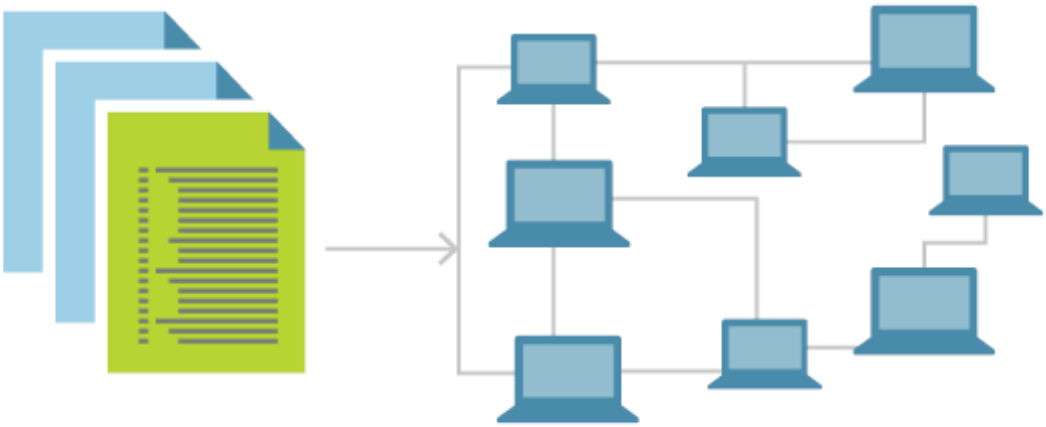
13 Feb 2025

Project - 4 - IaC - Infrastructure As Code

Infrastructure As Code - IaC

- Infrastructure as code is the **process** of **managing(update/delete) and provisioning(creating) computer data center(on the cloud) resources networks, firewalls, storage devices, servers, databases etc** through **CODE**



IaC Concept

- Infrastructure as Code (or IaC) is an **automated type of infrastructure** management.

- The IaC model approaches **networks**, **virtual machines**, load balancers, and connection topology in a descriptive manner via **versioning** (similar to using VCS in Development, in **DevOps we are tracking infrastructure code**).

- IaC is a key DevOps practice.

- Basically, **Infrastructure as Code is a model that codifies everything**.
    - Like **Developers** write **Application Code** to **build Applications**
        - Technologies - Java, Python, Node etc

    - As a **DevOps Engineer** you will be **writing Infrastructure Code** to **build Infrastructure**
        - Tools - AWS Cloudformation, Azure ARM, Pulumi, Ansible, **Terraform** etc

IaC Benefits

- IaC boosts productivity through **automation**

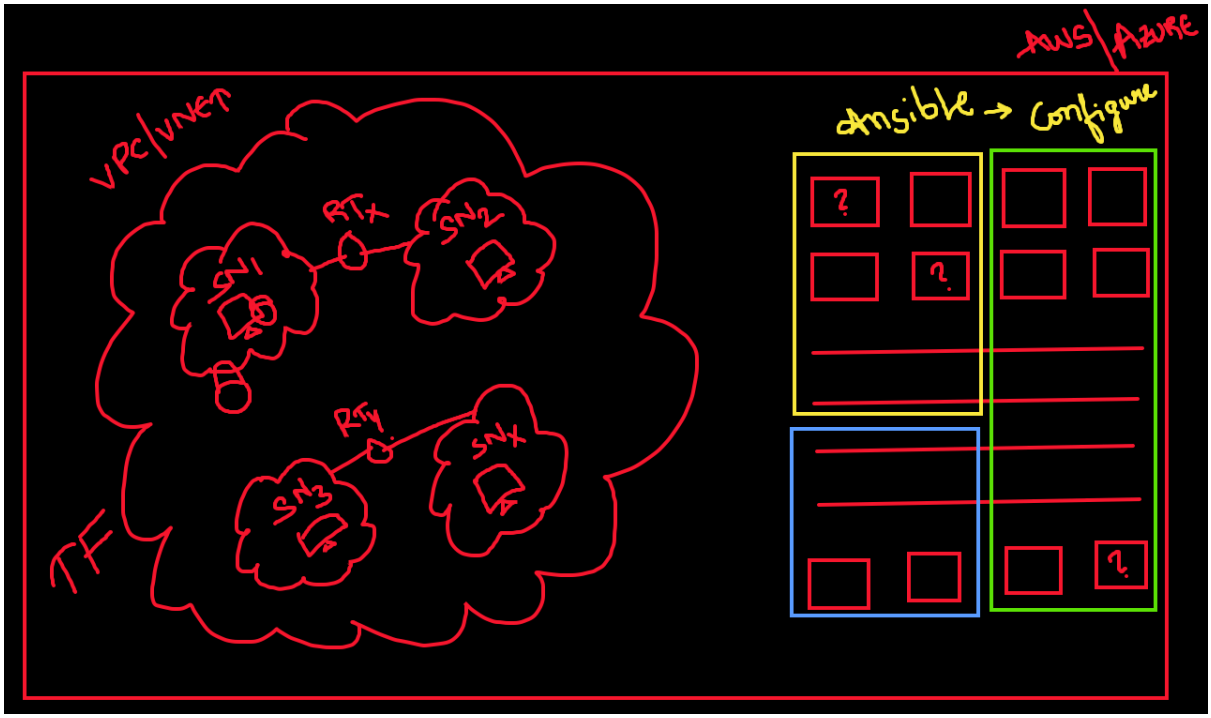
- Replicate your infrastructure easily, **CODE = REUSABILITY**

- **Consistency** in Setup & Configuration

- Minimising risk of human errors

Terraform vs Ansible

Terraform	Ansible
Terraform lays focus on infrastructure provisioning	Ansible fits really well in traditional automation
Terraform acts perfectly in configuring cloud infrastructures.	Ansible is employed for configuring servers.
Using Terraform, one can deploy load balancers, VPCs, etc	With Ansible, you can deploy apps on the top of infrastructure.



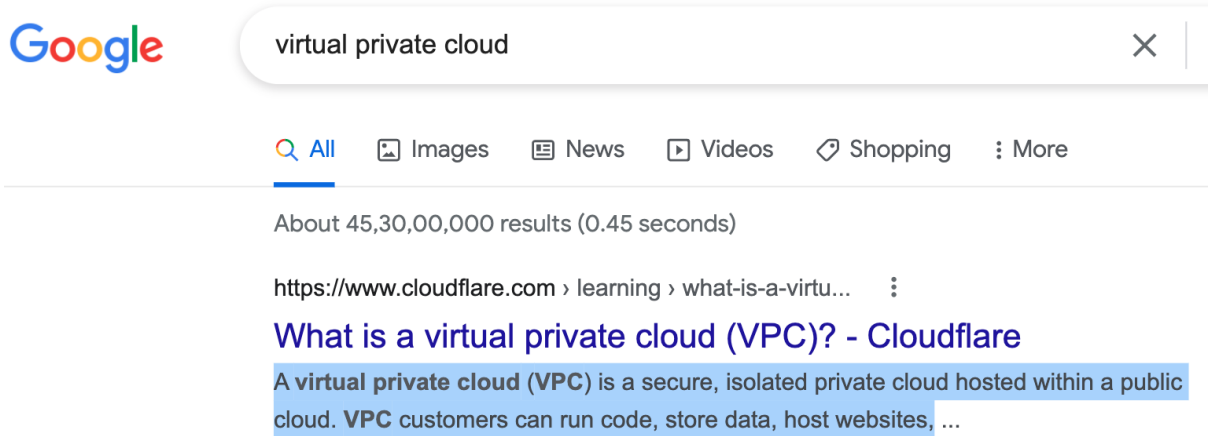
Automate infrastructure on any cloud with Terraform

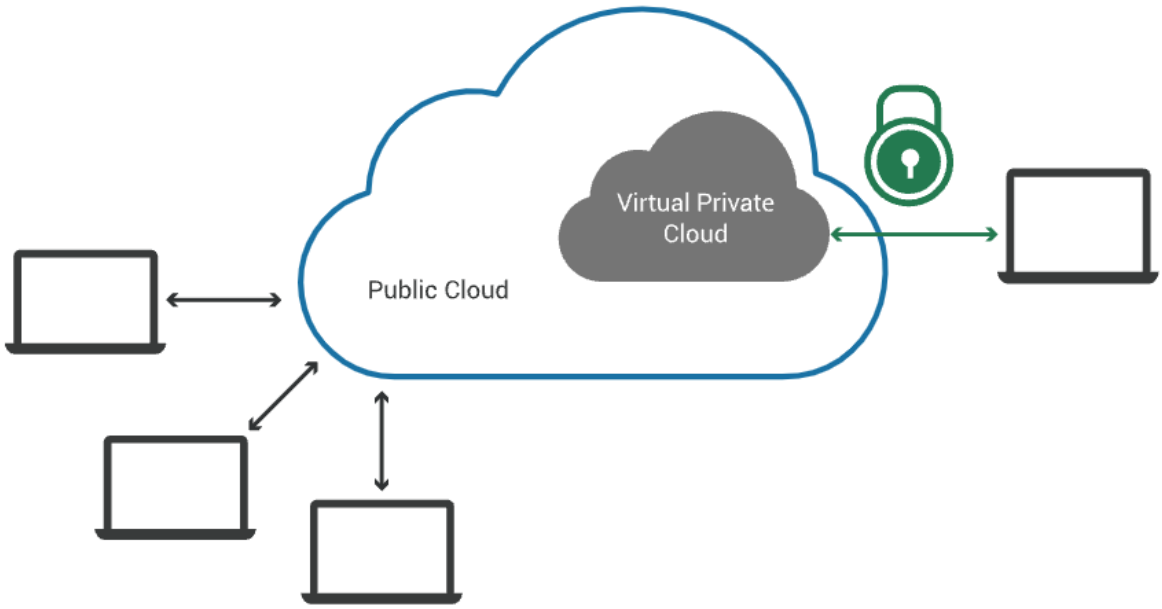
Infrastructure automation to provision and manage resources in any cloud or data center.

