Terraform Notes

What is Terraform?

- Hashicorp Terraform is an **open-source IaC** (Infrastructure-as-Code) tool for provisioning and managing cloud infrastructure.
- It **codifies** infrastructure in configuration files that describe the **desired state** for your Infrastructure topology.
- Terraform enables the management of any infrastructure - such as public clouds, private clouds, and SaaS services - by using **Terraform providers**.

Manual versus Automated Deployments

Using Terraform has several advantages over manually managing your infrastructure:

- Terraform can manage infrastructure on **multiple cloud platforms**.
- The **human-readable configuration language, HashiCorp Configuration Language (HCL)** helps you write infrastructure code quickly.
- Terraform's state allows you to **track resource changes** throughout your deployments.
- You can commit your configurations to **version control** to safely collaborate on infrastructure.

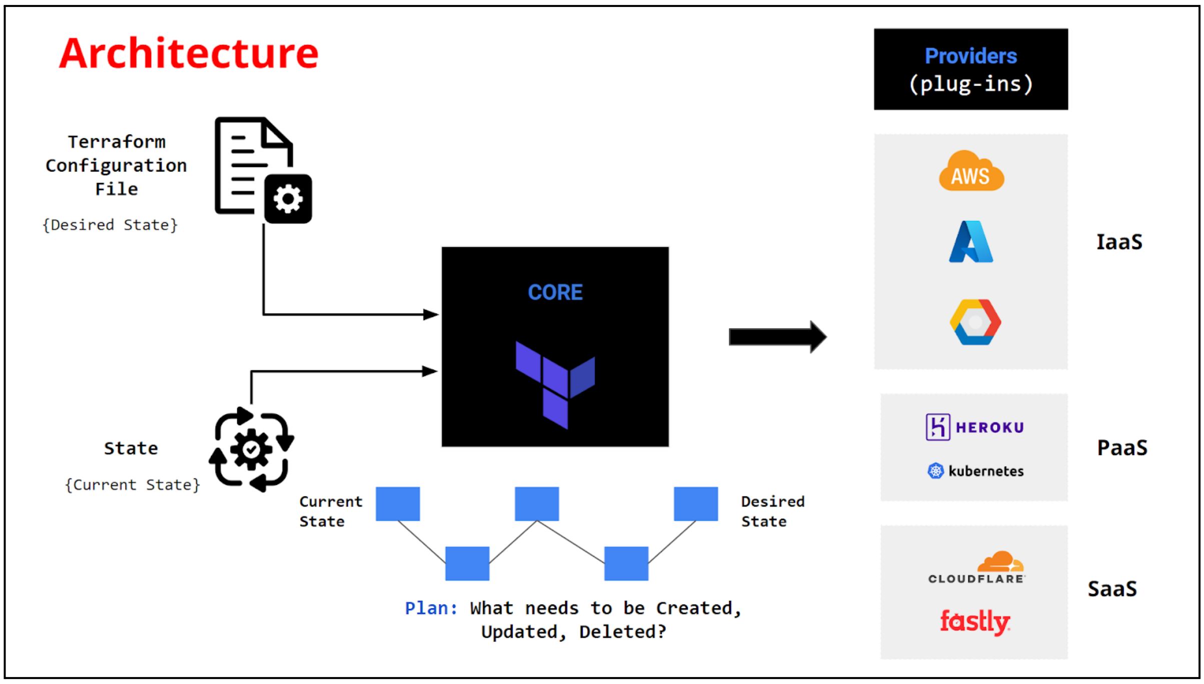
Manage any infrastructure

- Terraform plugins called **providers** let Terraform interact with cloud platforms and other services via their application programming interfaces (APIs).
- HashiCorp and the Terraform community have written over 1,000 providers to manage resources on Amazon Web Services (AWS), **Azure**, Google Cloud Platform (GCP), Kubernetes, Helm, GitHub, Splunk, and DataDog, just to name a few.
- Find providers for many of the platforms and services you already use in the Terraform Registry.
- If you don't find the provider you're looking for, you can write your own.

