# QATIP Intermediate
# Azure Lab04
# Terraform Modules

## Contents


Lab Objectives ..... 2

Before you begin ..... 2

Try It Yourself ..... 2

- Challenge ..... 2
- Key Requirements ..... 3

Step-by-Step Instructions ..... 3

- Step 1: Set Up the Project Directory ..... 3
- Step 2: Create the VNet Module ..... 3
- Step 3: Create the Root module Configuration ..... 4
- Step 4: Initialize, Review and Apply ..... 5
- Step 5: Verify the Deployment ..... 5
- Step 6: Clean Up Resources ..... 6

Solution Overview ..... 6

- Module-Based Approach ..... 6
- Exposing and Utilizing Module Outputs ..... 7
- Scalability and Reusability ..... 7

Demonstration Deployment ..... 7

- Overview ..... 7
- Vnet and subnet provisioning flow ..... 7
- Logic ..... 8
- Network Security Group (NSG) provisioning. ..... 11
- Logic ..... 12

NSG association provisioning ..... 13
Logic ..... 13
Network Peering provisioning ..... 14
Logic ..... 16
Lab Clean Up ..... 17

## Lab Objectives

- Develop reusable Terraform modules for Azure Virtual Networks.
- Parameterize the module using variables.
- Deploy multiple virtual networks by reusing the module with different parameters.
- Utilize module outputs in the root module to create dependent resources.

## Before you begin

- Ensure you have completed Lab0 before attempting this lab.
- In the IDE terminal pane, enter the following command...

  **cd c:\azure-tf-int\lab\04**
- This shifts your current working directory to labs\04. Ensure all commands are executed in this directory
- Close any open files and use the Explorer pane to navigate to and open the labs\04 folder.

## Try It Yourself

### Challenge

Using **c:\azure-tf-int\labs\04** as your root directory, create a reusable module structure to deploy 2 Azure virtual networks into resource group RG1, located in East US. As you deploy the module, expose outputs from it and use these, from the root module, to create a subnet on each network.

## Key Requirements

- Create **main.tf**, **variables.tf**, and **outputs.tf** files as part of the module
- Deploy at least two virtual networks using a module structure
- Pass output values from the module back to the root configuration
- Use the outputs from the module to create a subnet in each network

# Step-by-Step Instructions

## Step 1: Set Up the Project Directory

Ensure you have moved to the lab directory: **cd c:\azure-tf-int\labs\04**

Create a subdirectory for the module: **mkdir modules\vnet**

## Step 2: Create the VNet Module

Navigate to the new vnet directory and create the following files:

**main.tf**:

```
resource "azurerm_virtual_network" "vnet" {
  name = var.vnet_name
  address_space = var.address_space
  location = var.location
  resource_group_name = var.resource_group_name
}
```

**variables.tf**:

```
variable "vnet_name" {}
variable "address_space" {
  type = list(string)
}
variable "location" {}
variable "resource_group_name" {}
```

**outputs.tf**:

```
output "vnet_id" {
 value = azurerm_virtual_network.vnet.id
}

output "vnet_name" {
 value = azurerm_virtual_network.vnet.name
}

output "vnet_address_space" {
 value = azurerm_virtual_network.vnet.address_space
}
```

## Step 3: Create the Root module Configuration

In **c:\azure-tf-int\labs\04**, create **main.tf,** updating line **2** with your subscription id

```
provider "azurerm" {
 subscription_id = "<your subscription id>"
 features {}
}

resource "azurerm_resource_group" "example" {
 name    = "RG1"
 location = "East US"
}

module "vnet1" {
 source          = "./modules/vnet"
 vnet_name        = "vnet-01"
 address_space     = ["10.0.0.0/16"]
 location         = azurerm_resource_group.example.location
 resource_group_name = azurerm_resource_group.example.name
}

module "vnet2" {
```

```
  source          = "./modules/vnet"
  vnet_name       = "vnet-02"
  address_space   = ["10.1.0.0/16"]
  location        = azurerm_resource_group.example.location
  resource_group_name = azurerm_resource_group.example.name
}

resource "azurerm_subnet" "subnet1" {
  name = "subnet-01"
  resource_group_name  = azurerm_resource_group.example.name
  virtual_network_name = module.vnet1.vnet_name
  address_prefixes     = ["10.0.1.0/24"]
}

resource "azurerm_subnet" "subnet2" {
  name            = "subnet-02"
  resource_group_name  = azurerm_resource_group.example.name
  virtual_network_name = module.vnet2.vnet_name
  address_prefixes     = ["10.1.1.0/24"]
}
```

## Step 4: Initialize, Review and Apply

**terraform init**

**terraform plan**

**terraform apply**

Confirm with **yes** when prompted.

