

Implementando Recursos de Dados do Azure com Terraform

Wagner Crivelini



Apresentação:

Wagner Crivelini

- Consultor Senior da Microsoft Brasil
- Engenheiro de Dados
- Colunista de vários portais de tecnologia com +250 publicações



Agenda

[Introdução ao Terraform](#)

[Exemplo 1](#)

[Controle de Estado](#)

[Uma Estratégia Melhor](#)

[Exemplo 2](#)

[Múltiplos Recursos](#)

[Exemplo 3](#)

[Resumo](#)





Introdução ao Terraform

- Infrastructure as Code (IaC) adotado em larga escala
- Adoção de Terraform (HashiCorp) cresce rapidamente:
 1. Declarativo (evita excesso de detalhes)
 2. Idempotente (consistência)
 3. Agnóstico (trabalha com vários provedores)
 4. Open source (contribuições da comunidade)



Terraform – “Getting Started”

- Você pode instalar localmente e rodar com VS CODE / CMD

<https://www.terraform.io/downloads>

- Comandos básico:

C:\TEMP\terraform init

C:\TEMP\terraform validate #seus scripts

C:\TEMP\terraform plan #plano de execução dos scripts

C:\TEMP\terraform apply #executa scripts

C:\TEMP\terraform destroy #caso necessite fazer “rollback”

C:\TEMP\terraform show #mostra resultado da implementação



Terraform – Scripts

- Usa linguagem HCL ou JSON
- Permite escolher seu(s) provedor(es) de nuvem: para Azure, use **AzureRM**
<https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs>
- Cria/altera recursos apenas informando novos valores
- Usa **variáveis** ao invés de valores fixos
- Permite criar scripts mais legíveis (script único vs **múltiplos scripts**)
- Implementação de recursos é feita em paralelo



Terraform – Sintaxe Básica

- Scripts começam com a declaração de provedores e suas versões.

```
terraform {  
  required_providers {  
    azurerm = {  
      source  = "hashicorp/azurerm"  
      version = "~>3.0"  
      ##any azurerm version, from 3.0.0 or above  
    }  
  }  
}
```

- Identifica parâmetros dos recursos a serem implementados (ex:

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/resource_group)

```
resource "azurerm_resource_group" "rg" {  
  name      = "wagnerresourcegroup2024"  
  location = "East Us"  
}
```



Terraform – Como Executar Scripts

```
cd <path_to_scripts>
az login
rem to deploy to specific subscription run next command
az account set --subscription <subscriptionID>
terraform init
terraform validate
terraform plan [-out] <repositório/arquivoPlan.tfplan>
terraform apply [-auto-approve] [<repositório/arquivoPlan.tfplan>]
```



Exemplo 1 : Criando um Grupo de Recursos

DEMO



Controle de Estado

- Após primeira execução, novos arquivos serão criados
- Arquivo TFSTATE identifica o estado de cada recurso após última execução
- Antes da próxima execução, Terraform vai checar TFSTATE.
- **NUNCA PERCA/ALTERE TFSTATE**



Como Alterar ou Excluir Recursos

- Arquivo TFSTATE é a chave para implementações bem sucedidas
- Resultados esperados para próximas execuções de seus scripts:
 1. Se seus scripts não forem alterados – **nada acontece (IDEMPOTENT)**
 2. Se forem alterados valores de variáveis – **recursos afetados serão alterados**
 3. Se um recurso for excluído dos scripts – **recurso será excluído**
 4. Se o TFSTATE for perdido – **TERRAFORM assume que os recursos não existem e tentará recriá-los (gerando mensagem de erro na implantação)**



Uma Estratégia Melhor

- Dividir seu código em múltiplos scripts costuma ser uma ótima opção:
 1. Melhora gerenciamento (ex: em caso de atualização do AzureRM)
 2. Melhora escalabilidade (ex: executar implementações em paralelo)
 3. Melhora reusabilidade (ex: aplicação em ambientes Dev/Tst/Prd)

