terraform.
- Isso inclui configurações como versão do backend, backend remoto e configurações de provedor.

Especificações de Provedor:

- Você pode definir as versões específicas dos provedores que deseja utilizar em seu projeto.
- Isso garante que seu projeto seja compatível com as versões de provedor específicas.

Backend Remoto:

- É possível configurar um backend remoto para armazenar o estado do Terraform de forma segura e colaborativa.
- Isso facilita a colaboração em equipe e a escalabilidade.

```
1  # Configurações globais no Terraform
2  terraform {
3    required_providers {
4      azurerm = {
5        source  = "hashicorp/azurerm"
6        version = "2.0.0"
7      }
8    }
9
10   backend "azurerm" {
11     ...
12   }
13 }
```

TFTEC CLOUD

Terraform Provider Block

Definição do Provedor:

- O bloco provider é usado para definir qual provedor de nuvem será utilizado para provisionar recursos.
- Inclui informações como nome do provedor, versão e configurações específicas.

Exemplos de Provedores:

- Você pode utilizar provedores como **azurerm** para Azure, **aws** para AWS, **google** para Google Cloud, entre outros.
- Cada provedor possui suas próprias configurações e recursos disponíveis.

```
1  # Definindo o provedor Azure
2  provider "azurerm" {
3    features {}
4  }
```

TFTEC CLOUD

Terraform Resource Block

Definindo Recursos:

- O bloco resource é usado para definir os recursos que serão provisionados no provedor de nuvem.
- Inclui atributos específicos do recurso, como nome, localização e configurações detalhadas.

Exemplo de Recurso:

- Um exemplo pode ser a definição de uma máquina virtual em um provedor Azure.
- Isso envolve atributos como nome, localização, tamanho da máquina e configurações de rede.

```
# Definindo uma máquina virtual no Azure
resource "azurerm_virtual_machine" "example" {
  name                = "example-vm"
  location            = azurerm_resource_group.example.location
  ...
}
```

TFTEC CLOUD

Terraform Data Block

Consulta de Informações:

- O bloco data permite consultar informações sobre recursos já existentes no provedor.
- É útil quando você precisa acessar detalhes de recursos criados anteriormente.

Exemplo de Consulta:

- Pode ser usado para consultar informações sobre uma imagem do Azure existente, incluindo nome e grupo de recursos.

```
# Consultando informações sobre uma imagem do Azure
data "azurerm_image" "example" {
  name                = "UbuntuServer"
  resource_group_name = azurerm_resource_group.example.name
}
```

TFTEC CLOUD

Terraform Input and Local Variables Blocks

Variáveis de Entrada:

- Os blocos variable permitem a definição de variáveis que podem ser usadas em várias partes do código.
- Isso facilita a personalização e parametrização das configurações.

Variáveis Locais:

- Os blocos locals permitem a criação de variáveis locais para auxiliar na organização do código.
- Elas podem ser usadas para armazenar valores intermediários ou simplificar expressões complexas.

```
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 }
```

TFTEC CLOUD

Terraform Output Values Block

Exposição de Dados:

- O bloco output permite a exposição de informações úteis após a implantação.
- Essas informações podem ser usadas para comunicação com outros sistemas ou equipes.

Exemplo de Exposição:

- Um exemplo pode ser expor o endereço IP público de uma máquina virtual após a implantação.

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

TFTEC CLOUD

Terraform Modules Block

Módulos para Organização:

- O bloco module permite a organização e reutilização de código em projetos Terraform.
- Os módulos são usados para agrupar recursos relacionados e simplificar a gestão da infraestrutura.

