TERRAFORM

INTRO:

Terraform is an open source “Infrastructure as Code” tool, created by HashiCorp.

A *declarative* coding tool, Terraform enables developers to use a high-level configuration language called HCL (HashiCorp Configuration Language) to describe the desired “end-state” cloud or on-premises infrastructure for running an application.

Terraform uses a simple syntax, can provision infrastructure across multiple cloud and on-premises.

IAAC:

Infrastructure as a Code (IaaC) is the managing and provisioning of infrastructure through code instead of through manual processes.

With IaC, configuration files are created that contain your infrastructure specifications, which makes it easier to edit and distribute configurations.

IaC allows you to meet the growing needs of infrastructure changes in a scalable and trackable manner.

The infrastructure terraform could handle low-level elements like networking, storage, compute instances, also high-level elements like **SaaS features, DNS entries**, etc.

It is famous for easy to use but not true for complex environments it is not easy.

Terraform is not fully cloud agnostic

WHY:

It is a server orchestration tool (chef, ansible and puppet are configuration tools).

Declarative code

Immutable code

CLOUD ALTERNATES:

AWS -- > CloudFormation templates (JSON/YAML)

AZURE -- > ARM TEMPLATES (JSON)

TERRAFORM(Car) -- > AWS(IOL), AZURE(BP), GCP(HP), VMWARE

FUEL -- > CODE

ADVANTAGES:

Readable code.

Dry run.

Importing of Resources is easy.

Creating of multiple resources.

Can create modules for repeatable code.

DIS ADVANTAGES:

It is 3rd party tool. It takes time to accommodate new services.

BUGS

TERRAFORM SETUP

wget https://releases.hashicorp.com/terraform/1.1.3/terraform_1.1.3_linux_amd64.zip

sudo apt-get install zip -y

Unzip terraform

mv terraform /usr/local/bin/

terraform version

cd ~

mkdir terraform & vim main.tf

write the basic code

Go to IAM andcreate a user called terraform and give both access give admin access.

CREATING AN INSTANCE

```
provider "aws" {
 region= "ap-south-1"
 access_key     = "AKIAWW7WL2JMJKCCMORC"
 secret_key     = "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"
}

resource "aws_instance" "web" {
  ami= "ami-08e4e35cccc6189f4"
  instance_type = "t2.micro"

  tags = {
    name = "web-server"
  }
}
```

terraform init : now terraform will be initialized

Now see the hidden files you will find a terraform directory

terraform plan : Read config file and compare local state file.

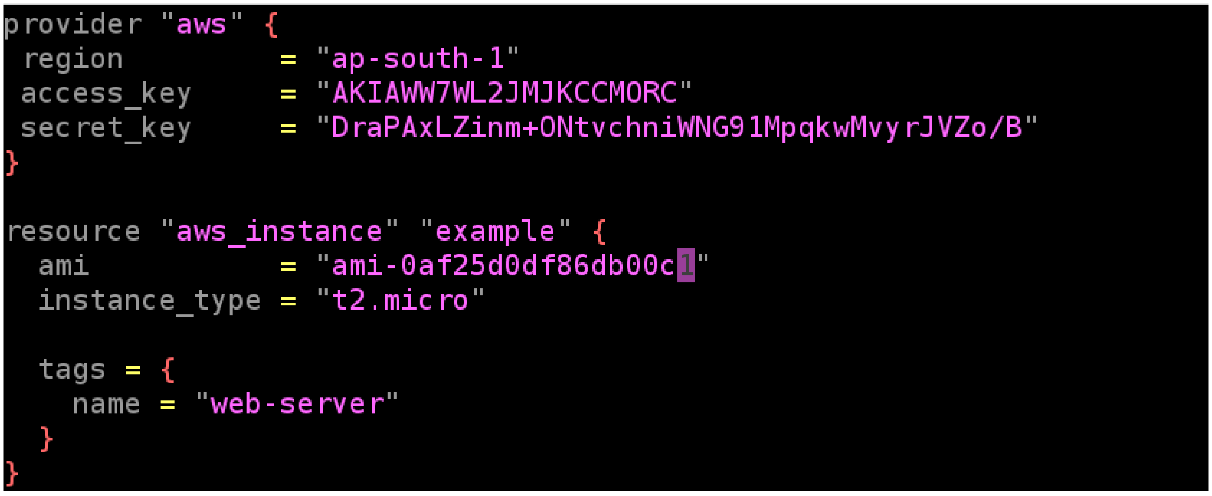
Terraform apply:

You will get an error think logically to get it.

You need to give your ami-id on ap-south-1 and instance will be created there only.

Now terraform.tfstate file will be created which consist all the metadata.

Terraform destroy : kill the instances



```
provider "aws" {
 region          = "ap-south-1"
 access_key      = "AKIAWW7WL2JMJKCCMORC"
 secret_key      = "DraPAxLZinm+ONtvchniWNG91MpqkwMvyrJVZo/B"
}

resource "aws_instance" "example" {
  ami            = "ami-0af25d0df86db00c1"
  instance_type = "t2.micro"

  tags = {
    name = "web-server"
  }
}
```

EC2-ROLE BASED AUTHENTICATION:

By using this without access key and secret access key we can perform the actions

IAM -- > Roles -- > Create -- > AWS Services : EC2 -- > AdministratorAccess -- > Name : EC2-Access -- > Role name : Terraform-role-base -- > Create

Select instance -- > Actions -- > Security -- > Modify IAM Role -- > Select Role -- > Save

Now remove both Access key and Secret Acess key and save the main.tf file.

Terraform plan and terraform apply.

Now the instance will be created.

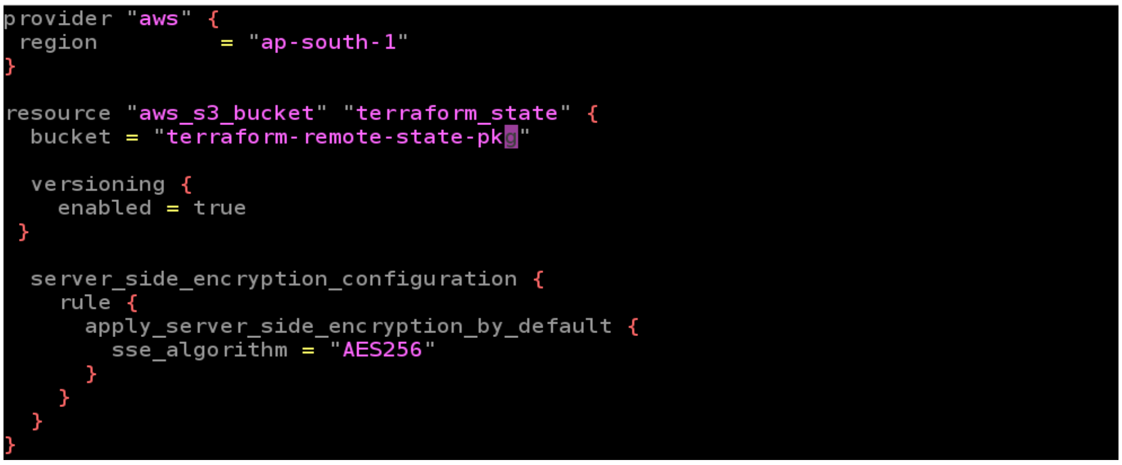
S3 BACKEND SETUP FOR REMOTE STATE FILE

In terraform we have two state files one is local state file and another is remote state file.
We use Local state file when we there is no involvement of other person.

Create a bucket with versioning enable

Initialize the backend with S3 using Terraform.

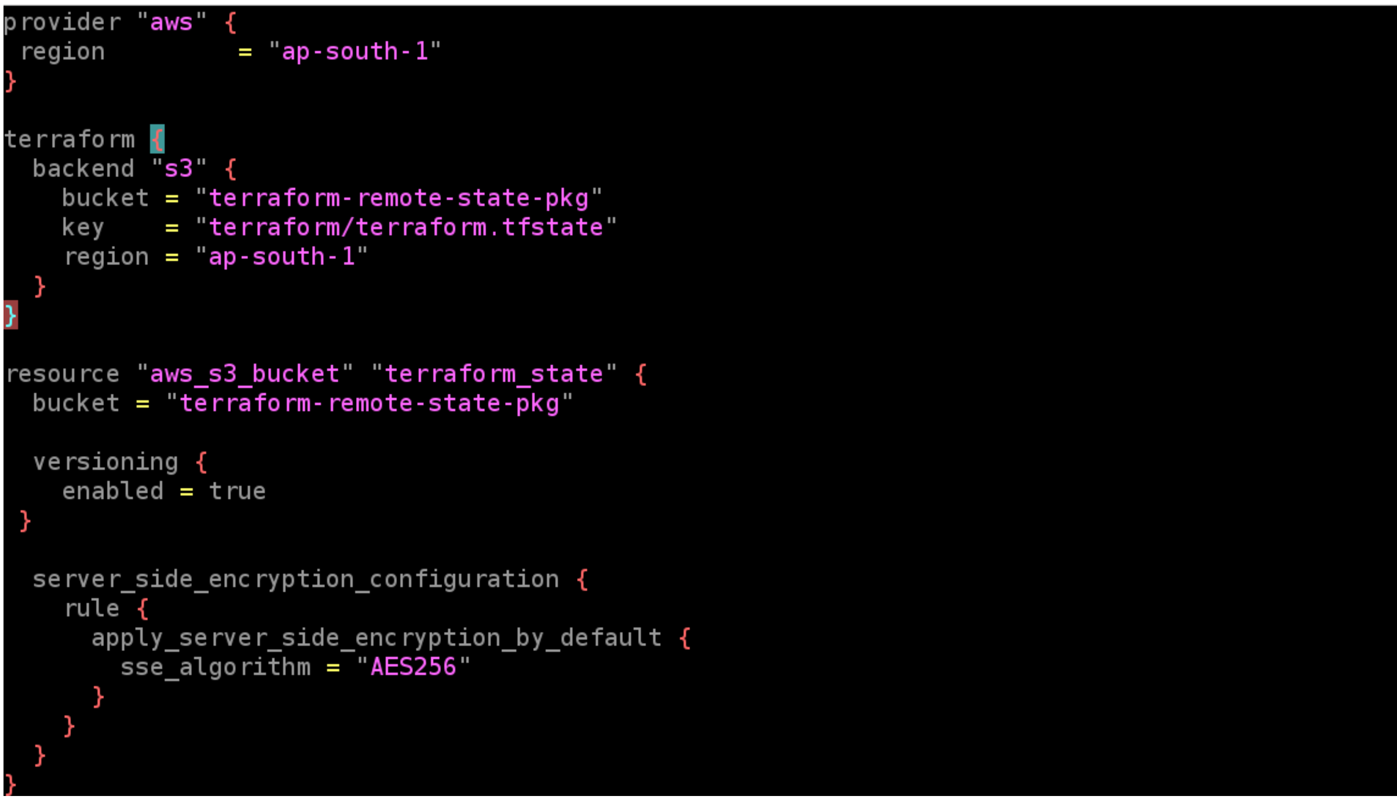
Launch the resources using terraform to validate the remote state file and Versioning.



```
provider "aws" {
 region          = "ap-south-1"
}

resource "aws_s3_bucket" "terraform_state" {
  bucket = "terraform-remote-state-pkg"

  versioning {
    enabled = true
 }

  server_side_encryption_configuration {
    rule {
      apply_server_side_encryption_by_default {
        sse_algorithm = "AES256"
      }
    }
  }
}
```

