Terraform with AWS

1.Terraform workflow

Terraform's primary function is to create, modify, and destroy infrastructure resources to match the desired state described in a Terraform configuration.

When people refer to "running Terraform," they generally mean performing these provisioning actions in order to affect real infrastructure objects. The Terraform binary has many other subcommands for a wide variety of administrative actions, but these basic provisioning tasks are the core of Terraform.

Terraform's provisioning workflow relies on three commands: plan, apply, and destroy. All of these commands require an initialized working directory, and all of them act only upon the currently selected workspace.

2. Terraform PLAN: 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 being managed with the current working directory and workspace. It uses state data to determine which real objects correspond to which declared resources, and checks the current state of each resource using the relevant infrastructure provider's API.

3.Terraform APPLY : The terraform apply command performs a plan just like terraform plan does, but then actually carries out the planned changes to each resource using the relevant infrastructure provider's API. It asks for confirmation from the user before making any changes, unless it was explicitly told to skip approval.

4.Terraform DESTROY : The terraform destroy command destroys all of the resources being managed by the current working directory and workspace, using state data to determine which real world objects correspond to managed resources. Like terraform apply, it asks for confirmation before proceeding.

5.Creating an EC2 instance:

The following code uses the aws\_instance resource to deploy an EC2 Instance:

resource "aws\_instance" "example" {

ami = "ami-0c55b159cbfafe1f0"

instance\_type = "t2.micro"

}

The general syntax for a Terraform resource is:

resource "<PROVIDER>\_<TYPE>" "<NAME>" {

[CONFIG …]

}

Where PROVIDER is the name of a provider (e.g., aws), TYPE is the type of resources to create in that provider (e.g., instance), NAME is an identifier you can use throughout the Terraform code to refer to this resource (e.g., example), and CONFIG consists of one or more arguments that are specific to that resource (e.g., ami = "ami-0c55b159cbfafe1f0").

The terraform binary contains the basic functionality for Terraform, but it does not come with the code for any of the providers (e.g., the AWS provider, Azure provider, GCP provider, etc), so when first starting to use Terraform, you need to run terraform init to tell Terraform to scan the code, figure out what providers you’re using, and download the code for them. By default, the provider code will be downloaded into a .terraform folder, which is Terraform’s scratch directory .

Now that you have the provider code downloaded, run the terraform plan command.

The plan command lets you see what Terraform will do before actually doing it. This is a great way to sanity check your changes before unleashing them onto the world. The output of the plan command is a little like the output of the diff command: resources with a plus sign (+) are going to be created, resources with a minus sign (-) are going to be deleted, and resources with a tilde sign (~) are going to be modified in-place.

To actually create the instance, run the terraform apply command.

You’ll notice that the apply command shows you the same plan output and asks you to confirm if you actually want to proceed with this plan. So while plan is available as a separate command, it’s mainly useful for quick sanity checks and during code reviews.

6.Variables:

Input variables serve as parameters for a Terraform module, allowing aspects of the module to be customized without altering the module's own source code, and allowing modules to be shared between different configurations.

Each input variable accepted by a module must be declared using a variable block:

variable "image\_id" {

type = string}

The label after the variable keyword is a name for the variable, which must be unique among all variables in the same module. This name is used to assign a value to the variable from outside and to reference the variable's value from within the module.

The name of a variable can be any valid identifier except the following: source, version, providers, count, for\_each, lifecycle, depends\_on, locals.

These names are reserved for meta-arguments in module configuration blocks, and cannot be declared as variable names.

Terraform CLI defines the following optional arguments for variable declarations:

default - A default value which then makes the variable optional.

type - This argument specifies what value types are accepted for the variable.

description - This specifies the input variable's documentation.

validation - A block to define validation rules, usually in addition to type constraints.

sensitive - Limits Terraform UI output when the variable is used in configuration.