Try HCP Terraform

NOTE: Learning Cloud is Must For Terraform

VPC - Virtual Private Cloud (Generic)





AWS VPC is a service that **lets customers provision a logically isolated section of the AWS Cloud** where you can **launch AWS resources (Servers, Database, Applications etc)**

AZURE VNET is a service that **lets customers provision a logically isolated section of the AZURE Cloud** where you can **launch AZURE resources (Servers, Database, Applications etc)**

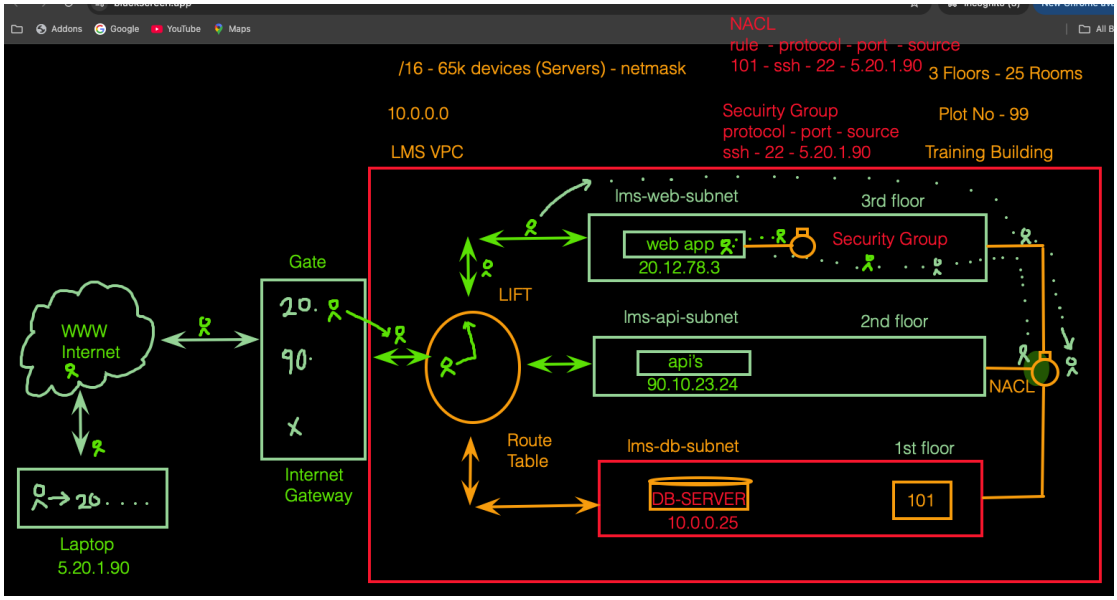
- **Having VPC is similar to having your own data center inside AWS**.
- **Having VENT is similar to having your own data center inside AZURE**.

14 Feb 2025

VPC Components (AWS)

- **Internet Gateway**

- **Subnets**
- **Route Tables**
- **Network Access Control List - NACL**
- **Security Group**



- **Internet Gateway**: A horizontally scalable, highly available VPC component that allows communication between your VPC and the internet. It enables instances in public subnets to access the internet and vice versa.
- **Subnets**: Segments of a VPC's IP address range where you can place groups of resources. Subnets can be public (accessible from the internet) or private (isolated from the internet).
- **Route Tables**: A set of rules (routes) that determine where network traffic from your subnets or gateway is directed. Each subnet must be associated with a route table.
- **Network Access Control List (NACL)**: A stateless firewall at the subnet level that controls inbound and outbound traffic. It uses rules to allow or deny traffic based on IP addresses, ports, and protocols.
- **Security Group**: A stateful firewall at the instance level that controls inbound and outbound traffic. It acts as a virtual firewall for EC2 instances and other AWS resources.

- "**To enable access to or from the internet to an instance in a VPC subnet, you must attach an Internet gateway to your VPC, ensure that your subnet route table points to the Internet gateway and ensure that instance has a public IP address, and ensure that your network access control and security group rules allow the relevant traffic to your instance**" -- **AWS**

17 Feb 2025

LMS VPC SETUP - 45 Mins

- Create VPC
- Create Public Subnets (**Web & API**)
    - Enable Public IP's
- Create Private Subnets (**Database**)
    - DON'T Enable Public IP's
- Create Internet Gateway
- Attach Internet Gateway to VPC
- Create Public Route Table (**Web & API**)
    - Create Internet Route
- Associate Public Subnets to Public Route Table
- Create Private Route Table (**Database**)
    - DON'T Create Internet Route
    - Local Route is already present
- Associate Private Subnets to Private Route Table
- Create NACL's
- Associate NACL with Subnets
- Add Firewall Rules in NACL, both INBOUND & OUTBOUND
- Create Security Groups

    - Web Security Group - 22 / 80
    - API Security Group - 22 / 8080
    - DB Security Group - 22 / 5432
- Add Firewall Rules in Security Groups, INBOUND
- Create EC2 Instances in above VPC
- Deploy Applications on EC2 Instances

GUI - LOGIN VPC SETUP - 45 Mins

- Infrastructure Setup w.r.t LMS
- Frontend
- API
- Database

GUI - ECOMM VPC SETUP - 45 Mins

- Infrastructure Setup w.r.t LMS
- Frontend
- API
- Database

GUI - FOOD VPC SETUP - 45 Mins

- Infrastructure Setup w.r.t LMS
- Frontend
- API
- Database

18 Feb 2025

AWS - CLI

AWS Command Line Interface - CLI

- The AWS Command Line Interface (CLI) is a unified tool to **manage your AWS services**.
- Refer https://aws.amazon.com/cli/
- With just one tool to download and configure, you can control multiple AWS services from the command line.

Setup AWS CLI

```
aws
sudo apt -y update && sudo apt -y install awscli
aws
```

```
aws configure
```

To authenticate and authorise the AWS account, we need to set up **access and secret keys.**

CLI - LMS VPC SETUP - 45 Mins

```
aws ec2 create-vpc --cidr-block 192.168.0.0/16

aws ec2 create-tags --resources vpc-079fddf2c2b18499b --tags
Key=Name,Value=ecomm-vpc

aws ec2 create-subnet --vpc-id vpc-079fddf2c2b18499b --cidr-block
192.168.0.0/24

aws ec2 create-tags --resources subnet-09483a9c7f4b17658 --tags
Key=Name,Value=ecomm-db-subnet

aws ec2 create-subnet --vpc-id vpc-079fddf2c2b18499b --cidr-block
192.168.1.0/24

aws ec2 create-tags --resources subnet-0c6a9deeda4d0f8e7 --tags
Key=Name,Value=ecomm-app-subnet

aws ec2 create-subnet --vpc-id vpc-079fddf2c2b18499b --cidr-block
192.168.2.0/24

aws ec2 create-tags --resources subnet-07a296b55743dc4b6 --tags
Key=Name,Value=ecomm-web-subnet

aws ec2 modify-subnet-attribute --subnet-id subnet-
07a296b55743dc4b6 --map-public-ip-on-launch

aws ec2 modify-subnet-attribute --subnet-id subnet-
0c6a9deeda4d0f8e7 --map-public-ip-on-launch

aws ec2 create-route-table --vpc-id  vpc-079fddf2c2b18499b

aws ec2 create-tags --resources rtb-0bab9e43621d723dc --tags
Key=Name,Value=ecomm-pub-rt
```

```
aws ec2 create-route-table --vpc-id  vpc-079fddf2c2b18499b

aws ec2 create-tags --resources rtb-07114d113d49414c5 --tags
Key=Name,Value=ecomm-pvt-rt

aws ec2 create-internet-gateway

aws ec2 create-tags --resources igw-0caed1c263e4bca07 --tags
Key=Name,Value=ecomm-internet-gateway

aws ec2 attach-internet-gateway --internet-gateway-id  igw-
0caed1c263e4bca07 --vpc-id  vpc-079fddf2c2b18499b

aws ec2 create-route --route-table-id rtb-0bab9e43621d723dc --
destination-cidr-block 0.0.0.0/0 --gateway-id  igw-
0caed1c263e4bca07

aws ec2 associate-route-table --route-table-id rtb-
0bab9e43621d723dc --subnet-id  subnet-0c6a9deeda4d0f8e7


aws ec2 associate-route-table --route-table-id rtb-
0bab9e43621d723dc --subnet-id  subnet-07a296b55743dc4b6

aws ec2 associate-route-table --route-table-id rtb-
07114d113d49414c5 --subnet-id  subnet-09483a9c7f4b17658

aws ec2 create-network-acl --vpc-id vpc-079fddf2c2b18499b

aws ec2 create-tags  --resources  acl-008daa130eb5357fe --tags
Key=Name,Value=ecomm-nacl

aws ec2 create-network-acl-entry --network-acl-id  acl-
008daa130eb5357fe --ingress --rule-number 100 --protocol tcp --
port-range From=0,To=65535 --cidr-block 0.0.0.0/0 --rule-action
allow

aws ec2 create-network-acl-entry --network-acl-id  acl-
008daa130eb5357fe --egress --rule-number 100 --protocol tcp --
port-range From=0,To=65535 --cidr-block 0.0.0.0/0 --rule-action
allow
```

```
aws ec2 describe-network-acls

aws ec2 replace-network-acl-association --association-id
aclassoc-074d26f05acf601b5 --network-acl-id  acl-008daa130eb5357fe

aws ec2 replace-network-acl-association --association-id
aclassoc-0b23acd27316f4825 --network-acl-id  acl-008daa130eb5357fe

aws ec2 replace-network-acl-association --association-id
aclassoc-09622c5537f70572b --network-acl-id  acl-008daa130eb5357fe

aws ec2 create-security-group --group-name ecomm-web-sg --
description "ecomm-web-sg" --vpc-id vpc-079fddf2c2b18499b

aws ec2 authorize-security-group-ingress --group-id sg-
0e9f9cde6a5c985da --protocol tcp  --port 22  --cidr 0.0.0.0/0

aws ec2 authorize-security-group-ingress --group-id sg-
0e9f9cde6a5c985da --protocol tcp  --port 80  --cidr 0.0.0.0/0

aws ec2 create-security-group --group-name ecomm-api-sg --
description "ecomm-api-sg" --vpc-id vpc-079fddf2c2b18499b

aws ec2 create-security-group --group-name ecomm-db-sg --
description "ecomm-db-sg" --vpc-id vpc-079fddf2c2b18499b
```

