Terraform

Introduction to Terraform

Terraform is an infrastructure-as-code (IaC) tool developed by Hashicorp.

It enables you to provision your infrastructure in a simple, efficient, and declarative manner through repeatable code.

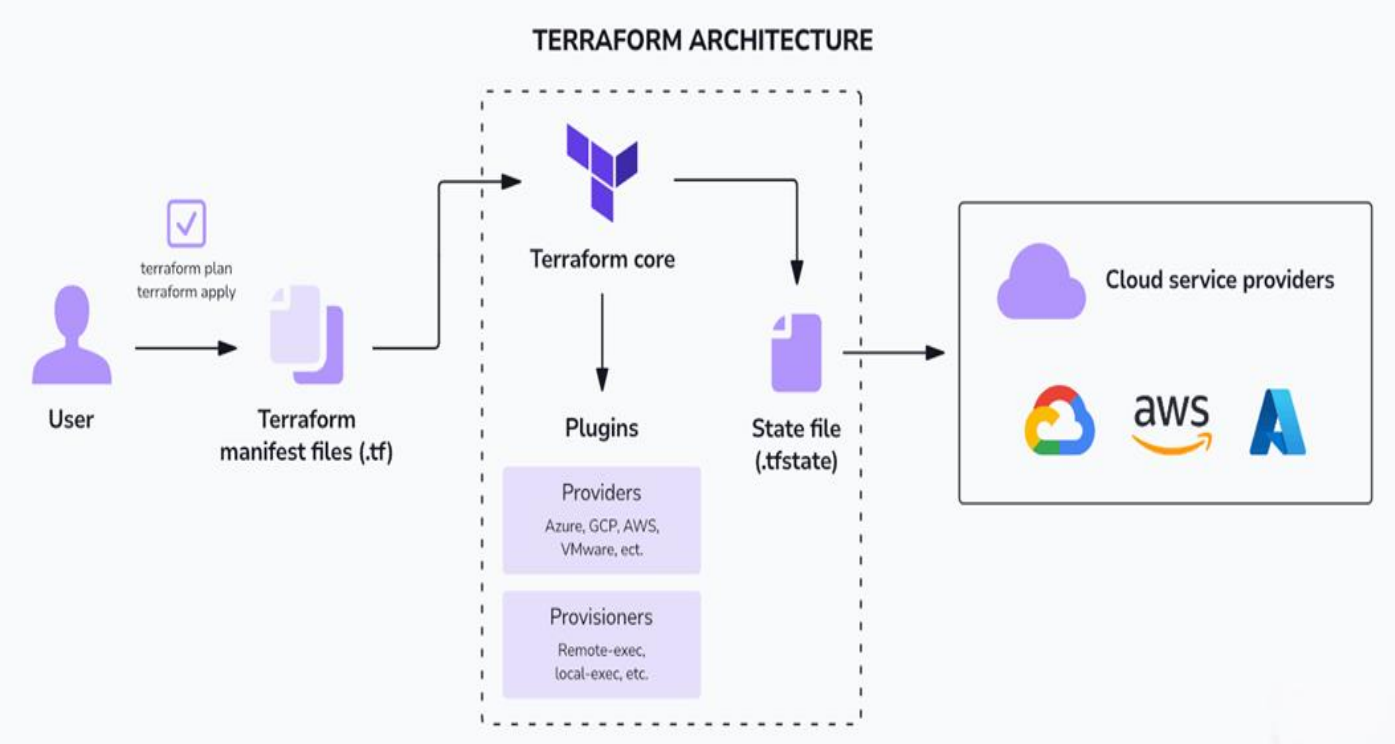
One of Terraform's key features is its cloud-agnostic nature; you can deploy infrastructure to any cloud environment, such as Azure, AWS, Google Cloud, and VMware, as well as on-premises environments.

This benefit allows you to automate your cloud infrastructure without having to learn or adopt different tools for each cloud service provider.

Prerequisites

- Terraform installed (https://developer.hashicorp.com/terraform/downloads)
- Basic understanding of cloud platforms (AWS, Azure, GCP)
- Basic CLI knowledge

Basic components of Terraform architecture



The basic components of Terraform architecture include Terraform core, providers, configuration files, and state management.

- **Terraform core** is the engine that reads configuration files, manages state, and executes plans to apply changes.
- **Providers** are responsible for interacting with APIs of services like AWS, Azure, or Kubernetes to provision resources.
- **Configuration files**, written in HashiCorp Configuration Language (HCL), define the desired infrastructure in a human-readable format.
- **State management** tracks the current infrastructure setup, allowing Terraform to compare it with the desired state and determine necessary changes.

