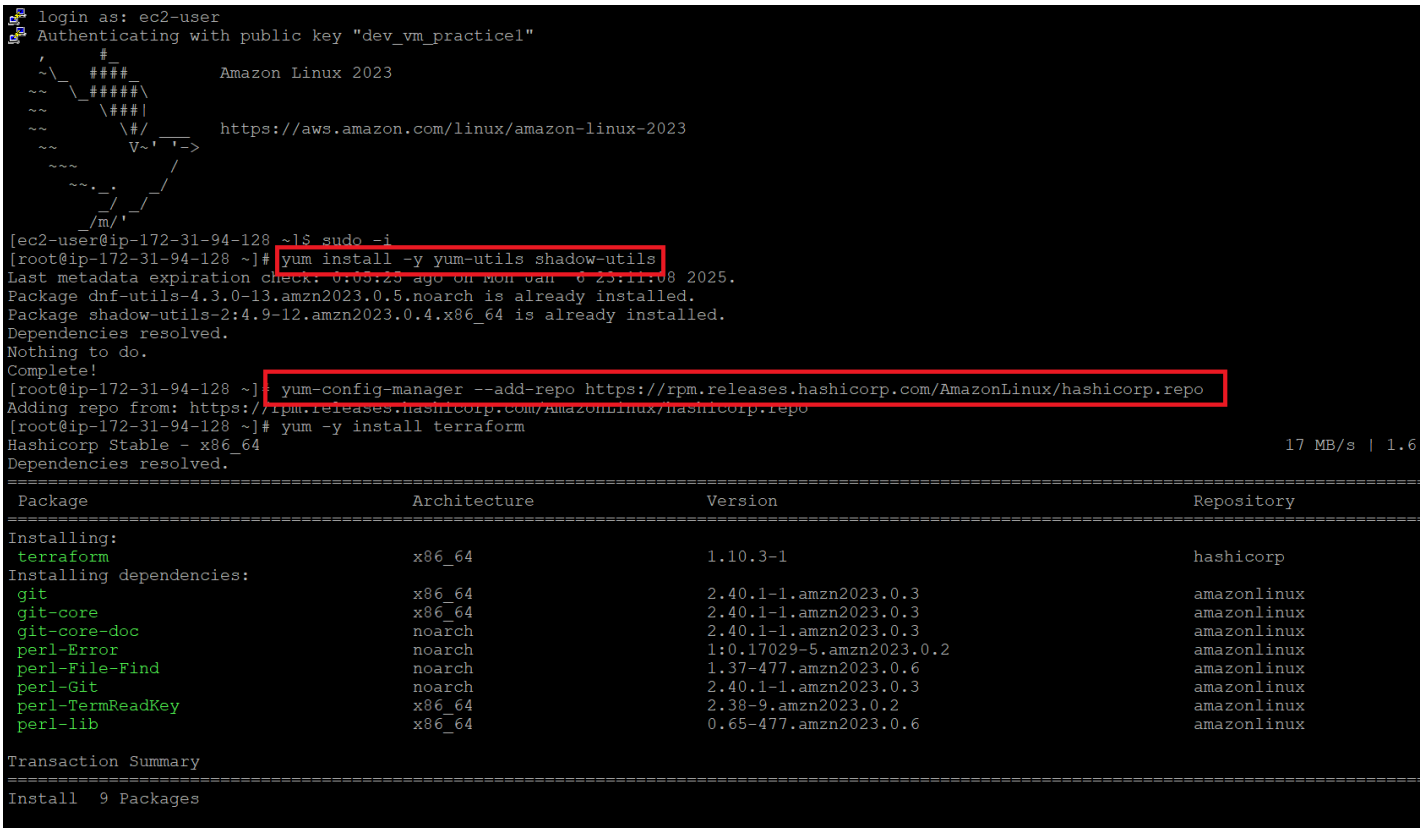
Terraform

Create VPC components using terraform
https://www.youtube.com/watch?v=sg86vd1J-qo

Step 1: First create an instance and login to putty
Step 2: Installing Terraform

Install Terraform https://developer.hashicorp.com/terraform/install
Under Linux Package manager select the required OS Amazon linux or any other
For Amazon Linux
sudo yum install -y yum-utils shadow-utils

