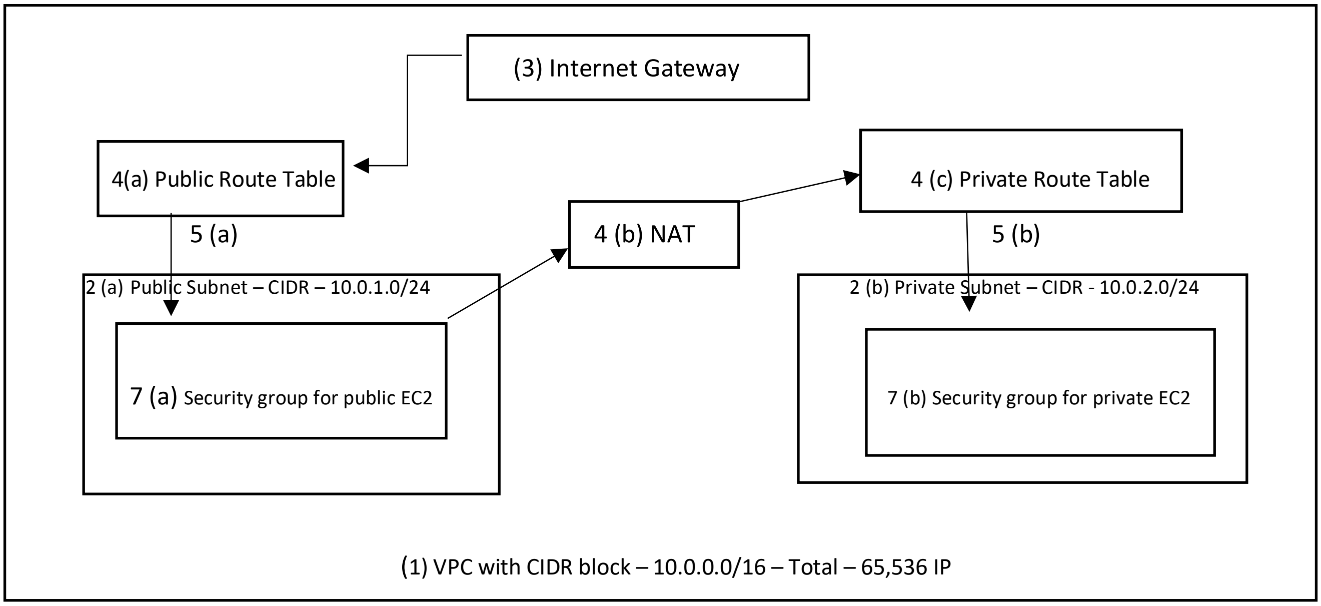
# Automate AWS Network Infrastructure Virtual Private Cloud Deployment using Terraform DevOps Tool

**TERRAFORM -** It is an Infrastructure as a code tool, used primarily by DevOps teams to automate infrastructure tasks.

To understand the concept of Virtual private cloud before writing the terraform code for VPC creation.


(3) Internet Gateway
4(a) Public Route Table
4 (c) Private Route Table
5 (a)
4 (b) NAT
5 (b)
2 (a) Public Subnet – CIDR – 10.0.1.0/24
2 (b) Private Subnet – CIDR - 10.0.2.0/24
7 (a) Security group for public EC2
7 (b) Security group for private EC2
(1) VPC with CIDR block – 10.0.0.0/16 – Total – 65,536 IP

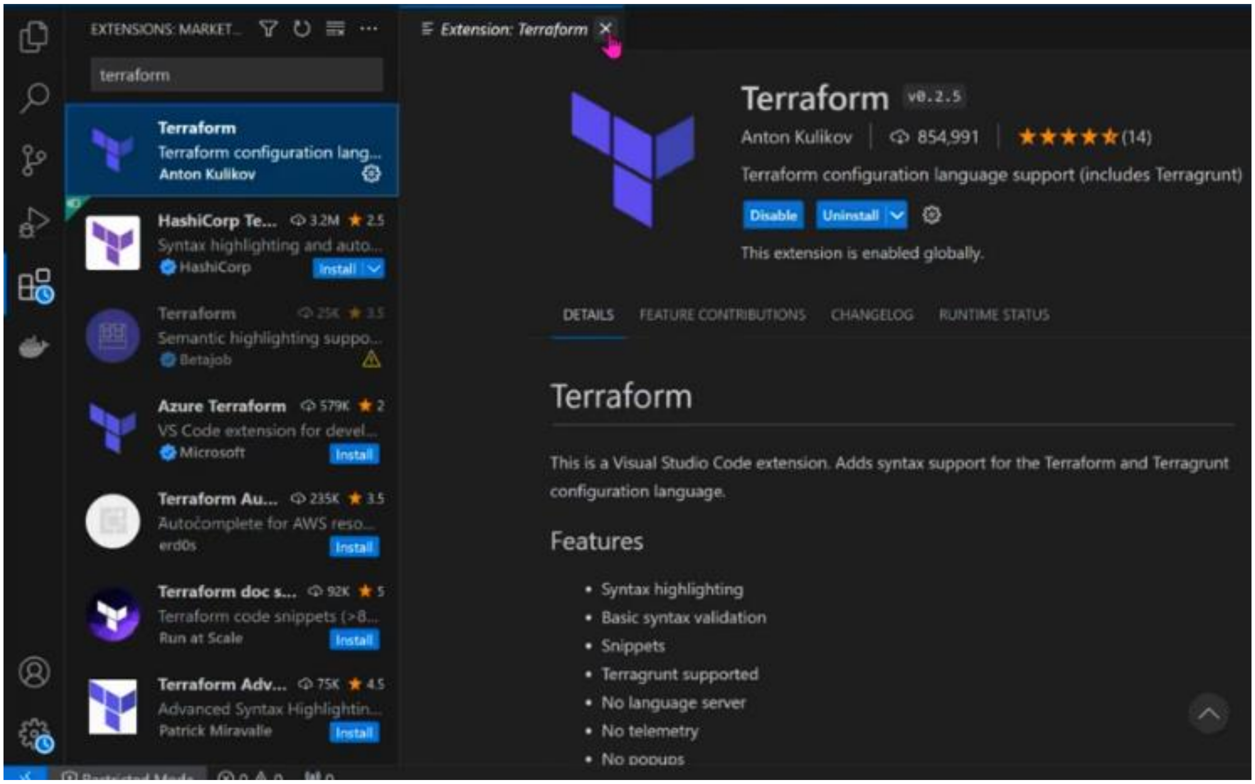
## Virtual Private Cloud

Now we have to write the terraform hashicorp programming language to automate AWS Network Infrastructure Virtual Private Cloud Deployment. Refer the terraform hashicorp programming language from terraform registry official website.

We have to write the code in **visual studio code.**

**The Main purpose of using Visual studio code: Syntax highlighting, Basic syntax validation, Terragrunt supported.**

Before writing the code in visual studio code, we have to complete terraform setup in visual studio code with file name extension as ".tf "


EXTENSIONS: MARKET...
terraform
Terraform
Terraform configuration lang...
Anton Kulikov
HashiCorp Te... 3.2M ★ 2.5
Syntax highlighting and auto...
HashiCorp
Install
Terraform 25K ★ 3.5
Semantic highlighting suppo...
Betajob
Azure Terraform 579K ★ 2
VS Code extension for devel...
Microsoft
Install
Terraform Au... 235K ★ 3.5
Autocomplete for AWS reso...
erd0s
Install
Terraform doc s... 92K ★ 5
Terraform code snippets (>8...
Run at Scale
Install
Terraform Adv... 75K ★ 4.5
Advanced Syntax Highlightin...
Patrick Miravalle
Install
Extension: Terraform
Terraform v0.2.5
Anton Kulikov | 854,991 | ★★★★★(14)
Terraform configuration language support (includes Terragrunt)
Disable
Uninstall
This extension is enabled globally.
DETAILS
FEATURE CONTRIBUTIONS
CHANGELOG
RUNTIME STATUS

Terraform

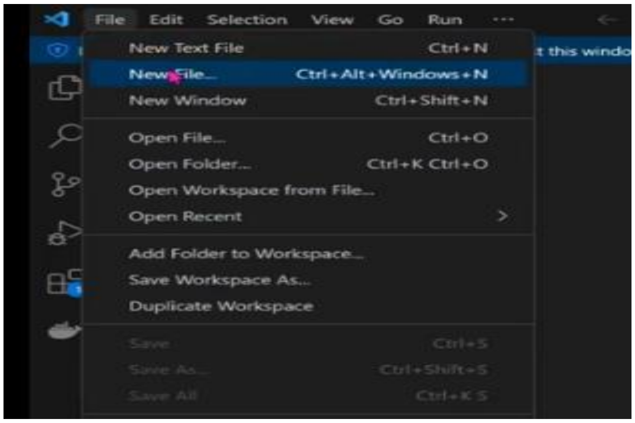
This is a Visual Studio Code extension. Adds syntax support for the Terraform and Terragrunt configuration language.

Features

- Syntax highlighting
- Basic syntax validation
- Snippets
- Terragrunt supported
- No language server
- No telemetry
- No popups

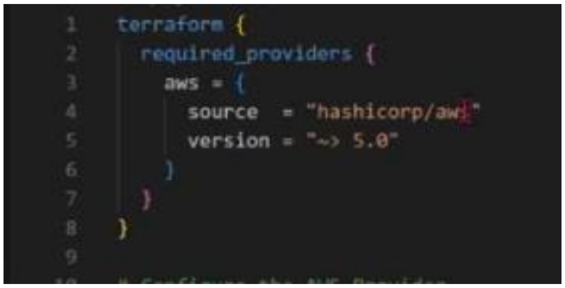
Create the new file with file extension as ".tf"

tf -> **T**erra**f**orm



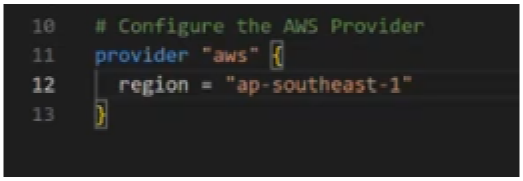
Now we will start to write the terraform hashicorp programming code to automate AWS Network Infrastructure Virtual Private Cloud Deployment.

