* **IAAC**

IAAC stands for infrastructure as a code, which involves in managing and provisioning infrastructure resources through machine readable code. Terraform is a tool of infrastructure as a code

* **Benefits of IAAC**

**Automation and repeatability** – enable the infrastructure provisioning allowing you to create, mange and modify resources, this will eliminate manual and error prone process leading faster and more predictable deployments

**Infrastructure versioning and collaboration** – the code written in terraform can be version control using tools like git. Facilitates among team members, enable code reviews, and provide history of changes for better tracing and troubleshooting

**Scalability** – easy scalability of infrastructure resources like EC2, containers,VPC,to meet changing demand by modifying the terraform configuration

**Multi-Cloud and Hybrid Cloud Support** – AWS,AZURE,GCP

**Cost optimization** – terraform enables you to define infrastructure resources and their configurations, including sizing, networking and other cost related parameters helps optimize resource tools and costs

* **Advantages and disadvantages in Terraform**

**Advantages**

**Infrastructure as Code** Terraform allows you to define your infrastructure configuration as code. This enables version control, collaboration, and repeatability. Infrastructure changes are managed through code, promoting consistency and reducing manual errors.

**Multi-Cloud and Hybrid Cloud Support** Terraform supports multiple cloud providers, allowing you to manage resources across different platforms in a unified manner. It provides a consistent workflow and abstraction layer for provisioning infrastructure, making it easier to adopt a multi-cloud or hybrid cloud strategy.

**Automation and Scalability** Terraform automates the provisioning and management of infrastructure resources. It can scale resources up or down based on demand, allowing you to easily accommodate changes in workload or application requirements.

**Disadvantages**

**Understanding the Terraform language**, resource providers, and best practices requires time and effort.

**Complexity for Large infrastructure** grows larger and more complex, managing and organizing Terraform configurations can become challenging. It may require additional effort to design modular and reusable code, maintain proper state management

**Limited Provider Support and Features** Terraform supports many popular cloud providers, some providers may have limited or delayed support for specific features compared to their native management consoles or APIs

* **What are the provider is Terraform**

**Amazon Web Services** extensive support for provisioning and managing AWS resources

**Microsoft Azure** comprehensive support for provisioning and managing resources

**Google cloud platform** major cloud provider provisioning and managing resources

**DigitalOcean** provision and manage Droplets (virtual machines), load balancers, block storage, and other resources

**VMware vSphere** Terraform can provision and manage virtual machines, networks, datastores, and other resources in VMware vSphere, a widely used virtualization platform.

**Kubernetes** not a cloud provider excellent support for provisioning and managing Kubernetes resources. It can create and manage Kubernetes clusters, namespaces, deployments, services

* **Top commands in Terraform used in daily basis**

\*terraform init : Initializes the working directory, downloads required provider plugins, and sets up the backend configuration.

\*terraform plan: Generates an execution plan to show the changes that Terraform will apply to reach the desired infrastructure state.

\*terraform apply: Applies the changes defined in the Terraform configuration, creating, modifying in infrastructure resources.

\*terraform destroy: Destroys all resources created by the Terraform configuration, effectively removing the infrastructure.

\*terraform validate: Validates the syntax and configuration of the Terraform files, checking for errors or inconsistencies.

\*terraform output: Displays the output values defined in the Terraform configuration, such as IP addresses, DNS names, or other resource attributes.

\*terraform state: Provides commands to manage the Terraform state file, including inspecting, modifying, and cleaning up the state.

\*terraform show: will show the entire information of state file.

\*terraform state list : online show for resource list view

\*terraform state rm <resource id> : to remove the particular resource from statefile.

\*terraform import: Imports existing infrastructure resources into the Terraform state, allowing their management with Terraform.

\*terraform workspace: Manages multiple workspaces (environment contexts) within a Terraform configuration, enabling separate deployments for different environments.

\*terraform refresh: Refreshes the Terraform state to reconcile the real-world infrastructure with the state file.

\*terraform taint: Manually taints a resource, forcing it to be destroyed and recreated on the next apply.

\*terraform untaint: Removes the tainted state from a resource, allowing it to be retained during the next apply.

\*terraform graph: Generates a visual representation of the Terraform resource graph, illustrating resource dependencies.

\*terraform fmt: Formats the Terraform configuration files to adhere to a consistent style and indentation.