Terraform plan and terraform apply

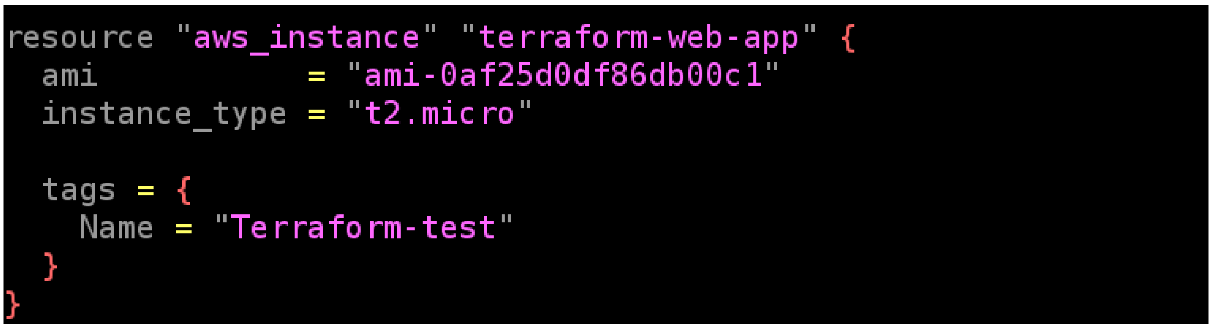
Now the bucket will be created on that region



```
provider "aws" {
 region          = "ap-south-1"
}

terraform {
  backend "s3" {
    bucket = "terraform-remote-state-pkg"
    key    = "terraform/terraform.tfstate"
    region = "ap-south-1"
  }
}

resource "aws_s3_bucket" "terraform_state" {
  bucket = "terraform-remote-state-pkg"

  versioning {
    enabled = true
 }

  server_side_encryption_configuration {
    rule {
      apply_server_side_encryption_by_default {
        sse_algorithm = "AES256"
      }
    }
  }
}
```

Terraform init : It will be successful

Now add EC2 Resource in same code.



```
resource "aws_instance" "terraform-web-app" {
  ami           = "ami-0af25d0df86db00c1"
  instance_type = "t2.micro"

  tags = {
    Name = "Terraform-test"
  }
}
```

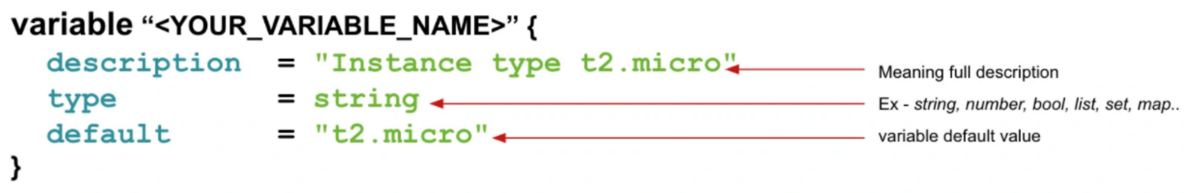
Add this part on the end

Terraform plan and terraform apply.

Now verify the versioning on that bucket you can two versions and new instance will be created.

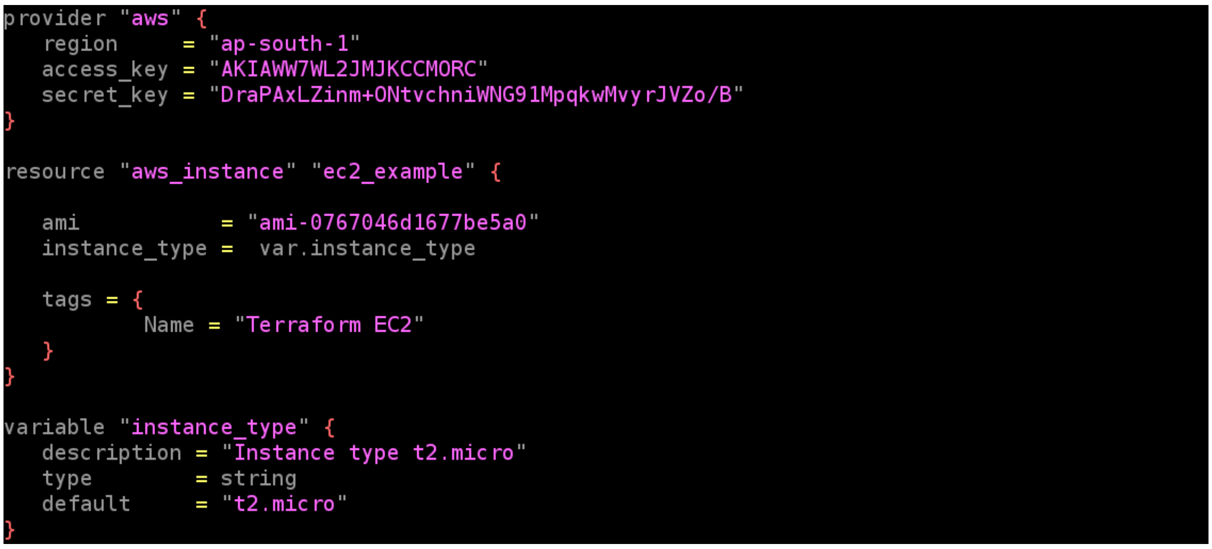
If you give a new tag then you can see the new version (Terraform plan and apply)

TERRAFORM TYPES (VARIABLE TYPE)



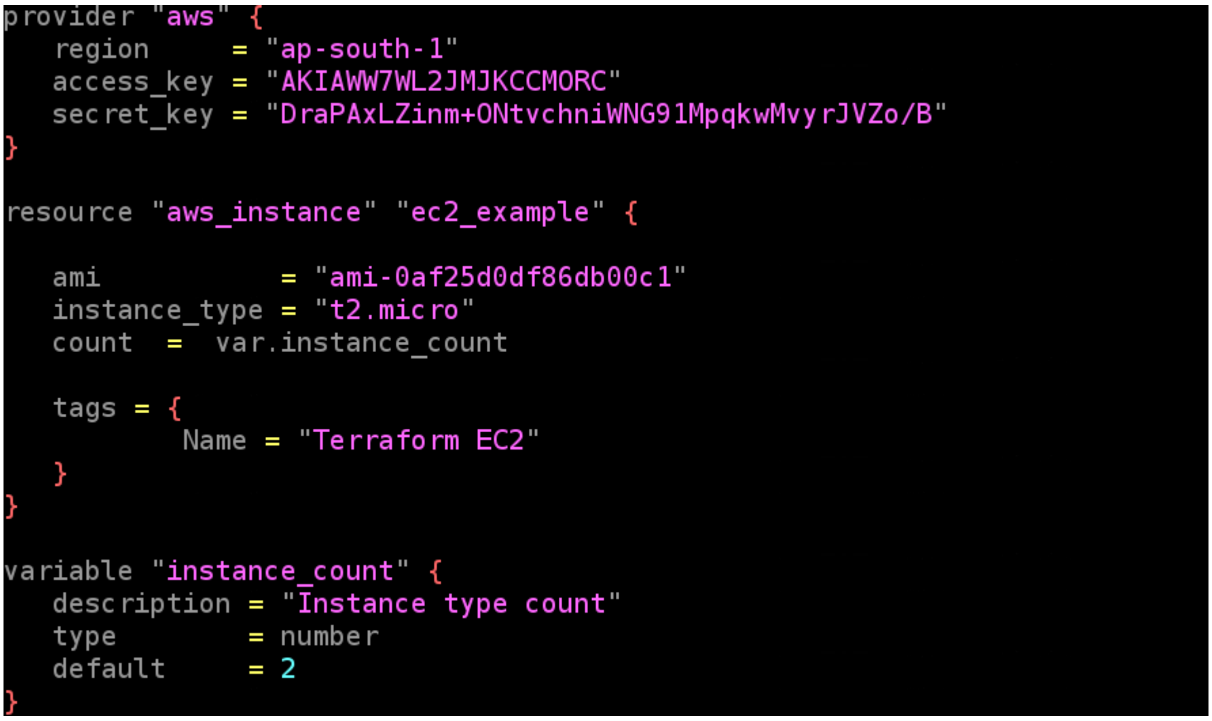
```
variable "<YOUR_VARIABLE_NAME>" {
   description    = "Instance type t2.micro"
   type           = string
   default        = "t2.micro"
}
```

- string: a sequence of Unicode characters representing some text, like "hello". (terraform init, plan, apply, destroy)



