**Build Infrastructure:**

**Configuration:**

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

**Provider:** Tells in which cloud infra terraform script to be applied and terraform downloads modules for that infra in our case "aws".

**Profile:** refers to the AWS Config File in ~/.aws/credentials on MacOS and Linux .

# vi ec2.tf

provider "aws" {

profile = "default"

region = "us-east-1"

}

resource "aws\_instance" "example" {

ami = "ami-2757f631"

instance\_type = "t2.micro"

vpc\_security\_group\_ids = ["sg-0037ce5c29994a22c"]

}

# terraform init

# terraform plan

# terraform apply

The output has a + next to aws\_instance.example, meaning that Terraform will create this resource. Beneath that, it shows the attributes that will be set. When the value displayed is (known after apply), it means that the value won't be known until the resource is created.

**Formatting and Validating Configurations:**

The **terraform fmt** command enables standardization which automatically updates configurations in the current directory for easy readability and consistency.

**terraform validate** command will check and report errors within modules, attribute names, and value types.

# terraform fmt

# terraform validate

**terraform.tfstate**it keeps track of the IDs of created resources so that Terraform knows what it is managing. You can inspect the current state using terraform show

# terraform show

**Destroy:**

Resources can be destroyed using the **terraform destroy** command. The - prefix indicates that the instance will be destroyed.

# terraform destroy

**Resource Dependencies:**

Terraform configurations can contain multiple resources, multiple resource types, and these types can even span multiple providers.

Here in example we will show Elasticip and ec2 resources are depended on each other.

we use an interpolation to use an attribute from the EC2 instance

The syntax for this interpolation should be straightforward: it requests the "id" attribute from the "aws\_instance.example" resource.

# vi ec2.tf

provider "aws" {

profile = "default"

region = "us-east-1"

}

resource "aws\_instance" "jenkins\_server" {

ami = "ami-2757f631"

instance\_type = "t2.micro"

}

resource "aws\_eip" "ip" {

vpc = true

instance = aws\_instance.example.id

}

Ec2 creation with new security group and vpc subnet

provider "aws" {

region = "ap-south-1"

}

resource "aws\_vpc" "mainvpc" {

cidr\_block = "192.168.3.0/24"

}

resource "aws\_security\_group" "mainsg" {

vpc\_id = aws\_vpc.mainvpc.id

ingress {

protocol = "tcp"

from\_port = 0

to\_port = 65535

}

egress {

from\_port = 0

to\_port = 65535

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

tags = {

Name = "allow\_all"

}

}

resource "aws\_subnet" "new1" {

vpc\_id = "${aws\_vpc.mainvpc.id}"

cidr\_block = "192.168.3.0/25"

map\_public\_ip\_on\_launch = true

tags = {

Name = "new"

}

}

resource "aws\_instance" "terraec2" {

ami = "ami-016ec067d44808c4f"

instance\_type = "t2.micro"

key\_name = "newmum"

subnet\_id = "${aws\_subnet.new1.id}"

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

user\_data = <<-EOT

#! /bin/bash

cd /tmp

sudo yum install -y https://s3.amazonaws.com/ec2-downloads-windows/SSMAgent/latest/linux\_amd64/amazon-ssm-agent.rpm

sudo systemctl start amazon-ssm-agent

EOT

connection {

type = "ssh"

user = "centos"

private\_key = file("~/Documents/newkey1.pem")

host = self.public\_ip

}

provisioner "remote-exec" {

inline = [

"sudo yum -y install httpd",

"sudo systemctl start httpd"

]

}

tags = {

Name = "terrafromec2"

}

}

**Implicit and Explicit Dependencies:**

In the example above, the reference to aws\_instance.example.id creates an *implicit dependency* on the aws\_instance named example.

By default terraform reads the tf file checks the dependency and apply resources. Sometimes there are dependencies between resources that are *not* visible to Terraform.

In this case we can right **depends\_on** condition in required resources.For example s3 need to create before ec2. So for we will mention depends\_on s3 condition in ec2 resource.

resource "aws\_s3\_bucket" "example" {

bucket = "terraform-getting-started-guide"

acl = "private"

}

resource "aws\_instance" "example" {

ami = "ami-2757f631"

instance\_type = "t2.micro"

*# Tells Terraform that this EC2 instance must be created only after the*

*# S3 bucket has been created.*

depends\_on = [aws\_s3\_bucket.example]

}