## Step 5: Verify the Deployment

**Terraform state list**

Verify in the Azure Portal.

### Step 6: Clean Up Resources

**terraform destroy** confirmed with **yes**

## Solution Review

This configuration is a well-structured solution to the given challenge. It adheres to best practices by utilizing **reusable modules** to deploy two Azure Virtual Networks (vnet1 and vnet2) within a single resource group (RG1), located in **East US**. The root module orchestrates the deployment by calling the virtual network module twice—once for each VNet—while ensuring outputs from the module are exposed and consumed to create subnets within each network.

### Module-Based Approach

A key requirement of the challenge is to create a **reusable module structure**. This is accomplished by defining a vnet module stored under c:\azure-tf-int\labs\04\modules\vnet. Instead of defining virtual networks directly in the root module, this module encapsulates the logic for provisioning a VNet with configurable parameters, such as:

- **Name (vnet_name)** – Allows different VNets to be created using the same module.
- **Address Space (address_space)** – Ensures each network has a unique IP range.
- **Resource Group (resource_group_name)** – Associates the VNets with the specified resource group.
- **Location (location)** – Ensures all resources are deployed in East US.

By using **module instantiation**, the root module deploys two virtual networks (vnet1 and vnet2) by calling the vnet module twice, passing different parameters for each instance.

### Exposing and Utilizing Module Outputs

A critical part of the challenge was to expose outputs from the module and utilize them in the root module to create subnets. This is achieved through the output block within the vnet module, which ensures that the virtual network's name and ID are accessible after deployment. The root module then references these outputs to create subnets within each network.

### Scalability and Reusability

By structuring the solution with modules, this approach is highly scalable. If additional virtual networks are required, they can be easily instantiated without modifying the module itself. Additionally, the structure promotes code reuse, making it applicable for future scenarios where multiple VNets need to be deployed across different resource groups or locations.