Sudo
yum-config-manager --add-repo https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo



sudo yum -y install terraform

```
[root@ip-172-31-94-128 ~ # yum -y install terraform
Last metadata expiration check: 0:03:50 ago on Mon Jan  6 23:16:56 2025.
Package terraform-1.10.3-1.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[root@ip-172-31-94-128 ~]#
```

aws –version
aws –configure
Enter access key: ******************************
Enter secret access key: **********************************
Enter default region name: us-east-1
Enter output format: JSON

```
[root@ip-172-31-94-128 ~]# aws --version
aws-cli/2.17.18 Python/3.9.20 Linux/6.1.119-129.201.amzn2023.x86_64 source/x86_64.amzn.2023
[root@ip-172-31-94-128 ~]# aws configure
AWS Access Key ID [None]:
AWS Secret Access Key [None]:
Default region name [None]: us-east-1
Default output format [None]: json
[root@ip-172-31-94-128 ~]#
```

Create a directory in any folder depends on the project requirement **mkdir /app**
Go to folder **cd /app**
Present working directory **pwd**

Provide information to terraform which cloud provider/platform we are using
Create a file **vi main.tf (all the file extension should be .tf)**
Inside the file main.tf we need to mention which platform we are using
Provider “aws”{
}
:wq!

Cat main.tf

```
provider "aws"{
}
~
~
~
~
~
~
~
~
~
```

```
[root@ip-172-31-94-128 app]# vi main.tf
[root@ip-172-31-94-128 app]# ls
instance.tf  main.tf
[root@ip-172-31-94-128 app]# instance.tf
-bash: instance.tf: command not found
[root@ip-172-31-94-128 app]# vi instance.tf
[root@ip-172-31-94-128 app]#
```

root@ip-172-31-94-128:/app

```
## Create basic VPC

resource "aws_vpc" "main_vpc" {
  cidr_block       = "10.0.0.0/16"
  instance_tenancy = "default"

  tags = {
    Name = "dev_vpc_practice"
  }
}


## Create PUBLIC SUBNET

resource "aws_subnet" "main_public_subnet1" {
  vpc_id     = aws_vpc.main_vpc.id
  cidr_block = "10.0.11.0/24"

  tags = {
    Name = "dev_public_subnet"
  }
}


## Create PRIVATE SUBNET1

resource "aws_subnet" "main_private_subnet1" {
  vpc_id     = aws_vpc.main_vpc.id
  cidr_block = "10.0.1.0/24"

  tags = {
    Name = "dev_private_subnet1"
  }
}


## Create INTERNET GATEWAY

resource "aws_internet_gateway" "gw" {
  vpc_id = aws_vpc.main_vpc.id

  tags = {
    Name = "dev_igw"
```

```
resource "aws_instance" "eip_ag" {
  # ... other arguments ...
  vpc = true
  depends_on = [aws_internet_gateway.gw]
}


## Create NAT GATEWAY

resource "aws_nat_gateway" "example" {
  allocation_id = aws_instance.eip_ag.id
  subnet_id     = aws_subnet.main_private_subnet1.id

  tags = {
    Name = "gw NAT"
  }


# To ensure proper ordering, it is recommended to add an explicit dependency
# on the Internet Gateway for the VPC.

depends_on = [aws_internet_gateway.gw]
}


##  Create default Route table

resource "aws_default_route_table" "example" {
  default_route_table_id = aws_vpc.example.default_route_table_id

  route {
    cidr_block = "10.0.1.0/24"
    gateway_id = aws_internet_gateway.gw.id
  }

  tags = {
    Name = "example"
  }
}
```

```
##   Create route table

resource "aws_route_table" "example" {
  vpc_id = aws_vpc.main_vpc.id

  route = []

  tags = {
    Name = "example"
  }
}

## Create Route table association

resource "aws_route_table_association" "a" {
  subnet_id      = aws_subnet.main_private_subnet1.id
  route_table_id = aws_route_table.example.id
}

## Create Route

resource "aws_route" "r" {
  route_table_id            = aws_route_table.example.id
  destination_cidr_block    = "0.0.0.0/0"
  nat_gateway_id = "aws_nat_gateway.example.id"
}

## Create security group

resource "aws_security_group" "security_grp" {
  # ... other configuration ...
name = "security_grp"
vpc_ip = aws_vpc.main_vpc.id


  ingress {
    from_port        = 22
    to_port          = 22
    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-grp"
}
}
```