Terraform Providers for Azure infrastructure

- Terraform users provision their infrastructure on the major cloud providers such as AWS, Azure, GCP etc.

- **A provider is a plugin** that interacts with the various APIs required to create, update, and delete various resources.



There are several Terraform providers that enable the management of Azure infrastructure:

AzureRM

- Manage stable Azure resources and functionality such as virtual machines, storage accounts, and networking interfaces.

AzureAD

- Manage Azure Active directory resources such as groups, users, service principals, and applications.

AzureDevops

- Manage Azure DevOps resources such as agents, repositories, projects, pipelines, and queries.

AzAPI

- Manage Azure resources and functionality using the Azure Resource Manager APIs directly.

- This provider compliments the AzureRM provider by enabling the management of Azure resources that aren't released.

Azure Stack

- Manage Azure Stack resources such as virtual machines, DNS, VNet, and storage.

To speak with Azure Cloud, create a file named providers.tf and insert the following code:

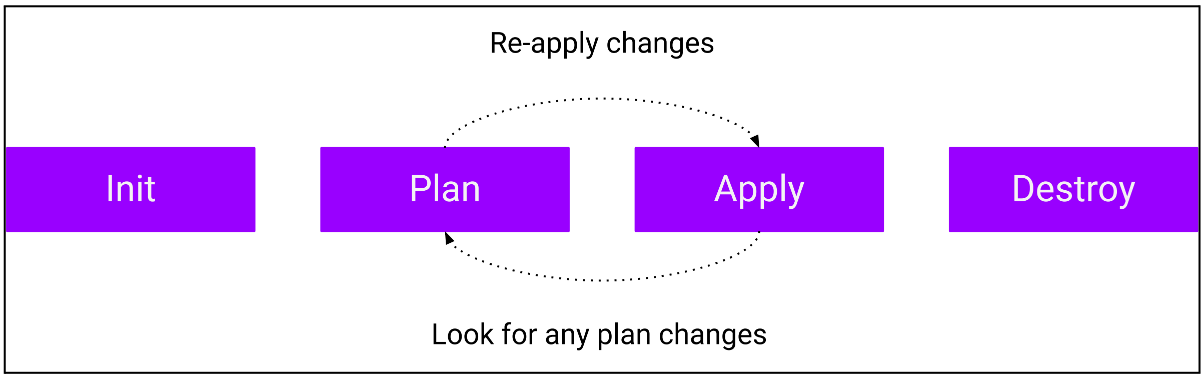
```
Unset
# Azure Provider source and version being used
terraform {
 required_version = ">=0.12"

 required_providers {
  azurerm = {
   source = "hashicorp/azurerm"
   version = "~>2.0"
  }
  random = {
   source = "hashicorp/random"
   version = "~>3.0"
  }
 }
}

# Configure the Microsoft Azure Provider
provider "azurerm" {
 features {}
}
```

To speak with AWS Cloud, create a file named providers.tf and insert the following code:

```
Unset
# AWS Provider source and version being used
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
  }
}
```

```
# Configure the AWS Provider
provider "aws" {
  region = "us-east-1"
}
```

Workflow



- **Initialize -** Install the plugins Terraform needs to manage the infrastructure.
- **Plan -** Preview the changes Terraform will make to match your configuration.
- **Apply -** Make the planned changes.

Track your infrastructure

- Terraform keeps track of your real infrastructure in a **state file**, which acts as a source of truth for your environment.
- Terraform uses the state file to determine the changes to make to your infrastructure so that it will match your configuration.

What is Terraform State?

- Terraform logs information about the resources it has created in a state file.
- This enables Terraform to know which resources are under its control and when to update and destroy them.
- The terraform state file, by default, is named **terraform.tfstate** and is held in the same directory where Terraform is run.
- It is created after running terraform apply.