Exemplo de Uso de Módulo:

- Mostrar um exemplo de como importar e usar um módulo em um arquivo de configuração Terraform.

```
# Utilizando um módulo personalizado
module "example" {
  source = "./modules/my_module"
  ...
}
```

TFTEC CLOUD

Terraform Basic Commands

•Existem vários comandos fundamentais no Terraform:

- **terraform init**: Inicializa um novo diretório de configuração.
- **terraform validate**: Verifica a sintaxe e a configuração.
- **terraform plan**: Gera um plano de implantação.
- **terraform apply**: Aplica as configurações e provisiona recursos.
- **terraform destroy**: Remove todos os recursos provisionados.

```
Usage: terraform [global options] <subcommand> [args]

Main commands:
  init          Prepare your working directory for other commands
  validate      Check whether the configuration is valid
  plan          Show changes required by the current configuration
  apply         Create or update infrastructure
  destroy       Destroy previously-created infrastructure

All other commands:
  console       Try Terraform expressions at an interactive command prompt
  fmt           Reformat your configuration in the standard style
  force-unlock  Release a stuck lock on the current workspace
  get           Install or upgrade remote Terraform modules
  graph         Generate a Graphviz graph of the steps in an operation
  import        Associate existing infrastructure with a Terraform resource
  login         Obtain and save credentials for a remote host
  logout        Remove locally-stored credentials for a remote host
  metadata      Metadata related commands
  output        Show output values from your root module
  providers     Show the providers required for this configuration
  refresh       Update the state to match remote systems
  show          Show the current state or a saved plan
  state         Advanced state management
  taint         Mark a resource instance as not fully functional
  test          Experimental support for module integration testing
  untaint       Remove the 'tainted' state from a resource instance
  version       Show the current Terraform version
  workspace     Workspace management

```

TFTEC CLOUD


Hands-On
Terraform HCL Basics
TFTEC CLOUD

Obrigado

TFTEC CLOUD


3. Understand Terraform basics
TFTEC CLOUD

Install and version Terraform providers

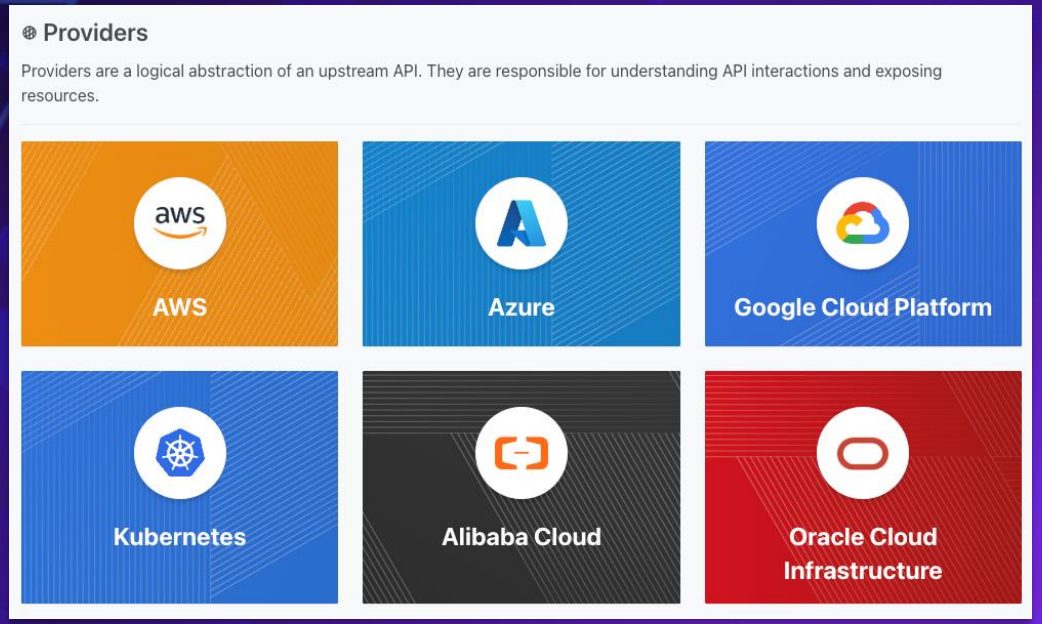
TFTEC CLOUD

Providers no Terraform

Os **providers** no Terraform desempenham um papel crucial. Eles atuam como intermediários, permitindo que o Terraform se comunique com sistemas remotos que possuem APIs. Esses providers implementam tipos de recursos que correspondem a serviços específicos em nuvens, plataformas e sistemas remotos.

