Working with Modules

Modules are containers for multiple resources that are used together. A module consists of a collection of **.tf** and/or .**tf.json** files kept together in a directory.

Modules are the main way to package and reuse resource configurations with Terraform.

**Module Types:**

* **Root Module:** Every Terraform configuration has at least one module, known as its **root module**, which consists of the resources defined in the .tf files in the main working directory.
* **Child Modules:** A Terraform module can call other modules to include their resources into the configuration. A module that has been called by another module is often referred to as a child **module.Child** modules can be called multiple times within the same configuration, and multiple configurations can use the same child module.
* **Published Modules:** In addition to modules from the local filesystem, Terraform can load modules from a **public** **or** **private registry**. This makes it possible to publish modules for others to use, and to use modules that others have published.

Note: The **Terraform Registry** hosts a broad collection of publicly available Terraform modules for configuring many kinds of common infrastructure. These modules are free to use, and Terraform can download them automatically if you specify the appropriate source and version in a module call block.

**Calling a Child Module**

Write the following in Root Module

**module** "**servers**" {

**source** = "./app-cluster"

servers = 5

}

* The label immediately after the module keyword is a local name, which the calling module can use to refer to this instance of the module.
* The **source** argument is mandatory for all modules.

**Meta-Argument**

* **count** - Creates multiple instances of a module from a single module block.
* **for\_each** - Creates multiple instances of a module from a single module block.
* [**depends\_on**](https://developer.hashicorp.com/terraform/language/modules/syntax#depends_on) - Creates explicit dependencies between the entire module and the listed targets.

**Note: A module containing its own provider configurations is not compatible with the for\_each, count, and depends\_on arguments.**

**Accessing Module Output Values:**

The resources defined in a module are encapsulated, so the calling module cannot access their attributes directly. However, the child module can declare **output values** to selectively export certain values to be accessed by the calling module.

resource "azurerm" "example" {

# ...

instances = **module.servers.instance\_ids**

}

**Invoking Child Module with For Each**

# my\_buckets.tf

module "bucket" {

for\_each = toset(["assets", "media"])

source = "./publish\_bucket"

name = "${each.key}\_bucket"

}

Example in Azure

**./create-vm/variables.tf**

variable "resource\_group\_name" {

type = string

nullable = false

default = "Terraform-rg"

}

variable "resource\_group\_location" {

type = string

default = "eastus"

}

**./create-vm/vm.tf**

**# Create virtual network**

resource "azurerm\_virtual\_network" "myNetwork" {

name = "myVnet"

address\_space = ["10.1.0.0/16"]

location = **var.resource\_group\_location**

resource\_group\_name = **var.resource\_group\_name**

}

**# Create subnet**

resource "azurerm\_subnet" "mySubnet" {

name = "mySubnet"

resource\_group\_name = var.resource\_group\_name

virtual\_network\_name = azurerm\_virtual\_network.myNetwork.name

address\_prefixes = ["10.1.1.0/24"]

}

**# Create public IPs**

resource "azurerm\_public\_ip" "myVMPublicIPs" {

count = 2

name = format("myVM%s-ip", count.index)

location = var.resource\_group\_location

resource\_group\_name = var.resource\_group\_name

allocation\_method = "Dynamic"

}

**# Create Network Security Group and rule**

resource "azurerm\_network\_security\_group" "myNSG" {

name = "myNetworkSecurityGroup"

location = var.resource\_group\_location

resource\_group\_name = var.resource\_group\_name

security\_rule {

name = "SSH"

priority = 1001

direction = "Inbound"

access = "Allow"

protocol = "Tcp"

source\_port\_range = "\*"

destination\_port\_range = "22"

source\_address\_prefix = "\*"

destination\_address\_prefix = "\*"

}

security\_rule {

name = "HTTP"

priority = 1000

direction = "Inbound"

access = "Allow"

protocol = "Tcp"

source\_port\_range = "\*"

destination\_port\_range = "80"

source\_address\_prefix = "\*"

destination\_address\_prefix = "\*"

