# Terraform and Azure

In this lab, you will exercise hands-on labs with Terraform. You will learn how to authenticate with Terraform's Azure Provider, manage Terraform state on Azure, and how to stand up resources in Azure using Terraform.

If you need to check your work at the end of any lab use the Solutions folder in the Terraform\_Lab\_Dir directory.

## Lab 1 – Setup and Install

### Section A - Install Visual Studio Code

1. Open the Visual Studio Code(VsCode) page https://code.visualstudio.com/
2. Click the button to download the latest build for Windows. (Note: You will want to use the arrow to select a different OS if not using Windows.)

Graphical user interface, text

Description automatically generated

1. Once it finishes downloading run the installer.
2. Install using default location, when prompted for “Select Additional Tasks” I prefer the following as it adds a few nice features such as right clicking in a folder to open it in VsCode:

Graphical user interface, text, application, email

Description automatically generated

1. Once the install finishes, launch the program.

### Section B - Install Extensions

1. Open VsCode
2. Open the extensions tab:



1. Search for “PowerShell”

Graphical user interface, text, application

Description automatically generated

1. Click on Install
2. Search for “azcli”

Graphical user interface, text, application

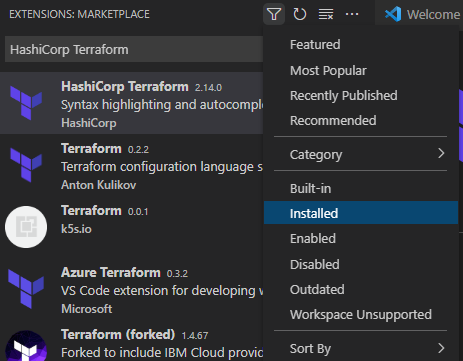
Description automatically generated

1. Click on Install
2. Search for “HashiCorp Terraform”

Graphical user interface, text, application

Description automatically generated

1. Click Install
2. Click on the Extension filter and change it to “Installed”



1. It should include all 3 of the extensions we installed

Text, chat or text message

Description automatically generated

### Section C - Install Terraform for Windows

1. **Open VsCode as an Administrator**
2. With VsCode Open, select the View menu and choose Terminal

**Graphical user interface, application

Description automatically generated**

1. You should now see the terminal in the bottom pane if it was not there before

**Text

Description automatically generated with low confidence**

1. Create a directory for Terraform to store the executable:

**mkdir 'C:\Program Files\Terraform'**

1. Create a Temp directory for our labs

**mkdir c:\Temp\Lab**

1. For our lab we will download Terraform v1.0.8.

(Note: The latest Terraform can be found at: [**https://www.terraform.io/downloads.html**](https://www.terraform.io/downloads.html)) **Invoke-WebRequest -Uri https://releases.hashicorp.com/terraform/1.0.3/terraform\_1.0.8\_windows\_amd64.zip -OutFile c:\temp\lab\terraform.zip**

1. The file is now saved in the “C:\Temp\Lab” directory. Now we will expand it and copy to our Terraform Directory.

**Expand-Archive -Path C:\Temp\Lab\terraform.zip -DestinationPath 'C:\Program Files\Terraform\'**

1. Close VsCode
2. Now we will make sure that the terraform binary is available on the PATH. **Open Control Panel** -> **System** -> **Advanced System settings** -> **Environment Variables**
3. In the **System Variables** panel, scroll down until you find **PATH**, then select **Path** and click **Edit**
4. In the **Edit environment variable** window, click **New**
5. Enter **C:\Program Files\Terraform\**
6. Click **OK** in all three windows closing the **System Properties** completely

### Section D – Validate Terraform Install

After installing Terraform, verify that the installation worked, and what is the Terraform version.