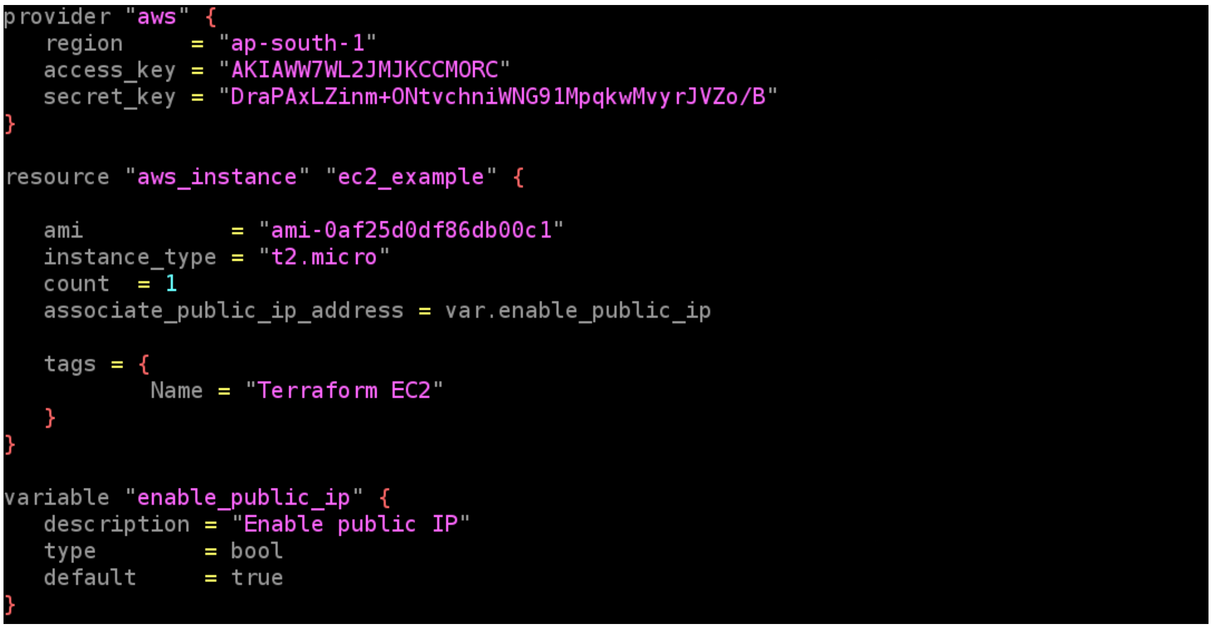
```
provider "aws" {
   region     = "ap-south-1"
   access_key = "AKIAWW7WL2JMJKCCMORC"
   secret_key = "DraPAxLZinm+ONtvchniWNG91MpqkwMvyrJVZo/B"
}

resource "aws_instance" "ec2_example" {

   ami           = "ami-0767046d1677be5a0"
   instance_type =  var.instance_type

   tags = {
           Name = "Terraform EC2"
   }
}

variable "instance_type" {
   description = "Instance type t2.micro"
   type        = string
   default     = "t2.micro"
}
```

- number: a numeric value. The number type can represent both whole numbers like 15 and fractional values like 6.283185.



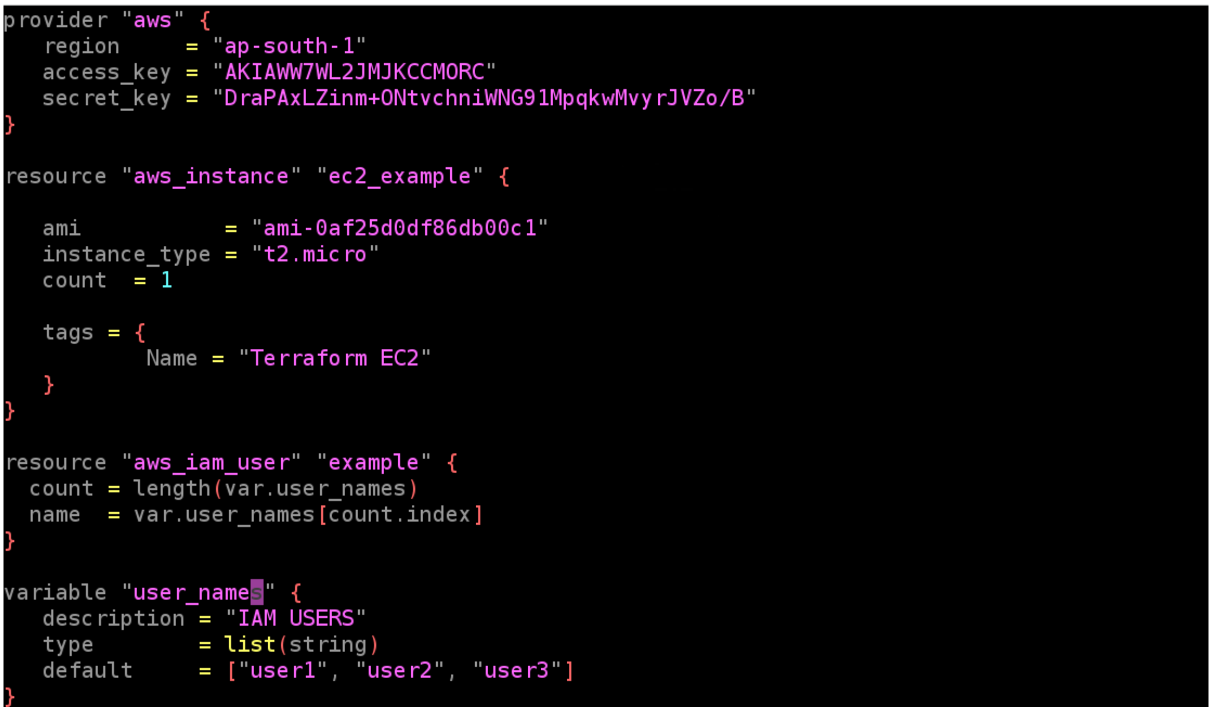
```
provider "aws" {
   region     = "ap-south-1"
   access_key = "AKIAWW7WL2JMJKCCMORC"
   secret_key = "DraPAxLZinm+ONtvchniWNG91MpqkwMvyrJVZo/B"
}

resource "aws_instance" "ec2_example" {

   ami           = "ami-0af25d0df86db00c1"
   instance_type = "t2.micro"
   count   =  var.instance_count

   tags = {
           Name = "Terraform EC2"
   }
}

variable "instance_count" {
   description = "Instance type count"
   type        = number
   default     = 2
}
```

- bool: a boolean value, either true or false. null: a value that represents *absence* or *omission.* If you set an argument of a resource to null, terraform behaves as though you had completely omitted it — it will use the argument's default value if it has one, or raise an error if the argument is mandatory. null is most useful in conditional expressions, so you can dynamically omit an argument if a condition isn't met.



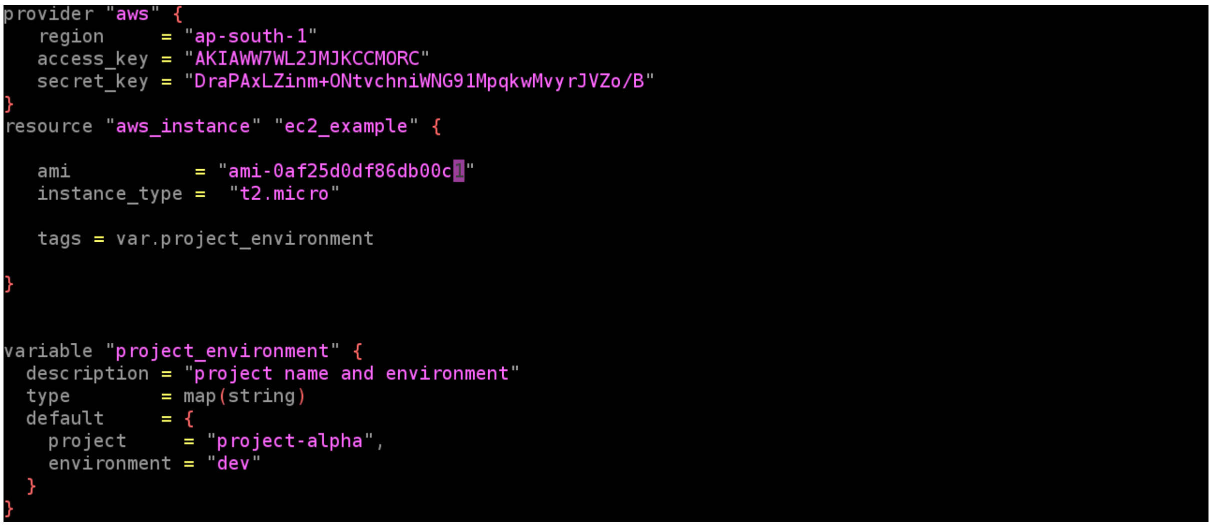
```
provider "aws" {
   region     = "ap-south-1"
   access_key = "AKIAWW7WL2JMJKCCMORC"
   secret_key = "DraPAxLZinm+ONtvchniWNG91MpqkwMvyrJVZo/B"
}

resource "aws_instance" "ec2_example" {

   ami           = "ami-0af25d0df86db00c1"
   instance_type = "t2.micro"
   count  = 1
   associate_public_ip_address = var.enable_public_ip

   tags = {
           Name = "Terraform EC2"
   }
}

variable "enable_public_ip" {
   description = "Enable public IP"
   type        = bool
   default     = true
}
```

- list (or tuple): a sequence of values, like ["user1", "user2", "user3"]. Elements in a list or tuple are identified by consecutive whole numbers, starting with zero.



