Terraform on Azure – Lab 2

In this lab you will learn how to create modules and use them as an approach for easier management and (re)usability of infrastructure created with Terraform

1. Since your main.tf has a lot of lines of code, you can imagine that adding any additional resource would make this confirugation much harder to maintain. The approach you are going to use now is using modules. In your terminal(conosle), if not already, navigate to terraform folder using the folowing command:

```
cd c:/terraform
```

2. Once in the terraform folder, you are going to create a new folder called modules, that will have all the modules needed for you infrastructure deplyoments on Azure. Create a new folder called modules and navigate to it:

```
mkdir modules
cd modules
```

3. After creating and navigating to modules folder, it is time to create a folder for each of the modules that we are going to have, and those are: randomizer, resource_group, network, compute, storage, and analytics. To do so create folder using the following commands:

```
mkdir randomizer
mkdir resource_group
mkdir network
mkdir compute
mkdir storage
mkdir analytics
```

4. The main idea of modularization is to have as less code as possible in the *root* main.tf configuration file, and to delegate all the configurations to the appropriate modules. That being said, every module will have it's own terraform configuration file as well as its own variables that are constantly going to be reused in these configuration files (e.g. resource group name and location). Next thing you want to do is to create those main.tffiles for each and every module, accompanied with variables.tf file. These files will be empty for start, but you will add some configuration to them later. To create those files in respective folders, use the following commands:

```
cd c:/terraform/modules/analytics
cd .> main.tf
cd .> variables.tf
cd ../storage
cd .> main.tf
cd .> output.tf
cd .> variables.tf
cd ../compute
cd .> main.tf
cd .> variables.tf
cd ../randomizer
cd .> main.tf
cd .> output.tf
```

5. You've probably noticed that you haven't created any files for resource_group and network module. The only reason for that is so it comes to your attention that we are going to create output.tf file since we will want to use some their output values as a variable in other modules (hence creating variables.tf for other modules).

6. Let's create configuration files for network first by using the set of following commands:

```
cd ../network
cd .> main.tf
cd .> variables.tf
cd .> output.tf
```

7. Now you only need to create files for network. Notice that you are not going to create variables.tf files, since you won't need them. However, we will create output.tf since we are going to use resource group name and location all throughout the modules. To do so, use the following set of commands:

```
cd ../resource_group
cd .> main.tf
cd .> output.tf
```

8. Since you have the file structure set correctly, it is time to configure the modules properly and offload the root configuration file created in the previous lab.

You are going to transfer bits and pieces of code all through out the modules. You will start with resource_group module by navigating to modules/resource_group folder and open and edit main.tf by appending the following code:

```
# Create the resource group in Azure
resource "azurerm_resource_group" "rg" {
  name     = "rgTerraform"
  location = "UK South"
}
```

9. Save main.tf and configure output.tf file by appending the folloqing code which will add create variables that are going to be reused by other modules.

```
output "rg_name" {
  value = azurerm_resource_group.rg.name
}

output "rg_location" {
  value = azurerm_resource_group.rg.location
}

output "rg_id" {
  value = azurerm_resource_group.rg.id
}
```

10. After saving the output.tf for a resource_gruop module, it is time to move on to randomizer module in modules/randomizer folder where you will open the main.tf file and append the following code that will create objects needed for producing random values:

```
resource "random_id" "random_id" {
  byte_length = 8
}

resource "random_pet" "ssh_key_name" {
  prefix    = "ssh"
  separator = ""
```

```
}
```

You will also need to define outputs in output.tf file where you are going to append the following:

```
output "ssh_id" {
  value = random_pet.ssh_key_name.id
}
output "random_id" {
  value = random_id.random_id.hex
}
```

11. After saving the files from randomizer module, next module that you are going to configure is the network module. To do so, navigate to modules/network folder and open main.tf file where you will define a VNET, subnet, private IP address, and network security group with a security rule. To do so, apply the following code:

```
# Create a virtual network in a resource group
resource "azurerm_virtual_network" "vnet" {
  name                = "vnetVM1"
  address_space       = ["10.0.0.0/16"]
  location            = var.rg_location
  resource_group_name = var.rg_name
}

# Create a subnet in a vnet
resource "azurerm_subnet" "subnet1" {
  name                 = "internal"
  resource_group_name  = var.rg_name
  virtual_network_name = azurerm_virtual_network.vnet.name
  address_prefixes     = ["10.0.2.0/24"]
}

# Create public IPs
resource "azurerm_public_ip" "my_public_ip" {
  name                = "myPublicIP"
  location            = var.rg_location
  resource_group_name = var.rg_name
```

```
  allocation_method   = "Dynamic"
}

# Create Network Security Group and rule
resource "azurerm_network_security_group" "my_nsg" {
  name                = "myNetworkSecurityGroup"
  location            = var.rg_location
  resource_group_name = var.rg_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 = "*"
  }
}
```

