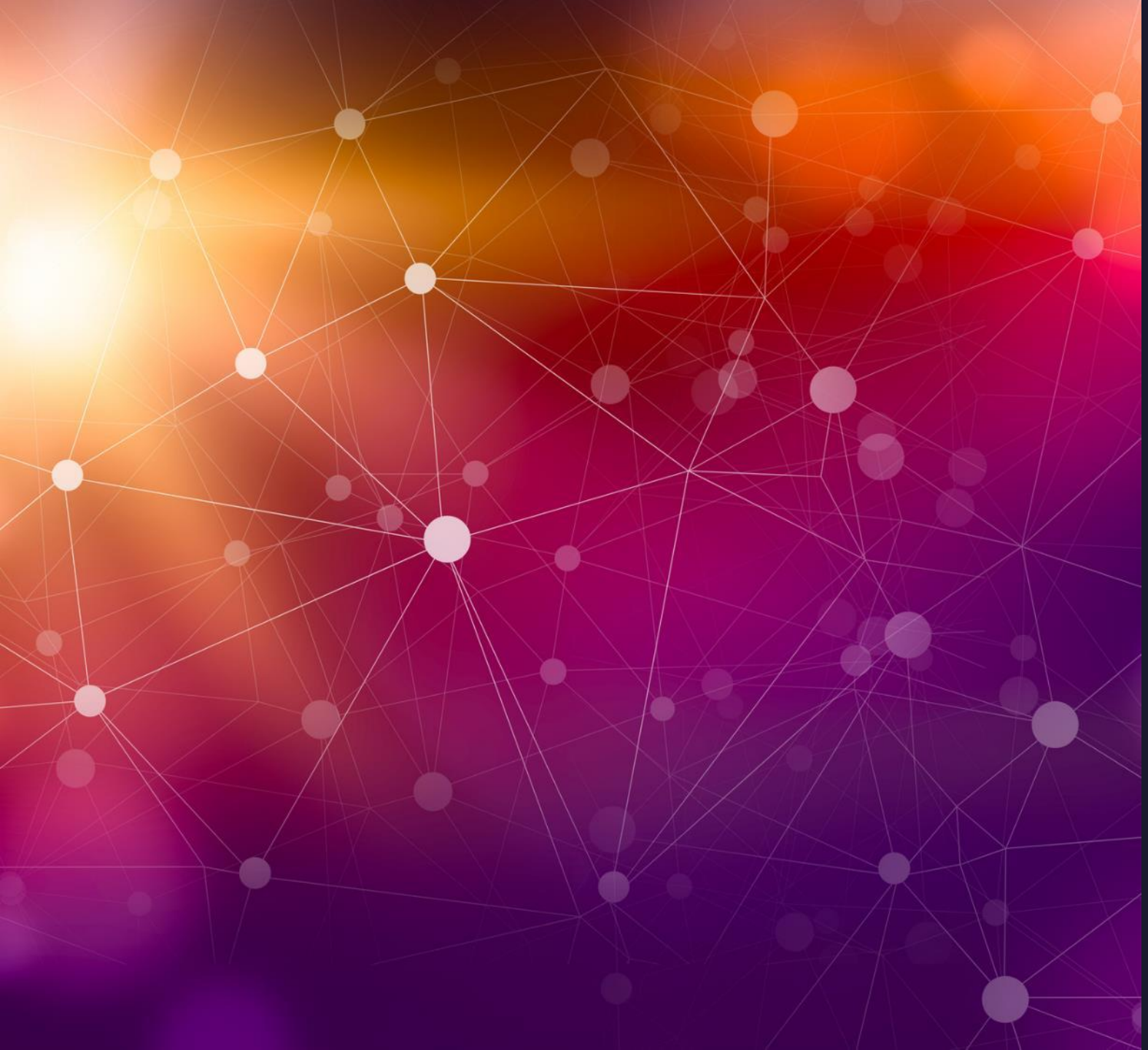


Deploying Azure Data Resources with Terraform