1. Open VsCode as an Administrator
2. Type **terraform** press **Enter**
3. You will see the following result

Text

Description automatically generated

1. Now type **terraform -version** to validate Terraform installed version (As of Oct 1st, 2021, the latest version is 1.0.8)

Text

Description automatically generated

### Section E – Azure PowerShell

1. Set the Execution policy with:

**Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser**

1. Now install Azure PowerShell by running:

**Install-Module -Name Az -Force**

1. Note the previous step can take several minutes to run

### Section F - Install Azure Cli

1. Run the download and install using the following command:

**Invoke-WebRequest -Uri https://aka.ms/installazurecliwindows -OutFile .\AzureCLI.msi; Start-Process msiexec.exe -Wait -ArgumentList '/I AzureCLI.msi /quiet'**

1. Close Visual Studio Code and Re-open it

### Section G – Setup Terraform Lab Files

1. Open a PowerShell window as Administrator
2. Create a new directory for your lab files with the following command:

**mkdir C:\Terraform**

1. Copy the Lab files:

**Invoke-WebRequest -Uri https://learnthecloud.blob.core.windows.net/training/terraform\_labs.zip -OutFile c:\terraform\terraform\_labs.zip**

1. Extract the files:

**Expand-Archive -LiteralPath c:\terraform\terraform\_labs.zip -DestinationPath c:\terraform**

## Lab 2 – Authenticating to Terraform

Terraform supports several different methods for authenticating to Azure:

* **Authenticating to Azure using the Azure CLI – Which we will do in this Lab**
* Authenticating to Azure using Managed Service Identity
* Authenticating to Azure using a Service Principal and a Client Certificate
* Authenticating to Azure using a Service Principal and a Client Secret

### Section A – Ready to Work

1. Open VsCode as an Administrator
2. Click on the File menu and select “Open Folder”

Graphical user interface, application

Description automatically generated

1. Navigate to the terraform\_lab\_dir where you will be writing code for your lab. “C:\terraform\terraform\_lab\_dir” and choose “Select Folder”
2. If asked if you trust the author of the files select yes.

### Section B – Authenticate with PowerShell

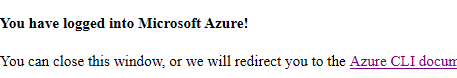
For this lab, we will simply authenticate with our user credentials. However, it is important to note that authenticating via a Service Principal with a client secret which has the minimum rights needed to the subscription is the standard authentication method for an automation pipeline.

1. In the bottom of the VS Code Window you will find the Terminal, if it it not currently open, Click on the Terminal menu and select “New terminal”
2. Establish a connection with PowerShell to your Azure account

Connect-AzAccount

1. Note that VsCode may not have a window visible right away. If this happens minimize the VsCOde window and you should see a login prompt.
2. Login with your credentials.
3. Enter Get-AzSubscription and copy the Id of the Subscription that you will be working in. Save this Subscription ID as well as we will use it later.
4. Enter Set-AzContext -Subscription <insert desired subscription id>

### Section C – Authenticate with Azure Cli

1. In the bottom of the VS Code Window you will find the Terminal, if it it not currently open, Click on the Terminal menu and select “New terminal”
2. Enter az login
3. Your default browser will pop up and prompt you for credentials. Input your credentials
4. Once logged in, you will see a page like this. At this point you may navigate back to your already authenticated powershell session 
5. Once logged in - We will need our Subscription ID from above.
6. Enter az account set --subscription <SUBSCRIPTION>
7. You have successfully authenticated and set your subscription

## Lab 3 – Creating Resources

Now that we are authenticated we can create resources.

### Section A – Creating our First Azure Resource

1. In the left hand pane right click and create a new file named main.tf file, open it in the editor.

1. Append the below code to the file, .\main.tf

provider "azurerm" {

    skip\_provider\_registration = true

features {

}

}

1. Save the file
2. Notice the “skip\_provider\_registration = true”. This setting will make our terraform skip registering of resource providers in our subscription. This is useful when we may be using accounts that do not have full Contributor rights in Azure. For our lab purposes all of the providers just already be registered in our subscription.
3. Now we can create our first resource…
4. Append the below code to the same file, .\main.tf

data "azurerm\_resource\_group" "main" {

name = var.rg\_name

}

1. Now we can deploy our resource group, but first we need to initialize our Terraform
2. Run:

Terraform init

1. Your Output should be similar to:

Text

Description automatically generated

Each time we update our module sources we need to do this so that Terraform can import all the necessary components for our deployments.

