Terraform Day2 Notes

1.Creating VirtualNetwork and Subnet using List and Map.

```
/*

The following links provide the documentation for the new blocks used
in this terraform configuration file

1.azurerm_network_interface -
https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/r
esources/network_interface

*/

terraform {
  required_providers {
    azurerm = {
      source = "hashicorp/azurerm"
      version = "3.10.0"
    }
  }
}

provider "azurerm" {
  subscription_id = "xxxxxxxxxxxxxxxxxxxxx"
  tenant_id = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
  client_id = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
  client_secret = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
  features {}
}

locals {
  resource_group_name="app-grp"
  location="North Europe"
  virtual_network={
    name="app-network"
    address_space="10.0.0.0/16"
  }

  subnets=[
    {
      name="subnetA"
```

```
      address_prefix="10.0.0.0/24"
    },
    {
      name="subnetB"
      address_prefix="10.0.1.0/24"
    }
  ]
}

resource "azurerm_resource_group" "appgrp" {
  name     = local.resource_group_name
  location = local.location
}

resource "azurerm_virtual_network" "appnetwork" {
  name                = local.virtual_network.name
  location            = local.location
  resource_group_name = local.resource_group_name
  address_space       = [local.virtual_network.address_space]

   depends_on = [
     azurerm_resource_group.appgrp
   ]
  }


  resource "azurerm_subnet" "subnetA" {
  name                 = local.subnets[0].name
  resource_group_name  = local.resource_group_name
  virtual_network_name = local.virtual_network.name
  address_prefixes     = [local.subnets[0].address_prefix]
  depends_on = [
    azurerm_virtual_network.appnetwork
  ]
}

resource "azurerm_subnet" "subnetB" {
  name                 = local.subnets[1].name
  resource_group_name  = local.resource_group_name
  virtual_network_name = local.virtual_network.name
  address_prefixes     = [local.subnets[1].address_prefix]
  depends_on = [
    azurerm_virtual_network.appnetwork
```

```
  ]
}

resource "azurerm_network_interface" "appinterface" {
  name                = "appinterface"
  location            = local.location
  resource_group_name = local.resource_group_name

  ip_configuration {
    name                          = "internal"
    subnet_id                     = azurerm_subnet.subnetA.id
    private_ip_address_allocation = "Dynamic"
  }
  depends_on = [
    azurerm_subnet.subnetA
  ]
}
```

2.What is List and Map in Terraform

In Terraform, list and map are two data types used to store collections of values.

A list is an ordered collection of values of the same type. You can define a list in Terraform using square brackets [], and each value in the list is separated by a comma. For example:

list_of_strings = ["value1", "value2", "value3"]
A map is an unordered collection of key-value pairs, where each key is unique and associated with a value. You can define a map in Terraform using curly braces {}, and each key-value pair is separated by an = symbol. For example:

map_of_strings = {
  key1 = "value1"
  key2 = "value2"
  key3 = "value3"
}
Both list and map are useful for representing complex data structures in Terraform, and can be used as inputs to Terraform resources, modules, and outputs.

3.Another way of creating Subnet.

```
terraform {
  required_providers {
    azurerm = {
      source = "hashicorp/azurerm"
      version = "3.10.0"
    }
  }
}

provider "azurerm" {
  subscription_id = "xxxxxxxxxxxxxxxxxxxxxxxx"
  tenant_id = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
  client_id = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
  client_secret = "xxxxxxxxxxxxxxxxxxxxxxxxxxxx"
  features {}
}

locals {
  resource_group_name="app-grp"
  location="North Europe"
  virtual_network={
    name="app-network"
    address_space="10.0.0.0/16"
  }

  subnets=[
    {
      name="subnetA"
      address_prefix="10.0.0.0/24"
    },
    {
      name="subnetB"
      address_prefix="10.0.1.0/24"
    }
  ]
}

resource "azurerm_resource_group" "appgrp" {
  name     = local.resource_group_name
  location = local.location
}
```

```
resource "azurerm_virtual_network" "appnetwork" {
  name                = local.virtual_network.name
  location            = local.location
  resource_group_name = local.resource_group_name
  address_space       = [local.virtual_network.address_space]

   depends_on = [
     azurerm_resource_group.appgrp
   ]
  }


  resource "azurerm_subnet" "subnetA" {
  name                 = local.subnets[0].name
  resource_group_name  = local.resource_group_name
  virtual_network_name = local.virtual_network.name
  address_prefixes     = [local.subnets[0].address_prefix]
  depends_on = [
    azurerm_virtual_network.appnetwork
  ]
}

resource "azurerm_subnet" "subnetB" {
  name                 = local.subnets[1].name
  resource_group_name  = local.resource_group_name
  virtual_network_name = local.virtual_network.name
  address_prefixes     = [local.subnets[1].address_prefix]
  depends_on = [
    azurerm_virtual_network.appnetwork
  ]
}
```