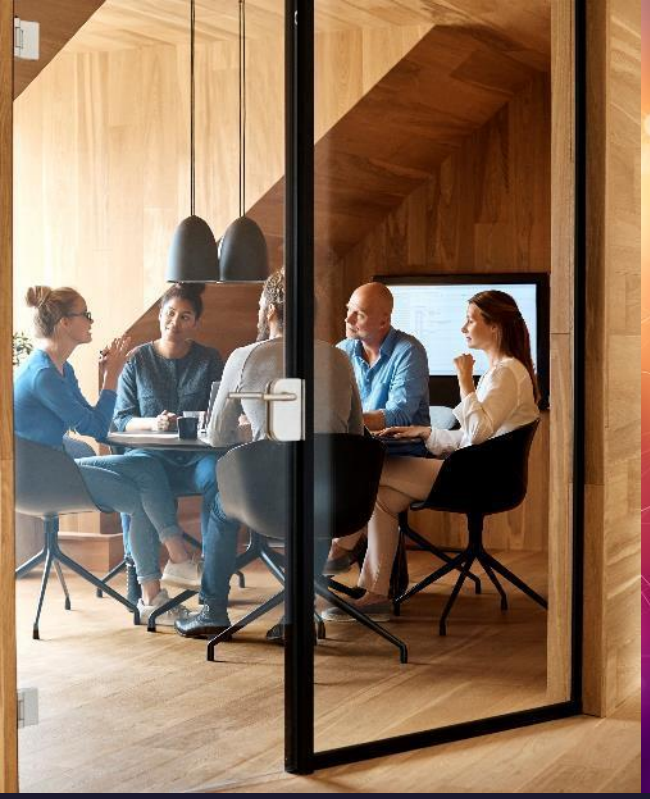
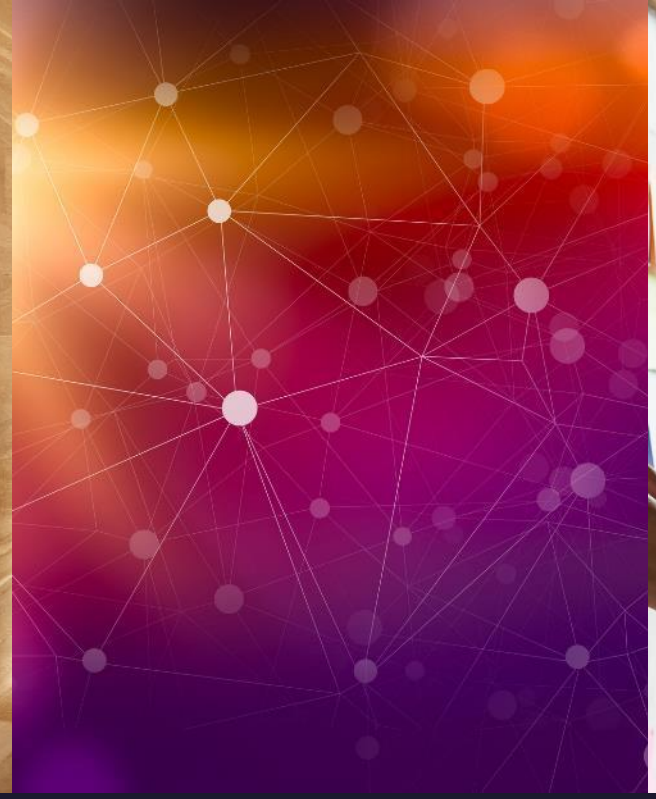
Wagner Crivelini