1. Now we can use the Plan command to see what will happen if we deploy the code.
2. Run:

Terraform plan -out LabPlan

1. Our output should be similar to:

Text

Description automatically generated

The Plan shows us what will happen if we deploy our code. Using the -out switch saves our plan to the file “LabPlan” so that we can deploy it using the same settings as our plan.

1. Now that our plan works, we can deploy the resource group. So now we run:

Terraform apply LabPlan

1. Our output should resemble:

Text

Description automatically generated

Our output here shows us what actions were taken and whether or not the succeeded.

1. Well that was our first resource. Now Going forward we will keep this code in our

### Section B – Referencing other values

Now that we have a resource created we can start to use data from those resources in other places.

1. First we will add another new resource. This time it will be a storage account where we will store our state files going forward.
2. Add the following code to our Main.tf

resource "azurerm\_storage\_account" "str\_StateStore" {

  name                     = "terraformlabs"

  location                 = "EastUs"

  resource\_group\_name      = "rgTerraformLabs"

  account\_tier             = "Standard"

  account\_replication\_type = "LRS"

}

1. You will notice that we are going to specify all our values here directly. This will work however, it is not best practice so we can clean this up by changing the values to reference our previous resource.
2. In the resource block of our storage account we can change the location and name to reflect that they are getting info from a previous resource. In this case our resource group:

  location                 = azurerm\_resource\_group.rg\_Labs.location

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

We use the format of azurerm\_RESOURCETYPE.NAME.PROPERTY

1. So no matter what we give the resource group for the name and location our storage account will now always reference those.
2. Now we should look at the name and remember that storage accounts are globally unique so we need a way to generate something that is not likely to be repeated
3. To generate a random string we can add the following block to our file:

resource "random\_string" "random" {

  length = 8

  special = false

}

1. Now we can add this to our storage account name

name                 = "terraformlabs${lower(random\_string.random.result)}"

Here we add in both combining a reference with also a transformation. We use a function called lower which convert everything inside the function to lowercase.

1. So now our name will be “terraformlabs(RANDOMCHARACTERS)”
2. Last tweak to our storage is to make sure that when this is created we have the resource group in place first. To do this we add a DepndsOn to our block:

  depends\_on = [

    azurerm\_resource\_group.rg\_Labs,

    random\_string.random

  ]

1. Now our complete resource block looks like this:

resource "azurerm\_storage\_account" "str\_StateStore" {

  name                     = "terraformlabs${lower(random\_string.random.result)}"

  location                 = azurerm\_resource\_group.rg\_Labs.location

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

  account\_tier             = "Standard"

  account\_replication\_type = "LRS"

  depends\_on = [

    azurerm\_resource\_group.rg\_Labs,

    random\_string.random

  ]

}

1. Now we can test our code. Remember we need to use init since we added a new resource provider(Random\_String).
2. Run:

Terraform init

1. Now that we imported the new provider we can do our plan:

Terraform plan -out LabPlan

1. If you formatted the code correctly it should prompt you to run the next command. SO we will do just that:

Terraform Apply LabPlan

### Section C – Using Variables

Now that we have another resource created we can start to take in runtime data in the form of variables.

1. Now we will add another new resource. This time it will be a Public IP for a VM that we will create later. So add the following code to our file:

resource "azurerm\_public\_ip" "pip" {

  name                = "pipUbuntuVM"

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

  location            = azurerm\_resource\_group.rg\_Labs.location

  allocation\_method   = "Static"

  tags = {

    "environment" = "Terraform"

  }

  depends\_on = [

    azurerm\_resource\_group.rg\_Labs

  ]

}

1. So here we have everything we need to deploy a public ip. Now we need to make this customized for our deployment. We can start with the tag values. We can make those take input each run by making them variables. So we can change the tags section to be this:

  tags = {

    "environment" = var.tagEnvironment

  }

Using the format “var.VARIABLENAME” we can reference a variable that is taken at runtime.