## Demonstration Deployment

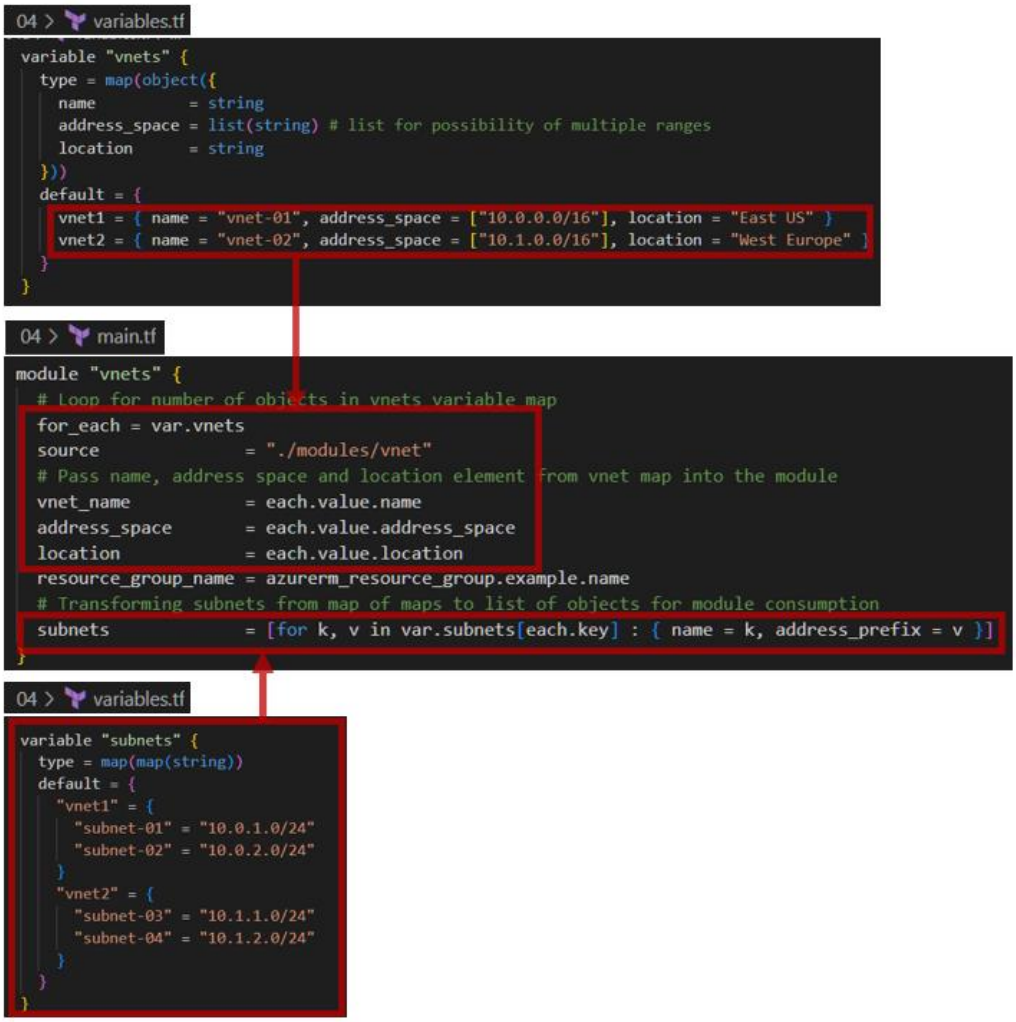
### Overview

1. This demonstration walks you through a more complex use of modules to deploy Virtual Networks (VNets), Network Security Groups (NSGs), and VNet Peering. The goal is to further highlight the benefits of dynamic and reusable infrastructure by leveraging Terraform modules and outputs.
2. In your IDE, navigate to **c:\azure-tf-int\labs\04demo**
3. Review the provisioned files
4. Update line **2** of **demo\main.tf** with your **subscription id** and save the changes

### VNet and subnet provisioning flow

The image below highlights the configuration of a module call in the root modules' **main.tf**. It shows how values can be passed into the module and that these values can be dynamically derived from variables. It also

demonstrates that a module can be invoked multiple times, with each iteration passing differing values.




## Logic

To efficiently provision multiple VNets with multiple subnets, a single module block (“vnets”) is used. Values to be passed down to the module are first read from **variables.tf**

The “**for_each = var.vnets**” loop in the module block will invoke the creation of as many VNets as there are objects in **var.vnets**, in this case 2’

**var.subnets** in **variables.tf** is a nested map. The outer map uses VNet names as keys and the inner map contains subnets.  When calling the