CLI - LOGIN VPC SETUP - 45 Mins

- Infrastructure Setup w.r.t LMS
- Frontend
- API
- Database

CLI - ECOMM VPC SETUP - 15 Mins

- Infrastructure Setup w.r.t LMS

- Frontend
- API
- Database

CLI - FOOD VPC SETUP - 15 Mins

- Infrastructure Setup w.r.t LMS
- Frontend
- API
- Database

19 Feb 2025

Comparison

1. AWS Management Console (GUI)

- **Pros**:
    - **User-Friendly**: Great for beginners who prefer visual guidance.
    - **Fine-Grained Customization**: Allows detailed configuration with a step-by-step approach.
- **Cons**:
    - **Time-Consuming**: Requires navigating through multiple screens, which increases human effort and the potential for manual errors.
    - **Not Easily Reproducible**: Changes are harder to track and replicate across environments.

2. AWS CLI

- **Pros**:
    - **Faster than GUI**: Commands reduce the steps needed compared to the GUI, making VPC creation quicker.

    - **Scriptable**: Commands can be scripted, allowing for some automation and easier replication.
- **Cons**:
    - **Requires CLI Knowledge**: Users need to know or reference specific commands, which can still be error-prone.
    - **Limited Reusability**: While it's more efficient than the GUI, manual command execution is still required each time.

3. Code - Infrastructure as Code (Terraform)

- **Pros**:
    - **Reusable and Modular**: Terraform code can be modularized and reused across multiple environments.
    - **Automated and Efficient**: Infrastructure can be provisioned, modified, and destroyed with minimal manual intervention, reducing human effort and the potential for errors.
    - **Version Control**: Code can be stored in repositories, enabling version control, auditing, and collaboration.
- **Cons**:
    - **Learning Curve**: Users must learn Terraform syntax and configuration.

Terraform

- Terraform is an **Infrastructure As Code [IaC]** software from **HashiCorp**.

Terraform

Software

Terraform is an open-source infrastructure as code software tool created by HashiCorp. Users define and provision data center infrastructure using a declarative configuration language known as HashiCorp Configuration Language, or optionally JSON. Wikipedia

Stable release: 0.13.5 / October 21, 2020; 36 days ago

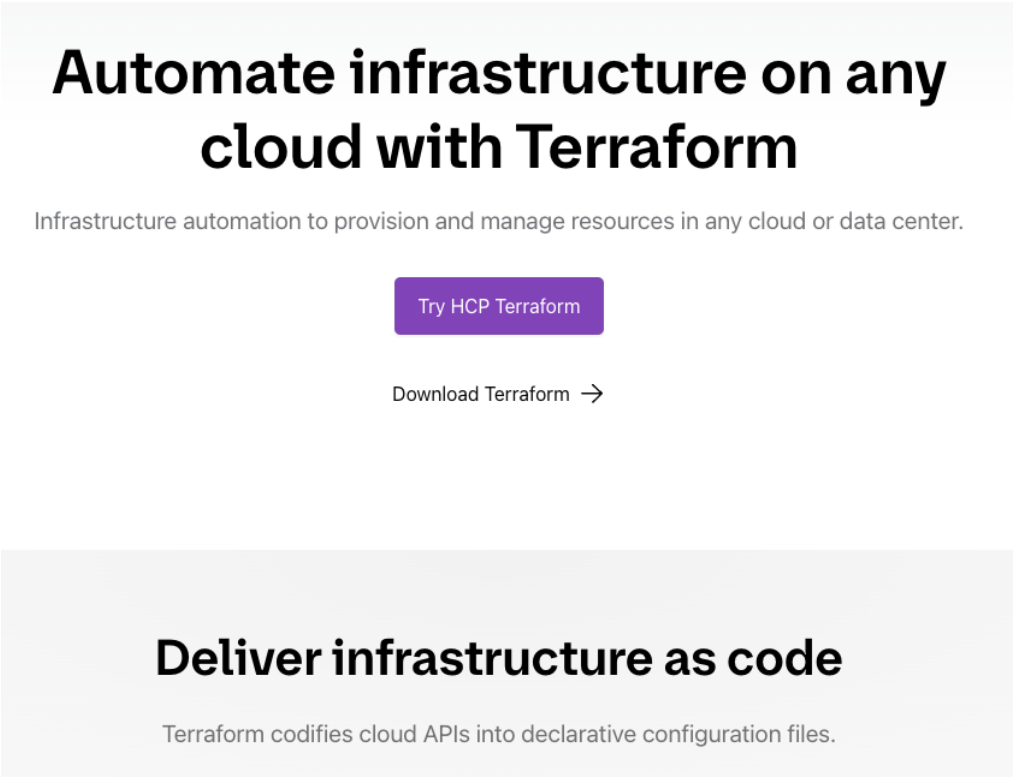
Initial release: July 28, 2014; 6 years ago

License: Mozilla Public License v2.0

Operating system: Linux, FreeBSD, macOS, OpenBSD, Solaris, and Microsoft Windows

Developer: HashiCorp

Programming language: Go

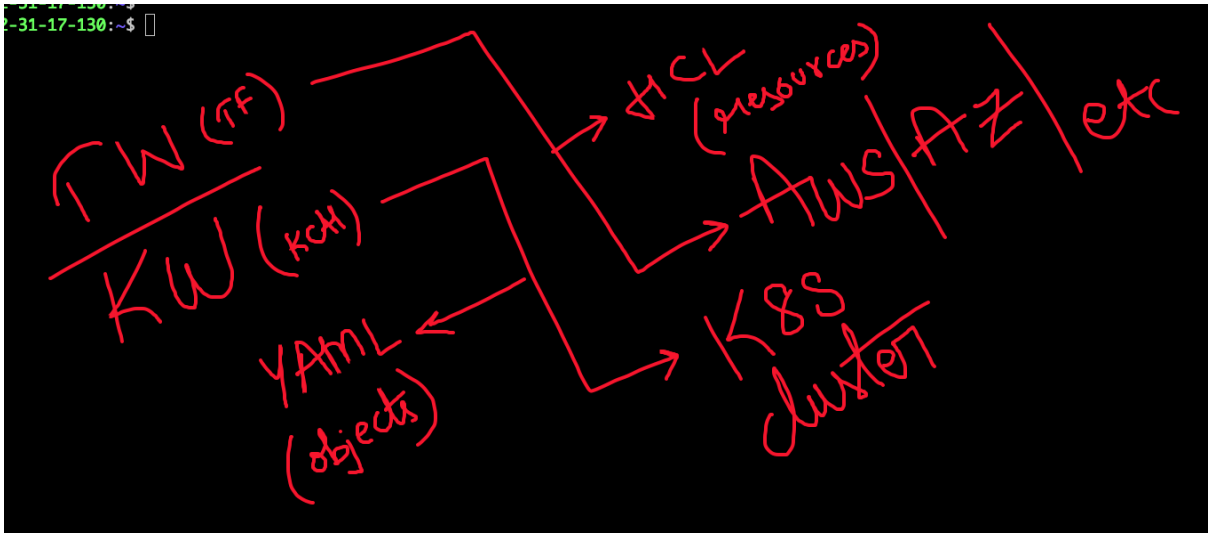


Terraform Setup

- Launch an Server and Setup terraform
- https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli

Tools Needed

- **Git Bash - Version Controlling**
- **Visual Studio Code - IDE**



Terraform Providers

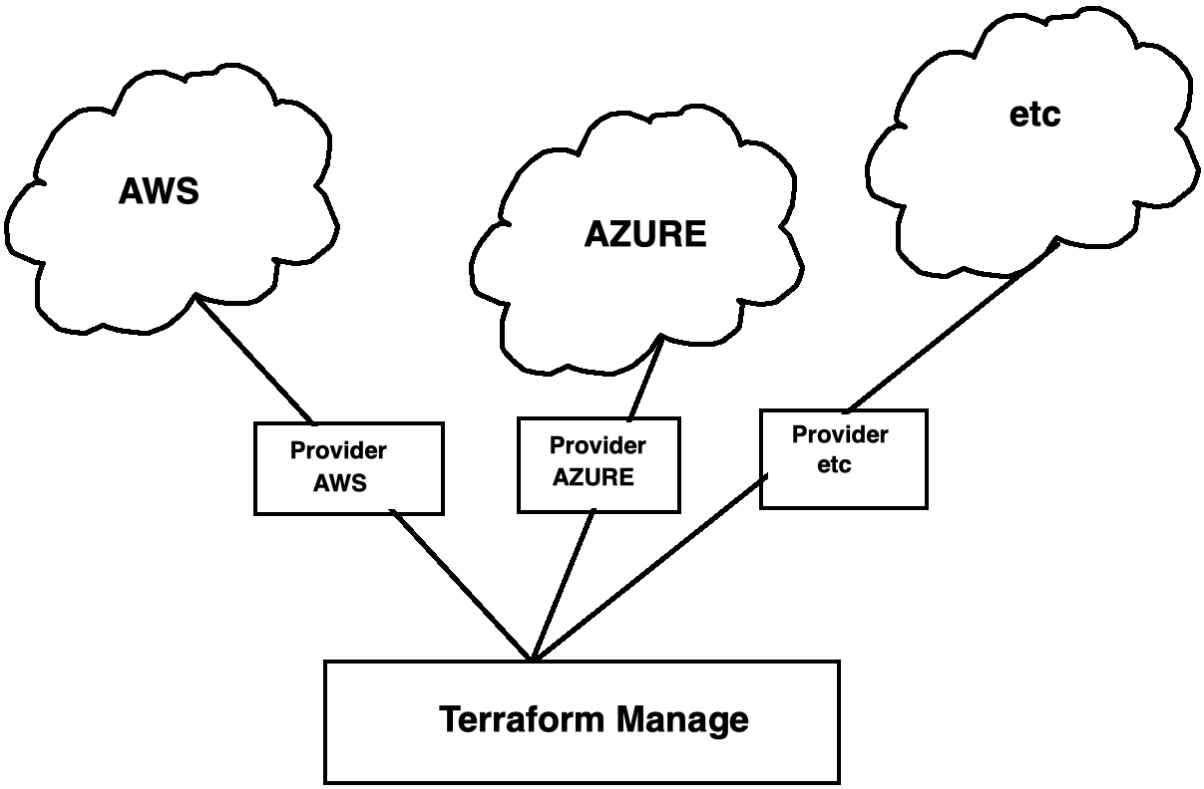
- Terraform relies on **plugins** called "**providers**" to interact with remote systems i.e **cloud vendors**.

