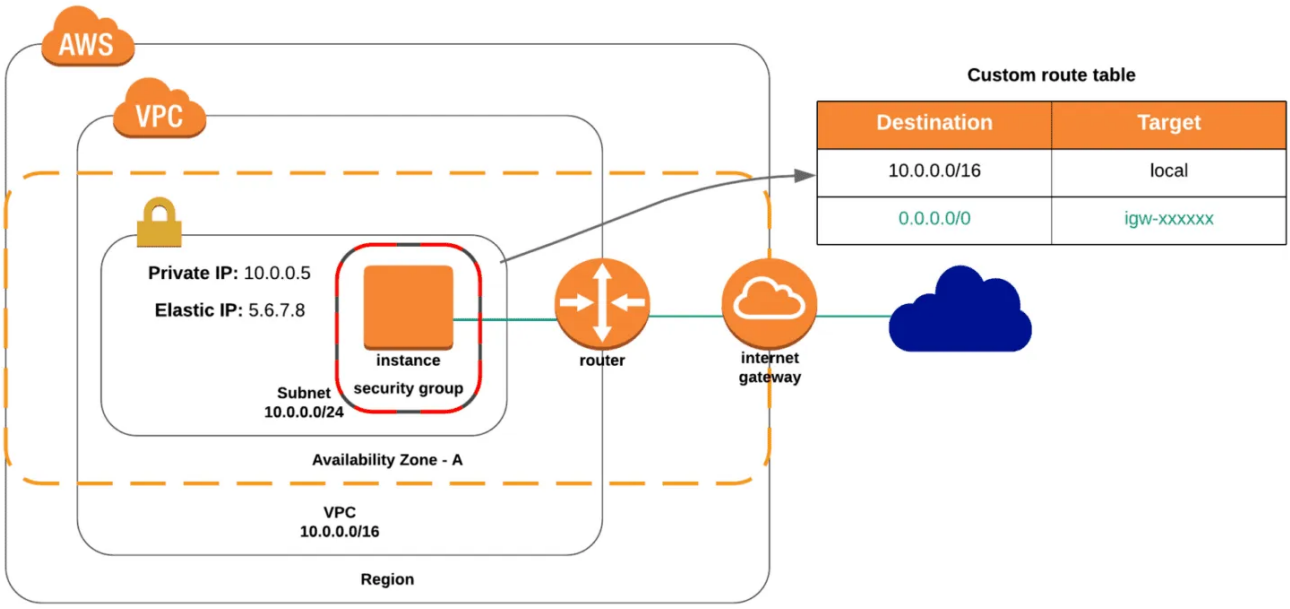
TERRAFROM – Managing a VPC With Public Subnets

Table of contents



One of the essential tasks of your cloud infrastructure management is managing your VPC's networks. In this article, we'll learn how to declare and manage the most commonly used configurations using Terraform: VPC and Public subnets.

In the Terraform recipe – Managing AWS VPC – Creating Private Subnets

VPC with a single public subnet

Configuration for this scenario includes a virtual private cloud (VPC) with a single public subnet, Internet Gateway, and Route Table to enable communication over the Internet. AWS recommends this configuration if you need to run a single-tier, public-facing web application, such as a blog or a simple website.

Creating VPC

First of all, you need to create a new terraform file with any name and `.tf` extension. I'll be using `vpc_with_single_public_subnet.tf`.

Next, we need to declare aws_vpc resource which will represent a new VPC with `10.0.0.0/16` address space:

```
resource "aws_vpc" "my_vpc" {
  cidr_block       = "10.0.0.0/16"
  enable_dns_hostnames = true

  tags = {
    Name = "My VPC"
  }
}
```

We're also enabling DNS support inside our VPC (`enable_dns_hostnames`) and setting the `Name` tag to `My VPC`, so we could easily find our VPC in the AWS console later need to.

Creating Public Subnet

As soon as the VPC resource is declared, we're ready to declare aws_subnet resource, which will describe our Public Subnet.

```
resource "aws_subnet" "public" {
  vpc_id     = aws_vpc.my_vpc.id
  cidr_block = "10.0.0.0/24"
  availability_zone = "us-east-1a"

  tags = {
    Name = "Public Subnet"
  }
}
```

Here we're asking Terraform to create our Subnet in a VPC by referring: `vpc_id` value is taken from `aws_vpc` resource declaration with name `my_vpc` by its `id`.

We're also specifying the Subnet address space within VPC by setting up a `cidr_block` option to `10.0.0.0/24` value.