4.Creating Virtual Network Interface

```
terraform {
  required_providers {
    azurerm = {
      source = "hashicorp/azurerm"
      version = "3.10.0"
    }
```

```
  }
}

provider "azurerm" {
  subscription_id = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
  tenant_id = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
  client_id = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
  client_secret = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
  features {}
}

locals {
  resource_group_name="app-grp"
  location="North Europe"
  virtual_network={
    name="app-network"
    address_space="10.0.0.0/16"
  }

  subnets=[
    {
      name="subnetA"
      address_prefix="10.0.0.0/24"
    },
    {
      name="subnetB"
      address_prefix="10.0.1.0/24"
    }
  ]
}

resource "azurerm_resource_group" "appgrp" {
  name     = local.resource_group_name
  location = local.location
}

resource "azurerm_virtual_network" "appnetwork" {
  name                = local.virtual_network.name
  location            = local.location
  resource_group_name = local.resource_group_name
  address_space       = [local.virtual_network.address_space]

   depends_on = [
```

```
    azurerm_resource_group.appgrp
  ]
 }


 resource "azurerm_subnet" "subnetA" {
 name                 = local.subnets[0].name
 resource_group_name  = local.resource_group_name
 virtual_network_name = local.virtual_network.name
 address_prefixes     = [local.subnets[0].address_prefix]
 depends_on = [
   azurerm_virtual_network.appnetwork
 ]
}

resource "azurerm_subnet" "subnetB" {
 name                 = local.subnets[1].name
 resource_group_name  = local.resource_group_name
 virtual_network_name = local.virtual_network.name
 address_prefixes     = [local.subnets[1].address_prefix]
 depends_on = [
   azurerm_virtual_network.appnetwork
 ]
}

resource "azurerm_network_interface" "appinterface" {
 name                = "appinterface"
 location            = local.location
 resource_group_name = local.resource_group_name

 ip_configuration {
   name                          = "internal"
   subnet_id                     = azurerm_subnet.subnetA.id
   private_ip_address_allocation = "Dynamic"
 }
 depends_on = [
   azurerm_subnet.subnetA
 ]
}
```

5.use of terraform validate

The **terraform validate** command is used to validate the syntax and overall structure of a Terraform configuration. It checks the configuration files for proper syntax, verifies that all required arguments are provided, and confirms that the configuration is logically structured.

The **validate** command is a good first step to take when troubleshooting issues with your Terraform code, as it helps identify syntax errors, missing required arguments, or other structural issues that could prevent Terraform from functioning correctly.

Using **terraform validate** is also a best practice for ensuring the quality of your Terraform code before applying changes to a real environment. It's a quick and easy way to catch mistakes early, before they cause bigger problems down the line.

Here's an example of how you can use terraform validate in your terminal:

$ terraform validate

If the syntax and structure of your Terraform code are correct, **terraform validate** will complete with no errors and output nothing. If there are errors or warnings, they will be displayed in the terminal.

6.what is Terraform Destroy and use of it

The **terraform destroy** command is used to destroy the resources that were created by Terraform. This command will remove all the resources and clean up any associated configurations that were created by Terraform.

The **destroy** command is useful for tearing down resources that are no longer needed or for cleaning up after a failed Terraform run. It can also be used to decommission resources in a test environment before destroying the entire environment.

It is important to use **terraform destroy** with caution, as it will permanently delete resources and their associated data. Before running **terraform destroy**, it's recommended to run a terraform plan to see what resources will be deleted, and to make sure that no important data will be lost.

Here's an example of how you can use terraform destroy in your terminal:

$ terraform destroy

This will prompt Terraform to destroy all the resources that were created by Terraform. The command will prompt you to confirm the destruction before proceeding. After confirming, Terraform will execute the destruction process, removing all the resources and cleaning up any associated configurations.