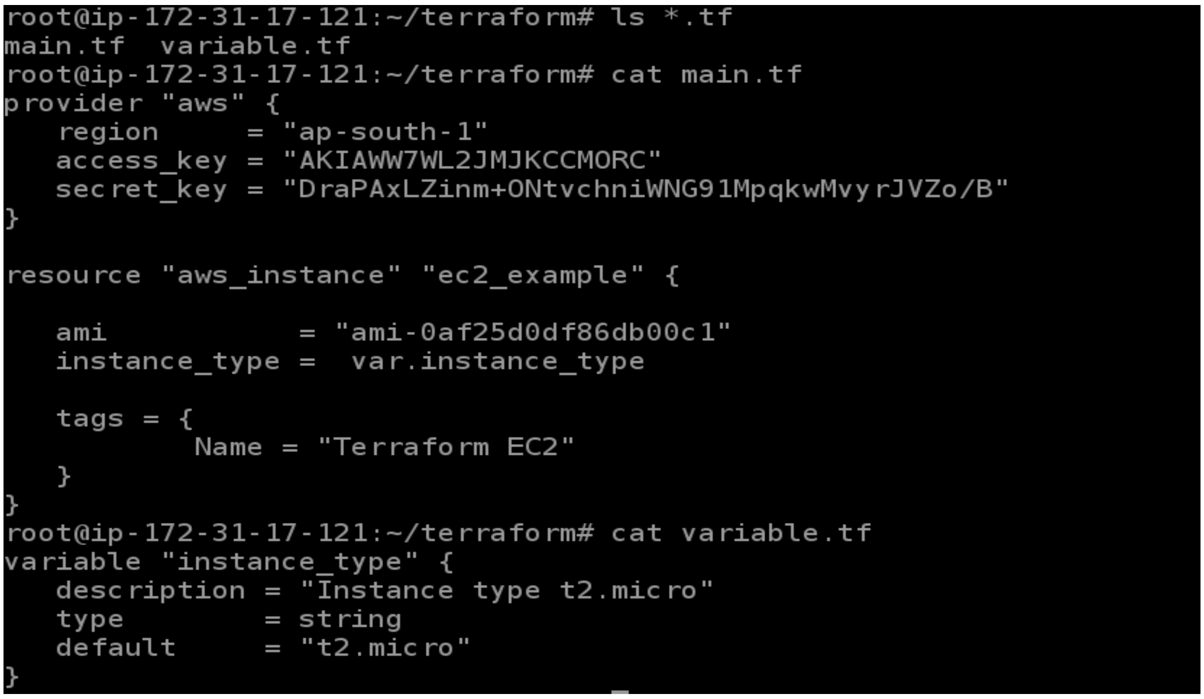
```
provider "aws" {
   region     = "ap-south-1"
   access_key = "AKIAWW7WL2JMJKCCMORC"
   secret_key = "DraPAxLZinm+ONtvchniWNG91MpqkwMvyrJVZo/B"
}

resource "aws_instance" "ec2_example" {

   ami           = "ami-0af25d0df86db00c1"
   instance_type = "t2.micro"
   count   = 1

   tags = {
           Name = "Terraform EC2"
   }
}

resource "aws_iam_user" "example" {
  count = length(var.user_names)
  name  = var.user_names[count.index]
}

variable "user_names" {
   description = "IAM USERS"
   type        = list(string)
   default     = ["user1", "user2", "user3"]
}
```

- map (or object): a group of values identified by named labels, like {project = "project-plan", environment = “dev”}.



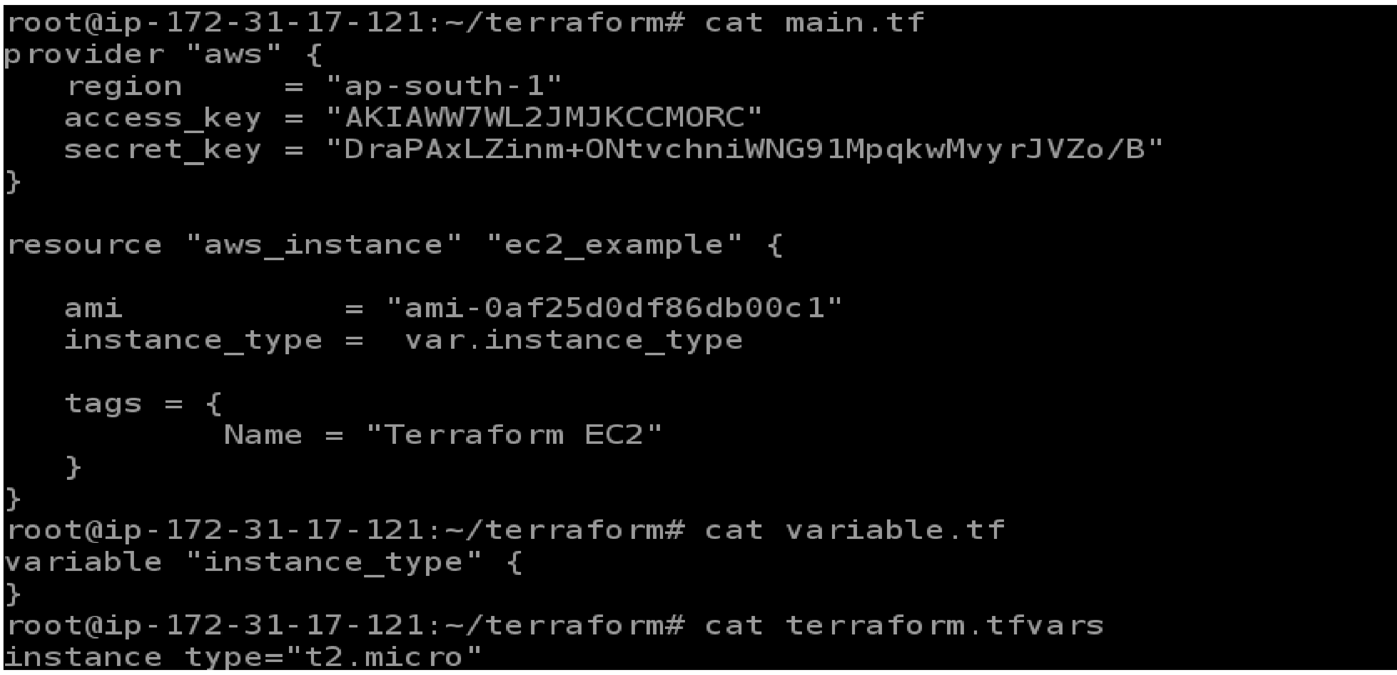
```
provider "aws" {
   region     = "ap-south-1"
   access_key = "AKIAWW7WL2JMJKCCMORC"
   secret_key = "DraPAxLZinm+ONtvchniWNG91MpqkwMvyrJVZo/B"
}
resource "aws_instance" "ec2_example" {

   ami           = "ami-0af25d0df86db00c1"
   instance_type =  "t2.micro"

   tags = var.project_environment

}


variable "project_environment" {
  description = "project name and environment"
  type        = map(string)
  default     = {
    project     = "project-alpha",
    environment = "dev"
  }
}
```

VARIABLE.TF



```
root@ip-172-31-17-121:~/terraform# ls *.tf
main.tf  variable.tf
root@ip-172-31-17-121:~/terraform# cat main.tf
provider "aws" {
   region     = "ap-south-1"
   access_key = "AKIAWW7WL2JMJKCCMORC"
   secret_key = "DraPAxLZinm+ONtvchniWNG91MpqkwMvyrJVZo/B"
}

resource "aws_instance" "ec2_example" {

   ami           = "ami-0af25d0df86db00c1"
   instance_type =  var.instance_type

   tags = {
           Name = "Terraform EC2"
   }
}
root@ip-172-31-17-121:~/terraform# cat variable.tf
variable "instance_type" {
   description = "Instance type t2.micro"
   type        = string
   default     = "t2.micro"
}
```

TERRAFORM.TFVARS

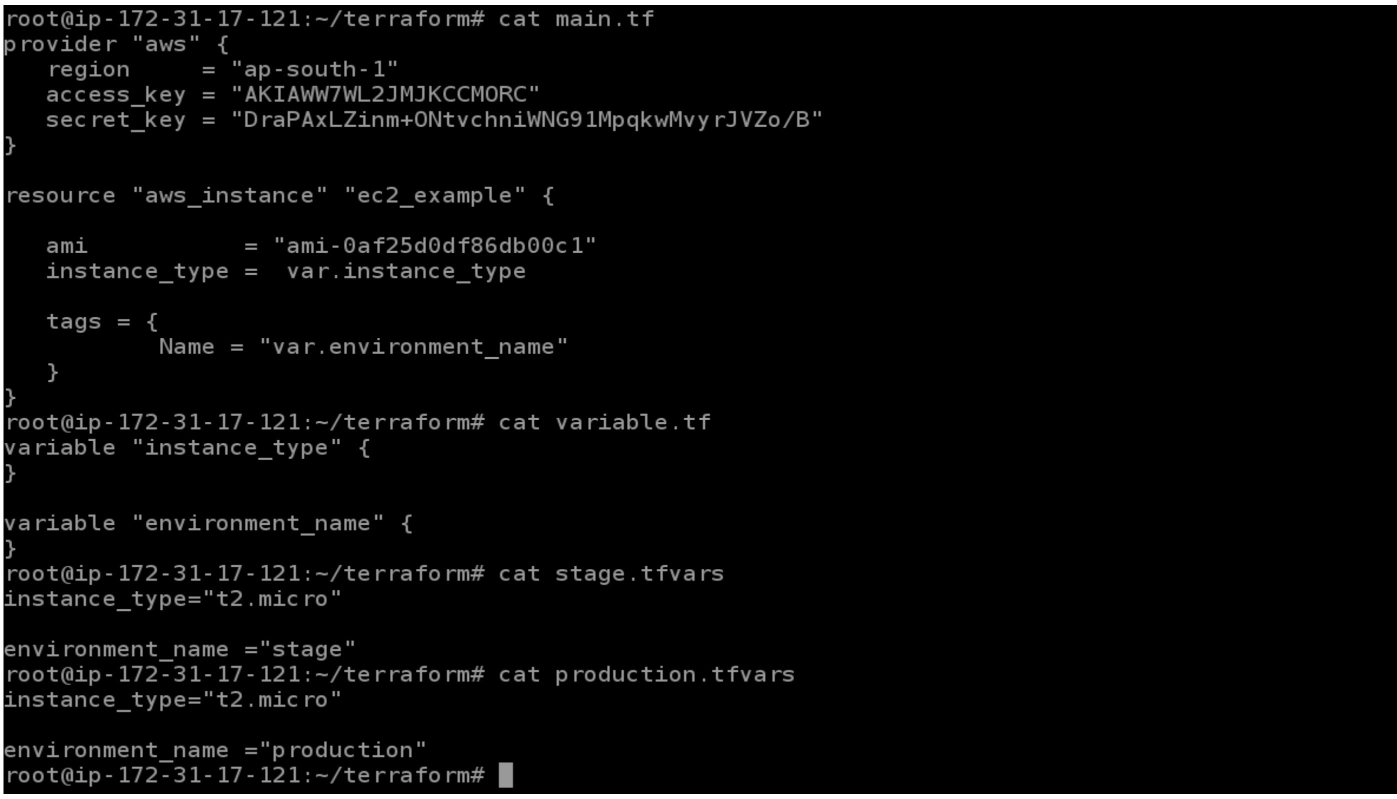


```
root@ip-172-31-17-121:~/terraform# cat main.tf
provider "aws" {
   region     = "ap-south-1"
   access_key = "AKIAWW7WL2JMJKCCMORC"
   secret_key = "DraPAxLZinm+ONtvchniWNG91MpqkwMvyrJVZo/B"
}

resource "aws_instance" "ec2_example" {

   ami           = "ami-0af25d0df86db00c1"
   instance_type =  var.instance_type

   tags = {
           Name = "Terraform EC2"
   }
}
root@ip-172-31-17-121:~/terraform# cat variable.tf
variable "instance_type" {
}
root@ip-172-31-17-121:~/terraform# cat terraform.tfvars
instance_type="t2.micro"
```