Command **terraform init**
Command terraform plan

```
[root@ip-172-31-94-128 app]# terraform init
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/tls from the dependency lock file
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/tls v4.0.6
- Using previously-installed hashicorp/aws v5.82.2

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.
[root@ip-172-31-94-128 app]# terraform plan

Terraform used the selected providers to generate the following execution plan. Resource actions
  + create

Terraform will perform the following actions:

  # aws_eip.eip_ag will be created
  + resource "aws_eip" "eip_ag" {
      + allocation_id        = (known after apply)
      + arn                  = (known after apply)
      + association_id       = (known after apply)
      + carrier_ip           = (known after apply)
      + customer_owned_ip    = (known after apply)
      + domain               = "vpc"
      + id                   = (known after apply)
      + instance             = (known after apply)
      + ipam_pool_id         = (known after apply)
      + network_border_group = (known after apply)
      + network_interface    = (known after apply)
      + private_dns          = (known after apply)
      + private_ip           = (known after apply)
      + ptr_record           = (known after apply)
```

```
    }

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

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.
[root@ip-172-31-94-128 app]#
```

```
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

tls_private_key.rsa-4096-example: Creating...
aws_key_pair.main_key: Creating...
aws_vpc.main_vpc: Creating...
aws_key_pair.main_key: Creation complete after 0s [id=main_key]
aws_vpc.main_vpc: Creation complete after 1s [id=vpc-0cce0328534712a78]
aws_subnet.main_private_subnet1: Creating...
aws_internet_gateway.gw: Creating...
aws_security_group.security_grp: Creating...
aws_route_table.example: Creating...
aws_subnet.main_public_subnet1: Creating...
aws_internet_gateway.gw: Creation complete after 1s [id=igw-0a7f56e827b444b47]
aws_route_table.example: Creation complete after 1s [id=rtb-0c065f0889a071fa7]
aws_route.r: Creating...
aws_subnet.main_public_subnet1: Creation complete after 1s [id=subnet-057ac76887a7683e8]
aws_subnet.main_private_subnet1: Creation complete after 1s [id=subnet-033a3eb501cffdf58]
aws_route_table_association.a: Creating...
aws_route_table_association.a: Creation complete after 0s [id=rtbassoc-054b55b38bf5a7341]
tls_private_key.rsa-4096-example: Creation complete after 2s [id=e8a898958169f72b885ddf836705c7a6ce3ac3e5]
aws_security_group.security_grp: Creation complete after 2s [id=sg-0a7ea2d354d408574]
aws_instance.main_instance: Creating...
aws_instance.main_instance: Still creating... [10s elapsed]
aws_instance.main_instance: Creation complete after 13s [id=i-0046b6fcb36acddf0]
aws_eip.eip_ag: Creating...
aws_eip.eip_ag: Creation complete after 2s [id=eipalloc-077d547ba61674a69]
aws_nat_gateway.example: Creating...
aws_nat_gateway.example: Still creating... [10s elapsed]
aws_nat_gateway.example: Still creating... [20s elapsed]
aws_nat_gateway.example: Still creating... [30s elapsed]
aws_nat_gateway.example: Still creating... [40s elapsed]
aws_nat_gateway.example: Still creating... [50s elapsed]
aws_nat_gateway.example: Still creating... [1m0s elapsed]
aws_nat_gateway.example: Still creating... [1m10s elapsed]
aws_nat_gateway.example: Still creating... [1m20s elapsed]
aws_nat_gateway.example: Still creating... [1m30s elapsed]
```

```
[root@ip-172-31-94-128 app]# terraform state list
aws_eip.eip_ag
aws_instance.main_instance
aws_internet_gateway.gw
aws_key_pair.main_key
aws_nat_gateway.example
aws_route_table.example
aws_route_table_association.a
aws_security_group.security_grp
aws_subnet.main_private_subnet1
aws_subnet.main_public_subnet1
aws_vpc.main_vpc
tls_private_key.rsa-4096-example
[root@ip-172-31-94-128 app]#
```