Step: 1 -> we will setup terraform cloud provider plugins such as Aws, Azure, GCP cloud etc. Now we will select terraform AWS cloud plugins



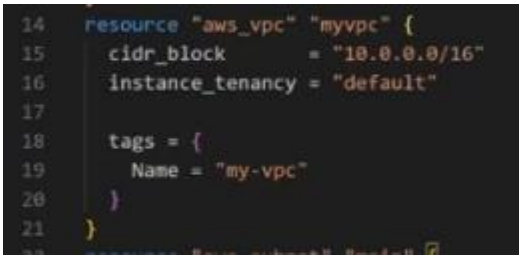
```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
  }
}

```

Step: 2 -> # Configure the AWS Provider with region = “ap-southeast-1”



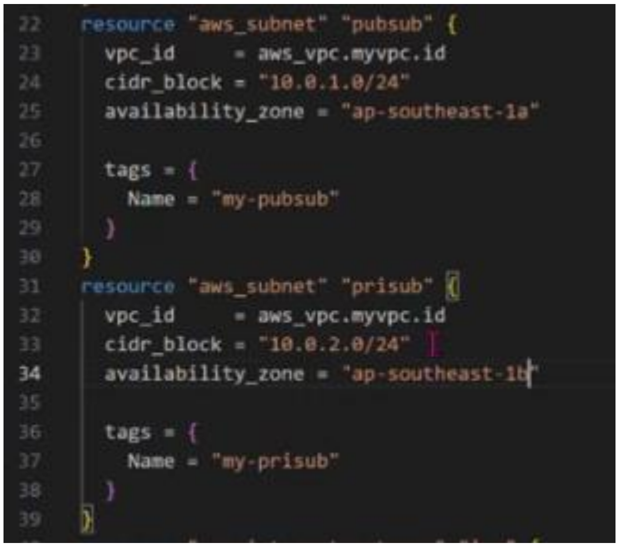
```
# Configure the AWS Provider
provider "aws" {
  region = "ap-southeast-1"
}
```

Step: 3 -> # Create VPC with CIDR = “10.0.0.0/16” (10.0.0.0 to 10.0.255.255 = Total -> 65,536 Ip Address)



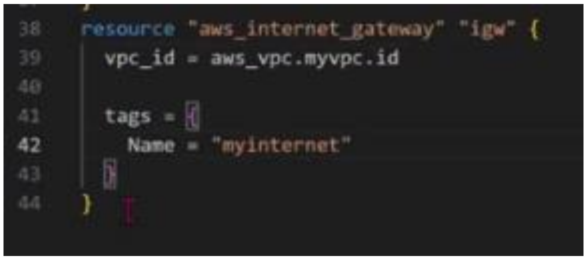
```
resource "aws_vpc" "myvpc" {
  cidr_block       = "10.0.0.0/16"
  instance_tenancy = "default"

  tags = {
    Name = "my-vpc"
  }
}
```

Step: 4 -> To create two subnets. One is for public subnet and another one is private subnets with two different availability zones and CIDR_block.



```
resource "aws_subnet" "pubsub" {
  vpc_id     = aws_vpc.myvpc.id
  cidr_block = "10.0.1.0/24"
  availability_zone = "ap-southeast-1a"

  tags = {
    Name = "my-pubsub"
  }
}
resource "aws_subnet" "prisub" {
  vpc_id     = aws_vpc.myvpc.id
  cidr_block = "10.0.2.0/24"
  availability_zone = "ap-southeast-1b"

  tags = {
    Name = "my-prisub"
  }
}
```

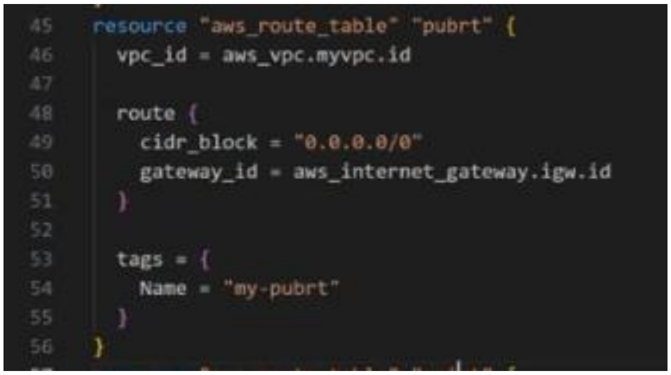
Step: 5 -> To create internet gateway and attach it to VPC and Public route table.



```
resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.myvpc.id

  tags = {
    Name = "myinternet"
  }
}
```

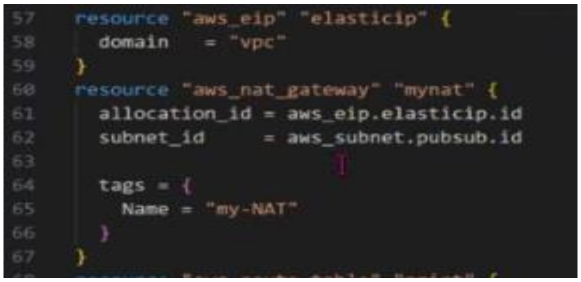
Step: 6 -> To create two route table. One is for public route table and another one is for private route table.

(a): To create public route table.



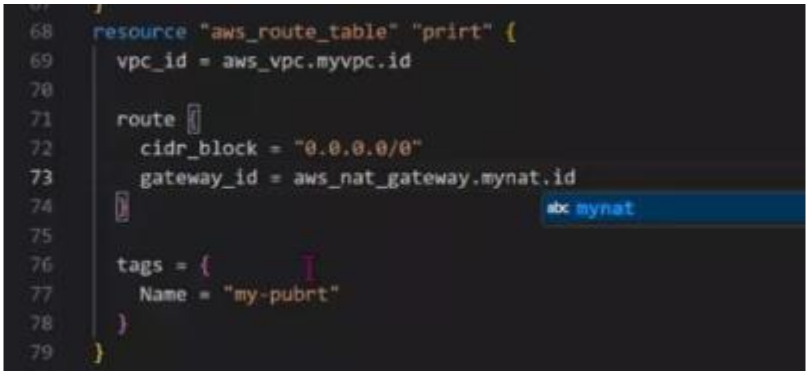
```
resource "aws_route_table" "pubrt" {
  vpc_id = aws_vpc.myvpc.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.igw.id
  }

  tags = {
    Name = "my-pubrt"
  }
}
```

(b): We have to create Elastic IP address and Network Address Translation (NAT) before private route table creation. After that we have attach Elastic IP address and Public subnet to NAT.



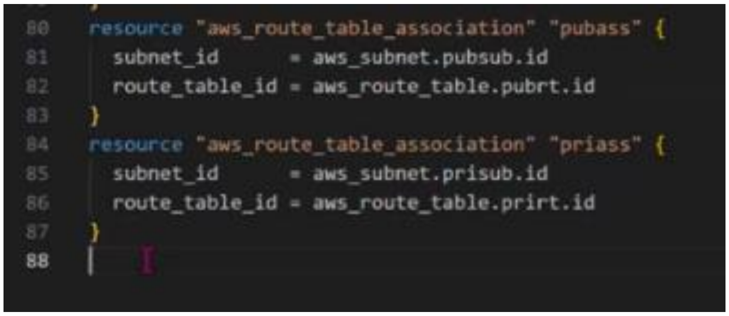
```
resource "aws_eip" "elasticip" {
  domain    = "vpc"
}
resource "aws_nat_gateway" "mynat" {
  allocation_id = aws_eip.elasticip.id
  subnet_id     = aws_subnet.pubsub.id

  tags = {
    Name = "my-NAT"
  }
}
```

(c): To create Private Route Table and attach it to NAT to give internet to private network.



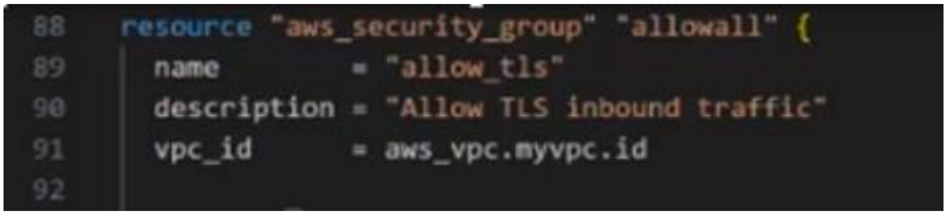
```
resource "aws_route_table" "prirt" {
  vpc_id = aws_vpc.myvpc.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_nat_gateway.mynat.id
  }

  tags = {
    Name = "my-pubrt"
  }
}
```

Step: 7 -> Route Table association -> To attach public and private subnets to the route table

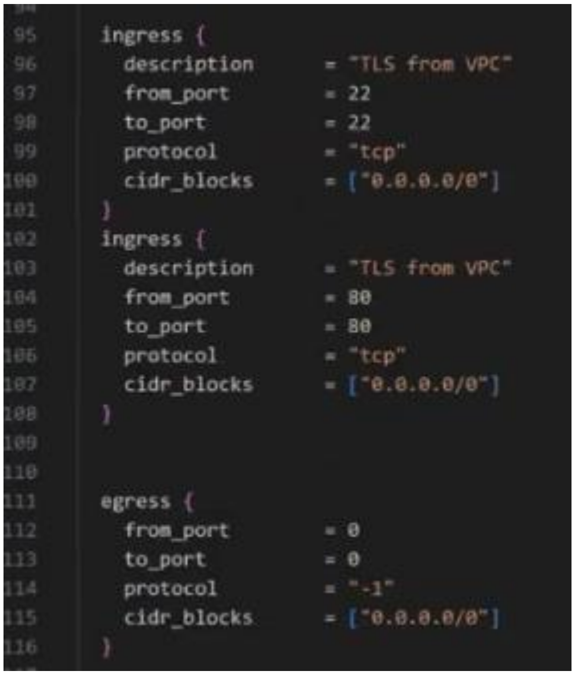


```
resource "aws_route_table_association" "pubass" {
  subnet_id      = aws_subnet.pubsub.id
  route_table_id = aws_route_table.pubrt.id
}
resource "aws_route_table_association" "priass" {
  subnet_id      = aws_subnet.prisub.id
  route_table_id = aws_route_table.prirt.id
}
```