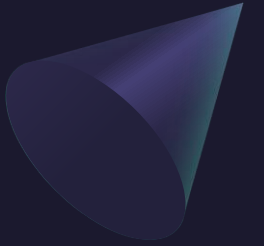


Terraform – Tipos de Scripts

- Na prática, é uma boa ideia dividir seu código entre vários scripts
- Seu projeto pode incluir (por exemplo):
 - **[main].tf** : script raiz; normalmente provedores e recursos a serem criados.
 - **variables.tf** : onde se define as variáveis, tipos de dados e comentários
 - **locals.tf** : permite criar expressões baseadas em variáveis (exemplo: criação de “tags”)
 - **output.tf** : coleta valores de atributos criados durante a implementação (ex: IDs).
 - **custom files** : arquivos genéricos para customização do seu código
 - **MÓDULOS** : bibliotecas que encapsulam a sintaxe de criação de cada recurso



Por Quê Usar Módulos

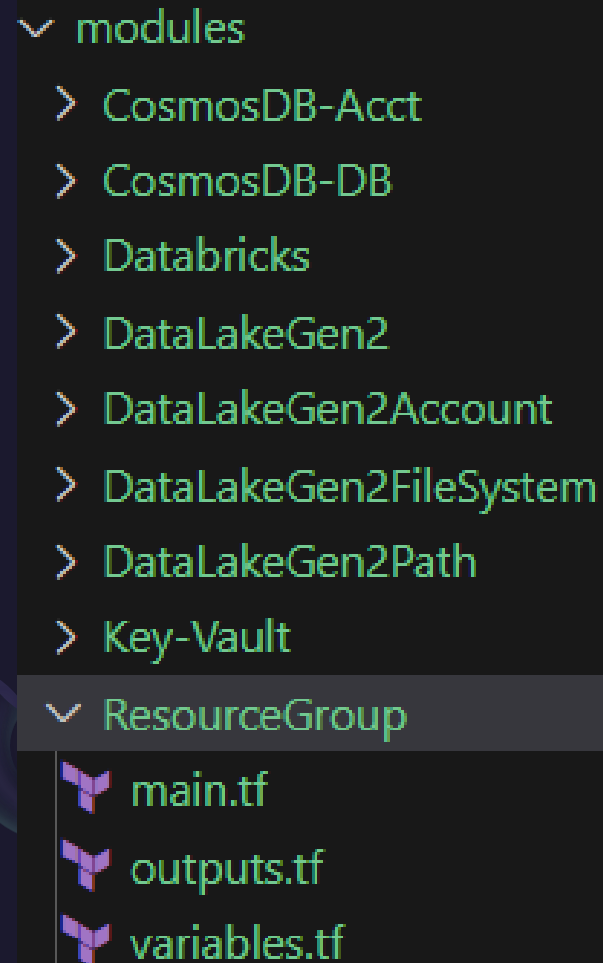


- Eles isolam a sintaxe correta de criação de cada recursos
- Essa biblioteca deve ser compartilhada com todo time de DEV
- Define um padrão corporativo
- Facilita gerenciamento de upgrades e mudanças de sintaxe relativos aos provedores



Organizando Scripts Para Usar Módulos

Crie um diretório de MÓDULOS



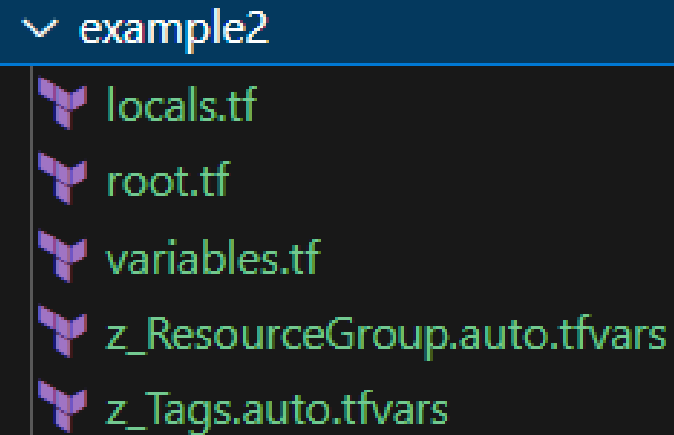
```

└─ modules
  ├── CosmosDB-Acct
  ├── CosmosDB-DB
  ├── Databricks
  ├── DataLakeGen2
  ├── DataLakeGen2Account
  ├── DataLakeGen2FileSystem
  ├── DataLakeGen2Path
  ├── Key-Vault
  └─ ResourceGroup
     ├── main.tf
     ├── outputs.tf
     └── variables.tf