}

}

**# Create network interface**

resource "azurerm\_network\_interface" "myNICs" {

count = 2

name = format("myVM%s-nic", count.index)

location = var.resource\_group\_location

resource\_group\_name = var.resource\_group\_name

ip\_configuration {

name = "myNIC"

subnet\_id = azurerm\_subnet.mySubnet.id

private\_ip\_address\_allocation = "Dynamic"

public\_ip\_address\_id = azurerm\_public\_ip.myVMPublicIPs[count.index].id

}

}

**# Connect the security group to the network interface**

resource "azurerm\_network\_interface\_security\_group\_association" "example" {

count = 2

network\_interface\_id = azurerm\_network\_interface.myNICs[count.index].id

network\_security\_group\_id = azurerm\_network\_security\_group.myNSG.id

}

**# Create (and display) an SSH key**

resource "tls\_private\_key" "mySSH" {

algorithm = "RSA"

rsa\_bits = 4096

}

resource "local\_file" "private\_key" {

content = tls\_private\_key.mySSH.private\_key\_pem

filename = "azure.pem"

file\_permission = "0600"

}

**# Create virtual machine**

resource "azurerm\_linux\_virtual\_machine" "my\_terraform\_vm" {

count = 2

name = format("myVM%s-vm", count.index)

location = var.resource\_group\_location

resource\_group\_name = var.resource\_group\_name

network\_interface\_ids = [azurerm\_network\_interface.myNICs[count.index].id]

size = "Standard\_DS1\_v2"

os\_disk {

name = format("myVM%s-OsDisk",count.index)

caching = "ReadWrite"

storage\_account\_type = "Premium\_LRS"

}

source\_image\_reference {

publisher = "Canonical"

offer = "UbuntuServer"

sku = "18.04-LTS"

version = "latest"

}

computer\_name = "myvm"

admin\_username = "azureuser"

admin\_password = "Password@123"

disable\_password\_authentication = false

admin\_ssh\_key {

username = "azureuser"

public\_key = tls\_private\_key.mySSH.public\_key\_openssh

}

depends\_on = [

azurerm\_network\_interface\_security\_group\_association.example

]

}

**output "VMIPAddress"** {

value = azurerm\_public\_ip.myVMPublicIPs.\*.ip\_address

depends\_on = [

azurerm\_linux\_virtual\_machine.my\_terraform\_vm

]

}

**./main.tf**

terraform {

required\_providers {

azurerm = {

source = "hashicorp/azurerm"

version = "3.11.0"

}

}

}

provider "azurerm" {

features {}

}

**#Creating a Resource Group**

resource "azurerm\_resource\_group" "rg" {

name = "Module-rg"

location = "eastus"

}

**module "create-vm" {**

**source** = "./create-vm"

**resource\_group\_name** = azurerm\_resource\_group.rg.name

**resource\_group\_location** = azurerm\_resource\_group.rg.location

**}**

**output** vm\_public\_ip {

value = **module.create-vm.VMIPAddress**

}

**Terraform Commands**

* terraform init
* terraform plan -out=tfplan
* terraform apply tfplan --auto-approve

Using GitHub for Modules

1. Create a New Repository in GitHub.com
2. On local machine execute the following
   1. cd create-vm
   2. git init
   3. git add .
   4. git commit -m "main.tf"
   5. git remote add origin https://github.com/sandeepsonihyd/terraformdemo.git
   6. git branch -M main
   7. git push -u origin main
3. Using Module in Git in **ROOT** Module

module "create-vm" {

source = "github.com/sandeepsonihyd/terraformdemo"

resource\_group\_name = azurerm\_resource\_group.rg.name

resource\_group\_location = azurerm\_resource\_group.rg.location

}

output vm\_public\_ip {

value = module.create-vm.VMIPAddress

}

**Terraform Commands**

* terraform init
* terraform plan -out=tfplan
* terraform apply tfplan --auto-approve

If the repository is private, you would be required to Generate PAT Token and login with below credentials:

Username: sandeepsonihyd

Password: ghp\_qZaIs6Vxu2n4lKESgRxMb7OmqiMQ1X1qx42G