É importante observar que o Terraform é agnóstico em relação à nuvem, o que significa que você pode escolher o provider que melhor se adapta ao seu ambiente, seja AWS, Azure, Google Cloud, VMware ou outros.

Os providers são essenciais para traduzir as configurações declarativas do Terraform em ações concretas nos sistemas remotos, tornando possível a automação e o gerenciamento da infraestrutura.



TFTEC CLOUD

Instalando o Provider do Azure

1. Configurar o Provider do Azure

```
terraform {
  required_version = ">= 1.0.0"
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "~> 2.0"
    }
  }
}
```

2. Após adicionar o provider ao arquivo de configuração, use o comando ***terraform init*** para baixar o provider.

```
terraform init
```

```
~/Desktop/MOD03-LAB1
terraform init

Initializing the backend...
Initializing modules...
- storage_module in storage_account

Initializing provider plugins...
- Finding hashicorp/azurerm versions matching "3.74.0"...
- Installing hashicorp/azurerm v3.74.0...
- Installed hashicorp/azurerm v3.74.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

TFTEC CLOUD

Versões do Terraform Providers

Cada provider do Terraform possui seu próprio conjunto de versões disponíveis, permitindo a evolução da funcionalidade ao longo do tempo.

Em suas configurações, você deve especificar restrições de versão para os provedores que seu projeto depende. Isso é feito usando o argumento "**version**" dentro do bloco de configuração.

- Em uma configuração do Terraform, você pode especificar as versões dos providers com as quais sua configuração é compatível.
- A declaração de dependências dos providers deve incluir uma restrição de versão no argumento "**version**", para que o Terraform possa selecionar uma versão específica compatível.

```
terraform {
  required_version = ">= 1.0.0"
  required_providers {
    mycloud = {
      source  = "hashicorp/azurerm"
      version = "~> 3.0"
    }
    random = {
      source  = "hashicorp/random"
      version = "3.1.0"
    }
  }
}
```

TFTEC CLOUD

Controlando versões no Terraform

1. Versão Exata (Bloqueio Rígido):

```
terraform {
  required_providers {
    azure = {
      source  = "hashicorp/azurerm"
      version = "= 2.0.0"
    }
  }
}
```

- Esta opção permite que você especifique uma versão exata do provider. Isso garante que apenas a versão especificada será usada pelo seu projeto.
- Neste exemplo, estamos indicando explicitamente que o Terraform deve usar a versão exata 2.0.0 do provider "hashicorp/azurerm" para o Azure.

TFTEC CLOUD

Controlando versões no Terraform

2. Versão Mínima (>=)

```
terraform {
  required_providers {
    azure = {
      source  = "hashicorp/azurerm"
      version = ">= 1.0.0"
    }
  }
}
```

- Esta opção permite que o Terraform utilize qualquer versão igual ou superior à especificada. Isso proporciona flexibilidade, permitindo que o Terraform utilize novas funcionalidades e correções de bugs introduzidas em versões posteriores.

- Neste exemplo, estamos indicando que o Terraform pode usar qualquer versão igual ou superior a 1.0.0 do provider "hashicorp/azurerm" para o Azure.

TFTEC CLOUD

Controlando versões no Terraform

3. Versão Máxima (<=):

```
terraform {
  required_providers {
    azure = {
      source  = "hashicorp/azurerm"
      version = "<= 2.0.0"
    }
  }
}
```

- Esta opção permite que o Terraform utilize qualquer versão igual ou inferior à especificada. Isso pode ser útil para evitar incompatibilidades com versões mais recentes.
- Neste exemplo, estamos indicando que o Terraform pode usar qualquer versão igual ou inferior a 2.0.0 do provider "hashicorp/azurerm" para o Azure.

TFTEC CLOUD

Controlando versões no Terraform

4. Versão com Operador de Incremento (~>):