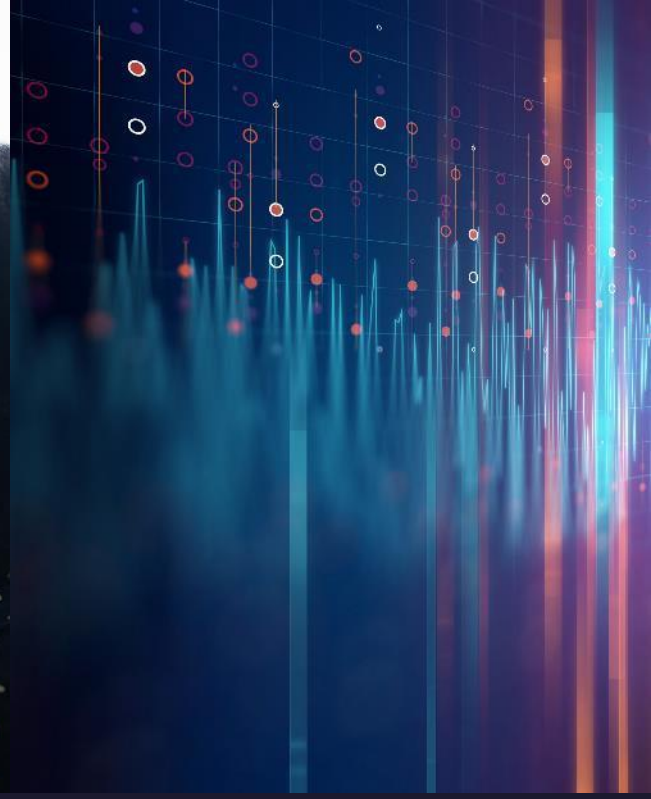
Speaker Bio:

Wagner Crivelini

- Senior Consultant at Microsoft Brazil
- Data Engineer
- Columnist in several tech portals with +250 publications

Agenda



Terraform Introduction

- Infrastructure as Code is now mainstream
- Terraform (HashiCorp) is growing fast, because it is:
  1. Declarative (avoid excessive details)
  2. Idempotent (consistency)
  3. Agnostic (works with major cloud providers)
  4. Open source (community contributes)

Terraform – Getting Started

- You can install it locally and run with VS CODE / CMD

  https://www.terraform.io/downloads

- Basic commands:

```
C:\TEMP\terraform init
C:\TEMP\terraform validate   #your scripts
C:\TEMP\terraform plan       #plan execution of your scripts
C:\TEMP\terraform apply      #execute your scripts
C:\TEMP\terraform destroy    #in case you want to rollback
C:\TEMP\terraform show       #show results of your deployment
```

Terraform – Scripts

- Use HCL language or JSON
- Choose your cloud provider (for Azure, use AzureRM)

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

- Inform values for parameters of the chosen resources to be created/updated
- Use variables instead of hard-coded values
- Design your scripts to be readable (a single script vs multiple scripts)
- When deploying multiple objects, deployments run in parallel

Terraform – Basic Syntax

- Scripts must begin declaring the providers to use and their versions.

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

- Identify parameters of the resources to be deployed (ex: https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/resource_group)

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

Terraform – How to Run Your Scripts

- How to validate and execute your scritps

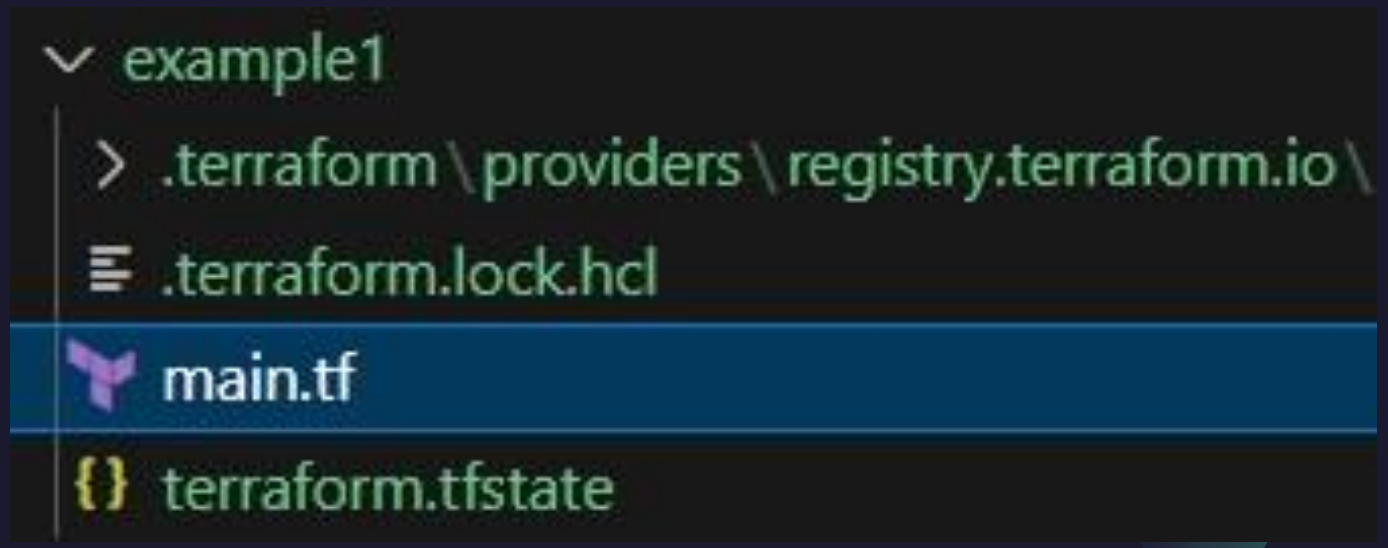
```
cd <path_to_scripts>
az login
rem to deploy to specific subscription run next command
az account set --subscription <subscriptionID>
terraform init
terraform validate
terraform plan  [-out]  <repositório/arquivoPlan.tfplan>
terraform apply [-auto-approve] [<repositório/arquivoPlan.tfplan>]
```