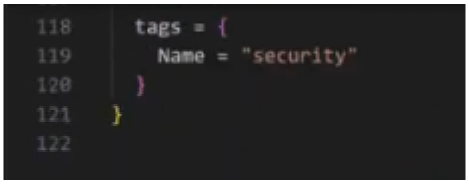
Step: 8 -> To create Security Group and attach it to VPC.



```
resource "aws_security_group" "allowall" {
  name        = "allow_tls"
  description = "Allow TLS inbound traffic"
  vpc_id      = aws_vpc.myvpc.id
```

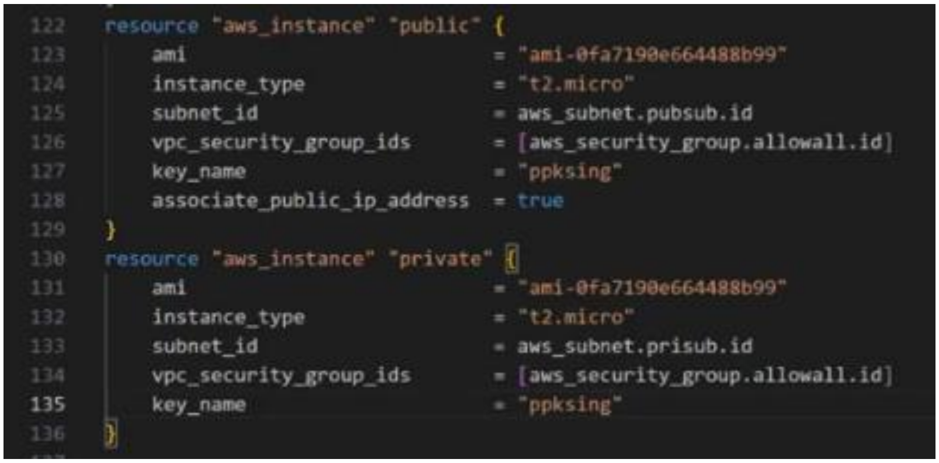


```
  ingress {
    description      = "TLS from VPC"
    from_port        = 22
    to_port          = 22
    protocol         = "tcp"
    cidr_blocks      = ["0.0.0.0/0"]
  }
  ingress {
    description      = "TLS from VPC"
    from_port        = 80
    to_port          = 80
    protocol         = "tcp"
    cidr_blocks      = ["0.0.0.0/0"]
  }


  egress {
    from_port        = 0
    to_port          = 0
    protocol         = "-1"
    cidr_blocks      = ["0.0.0.0/0"]
  }
```



```
  tags = {
    Name = "security"
  }
}
```

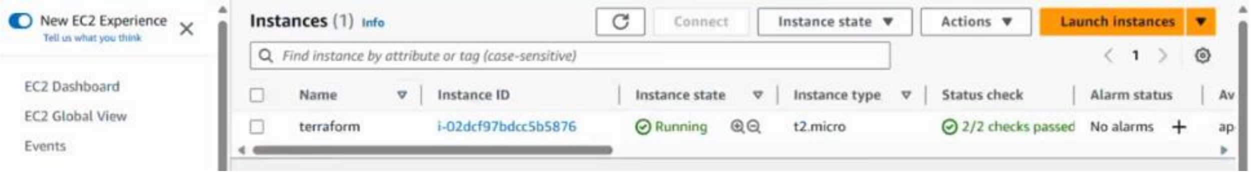
Step: 9 -> To create 2 virtual Machines. One is for public and another one is for private.



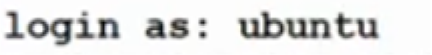
```
resource "aws_instance" "public" {
    ami                         = "ami-0fa7190e664488b99"
    instance_type               = "t2.micro"
    subnet_id                   = aws_subnet.pubsub.id
    vpc_security_group_ids      = [aws_security_group.allowall.id]
    key_name                    = "ppksing"
    associate_public_ip_address = true
}
resource "aws_instance" "private" {
    ami                         = "ami-0fa7190e664488b99"
    instance_type               = "t2.micro"
    subnet_id                   = aws_subnet.prisub.id
    vpc_security_group_ids      = [aws_security_group.allowall.id]
    key_name                    = "ppksing"
}
```

Step: 10 -> To create virtual server (EC2 Instance) in AWS for terraform setup in linux.

To login as Ubuntu linux by using public key

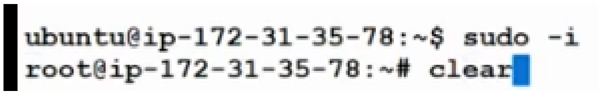


| Name | Instance ID | Instance state | Instance type | Status check | Alarm status | Av |
|---|---|---|---|---|---|---|
| terraform | i-02dcf97bdcc5b5876 | Running | t2.micro | 2/2 checks passed | No alarms | ap |



```
login as: ubuntu
```

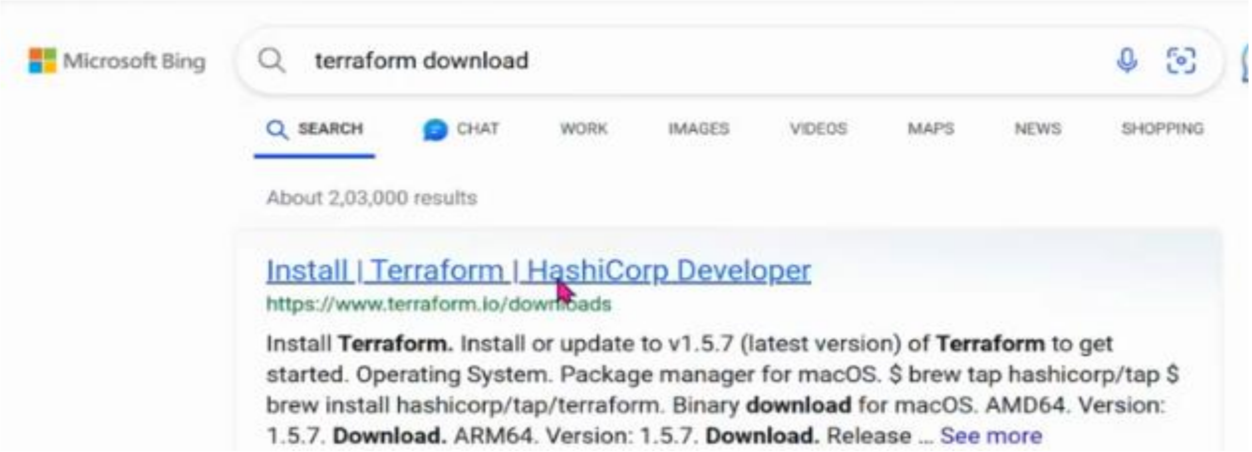
To convert root user / admin user using command as “sudo –i”



```
ubuntu@ip-172-31-35-78:~$ sudo -i
root@ip-172-31-35-78:~# clear
```

Now we have to install terraform in Ubuntu linux after logged in Ubuntu linux.

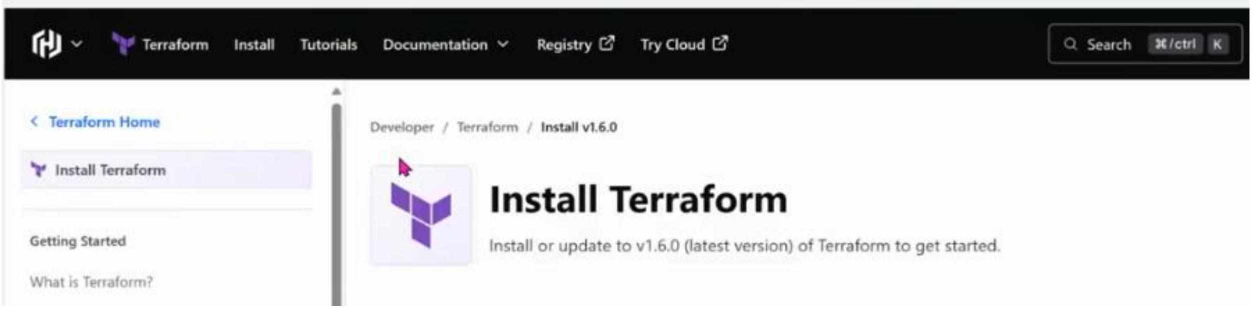
Go to Terraform Official website



About 2,03,000 results

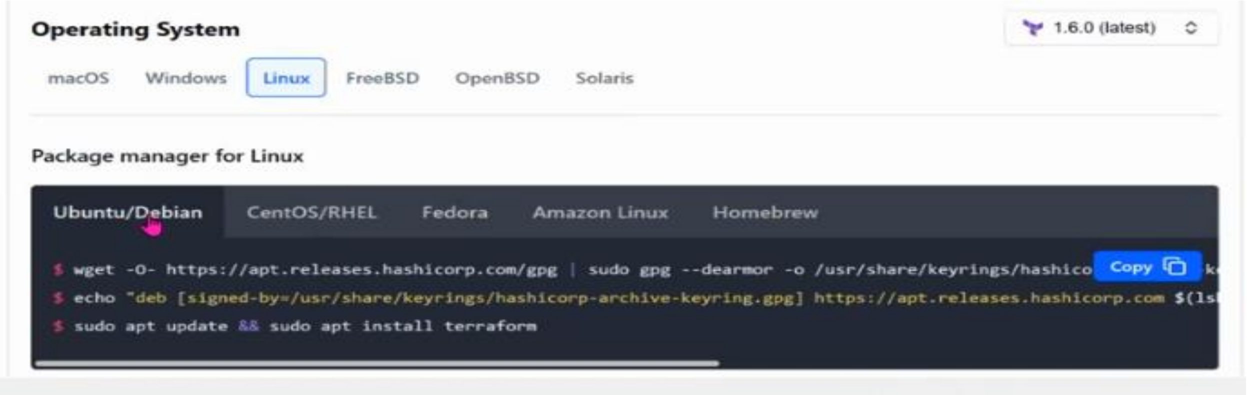
Install | Terraform | HashiCorp Developer
https://www.terraform.io/downloads