```
terraform {
  required_providers {
    azure = {
      source  = "hashicorp/azurerm"
      version = "~> 2.0.1"
    }
  }
}
```

- Este operador permite atualizações de correção (patch) dentro da mesma versão principal. Isso é útil para receber correções de segurança e pequenas melhorias sem a introdução de alterações incompatíveis.

- Neste exemplo, estamos indicando que o Terraform pode usar qualquer versão dentro da faixa 2.0.x, mas não superior a 2.0.1, do provider "hashicorp/azurerm" para o Azure.

TFTEC CLOUD

Controlando versões no Terraform

5. Versão de Intervalo:

```
terraform {
  required_providers {
    azure = {
      source  = "hashicorp/azurerm"
      version = ">= 2.0.0, < 3.0.0"
    }
  }
}
```

- Esta opção permite especificar um intervalo de versões aceitáveis. Isso oferece um controle mais granular sobre as versões utilizadas.
- Neste exemplo, estamos indicando que o Terraform pode usar qualquer versão igual ou superior a 2.0.0, mas inferior a 3.0.0, do provider "hashicorp/azurerm" para o Azure.

TFTEC CLOUD

Controlando versões no Terraform

6. Sem Especificação de Versão:

```
terraform {
  required_providers {
    azure = {
      source  = "hashicorp/azurerm"
    }
  }
}
```

- Se você não especificar uma versão, o Terraform aceitará qualquer versão disponível do provider. Isso pode ser útil quando você confia no Terraform para selecionar a versão mais apropriada.

- Neste exemplo, estamos permitindo que o Terraform use qualquer versão do provider "hashicorp/azurerm" disponível para o Azure.

TFTEC CLOUD

Verificando a versão do Provider do Azure

1. Para verficar a versão do Terraform e do provider instalados.

```
terraform version
```

2. Isso irá exibir a versão do Terraform e também o provider Azure instalado.