1. Now we can define this variable. We can do this in our main.tf file, however best practice is to separate out our variables into a new file since terraform processes all files in the directory as a single module. Create a file named “variables.tf”.
2. In our new variables.tf file add the following code:

variable "tagEnvironment" {

    type = string

    description = "The environment descriptor"

    default = "dev"

}

Now when we run our deployment we can give a value for our tag each time. The default property makes it so we will use that value if none is specified.

1. Now so we do not have to manually input the value each run we can make an input file. This file uses a t.tfvars extension to signify that it contains variables for terraform. SO make a file and name it “inputs.tfvars”
2. Add the following to that file:

tagEnvironment = "TerraformLab"

1. Now each time we run this we can tell terraform to use this file and supply the value “TerraformLab” for our tag.
2. Now we can run our code…
3. We can start with our plan, using the -var-file option to specify where to get our values from:

Terraform plan -var-file .\inputs.tfvars -out LabPlan

1. Notice that on the commands completion if you review the new infrastructure it will show that we are using the value from our variables file.
2. If you formatted the code correctly it should prompt you to run the next command. So we will do just that:

Terraform Apply LabPlan

### Section D – Working with Outputs

1. Let's ensure we output the main object upon deploying. Later, you will run a command that will output this variable. Create a file named .\outputs.tf and add the code below.

output "rg\_main\_output" {

  value = azurerm\_resource\_group.rg\_Labs

}

output "PubIp" {

  value = azurerm\_public\_ip.pip.ip\_address

}

This code shows 2 different ways to format our outputs. First, we are displaying the whole ResourceGroup object and second we have a single value from our IP Address. Both are useful in different situations.

1. Now we can run our deployment and see outputs:
2. We can start with our plan, again this time , using the -var-file option to specify where to get our values from:

Terraform plan -var-file .\inputs.tfvars -out LabPlan

1. If you formatted the code correctly it should prompt you to run the next command. So we will do just that:

Terraform Apply LabPlan

1. This completes our Lab on creating resources. Now we can tear down our resources:

## Lab 3 – Azure as a Backend

Now that we are authenticated we can create resources.

### Section A – Storage Container and Access Key

Now that we have a key vault we need to make a container and store the access key for the storage account in our KeyVault.

1. We will do this quickly with a powershell script

$storage = Get-AzStorageAccount -ResourceGroupName "rgTerraformLabs"

$storagekey = Get-AzStorageAccountKey -Name $storage.StorageAccountName -ResourceGroupName $storage.ResourceGroupName

$context = New-AzStorageContext -StorageAccountName $storage.StorageAccountName -StorageAccountKey $storagekey[0].Value

New-AzStorageContainer -Name "terraformstate" -Context $context

$id = $storage.StorageAccountName.Substring(9)

$vault = New-AzKeyVault -Name "vault-$id" -ResourceGroupName $storage.ResourceGroupName -Location EastUs

$secret = ConvertTo-SecureString -String $storagekey[0].Value -AsPlainText -Force

Set-AzKeyVaultSecret -VaultName $vault.VaultName -SecretValue $secret -Name 'Terraform'

1. Open the file .\main.tf
2. **Append** the below code to the top of the file main.tf and replace the storage account value with yours. (Note: you can place this resource anywhere in the file or even in a new .tf file by itself, but for the sake of uniformity among the class, simply place at the top of main.tf)

terraform {

  backend "azurerm" {

    storage\_account\_name = "REPLACE ME"

    container\_name       = "terraformstate"

    key                  = "prod.terraform.tfstate"

  }

}

**NOTE:** Below we will be removing the terraform state and the terraform directory. Normally we would not do this. We only do this here because we are no longer managing a local state.