Install **Terraform.** Install or update to v1.5.7 (latest version) of **Terraform** to get started. Operating System. Package manager for macOS. $ brew tap hashicorp/tap $ brew install hashicorp/tap/terraform. Binary **download** for macOS. AMD64. Version: 1.5.7. **Download.** ARM64. Version: 1.5.7. **Download.** Release ... See more



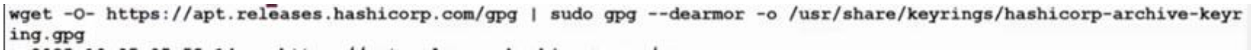
Install Terraform

Install or update to v1.6.0 (latest version) of Terraform to get started.

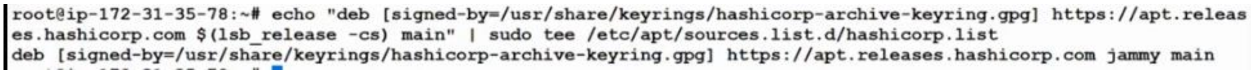


```
$ wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashico
$ echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(ls
$ sudo apt update && sudo apt install terraform
```

Copy the above commands and paste in Ubuntu linux

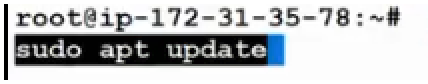


```
wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyr
ing.gpg
```



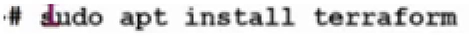
```
root@ip-172-31-35-78:~# echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releas
es.hashicorp.com $(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list
deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com jammy main
```

The sudo apt update command is a Linux system administration command that updates the list of available packages and their versions stored in the system's package index



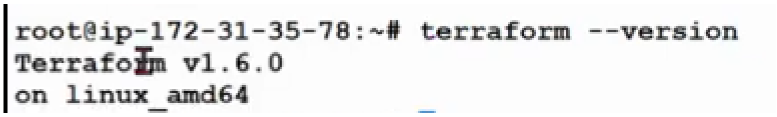
```
root@ip-172-31-35-78:~#
sudo apt update
```

To install terraform by using command as # sudo apt install terraform



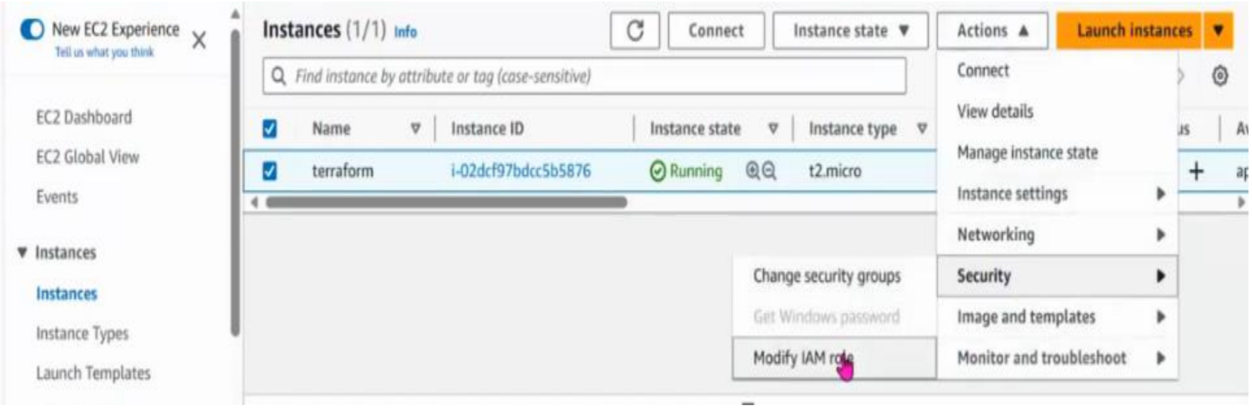
```
# sudo apt install terraform
```

Now we have to check whether the terraform installed or not



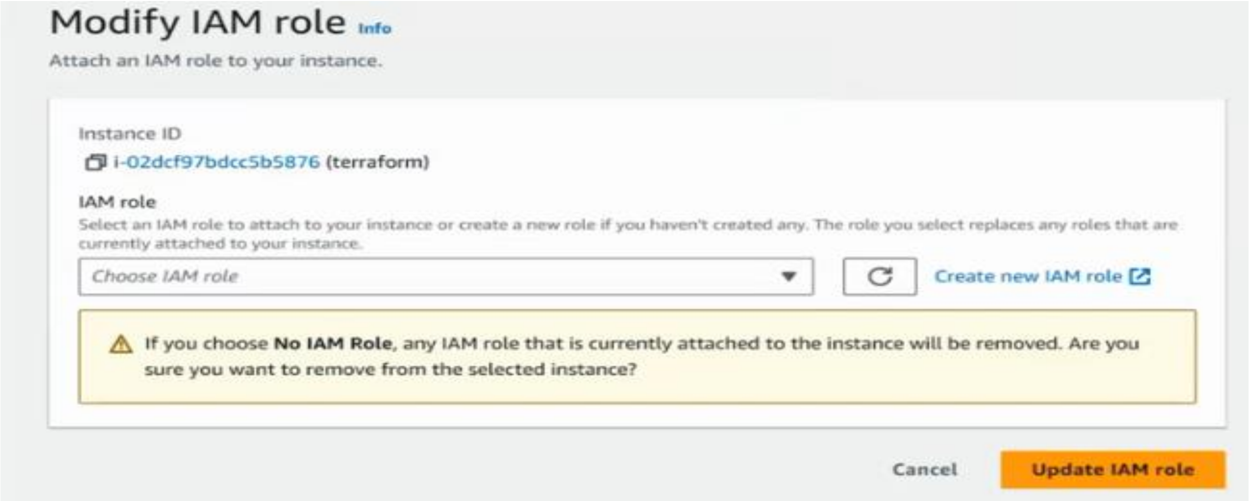
```
root@ip-172-31-35-78:~# terraform --version
Terraform v1.6.0
on linux_amd64
```

Now we have to create IAM role to give permission to terraform virtual machine to build VPC infrastructure.



| Name | Instance ID | Instance state | Instance type |
|---|---|---|---|
| terraform | i-02dcf97bdcc5b5876 | Running | t2.micro |

Click on “Create new IAM role”



Modify IAM role Info

Attach an IAM role to your instance.

Instance ID

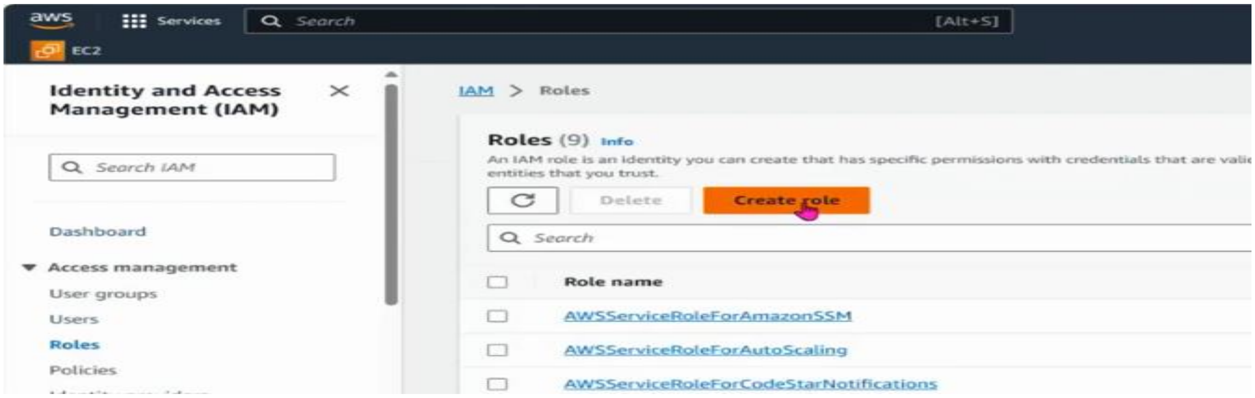
i-02dcf97bdcc5b5876 (terraform)

IAM role

Select an IAM role to attach to your instance or create a new role if you haven't created any. The role you select replaces any roles that are currently attached to your instance.

If you choose **No IAM Role**, any IAM role that is currently attached to the instance will be removed. Are you sure you want to remove from the selected instance?