Example 1 : Deploying a Resource Group

DEMO

State Control

- New files will appear in the folder first time you run your scripts
- The TFSTATE file controls the state of your infrastructure
- Next time you run your scripts, Terraform will check state first.
- NEVER LOSE/ CHANGE TFSTATE



To Change or Delete Resources

- TFSTATE file is key for successful deployments
- Results expected when you run your scripts again:
    1. If your scripts didn't change - nothing is deployed
    2. If you changed values for the variables - resource will be updated
    3. If you remove a resource from script – resource will be deleted
    4. If you lose TFSTATE – it is assumed resource doesn't exist (you might get error)

Better Strategy for Scripts

- Splitting your script into several ones is a better way:
  1. Improve manageability (ex: handling upgrades in AzureRM)
  2. Improve scalability (ex: deploy several resources of same type)
  3. Improve reusability (ex: across environments)

Terraform – Types of Scripts

- In real world, it is good practice to split your code into several scripts
- Your project might include:
  - [main].tf : "root" file; may contain providers and resources to be deployed.
  - variables.tf : where you attribute values to your variables
  - locals.tf : allows to create expressions based on variables (example: creating tags)
  - output.tf : allows to collect values generated during deployment, such as IDs.
  - custom files : you use to better organize your project
  - MODULES : library of reusable code that encapsulate the logic of resources

Why Use Modules

- Isolate resource syntax necessary during deployment
- Define a shared library within the entire team of developers
- Define a standard within the company
- Make it easier to manage upgrades and syntax changes in the cloud provider

Organizing Scripts to Use Modules

Create a shared folder for MODULES

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

Create a PROJECT folder

- example2
  - locals.tf
  - root.tf
  - variables.tf
  - z_ResourceGroup.auto.tfvars
  - z_Tags.auto.tfvars

Scripts' Content: Module

modules > ResourceGroup > main.tf

```
#reference https://registry.terraform.io/providers/h
resource "azurerm_resource_group" "ResourceGroup" {
    name                    = var.rg_name
    location                = var.location
    tags                    = var.tags
}
```

modules > ResourceGroup > variables.tf

```
variable "rg_name" {
    type  = string
    description = "resource group name"
}

variable "location" {
    type  = string
    description = "code related to Azure Region"
}


variable "tags" {
    type  = map(string)
    description = "tags related to project"
}
```

modules > ResourceGroup > outputs.tf

```
output "newResourceGroupName" {
  value = azurerm_resource_group.ResourceGroup.name
}

output "id" {
  value = azurerm_resource_group.ResourceGroup.id
}

#output "tenant_id" {
#  value = azurerm_resource_group.ResourceGroup.tenant_id
#}

