**gcr.io/distroless/base multistage**

172.31.0.0/20

Terraform

Multiple reagion

provider "aws" {

    alias = "us-east-1"

    region = "us-east-1"

}

provider "aws" {

    alias = "us-west-2"

    region = "us-wast-2"

}

resource "aws\_instance" "multi" {

    ami = "56789"

    instance\_type = "t2.micro"

    subnet\_id = "567898765678"

    provider = "aws.us-east-1"

}

resource "aws\_instance" "multi" {

    ami = "56789"

    instance\_type = "t2.micro"

    subnet\_id = "567898765678"

    provider = "aws.us-west-1"

}

A Terraform lock file, named \*.lock.hcl, is automatically generated by Terraform to maintain the integrity of the dependencies used within your Terraform configurations. Here’s what it does and why it’s important:

### Purpose of a Terraform Lock File:

1. **Dependency Version Control:** The lock file records the specific versions of provider plugins (e.g., AWS, Azure, etc.) that Terraform should use. This ensures that everyone using the same Terraform configuration is working with the exact same versions of providers, which reduces the risk of inconsistencies or unexpected changes in behavior.
2. **Reproducible Builds:** By locking the provider versions, the lock file helps make your Terraform deployments reproducible. This is crucial for ensuring that the sainfrastructure is created or modified across different environments (e.g., dev, staging, production).
3. **Security:** It helps prevent introducing vulnerabilities by locking to specific versions of providers that have been verified to work correctly.

### Locking the File (Preventing Updates):

If you want to prevent the lock file from being accidentally updated, you can apply filesystem permissions to make it read-only. Here’s how you can do it:

* **On Unix/Linux/MacOS:**

bash

chmod 444 .terraform.lock.hcl

Input variable: when you will going to give input to pass the value ex: create variable ->input🡪Variable

#inside the

variable "ami" {

    description = "value  ami value"

type = "string"

}

#mention the var

resource "aws" "forvariable" {

    ami = var.ami

}

Output variable: when you want to print the public or anything just you can mention in the output.tf

output "public\_ip" {

    description = "public ip"

    value = aws\_instance.multi.public\_ip

}

Main.tf

Provider.tf

Input.tf

Output.tf

Terraform.tfvars

TFvars:

In the terraform if you want to create ec2 if you are using the input variable you will mention like t2.micro

So if another person want to create the same means he need to change so

Using the tf vars you can just mention example ami \

Conditional expression:

Ex if you restrict like only access to s3 public for dev

Not for production env

condition ? true\_val : false\_val

  ingress {

    from\_port   = 22

    to\_port     = 22

    protocol    = "tcp"

    cidr\_blocks = var.environment == "production" ? [var.production\_subnet\_cidr] : [var.development\_subnet\_cidr]

  }

}

Module:

Instead of running all the resource like ec2,s3,vpc in the one main.tf

You can write create module for the ec2 seperately

* If other team required means you can give and if you required you can ask for the vpc module
* provider "aws" {
* region = "us-east-1"
* }
* resource "aws\_instance" "nnagar" {
* ami = var.ami
* instance\_type = var.instance\_type
* subnet\_id = var.subnet\_id
* associate\_public\_ip\_address = true
* }
* Module includes🡪Main.tf,variable.tf,output.tf

In the main.tf just you can call the variables but in the terraform.tfvars

You can mention clearly like ami=456789,instance.type,subnet.

* Varaible.tf

variable "ami11" {

    description = "value for ami"

}

* Tearrform.tfvars

ami11 = "456789876"

instance\_type = "t2.micro"

* Main.tf

resource "aws\_instance" "name" {

    instance\_type = var.instance\_type

    ami = var.ami11

    subnet\_id = var.subnet-56789

}

* Output.tf

output "public-ip-address" {

  value = aws\_instance.example.public\_ip

}

* Provider.tf

You can create module in terraform if you want just write module

Main.tf

module "ec2\_instance" {

    source = "../Day-12/modules"

     ami\_value = "345678"

     instance\_type = "t2.micro"

     subnet\_id = ""