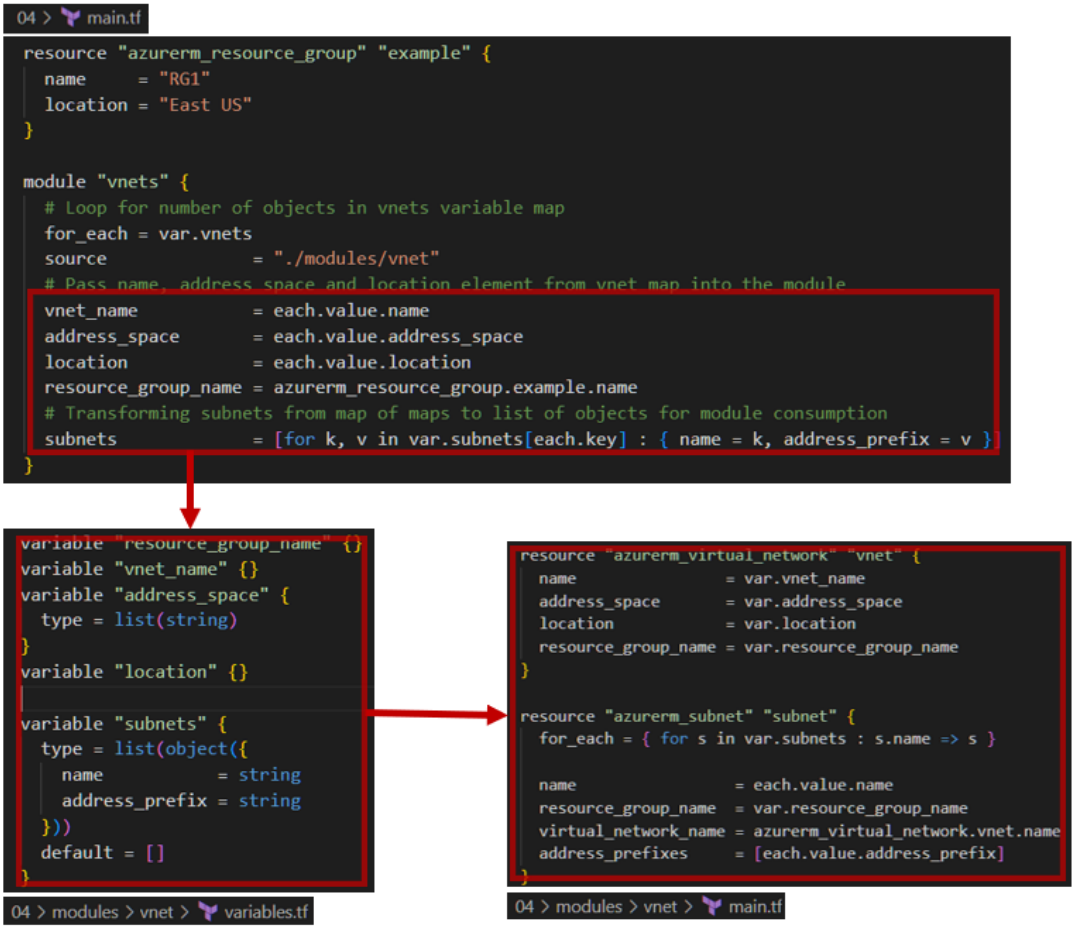
module we only pass the subnets that belong to that specific VNet. This is achieved through **list comprehension**, used here in the format:

***subnets =***

***[for k, v in var.subnets[each.key] : { name = k, address_prefix = v }]***

**each.key** Gets the current VNet's name (vnet1 or vnet2). The key value used each time is derived from the main **for_each** loop

**var.subnets[each.key]** Extracts only the subnets for that VNet List conversion converts the inner map into a list of objects ({ name, address_prefix })




The image above highlights the workflow when a module is invoked. In the vnets module, the variables file is populated with the values passed down to it from the root module. These are then used by the modules' **main.tf** to deploy the VNet and subnets.

So, in summary..

- A single module block provisions all VNets
- Looping through the **var.vnets** variable using **for_each** creates VNets dynamically.
- Subnets are stored as a map of maps variable, grouped under their respective VNets.
- List comprehension ([for k, v in var.subnets[each.key] ...]) ensures only the correct subnets are assigned to each VNet.
- The VNet module receives these values, creating subnets dynamically using **for_each**.

Run **terraform init**

```
Initializing the backend...
Initializing modules...
- vnets in modules\vnet
Initializing provider plugins...
- Reusing previous version of hashicorp/azurerm from the dependency lock file
- Installing hashicorp/azurerm v4.18.0...
- Installed hashicorp/azurerm v4.18.0 (signed by HashiCorp)
```

Note how the vnet module is initialized

Run **terraform plan** and then **apply**

Run **terraform state list**

```
azurerm_resource_group.example
module.vnets["vnet1"].azurerm_subnet.subnet["subnet-01"]
module.vnets["vnet1"].azurerm_subnet.subnet["subnet-02"]
module.vnets["vnet1"].azurerm_virtual_network.vnet
module.vnets["vnet2"].azurerm_subnet.subnet["subnet-03"]
module.vnets["vnet2"].azurerm_subnet.subnet["subnet-04"]
module.vnets["vnet2"].azurerm_virtual_network.vnet
```