7.Creating VPC,Subnet and security groups:

vpc.tf:

# vpc.tf

# Create VPC/Subnet/Security Group/Network ACL

provider "aws" {

version = "~> 2.0"

access\_key = var.access\_key

secret\_key = var.secret\_key

region = var.region

}

# create the VPC

resource "aws\_vpc" "My\_VPC" {

cidr\_block = var.vpcCIDRblock

instance\_tenancy = var.instanceTenancy

enable\_dns\_support = var.dnsSupport

enable\_dns\_hostnames = var.dnsHostNames

tags = {

Name = "My VPC"

}

} # end resource

# create the Subnet

resource "aws\_subnet" "My\_VPC\_Subnet" {

vpc\_id = aws\_vpc.My\_VPC.id

cidr\_block = var.subnetCIDRblock

map\_public\_ip\_on\_launch = var.mapPublicIP

availability\_zone = var.availabilityZone

tags = {

Name = "My VPC Subnet"

}

} # end resource

# Create the Security Group

resource "aws\_security\_group" "My\_VPC\_Security\_Group" {

vpc\_id = aws\_vpc.My\_VPC.id

name = "My VPC Security Group"

description = "My VPC Security Group"

# allow ingress of port 22

ingress {

cidr\_blocks = var.ingressCIDRblock

from\_port = 22

to\_port = 22

protocol = "tcp"

}

# allow egress of all ports

egress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

tags = {

Name = "My VPC Security Group"

Description = "My VPC Security Group"

}

} # end resource

# create VPC Network access control list

resource "aws\_network\_acl" "My\_VPC\_Security\_ACL" {

vpc\_id = aws\_vpc.My\_VPC.id

subnet\_ids = [ aws\_subnet.My\_VPC\_Subnet.id ]

# allow ingress port 22

ingress {

protocol = "tcp"

rule\_no = 100

action = "allow"

cidr\_block = var.destinationCIDRblock

from\_port = 22

to\_port = 22

}

# allow ingress port 80

ingress {

protocol = "tcp"

rule\_no = 200

action = "allow"

cidr\_block = var.destinationCIDRblock

from\_port = 80

to\_port = 80

}

# allow ingress ephemeral ports

ingress {

protocol = "tcp"

rule\_no = 300

action = "allow"

cidr\_block = var.destinationCIDRblock

from\_port = 1024

to\_port = 65535

}

# allow egress port 22

egress {

protocol = "tcp"

rule\_no = 100

action = "allow"

cidr\_block = var.destinationCIDRblock

from\_port = 22

to\_port = 22

}

# allow egress port 80

egress {

protocol = "tcp"

rule\_no = 200

action = "allow"

cidr\_block = var.destinationCIDRblock

from\_port = 80

to\_port = 80

}

# allow egress ephemeral ports

egress {

protocol = "tcp"

rule\_no = 300

action = "allow"

cidr\_block = var.destinationCIDRblock

from\_port = 1024

to\_port = 65535

}

tags = {

Name = "My VPC ACL"

}

} # end resource

# Create the Internet Gateway

resource "aws\_internet\_gateway" "My\_VPC\_GW" {

vpc\_id = aws\_vpc.My\_VPC.id

tags = {

Name = "My VPC Internet Gateway"

}

} # end resource

# Create the Route Table

resource "aws\_route\_table" "My\_VPC\_route\_table" {

vpc\_id = aws\_vpc.My\_VPC.id

tags = {

Name = "My VPC Route Table"

}

} # end resource

# Create the Internet Access

resource "aws\_route" "My\_VPC\_internet\_access" {

route\_table\_id = aws\_route\_table.My\_VPC\_route\_table.id

destination\_cidr\_block = var.destinationCIDRblock

gateway\_id = aws\_internet\_gateway.My\_VPC\_GW.id

} # end resource

# Associate the Route Table with the Subnet

resource "aws\_route\_table\_association" "My\_VPC\_association" {

subnet\_id = aws\_subnet.My\_VPC\_Subnet.id

route\_table\_id = aws\_route\_table.My\_VPC\_route\_table.id

} # end resource

# end vpc.tf

variables.tf:

# variables.tf

variable "access\_key" {

default = "<PUT IN YOUR AWS ACCESS KEY>"

}

variable "secret\_key" {

default = "<PUT IN YOUR AWS SECRET KEY>"