To create an EC2 instance in a **VPC** using **Terraform** on **Amazon Linux**, you will need to define a number of components in your Terraform configuration. The most important components of a VPC and the EC2 instance setup include:

Required Components:

1. **VPC (Virtual Private Cloud)**: A logically isolated section of the AWS cloud where you can define your network resources.
2. **Subnet**: A range of IP addresses within the VPC.
3. **Internet Gateway**: Allows communication between instances in your VPC and the internet.
4. **Route Table**: A set of rules, known as routes, used to determine where network traffic is directed.
5. **Security Group**: A virtual firewall for your EC2 instances to control inbound and outbound traffic.
6. **EC2 Instance**: The actual virtual machine (or instance) that will run your applications.

Basic Steps to Create an EC2 Instance in a VPC using Terraform

Here's a high-level overview of the Terraform configuration to create a basic setup for an EC2 instance within a VPC:

1. Create the VPC

hcl
Copy code

```
resource "aws_vpc" "main_vpc" {
  cidr_block = "10.0.0.0/16"  # This defines the IP range for the VPC
}
```

2. Create a Subnet

Subnets divide the VPC into smaller networks. You need at least one subnet in the VPC where your EC2 instance will reside.

hcl
Copy code

```
resource "aws_subnet" "main_subnet" {
  vpc_id                  = aws_vpc.main_vpc.id
  cidr_block              = "10.0.1.0/24"  # Define a smaller subnet within the
VPC's CIDR block
  availability_zone       = "us-east-1a"    # Define the availability zone
(ensure this is correct for your region)
  map_public_ip_on_launch = true            # Allow public IPs for instances in
this subnet
}
```

3. Create an Internet Gateway

An **Internet Gateway** is required if you want your EC2 instance to communicate with the outside world.

hcl
Copy code
```hcl
resource "aws_internet_gateway" "main_gateway" {
  vpc_id = aws_vpc.main_vpc.id
}
```

4. Create a Route Table and Associate It with the Subnet

You'll need a route table to ensure that traffic from the EC2 instance can flow to the internet through the Internet Gateway.

hcl
Copy code
```hcl
resource "aws_route_table" "main_route_table" {
  vpc_id = aws_vpc.main_vpc.id

  route {
    cidr_block = "0.0.0.0/0"  # Allow all traffic to flow to the internet
    gateway_id = aws_internet_gateway.main_gateway.id
  }
}

resource "aws_route_table_association" "main_association" {
  subnet_id      = aws_subnet.main_subnet.id
  route_table_id = aws_route_table.main_route_table.id
}
```

5. Create a Security Group

A **Security Group** is a virtual firewall to control inbound and outbound traffic. For an EC2 instance, you typically want to allow SSH (port 22) and HTTP (port 80) for Linux instances.

hcl
Copy code
```hcl
resource "aws_security_group" "instance_sg" {
  vpc_id = aws_vpc.main_vpc.id

  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
```

```
    cidr_blocks = ["0.0.0.0/0"]  # Allow SSH from anywhere (be cautious in
production environments)
  }

  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]  # Allow HTTP from anywhere
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"   # Allow all outbound traffic
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

6. Create the EC2 Instance

Finally, you'll create an EC2 instance inside the subnet, using the security group and VPC.

hcl
Copy code

```
resource "aws_instance" "main_instance" {
  ami             = "ami-01816d07b1128cd2d"  # Replace with the correct AMI ID
for Amazon Linux 2 or another Linux AMI
  instance_type   = "t2.micro"               # You can choose another instance
type if needed
  subnet_id       = aws_subnet.main_subnet.id
  security_groups = [aws_security_group.instance_sg.name]

  tags = {
    Name = "MyEC2Instance"
  }
}
```

Complete Example in Terraform:

hcl
Copy code