- The actual content of this file is a JSON formatted mapping of the resources defined in the configuration and those that exist in your infrastructure.
- When Terraform is run, it can then use this mapping to compare infrastructure to the code and make any adjustments as necessary.

Terraform Files

- **provider.tf** – connecting to cloud platforms and authenticating.
- **main.tf** – containing the resource blocks which define the resources to be created in the target cloud platform.
- **variables.tf** – containing the variable declarations used in the resource blocks.
- **output.tf** – containing the output that needs to be generated on successful completion of “apply” operation.
- ***.tfvars** – containing the environment-specific default values of variables.

Lab: Installing on Windows and connect to Azure

- Install the New PowerShell Az Module
- Install the Azure CLI
- Install Terraform
- Authenticate Terraform to Azure
- Create a service principal using Azure PowerShell
- Specify service principal credentials in environment variables
- Specify service principal credentials in a Terraform provider block

Install the PowerShell Az Module

```
Unset
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope
CurrentUser

Install-Module -Name PowerShellGet -Force

Install-Module -Name Az -Force
```

Install the Azure CLI

```
Unset
winget install -e --id Microsoft.AzureCLI

or

$ProgressPreference = 'SilentlyContinue'; Invoke-WebRequest
-Uri https://aka.ms/installazurecliwindows -OutFile
.\AzureCLI.msi; Start-Process msiexec.exe -Wait -ArgumentList
'/I AzureCLI.msi /quiet'; Remove-Item .\AzureCLI.msi
```

Install Terraform

Download Terraform:
https://releases.hashicorp.com/terraform/1.5.6/terraform_1.5.6_windows_amd64.zip

Find more details on versions: https://github.com/hashicorp/terraform/releases

From the download, extract the executable to a directory of your choosing (for example, **c:\terraform**).

- Update your **system's global path** to the executable.
- Go to **RUN** and enter: **SystemPropertiesAdvanced**
- Scroll down in **system variables** until you find **PATH**.
- Click **edit and change** accordingly.

Open a terminal window and verify the version:

```
Unset
terraform -version
```

Authenticate Terraform to Azure

- Terraform only supports authenticating to Azure via the Azure CLI.
- Authenticating using Azure PowerShell isn't supported.
- Therefore, while you can use the Azure PowerShell module when doing your Terraform work, you first need to authenticate to Azure using the Azure CLI.

```
Unset
az login
```

To confirm the current Azure subscription:

```
Unset
az account show
```

Create a service principal

- Automated tools that deploy or use Azure services - such as Terraform - should always have restricted permissions.
- Instead of having applications sign in as a fully privileged user, Azure offers **service principals**.
- The most common pattern is to interactively sign in to Azure, create a service principal, test the service principal, and then use that service principal for future authentication (either interactively or from your scripts).

```
Unset
Connect-AzAccount
Get-AzContext
$sp = New-AzADServicePrincipal -DisplayName sp-terraform -Role
"Contributor"
```

Display the service principal ID:

Make note of the service principal application ID as it's needed to use the service principal.

```
Unset
$sp.AppId
```

Get the auto generated password to text:

```
Unset
$sp.PasswordCredentials.SecretText
```

Make note of the password as it's needed to use the service principal. The password can't be retrieved if lost. As such, you should store your password in a safe place.

If you forget your password, you can reset the service principal credentials:

```
Unset
Get-AzAdServicePrincipal -DisplayName sp-terraform

Remove-AzADSpCredential -DisplayName sp-terraform

$newCredential = New-AzADSpCredential -ObjectId
04713a06-636a-40b5-919b-94ed6c9d1e73

$newCredential.SecretText
```

Note and save that password.

Specify service principal credentials in environment variables

Once you create a service principal, you can specify its credentials to Terraform via environment variables.