How to structure Terraform files

Terraform files always end with the extension ".tf". The basic Terraform file structure contains the following elements.

Terraform block

A Terraform block specifies the required providers for Terraform to execute the script. This block also contains the source block, which specifies where Terraform should download the provider and the required version. e.g.

```
terraform {
 required_providers {
  azurerm = {
   source  = "hashicorp/azurerm"
   version = "=3.0.0"
  }
 }
}
```

Provider block

A provider block specifies the cloud provider and the API credentials required to connect to the provider's services. It includes the provider name, version, access key, and secret key.

e.g. if you are using Azure as your service provider, it would look as follows:

```
provider "azurerm" {
 features {}
 subscription_id = "00000000-0000-0000-0000-000000000000"
 tenant_id = "11111111-1111-1111-1111-111111111111"
}
```

Output block

An output block defines the output values generated by the Terraform configuration. It includes the output name and value.

```
output "resource_group_id" {
 value = azurerm_resource_group.rg.id
}
```

Resource block

A resource block represents a particular resource in the cloud provider's services. It includes the resource type, name, and configuration details. This is the main block that specifies the type of resource we are trying to deploy.

Below is an example of creating a resource group in Azure:

```
resource "azurerm_resource_group" "example" {
  name = "example"
  location = "West Europe"
}
```

Data block

A data block fetches data from the provider's services, which can be used in resource blocks. It includes the data type and configuration details.

This is used in scenarios where the resource has already been deployed and you would like to retrieve its details.

The code snippet below helps you fetch details of an existing resource group that has already been deployed.

```
data "azurerm_resource_group" "example" {
  name = "existing"
}
```

Variable block

A variable block defines input variables for the Terraform configuration. It includes the variable name, type, and default value.

The following is an example of a variable block in Terraform:

```
variable "resource_group_name" {
  default = "myTFResourceGroup"
}
```

Basic Terraform workflow

1. **Write:** to author your IaC in a Terraform configuration file with all the required blocks.

2. **Init**: to initialize the Terraform working directory and download any necessary plugins.

3. **Plan: to** show what actions Terraform will take to achieve the desired state defined in your configuration, without making any changes.

4. **Apply: to** execute the changes proposed in the plan, creating, updating, or deleting infrastructure

5. Destroy: to delete all resources defined in your Terraform configuration. It helps clean up your infrastructure by removing everything that was provisioned.

Terraform Most used Commands

Provider Configuration (AWS)

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

Creating an EC2 Instance

```
resource "aws_instance" "example" {
  ami           = "ami-0c55b159cbfafe1f0"
  instance_type = "t2.micro"
}
```

Using Variables

```
variable "instance_type" {
  default = "t2.micro"
}
resource "aws_instance" "example" {
  ami           = "ami-0c55b159cbfafe1f0"
  instance_type = var.instance_type
}
```

Output Values

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

Using Terraform State

.tfstate file holds infrastructure state

```
terraform show      # View current state
```

Creating an S3 Bucket

```
resource "aws_s3_bucket" "my_bucket" {
  bucket = "my-unique-bucket-name-123"
  acl    = "private"
}
```

Using Locals

```
locals {
  environment = "dev"
}

output "env" {
  value = local.environment
}
```

Data Sources Example

```
data "aws_ami" "latest" {
  most_recent = true
  owners      = ["amazon"]
  filter {
    name   = "name"
    values = ["amzn2-ami-hvm-*-x86_64-gp2"]
  }
}
```

Using Multiple Providers

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

provider "aws" {
  alias  = "oregon"
  region = "us-west-2"
}
```

Resource with Depends On

```
resource "aws_instance" "example" {
  ami           = "ami-0c55b159cbfafe1f0"
  instance_type = "t2.micro"
  depends_on    = [aws_s3_bucket.my_bucket]
}
```

Using Terraform Modules

```
module "vpc" {
  source = "terraform-aws-modules/vpc/aws"
  version = "3.5.0"
  name = "my-vpc"
  cidr = "10.0.0.0/16"
}
```

Creating Security Group

```
resource "aws_security_group" "allow_ssh" {
  name        = "allow_ssh"
  description = "Allow SSH inbound traffic"

  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

Azure Resource Group Creation

```
provider "azurerm" {
  features {}
}
resource "azurerm_resource_group" "example" {
  name     = "example-resources"
  location = "East US"
}
```

Using Count Parameter

```
resource "aws_instance" "multiple" {
  count         = 3
  ami           = "ami-0c55b159cbfafe1f0"
  instance_type = "t2.micro"
}
```

Using For Each

```
resource "aws_s3_bucket" "buckets" {
  for_each = toset(["bucket1", "bucket2"])
  bucket   = each.value
  acl      = "private"
}
```