- https://registry.terraform.io/browse/providers

- In Terraform, **providers** allow you to interact with APIs of various cloud platforms and services. They define the resources and data sources available for use in your configurations.

- Each provider is responsible for understanding API interactions with a specific service or platform (like AWS, Azure, GCP, etc.).
- Terraform configurations **must** declare **which providers(azure,aws,etc)** they require so that Terraform can install and use them.
- Each provider adds a set of resource types

- Every resource type is implemented by a provider, **without providers, Terraform can't manage any kind of infrastructure**.

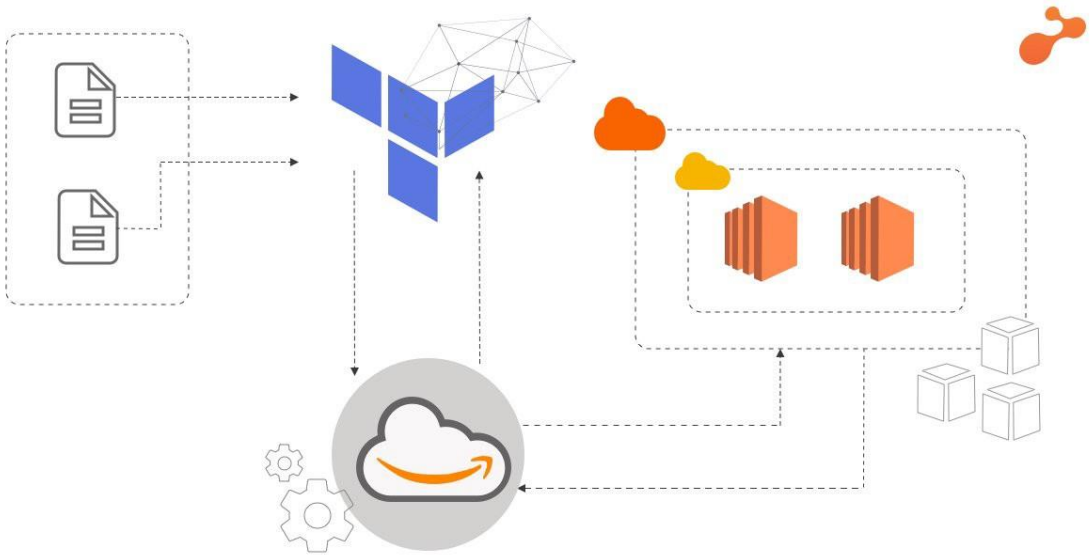


AWS - AZURE Providers

- AWS, Azure
    - AWS - https://registry.terraform.io/providers/hashicorp/aws/latest/docs
    - AZURE - https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs

AWS Provider

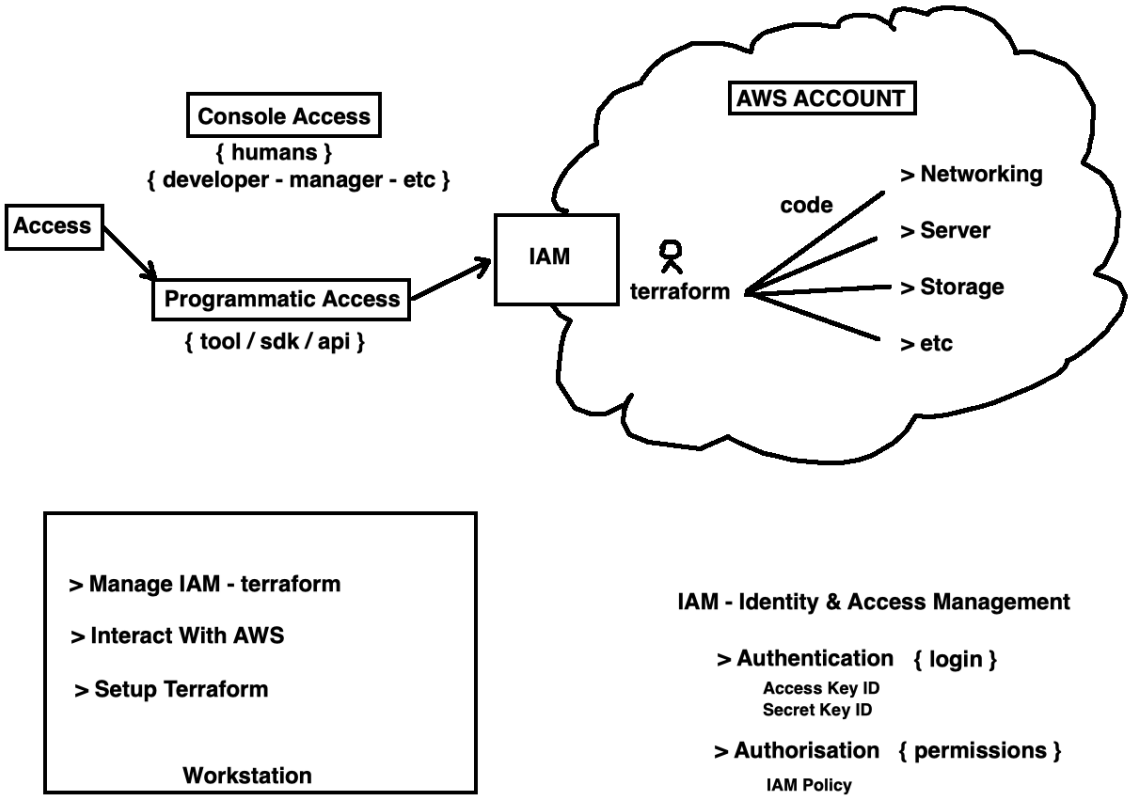
- Use the Amazon Web Services (AWS) provider to interact with the many resources supported by AWS.
- You must **configure the provider with the proper credentials** before you can use it.



AWS IAM

- With AWS **Identity and Access Management** (IAM), you can specify who or what (**terraform**) can access services and resources in AWS.

- You use IAM to control who is **authenticated** (**signed in**) and **authorised** (has **permissions**) to use resources.



Manage infrastructure with Terraform

Write configuration

- The set of files used to describe infrastructure in Terraform is known as a **Terraform configuration**.

- Each Terraform configuration must be in its own working directory. **Create a directory for your configuration**.

```
mkdir ~/tf-aws-project
cd ~/tf-aws-project
```

```
terraform init

# Create a file to define your infrastructure.
vi provider.tf

# Using AWS Provider -
https://registry.terraform.io/providers/hashicorp/aws/latest/docs

# AWS Provider
provider "aws" {
  region     = "eu-west-2"
  access_key = "JSHHSHSJSJSKSKSKKSKS"
  secret_key = "SJSHSJSJJSJSKSKSKKSKSKSKSKKS"
}

terraform init
terraform plan

# No changes. Your infrastructure matches the configuration.
```

Terraform Resources

- Resources are the **most important element** in the Terraform language.
- Each **resource block describes one or more infrastructure objects**, such as virtual networks, compute instances etc
- We use terraform resource blocks to define components of infrastructure which can be a VPC, Subnet, RouteTable, Security Group, EC2 instance etc

Resource Syntax

- https://developer.hashicorp.com/terraform/language/resources/syntax

- A **resource** block declares a resource of a **specific type** with a **specific local name**.

- In the following example, the **aws_instance** resource type is named **web**. The resource type and name must be unique within a module because they serve as an identifier for a given resource.

```
resource "aws_instance" "web" {
  ami           = "ami-a1b2c3d4"
  instance_type = "t2.micro"
}
```

- Within the block body (between **{ and }**) are the **configuration arguments** for the resource itself. The arguments often depend on the resource type. In this example, both ami and instance_type are special arguments for the aws_instance resource type.
- Check the Provider Resource Document Always
    - https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/vpc
    - https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/ebs_volume

Manage infrastructure with Terraform - AWS

- You can perform following operations on any resource in cloud, using **EBS volume** to demonstrate following operations in AWS Console
    - Create
    - Modify/Update
    - Delete

- With Terraform installed, you are ready to create your first infrastructure.

- In this demo, you will provision an EBS volume on Amazon Web Services (AWS). EBS volumes are a common component of many infrastructure projects.

Write configuration

- The set of files used to describe infrastructure in Terraform is known as a Terraform configuration.

- Each Terraform configuration must be in its own working directory. Create a directory for your configuration. You will write your first configuration to define a single EBS Volume in AWS.

Initialize the directory/project

- When you create a new configuration or check out an existing configuration from version control, you need to initialize the directory with **terraform init**.

- Initializing a configuration directory downloads and installs the providers defined in the configuration, which in this case is the aws provider.