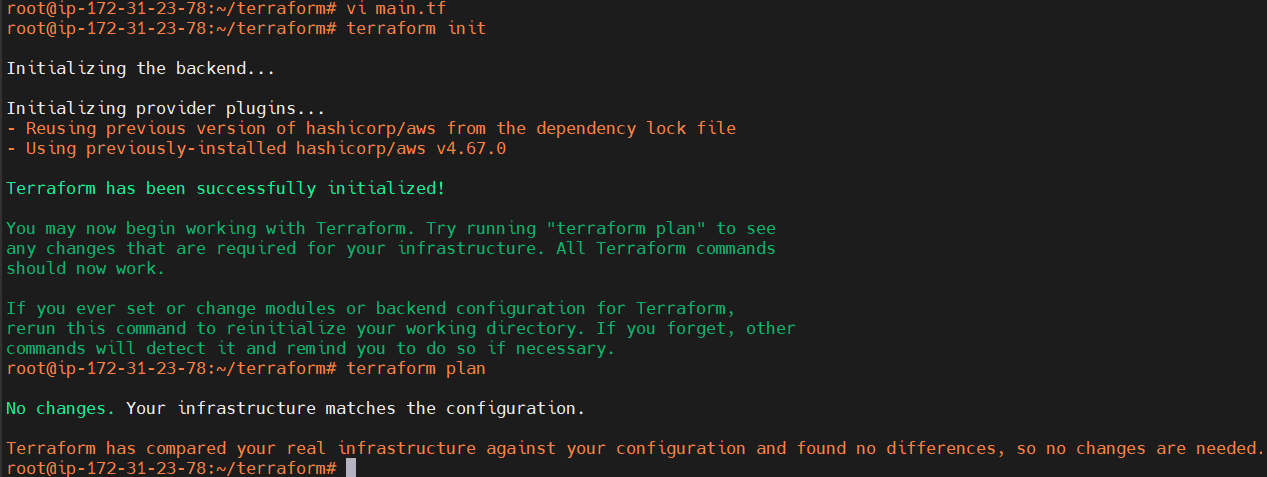
* **Configure single and Multiple providers: if single provider write single provider script and paste the same in main.tf**  **\*if multiple provider continue to write script as below follows script as highlighted\***

wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg

echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb\_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list

sudo apt update && sudo apt install terraform -y

* Mkdir terraform
* Create file vi main.tf
* Paste the below yaml script configuration in vi main.tf

#Terraform Block

terraform {

required\_providers {

aws = {

source = "hashicorp/aws"

version = "4.67.0"

}

}

}

provider "aws" {

region = "ap-south-2"

#profile = "default"

}

**provider "aws" {**

**region = "ap-south-1"**

**#profile = "default"**

**alias = "mumbai-bombay"**

* \*terraform init\* to initiate working directory,plugins and configurationd
* \*terraform plan\* to initiate the necessary resources
* **create a EC2 via terraform open security machine port 22 source permission my ip**

create a provider,and write yaml script as below provided name <file name>.tf

# Terraform Block

terraform {

required\_providers {

aws = {

source = "hashicorp/aws"

version = "4.67.0"

}

}

}

provider "aws" {

region = "ap-south-2"

# profile = "default"

}

# Pub SGP

resource "aws\_security\_group" "pubsgp" {

name = "allow\_tls"

description = "Allow TLS inbound traffic"

ingress {

description = "ssh"

from\_port = 22

to\_port = 22

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

}

# Website

resource "aws\_instance" "web" {

ami = "ami-0b584e5449fa1dd56"

instance\_type = "t3.micro"

associate\_public\_ip\_address = true

vpc\_security\_group\_ids = [aws\_security\_group.pubsgp.id]

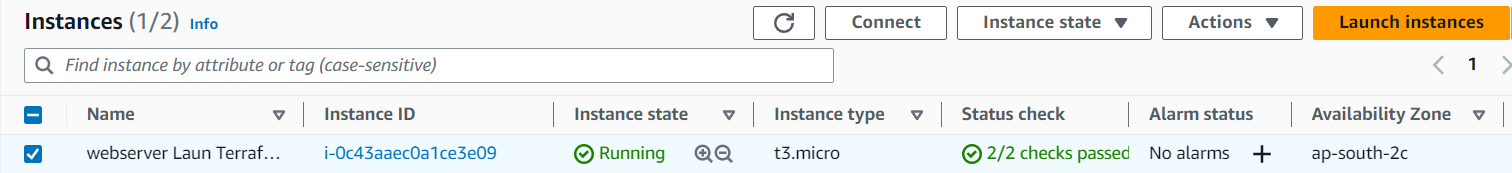
key\_name = "HYD-PEM-HLV"

tags = {

Name = "webserver"

