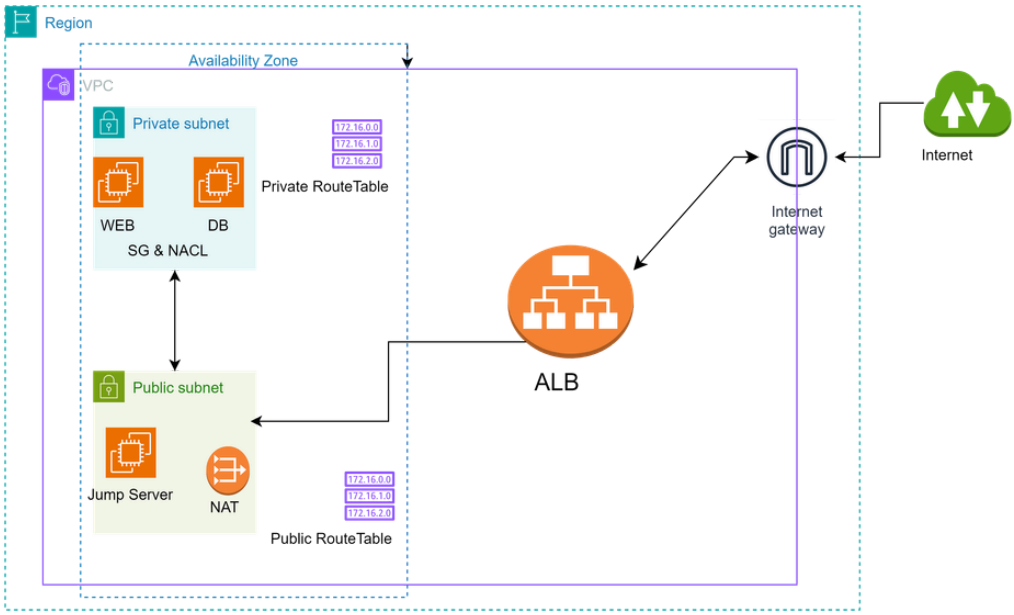
Infra Creation for Web Application

Scenario: Let's consider the following situatio

You've been given the task of setting up a private network for multiple remote machines that need secure communication. This involves creating a Virtual Private Cloud (VPC) in AWS, provisioning EC2 instances for a web application and PostgreSQL database, ensuring secure communication channels, and automating configuration using Terraform.

First, let's explore how we can create manually

Architecture Diagram:



Step 1: Set Up a Private Network

Create new vpc with following details

Name : Task -VPC

IPv4 CIDR block: 10.0.0.0/16

Tenancy : Default

Create Subnets

Create two subnets within the VPC:

1. Subnet 1: WebSubnet
   - IPv4 CIDR block: 10.0.1.0/24
   - Availability Zone: us-east-1a
2. Subnet 2: DBSubnet
   - IPv4 CIDR block: 10.0.2.0/24
   - Availability Zone: us-east-1a

Configure Route Tables

1. Create a route table and associate it with both subnets:
   - Name: PrivateRouteTable
   - Route Table Associations: WebSubnet, DBSubnet

Step 2: Provision Remote Machines

Let's start by launching two EC2 instances within the VPC:

- For Instance 1 (Web Application Server):
  - AMI: Amazon Linux 2
  - Instance Type: t2.micro
  - Subnet: WebSubnet
- For Instance 2 (PostgreSQL Database Server):
  - AMI: Amazon Linux 2
  - Instance Type: t2.micro
  - Subnet: DBSubnet

Step 3: Application and Database Configuration

Install and Configure the Web Application

1. Connect to the Web Application Server via SSH.
2. Install necessary packages (e.g., Node.js)
3. We need to install necessary packages ,dependeices to start the application

```
sudo yum update -y

sudo yum install -y nodejs npm
```

Install and Configure PostgreSQL

1. Connect to the PostgreSQL Server via SSH.
2. Installing and configuring postgreSQL for DB instance .

```
sudo yum update -y
sudo amazon-linux-extras install postgresql10 -y
sudo yum install postgresql-server -y
sudo postgresql-setup initdb
sudo systemctl start postgresql
sudo systemctl enable postgresql
```

make changes Ensure Web Application Can Connect to PostgreSQL .