- Terraform downloads the aws provider and installs it in a hidden subdirectory of your current working directory, named **.terraform**

- The terraform init command prints out which version of the provider was installed. Terraform also creates a lock file named **.terraform.lock.hcl** which specifies the exact provider versions used, so

that you can control when you want to update the providers used for your project.

- Reference - https://www.terraform.io/docs/cli/init/index.html

```
mkdir ~/tf-aws-project
cd ~/tf-aws-project
terraform init

# Create a file to define your infrastructure.
vi provider.tf

# Using AWS Provider -
https://registry.terraform.io/providers/hashicorp/aws/latest/docs

# AWS Provider
provider "aws" {
  region     = "eu-west-2"
  access_key = "AKIAQQABK7ATZ3SFU"
  secret_key = "tyRcKjHlvt7lw8DnsoMHnxA2foyFQoD5oHfVU"
}


> terraform init

> terraform plan

# No changes. Your infrastructure matches the configuration.
```

```
terraform plan

# No changes. Your infrastructure matches the configuration.
```

```
> cat ebs.tf

resource "aws_ebs_volume" "vol-1" {
      availability_zone = "eu-west-2a"
  size              = 5

  tags = {
          Name = "terraform-volume"
  }
}
```

- **Terraform Block** - The terraform {} block contains Terraform settings, including the required providers Terraform will use to provision your infrastructure. For each provider, the source attribute defines an optional hostname, a namespace, and the provider type. Terraform installs providers from the Terraform Registry by default. In this example configuration, the aws provider's source is defined as hashicorp/aws, which is shorthand for registry.terraform.io/hashicorp/aws.

- You can also set a version constraint for each provider defined in the required_providers block. The version attribute is optional,

- If you do not specify a provider version, Terraform will automatically download the most recent version during initialization.

- **Provider block** - configures the specified provider, in this case aws. A provider is a plugin that Terraform uses to create and manage your resources.

- **Resources block** - Use resource blocks to define components of your infrastructure.
- Resource blocks have two strings before the block: the resource type and the resource name.

- Together, the resource type and resource name form a unique ID for the resource. For example, the ID for your EBS volume is aws_ebs_volume.my-volume
- Resource blocks contain arguments which you use to configure the resource.

Format and validate the configuration

- It is recommended to use **consistent formatting** in all of your configuration files.

- The **terraform fmt** command automatically updates configurations in the current directory for readability and consistency.

- Format your configuration. Terraform will print out the names of the files it modified, if any. In this case, your configuration file was already formatted correctly, so Terraform won't return any file names.

```
> cat ebs.tf

resource "aws_ebs_volume" "vol-1" {
      availability_zone = "eu-west-2a"
  size              = 5

  tags = {
          Name = "terraform-volume"
  }
}

> terraform fmt
> cat ebs.tf
```

- You can also make sure your **configuration is syntactically valid** and internally consistent by using the **terraform validate** command.

```
> cat ebs.tf

resource "aws_ebs_volume" "vol-1" {
  availability_zone = "eu-west-2a"
  sie               = 5 # added issue with size

  tags = {
    Name = "terraform-volume"
  }
}

> terraform validate

# Correct the word size and re validate

> terraform validate
```

Plan infrastructure

- The **terraform plan** command evaluates a Terraform configuration to determine the desired state of all the resources it declares, then compares that desired state to the real infrastructure objects.
- Once it has determined the difference between the current state and the desired state, the **terraform plan** presents a description of the changes necessary to achieve the desired state.
- The **terraform plan** command ***does not* perform any actual changes** to real world infrastructure objects; it **only presents a plan** for making changes.
- The **terraform plan** command is **dry run of your script**, it states what will happen with your infra like what resources will be created, modified, deleted etc
- **terraform plan** might be run before committing a change to version control, to create confidence that it will behave as expected.

- It is always best practice to run **terraform plan** command

```
terraform plan
```

```
> cat ebs.tf

resource "aws_ebs_volume" "vol-1" {
  availability_zone = "eu-west-2a"
  size              = 5

  tags = {
    Name = "terraform-volume"
  }
}

> terraform plan
```

Create infrastructure

- The **terraform apply** command is used to apply the changes required to reach the desired state of the configuration

- Before it applies any changes, Terraform prints out the execution plan which describes the actions Terraform will take in order to change your infrastructure to match the configuration.

```
terraform apply
```

- Terraform will now pause and **wait for your approval** before proceeding. If anything in the plan seems incorrect or dangerous, it is

safe to abort here with no changes made to your infrastructure. In this case the plan is acceptable, so type **yes**

```
> cat ebs.tf

resource "aws_ebs_volume" "vol-1" {
  availability_zone = "eu-west-2a"
  size              = 5

  tags = {
    Name = "terraform-volume"
  }
}

> terraform apply



terraform apply
```

- You have now created infrastructure using Terraform! Visit the EC2 console and find your new EBS Volume.

```
# Modify the changes (decreasing size is not possible)

# Destroy Resource
terraform destroy
```

- You have now created / modified / destroyed infrastructure using Terraform!

20 Feb 2025

Manage infrastructure with Terraform - AZURE

- You can perform following operations on any resource in cloud, using **Azure Disk** to demonstrate following operations in Azure Portal
    - Create
    - Modify/Update
    - Delete

Azure CLI

- The **Azure command-line interface** (**Azure CLI**) is a **set of commands** used to **create and manage Azure resources.**
- with CLI **emphasis on automation**.

https://learn.microsoft.com/en-us/cli/azure/

Azure CLI Installation

- https://learn.microsoft.com/en-us/cli/azure/install-azure-cli

- https://learn.microsoft.com/en-us/cli/azure/install-azure-cli-windows?source=recommendations&tabs=azure-cli

Azure CLI Commands

- az login --use-device-code

```
az login --use-device-code
```

➢ The Azure CLI's default authentication method for logins uses a web browser and access token to sign in.

- Manage Azure Disk with AZ-CLI commands

```
az disk create -g 2502 -n TestDisk --size-gb 5
az disk update --name TestDisk --resource-group  2502 --size-gb 6
az disk delete --name TestDisk --resource-group  2502
```

Azure Terraform Disk Example

```
mkdir ~/tf-az-project
cd ~/tf-az-project

> terraform init
```

```
cat provider.tf

# Configure the Microsoft Azure Provider
terraform {
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "=3.0.0"
    }
  }
}

provider "azurerm" {
# resource_provider_registrations = "none"
  features {}
}

> terraform init

> terraform plan

> cat disk.tf

resource "azurerm_resource_group" "tf-rg" {
  name     = "tf-resources"
  location = "East US"
}

> terraform plan
> terraform apply




cat disk.tf
resource "azurerm_resource_group" "tf-rg" {
  name     = "tf-resources"
  location = "East US"
```

```
}

resource "azurerm_managed_disk" "disk-1" {
  name                 = "tf-disk"
  location             = azurerm_resource_group.tf-
rg.location
  resource_group_name  = azurerm_resource_group.tf-rg.name
  storage_account_type = "Standard_LRS"
  create_option        = "Empty"
  disk_size_gb         = "5"

  tags = {
    environment = "staging"
  }
}

> terraform apply
```

```
mkdir ~/tf-az-project
cd ~/tf-az-project

cat provider.tf
# Configure the Microsoft Azure Provider
terraform {
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "=3.0.0"
    }
  }
```

```
}

provider "azurerm" {
# resource_provider_registrations = "none"
  features {}
}

> terraform init

> cat disk.tf

resource "azurerm_resource_group" "tf-rg" {
  name     = "tf-resources"
  location = "East US"
}

> terraform plan
> terraform apply




cat disk.tf
resource "azurerm_resource_group" "tf-rg" {
  name     = "tf-resources"
  location = "East US"
}

resource "azurerm_managed_disk" "disk-1" {
  name                 = "tf-disk"
  location             = azurerm_resource_group.tf-
rg.location
  resource_group_name  = azurerm_resource_group.tf-rg.name
  storage_account_type = "Standard_LRS"
  create_option        = "Empty"
  disk_size_gb         = "5"

  tags = {
    environment = "staging"
```

```
  }
}

> terraform apply
```

21 Feb 2025

Terraform State File

- We create terraform files for managing resources like EC2,VPC, ELB etc
- Now somewhere this information should be recorded like in a database ?
- Terraform records information about what infrastructure it created in a Terraform state file.
- This state is stored by default in a local file named "**terraform.tfstate**"
- The primary purpose of Terraform state is to store **bindings** between objects in a remote system and resource instances declared in your configuration.
- So when we run **terraform plan** command, it will compare the current state from **terraform.tfstate** file and actual configuration .tf file

VARIABLES

- From terraform context, **variables** are also known as **input variables** or **terraform variables**

- Terraform configurations can include variables to make your configuration **more dynamic and flexible** and **easier** to **re-use** .