```

A screenshot of a file explorer showing a directory structure for Terraform modules. The 'modules' directory is expanded, showing a list of sub-directories: CosmosDB-Acct, CosmosDB-DB, Databricks, DataLakeGen2, DataLakeGen2Account, DataLakeGen2FileSystem, DataLakeGen2Path, Key-Vault, and ResourceGroup. The 'ResourceGroup' directory is selected and expanded, showing three files: main.tf, outputs.tf, and variables.tf.

Crie diretórios de PROJETOS



```

└─ example2
  ├── locals.tf
  ├── root.tf
  ├── variables.tf
  ├── z_ResourceGroup.auto.tfvars
  └── z_Tags.auto.tfvars

```

A screenshot of a file explorer showing a directory structure for Terraform projects. The 'example2' directory is expanded, showing a list of files: locals.tf, root.tf, variables.tf, z_ResourceGroup.auto.tfvars, and z_Tags.auto.tfvars.



Conteúdo dos Scripts: Módulo

```
modules > ResourceGroup > main.tf
1  #reference https://registry.terraform.io/providers/h
2  resource "azurerm_resource_group" "ResourceGroup" {
3      name
4      location
5      tags
6  }
```

```
modules > ResourceGroup > outputs.tf
1  output "newResourceGroupName" {
2      value = azurerm_resource_group.ResourceGroup.name
3  }
4
5  output "id" {
6      value = azurerm_resource_group.ResourceGroup.id
7  }
8
9  #output "tenant_id" {
10     # value = azurerm_resource_group.ResourceGroup.tenant_id
11     #}
12
```

```
modules > ResourceGroup > variables.tf
1  variable "rg_name" {
2      type = string
3      description = "resource group name"
4  }
5
6  variable "location" {
7      type = string
8      description = "code related to Azure Region"
9  }
10
11
12  variable "tags" {
13      type = map(string)
14      description = "tags related to project"
15  }
```



Conteúdo dos Scripts: Main & Variables

example2 > root.tf


```
1  # Terraform and Azure Provider configuration
2  terraform {
3      required_providers {
4          azurerm = {
5              source = "hashicorp/azurerm"
6              version = "~>3.0"
7              ##any azurerm version, from 3.0.0 or above
8          }
9      }
10 }
11 provider "azurerm" {
12     features {}
13 }
14
15 module "ResourceGroup1" {
16     source = "../modules/ResourceGroup"
17     rg_name = local.rgname1
18     location = var.location1
19     tags = local.tag
```

example2 > variables.tf


```
33 # #####
34 # variables related to RESOURCE GROUP
35 # #####
36 variable "resourceIdentifier" {
37     type = string
38     description = "prefix to resource group name"
39 }
40
41 variable "location1" {
42     type = string
43     description = "code related to Azure Region"
44 }
45
46 variable "shortlocation1" {
47     type = string
48     description = "short code related to Azure Region"
49 }
```



Conteúdo dos Scripts: TFVARS & Locals

example2 >  z_Tags.auto.tfvars

```
1  company           = "WWI"
2  project            = "platdt"
3  costcenter         = "xyz"
4  environment        = "dev"
5  shortprovider      = "az"
6  resource_id        = "10"
7
```

example2 >  z_ResourceGroup.auto.tfvars

```
1
2  resourcegroup_identifier= "rg"
3  location                = "southcentralus"
4  shortlocation           = "ussc"
```

example2 >  locals.tf

```
1
2  locals {
3
4      rgname1 = format("%s-%s-%s-%s-01"
5          , var.shortprovider, var.shortlocation1
6          , var.environment, var.resourceIdentifier)
7
8      tag = {
9          company      = var.company
10         project       = "${var.company}-${var.project}"
11         costcenter    = var.costcenter
12         environment   = var.environment
13     }
14 }
```