}

}

* terraform init 🡪terraform plan 🡪 terraform apply This Terraform code provisions an AWS security group and an EC2 instance, which can be used to host a web server with SSH access allowed from any IP address.
* The provider block configures the AWS provider and sets the region to "ap-south-2". The profile option is commented out, which means it will use the default AWS profile.
* The "aws\_security\_group" resource named "pubsgp" is defined. It creates a security group named "allow\_tls" with a description of "Allow TLS inbound traffic". It allows inbound SSH traffic on port 22 from any IP address (0.0.0.0/0).
* The "aws\_instance" resource named "web" It creates an EC2 instance with the specified (AMI), instance type, and other configurations. It associates a public IP address, assigns the security group "aws\_security\_group.pubsgp.id" to the instance, and specifies a key pair named "HYD-PEM-HLV" for SSH access.
* It also assigns a tag "Name" with the value "webserver" to the instance.
* **Create a VPC with 2 subnets one is private and another one public create a nat gat way setup and associate the same in via terraform**

# Terraform Block

terraform {

required\_providers {

aws = {

source = "hashicorp/aws"

version = "~> 3.0"

}

}

}

# Configure the AWS Provider

provider "aws" {

region = "ap-south-2"

}

# Create VPC

resource "aws\_vpc" "vpcterraform" {

cidr\_block = "10.0.0.0/16"

instance\_tenancy = "default"

tags = {

Name = "vpcterraform"

}

}

# SUBNET PUBLIC

resource "aws\_subnet" "subpubterra" {

vpc\_id = aws\_vpc.vpcterraform.id

cidr\_block = "10.0.1.0/24"

availability\_zone = "ap-south-2a"

tags = {

Name = "subpubterra"

}

}

# SUBNET PRIVATE

resource "aws\_subnet" "subpvtterra" {

vpc\_id = aws\_vpc.vpcterraform.id

cidr\_block = "10.0.2.0/24"

availability\_zone = "ap-south-2a"

tags = {

Name = "subpvtterra"

}

}

# INTERNET GATEWAY

resource "aws\_internet\_gateway" "itgatewayterra" {

vpc\_id = aws\_vpc.vpcterraform.id

tags = {

Name = "itgatewayterra"

}

}

# RT PUB

resource "aws\_route\_table" "rtpubterra" {

vpc\_id = aws\_vpc.vpcterraform.id

route {

cidr\_block = "0.0.0.0/0"

gateway\_id = aws\_internet\_gateway.itgatewayterra.id

}

tags = {

Name = "rtpubterra"

}

}

# RTSUB-PUB - ASSOCIATION

resource "aws\_route\_table\_association" "rtassopubterra" {

subnet\_id = aws\_subnet.subpubterra.id

route\_table\_id = aws\_route\_table.rtpubterra.id

}

# RT PVT

resource "aws\_route\_table" "rtpvtterra" {

vpc\_id = aws\_vpc.vpcterraform.id

tags = {

Name = "rtpvtterra"

}

}

# RTSUB-PVT - ASSOCIATION

resource "aws\_route\_table\_association" "rtassopvtterra" {

subnet\_id = aws\_subnet.subpvtterra.id

route\_table\_id = aws\_route\_table.rtpvtterra.id

}

# Create Security group

resource "aws\_security\_group" "mysgpubterra" {

name = "allow\_tls"

description = "Allow TLS inbound traffic"

vpc\_id = aws\_vpc.vpcterraform.id

ingress {

description = "http"

from\_port = 80

to\_port = 80

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

ingress {

description = "ssh"

from\_port = 22

to\_port = 22

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

egress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

ipv6\_cidr\_blocks = ["::/0"]

}

tags = {

Name = "allow\_tls"

}

}

# EIP

resource "aws\_eip" "myeipterra" {

vpc = true

}

# NAT GATEWAY

resource "aws\_nat\_gateway" "tnatterra" {

allocation\_id = aws\_eip.myeipterra.id

subnet\_id = aws\_subnet.subpubterra.id

tags = {

Name = "gw NATterra"

}

}

# EC2-PUBLIC- Creation

resource "aws\_instance" "webnatterra" {

ami = "ami-0b584e5449fa1dd56"

instance\_type = "t3.micro"

associate\_public\_ip\_address = true

vpc\_security\_group\_ids = [aws\_security\_group.mysgpubterra.id]

subnet\_id = aws\_subnet.subpubterra.id

tags = {

Name = "Terraform-KING"