MULTIPE TFVAR FILES

There can be situation where you need create multiple tfvars files based on the environment like stage, production.
So in such scenario you can create one tfvars file for each environment -

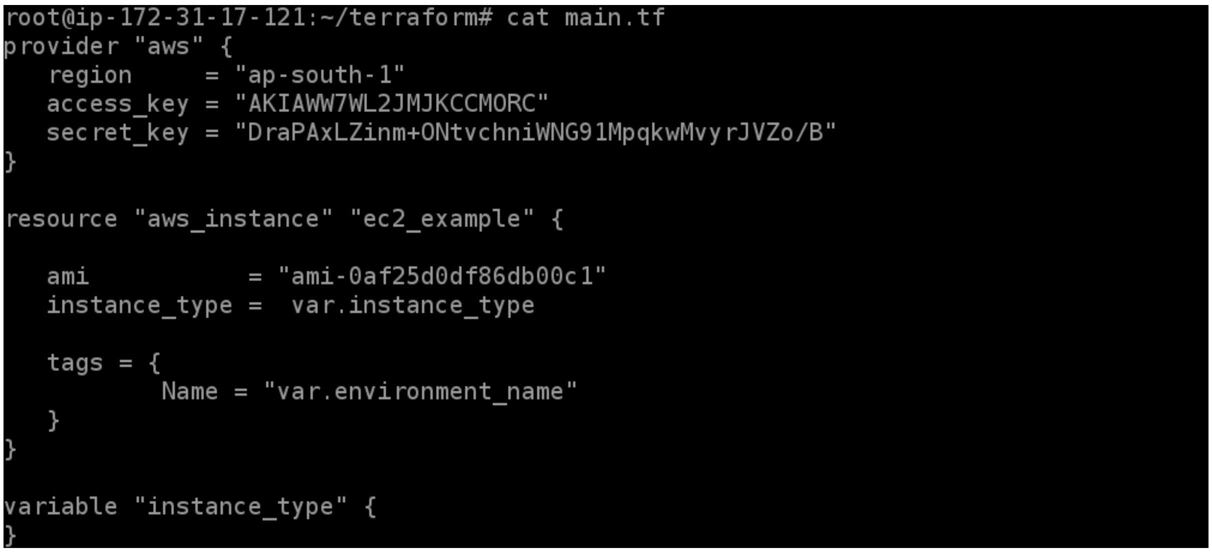
1. stage.tfvars
2. production.tfvars



```
root@ip-172-31-17-121:~/terraform# cat main.tf
provider "aws" {
   region     = "ap-south-1"
   access_key = "AKIAWW7WL2JMJKCCMORC"
   secret_key = "DraPAxLZinm+ONtvchniWNG91MpqkwMvyrJVZo/B"
}

resource "aws_instance" "ec2_example" {

   ami           = "ami-0af25d0df86db00c1"
   instance_type =  var.instance_type

   tags = {
           Name = "var.environment_name"
   }
}
root@ip-172-31-17-121:~/terraform# cat variable.tf
variable "instance_type" {
}

variable "environment_name" {
}
root@ip-172-31-17-121:~/terraform# cat stage.tfvars
instance_type="t2.micro"

environment_name ="stage"
root@ip-172-31-17-121:~/terraform# cat production.tfvars
instance_type="t2.micro"

environment_name ="production"
root@ip-172-31-17-121:~/terraform#
```

```
terraform plan -var-file="stage.tfvars"
terraform apply -var-file="stage.tfvars"
terraform destroy -var-file="stage.tfvars"
```

TERRAFROM COMMANDLINE VARIABLE



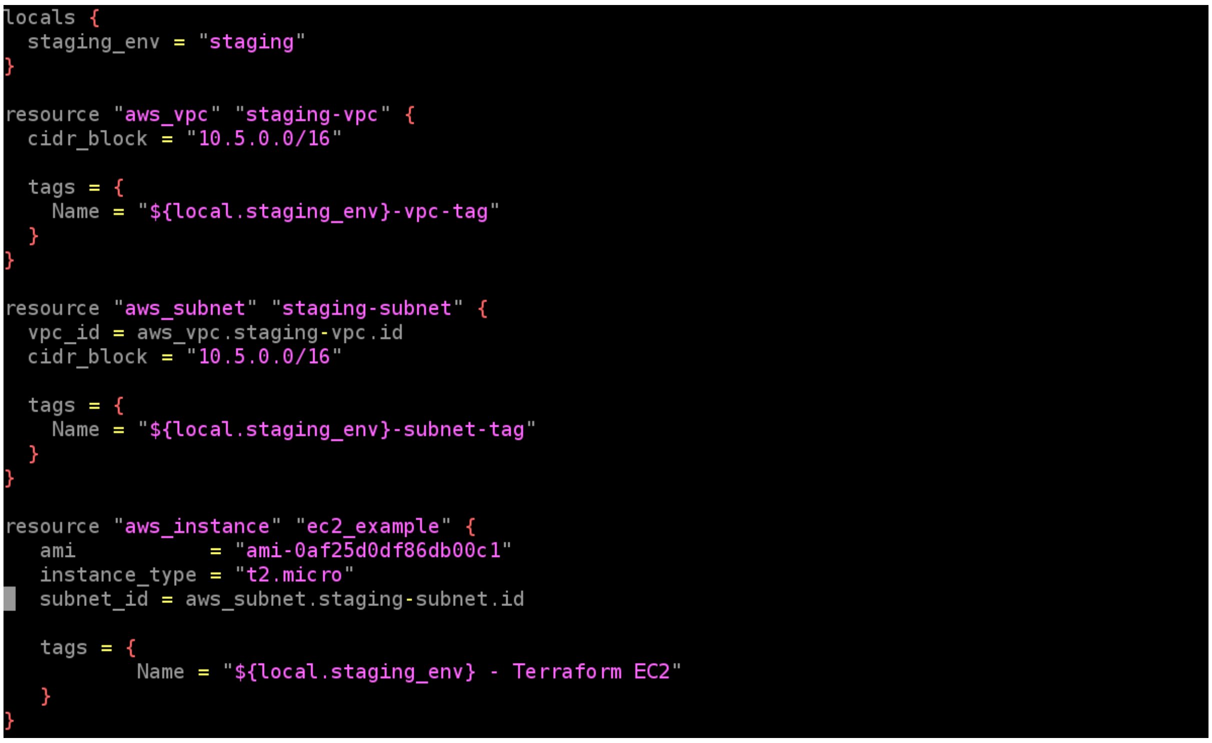
```
root@ip-172-31-17-121:~/terraform# cat main.tf
provider "aws" {
   region     = "ap-south-1"
   access_key = "AKIAWW7WL2JMJKCCMORC"
   secret_key = "DraPAxLZinm+ONtvchniWNG91MpqkwMvyrJVZo/B"
}

resource "aws_instance" "ec2_example" {

   ami           = "ami-0af25d0df86db00c1"
   instance_type =  var.instance_type

   tags = {
           Name = "var.environment_name"
   }
}

variable "instance_type" {
}
```

```
terraform plan -var="instance_type=t2.micro"
terraform apply -var="instance_type=t2.micro"
terraform destroy -var="instance_type=t2.micro"
```

TERRAFORM LOCALS

Terraform locals are quite similar to terraform variables but Terraform locals do not change their value. On the other hand, if you talk about Terraform input variables then it is dependent on user

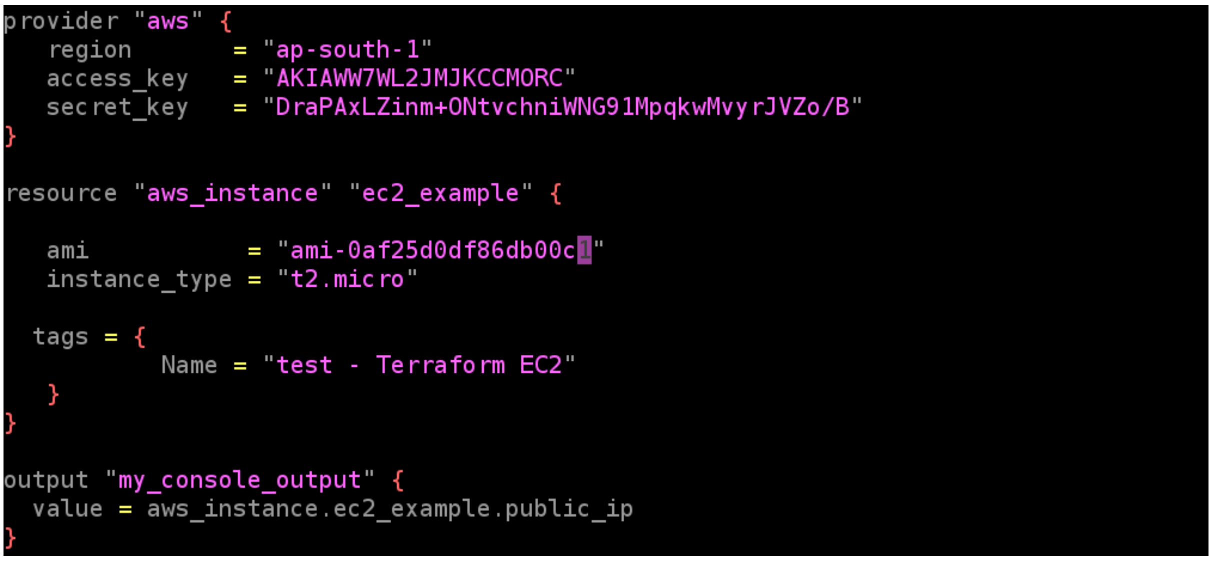
input and it can change its value. So if you have a very large Terraform file where you need to use the same values or expressions multiple times then Terraform local can be useful for you.
NOTE: Give the Entire Provide block as usually.



```
locals {
  staging_env = "staging"
}

resource "aws_vpc" "staging-vpc" {
  cidr_block = "10.5.0.0/16"

  tags = {
    Name = "${local.staging_env}-vpc-tag"
  }
}

resource "aws_subnet" "staging-subnet" {
  vpc_id = aws_vpc.staging-vpc.id
  cidr_block = "10.5.0.0/16"

  tags = {
    Name = "${local.staging_env}-subnet-tag"
  }
}

resource "aws_instance" "ec2_example" {
   ami              = "ami-0af25d0df86db00c1"
   instance_type = "t2.micro"
   subnet_id = aws_subnet.staging-subnet.id

   tags = {
           Name = "${local.staging_env} - Terraform EC2"
   }
}
```