- You can set variables in a **separate file(variables.tf or vars.tf)** or in the **same existing configuration file.**

```
# This is String
variable "key" {
     type = string
     default = "this is my first string"
}

output "first_string_output" {
     value = var.key
}
```

```
terraform apply

terraform apply -var “key=ravi”
```

> cat variables.tf

```
# Variables
variable "key" {
  type = string
  default = "this is my first string"
}

# Variable Size
variable "size" {
```

```
  type = string
  description = "Specify the Volume Size"
  default = "5"
}

# Variable Availability Zone
variable "zone" {
  type = string
  description = "Specify the Availability Zone"
  default = "us-west-2a"
}

# Outputs
output "first_string_output" {
  value = var.key
}

output "size_output" {
  value = var.size
}

output "zone_output" {
  value = var.zone
}
```

```
terraform apply

terraform apply -var “key=ravi”
```

 # without variables

```
# EBS Volume Resource - without variables
resource "aws_ebs_volume" "my-volume" {
  availability_zone = "us-west-2b"
  size              = 5
```

```
  tags = {
    Name = "2426-volume"
  }

}
```

 # with variables

```
# EBS Volume Resource - with variables
resource "aws_ebs_volume" "my-volume" {
  availability_zone = var.zone
  size              = var.size

  tags = {
    Name = "2426-volume"
  }

}
```

```
terraform apply

terraform apply -var "zone=us-west-2c" -var "size=10"
```

VARIABLES for AWS Secrets

> cat vars.tf

```
# Variables
variable "key" {
  type = string
  default = "this is my first string"
}

# Variable Size
variable "size" {
  type = string
  description = "Specify the Volume Size"
  default = "5"
}

# Variable Availability Zone
variable "zone" {
  type = string
  description = "Specify the Availability Zone"
  default = "us-west-2a"
}

# Access Key
variable "aws_access_key" {}

# Secret Key
variable "aws_secret_key" {}

# Outputs
output "first_string_output" {
  value = var.key
}

output "size_output" {
  value = var.size
}

output "zone_output" {
  value = var.zone
}
```

> cat provider.tf → OLD

```
# - Comment
# AWS Provider
provider "aws" {
region = "us-west-2"
access_key = "AKIAZJQNC64QHRESDF"
secret_key = "RUQOOykx4qKPiy6Ddffffgzaquw76zRkSJDm39VFnmn"
}
```

> cat provider.tf → NEW

```
# - Comment
# AWS Provider
provider "aws" {
region = "us-west-2"
access_key = var.aws_access_key
secret_key = var.aws_secret_key
}
```

```
terraform apply # Prompts for Access & Secret Key

terraform apply -var "aws_access_key=AKIAZJQNC64QHREQOIMR" -
var "aws_secret_key=RUQOOykx4qKPiy6lDFgzaquW76zRkSJDm39VFnmn"

# On Your Project do git init
```

24 - 25 Feb 2025

CODE - LMS VPC SETUP - 45 Mins

```
# VPC
resource "aws_vpc" "lms" {
  cidr_block       = "10.0.0.0/16"
  instance_tenancy = "default"

  tags = {
    Name = "lms"
  }
}

# Web Subnet
resource "aws_subnet" "lms-web-sn" {
  vpc_id     = aws_vpc.lms.id
  cidr_block = "10.0.1.0/24"
  map_public_ip_on_launch = "true"
```

```
  tags = {
    Name = "lms-web-subnet"
  }
}

# API Subnet
resource "aws_subnet" "lms-api-sn" {
  vpc_id     = aws_vpc.lms.id
  cidr_block = "10.0.2.0/24"
  map_public_ip_on_launch = "true"

  tags = {
    Name = "lms-api-subnet"
  }
}

# Database Subnet
resource "aws_subnet" "lms-db-sn" {
  vpc_id     = aws_vpc.lms.id
  cidr_block = "10.0.3.0/24"
  map_public_ip_on_launch = "false"

  tags = {
    Name = "lms-db-subnet"
  }
}

# Internet Gateway
resource "aws_internet_gateway" "lms-igw" {
  vpc_id = aws_vpc.lms.id

  tags = {
    Name = "lms-internet-gateway"
  }
}

# Route Table
resource "aws_route_table" "lms-pub-rt" {
  vpc_id = aws_vpc.lms.id
```

```
  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.lms-igw.id
  }

  tags = {
    Name = "lms-public-rt"
  }
}

# Web Subnet Association
resource "aws_route_table_association" "lms-web-asc" {
  subnet_id      = aws_subnet.lms-web-sn.id
  route_table_id = aws_route_table.lms-pub-rt.id
}

# API Subnet Association
resource "aws_route_table_association" "lms-api-asc" {
  subnet_id      = aws_subnet.lms-api-sn.id
  route_table_id = aws_route_table.lms-pub-rt.id
}

# Private Route Table
resource "aws_route_table" "lms-pvt-rt" {
  vpc_id = aws_vpc.lms.id

  tags = {
    Name = "lms-private-rt"
  }
}

# DB Subnet Association
resource "aws_route_table_association" "lms-db-asc" {
  subnet_id      = aws_subnet.lms-db-sn.id
  route_table_id = aws_route_table.lms-pvt-rt.id
}

# NACL
```

```
resource "aws_network_acl" "lms-nacl" {
  vpc_id = aws_vpc.lms.id

  egress {
    protocol   = "tcp"
    rule_no    = 100
    action     = "allow"
    cidr_block = "0.0.0.0/0"
    from_port  = 0
    to_port    = 65535
  }

  ingress {
    protocol   = "tcp"
    rule_no    = 100
    action     = "allow"
    cidr_block = "0.0.0.0/0"
    from_port  = 0
    to_port    = 65535
  }

  tags = {
    Name = "lms-nacl"
  }
}

# NACL Associations - Web
resource "aws_network_acl_association" "lms-nacl-asc" {
  network_acl_id = aws_network_acl.lms-nacl.id
  subnet_id      = aws_subnet.lms-web-sn.id
}

# NACL Associations - API
resource "aws_network_acl_association" "lms-nacl-asc-api" {
  network_acl_id = aws_network_acl.lms-nacl.id
  subnet_id      = aws_subnet.lms-api-sn.id
}

# NACL Associations - DB
```

```
resource "aws_network_acl_association" "lms-nacl-asc-db" {
  network_acl_id = aws_network_acl.lms-nacl.id
  subnet_id      = aws_subnet.lms-db-sn.id
}

# Web Security Group
resource "aws_security_group" "lms-web-sg" {
  name        = "lms-web-sg"
  description = "Allow Web Traffic"
  vpc_id      = aws_vpc.lms.id

  tags = {
    Name = "lms-web-sg"
  }
}

# Web Security Group Ingress Rule - ssh
resource "aws_vpc_security_group_ingress_rule" "lms-web-sg-
ssh" {
  security_group_id = aws_security_group.lms-web-sg.id
  cidr_ipv4         = "0.0.0.0/0"
  from_port         = 22
  ip_protocol       = "tcp"
  to_port           = 22
}

# Web Security Group Ingress Rule - http
resource "aws_vpc_security_group_ingress_rule" "lms-web-sg-
http" {
  security_group_id = aws_security_group.lms-web-sg.id
  cidr_ipv4         = "0.0.0.0/0"
  from_port         = 80
  ip_protocol       = "tcp"
  to_port           = 80
}

# Web Security Group Egress Rule - All
resource "aws_vpc_security_group_egress_rule" "lms-web-sg-
all" {
```

```
  security_group_id = aws_security_group.lms-web-sg.id
  cidr_ipv4         = "0.0.0.0/0"
  ip_protocol       = "-1" # semantically equivalent to all
ports
}

# API Security Group
resource "aws_security_group" "lms-api-sg" {
  name        = "lms-api-sg"
  description = "Allow API Traffic"
  vpc_id      = aws_vpc.lms.id

  tags = {
    Name = "lms-api-sg"
  }
}

# API Security Group Ingress Rule - ssh
resource "aws_vpc_security_group_ingress_rule" "lms-api-sg-
ssh" {
  security_group_id = aws_security_group.lms-api-sg.id
  cidr_ipv4         = "0.0.0.0/0"
  from_port         = 22
  ip_protocol       = "tcp"
  to_port           = 22
}

# API Security Group Ingress Rule - http
resource "aws_vpc_security_group_ingress_rule" "lms-api-sg-
http" {
  security_group_id = aws_security_group.lms-api-sg.id
  cidr_ipv4         = "0.0.0.0/0"
  from_port         = 8080
  ip_protocol       = "tcp"
  to_port           = 8080
}

# API Security Group Egress Rule - All
resource "aws_vpc_security_group_egress_rule" "lms-api-sg-
```

```
all" {
  security_group_id = aws_security_group.lms-api-sg.id
  cidr_ipv4         = "0.0.0.0/0"
  ip_protocol       = "-1" # semantically equivalent to all
ports
}

# DB Security Group
resource "aws_security_group" "lms-db-sg" {
  name        = "lms-db-sg"
  description = "Allow DB Traffic"
  vpc_id      = aws_vpc.lms.id

  tags = {
    Name = "lms-db-sg"
  }
}

# DB Security Group Ingress Rule - ssh
resource "aws_vpc_security_group_ingress_rule" "lms-db-sg-
ssh" {
  security_group_id = aws_security_group.lms-db-sg.id
  cidr_ipv4         = "0.0.0.0/0"
  from_port         = 22
  ip_protocol       = "tcp"
  to_port           = 22
}

# DB Security Group Ingress Rule - postgres
resource "aws_vpc_security_group_ingress_rule" "lms-db-sg-
postgres" {
  security_group_id = aws_security_group.lms-db-sg.id
  cidr_ipv4         = "0.0.0.0/0"
  from_port         = 5432
  ip_protocol       = "tcp"
  to_port           = 5432
}

# DB Security Group Egress Rule - All
```

```
resource "aws_vpc_security_group_egress_rule" "lms-db-sg-all"
{
  security_group_id = aws_security_group.lms-db-sg.id
  cidr_ipv4         = "0.0.0.0/0"
  ip_protocol       = "-1" # semantically equivalent to all
ports
}

# EC2 Web Server
resource "aws_instance" "lms-web-server" {
  ami           = "ami-06cff85354b67982b"
  instance_type = "t2.micro"
  key_name      = "2502"
  subnet_id     = aws_subnet.lms-web-sn.id
  vpc_security_group_ids = [aws_security_group.lms-web-sg.id]

  tags = {
    Name = "lms-web-server"
  }
}
```