Step 4: Secure Communication (SECURITY GROUPS)

Create security groups for db and web instances arrange allowing incomming and outgoing traffic like port 80 (HTTP) ,22 (SSH) and 5432 (PostgreSQL) make arrangements in inbound and outbound rules .

Set Up a Bastion Host(Jump server):

setup a jumpserver by launching new instance in public subnet and Configure the jump server to allow SSH access to the private instances.

SETP 5: Automate the Setup with Terraform:

Now we are going automate the entire set up of above components using terraform that covers VPC, subnets, security groups, route tables and EC2 instances.

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

# VPC Creation
resource "aws_vpc" "task_vpc" {
  cidr_block = "10.0.0.0/16"
  tags = {
    Name = "TASK-VPC"
  }
}

# Public Subnet for the Jump Server
resource "aws_subnet" "public_subnet" {
  vpc_id                  = aws_vpc.task_vpc.id
  cidr_block              = "10.0.1.0/24"
  availability_zone       = "us-east-1a"
  map_public_ip_on_launch = true
  tags = {
    Name = "PublicSubnet"
  }
}

# Private Subnet for Web and DB Servers
resource "aws_subnet" "private_subnet" {
  vpc_id            = aws_vpc.task_vpc.id
  cidr_block        = "10.0.2.0/24"
  availability_zone = "us-east-1a"
  tags = {
    Name = "PrivateSubnet"
  }
}

# Internet Gateway
resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.task_vpc.id
  tags = {
    Name = "InternetGateway"
  }
}

# Public Route Table
resource "aws_route_table" "public_rt" {
  vpc_id = aws_vpc.task_vpc.id

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

# Associate Public Route Table with Public Subnet
resource "aws_route_table_association" "public_association" {
  subnet_id      = aws_subnet.public_subnet.id
  route_table_id = aws_route_table.public_rt.id
```

```
}

# Security Group for Web and DB Servers (Private Subnet)
resource "aws_security_group" "private_sg" {
  vpc_id = aws_vpc.task_vpc.id

  ingress {
    from_port   = 5432
    to_port     = 5432
    protocol    = "tcp"
    cidr_blocks = ["10.0.2.0/24"]
  }

  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["10.0.1.0/24"]
  }

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

  tags = {
    Name = "PrivateSG"
  }
}

# Security Group for Jump Server (Public Subnet)
resource "aws_security_group" "public_sg" {
  vpc_id = aws_vpc.task_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 = "PublicSG"
  }
}

# EC2 Instance for Web Server
resource "aws_instance" "web" {
  ami             = "ami-0c55b159cbfafe1f0"  # Amazon Linux 2 AMI
```

```
  instance_type   = "t2.micro"
  subnet_id       = aws_subnet.private_subnet.id
  security_groups = [aws_security_group.private_sg.name]

  tags = {
    Name = "WebServer"
  }
}

# EC2 Instance for DB Server
resource "aws_instance" "db" {
  ami             = "ami-0c55b159cbfafe1f0"
  instance_type   = "t2.micro"
  subnet_id       = aws_subnet.private_subnet.id
  security_groups = [aws_security_group.private_sg.name]

  tags = {
    Name = "DBServer"
  }
}

# EC2 Instance for Jump Server
resource "aws_instance" "jump_server" {
  ami             = "ami-0c55b159cbfafe1f0"
  instance_type   = "t2.micro"
  subnet_id       = aws_subnet.public_subnet.id
  security_groups = [aws_security_group.public_sg.name]

  tags = {
    Name = "JumpServer"
  }
}

```

- Once infra was created we need to make our arrangements to deploy our application make sure it is accessible for end user with high secruity and configuring installing dependencies and necessary packages