Run **terraform console**

Enter **module.vnets** to view details of the module resources

```
> module.vnets
{
  "vnet1" = {
    "subnets" = {
      "subnet-01" = "/subscriptions/911e746f-0030-
      "subnet-02" = "/subscriptions/911e746f-0030-
    }
    "vnet_details" = {
      "id" = "/subscriptions/911e746f-0030-41d8-83
      "location" = "eastus"
      "name" = "vnet-01"
      "resource_group" = "RG1"
    }
  }
  "vnet2" = {
    "subnets" = {
      "subnet-03" = "/subscriptions/911e746f-0030-
      "subnet-04" = "/subscriptions/911e746f-0030-
    }
    "vnet_details" = {
      "id" = "/subscriptions/911e746f-0030-41d8-83
      "location" = "westeurope"
      "name" = "vnet-02"
```

Enter **exit** to exit the console

## Network Security Group (NSG) provisioning.

Uncomment lines 30-36 in **04\demo\main.tf** and save the changes

## Logic

**modules\vnets\outputs.tf** exposes the details of the VNets created

**maint.tf** uses this information to populate its **nsgs** module call, looping for as many VNets that exist

**modules\nsg\variables.tf** is populated with this information

**modules\nsg\main.tf** deploys the NSGs based on these module variables

Run **terraform init** to initialize the new module

```
Initializing the backend...
Initializing modules...
- nsgs in modules\nsg
Initializing provider plugins...
- Reusing previous version of hashicorp/azurerm from the dependency lock file
- Using previously-installed hashicorp/azurerm v4.18.0
```

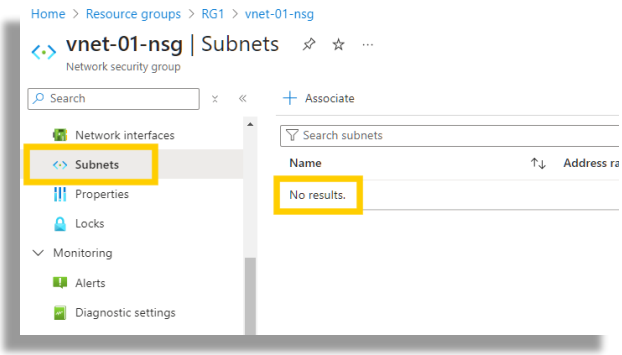
Note how the **nsg** module is now initialized

Run **terraform plan** and then **apply**

Run **terraform state list**

```
azurerm_resource_group.example
module.nsgs["vnet1"].azurerm_network_security_group.nsg
module.nsgs["vnet2"].azurerm_network_security_group.nsg
module.vnets["vnet1"].azurerm_subnet.subnet["subnet-01"]
module.vnets["vnet1"].azurerm_subnet.subnet["subnet-02"]
module.vnets["vnet1"].azurerm_virtual_network.vnet
module.vnets["vnet2"].azurerm_subnet.subnet["subnet-03"]
module.vnets["vnet2"].azurerm_subnet.subnet["subnet-04"]
module.vnets["vnet2"].azurerm_virtual_network.vnet
```

Use the Portal and verify that the NSGs exist but that our subnets are not associated with them yet...



## NSG association provisioning

Uncomment lines **38-54** in **04\demo\main.tf** and save the changes

```
resource "azurerm_subnet_network_security_group_association" "nsg_association" {
  for_each = merge([
    for k, v in module.vnets : {for subnet_name, subnet_id in v.subnets : subnet_name => {
      vnet  = k
      id    = subnet_id
      }
    }
  ]...)

  subnet_id = each.value.id

  network_security_group_id = module.nsgs[each.value.vnet].nsg_id
}
```

