Terraform on Azure – Lab 1

In this lab you are going to use Terraform to create various resources on your Azure subscription.

Log in to Azure and create Azure Service Principal

1. Open your terminal or command prompt (you are NOT going to use Powershell during this lab)
2. Navigate to the root folder of your disk using the following command

```
cd C:
```

3. Create a new folder you are going to be working in, and navigate to it using the following commands:

```
mkdir terraform
cd terraform
```

4. Using terminal, log in to azure portal using the credentials provided. *IMPORTANT* when logging in to Azure, choose option that skips enabling MFA for the account provided:

```
az login
```

5. Once logged in, use the following command to list your subscription:

```
az account list
```

6. Look for the value of *id* and be sure to write it down somewhere since you are going to need it. Now you will run the command that will create a service principal which will have permission to create and manage resources on Azure:

```
az ad sp create-for-rbac --role="Contributor" --scopes="/subscriptions/XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"
```

Of course, instead of *XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX* you are going to use your own subscription ID that you can find in when logged in

7. You are going to get an output similar to this one:

```
{
  "appId": "00000000-0000-0000-0000-000000000000",
  "displayName": "azure-cli-2024-02-22-10-41-15",
  "name": "http://azure-cli-2024-02-22-10-41-15",
  "password": "0000-0000-0000-0000-000000000000",
  "tenant": "00000000-0000-0000-0000-000000000000"
}
```

You are going to need these values for Terraform Azure resource provider. Make sure that you write these values down.

Create infrastructre in Azure using Terraform

1. a) Assuming that you have Visual Studio Code installed, you can launch the Visual Studio Code using the following command:

```
code .
```

2. b) If you don't have Visual Studio Code installed, you can use any other code or text editor
3. Create a new file called main.tf
4. First you need to declare the providers you are going to use to create resources using Terraform. In this lab you are going to use *azurerm* for creating Azure resources, *random* to generate random values needed, and *azapi* to use Azure Resource Manager In main.tf insert the following block of code:

```
# 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"
    }
  }
}
```

5. To create resources in your subscription, Terraform needs to be able to authenticate itself in Microsoft Azure, and that's why you are going to use the Service prinicipal credentials you have created at the begining of this lab. Further in main.tf append the following code:

```
# 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
}
```

6. Since we are refering to some variables we haven't delcared yet, it is time to do so. Now save the main.tf file, and create a new file called variables.tf, and inside that file append the following values:

```
variable "client_id" {
  description = "Service principal ID"
  type        = string
  sensitive   = true
}

variable "client_secret" {
  description = "Service prinicipal secret"
  type        = string
  sensitive   = true
```

```
}


variable "subscription_id" {
  description = "Azure subscription ID"
  type        = string
}
variable "tenant_id" {
  description = "Azure tenant ID"
  type        = string
}
```

7. You have defined the variables, and now you should provide some actual value for those variables. Save variables.tf file, and create a new file secret.tfvars and append your values in it:

```
client_id="XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"
client_secret="cw48xxxxxxxxxxxsoifjaspf"
tenant_id="YYYYYYYY-YYYY-YYYY-YYYY-YYYYYYYYYYYY"
subscription_id="ZZZZZZZZ-ZZZZ-ZZZZ-ZZZZ-ZZZZZZZZZZZZ"
```

Of course you are now going to use the values you've got in the step 7. when you have created a service principal. You should map those values to the Terraform ones where appId is client_id, password is client_secret and tenant is tenant_id

8. Save your secret.tfvars file and reopen the main.tf file. It is time to create something with on Azure, and the first thing you are going to create is a resource group. To do so, append the following code in main.tf:

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

9. Save the main.tf and get back to terminal where you are going to run a command to create a plan for creating a resource group. Notice that you have stated the *var-file* flag which indicates the values for varibles that you are going to use, and in your case that is secret.tfvars file. But before that, you need to initialize Terraform in your root directory for Terraform configuration files, running the following command:

```
terraform init
```

And now, you run the plan:

```
terraform plan -var-file=secret.tfvars -out lab1.tfplan
```

10. If everything is ok with the plan, you should have 1 resource to be created. To apply the plan use the following command:

```
terraform apply lab1.tfplan
```

11. Check your Azure Subscription, and you should see an empty resource group created.
12. Now it's time to create some resources in that resource group. To do so, reopen the main.tf.

