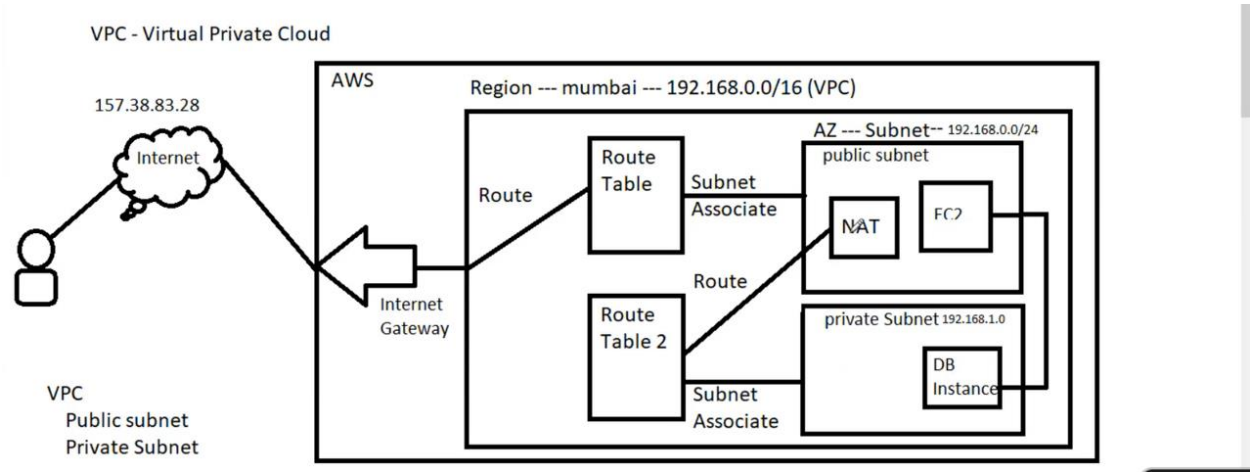
End-to-End VPC Architecture on AWS using Terraform

VPC (Virtual Private Cloud) Architecture



Step-by-Step Provisioning using Terraform:

1. **Create VPC – Define CIDR block (e.g., `192.168.0.0/16`)**

vpc.tf > resource "aws_vpc" "Custom-vpc"

```
1  resource "aws_vpc" "Custom-vpc" {
2    cidr_block = "192.168.0.0/16"
3
4    tags = {
5      Name = "Custom-vpc"
6    }
7  }
```

2. **Create Subnets** – Public & Private with respective CIDRs

➔ ***Public***

```
resource "aws_subnet" "public" {
  vpc_id            = aws_vpc.Custom-vpc.id
  cidr_block        = "192.168.0.0/24"
  availability_zone = "us-east-1a"

  tags = {
    Name = "public-subnet"
  }
}
```

➔ ***Private***

```
resource "aws_subnet" "private" {
  vpc_id            = aws_vpc.Custom-vpc.id
  cidr_block        = "192.168.1.0/26"
  availability_zone = "us-east-1b"

  tags = {
    Name = "private-subnet"
  }
}
```

3. Create Internet Gateway (IGW) – Attach to VPC

```
resource "aws_internet_gateway" "gw" {
  vpc_id = aws_vpc.Custom-vpc.id

  tags = {
    Name = "igw"
  }
}
```

4. Map IGW routes with route table and map with subnet Association.

➔ **Map Routes & Associations** –

- Route IGW in public route table

```
resource "aws_route_table" "NAT-route" {
  vpc_id = aws_vpc.Custom-vpc.id

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

  tags = {
    Name = "Nat-route"
  }
}
```

➔ **Associate route tables with subnets**

```
resource "aws_route_table_association" "a" {
  subnet_id      = aws_subnet.public.id
  route_table_id = aws_route_table.route-table.id
}
```

5. **Create Security Groups** – Open required ports for EC2, DB, etc.

```
resource "aws_security_group" "allow_tls" {
  name        = "allow_tls"
  description = "Allow TLS inbound traffic and all outbound traffic"
  vpc_id      = aws_vpc.Custom-vpc.id

  tags = {
    Name = "allow_tls"
  }
}

resource "aws_vpc_security_group_ingress_rule" "port-ssh" {
  security_group_id = aws_security_group.allow_tls.id
  cidr_ipv4         = "0.0.0.0/0"
  from_port         = 22
  ip_protocol       = "tcp"
  to_port           = 22
}
```

Note:→ "**For private subnet connectivity**, create a separate route table and a NAT Gateway. Route all internet-bound traffic (`0.0.0.0/0`) from the private subnet to the NAT Gateway (which resides in the public subnet). To access instances in the private subnet (e.g., DB), use a **Bastion Host** in the public subnet for secure SSH tunneling."

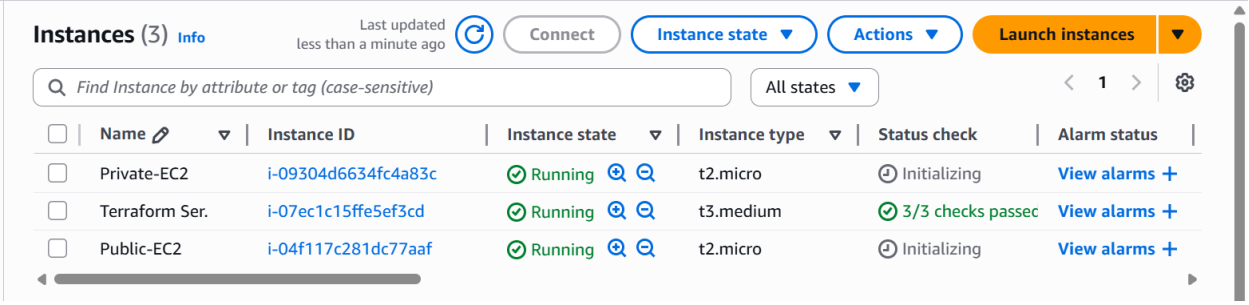
Terraform command →

```
[root@kushal-terraform aws]# terraform validate
Success! The configuration is valid.

[root@kushal-terraform aws]# terraform fmt
[root@kushal-terraform aws]# terraform plan

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

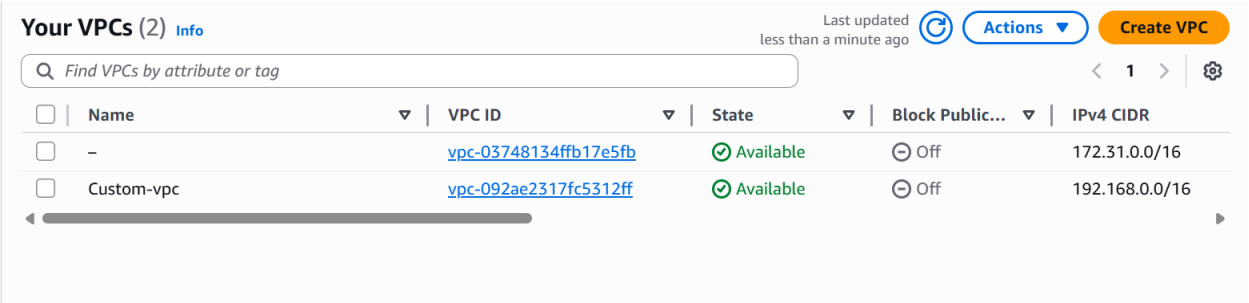
```
Plan: 18 to add, 0 to change, 0 to destroy.

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@kushal-terraform aws]#
[root@kushal-terraform aws]# terraform apply --auto-approve
```

AWS Output→

➔ Instance



➔ VPC



ThankYou!