```

Scripts' Content: Main & Variables

example2 > root.tf

```
# Terraform and Azure Provider configuration
terraform {
  required_providers {
    azurerm = {
      source = "hashicorp/azurerm"
      version = "~>3.0"
      ##any azurerm version, from 3.0.0 or above
    }
  }
}
provider "azurerm" {
  features {}
}

module "ResourceGroup1" {
  source      = "../modules/ResourceGroup"
  rg_name     = local.rgname1
  location    = var.location1
  tags        = local.tag
```

example2 > variables.tf

```
# ##########################################################
# variables related to RESOURCE GROUP
# ##########################################################
variable "resourceIdentifier" {
    type  = string
    description = "preffix to resource group name"
}

variable "location1" {
    type  = string
    description = "code related to Azure Region"
}

variable "shortlocation1" {
    type  = string
    description = "short code related to Azure Region"
}
```

Scripts' Content: TFVARS & Locals

example2 > z_Tags.auto.tfvars

```
company                 = "WWI"
project                 = "platdt"
costcenter              = "xyz"
environment             = "dev"
shortprovider           = "az"
resource_id             = "10"

```

example2 > z_ResourceGroup.auto.tfvars

```

resourcegroup_identifier= "rg"
location                = "southcentralus"
shortlocation           = "ussc"
```

example2 > locals.tf

```

locals {

  rgname1 = format("%s-%s-%s-%s-01"
      , var.shortprovider, var.shortlocation1
      , var.environment, var.resourceIdentifier)

  tag = {
    company       = var.company
    project       = "${var.company}-${var.project}"
    costcenter    = var.costcenter
    environment   = var.environment
  }
}
```

Example 2 : Using Modules

DEMO

Multiple Resources

- Attention to naming conventions

  https://bit.ly/azurenames

- Beware of unique names requested
- Consider dependencies
- You can combine DIFFERENT providers

About Unique Names

- You must assure your resources' names are unique within their scope
- Some resources are scoped to work globally, and you have no control over names
- One alternative is to add a random number to your unique name (locals.tf)

locals.tf

```
#Random ID for unique naming
resource "random_integer" "rand" {
  min = 000001
  max = 999999
}

locals {
  randomSuffix = "${format("%06s", random_integer.rand.result)}"
  rgname = "${var.rgPreffix}${var.rgName}-${local.randomSuffix}"
}
```

Resource Dependencies

- Several objects depend on the creation of a "parent" object
- In these cases, we have 2 major concerns:
  - The "child" resource must be deployed AFTER the parent
  - Make sure the "parent" resource will output the necessary info to child

modules > SynapseWorkspace > outputs.tf

```
output "name" {
  description = "The name of the resoruce created."
  value       = azurerm_synapse_workspace.synapsewokspace.name
}

output "id" {
  description = "The id of the resource created."
  value       = azurerm_synapse_workspace.synapsewokspace.id
}
```

example3 > root.tf

```
module "synapsesqlpool" {
  source                                  = "../modules/SynapseSQLPool"
  synsqlpool_name                         = var.synsqlpool_name
  synsqlpool_wrkspcid                     = module.synapseworkspace.id
  synsqlpool_sku                          = var.synsqlpool_sku
  synsqlpool_mode                         = var.synsqlpool_mode

  depends_on = [module.synapseworkspace]
}
```

Deployment Messages

- Terraform shows you information during deployment

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

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

Example 3 : Deploying Multiple Resources

DEMO

Summary

- You can deploy virtually any Azure resource with Terraform
- Terraform's HCL is very powerful
- Principles presented here apply to any other cloud provider
- There are several open source projects
- EX: TFSEC & TRIVY (https://github.com/aquasecurity/trivy#how-to-pronounce-the-name-trivy)

Source References

- GitHub – all files presented in this workshop:
  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
- Database Deployment with Terraform – Modules:
  https://www.sqlservercentral.com/articles/database-deployment-with-terraform-modules

Thank You

Wagner Crivelini

email
wagner.crivelini@microsoft.com

Linkedin
https://www.linkedin.com/in/wagner-crivelini-107850/