QATIP Intermediate
AWS Lab04
Terraform Modules

Lab Objectives

- Develop reusable Terraform modules for AWS Virtual Private Clouds (VPCs).
- Parameterize the module using variables.
- Deploy multiple VPCs by reusing the module with different parameters in the same AWS region.
- Utilize module outputs in the root module to create dependent resources.

Before you begin

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

   ```
   cd ~/environment/aws-tf-int/labs/04
   ```

3. This shifts your current working directory to labs/04. Ensure all commands are executed in this directory
4. Close any open files and use the Explorer pane to navigate to and open the labs/04 folder.

Try It Yourself

Challenge

Using **labs/04** as your root directory, create a reusable module structure to deploy two AWS VPCs in the same AWS region, us-east-1. As you deploy the module, expose outputs from it and use these outputs in the root module to create subnets within each VPC.

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 ~/environment/aws-tf-int/labs/04
```

Create a subdirectory for the module:

```
mkdir modules/vpc
```

Step 2: Create the VPC Module

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

main.tf:

```
resource "aws_vpc" "lab_vpc" {
 cidr_block = var.cidr_block
 enable_dns_support = true
 enable_dns_hostnames = true
 tags = {
  Name = var.vpc_name
 }
}
```

variables.tf:

```
variable "vpc_name" {}
variable "cidr_block" {}
```

outputs.tf:

```
output "vpc_id" {
 value = aws_vpc.lab_vpc.id
}
output "vpc_cidr" {
 value = aws_vpc.lab_vpc.cidr_block
}
```

Step 3: Create the Root module Configuration

In **labs/04**, create **main.tf**

```
provider "aws" {
 region = "us-east-1"
}

module "vpc1" {
 source   = "./modules/vpc"
 vpc_name  = "vpc-01"
 cidr_block = "10.0.0.0/16"
}

module "vpc2" {
 source   = "./modules/vpc"
 vpc_name  = "vpc-02"
 cidr_block = "10.1.0.0/16"
}

resource "aws_subnet" "subnet1" {
 vpc_id    = module.vpc1.vpc_id
 cidr_block = "10.0.1.0/24"
 tags = {
  Name = "subnet-01"
 }
```