}

If you are in different folder

* provider "aws" {
* region = "us-east-1"
* }
* module "ec2" {
* source = "/workspaces/Terra\_form/Day-15/Day-15/module"
* ami = "ami-0e86e20dae9224db8"
* subnet\_id = "subnet-0a281ef1118b756e5" instance\_type = "t2.micro"

}

* **Statefile**

Statefile is heart of the terraform

Whenever you will cerate ec2 It will store the info in like t2.micro.type everything in the statefile

* If does not record the statefile

If terraform is not there or its nit there evondh ur modifying the ec2 but statefile is not there so it will create again

Ex : ur destroying or updating the infrastructure using staefile only it will it get know

* **Dis-advantage**
* If will store sensitive info it will store in statefile on ur machine
* If the other person will access the statefile then he can modifiy anything
* If u will store in github and 5 pepole have access he had modify and he need to push to pravite version controller system
* **REMOTE BACKEND**

You can store statefile in s3 bucket and you can configure with I AM role and restrict the access

Before applying terraform init so terraform will think like person using remote backend so my statefile is in s3 bucket so now it will compare both terraform backend and s3 bucket

Main.tf

provider "aws" {

    region = "us-west-1"

}

resource "aws\_instance" "naga" {

    instance\_type = "t2.micro"

    ami = "ami-0e86e20dae9224db8"

    subnet\_id = "subnet-0a281ef1118b756e5"

}

resource "aws\_s3\_bucket" "naga" {

    bucket = "nmnmn"

}

Backend.tf

terraform {

  backend "s3" {

    bucket = "nmnmn"

    region = "us-east-1"

    key = "nagara/terraform.tfstate"

  }

}

**Lock file**

Whenever you will do terraform commands that time terraform will take the lock file or it will control the lock file

Ex: bcz two people updating the same time only one person can hold the lockother person should wait until he will complete his task

**DynamoDB**

You can implement the locking mechanism using the dynamodb

terraform {

  backend "s3" {

    bucket = "nmnmn"

    region = "us-east-1"

    key = "nagara/terraform.tfstate"

    dynamodb\_table = "terraform-lock"

  }

}

**Using dynamoDB**

provider "aws" {

    region = "us-west-1"

}

resource "aws\_instance" "naga" {

    instance\_type = "t2.micro"

    ami = "ami-0e86e20dae9224db8"

    subnet\_id = "subnet-0a281ef1118b756e5"

}

resource "aws\_s3\_bucket" "naga" {

    bucket = "nmnmn"

}

resource "aws\_dynamodb\_table" "terraform\_lock" {

    name = "terraform-lock"

    billing\_mode = "PAY\_PER\_REQUEST"

    hash\_key = "LockID"}

 attribute {

    name = "LockID"

    type = "S"

}

**Terraform project**

**|**

**Vpc**

**|**

**Public subnet**

**|**

**Route table(internet gateway)**

**|**

**Ec2—security group**

**|**

**10.190.178.0/0**

**VPC creted**

provider "aws" {

    region = "us-east-1"

}

#this cidr for vpc

variable "cidr" {

    default = "10.0.0.0/17"

}

#need to create public key for ec2 instance #ssh keygen -t rsahead -n 10 ~/.ssh/id\_rsa

resource "aws\_key\_pair" "demo" {

    key\_name = "terrform-demo"

    public\_key = file("~/.ssh/id\_rsa.pub")

}

resource "aws\_vpc" "myvpc" {

    cidr\_block = var.cidr

}

#this cidr block is for subnet

resource "aws\_subnet" "my-aws\_subnet" {

    vpc\_id = aws\_vpc.myvpc.id

    cidr\_block = "10.0.0.0/18"

    availability\_zone = "us-east-1a"

    map\_public\_ip\_on\_launch = true

}

resource "aws\_internet\_gateway" "igw" {

    vpc\_id = aws\_vpc.myvpc.id

}

resource "aws\_route\_table" "RT" {

    vpc\_id = aws\_vpc.myvpc.id

    route {

    cidr\_block = "0.0.0.0/0"