1. Delete the current terraform.tfstate file as well as our .terraform folder. This will no longer be needed as it will be uploaded to our Blob container instead

**rm .\terraform.tfstate**

**rm .\.terraform**

### Section B – Retrieve and Set the Access Key

1. In order to use the backend we need our storage access key. We will retrieve it from our key vault. Once we do we will store it as an environment variable
2. From our terminal run:

$ResourceGroup = "rgTerraformLabs"

$Vault = Get-AzKeyVault -ResourceGroupName $ResourceGroup

$env:ARM\_ACCESS\_KEY= $(az keyvault secret show --name terraform --vault-name $Vault.VaultName --query value -o tsv)

### Section C – Run Terraform

1. Init

**terraform init -backend-config="access\_key=$env:ARM\_ACCESS\_KEY"**

**Our state is now configured to be created in our storage account**

1. Plan

**terraform plan -var-file .\inputs.tfvars -out LabPlan**

1. Notice we get an error now stating we are using an existing resource. So Time to convert these existing resources to data sources.
2. Open our Main.tf and change the Resource group entry to:

data "azurerm\_resource\_group" "rg\_Labs" {

  name     = "rgTerraformLabs"

}

1. Change our storage account block to:

data "azurerm\_storage\_account" "str\_StateStore" {

  name                     = "terraformlabsbgelstcz"

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

  depends\_on = [

    data.azurerm\_resource\_group.rg\_Labs,

    random\_string.random

  ]

}

1. Change our publicIp to:

data "azurerm\_public\_ip" "pip" {

  name                = "pipUbuntuVM"

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

  depends\_on = [

    data.azurerm\_resource\_group.rg\_Labs

  ]

}

1. Now a quick change for our outputs. Change your outputs.tf to be:

output "rg\_main\_output" {

  value = data.azurerm\_resource\_group.rg\_Labs

}

output "PubIp" {

  value = data.azurerm\_public\_ip.pip.ip\_address

}

1. Now we can run our setup one more time…
2. Init

**terraform init -backend-config="access\_key=$env:ARM\_ACCESS\_KEY"**

**Our state is now configured to be created in our storage account**

1. Plan

**terraform plan -var-file .\inputs.tfvars -out LabPlan**

1. Apply

**terraform apply LabPlan**

1. Navigate to your subscription in the Azure Portal at https://portal.azure.com and locate your storage container and examine the blobs.
2. Click on terraformstate and see that the pro.terraform.tfstate file has been updated