```
}

resource "aws_subnet" "subnet2" {
 vpc_id    = module.vpc2.vpc_id
 cidr_block = "10.1.1.0/24"
 tags = {
  Name = "subnet-02"
 }
}
```

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 lab follows best practices by utilizing reusable Terraform modules to deploy two AWS VPCs and creating subnets in the root module based on module outputs. The root module orchestrates the deployment by calling the VPC module twice, each time with different parameters.

Module-Based Approach

Instead of defining VPCs directly in the root module, this lab uses a modular approach, which provides:

- Scalability: Easily extendable to deploy additional VPCs.
- Reusability: The same module can be reused with different parameters.
- Maintainability: Reduces duplication and simplifies future modifications.

Exposing and Utilizing Module Outputs

A critical part of the lab is exposing outputs from the module and utilizing them in the root module. The output block in outputs.tf ensures that the VPC ID and CIDR block are accessible after deployment and can be referenced in subnet creation.

AWS Demonstration Deployment

Overview

This demonstration walks you through a more complex use of modules to deploy VPCs (Virtual Private Clouds) with subnets, Security Groups (SGs), and VPC Peering. The goal is to highlight the benefits of dynamic and reusable infrastructure by leveraging Terraform modules and outputs.

1. In your IDE, navigate to /labs/04/demo

2. Review the provisioned files.

VPC and Subnet Provisioning Flow

The diagram below highlights the configuration of a module call in the root module's main.tf. It shows how values can be passed into the module and dynamically derived from variables. It also demonstrates that a module can be invoked multiple times, with each iteration passing different values.

Logic

To efficiently provision multiple VPCs with multiple subnets, a single module block (vpcs) is used. Values passed into the module are first read from variables.tf.

- The for_each = var.vpcs loop in the module block will create as many VPCs as there are objects in var.vpcs, in this case, 2.
- var.subnets in variables.tf is a nested map. The outer map uses VPC names as keys, and the inner map contains subnets.
- When calling the module, only the subnets that belong to that specific VPC are passed.

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

- each.key gets the current VPC's name (vpc1 or vpc2).
- var.subnets[each.key] extracts only the subnets for that VPC.

- List conversion converts the inner map into a **list of objects** ({ name, cidr_block }).

Deploy the infrastructure

1. Initialize the Terraform working directory:

   terraform init

   Note how the vpc module is initialized.

2. Run terraform plan and terraform apply

   terraform plan

   terraform apply

   Confirm with **yes** when prompted.

3. Run terraform state list to verify deployed resources

   terraform state list

4. Run terraform console to view vpc details

   terraform console

   Enter **module.vpcs** to view details of the module resources, then type **exit** to close the console.

Security Group (SG) Provisioning

1. Uncomment lines 23-28 in 04/demo/main.tf and save the changes.

   #### Logic

   - modules/vpcs/outputs.tf exposes the details of the VPCs created.
   - main.tf uses this information to populate its sgs module call, looping for as many VPCs that exist.
   - modules/sg/variables.tf is populated with this information.
   - modules/sg/main.tf deploys the Security Groups based on these module variables.

2. Run:
   terraform init
   terraform plan

terraform apply

3. Run **terraform state list** to verify the SGs.
4. Use the **AWS Console** to verify that the Security Groups exist but that subnets are not associated with them yet.

SG Association Provisioning

1. Uncomment lines 29-41 in 04/demo/main.tf and save the changes.

   ### Logic

   - Extracts subnets from each VPC.
   - Stores VPC name and Subnet ID in a structured map.
   - Flattens the map using merge([...]) to allow for_each to iterate over subnets.
   - Dynamically looks up the correct Security Group based on each.value.vpc.

2. Run:

   terraform plan
   terraform apply

3. Run **terraform state list**.
4. Switch to the **AWS Console** to verify that Subnets are associated with the Security Groups.

VPC Peering Provisioning

1. Uncomment lines 43-53 in c:\aws-tf-int\labs\04\demo\main.tf and save the changes.

```
resource "aws_vpc_peering_connection" "peerings" {
 for_each = {
  peer1to2 = { local = "vpc1", remote = "vpc2" }
  peer2to1 = { local = "vpc2", remote = "vpc1" }
 }