    You are going to create several resources: a virtual machine, virtual network, some storage accounts and Azure Synapse Analytics workspace. First you are going to create a randomizer for creating ssh key that is going to be used for your VM. Append the code:

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

Create a randomizer for a storage account and append the following:

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

13. Now it is time to use the SSH generator, and to do so, append the following code:

```
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      = random_pet.ssh_key_name.id
  location  = azurerm_resource_group.rg.location
  parent_id = azurerm_resource_group.rg.id
}

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

14. Now it is time to start building actual components in Azure, and you are going to start from network. Notice that in the upcoming configurations instead of actual name or location of the resource group, you will be referencing the actual property of a resource group that you have created(e.g. in VNET location you would expect to type in "UK South" but you will notice that instead there is azurerm_resource_group.rg.location propoerty). That can come in handy if you decide to retarget the deplyoment for a completely new resource group, you would just change resource group name at the resource group level and that change would reflect all through out the configuration wherever that property is used. Now,use the following code and append it to create a VNET:

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

15. Next, you are going to want to create a subnet in that VNET by appending the following code:

```
# Create a subnet in a vnet
resource "azurerm_subnet" "subnet1" {
  name                 = "internal"
  resource_group_name  = azurerm_resource_group.rg.name
  virtual_network_name = azurerm_virtual_network.vnet.name
```

```
  address_prefixes     = ["10.0.2.0/24"]
}
```

16. Since you will probably want to access your VM from the *outside*, you will want to assign a public IP to your IP, and now you are going to create one by appending the following cofe:

```
# Create public IPs
resource "azurerm_public_ip" "my_public_ip" {
  name                = "myPublicIP"
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
  allocation_method   = "Dynamic"
}
```

17. You will most probably want to create a Network Security Group and a rule to access your VM via SSH. To do so, append the following code:

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

18. There is one last thing that needs to be defined before defining the actual VM, and that is the Network Interface. To create one and connect the security group to network interface, append the following code:

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

  ip_configuration {
    name                          = "internal"
    subnet_id                     = azurerm_subnet.subnet1.id
    private_ip_address_allocation = "Dynamic"
    public_ip_address_id          = azurerm_public_ip.my_public_ip.id
  }
}

# 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
}
```

19. Now, everything is set for you to define a Linux VM. You are going to create a VM based on an Ubuntu distribution of Linux, on F2 type of virtual machine, and you are going to assign it to network interface that you've previously created.

```
# Create a virtual machine based on Ubuntu Linux distribution
resource "azurerm_linux_virtual_machine" "vm1" {
  name                = "vmlinux"
  resource_group_name = azurerm_resource_group.rg.name
  location            = azurerm_resource_group.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"
 }
}
```

20. You've done a lot, and it is time to create what have you actually defined in your Terraform configuration file. Save your main.tf Once again,in your terminal(console) run the commands for plan, and if everything is ok, apply the configuration file:

```
terraform plan -var-file=secret.tfvars -out lab1.tfplan


terraform apply lab1.tfplan
```

21. Now go back to your main.tf file, and create some storage by appending the following code (keep in mind that you are using randomized value since storage accounts need to have unique names). Also later you are going to create Azure Synapse Analytics workspace, and for that purpose, you are going to create a storage account of type Azure Data Lake Storage Gen2 :

```
# Create a new Azure storage account
resource "azurerm_storage_account" "blobstorage" {
 name                     = "sablob${random_id.random_id.hex}"
```

```
  resource_group_name      = azurerm_resource_group.rg.name
  location                 = azurerm_resource_group.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 ADLS Gen2 storage for Azure Synapse Analytics
resource "azurerm_storage_account" "synapsestorage" {
  name                     = "sasyn${random_id.random_id.hex}"
  resource_group_name      = azurerm_resource_group.rg.name
  location                 = azurerm_resource_group.rg.location
  account_tier             = "Standard"
  account_replication_type = "LRS"
  account_kind             = "StorageV2"
  is_hns_enabled           = "true"
}
```

22. Alongside storage, you are going to create another resource: Azure Synapse Analytics workspaceto do so, append the following code:

```
# Create Azure Synapse Analytics workspace
resource "azurerm_storage_data_lake_gen2_filesystem" "synapsedatalake" {
  name               = "adls"
  storage_account_id = azurerm_storage_account.synapsestorage.id
}

resource "azurerm_synapse_workspace" "synapse" {
  name                                 = "asaworkshop"
  resource_group_name                  = azurerm_resource_group.rg.name
```

```
  location                            = azurerm_resource_group.rg.location
  storage_data_lake_gen2_filesystem_id =
azurerm_storage_data_lake_gen2_filesystem.synapsedatalake.id
  sql_administrator_login             = "sqladminuser"
  sql_administrator_login_password    = "H@Sh1CoR3!"

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

23. Once again,in your terminal(console) run the commands for plan, and if everything is ok, apply the configuration file:

```
terraform plan -var-file=secret.tfvars -out lab1.tfplan


terraform apply lab1.tfplan
```

24. After successfuly adding Azure Synapse Analytics update it by changing the sql_administrator_login_password in main.tf file, and instead of H@Sh1CoR3! put B0ET3rr@F!

Save it, and once again, in your terminal(console) run the commands for plan, and if everything is ok, apply the configuration file:

```
terraform plan -var-file=secret.tfvars -out lab1.tfplan


terraform apply lab1.tfplan
```

25. Although it looks like you have done a lot of of work, you see that this is isn't the way of how you should deal with your infrastructure, and that's why you are going to destroy everything and start from scratch with a new approach. Run the following command:

```
terraform destroy -var-file=secret.tfvars
```

Congratulations, you have finished Lab 1!