## Logic

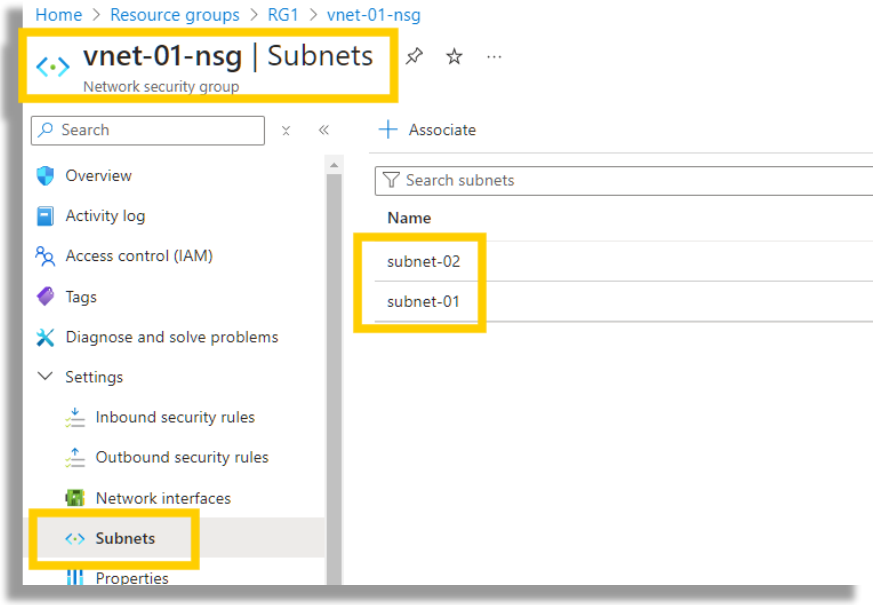
- From the output of the vnets module, loop over each VNet and extract its subnets.
- Store both **VNet name and Subnet ID** in a structured map.
- Flatten the map using merge([...]), allowing **for_each** to iterate over each subnets.
- Dynamically look up the correct NSG based on each.value.vnet

Run **terraform plan** and then **apply**

Run **terraform state list**

```
azurerm_resource_group.example
azurerm_subnet_network_security_group_association.nsg_association["subnet-01"]
azurerm_subnet_network_security_group_association.nsg_association["subnet-02"]
azurerm_subnet_network_security_group_association.nsg_association["subnet-03"]
azurerm_subnet_network_security_group_association.nsg_association["subnet-04"]
module.nsgs["vnet1"].azurerm_network_security_group.nsg
module.nsgs["vnet2"].azurerm_network_security_group.nsg
module.vnets["vnet1"].azurerm_subnet.subnet["subnet-01"]
module.vnets["vnet1"].azurerm_subnet.subnet["subnet-02"]
module.vnets["vnet1"].azurerm_virtual_network.vnet
module.vnets["vnet2"].azurerm_subnet.subnet["subnet-03"]
module.vnets["vnet2"].azurerm_subnet.subnet["subnet-04"]
module.vnets["vnet2"].azurerm_virtual_network.vnet
```

Switch to the Portal to verify the Subnet-NSG associations




## Network Peering provisioning

Uncomment lines **56-66** in **c:\azure-tf-int\labs\04\demo\main.tf** and save the changes

```
resource "azurerm_virtual_network_peering" "peerings" {
  for_each = {
    peer1to2 = { local = "vnet1", remote = "vnet2" }
    peer2to1 = { local = "vnet2", remote = "vnet1" }
  }

  name                      = each.key
  resource_group_name       = azurerm_resource_group.example.name
  virtual_network_name      = module.vnets[each.value.local].vnet_details.name
  remote_virtual_network_id = module.vnets[each.value.remote].vnet_details.id
}
```

**resource "azurerm_virtual_network_peering" "peerings"**

This defines a resource of type **azurerm_virtual_network_peering**.

Instead of defining multiple individual resources, the **for_each** is used to dynamically create two peerings.

**for_each = {...}**

This is a **map** that contains two key-value pairs.

The **keys** (peer1to2, peer2to1) represent the names of the two peering relationships.

Each key maps to an **object** { local = "vnetX", remote = "vnetY" }, which holds the names of the local and remote virtual networks.

**name = each.key**

The each.key will be "peer1to2" for the first iteration and "peer2to1" for the second.

This dynamically sets the peering name based on the map keys.

**resource_group_name**

This stays the same for both peering resources since both virtual networks belong to the same resource group

**module.vnets[each.value.local].vnet_details.name**

each.value.local refers to the local virtual network for the current iteration.

On the first iteration (peer1to2), each.value.local = "vnet1", so it resolves to module.vnets["vnet1"].vnet_details.name.