## Lab 4 – Deploying Resources

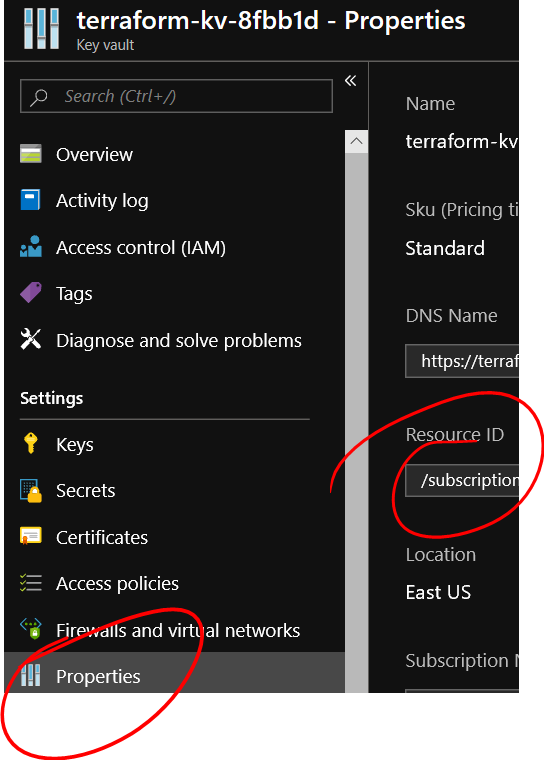
### Section A – Layout and Structure

1. Now we will create a module structure: Create the following Folder and structure inside your lab folder:

Graphical user interface, text, application, chat or text message

Description automatically generated

1. Each of the Readme.md files as well as the Main.tf, variables.tf and outputs.tf can be blank for now.
2. Now we need to add our Key Vault secret.
3. Go to <https://portal.azure.com>
4. Click on secrets
5. Click Generate/import
6. Create a new Secret named “admin-pw”
7. For the value we will use “Terraform4Azure!”
8. Save the new secret
9. Navigate Properties, **copy** the resource id and save it for later



### Section B – Setup root variables.tf

1. Open *./variables.tf*
2. We want to setup some variables to define our environment. Ensure the below code exists

###################################################

# Environment Specs

###################################################

variable "location" {

  type        = string

  description = "The location of the resource group"

  default     = "eastus"

}

variable "environment" {

  type        = string

  description = "The release stage of the environment"

  default     = "dev"

}

variable "rg\_name" {

  type        = string

  description = "The name of the resource group"

  default     = "MyResourceGroup"

}

1. With the code above we are defining the **Location, Environment** and **Resource Group**. These are all basic and provide a solid baseline for any deployment.
2. Next, we will define the variables for values we are retrieving from KeyVault
3. Append to *./variables.tf*

###################################################

# Key Vault Components

###################################################

variable "key\_vault\_name" {

  type        = string

  description = "the name of the main key vault"

  default     = "mykeyvault"

}

variable "key\_vault\_resource\_id" {

  type        = string

  description = "the resource id of the main key vault"

  default     = "MyResourceGroup"

}

variable "admin\_pw\_name" {

  type        = string

  description = "the admin password of the vm"

  default     = "admin-pw"

}

1. Here we define the: **Vault Name, Resource Id**, and **Admin Password**, which is the secret we will retrieve.
2. Finally specific to our deployment we want to name the VM:

###################################################

# Instance Specific

###################################################

variable "vm\_name" {

  type        = string

  description = "the name to give the Virtual Machine"

  default     = "vm"

}

1. Now that we have our root variables file configured make sure to save the file.

### Section C – Code the root main.tf

1. Add in our first Module Reference:

module "vnet" {

  source = "./Modules/Network/VirtualNetwork"

  location    = var.location

  environment = var.environment

}

1. With the above we add the VirtualNetwork module into our build. We are passing in variable values as well as module output that we are going to take in at runtime.
2. Now, we just need to update these to make sure we pass in the right dependencies and variables for each module that we are making.
3. So for our first block we will just add a single line, “Depends on” as well as 2 additional variables. So it should look like this:

module "vnet" {

  source = "./Modules/Network/VirtualNetwork"

  depends\_on = [

    module.nsg

  ]

  location    = var.location

  environment = local.environment

  rg\_name     = data.azurerm\_resource\_group.rg\_Labs.name

  nsg\_id      = module.nsg.id\_out

}

1. Now we update the Module code for the NSG. To do this we will add in another variable to account for the SSH Port.

module "nsg" {

  source = "./Modules/Network/NetworkSecurityGroup"

  location    = var.location

  environment = var.environment

  rg\_name     = data.azurerm\_resource\_group.rg\_Labs.name

  port = 22

}

1. Finally we update the VM Module:

module "vm" {

  source      = "./Modules/Compute/VirtualMachines"

  location    = var.location

  environment = local.environment

  rg\_name     = data.azurerm\_resource\_group.rg\_Labs.name

  vm\_name     = var.vm\_name

  subnet      = module.vnet.subnet\_id

  password    = data.azurerm\_key\_vault\_secret.main.value

  user        = local.vm.user\_name

}

1. As you can see we added a lot more than we did previous, including referencing the output for the **Subnet\_Id** from the Network module, however, we are still just passing values and some are referenced now with a **Locals** block. We will create that next.
2. Locals are values that we want short aliases for and reference just within the same module. However, we can still pass them to other modules.
3. Add the following block to main.tf

locals {

  environment = var.environment

  vm = {

    computer\_name = var.vm\_name

    user\_name     = "admin1234"