Terraform Workspace Example

```
terraform workspace new dev
terraform workspace select dev
```

Remote Backend (S3)

```
terraform {
  backend "s3" {
    bucket = "my-tf-state-bucket"
    key    = "state.tfstate"
    region = "us-east-1"
  }
}
```

Conditional Expressions

```
variable "is_prod" {
  default = false
}
resource "aws_instance" "example" {
  instance_type = var.is_prod ? "t3.large" : "t2.micro"
}
```

Dynamic Blocks

```
resource "aws_security_group" "example" {
 name = "example"
 dynamic "ingress" {
  for_each = [22, 80, 443]
  content {
   from_port   = ingress.value
   to_port     = ingress.value
   protocol    = "tcp"
   cidr_blocks = ["0.0.0.0/0"]
  }
 }
}
```

Terraform Version Constraint

```
terraform {
 required_version = ">= 1.0.0"
}
```

Auto-scaling Group Setup

```
resource "aws_autoscaling_group" "example" {
 name                 = "example-asg"
 max_size             = 3
 min_size             = 1
 desired_capacity     = 2
 launch_configuration = aws_launch_configuration.example.name
 vpc_zone_identifier  = [aws_subnet.example.id]
}
```

VPC with Public and Private Subnets

```
module "vpc" {
  source = "terraform-aws-modules/vpc/aws"
  name   = "my-vpc"
  cidr   = "10.0.0.0/16"
  public_subnets  = ["10.0.1.0/24"]
  private_subnets = ["10.0.2.0/24"]
}
```

RDS Instance Provisioning

```
resource "aws_db_instance" "example" {
  identifier = "example-db"
  engine     = "mysql"
  instance_class = "db.t2.micro"
  allocated_storage = 20
  username   = "admin"
  password   = "password"
  skip_final_snapshot = true
}
```

Route 53 DNS Record Creation

```
resource "aws_route53_record" "www" {
  zone_id = aws_route53_zone.primary.zone_id
  name    = "www"
  type    = "A"
  ttl     = "300"
  records = [aws_instance.example.public_ip]
}
```

CloudWatch Alarm Configuration

```
resource "aws_cloudwatch_metric_alarm" "cpu_alarm" {
  alarm_name        = "cpu-util-high"
  comparison_operator = "GreaterThanThreshold"
  evaluation_periods  = "2"
  metric_name       = "CPUUtilization"
  namespace         = "AWS/EC2"
  period            = "120"
  statistic         = "Average"
  threshold         = "80"
  alarm_description   = "This metric monitors EC2 CPU utilization"
  dimensions = {
    InstanceId = aws_instance.example.id
  }
}
```

IAM Role and Policy Creation

```
resource "aws_iam_role" "example" {
  name = "example-role"
  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [{
      Action = "sts:AssumeRole"
      Effect = "Allow"
      Principal = {
        Service = "ec2.amazonaws.com"
      }
    }]
  })
}
```

Lambda Function Deployment

```
resource "aws_lambda_function" "example" {
  function_name = "example_lambda"
  handler      = "index.handler"
  runtime      = "python3.8"
  role         = aws_iam_role.example.arn
  filename     = "lambda_function_payload.zip"
}
```

GCP Compute Instance Example

```
provider "google" {
  project = "my-project"
  region  = "us-central1"
}
resource "google_compute_instance" "vm_instance" {
  name         = "test-instance"
  machine_type = "f1-micro"
  zone         = "us-central1-a"
  boot_disk {
    initialize_params {
      image = "debian-cloud/debian-9"
    }
  }
  network_interface {
    network = "default"
    access_config {}
  }
}
```

Kubernetes Cluster with Terraform

```
provider "kubernetes" {
  config_path = "~/.kube/config"
}
resource "kubernetes_namespace" "example" {
  metadata {
    name = "example"
  }
}
```

Azure Virtual Network Creation

```
resource "azurerm_virtual_network" "example" {
  name                = "example-vnet"
  address_space       = ["10.1.0.0/16"]
  location            = azurerm_resource_group.example.location
  resource_group_name = azurerm_resource_group.example.name
}
```

Azure Storage Account Provisioning

```
resource "azurerm_storage_account" "example" {
  name                     = "examplestorageacct"
  resource_group_name      = azurerm_resource_group.example.name
  location                 = azurerm_resource_group.example.location
  account_tier             = "Standard"
  account_replication_type = "LRS"
}
```

Conclusion

This tutorial provides over 30 Terraform scripts and examples to help you get started with Infrastructure as Code across AWS, Azure, and GCP.