On the second iteration (peer2to1), each.value.local = "vnet2", so it resolves to module.vnets["vnet2"].vnet_details.name.

**module.vnets[each.value.remote].vnet_details.id**

each.value.remote refers to the remote virtual network for the current iteration.

On the first iteration (peer1to2), each.value.remote = "vnet2", resolving to module.vnets["vnet2"].vnet_details.id.

On the second iteration (peer2to1), each.value.remote = "vnet1", resolving to module.vnets["vnet1"].vnet_details.id.

## Logic

Terraform **loops over** the **for_each** map.

It creates **two instances** of azurerm_virtual_network_peering:

- One for peer1to2, linking vnet1 to vnet2.
- Another for peer2to1, linking vnet2 to vnet1.

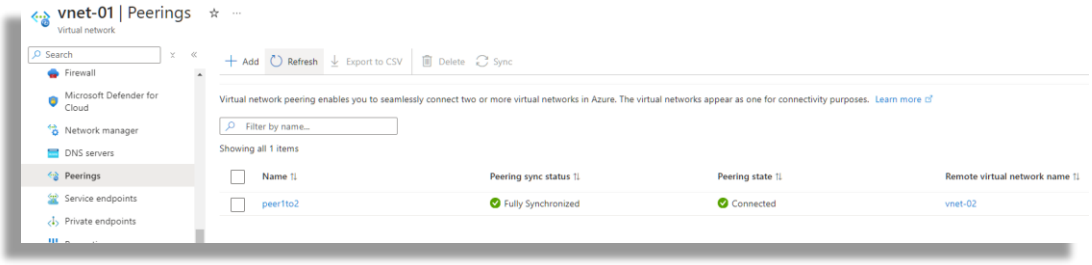
Each instance gets dynamically populated with the correct name, virtual_network_name, and remote_virtual_network_id

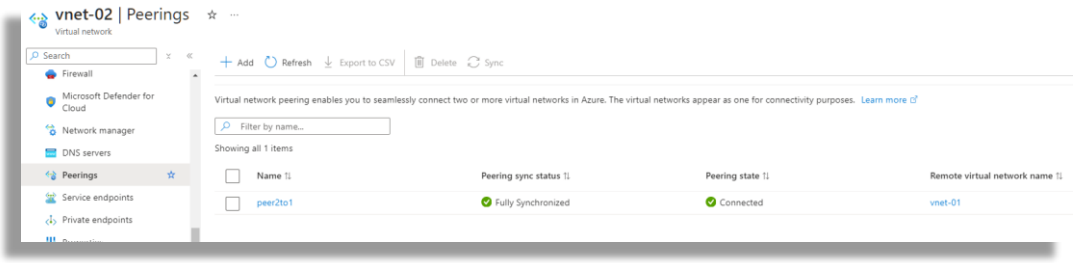
Run **terraform plan** and then **apply**

Run **terraform state list**

```
azurerm_resource_group.example
azurerm_subnet_network_security_group_association.nsg_association["subnet-01"]
azurerm_subnet_network_security_group_association.nsg_association["subnet-02"]
azurerm_subnet_network_security_group_association.nsg_association["subnet-03"]
azurerm_subnet_network_security_group_association.nsg_association["subnet-04"]
azurerm_virtual_network_peering.peer1to2
azurerm_virtual_network_peering.peer2to1
module.nsgs["vnet1"].azurerm_network_security_group.nsg
module.nsgs["vnet2"].azurerm_network_security_group.nsg
module.vnets["vnet1"].azurerm_subnet.subnet["subnet-01"]
module.vnets["vnet1"].azurerm_subnet.subnet["subnet-02"]
module.vnets["vnet1"].azurerm_virtual_network.vnet
module.vnets["vnet2"].azurerm_subnet.subnet["subnet-03"]
module.vnets["vnet2"].azurerm_subnet.subnet["subnet-04"]
module.vnets["vnet2"].azurerm_virtual_network.vnet
```

Confirm the peerings in the Portal



## Lab Clean Up

Destroy all resources with **terraform destroy** confirmed with **yes**

**## Congratulations, you have completed this lab ##**