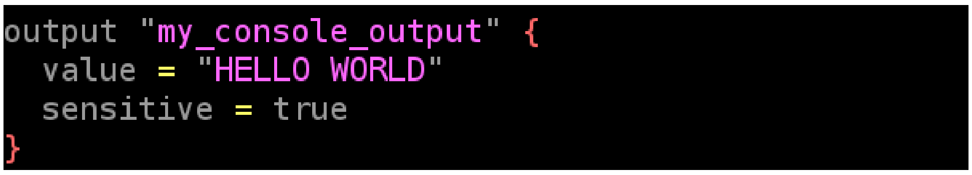
TERRAFORM OUTPUT VALUES

Terraform output values will be really useful when you want to debug your terraform code. Terraform output values can help you to print the attributes reference(arn, instance_state, outpost_arn, public_ip, public_dns etc) on your console.



```
provider "aws" {
    region        = "ap-south-1"
    access_key    = "AKIAWW7WL2JMJKCCMORC"
    secret_key    = "DraPAxLZinm+ONtvchniWNG91MpqkwMvyrJVZo/B"
}

resource "aws_instance" "ec2_example" {

    ami            = "ami-0af25d0df86db00c1"
    instance_type = "t2.micro"

   tags = {
           Name = "test - Terraform EC2"
    }
}

output "my_console_output" {
  value = aws_instance.ec2_example.public_ip
}
```

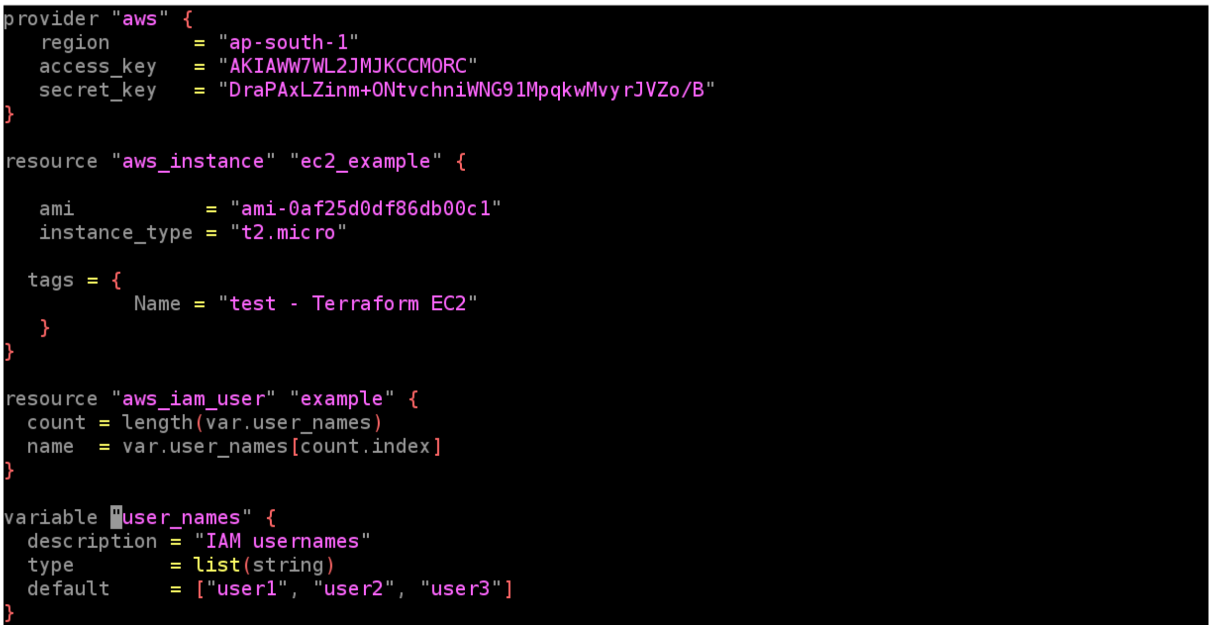
Now if you want to hide the sensitive info (like IP) use the key called sensitive.



```
output "my_console_output" {
  value = "HELLO WORLD"
  sensitive = true
}
```

LOOPS WITH COUNT

As the name suggests we need to use count but to use the count first we need to declare collections inside our terraform file.



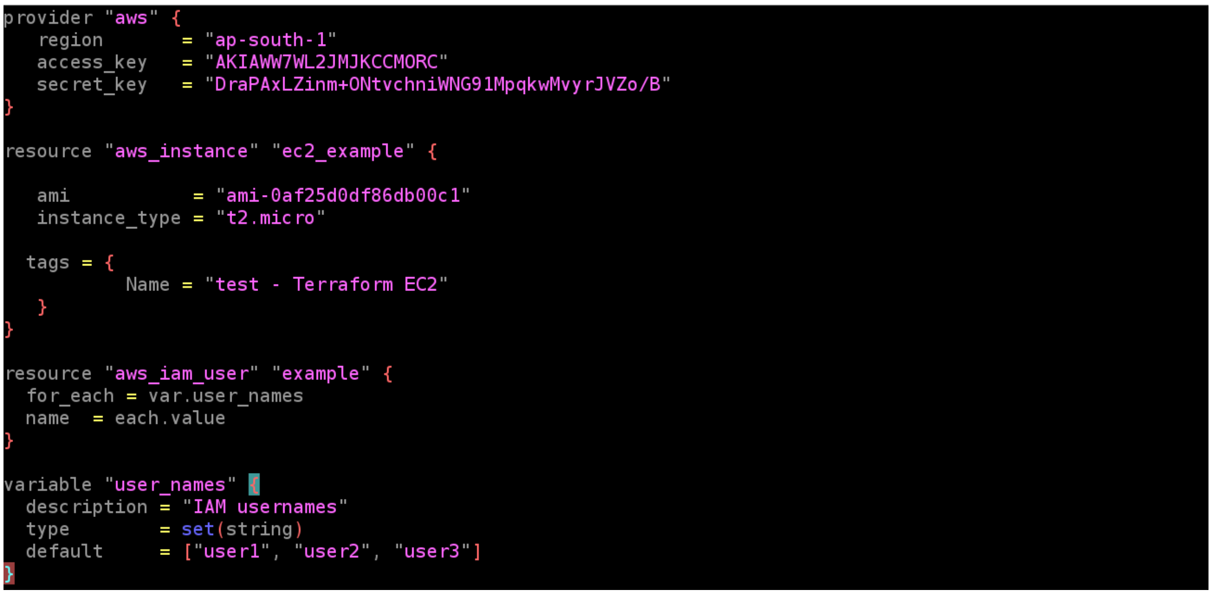
```
provider "aws" {
   region        = "ap-south-1"
   access_key    = "AKIAWW7WL2JMJKCCMORC"
   secret_key    = "DraPAxLZinm+ONtvchniWNG91MpqkwMvyrJVZo/B"
}

resource "aws_instance" "ec2_example" {

   ami           = "ami-0af25d0df86db00c1"
   instance_type = "t2.micro"

  tags = {
           Name = "test - Terraform EC2"
   }
}

resource "aws_iam_user" "example" {
  count = length(var.user_names)
  name  = var.user_names[count.index]
}

variable "user_names" {
  description = "IAM usernames"
  type        = list(string)
  default     = ["user1", "user2", "user3"]
}
```

LOOPS WITH FOR_EACH

The for_each is a little special in terraforming and you can not use it on any collection variable.

 *Note : - **It can only be used on** set(string) **or** map(string).*

The reason why for_each does not work on list(string) is because a list can contain duplicate values but if you are using set(string) or map(string) then it does not support duplicate values.

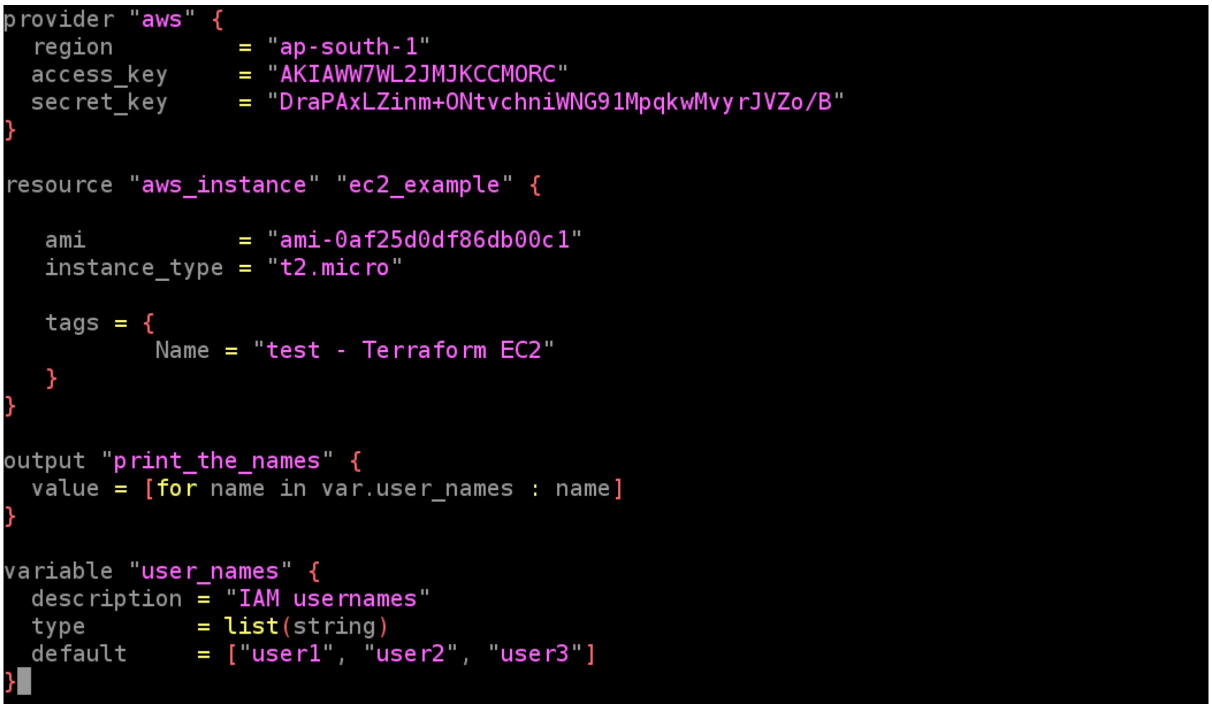


```
provider "aws" {
   region       = "ap-south-1"
   access_key   = "AKIAWW7WL2JMJKCCMORC"
   secret_key   = "DraPAxLZinm+ONtvchniWNG91MpqkwMvyrJVZo/B"
}

resource "aws_instance" "ec2_example" {

   ami           = "ami-0af25d0df86db00c1"
   instance_type = "t2.micro"

  tags = {
          Name = "test - Terraform EC2"
   }
}

resource "aws_iam_user" "example" {
  for_each = var.user_names
  name  = each.value
}

variable "user_names" {
  description = "IAM usernames"
  type        = set(string)
  default     = ["user1", "user2", "user3"]
}
```