Click on create role

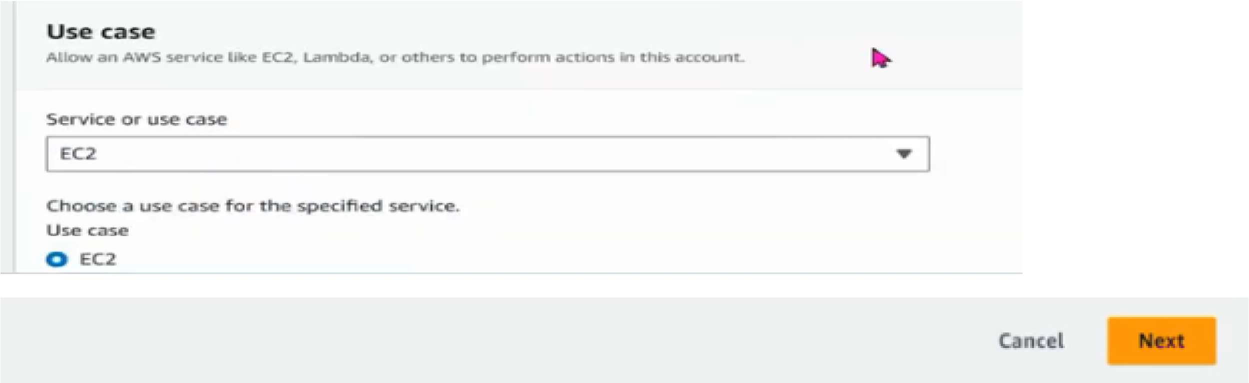


Roles (9) Info

An IAM role is an identity you can create that has specific permissions with credentials that are vali
entities that you trust.

| Role name |
|---|
| AWSServiceRoleForAmazonSSM |
| AWSServiceRoleForAutoScaling |
| AWSServiceRoleForCodeStarNotifications |

Select EC2 and click on next



**Use case**

Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

Service or use case

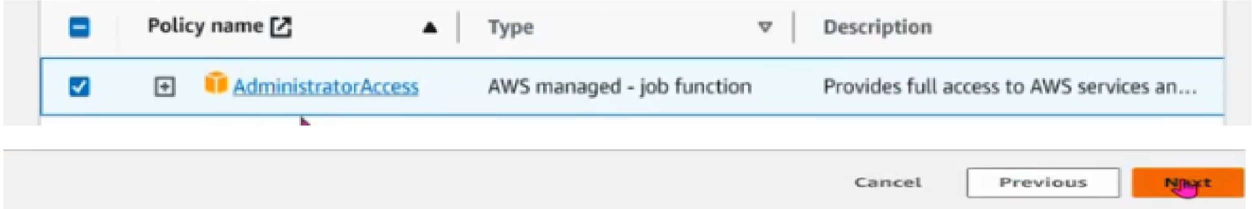
EC2

Choose a use case for the specified service.

Use case

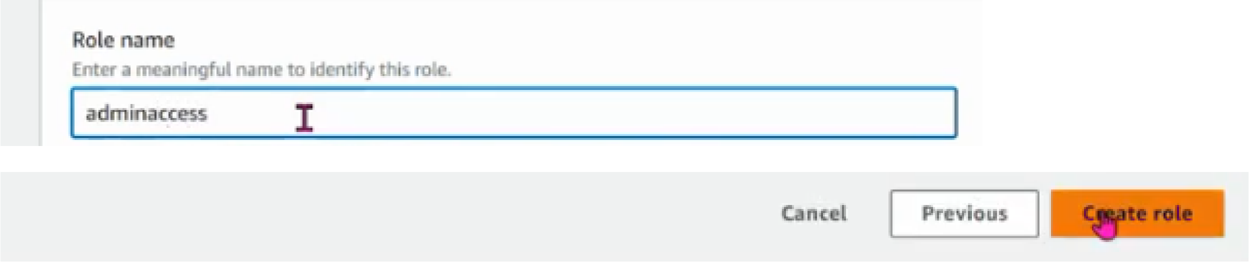
● EC2

Now we have to give full admin access and click on “Next”



| | | Policy name | Type | Description |
|---|---|---|---|---|
| ☑ | ⊞ | AdministratorAccess | AWS managed - job function | Provides full access to AWS services an... |

Set name of the role and click on “Create Role”

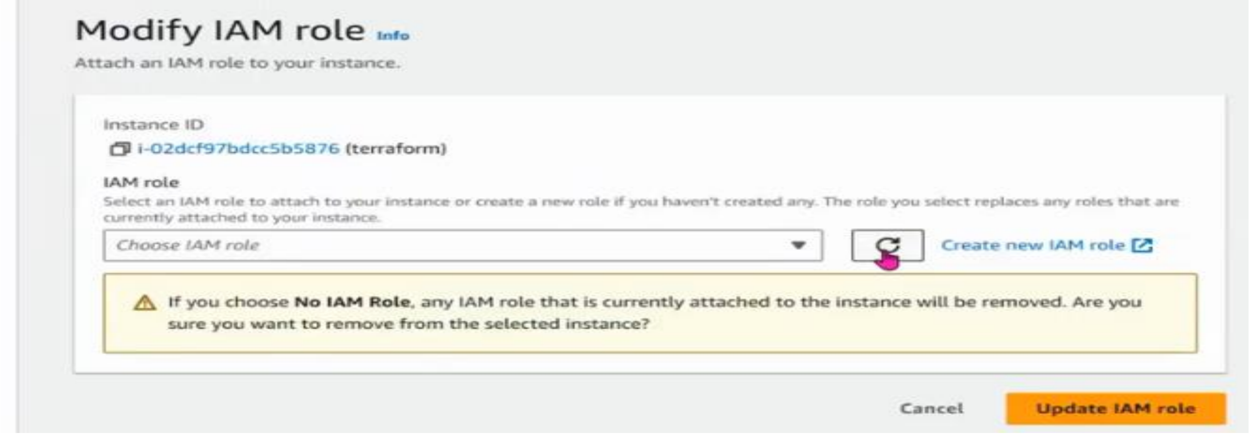


Role name

Enter a meaningful name to identify this role.

adminaccess

Now refresh IAM role and select “adminaccess” role which we have created and click on “Update IAM role”



**Modify IAM role** Info

Attach an IAM role to your instance.

Instance ID

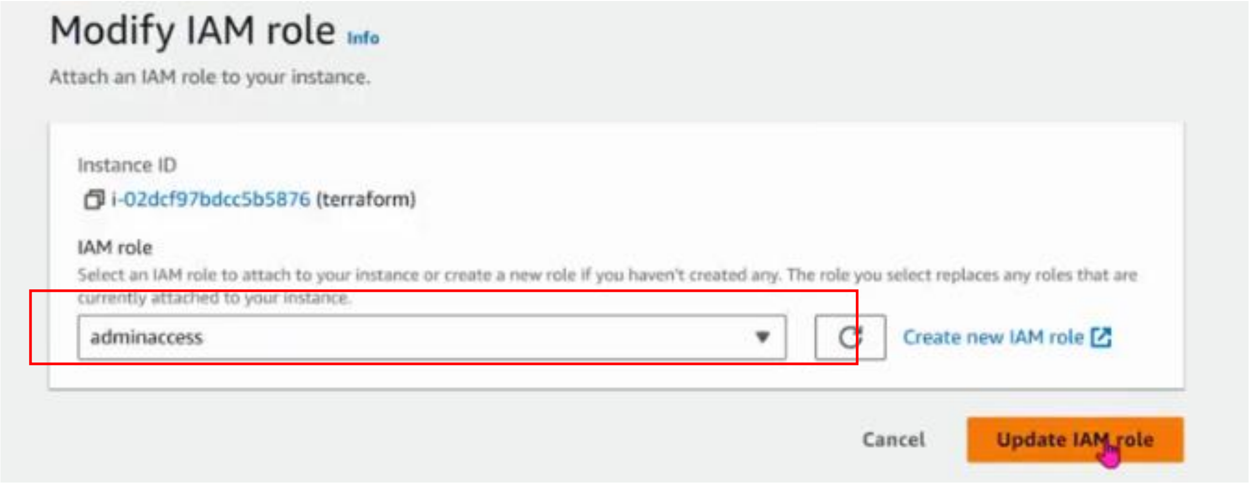
i-02dcf97bdcc5b5876 (terraform)

IAM role

Select an IAM role to attach to your instance or create a new role if you haven't created any. The role you select replaces any roles that are currently attached to your instance.

Choose IAM role

⚠ If you choose **No IAM Role**, any IAM role that is currently attached to the instance will be removed. Are you sure you want to remove from the selected instance?



Modify IAM role

Attach an IAM role to your instance.

Instance ID

i-02dcf97bdcc5b5876 (terraform)

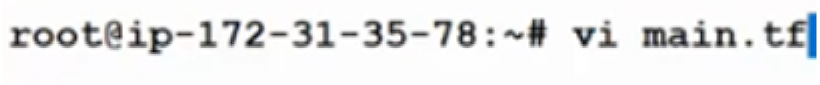
IAM role

Select an IAM role to attach to your instance or create a new role if you haven't created any. The role you select replaces any roles that are currently attached to your instance.