 vpc_id      = module.vpcs[each.value.local].vpc_id
 peer_vpc_id = module.vpcs[each.value.remote].vpc_id
 auto_accept = true
```

}

Logic

Terraform loops over the for_each map to create two **VPC Peering Connections**:

- **One for peer1to2**, linking vpc1 to vpc2.
- **Another for peer2to1**, linking vpc2 to vpc1.

2. Run:

terraform plan

terraform apply

3. Run terraform state list.
4. Confirm the peerings in the **AWS Console**.

Lab Clean Up

Destroy all resources with:

terraform destroy

Confirm with **yes** when prompted.

Summary

☑ **A single module block provisions all VPCs**. ☑ **Looping through var.vpcs dynamically creates VPCs**. ☑ **Subnets are stored as a map of maps, grouped by VPC**. ☑ **List comprehension ensures only the correct subnets are assigned to each VPC**. ☑ **The VPC module provisions subnets dynamically using for_each**. ☑ **Security Groups and Subnets are provisioned using Terraform modules**. ☑ **VPC Peering is dynamically created between the two VPCs**.

🚀 **Congratulations! You have successfully completed this AWS deployment demonstration.**

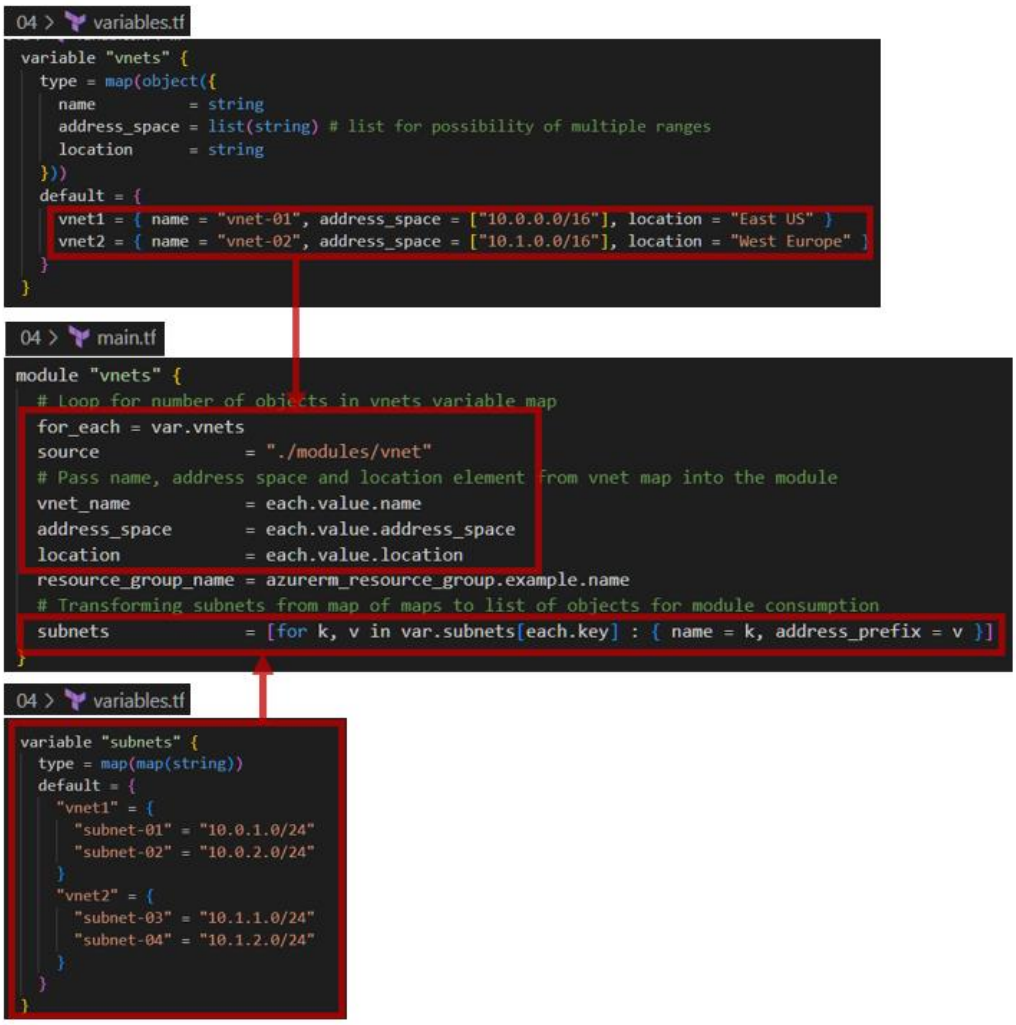
Demonstration Deployment

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'

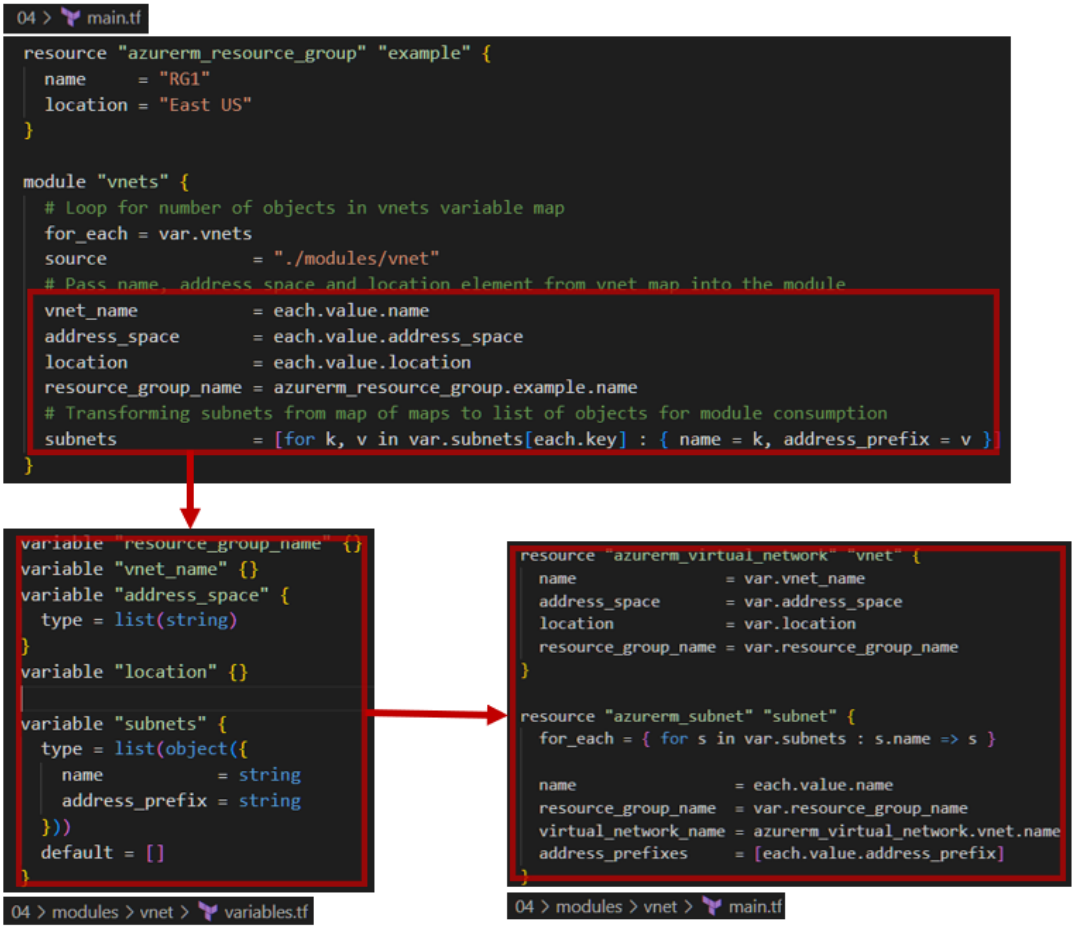
var.subnets in **variables.tf** is a nested map. The outer map uses VNet names as keys and the inner map contains subnets. When calling the module we only pass the subnets that belong to that specific VNet. This is achieved through **list comprehension**, used here in the format:

***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 21-27 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
```