12. After configuring and saving main.tf networking part, it is time to navigate to variables.tf and enter variables that are used throughout main.tf. To do so, append the following code in variables.tf:

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

13. Since you are going to use a subnet id in further infrastructure definition, you will navigate to the output.tf file and append the code that will make subnet_id available in other variables:

```
output "subnet_id" {
  value = azurerm_subnet.subnet1.id
}
output "public_ip" {
  value = azurerm_public_ip.my_public_ip.id
}
output "nsg_id" {
  value = azurerm_network_security_group.my_nsg.id
}
```

18. Now it is time to define compute module, and to do so, navigate to modules/compute folder, open the main.tf, and append the following code where you are going to create a ssh key to connect to your Linux VM, network interface for your VM(and connect yor NIC with NSG). You may notice that something is bit different in this case. We add the block with the azapi provider again, and that is because Terraform handles providers as part of hashicorp namespace by default, and all the other modules must be specified inside the modules they are being used. The complete configuration code for compute is as follows:

```
terraform {
  required_providers {
    azapi = {
      source  = "azure/azapi"
      version = "~>1.5"
    }
  }
}
# Create SSH key
resource "azapi_resource_action" "ssh_public_key_gen" {
  type        = "Microsoft.Compute/sshPublicKeys@2022-11-01"
  resource_id = azapi_resource.ssh_public_key.id
  action      = "generateKeyPair"
  method      = "POST"
```

```
  response_export_values = ["publicKey", "privateKey"]
}

resource "azapi_resource" "ssh_public_key" {
  type      = "Microsoft.Compute/sshPublicKeys@2022-11-01"
  name      = var.ssh_id
  location  = var.rg_location
  parent_id = var.rg_id
}

output "key_data" {
  value = jsondecode(azapi_resource_action.ssh_public_key_gen.output).publicKey
}

# Create a network interface that is going to be used for a VM
resource "azurerm_network_interface" "nic1" {
  name                = "vm-nic"
  location            = var.rg_location
  resource_group_name = var.rg_name

  ip_configuration {
    name                          = "internal"
    subnet_id                     = var.subnet_id
    private_ip_address_allocation = "Dynamic"
    public_ip_address_id          = var.public_ip
  }
}

# Connect the security group to the network interface
resource "azurerm_network_interface_security_group_association" "nsg_to_nic" {
  network_interface_id      = azurerm_network_interface.nic1.id
  network_security_group_id = azurerm_network_security_group.my_nsg.id
}
# Create a virtual machine based on Ubuntu Linux distribution
resource "azurerm_linux_virtual_machine" "vm1" {
```

```
 name                = "vmlinux"
 resource_group_name = var.rg_name
 location            = var.rg_location
 size                = "Standard_F2"
 admin_username      = "adminuser"
 network_interface_ids = [
  azurerm_network_interface.nic1.id,
 ]

 admin_ssh_key {
  username   = "adminuser"
  public_key = jsondecode(azapi_resource_action.ssh_public_key_gen.output).publicKey
 }

 os_disk {
  caching              = "ReadWrite"
  storage_account_type = "Standard_LRS"
 }

 source_image_reference {
  publisher = "Canonical"
  offer     = "0001-com-ubuntu-server-jammy"
  sku       = "22_04-lts"
  version   = "latest"
 }
}
```

19. Save the main.tf file and focus on variables. Since you are going to use a couple of variables when defining compute resources, you will append the following values into variables.tf file:

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

```
variable "rg_location" {
  description = "Azure resource group location"
  type        = string
}

variable "rg_id" {
  description = "Resource group ID"
  type        = string
}

variable "subnet_id"{
  description = "VNETs subnet ID"
  type        = string
}
variable "ssh_id"{
  description = "ID of generated SSH"
  type        = string
}

variable "public_ip"{
  description = "Public IP ID"
  type        = string
}
```

20. With only two modules to go, the next step is to set up configuration for storage, and that's why you will navigate to modules/storage folder, and in main.tf configuration file, you are going to create a simple blob storage with a container, and Azure Data Lake Gen2 for Azure Synapse Analytics.

```
# Create a new Azure storage account
resource "azurerm_storage_account" "blobstorage" {
  name                     = "sablob${var.random_id}"
  resource_group_name      = var.rg_name
  location                 = var.rg_location
  account_tier             = "Standard"
  account_replication_type = "LRS"
}
```

```
# Create a container in storage account
resource "azurerm_storage_container" "workshopcontainer" {
  name                  = "workshop"
  storage_account_name  = azurerm_storage_account.blobstorage.name
  container_access_type = "blob"
}



# Create Azure Synapse Analytics workspace
resource "azurerm_storage_account" "synapsestorage" {
  name                     = "sasyn${var.random_id}"
  resource_group_name      = var.rg_name
  location                 = var.rg_location
  account_tier             = "Standard"
  account_replication_type = "LRS"
  account_kind             = "StorageV2"
  is_hns_enabled           = "true"
}

resource "azurerm_storage_data_lake_gen2_filesystem" "synapsedatalake" {
  name               = "adls"
  storage_account_id = azurerm_storage_account.synapsestorage.id
}
```

Since there you are using variables for storage you are going to append the following code to variables.tf file...