DONOT FOLLOW THIS

```
Unset
Get-AzAdServicePrincipal -DisplayName sp-terraform

$env:ARM_CLIENT_ID="<service_principal_app_id>"
$env:ARM_SUBSCRIPTION_ID="<azure_subscription_id>"
$env:ARM_TENANT_ID="<azure_subscription_tenant_id>"
$env:ARM_CLIENT_SECRET="<service_principal_password>"

az account show

or

Get-AzSubscription

$env:ARM_CLIENT_ID="7c42fad7-effb-4567-b801-fbb0b25ed60d"
```

```
$env:ARM_SUBSCRIPTION_ID="9106772c-0e98-4a3a-a2d2-952c510016c9
"
$env:ARM_TENANT_ID="4954dd78-1aa7-45fd-97e9-8048247d270b"
$env:ARM_CLIENT_SECRET="~j48Q~WKgm14WxCbdzpa_xGND8WPNPxJuEGw.c
7q"
```

Run the following PowerShell command to verify the Azure environment variables:

```
Unset
gci env:ARM_*
```

Specify service principal credentials in a Terraform provider block

Usual way

```
Unset
terraform {
  required_providers {
    azurerm = {
      source = "hashicorp/azurerm"
      version = "~>2.0"
    }
  }
}

provider "azurerm" {
  features {}

  subscription_id   = "<azure_subscription_id>"
  tenant_id         = "<azure_subscription_tenant_id>"
  client_id         = "<service_principal_appid>"
  client_secret     = "<service_principal_password>"
}
```

Safe way

Unset
```
terraform {
  required_providers {
    azurerm = {
      source = "hashicorp/azurerm"
      version = "~>2.0"
    }
  }
}

provider "azurerm" {
  features {}

  subscription_id   = "${env.ARM_SUBSCRIPTION_ID}"
  tenant_id         = "$env:ARM_TENANT_ID"
  client_id         = "$env:ARM_CLIENT_ID"
  client_secret     = "$env:ARM_CLIENT_SECRET"
}
```

Lab: Create a Resource group

Create a folder: c:\terraform-scripts
Create a folder: c:\terraform-scripts\rg

In PowerShell:
Go to folder rg: c:\terraform-scripts\rg

Create a file named providers.tf and insert the following code:

Unset
```
terraform {
  required_providers {
    azurerm = {
      source = "hashicorp/azurerm"
      version = "~>2.0"
    }
  }
}

provider "azurerm" {
  features {}

  subscription_id   = "9106772c-0e98-4a3a-a2d2-952c510016c9"
  tenant_id         = "4954dd78-1aa7-45fd-97e9-8048247d270b"
  client_id         = "7c42fad7-effb-4567-b801-fbb0b25ed60d"
  client_secret     =
"~j48Q~WKgm14WxCbdzpa_xGND8WPNPxJuEGw.c7q"
}
```

Create a file named main.tf and insert the following code:

Unset
```
resource "random_pet" "rg_name" {
  prefix = var.resource_group_name_prefix
}

resource "azurerm_resource_group" "rg" {
  location = var.resource_group_location
  name     = random_pet.rg_name.id
}
```

Create a file named variables.tf and insert the following code:

Unset
```
variable "resource_group_location" {
  default     = "eastus"
  description = "Location of the resource group."
}

variable "resource_group_name_prefix" {
  default     = "rg"
  description = "Prefix of the resource group name that's
combined with a random ID so name is unique in your Azure
subscription."
}
```

Create a file named outputs.tf and insert the following code:

```
Unset
output "resource_group_name" {
  value = azurerm_resource_group.rg.name
}
```

Initialize Terraform:

```
Unset
terraform init -upgrade
```

Create a Terraform execution plan:

```
Unset
terraform plan -out main.tfplan
```

Apply a Terraform execution plan:

```
Unset
terraform apply main.tfplan
```

Verify the results:

```
Unset
echo "$(terraform output resource_group_name)"

# Using CLI
$rgname= echo "$(terraform output resource_group_name)"
az group show --name $rgname

# Using Azure PowerShell
Get-AzResourceGroup -Name <resource_group_name>
```