  }

}

1. Now since we added the **Environment** value to our local, We can replace it on our module blocks. So lets do that now:

 environment = local.environment

1. We have 1 final block to add which is a data block to retrieve the Key Vault data for us

data "azurerm\_key\_vault\_secret" "main" {

  name         = var.admin\_pw\_name

  key\_vault\_id = var.key\_vault\_resource\_id

}

1. Alright, now make sure to save the main.tf as we are done updating it for now.

### Section D – Adding Outputs

1. Open the root outputs.tf file
2. Add the following lines:

output "vmEndpoint" {

  value = module.vm.ip

}

output "username" {

  value = local.vm.user\_name

}

1. Here we are outputting our **public ip** as well as the **username** of the VM.

### Section E – Inputs file(tfvars)

1. Open the root inputs.tf file
2. Add the following lines:

key\_vault\_name="<insert key\_vault\_name >"

key\_vault\_resource\_id="<insert key\_vault\_resource\_id value>"

admin\_pw\_name         = "admin-pw"

location              = "eastus"

environment           = "dev"

rg\_name               = "<REPLACE ME>"

vm\_name               = "vm1"

1. Replace the associated key vault values with the name and resource ID that you saved from earlier.

### Section F – NetworkSecurityGroup module

1. Open the /terraform\_lab\_dir/Modules/Network/NetworkSecurityGroup/main.tf file and add the following lines:

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

  name                = "${var.environment}nsg"

  location            = var.location

  resource\_group\_name = var.rg\_name

  security\_rule {

    name                       = "AllowSSHIn"

    priority                   = 1300

    direction                  = "Inbound"

    access                     = "Allow"

    protocol                   = "Tcp"

    source\_port\_range          = "\*"

    destination\_port\_range     = var.port

    source\_address\_prefix      = "\*"

    destination\_address\_prefix = "\*"

  }

  tags = {

    environment = var.environment

  }

}

1. Open the /terraform\_lab\_dir/Modules/Network/NetworkSecurityGroup/variables.tf file and add the following code:

variable "location" {

  type        = string

  description = "The location of the resource group"

  default     = "eastus"

}

variable "environment" {

  type        = string

  description = "The release stage of the environment"

  default     = "dev"

}

variable "rg\_name" {

  type        = string

  description = "The name of the resource group"

  default     = "MyResourceGroup"

}

variable "port" {

  type        = number

  description = "The port to open"

}

1. Open the /terraform\_lab\_dir/Modules/Network/NetworkSecurityGroup/outputs.tf file and add the following code:

output "id\_out" {

  value = azurerm\_network\_security\_group.nsg.id

}

1. Save all 3 files

### Section G – VirtualNetwork module

1. Open the /terraform\_lab\_dir/Modules/Network/VirtualNetwork/main.tf file and add the following lines:

resource "azurerm\_virtual\_network" "main" {

  name                = "${var.environment}-network"

  address\_space       = ["10.0.0.0/16"]

  location            = var.location

  resource\_group\_name = var.rg\_name

  subnet {

    name           = "subnet1"

    address\_prefix = "10.0.0.0/24"

    security\_group = var.nsg\_id

  }

}

1. Open the /terraform\_lab\_dir/Modules/Network/VirtualNetwork /variables.tf file and add the following code:

variable "location" {

  type        = string

  description = "The location of the resource group"

  default     = "eastus"

}

variable "environment" {

  type        = string

  description = "The release stage of the environment"

  default     = "dev"

}

variable "rg\_name" {

  type        = string

  description = "The name of the resource group"

  default     = "MyResourceGroup"