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

  }

}

resource "aws\_route\_table\_association" "rta1" {

    subnet\_id = aws\_subnet.my-aws\_subnet.id

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

}

resource "aws\_security\_group" "webSg" {

    name = "web"

    vpc\_id = aws\_vpc.myvpc.id

     ingress {

    description = "HTTP from VPC"

    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"]

 }

#accces anything from outside

egress {

    from\_port = 0

    to\_port = 0

    protocol = "-1"

    cidr\_blocks = ["0.0.0.0/0"]

}

  tags = {

    Name = "webSg"

  }

}

resource "aws\_instance" "demo1" {

    ami = "ami-0e86e20dae9224db8"

    instance\_type = "t2.micro"

    subnet\_id = aws\_subnet.my-aws\_subnet.id

    key\_name = aws\_key\_pair.demo.key\_name

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

 connection {

      type = "ssh"

      user = "ubuntu"

      private\_key = file("~/.ssh/id\_rsa")

      host = self.public\_ip

    }

    #file provisioner to copy file from local to the ec2 instance

    provisioner "file" {

        source = "/workspaces/Terra\_form/practice/main.tf"

        destination = "/home/ubuntu/app.py"

    }

    #using the remote exec you can install the application also

 provisioner "remote-exec" {

    inline = [

        "echo 'Hello from the remote instance'",

        "sudo apt update -y",

        "sudo apt-get install -y python3-pip",

        "cd /home/ubuntu",

        "sudo pip3 install flask",

        "sudo python3 app.py &",

     ]

 }

}

**Pravite key and public are saved In**

**@NagarajCha ➜ / $ ls ~/.ssh/**

**id\_rsa id\_rsa.pub**

**for verifying the key ->ls -l ~/.ssh/id\_rsa**

**login from visual code/cmd 🡪 ssh -i ~/.ssh/id\_rsa ubuntu@54.234.128.148**

**While creating the VPC got the errors**

**Error: creating EC2 Subnet: operation error EC2: CreateSubnet, https response error StatusCode: 400, RequestID: 5bc00e26-91d8-4621-bd5a-78b7439fb5c2, api error InvalidParameterValue: Value (us-east-1) for parameter availabilityZone is invalid. Subnets can currently only be created in the following availability zones: us-east-1a, us-east-1b, us-east-1c, us-east-1d, us-east-1e, us-east-1f.**

**│**

**│ with aws\_subnet.my-aws\_subnet,**

**│ on main.tf line 22, in resource "aws\_subnet" "my-aws\_subnet":**

**│ 22: resource "aws\_subnet" "my-aws\_subnet" {**

**│**

**╵**

**╷**

**│ Error: creating EC2 Instance: operation error EC2: RunInstances, https response error StatusCode: 400, RequestID: 0e69f160-5d25-476f-89a0-f87dbbafa621, api error InvalidParameter: Security group sg-0f03267b133bf4501 and subnet subnet-0a281ef1118b756e5 belong to different networks.**

**│**

**│ with aws\_instance.demo,**

**│ on main.tf line 80, in resource "aws\_instance" "demo":**

**│ 80: resource "aws\_instance" "demo" {**

**Error: creating EC2 Instance: operation error EC2: RunInstances, https response error StatusCode: 400, RequestID: f3a97281-ba24-4706-bc8f-a1c857c6b9be, api error InvalidParameter: Security group sg-06e665c29c8bf1d6a and subnet subnet-0a281ef1118b756e5 belong to different networks.**

**│**

**│ with aws\_instance.demo,**

**│ on main.tf line 80, in resource "aws\_instance" "demo":**

**│ 80: resource "aws\_instance" "demo" {**

**Provisioner**

**Using the provisioner execute and implement and creation**

**During the execution time it will create some file and destruction**

**Remote Excec:**

**At the time of creating ec2 instance you can connect to the ec2 you will execute install python,java you run all the application**

**Local Excec:**

**during the creation of ec2 you can print the statement like echo command and**

**ex->if 1000 line there means it will not show all the lines show just using the local exec you can copy all the lines**

**File Provisioner:**

**You can copy the files from local to ec2**