}

variable "region" {

default = "us-east-1"

}

variable "availabilityZone" {

default = "us-east-1a"

}

variable "instanceTenancy" {

default = "default"

}

variable "dnsSupport" {

default = true

}

variable "dnsHostNames" {

default = true

}

variable "vpcCIDRblock" {

default = "10.0.0.0/16"

}

variable "subnetCIDRblock" {

default = "10.0.1.0/24"

}

variable "destinationCIDRblock" {

default = "0.0.0.0/0"

}

variable "ingressCIDRblock" {

type = list

default = [ "0.0.0.0/0" ]

}

variable "egressCIDRblock" {

type = list

default = [ "0.0.0.0/0" ]

}

variable "mapPublicIP" {

default = true

}

# end of variables.tf

8.Launching RDS using terraform:

resource "aws\_db\_instance" "default" {

allocated\_storage = 20

identifier = "sampleinstance"

storage\_type = "gp2"

engine = "mysql"

engine\_version = "5.7"

instance\_class = "db.m4.medium"

name = "sample"

username = "dbadmin"

password = "DBAdmin@5#41$32"

parameter\_group\_name = "default.mysql5.7"

}

aws\_db\_instance – RDS instance as a resource

identifier – A unique name for the DB Instance

engine\_version – DB version to use

9. S3 using terraform

Module S3

s3.tf

resource "aws\_s3\_bucket" "demos3" {

bucket = "${var.bucket\_name}"

acl = "${var.acl\_value}"

}

variables.tf

variable "bucket\_name" {}

variable "acl\_value" {

default = "private"

}

Main.tf

provider "aws" {

access\_key = "${var.aws\_access\_key}"

secret\_key = "${var.aws\_secret\_key}"

region = "${var.region}"

}

module "s3" {

source = "path-to-S3-folder"

#bucket name should be unique

bucket\_name = "Bucket-name" }

Explanation

* We have a block with the key name “resource” with resource type “aws\_s3\_bucket”– which we want to create. It has a fixed value, and it depends on the provider. Here we have an AWS S3 resource where AWS is our provider and S3 is our resource. “Demos3” is the resource name that the user provides.
* Bucket and ACL are the argument types for our resource. We can have different arguments according to our needs and their corresponding values.
* Either we can provide value directly or use the var.tf file to declare the value of an argument.

10. IAM policies and users with terraform

Main.tf

terraform {

required\_providers {

aws = {

source = "hashicorp/aws"

version = "3.42.0"

}}}

provider "aws" {

region = var.region

}

data "aws\_iam\_policy\_document" "s3\_policy" {

statement {

actions = ["s3:ListAllMyBuckets"]

resources = ["arn:aws:s3:::\*"]

effect = "Allow"

}

statement {

actions = ["s3:\*"]

resources = [aws\_s3\_bucket.bucket.arn]

effect = "Allow"

}}

resource "aws\_iam\_user" "new\_user" {

name = "new\_user"}

resource "aws\_iam\_access\_key" "my\_access\_key" {

user = aws\_iam\_user.new\_user.name

pgp\_key = var.pgp\_key

}

resource "random\_pet" "pet\_name" {

length = 3

separator = "-"

}

resource "aws\_s3\_bucket" "bucket" {

bucket = "${random\_pet.pet\_name.id}-bucket"

acl = "private"

tags = {

Name = "My bucket"

Environment = "Dev"

}

}

resource "aws\_iam\_policy" "policy" {

name = "${random\_pet.pet\_name.id}-policy"

description = "My test policy"

policy = data.aws\_iam\_policy\_document.s3\_policy.json

}

output "rendered\_policy" {

value = data.aws\_iam\_policy\_document.s3\_policy.json

}

output "secret" {

value = aws\_iam\_access\_key.my\_access\_key.encrypted\_secret

sensitive = true}

Variables.tf:

variable "region" {

default = "us-east-1"

}

variable "pgp\_key" {

description = "Either a base-64 encoded PGP public key, or a keybase username in the form keybase:username. Used to encrypt the password and the access key on output to the console."

default = ""

}