Exemplo 2 : Usando Módulos

DEMO

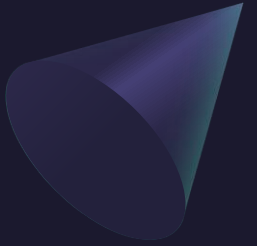


Múltiplos Recursos

- Atenção às convenções de nomenclatura
<https://bit.ly/azurenames>
- Esteja atento a nomes únicos
- Considere dependências entre recursos
- Lembre-se que seu script pode combinar o uso de múltiplos provedores



Sobre os Nomes Únicos



- No Azure, os recursos precisam ter nomes únicos dentro do seu escopo
- Alguns recursos tem escopo global, portanto você não tem controle sobre nomes
- Uma alternativa é adicionar um valor aleatório para gerar um nome único ([locals.tf](#))

```
locals.tf
1  #Random ID for unique naming
2  resource "random_integer" "rand" {
3      min = 000001
4      max = 999999
5  }
6
7  locals {
8      randomSuffix = "${format("%06s", random_integer.rand.result)}"
9      rgname = "${var.rgPrefix}${var.rgName}-${local.randomSuffix}"
10 }
```



Dependências Entre Recursos

- Muitos recursos dependem de um recurso “pai”
- Nesses casos, esteja atento a 2 pontos principais:
 - O recurso “filho” deve ser implementado DEPOIS do “pai”.
 - Garanta que o “pai” forneça as informações necessárias para definição do recurso “filho” (**outputs.tf**)

```
modules > SynapseWorkspace > outputs.tf
1  output "name" {
2    description = "The name of the resource created."
3    value       = azurerm_synapse_workspace.synapseworkspace.name
4  }
5
6  output "id" {
7    description = "The id of the resource created."
8    value       = azurerm_synapse_workspace.synapseworkspace.id
9  }
```

```
example3 > root.tf
230 module "synapsesqlpool" {
231   source              = "../modules/SynapseSQLPool"
232   synsqlpool_name     = var.synsqlpool_name
233   synsqlpool_wrkspcid = module.synapseworkspace.id
234   synsqlpool_sku      = var.synsqlpool_sku
235   synsqlpool_mode     = var.synsqlpool_mode
236
237   depends_on = [module.synapseworkspace]
238 }
239
```

Mensagens Exibidas

- Terraform apresentação mensagens de status durante a implementação (CMD)