```
provider "aws" {
```

```
  region = "us-east-1"  # Specify your region
}

# Create VPC
resource "aws_vpc" "main_vpc" {
  cidr_block = "10.0.0.0/16"
}

# Create Subnet
resource "aws_subnet" "main_subnet" {
  vpc_id                  = aws_vpc.main_vpc.id
  cidr_block              = "10.0.1.0/24"
  availability_zone       = "us-east-1a"
  map_public_ip_on_launch = true
}

# Create Internet Gateway
resource "aws_internet_gateway" "main_gateway" {
  vpc_id = aws_vpc.main_vpc.id
}

# Create Route Table and Association
resource "aws_route_table" "main_route_table" {
  vpc_id = aws_vpc.main_vpc.id

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

resource "aws_route_table_association" "main_association" {
  subnet_id      = aws_subnet.main_subnet.id
  route_table_id = aws_route_table.main_route_table.id
}

# Create Security Group
resource "aws_security_group" "instance_sg" {
  vpc_id = aws_vpc.main_vpc.id

  ingress {
    from_port   = 22
    to_port     = 22
```

```
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    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"]
  }
}

# Create EC2 Instance
resource "aws_instance" "main_instance" {
  ami             = "ami-01816d07b1128cd2d"  # Use the correct Amazon Linux 2
AMI ID
  instance_type   = "t2.micro"
  subnet_id       = aws_subnet.main_subnet.id
  security_groups = [aws_security_group.instance_sg.name]

  tags = {
    Name = "MyEC2Instance"
  }
}
```

Summary of Components:

1. **VPC**: Defines the network environment.
2. **Subnet**: Defines a specific network segment within the VPC for your EC2 instance.
3. **Internet Gateway**: Provides internet connectivity to resources in the VPC.
4. **Route Table**: Configures routes for traffic flow in the VPC.
5. **Security Group**: Specifies firewall rules for the EC2 instance.
6. **EC2 Instance**: Creates the virtual machine within the VPC.

Optional Additional Configurations:

- **Elastic IP**: If you need a static public IP for your EC2 instance.

- **Key Pair**: To SSH into your EC2 instance securely.

Let me know if you need more information or help with specific configurations!

```
aws_route_table_association.a: Destroying... [id=rtbassoc-0c76b3e132a05c8fa]
aws_default_route_table.example: Modifying... [id=rtb-030ead404cf05ff31]
aws_route_table_association.a: Destruction complete after 1s
aws_subnet.public_subnet: Destroying... [id=subnet-08c5f60fcc48f5ef8]
aws_subnet.public_subnet: Destruction complete after 0s
aws_subnet.public_subnet: Creating...
aws_default_route_table.example: Modifications complete after 1s [id=rtb-030ead404cf05ff31]
aws_subnet.public_subnet: Creation complete after 1s [id=subnet-03ab1f2ae6f1d9cc8]
aws_instance.ec2_main: Creating...
aws_route_table_association.a: Creating...
aws_route_table_association.a: Creation complete after 0s [id=rtbassoc-026e9bb4171431ba0]
aws_instance.ec2_main: Still creating... [10s elapsed]
aws_instance.ec2_main: Creation complete after 12s [id=i-01f532b8c1a52f0c1]
aws_volume_attachment.ebs_attachment: Creating...
aws_eip.example: Modifying... [id=eipalloc-00cfc883922f1ee64]
aws_eip.example: Modifications complete after 2s [id=eipalloc-00cfc883922f1ee64]
aws_volume_attachment.ebs_attachment: Still creating... [10s elapsed]
aws_volume_attachment.ebs_attachment: Still creating... [20s elapsed]
aws_volume_attachment.ebs_attachment: Creation complete after 21s [id=vai-2762006552]

Apply complete! Resources: 4 added, 2 changed, 2 destroyed.
```

```
[root@ip-172-31-17-19 app]# terraform state list
data.aws_availability_zones.available
aws_default_route_table.example
aws_ebs_volume.ebs_main
aws_eip.example
aws_instance.ec2_main
aws_internet_gateway.gw
aws_route_table.route_main
aws_route_table_association.a
aws_security_group.sg_grp
aws_subnet.public_subnet
aws_volume_attachment.ebs_attachment
aws_vpc.vpc_main
[root@ip-172-31-17-19 app]#
```