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

... and append the following the outputs that are going to be used by Azure Synapse Analytics:

```
output "synapsedl_id" {
  value = azurerm_storage_data_lake_gen2_filesystem.synapsedatalake.id
}
```

21. One last type of resource to be configured is Azure Synapse Analytics and to do so, you will navigate to modules/analytics folder append the following code in main.tf file:

```
# Create Azure Synapse Analytics workspace
resource "azurerm_synapse_workspace" "synapse" {
  name                                 = "asaworkshop"
  resource_group_name                  = var.rg_name
  location                             = var.rg_location
  storage_data_lake_gen2_filesystem_id = var.synapsedl_id
  sql_administrator_login              = "sqladminuser"
  sql_administrator_login_password     = "B0ET3rr@F!"

  identity {
    type = "SystemAssigned"
  }
}
```

Since, you are going to use variables, don't forget to append the following into variables.tf file:

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

variable "synapsedl_id" {
  description = "Azure resource group location"
  type        = string
}
```

22. Now when you've got all the modules configured properly, you need to clean up the main.tf configuration file in the root folder and instruct it to use the modules you've created. To do so, navigate back to your root folder (should be in C:/Terraform if you've have followed the instructions) and open main.tf file. The first thing you need to do is to get rid of all the unnecssary code, and delete everything *after* configuration of the Microsoft Azure provider. That being said, your code in main.tf should look like this:

```
# Use the required_providers block to set the
# Azure Provider source and version being used
terraform {
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = ">=3.44.0"
    }
     # Add provider for random values
    random = {
      source  = "hashicorp/random"
      version = "~>3.0"
    }
    # Add provider for Azure Resource Manager
    azapi = {
      source  = "azure/azapi"
      version = "~>1.5"
    }
  }
}

# Configure the Microsoft Azure Provider
provider "azurerm" {
  features {}

  client_id       = var.client_id
  client_secret   = var.client_secret
  tenant_id       = var.tenant_id
  subscription_id = var.subscription_id
```

```
}
```

23. To properly reference all the modules you've created, you will need to reference all of the modules. To do so, append the following code to the main.tf file:

```
module "randomizer"{
  source="./modules/randomizer"
}

module "resource_group"{
  source="./modules/resource_group"
}

module "storage"{
  source="./modules/storage"
 rg_name= module.resource_group.rg_name
 rg_location = module.resource_group.rg_location
 random_id = module.randomizer.random_id
}

module "network"{
  source="./modules/network"
  rg_name= module.resource_group.rg_name
  rg_location = module.resource_group.rg_location
}

module "compute"{
  source="./modules/compute"
  rg_name= module.resource_group.rg_name
  rg_location = module.resource_group.rg_location
  rg_id = module.resource_group.rg_id
  subnet_id = module.network.subnet_id
  ssh_id = module.randomizer.ssh_id
  public_ip = module.network.public_ip
  nsg_id = module.network.nsg_id
```

```
}


module "analytics"{
  source="./modules/analytics"
  rg_name= module.resource_group.rg_name
  rg_location = module.resource_group.rg_location
  synapsedl_id = module.storage.synapsedl_id
}
```

Notice that, wherever an output value from some other module is used, it needs to be referenced properly in module reference inside the main.tf file.

24. It is time to create a terraform plan for this lab and apply it by using the following commands in your terminal(console), but first you are going to need to run terraform init because you have some new modules:

```
terraform init


terraform plan -var-file=secret.tfvars -out lab2.tfplan


terraform apply lab2.tfplan
```

25. To facilitate the new approach of using modules, add another VNET in network module. To do so, append the following code to the main.tf file in modules/network folder:

```
# Create another virtual network
resource "azurerm_virtual_network" "second_vnet" {
  name                = "vnetLAB2"
  address_space       = ["10.1.0.0/16"]
  location            = var.rg_location
  resource_group_name = var.rg_name
}
```

and run the commands in your terminal(console) one more time:

```
terraform plan -var-file=secret.tfvars -out lab2.tfplan


terraform apply lab2.tfplan
```

You have successfuly completed Lab 2! Congratulations!