}

}

# EC2-PRIVATE-Creation

resource "aws\_instance" "DBterra" {

ami = "ami-0b584e5449fa1dd56"

instance\_type = "t3.micro"

vpc\_security\_group\_ids = [aws\_security\_group.mysgpubterra.id]

subnet\_id = aws\_subnet.subpvtterra.id

key\_name = "HYD-PEM-HLV"

tags = {

Name = "DATABASE-TERRAFORM"

}

}

Terraform plan

Terraform apply

Yes

Check the instance and connect

* **Variable concept call**

Main.tf

# Create EC2 instance

resource "aws\_instance" "my-ec2-vm" {

ami = var.ec2\_ami

instance\_type = var.ec2\_instance\_type

count = var.ec2\_instance\_count

tags = {

"Name" = "myec2vm"

}

}

Provider.tf

# Terraform Block

terraform {

required\_providers {

aws = {

source = "hashicorp/aws"

version = "~> 3.0"

}

}

}

# Provider Block

provider "aws" {

region = "ap-south-2"

profile = "default"

}

Var.tf

# Input Variables

variable "aws\_regions\_mumbai" {

description = "Region in which AWS resources to be created"

type = string

default = "ap-south-2"

}

variable "ec2\_ami" {

description = "AMI ID"

type = string

default = "ami-0b584e5449fa1dd56" # Ubuntu, 22.04 LTS Linux AMI ID

}

variable "ec2\_instance\_type" {

description = "EC2 Instance Type"

type = string

default = "t3.micro"

}

variable "ec2\_instance\_count" {

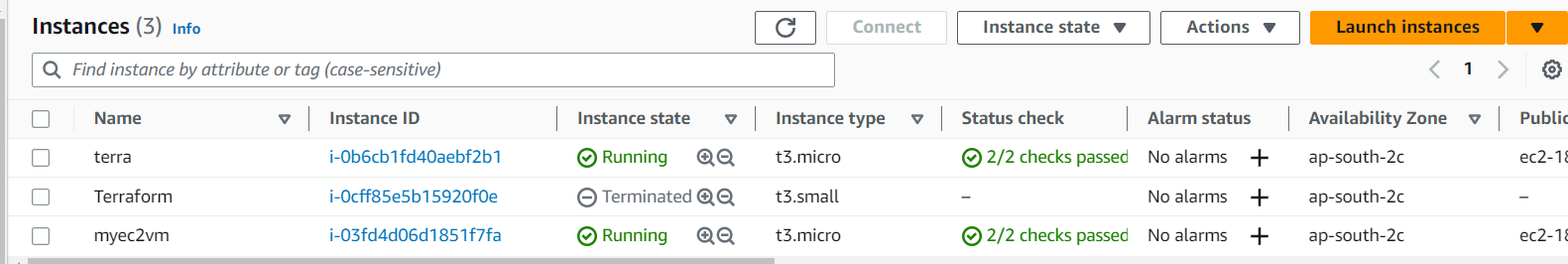
description = "EC2 Instance Count"

type = number

default = 1

}

* Create a EC2 and launch
* Install terraform
* Install awscli
* Configure aws using access key and secret key
* Create tf files of main,var,provider
* Terraform init
* Terraform plan
* Terraform apply
* Yes



* Terraform destroy to delete all resources
* **Create S3 bucket with version enabling bucket should be public**

terraform {

required\_providers {

aws = {

source = "hashicorp/aws"

version = "~> 4.0"

}

}

}

# Configure the AWS Provider

provider "aws" {

region = "ap-south-2"

}

# Create S3 bucket

resource "aws\_s3\_bucket" "citybucket2" {

bucket = "my-tf-example-bucket586"

versioning {

enabled = true

}

}

resource "aws\_s3\_bucket\_ownership\_controls" "citybucket2" {

bucket = aws\_s3\_bucket.citybucket2.id

rule {

object\_ownership = "BucketOwnerPreferred"

}

}

resource "aws\_s3\_bucket\_public\_access\_block" "citybucket2" {

bucket = aws\_s3\_bucket.citybucket2.id

block\_public\_acls = false

block\_public\_policy = false

ignore\_public\_acls = false

restrict\_public\_buckets = false

}

resource "aws\_s3\_bucket\_acl" "citybucket" {

depends\_on = [

aws\_s3\_bucket\_ownership\_controls.citybucket2,

aws\_s3\_bucket\_public\_access\_block.citybucket2,

]

bucket = aws\_s3\_bucket.citybucket2.id

acl = "public-read"

}