CODE - LOGIN VPC SETUP - 45 Mins

- Infrastructure Setup w.r.t LMS
- Frontend
- API
- Database

CODE - ECOMM VPC SETUP - 1 Mins

- Infrastructure Setup w.r.t LMS
- Frontend

- API
- Database

CODE - FOOD VPC SETUP - 1 Mins

- Infrastructure Setup w.r.t LMS
- Frontend
- API
- Database

26 - 27 Feb 2025

CODE - LMS VNET SETUP - 45 Mins

```
# Resource Group
resource "azurerm_resource_group" "lms-rg" {
  name     = "lms-rg"
  location = "East US"
}

# VNET
resource "azurerm_virtual_network" "lms-vnet" {
  name                = "lms"
  location            = azurerm_resource_group.lms-
rg.location
  resource_group_name = azurerm_resource_group.lms-rg.name
  address_space       = ["10.0.0.0/16"]
    tags = {
```

```
    environment = "dev"
  }
}

# WEB Subnet
resource "azurerm_subnet" "lms-web-sn" {
  name                 = "lms-web-subnet"
  resource_group_name  = azurerm_resource_group.lms-rg.name
  virtual_network_name = azurerm_virtual_network.lms-
vnet.name
  address_prefixes     = ["10.0.1.0/24"]
}

# API Subnet
resource "azurerm_subnet" "lms-api-sn" {
  name                 = "lms-api-subnet"
  resource_group_name  = azurerm_resource_group.lms-rg.name
  virtual_network_name = azurerm_virtual_network.lms-
vnet.name
  address_prefixes     = ["10.0.2.0/24"]
}

# DB Subnet
resource "azurerm_subnet" "lms-db-sn" {
  name                 = "lms-db-subnet"
  resource_group_name  = azurerm_resource_group.lms-rg.name
  virtual_network_name = azurerm_virtual_network.lms-
vnet.name
  address_prefixes     = ["10.0.3.0/24"]
}

# Web Public IP
resource "azurerm_public_ip" "lms-web-pip" {
  name                = "lms-web-pip"
  resource_group_name = azurerm_resource_group.lms-rg.name
  location            = azurerm_resource_group.lms-
rg.location
  allocation_method   = "Static"
```

```
  tags = {
    environment = "dev"
  }
}

# API Public IP
resource "azurerm_public_ip" "lms-api-pip" {
  name                = "lms-api-pip"
  resource_group_name = azurerm_resource_group.lms-rg.name
  location            = azurerm_resource_group.lms-
rg.location
  allocation_method   = "Static"

  tags = {
    environment = "dev"
  }
}

# Web Network Security Group - NSG
resource "azurerm_network_security_group" "lms-web-nsg" {
  name                = "lms-web-nsg"
  location            = azurerm_resource_group.lms-
rg.location
  resource_group_name = azurerm_resource_group.lms-rg.name
}

# Web Network Security Group - NSG - Rules
resource "azurerm_network_security_rule" "lms-web-nsg-ssh" {
  name                        = "ssh"
  priority                    = 100
  direction                   = "Inbound"
  access                      = "Allow"
  protocol                    = "Tcp"
  source_port_range           = "*"
  destination_port_range      = "22"
  source_address_prefix       = "*"
  destination_address_prefix  = "*"
  resource_group_name         = azurerm_resource_group.lms-
rg.name
```

```
  network_security_group_name =
azurerm_network_security_group.lms-web-nsg.name
}

# Web Network Security Group - NSG - Rules
resource "azurerm_network_security_rule" "lms-web-nsg-http" {
  name                        = "http"
  priority                    = 101
  direction                   = "Inbound"
  access                      = "Allow"
  protocol                    = "Tcp"
  source_port_range           = "*"
  destination_port_range      = "80"
  source_address_prefix       = "*"
  destination_address_prefix  = "*"
  resource_group_name         = azurerm_resource_group.lms-
rg.name
  network_security_group_name =
azurerm_network_security_group.lms-web-nsg.name
}

# API Network Security Group - NSG
resource "azurerm_network_security_group" "lms-api-nsg" {
  name                = "lms-api-nsg"
  location            = azurerm_resource_group.lms-
rg.location
  resource_group_name = azurerm_resource_group.lms-rg.name
}

# API Network Security Group - NSG - Rules
resource "azurerm_network_security_rule" "lms-api-nsg-ssh" {
  name                        = "ssh"
  priority                    = 100
  direction                   = "Inbound"
  access                      = "Allow"
  protocol                    = "Tcp"
  source_port_range           = "*"
  destination_port_range      = "22"
  source_address_prefix       = "*"
```

```
  destination_address_prefix  = "*"
  resource_group_name         = azurerm_resource_group.lms-
rg.name
  network_security_group_name =
azurerm_network_security_group.lms-api-nsg.name
}

# API Network Security Group - NSG - Rules
resource "azurerm_network_security_rule" "lms-api-nsg-http" {
  name                        = "http"
  priority                    = 101
  direction                   = "Inbound"
  access                      = "Allow"
  protocol                    = "Tcp"
  source_port_range           = "*"
  destination_port_range      = "8080"
  source_address_prefix       = "*"
  destination_address_prefix  = "*"
  resource_group_name         = azurerm_resource_group.lms-
rg.name
  network_security_group_name =
azurerm_network_security_group.lms-api-nsg.name
}

# DB Network Security Group - NSG
resource "azurerm_network_security_group" "lms-db-nsg" {
  name                = "lms-db-nsg"
  location            = azurerm_resource_group.lms-
rg.location
  resource_group_name = azurerm_resource_group.lms-rg.name
}

# DB Network Security Group - NSG - Rules
resource "azurerm_network_security_rule" "lms-db-nsg-ssh" {
  name                        = "ssh"
  priority                    = 100
  direction                   = "Inbound"
  access                      = "Allow"
  protocol                    = "Tcp"
```

```
  source_port_range           = "*"
  destination_port_range      = "22"
  source_address_prefix       = "*"
  destination_address_prefix  = "*"
  resource_group_name         = azurerm_resource_group.lms-
rg.name
  network_security_group_name =
azurerm_network_security_group.lms-db-nsg.name
}

# DB Network Security Group - NSG - Rules
resource "azurerm_network_security_rule" "lms-db-nsg-
postgres" {
  name                        = "postgres"
  priority                    = 101
  direction                   = "Inbound"
  access                      = "Allow"
  protocol                    = "Tcp"
  source_port_range           = "*"
  destination_port_range      = "5432"
  source_address_prefix       = "*"
  destination_address_prefix  = "*"
  resource_group_name         = azurerm_resource_group.lms-
rg.name
  network_security_group_name =
azurerm_network_security_group.lms-db-nsg.name
}

# WEB NIC
resource "azurerm_network_interface" "lms-web-nic" {
  name                = "lms-web-nic"
  location            = azurerm_resource_group.lms-
rg.location
  resource_group_name = azurerm_resource_group.lms-rg.name

  ip_configuration {
    name                          = "internal"
    subnet_id                     = azurerm_subnet.lms-web-
sn.id
```

```
    private_ip_address_allocation = "Dynamic"
    public_ip_address_id = azurerm_public_ip.lms-web-pip.id
  }
}

# WEB NIC Association
resource
"azurerm_network_interface_security_group_association" "lms-
web-nic-asc" {
  network_interface_id      = azurerm_network_interface.lms-
web-nic.id
  network_security_group_id =
azurerm_network_security_group.lms-web-nsg.id
}

# API NIC
resource "azurerm_network_interface" "lms-api-nic" {
  name                = "lms-api-nic"
  location            = azurerm_resource_group.lms-
rg.location
  resource_group_name = azurerm_resource_group.lms-rg.name

  ip_configuration {
    name                          = "internal"
    subnet_id                     = azurerm_subnet.lms-api-
sn.id
    private_ip_address_allocation = "Dynamic"
    public_ip_address_id = azurerm_public_ip.lms-api-pip.id
  }
}

# API NIC Association
resource
"azurerm_network_interface_security_group_association" "lms-
api-nic-asc" {
  network_interface_id      = azurerm_network_interface.lms-
api-nic.id
  network_security_group_id =
azurerm_network_security_group.lms-api-nsg.id
```

```
}

# DB NIC
resource "azurerm_network_interface" "lms-db-nic" {
  name                = "lms-db-nic"
  location            = azurerm_resource_group.lms-
rg.location
  resource_group_name = azurerm_resource_group.lms-rg.name

  ip_configuration {
    name                          = "internal"
    subnet_id                     = azurerm_subnet.lms-db-
sn.id
    private_ip_address_allocation = "Dynamic"
  }
}

# DB NIC Association
resource
"azurerm_network_interface_security_group_association" "lms-
db-nic-asc" {
  network_interface_id      = azurerm_network_interface.lms-
db-nic.id
  network_security_group_id =
azurerm_network_security_group.lms-db-nsg.id
}

# WEB VM
resource "azurerm_linux_virtual_machine" "lms-web-vm" {
  name                = "lms-web-vm"
  resource_group_name = azurerm_resource_group.lms-rg.name
  location            = azurerm_resource_group.lms-
rg.location
  size                = "Standard_F2"
  admin_username      = "ubuntu"
  network_interface_ids = [
    azurerm_network_interface.lms-web-nic.id,
  ]
```

```
  admin_ssh_key {
    username   = "ubuntu"
    public_key = file("~/.ssh/id_rsa.pub")
  }

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

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

Terraform conditional expressions

- A conditional expression uses the value of a **boolean expression** to select one of two values.

- Conditional expressions in Terraform are used to decide which value to assign to a variable or an attribute, based on a condition.

- The **syntax** of a conditional expression is as follows:

```
condition ? true_val : false_val
```

- If the **condition is true** then the result is **true_val**.
- If the **condition is false** then the result is **false_val**.