FOR LOOP

The for loop is pretty simple and if you have used any programming language before then I guess you will be pretty much familiar with the for loop.
Only the difference you will notice over here is the syntax in Terraform.

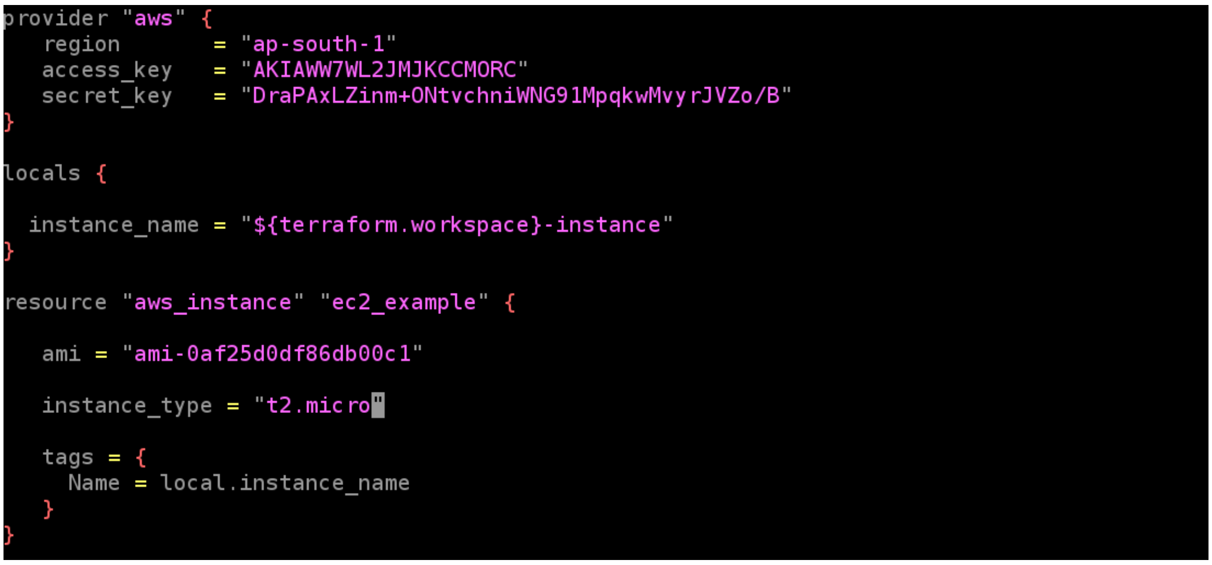
I am going to take the same example by declaring a list(string) and adding three users to it - user1, user2, user3



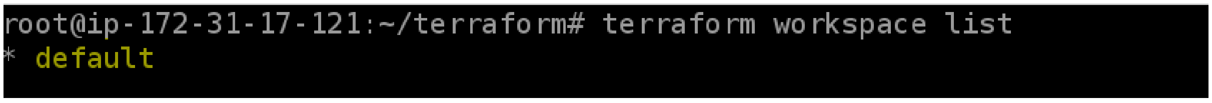
```
provider "aws" {
  region        = "ap-south-1"
  access_key    = "AKIAWW7WL2JMJKCCMORC"
  secret_key    = "DraPAxLZinm+ONtvchniWNG91MpqkwMvyrJVZo/B"
}

resource "aws_instance" "ec2_example" {

   ami           = "ami-0af25d0df86db00c1"
   instance_type = "t2.micro"

   tags = {
           Name = "test - Terraform EC2"
   }
}

output "print_the_names" {
  value = [for name in var.user_names : name]
}

variable "user_names" {
  description = "IAM usernames"
  type        = list(string)
  default     = ["user1", "user2", "user3"]
}
```

TERRAFORM WORKSPACE

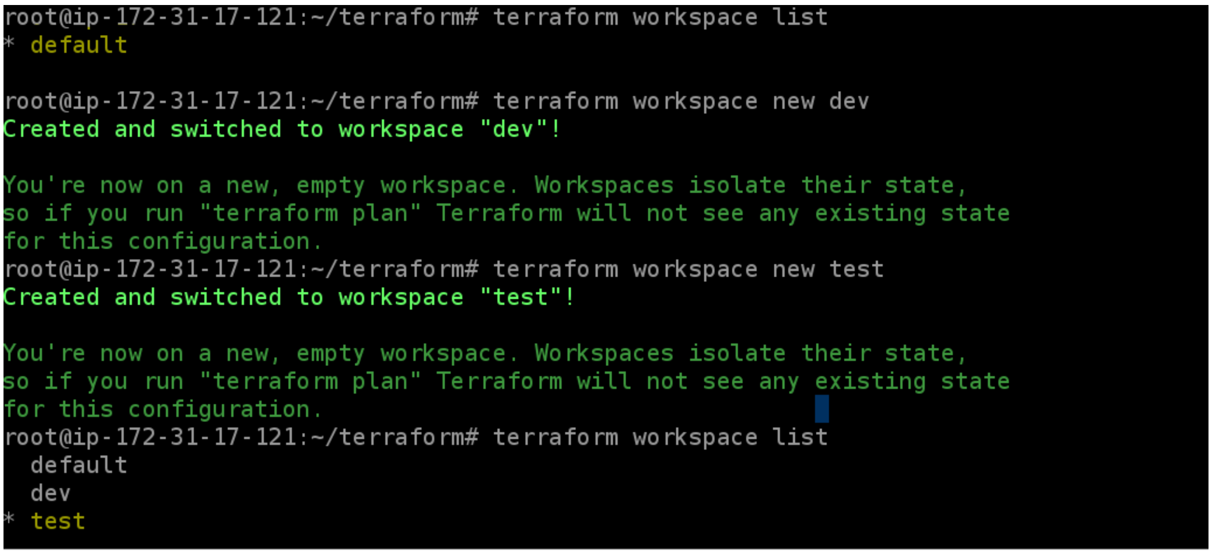
To create a new workspace	: terraform workspace new workspace_name
To list the workspace	: terraform workspace list
To show current workspace	: terraform workspace show
To switch workspace	: terraform workspace select workspace_name



```
provider "aws" {
   region       = "ap-south-1"
   access_key   = "AKIAWW7WL2JMJKCCMORC"
   secret_key   = "DraPAxLZinm+ONtvchniWNG91MpqkwMvyrJVZo/B"
}

locals {

  instance_name = "${terraform.workspace}-instance"
}

resource "aws_instance" "ec2_example" {

   ami = "ami-0af25d0df86db00c1"

   instance_type = "t2.micro"

   tags = {
     Name = local.instance_name
   }
}
```



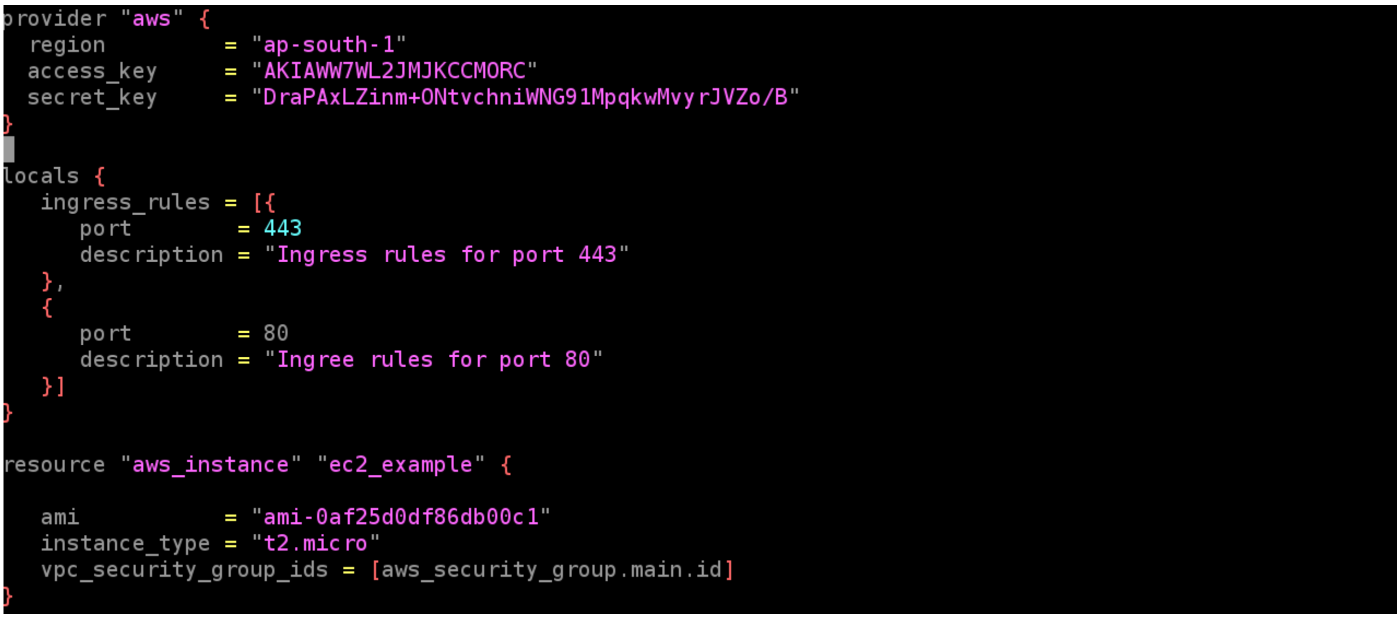
```
root@ip-172-31-17-121:~/terraform# terraform workspace list
* default
```