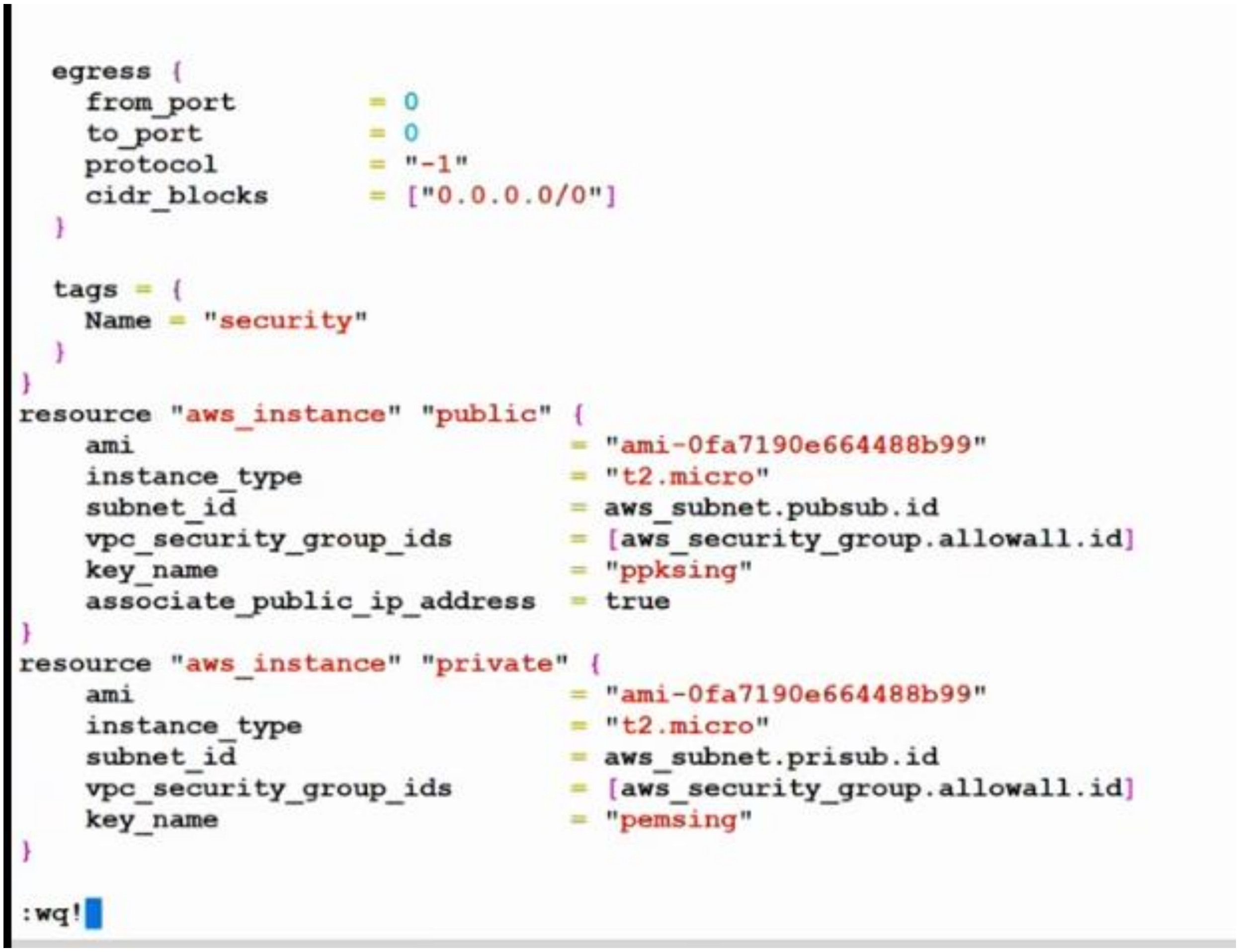
adminaccess

Now, we have successfully attached IAM role to EC2 instance.

Now, we have to create the file in Ubuntu linux then copy the entire code from visual studio code and paste in **main.tf file** then save and quit.

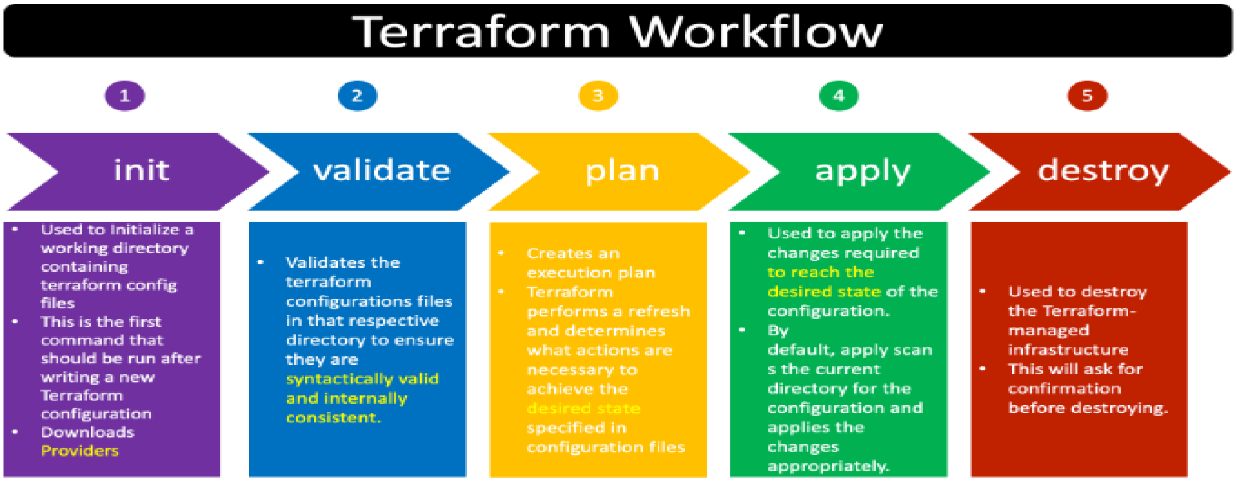


```
root@ip-172-31-35-78:~# vi main.tf
```



```
  egress {
    from_port          = 0
    to_port            = 0
    protocol           = "-1"
    cidr_blocks        = ["0.0.0.0/0"]
  }

  tags = {
    Name = "security"
  }
}
resource "aws_instance" "public" {
    ami                              = "ami-0fa7190e664488b99"
    instance_type                    = "t2.micro"
    subnet_id                        = aws_subnet.pubsub.id
    vpc_security_group_ids           = [aws_security_group.allowall.id]
    key_name                         = "ppksing"
    associate_public_ip_address      = true
}
resource "aws_instance" "private" {
    ami                              = "ami-0fa7190e664488b99"
    instance_type                    = "t2.micro"
    subnet_id                        = aws_subnet.prisub.id
    vpc_security_group_ids           = [aws_security_group.allowall.id]
    key_name                         = "pemsing"
}

:wq!
```

Now we need to apply terraform workflow command one by one in linux.



Terraform Workflow

1 init

- Used to Initialize a working directory containing terraform config files
- This is the first command that should be run after writing a new Terraform configuration
- Downloads Providers

2 validate

- Validates the terraform configurations files in that respective directory to ensure they are syntactically valid and internally consistent.

3 plan

- Creates an execution plan
- Terraform performs a refresh and determines what actions are necessary to achieve the desired state specified in configuration files

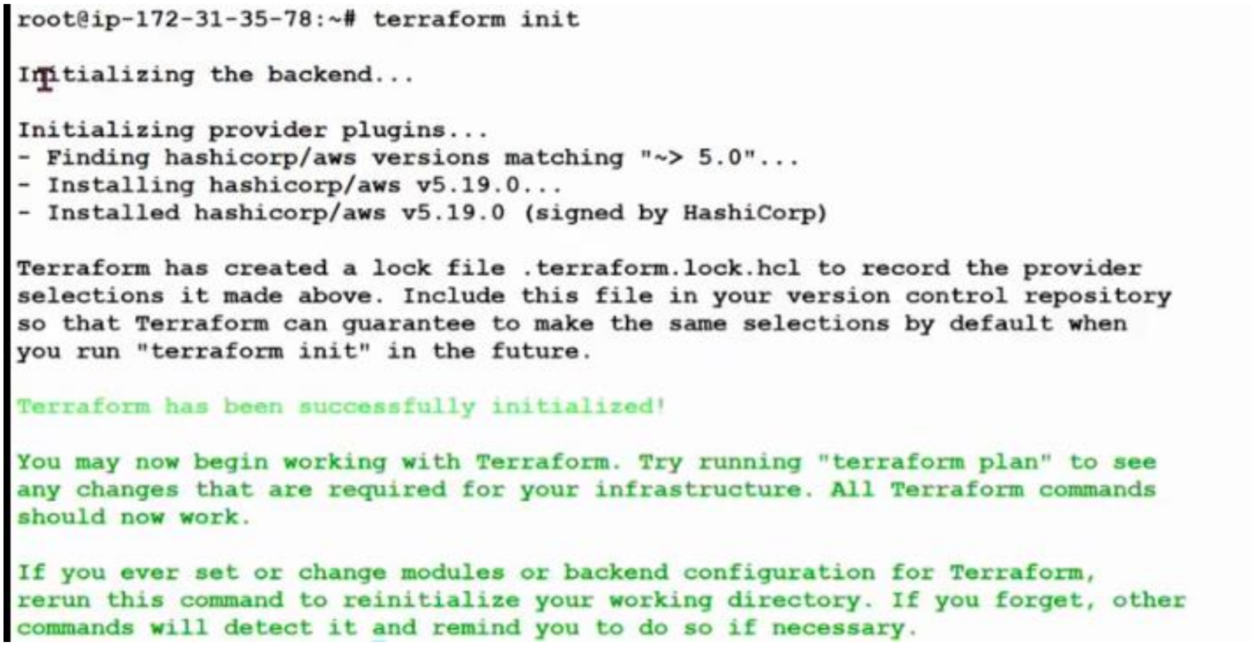
4 apply

- Used to apply the changes required to reach the desired state of the configuration.
- By default, apply scan s the current directory for the configuration and applies the changes appropriately.

5 destroy

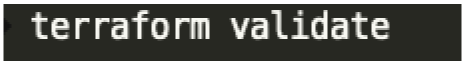
- Used to destroy the Terraform-managed infrastructure
- This will ask for confirmation before destroying.

- **Terraform init** command to initiazing provider plugins in backend.