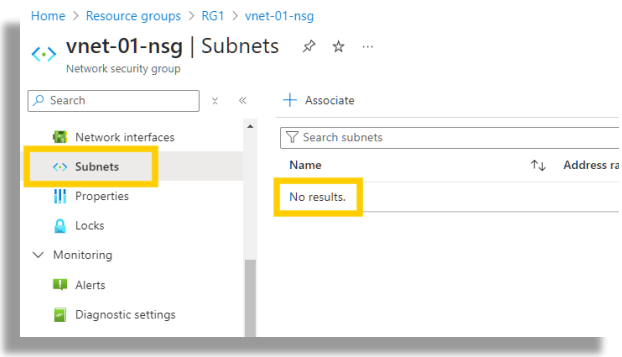
Note how the **nsg** module is now initialized

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 **29-41** 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

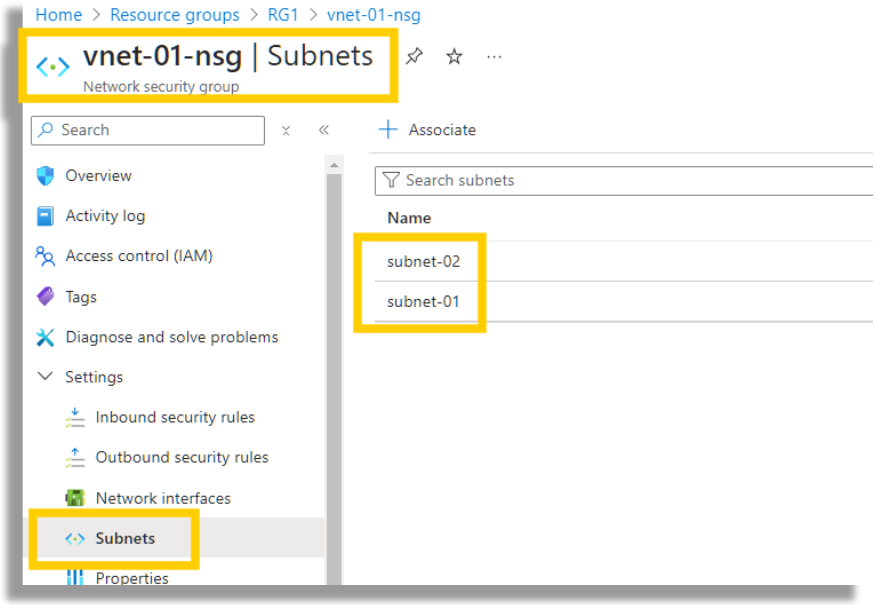
- From the output of the vnets module, loop over each VNet and extract its subnets.
- Store both **VNet name and Subnet ID** in a structured map.
- Flatten the map using merge([...]), allowing **for_each** to iterate over each subnets.
- Dynamically look up the correct NSG based on each.value.vnet

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 **43-53** 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.

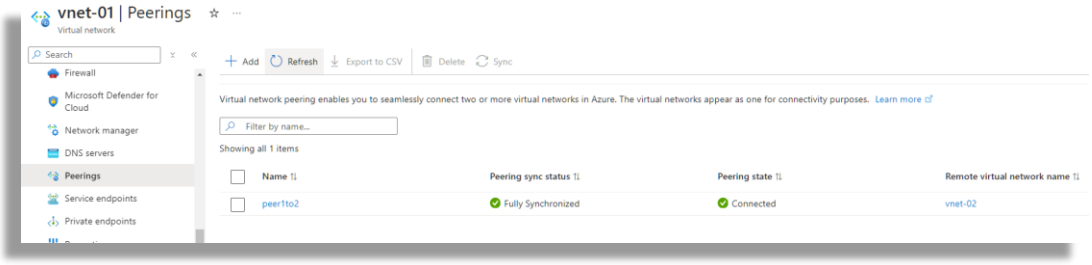
Each instance gets dynamically populated with the correct name, virtual_network_name, and remote_virtual_network_id

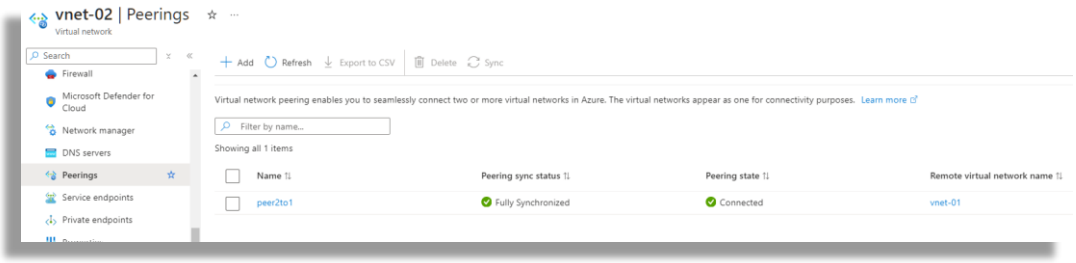
Run **terraform plan** and then **apply**

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