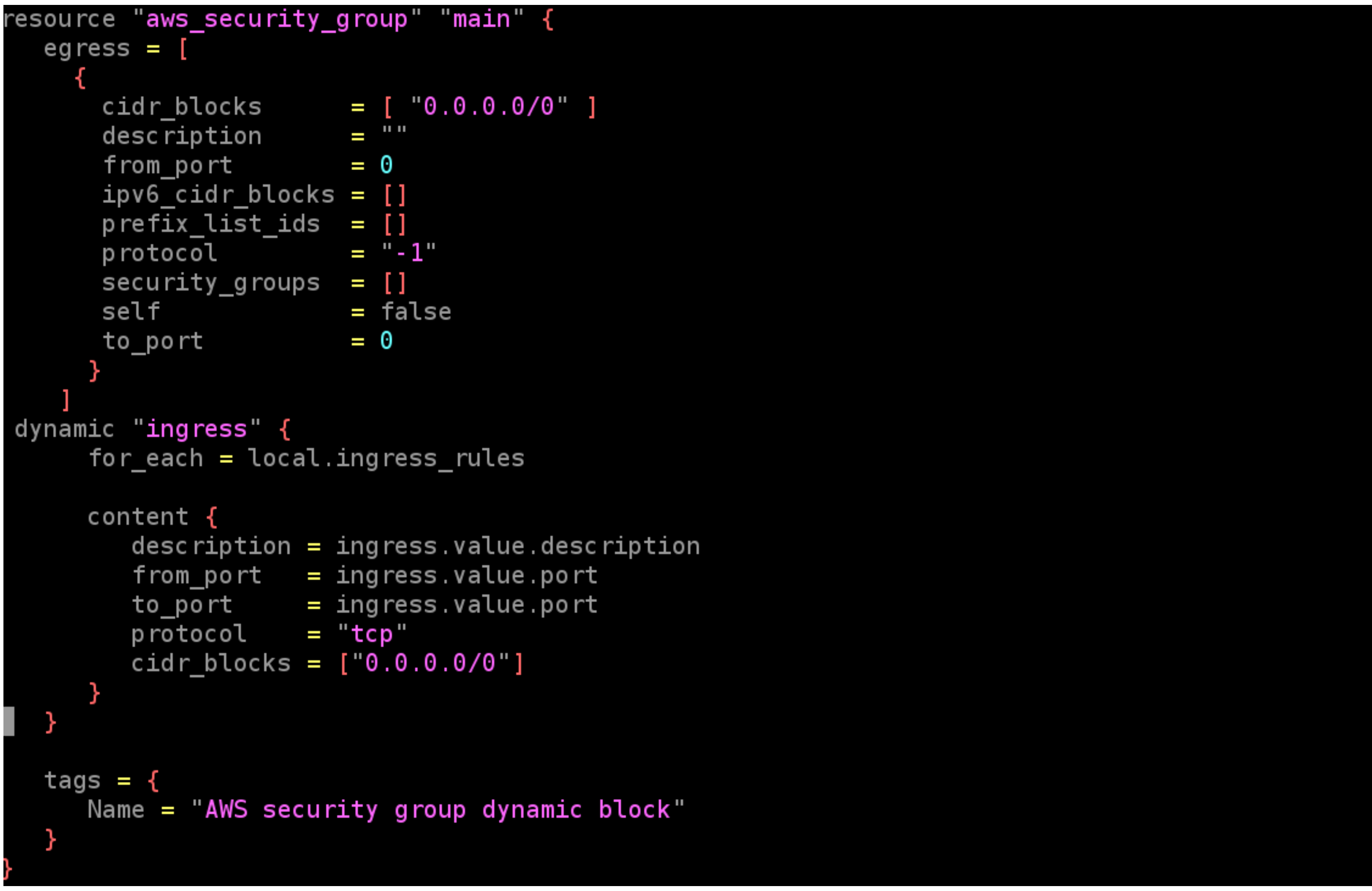
```
root@ip-172-31-17-121:~/terraform# terraform workspace list
* default

root@ip-172-31-17-121:~/terraform# terraform workspace new dev
Created and switched to workspace "dev"!

You're now on a new, empty workspace. Workspaces isolate their state,
so if you run "terraform plan" Terraform will not see any existing state
for this configuration.
root@ip-172-31-17-121:~/terraform# terraform workspace new test
Created and switched to workspace "test"!

You're now on a new, empty workspace. Workspaces isolate their state,
so if you run "terraform plan" Terraform will not see any existing state
for this configuration.
root@ip-172-31-17-121:~/terraform# terraform workspace list
  default
  dev
* test
```

DYNAMIC BLOCK

Reduces the line of the code and makes the code reusable for us.



```
provider "aws" {
  region         = "ap-south-1"
  access_key     = "AKIAWW7WL2JMJKCCMORC"
  secret_key     = "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"
}

locals {
   ingress_rules = [{
      port        = 443
      description = "Ingress rules for port 443"
   },
   {
      port        = 80
      description = "Ingree rules for port 80"
   }]
}

resource "aws_instance" "ec2_example" {

   ami           = "ami-0af25d0df86db00c1"
   instance_type = "t2.micro"
   vpc_security_group_ids = [aws_security_group.main.id]
}
```



```
resource "aws_security_group" "main" {
   egress = [
     {
       cidr_blocks      = [ "0.0.0.0/0" ]
       description      = ""
       from_port        = 0
       ipv6_cidr_blocks = []
       prefix_list_ids  = []
       protocol         = "-1"
       security_groups  = []
       self             = false
       to_port          = 0
     }
    ]
 dynamic "ingress" {
      for_each = local.ingress_rules

      content {
         description = ingress.value.description
         from_port   = ingress.value.port
         to_port     = ingress.value.port
         protocol    = "tcp"
         cidr_blocks = ["0.0.0.0/0"]
      }
   }

   tags = {
      Name = "AWS security group dynamic block"
   }
}
```

CODE:

```
locals {
  ingress_rules = [{
port= 443
description = "Ingress rules for port 443"
  },
  {
```

```
port= 80
description = "Ingree rules for port 80"
  }]
}

resource "aws_instance" "ec2_example" {
   ami = "ami-0c02fb55956c7d316"
   instance_type = "t2.micro"
   vpc_security_group_ids = [aws_security_group.main.id]
   tags = {
Name = "Terraform EC2"
   }
}

resource "aws_security_group" "main" {

egress = [
{
cidr_blocks      = [ "0.0.0.0/0" ]
description      = "*"
from_port= 0
ipv6_cidr_blocks = []
prefix_list_ids  = []
protocol= "-1"
security_groups  = []
self= false
to_port= 0
} ]
```

EBS:

```
resource "aws_ebs_volume" "example" {
  availability_zone = "us-west-2a"
  size              = 40

  tags = {
    Name = "HelloWorld"
  }
}
```

EFS:

```
resource "aws_efs_file_system" "foo" {
  creation_token = "my-product"

  tags = {
    Name = "MyProduct"
  }
}
```

S3:

```
resource "aws_s3_bucket" "example" {
  bucket = "my-tf-example-bucket"
}
```

```
resource "aws_s3_bucket_acl" "example_bucket_acl" {
  bucket = aws_s3_bucket.example.id
  acl    = "private"
}
```

RDS:

```
resource "aws_rds_cluster" "default" {
  cluster_identifier      = "aurora-cluster-demo"
  engine                  = "aurora-mysql"
  engine_version          = "5.7.mysql_aurora.2.03.2"
  availability_zones      = ["us-west-2a", "us-west-2b", "us-west-2c"]
  database_name           = "mydb"
  master_username         = "foo"
  master_password         = "bar"
  backup_retention_period = 5
  preferred_backup_window = "07:00-09:00"
}
```

ALIAS & PROVIDER:

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

resource "aws_instance" "example" {
 ami           = "ami-0e6329e222e662a52"
 instance_type = "t2.micro"
}

resource "aws_instance" "example1" {
 ami           = "ami-0c4e4b4eb2e11d1d4"
 instance_type = "t2.medium"
 provider      = "aws.east"
}
```

LOCAL:

```
resource "local_file" "abc" {
filename = “/root/abc.txt”
}
```

```
resource "local_filw" "abc" {
filename = "/root/abc.txt"
content = "hai all"
}
```

```
resource "local_file" "abc" {
filename = "/root/abc.txt"
content = "hai all"
file_permission = "777"
}
```

MULTIPLE PROVIDERS:
Lets work with multiple providers now

RANDOM PROVIDER:
The "random" provider allows the use of randomness within Terraform configurations. This is a logical provider, which means that it works entirely within Terraform's logic, and doesn't interact with any other services.
it provides resources that generate random values during their creation and then hold those values steady until the inputs are changed.

```
terraform plan -lock=false
```

Version Constraints:
If you want to work on specific version then u can use this one:
Provider – > hashicorp – > local – > version

```
Terraform {
required_providers {
local = {
source = "hashicorp/local"
version = "1.3.0"
}
}
}
```

```
resource "local_file" "abc" {
filename = "/root/abc.txt"
content = "hai all"
file_permission = "777"
}
```

```
terraform init -upgrade
```

Terraform plan & apply
Check the local version now by using terraform version

Greater than :
Replace 1.3.0 with > 1.3.0 : It will be upgraded to more than version 1.3.0
Less than :
Replace 1.3.0 with < 1.3.0 : It will be upgraded to less than version 1.3.0

In between :
Give “ > 1.2.0, <2.0.0, !=1.4.0”

Specific version/Incremental version:
“~>1.2” : It will work on the incremental values of last version (ie 1.4)