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

module.synapseworkspace.azure_terraform_synapse_workspace.synapseworkspace: Still creating... [3m50s elapsed]
module.synapseworkspace.azure_terraform_synapse_workspace.synapseworkspace: Still creating... [4m0s elapsed]
module.synapseworkspace.azure_terraform_synapse_workspace.synapseworkspace: Still creating... [4m10s elapsed]
module.synapseworkspace.azure_terraform_synapse_workspace.synapseworkspace: Still creating... [4m20s elapsed]
module.synapseworkspace.azure_terraform_synapse_workspace.synapseworkspace: Still creating... [4m30s elapsed]
module.synapseworkspace.azure_terraform_synapse_workspace.synapseworkspace: Still creating... [4m40s elapsed]
module.synapseworkspace.azure_terraform_synapse_workspace.synapseworkspace: Creation complete after 4m49s [id=/subscriptions/56508de2-630e-4d64-a40f-45bfa674520b/resourceGroups/az-ussc-dev-rg-563645/providers/Microsoft.Synapse/workspaces/az-ussc-dev-syn-563645]
module.synapseworkspace.azure_terraform_synapse_workspace.synapseworkspace: Creating...
module.synapseworkspace.azure_terraform_synapse_workspace.synapseworkspace: Still creating... [10s elapsed]
module.synapseworkspace.azure_terraform_synapse_workspace.synapseworkspace: Still creating... [20s elapsed]
module.synapseworkspace.azure_terraform_synapse_workspace.synapseworkspace: Still creating... [30s elapsed]
module.synapseworkspace.azure_terraform_synapse_workspace.synapseworkspace: Still creating... [40s elapsed]
module.synapseworkspace.azure_terraform_synapse_workspace.synapseworkspace: Still creating... [50s elapsed]
module.synapseworkspace.azure_terraform_synapse_workspace.synapseworkspace: Still creating... [1m0s elapsed]
module.synapseworkspace.azure_terraform_synapse_workspace.synapseworkspace: Still creating... [1m10s elapsed]
module.synapseworkspace.azure_terraform_synapse_workspace.synapseworkspace: Still creating... [1m20s elapsed]
module.synapseworkspace.azure_terraform_synapse_workspace.synapseworkspace: Still creating... [1m30s elapsed]
module.synapseworkspace.azure_terraform_synapse_workspace.synapseworkspace: Still creating... [1m40s elapsed]
module.synapseworkspace.azure_terraform_synapse_workspace.synapseworkspace: Still creating... [1m50s elapsed]
module.synapseworkspace.azure_terraform_synapse_workspace.synapseworkspace: Still creating... [2m0s elapsed]
module.synapseworkspace.azure_terraform_synapse_workspace.synapseworkspace: Still creating... [2m10s elapsed]
module.synapseworkspace.azure_terraform_synapse_workspace.synapseworkspace: Still creating... [2m20s elapsed]
module.synapseworkspace.azure_terraform_synapse_workspace.synapseworkspace: Still creating... [2m30s elapsed]
module.synapseworkspace.azure_terraform_synapse_workspace.synapseworkspace: Still creating... [2m40s elapsed]
module.synapseworkspace.azure_terraform_synapse_workspace.synapseworkspace: Still creating... [2m50s elapsed]
module.synapseworkspace.azure_terraform_synapse_workspace.synapseworkspace: Creation complete after 2m54s [id=/subscriptions/56508de2-630e-4d64-a40f-45bfa674520b/resourceGroups/az-ussc-dev-rg-563645/providers/Microsoft.Synapse/workspaces/az-ussc-dev-syn-563645/sqlPools/whiterabbit]

Apply complete! Resources: 17 added, 0 changed, 0 destroyed.
PS C:\temp\Terraform101\example3>
```

Exemplo 3 : Múltiplos Recursos

DEMO





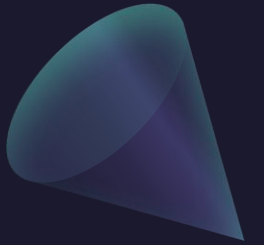
Resumo

- Terraform permite criar virtualmente qualquer recurso do Azure
- Linguagem do Terraform é bastante versátil e poderosa
- Princípios apresentados aqui também se aplicam a outros provedores
- Existem vários projetos open source que melhoram o Terraform
- EX: TFSEC & TRIVY (<https://github.com/aquasecurity/trivy#how-to-pronounce-the-name-trivy>)



Referências

- [GitHub – all files presented in this workshop:](https://github.com/wcrivelini/articles/tree/main/Azure_Terraform)
https://github.com/wcrivelini/articles/tree/main/Azure_Terraform
- [Database Deployment with Terraform - The Basics:](https://www.sqlservercentral.com/articles/database-deployment-with-terraform-the-basics)
<https://www.sqlservercentral.com/articles/database-deployment-with-terraform-the-basics>
- [Database Deployment with Terraform – Modules:](https://www.sqlservercentral.com/articles/database-deployment-with-terraform-modules)
<https://www.sqlservercentral.com/articles/database-deployment-with-terraform-modules>



Obrigado

Wagner Crivelini

email

wagner.crivelini@microsoft.com

Linkedin

<https://www.linkedin.com/in/wagner-crivelini/>