```
> cat variables.tf

variable "aws_access_key" {
  type        = string
  description = "Enter AWS Access Key"
}

variable aws_secret_key {
  type        = string
  description = "Enter AWS Secret Access Key"
}

variable env {
  type        = string
  description = "Enter Environment"
}
```

```
> cat vpc.tf

# VPC
resource "aws_vpc" "ecomm" {
  cidr_block       = var.env == "prod" ? "10.0.0.0/16" :
"192.168.0.0/24"
  instance_tenancy = "default"

  tags = {
    Name = "${var.env}-ecomm"
  }
}
```

Here's how you can use conditional expressions to handle multiple environments like dev, qa, uat, and prod and **dynamically** generate different VPCs for each environment.

Like a switch case statement

Using var.env with Multiple Conditional Expressions

You can update your **env** variable to accept multiple values and use conditional expressions to handle each environment.

> cat vars.tf

```
# Variable for Access Key
variable "aws_access_key" {}

# Variable for Secret Key
variable "aws_secret_key" {}

# Declare a variable for the environment
variable "env" {
  default = "prod"
  description = "Environment type (dev, qa, uat, prod)"
}
```

> cat vpc-resources.tf

```
# Use conditional expressions to define different CIDR blocks and
tags for each environment
# VPC
resource "aws_vpc" "login-vpc" {
  cidr_block = (
    var.env == "prod" ? "10.0.0.0/16" :
```

```
    var.env == "uat"   ? "192.168.0.0/20" :
    var.env == "qa"    ? "172.31.16.0/20" :
                         "10.1.0.0/20"    # Default to "dev"
  )

  tags = {
    Name = "${var.env}-login-vpc"
  }
}
```

```
terraform apply
# Verify CIDR Block & Tag

terraform apply -var "env=uat"
terraform apply -var "env=qa"
terraform apply -var "env=xyz"
```

The parentheses () are used to wrap the entire conditional expression. This is required for Terraform to correctly parse and understand the conditional logic.

3 Mar 2025

Terraform Modules

- https://developer.hashicorp.com/terraform/language/modules
- Modules are containers for multiple resources that are used together. A module consists of a **collection of .tf files** kept together in a directory.
- Modules are the main way to **package and reuse resource configurations** with Terraform.
- This is almost similar to using **helm charts** in kubernetes.

Modules Lab

- Using Terraform Modules to Create a VPC and EC2 Instances for **Multiple Applications**

- This activity demonstrates how to use Terraform modules to set up a VPC and launch EC2 instances for three different web applications (**login_app**, **ecomm_app**, **food_app**) using a modular approach. By creating **reusable** modules, we can avoid repeating code and easily replicate infrastructure setups.

- **Create the VPC Module**:
    - A module is created to define a **VPC** and a **public subnet**.
    - Parameters such as VPC CIDR, subnet CIDR, and availability zone are passed as variables.
    - The module outputs the VPC ID and subnet ID for further use.
- **Create the EC2 Module**:
    - Another module is created to define the **EC2 instance**.
    - The instance is provisioned using inputs like **AMI**, **instance type**, **subnet ID**, and **key name**.
    - The EC2 module also outputs the instance ID.
- **Use the Modules in Application Configurations**:
    - For each application (`login_app`, `ecomm_app`, `food_app`), a separate Terraform configuration file (`login_app.tf`, `ecomm_app.tf`, `food_app.tf`) is created.
    - Each file uses the same VPC and EC2 modules but with different subnet CIDR blocks and availability zones to ensure separation between the applications.
- **Deployment Process**:
    - Terraform initialises and applies the configurations for each application individually, allowing for easy infrastructure setup.

```
mkdir -p ~/proj_modules

# Create VPC Module
mkdir -p ~/proj_modules/modules/vpc

> cat ~/proj_modules/modules/vpc/variables.tf

variable "vpc_cidr" {
  description = "CIDR block for the VPC"
  type        = string
}

variable "vpc_name" {
  description = "CIDR block for the VPC"
  type        = string
}

variable "public_subnet_cidr" {
  description = "CIDR block for the public subnet"
  type        = string
}

variable "availability_zone" {
  description = "Availability Zone for the public subnet"
  type        = string
}

> cat ~/proj_modules/modules/vpc/main.tf

resource "aws_vpc" "vpc" {
  cidr_block = var.vpc_cidr
```

```
  tags = {
    Name = "${var.vpc_name}"
  }
}

resource "aws_subnet" "public" {
  vpc_id                  = aws_vpc.vpc.id
  cidr_block              = var.public_subnet_cidr
  availability_zone       = var.availability_zone
  map_public_ip_on_launch = true

  tags = {
    Name = "${var.vpc_name}-public-subnet"
  }
}

resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.vpc.id

  tags = {
    Name = "${var.vpc_name}-internet-gateway"
  }
}

resource "aws_route_table" "pub_rt" {
  vpc_id = aws_vpc.vpc.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.igw.id
  }

  tags = {
    Name = "${var.vpc_name}-public-route"
  }
}

resource "aws_route_table_association" "rt_asc" {
  subnet_id      = aws_subnet.public.id
```

```
  route_table_id = aws_route_table.pub_rt.id
}

resource "aws_security_group" "pub_sg" {
  name        = "${var.vpc_name}-SG"
  description = "Allow Frontend Traffic"
  vpc_id      = aws_vpc.vpc.id

  tags = {
    Name = "${var.vpc_name}-SG"
  }
}

resource "aws_vpc_security_group_ingress_rule" "pub_sg_ssh" {
  security_group_id = aws_security_group.pub_sg.id
  cidr_ipv4         = "0.0.0.0/0"
  from_port         = 22
  ip_protocol       = "tcp"
  to_port           = 22
}

resource "aws_vpc_security_group_ingress_rule" "pub_sg_http"
{
  security_group_id = aws_security_group.pub_sg.id
  cidr_ipv4         = "0.0.0.0/0"
  from_port         = 80
  ip_protocol       = "tcp"
  to_port           = 80
}

resource "aws_vpc_security_group_egress_rule" "pub_sg_out" {
  security_group_id = aws_security_group.pub_sg.id
  cidr_ipv4         = "0.0.0.0/0"
  from_port         = 0
  ip_protocol       = "tcp"
  to_port           = 65535
}

> cat ~/proj_modules/modules/vpc/output.tf
```

```
output "vpc_id" {
  value = aws_vpc.vpc.id
}

output "public_subnet_id" {
  value = aws_subnet.public.id
}

output "public_sg_id" {
  value = aws_security_group.pub_sg.id
}


# Create EC2 Module
mkdir -p ~/proj_modules/modules/ec2


> cat ~/proj_modules/modules/ec2/variables.tf

variable "ami" {
  description = "AMI to use for the instance"
  type        = string
}

variable "instance_type" {
  description = "Type of EC2 instance"
  type        = string
  default     = "t2.micro"
}

variable "subnet_id" {
  description = "Subnet ID to launch the instance in"
  type        = string
}

variable "key_name" {
  description = "EC2 Key Pair"
  type        = string
```

```
}

variable "vpc_security_group_ids" {
  description = "Security Group For Instance"
  type        = list(string)
}

variable "user_data" {
  description = "User Data For Instance"
  type        = string
}

variable "instance_name" {
  description = "Name of EC2 instance"
  type        = string
}



> cat ~/proj_modules/modules/ec2/main.tf

resource "aws_instance" "ec2" {
  ami                    = var.ami
  instance_type          = var.instance_type
  subnet_id              = var.subnet_id
  key_name               = var.key_name
  vpc_security_group_ids = var.vpc_security_group_ids
  user_data              = var.user_data

  tags = {
    Name = "${var.instance_name}"
  }
}

> cat ~/proj_modules/modules/ec2/output.tf

output "instance_id" {
  value = aws_instance.ec2.id
}
```

```
output "instance_ip" {
  value = aws_instance.ec2.public_ip
}




> cat ~/proj_modules/provider.tf

# Variable Access Key
variable "aws_access_key" {}

# Variable Secret Key
variable "aws_secret_key" {}

# AWS Provider
provider "aws" {
  region     = "us-west-2"
  access_key = var.aws_access_key
  secret_key = var.aws_secret_key
}

> cat ~/proj_modules/terraform.tfvars

# AWS Provider
aws_access_key = "AKIAZJQNC64QHREQOIMR"
aws_secret_key = "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"

> cat ~/proj_modules/login_app.sh

#!/bin/bash
apt update -y
apt install nginx -y
rm -r /var/www/html/*
git clone https://github.com/ravi2krishna/login-2430.git /var/www/html/

# Using the Modules in Your Application Configurations
```

```
> cat ~/proj_modules/login_app.tf

module "login_vpc" {
  source              = "./modules/vpc"
  vpc_cidr            = "10.0.0.0/16"
  public_subnet_cidr  = "10.0.1.0/24"
  availability_zone   = "us-west-2a"
  vpc_name            = "login"
}

module "login_ec2" {
  source                 = "./modules/ec2"
  ami                    = "ami-0075013580f6322a1"
  instance_type          = "t2.micro"
  subnet_id              = module.login_vpc.public_subnet_id
  vpc_security_group_ids = [module.login_vpc.public_sg_id]
  key_name               = "2429"
  user_data              = file("login_app.sh")
  instance_name          = "login"
}

output "instance_ip" {
  value = module.login_ec2.instance_ip
}

> cat ~/proj_modules/food_app.sh

#!/bin/bash
apt update -y
apt install nginx -y
rm -r /var/www/html/*
git clone https://github.com/ravi2krishna/food.git /var/www/html/


> cat ~/proj_modules/food_app.tf

module "food_vpc" {
  source              = "./modules/vpc"
  vpc_cidr            = "192.168.0.0/16"
```

```
  public_subnet_cidr = "192.168.1.0/24"
  availability_zone  = "us-west-2b"
  vpc_name           = "food"
}

module "food_ec2" {
  source                 = "./modules/ec2"
  ami                    = "ami-0075013580f6322a1"
  instance_type          = "t2.micro"
  subnet_id              = module.food_vpc.public_subnet_id
  vpc_security_group_ids = [module.food_vpc.public_sg_id]
  key_name               = "2429"
  user_data              = file("food_app.sh")
  instance_name          = "food"
}
```

```
cd ~/proj_modules
terraform init
terraform plan
terraform plan -target=module.food_vpc
terraform plan -target=module.food_vpc -
target=module.food_ec2
terraform apply
terraform destroy
```

IMPORTANT **Terminate Terraform Server**