```
Terraform v1.0.8
on linux_amd64
+ provider registry.terraform.io/hashicorp/azurerm v2.46.0
+ provider registry.terraform.io/hashicorp/random v3.1.0
```

TFTEC CLOUD

Obrigado

TFTEC CLOUD


3. Understand Terraform basics
TFTEC CLOUD

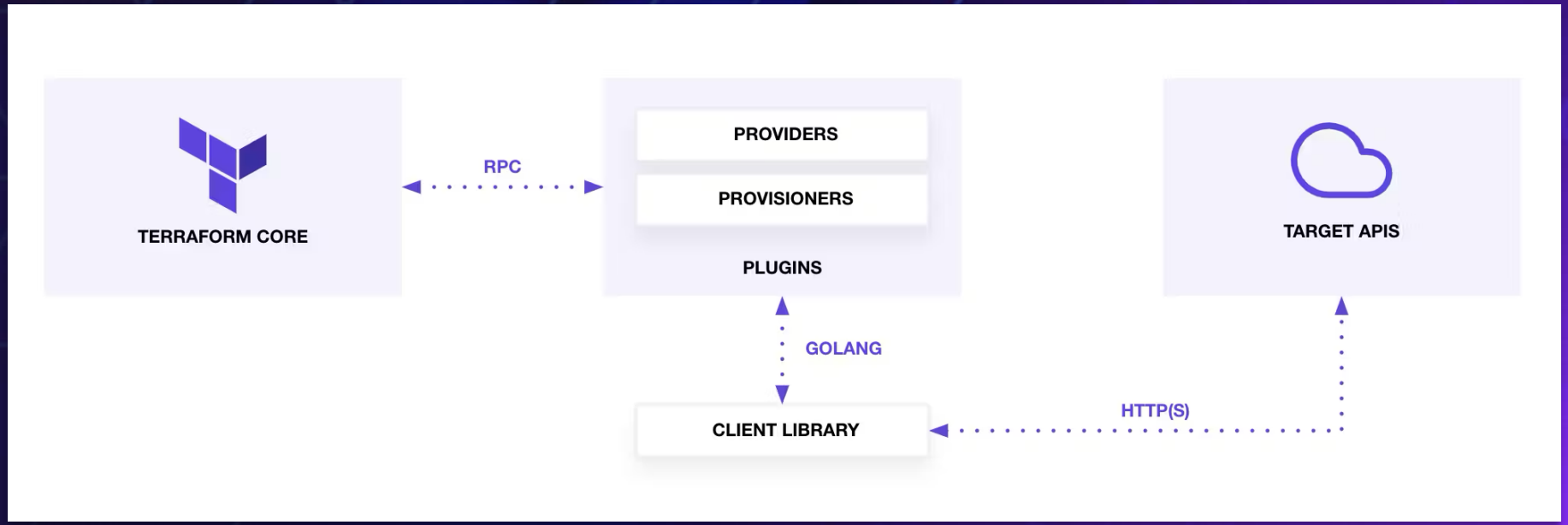
Describe plugin-based architecture

TFTEC CLOUD

Arquitetura Baseada em Plug-in do Terraform

O Terraform é construído sobre uma arquitetura baseada em plug-ins que permite a extensão e a flexibilidade da plataforma.

Existem dois componentes-chave na arquitetura do Terraform: **Terraform Core** e **Terraform plugins**.



TFTEC CLOUD

Terraform Core

Terraform Core é estaticamente binário compilado escrito na linguagem de programação Go.

Ele usa **RPCs** para se comunicar com plug-ins do Terraform e oferece várias maneiras de descobrir e carregar plug-ins para uso. O binário compilado é o Terraform CLI.

Responsabilidades do Terraform Core:

- IaC: Lendo e interpolando arquivos e módulos de configuração
- Gerenciamento de estado de recursos
- Construção de gráfico de recursos
- Execução do plano
- Comunicação com plugins via RPC

TFTEC CLOUD

Terraform Plugins

Os Terraform plugins são escritos em Go e são binários executáveis. Cada plugin expõe uma implementação para um serviço ou provedor específico, como Azure ou um provisionador.

- Os **plug-ins de provedores** são responsáveis pela inicialização de bibliotecas, autenticação e definição de recursos.
- Os **plug-ins de provisionadores** executam comandos ou scripts após a criação ou destruição de recursos.

TFTEC CLOUD

Utilizando os Plugins

- Ao declarar provedores e recursos no código do Terraform, você está indicando quais plug-ins devem ser usados.
- Os plug-ins são carregados automaticamente com base nas declarações feitas no código do Terraform.

Seleção e Atualização de Plug-ins

- O **terraform init** verifica as versões dos plug-ins instalados e escolhe as versões compatíveis de acordo com as restrições definidas no arquivo de configuração.
- O uso da opção **-upgrade** no **terraform init** permite verificar e baixar versões mais recentes dos provedores.

TFTEC CLOUD

Obrigado

TFTEC CLOUD


3. Understand Terraform basics
TFTEC CLOUD

Write Terraform configuration using multiple providers

TFTEC CLOUD

O que são Múltiplos Provedores?

```
terraform {
  backend "azurerm" {}

  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = ">= 3.28.0"
    }

    azuread = {
      source  = "hashicorp/azuread"
      version = ">= 2.29.0"
    }

    random = {
      source  = "hashicorp/random"
      version = "~> 3.4.3"
    }

    kubernetes = {
      source  = "hashicorp/kubernetes"
      version = "~> 2.13.0"
    }
  }
}
```

Múltiplos provedores no Terraform referem-se à capacidade de gerenciar recursos em diferentes ambientes de nuvem ou serviços.

Isso significa que você pode configurar e gerenciar recursos em provedores como AWS, Azure, Google Cloud, entre outros, dentro do mesmo projeto.

TFTEC CLOUD

Benefícios dos Múltiplos Provedores

- **Maior Flexibilidade**: Permite a alocação de recursos em diferentes provedores, de acordo com as necessidades específicas de cada aplicação.
- **Redundância e Disponibilidade**: Facilita a criação de ambientes redundantes em provedores distintos, aumentando a disponibilidade do sistema.
- **Melhor Utilização de Recursos**: Possibilita a escolha do provedor mais adequado para cada tipo de recurso, otimizando custos e desempenho.

TFTEC CLOUD


Hands-On
Using Multiple
Terraform Providers
TFTEC CLOUD

Obrigado

TFTEC CLOUD