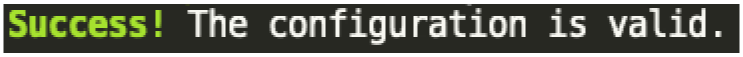


```
root@ip-172-31-35-78:~# terraform init

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 5.0"...
- Installing hashicorp/aws v5.19.0...
- Installed hashicorp/aws v5.19.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

- **terraform validate command** is used to verify the correctness of Terraform configuration files

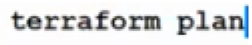


```
terraform validate
```

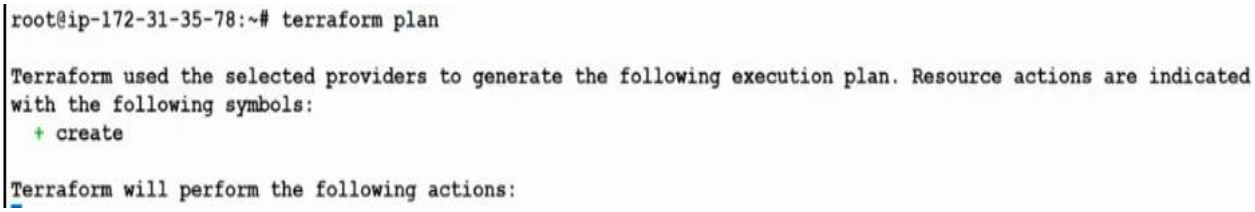


```
Success! The configuration is valid.
```

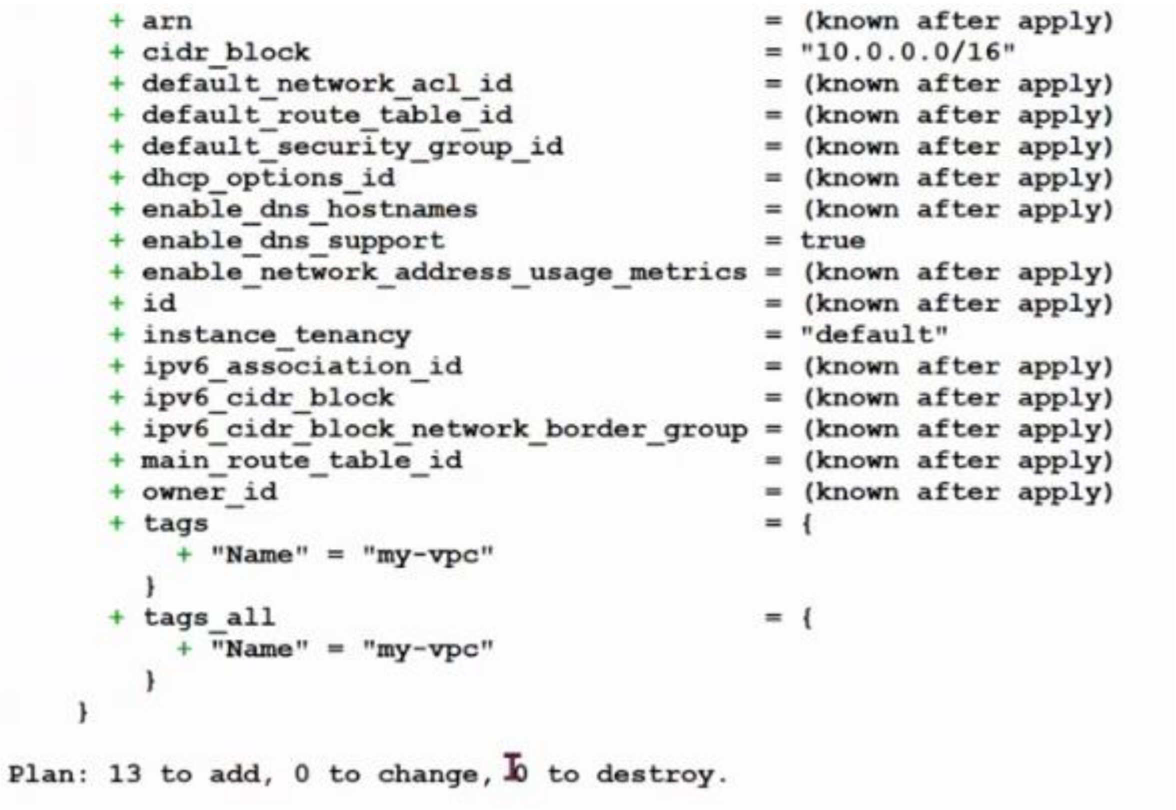
- The **terraform plan command** creates a plan consisting of a set of changes that will make your resources match your configuration.



```
terraform plan
```

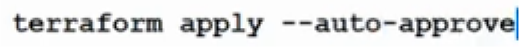


```
root@ip-172-31-35-78:~# terraform plan

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated
with the following symbols:
  + create

Terraform will perform the following actions:
```



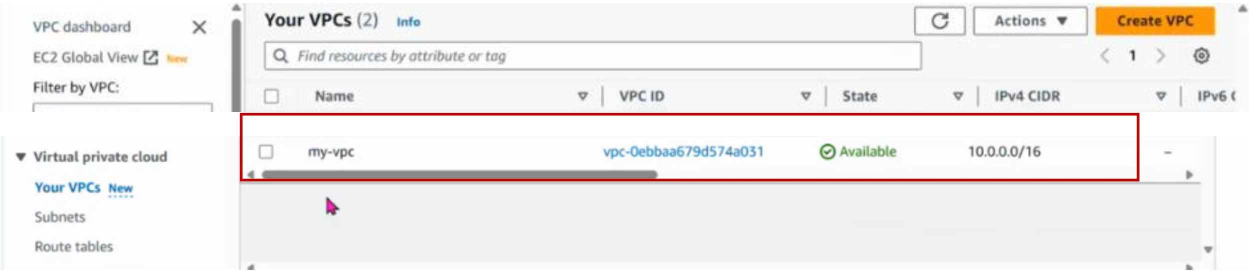
```
      + arn                                  = (known after apply)
      + cidr_block                           = "10.0.0.0/16"
      + default_network_acl_id               = (known after apply)
      + default_route_table_id               = (known after apply)
      + default_security_group_id            = (known after apply)
      + dhcp_options_id                      = (known after apply)
      + enable_dns_hostnames                 = (known after apply)
      + enable_dns_support                   = true
      + enable_network_address_usage_metrics = (known after apply)
      + id                                   = (known after apply)
      + instance_tenancy                     = "default"
      + ipv6_association_id                  = (known after apply)
      + ipv6_cidr_block                      = (known after apply)
      + ipv6_cidr_block_network_border_group = (known after apply)
      + main_route_table_id                  = (known after apply)
      + owner_id                             = (known after apply)
      + tags                                 = {
          + "Name" = "my-vpc"
        }
      + tags_all                             = {
          + "Name" = "my-vpc"
        }
    }

Plan: 13 to add, 0 to change, 0 to destroy.
```

- **terraform apply –auto-approve is the final command for execution**



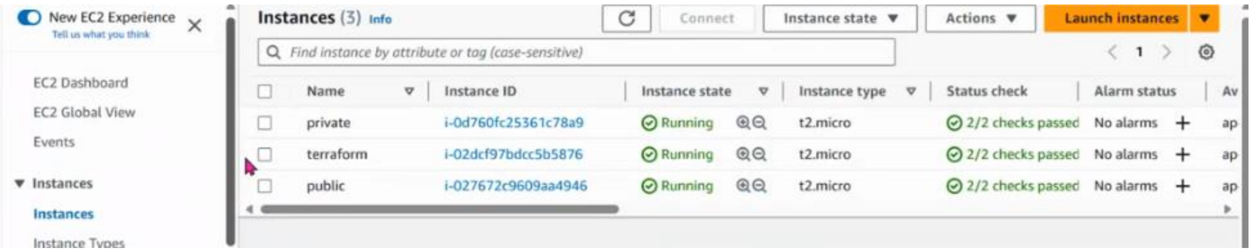
```
terraform apply --auto-approve
```

Now Virtual Private Cloud has been successfully created on AWS cloud.



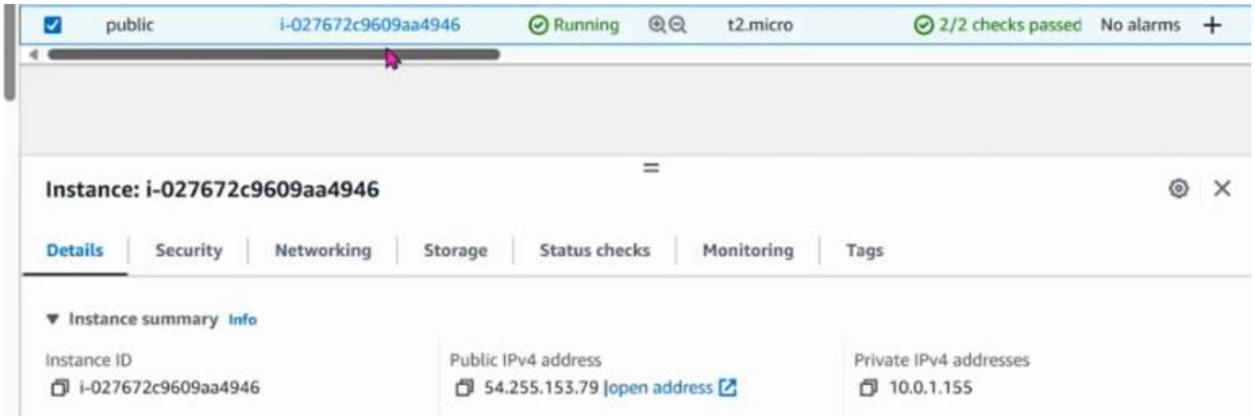
| □ | Name | VPC ID | State | IPv4 CIDR | IPv6 ( |
|---|---|---|---|---|---|
| □ | my-vpc | vpc-0ebbaa679d574a031 | ✓ Available | 10.0.0.0/16 | – |

Now two virtual machines (Public and private) has been successfully created



| □ | Name | Instance ID | Instance state | Instance type | Status check | Alarm status | Av |
|---|---|---|---|---|---|---|---|
| □ | private | i-0d760fc25361c78a9 | ✓ Running | t2.micro | ✓ 2/2 checks passed | No alarms + | ap |
| □ | terraform | i-02dcf97bdcc5b5876 | ✓ Running | t2.micro | ✓ 2/2 checks passed | No alarms + | ap |
| □ | public | i-027672c9609aa4946 | ✓ Running | t2.micro | ✓ 2/2 checks passed | No alarms + | ap |

Now we have to connect public virtual machine and to check whether the internet is working or not.



|  | Name | Instance ID | Instance state |  | Instance type | Status check | Alarm status |  |
|---|---|---|---|---|---|---|---|---|
| ☑ | public | i-027672c9609aa4946 | Running | 🔍🔍 | t2.micro | 2/2 checks passed | No alarms | + |

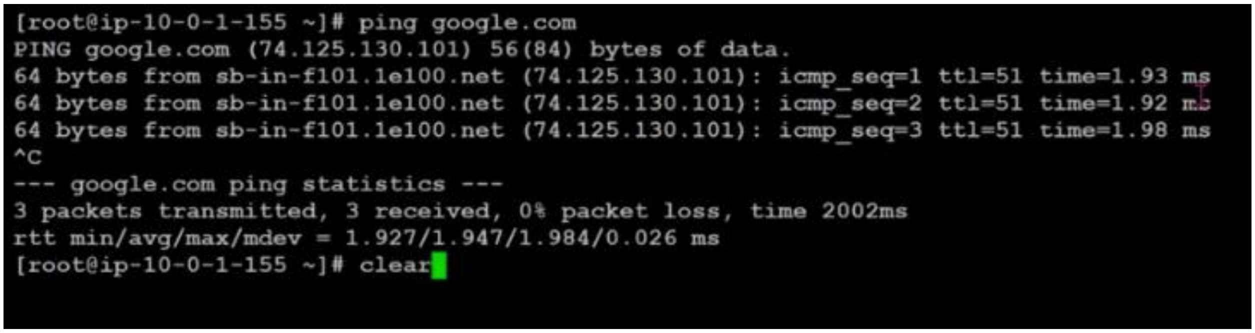
Instance: i-027672c9609aa4946

▼ Instance summary Info

Instance ID
i-027672c9609aa4946

Public IPv4 address
54.255.153.79 | open address

Private IPv4 addresses
10.0.1.155

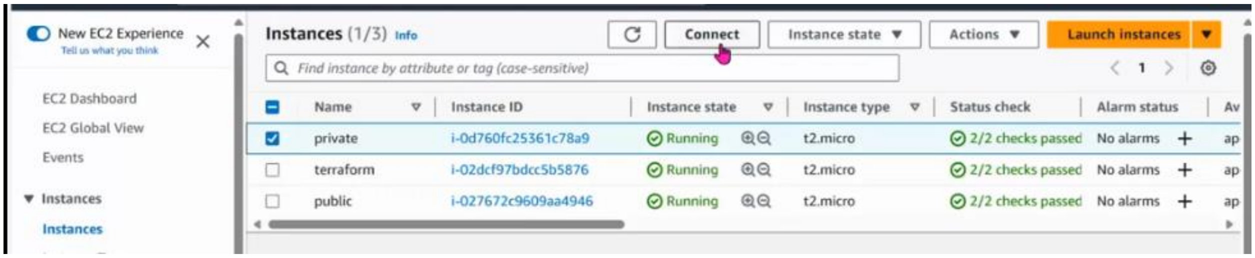