Lab: Create a VM

Create a folder: c:\terraform-scripts
Create a folder: c:\terraform-scripts\vm

In PowerShell:
Go to folder rg: c:\terraform-scripts\vm

Create a file named providers.tf and insert the following code:

```
Unset
terraform {
  required_providers {
    azurerm = {
      source = "hashicorp/azurerm"
      version = "~>2.0"
    }
  }
}

provider "azurerm" {
  features {}
  subscription_id   = "9106772c-0e98-4a3a-a2d2-952c510016c9"
  tenant_id         = "4954dd78-1aa7-45fd-97e9-8048247d270b"
  client_id         = "7c42fad7-effb-4567-b801-fbb0b25ed60d"
  client_secret     =
"~j48Q~WKgm14WxCbdzpa_xGND8WPNPxJuEGw.c7q"

}
```

Create a file named main.tf and insert the following code:

```
Unset
resource "azurerm_resource_group" "rg" {
  name     = "rg-webservers"
  location = "Central India"
}


resource "azurerm_virtual_network" "vnet" {
  name                = "vnet-centralindia"
  address_space       = ["10.0.0.0/16"]
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
}

resource "azurerm_subnet" "subnet" {
  name                 = "default"
  resource_group_name  = azurerm_resource_group.rg.name
  virtual_network_name = azurerm_virtual_network.vnet.name
  address_prefixes     = ["10.0.2.0/24"]
}

resource "azurerm_public_ip" "public_ip" {
  name                = "vm1_public_ip"
  resource_group_name = azurerm_resource_group.rg.name
  location            = azurerm_resource_group.rg.location
  allocation_method   = "Dynamic"
}

resource "azurerm_network_interface" "nic" {
  name                = "vm1-nic"
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name

  ip_configuration {
    name                          = "internal"
    subnet_id                     = azurerm_subnet.subnet.id
    private_ip_address_allocation = "Dynamic"
    public_ip_address_id = azurerm_public_ip.public_ip.id
  }
}

resource "azurerm_network_security_group" "nsg" {
  name                = "ssh_nsg"
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name

  security_rule {
    name                       = "allow_ssh_sg"
    priority                   = 100
    direction                  = "Inbound"
    access                     = "Allow"
    protocol                   = "Tcp"
    source_port_range          = "*"
    destination_port_range     = "22"
    source_address_prefix      = "*"
    destination_address_prefix = "*"
  }
}

resource
"azurerm_network_interface_security_group_association"
"association" {
  network_interface_id      = azurerm_network_interface.nic.id
  network_security_group_id =
azurerm_network_security_group.nsg.id
}

resource "azurerm_linux_virtual_machine" "vm" {
  name                = "vm1"
  resource_group_name = azurerm_resource_group.rg.name
  location            = azurerm_resource_group.rg.location
  size                = "Standard_B1s"
  admin_username      = "linuxadmin"

  source_image_reference {
      publisher = "Canonical"
      offer     = "0001-com-ubuntu-server-jammy"
      sku       = "22_04-lts-gen2"
      version   = "latest"
    }

  network_interface_ids = [
    azurerm_network_interface.nic.id,
  ]

  admin_ssh_key {
    username   = "linuxadmin"
    public_key = file("id_rsa.pub")
  }

  os_disk {
    caching              = "ReadWrite"
    storage_account_type = "Standard_LRS"
  }
```

```
}

output "public_ip" {
  value = azurerm_linux_virtual_machine.vm.public_ip_address
}
```

Initialize Terraform:

```
Unset
terraform init -upgrade
```

Create a Terraform execution plan:

```
Unset
terraform plan -out main.tfplan
```

Apply a Terraform execution plan:

```
Unset
terraform apply main.tfplan
```

Verify the results:

```
Unset
Get-AzVM
```