}

variable "nsg\_id" {

  type        = string

  description = "Resource ID of the NSG"

}

1. Open the /terraform\_lab\_dir/Modules/Network/VirtualNetwork/output.tf file and add the following code:

output "subnet\_id" {

  value = azurerm\_virtual\_network.main.subnet.\*.id[0]

}

1. Save all 3 files

### Section H – VirtualMachines module

1. Open the /terraform\_lab\_dir/Modules/Compute/VirtualMachines/main.tf file and add the following lines:

resource "azurerm\_network\_interface" "vm" {

  name                = "${var.environment}-nic"

  location            = var.location

  resource\_group\_name = var.rg\_name

  ip\_configuration {

    name                          = "ipconfig"

    subnet\_id                     = var.subnet

    private\_ip\_address\_allocation = "Dynamic"

    public\_ip\_address\_id          = azurerm\_public\_ip.vm.id

  }

  depends\_on = [azurerm\_public\_ip.vm]

}

resource "azurerm\_public\_ip" "vm" {

  name                = "${var.environment}-pip"

  location            = var.location

  resource\_group\_name = var.rg\_name

  allocation\_method   = "Static"

}

resource "azurerm\_virtual\_machine" "vm" {

  name                  = var.vm\_name

  location              = var.location

  resource\_group\_name   = var.rg\_name

  network\_interface\_ids = [azurerm\_network\_interface.vm.id]

  vm\_size               = "Standard\_DS1\_v2"

  storage\_image\_reference {

    publisher = "Canonical"

    offer     = "UbuntuServer"

    sku       = "16.04-LTS"

    version   = "latest"

  }

  storage\_os\_disk {

    name              = "myosdisk1"

    caching           = "ReadWrite"

    create\_option     = "FromImage"

    managed\_disk\_type = "Standard\_LRS"

  }

  os\_profile {

    computer\_name  = var.vm\_name

    admin\_username = var.user

    admin\_password = var.password

  }

  os\_profile\_linux\_config {

    disable\_password\_authentication = false

  }

  tags = {

    environment = var.environment

  }

}

1. Open the /terraform\_lab\_dir/Modules/Compute/VirtualMachines/variables.tf file and add the following code:

variable "location" {

  type        = string

  description = "The location of the resource group"

  default     = "eastus"

}

variable "environment" {

  type        = string

  description = "The release stage of the environment"

  default     = "dev"

}

variable "rg\_name" {

  type        = string

  description = "The name of the resource group"

  default     = "MyResourceGroup"

}

variable "subnet" {

  type        = string

  description = "Resource ID of the subnet to join"

}

variable "user" {

  type        = string

  description = "Admin user for the VM"

}

variable "vm\_name" {

  type        = string

  description = "Name to give the VM"

}

variable "password" {

  type        = string

  sensitive   = true

  description = "Password for the VM"

}

1. Open the /terraform\_lab\_dir/Modules/Compute/VirtualMachines/output.tf file and add the following code:

output "ip" {

  value = azurerm\_public\_ip.vm.ip\_address

}

1. Save all 3 files

### Section I – Deploy Infrastructure

1. First, let us make sure that all our files are formatted correctly. To do this run:

**Terraform fmt -recursive**

1. Now that everything is formatted correctly let us run our init:

**terraform init -backend-config="access\_key=$env:ARM\_ACCESS\_KEY" -reconfigure**

1. Now we can run our plan:

**terraform plan -var-file .\inputs.tfvars -out LabPlan**

1. Now we can run our apply:

**terraform apply LabPlan**

1. After you apply, you should see credentials for logging on.
2. Time to test the login:

Run ssh admin1234@<vmEndpoint value>

1. When prompted, enter yes
2. Enter the password value, “Terraform4Azure!”
3. Your prompt should now change to admin1234@vm1
4. Type Exit and hit enter to close the SSH Session

If you ran into any issues you can reference the Solutions/Lab 6 directory