```
[root@ip-10-0-1-155 ~]# ping google.com
PING google.com (74.125.130.101) 56(84) bytes of data.
64 bytes from sb-in-f101.1e100.net (74.125.130.101): icmp_seq=1 ttl=51 time=1.93 ms
64 bytes from sb-in-f101.1e100.net (74.125.130.101): icmp_seq=2 ttl=51 time=1.92 ms
64 bytes from sb-in-f101.1e100.net (74.125.130.101): icmp_seq=3 ttl=51 time=1.98 ms
^C
--- google.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 1.927/1.947/1.984/0.026 ms
[root@ip-10-0-1-155 ~]# clear
```

Now the internet is available and working fine with public network.

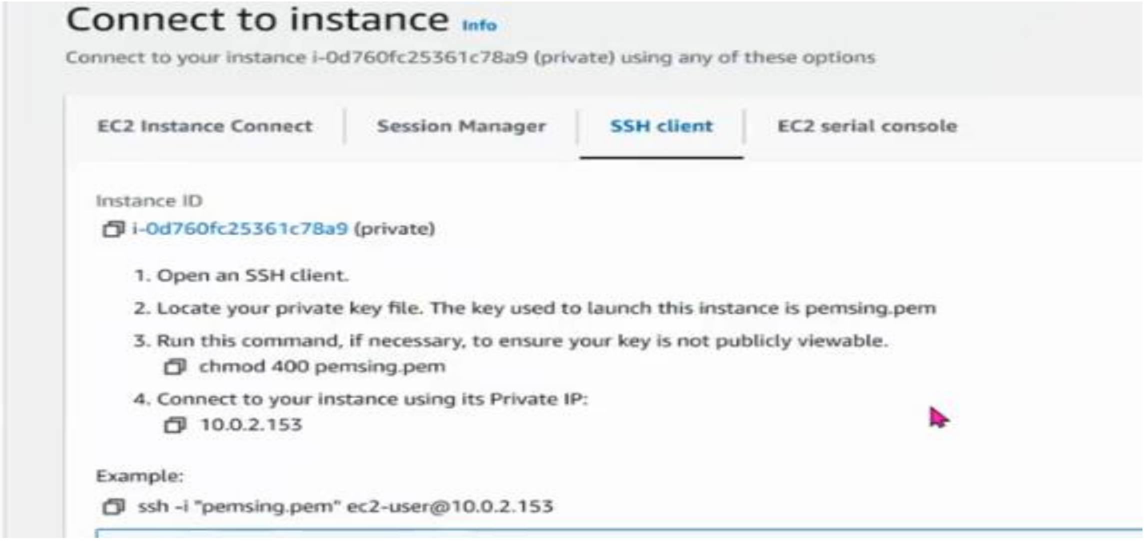
After that we have to check whether the internet is working or not in private network.

Now we need to connect private network and click on connect



Instances (1/3) Info

| ☐ | Name | Instance ID | Instance state |  | Instance type | Status check | Alarm status |  | Av |
|---|---|---|---|---|---|---|---|---|---|
| ☑ | private | i-0d760fc25361c78a9 | Running | 🔍🔍 | t2.micro | 2/2 checks passed | No alarms | + | ap |
| ☐ | terraform | i-02dcf97bdcc5b5876 | Running | 🔍🔍 | t2.micro | 2/2 checks passed | No alarms | + | ap |
| ☐ | public | i-027672c9609aa4946 | Running | 🔍🔍 | t2.micro | 2/2 checks passed | No alarms | + | ap |



Connect to instance

Connect to your instance i-0d760fc25361c78a9 (private) using any of these options

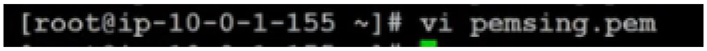
Instance ID

i-0d760fc25361c78a9 (private)

1. Open an SSH client.
2. Locate your private key file. The key used to launch this instance is pemsing.pem
3. Run this command, if necessary, to ensure your key is not publicly viewable.
   chmod 400 pemsing.pem
4. Connect to your instance using its Private IP:
   10.0.2.153

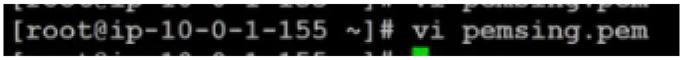
Example:

ssh -i "pemsing.pem" ec2-user@10.0.2.153

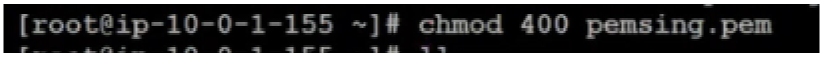


```
[root@ip-10-0-1-155 ~]# vi pemsing.pem
```

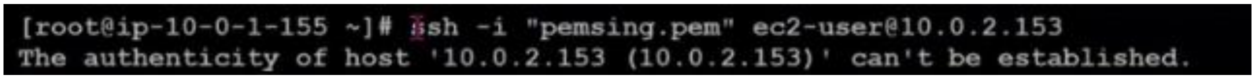
Open the file and paste the private key credential then save and quite then give the permission using the command as “chmod 400 pemsing.pem”



```
[root@ip-10-0-1-155 ~]# vi pemsing.pem
```

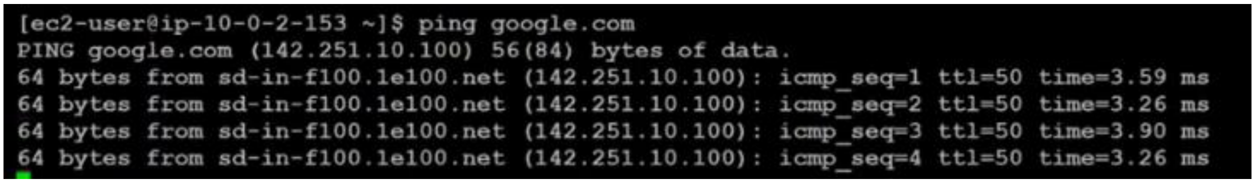


```
[root@ip-10-0-1-155 ~]# chmod 400 pemsing.pem
```



```
[root@ip-10-0-1-155 ~]# ssh -i "pemsing.pem" ec2-user@10.0.2.153
The authenticity of host '10.0.2.153 (10.0.2.153)' can't be established.
```

Now the internet is working fine with private network.



```
[ec2-user@ip-10-0-2-153 ~]$ ping google.com
PING google.com (142.251.10.100) 56(84) bytes of data.
64 bytes from sd-in-f100.1e100.net (142.251.10.100): icmp_seq=1 ttl=50 time=3.59 ms
64 bytes from sd-in-f100.1e100.net (142.251.10.100): icmp_seq=2 ttl=50 time=3.26 ms
64 bytes from sd-in-f100.1e100.net (142.251.10.100): icmp_seq=3 ttl=50 time=3.90 ms
64 bytes from sd-in-f100.1e100.net (142.251.10.100): icmp_seq=4 ttl=50 time=3.26 ms
```