Each subnet in a VPC belongs to one of the available AWS Availability Zones within AWS Regions. So, we're also specifying it by setting the `availability_zone` option to `us-east-1a` value.

Creating Internet Gateway

We call Subnets Public because they have an available route (`0.0.0.0/0`) in their Route Table attached to VPC Internet Gateway.

So, let's create an Internet Gateway now by specifying aws_internet_gateway resource:

```
resource "aws_internet_gateway" "my_vpc_igw" {
  vpc_id = aws_vpc.my_vpc.id

  tags = {
    Name = "My VPC - Internet Gateway"
  }
}
```

This entity attached to a VPC will allow Internet traffic flow to the Public Subnet.

Creating Route Table

As we already discussed, we also need to create a Route Table to route the outside world and map it to our Internet Gateway. Let's do it by declaring aws_route_table and aws_route_table_association resources:

```
resource "aws_route_table" "my_vpc_us_east_1a_public" {
    vpc_id = aws_vpc.my_vpc.id

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

    tags = {
        Name = "Public Subnet Route Table."
    }
}

resource "aws_route_table_association" "my_vpc_us_east_1a_public" {
    subnet_id = aws_subnet.public.id
    route_table_id = aws_route_table.my_vpc_us_east_1a_public.id
}
```

Here we just declared Route Table for our Subnet and made an association between them.

Our Public Subnet is ready to launch new instances inside of it. Let's do it now.

Creating Security Group

One of AWS's security features is the Security Group – it is a stateful firewall rule that allows inbound traffic to the network object. In our case, we'll use it to block any outside connections to our instance except SSH.

Let's add Security Group by adding aws_security_group resource to our `.tf` file:

```
resource "aws_security_group" "allow_ssh" {
  name        = "allow_ssh_sg"
  description = "Allow SSH inbound connections"
  vpc_id = aws_vpc.my_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 = "allow_ssh_sg"
  }
}
```

Here we're allowing incoming SSH connections (`22/tcp`) from any addresses (`0.0.0.0/0`) inside the Security Group, and also we're allowing any connection initiation to the outside world from the Security Group. So, we'll be able to SSH to the instance protected by this Security Group and make any connections from it.

Creating EC2 Instance

It's time to create our instance to test everything. Let's declare aws_instance resource:

```
resource "aws_instance" "my_instance" {
  ami           = "ami-0ac019f4fcb7cb7e6"
  instance_type = "t2.micro"
  key_name = "Lenovo T410"
  vpc_security_group_ids = [ aws_security_group.allow_ssh.id ]
  subnet_id = aws_subnet.public.id
  associate_public_ip_address = true

  tags = {
    Name = "My Instance"
  }
}
```

To allow connection from the outside world, we also asked AWS to attach a temporary Public IP address to our instance by setting the `associate_public_ip_address` option to `true`.

And the last thing we need to add to our `.tf` file is the output resource, which will print us our instance Public IP address:

```
output "instance_public_ip" {
  value = "${aws_instance.my_instance.public_ip}"
}
```

Deploying infrastructure

To apply this configuration, all you need to do is to go to the project folder and run the following commands:

```
terraform init
terraform apply
```

Connect To EC2 Instance Using SSH

At the end of the infrastructure creation process, Terraform printed you a Public IP address of your instance. To SSH to it, you need to run the following command:

```
ssh ubuntu@public_host_ip
```

Tiering down infrastructure

To remove all created resources, all you need to do is to go to the project folder and run the following command:

```
terraform destroy
```

TF file

```
# declare a VPC
resource "aws_vpc" "my_vpc" {
  cidr_block       = "10.0.0.0/16"
  enable_dns_hostnames = true

  tags = {
    Name = "My VPC"
  }
}

resource "aws_subnet" "public" {
  vpc_id     = aws_vpc.my_vpc.id
  cidr_block = "10.0.0.0/24"
  availability_zone = "us-east-1a"
```

```
  tags = {
    Name = "Public Subnet"
  }
}

resource "aws_internet_gateway" "my_vpc_igw" {
  vpc_id = aws_vpc.my_vpc.id

  tags = {
    Name = "My VPC - Internet Gateway"
  }
}

resource "aws_route_table" "my_vpc_us_east_1a_public" {
    vpc_id = aws_vpc.my_vpc.id

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

    tags = {
        Name = "Public Subnet Route Table"
    }
}

resource "aws_route_table_association" "my_vpc_us_east_1a_public" {
    subnet_id = aws_subnet.public.id
    route_table_id = aws_route_table.my_vpc_us_east_1a_public.id
}

resource "aws_security_group" "allow_ssh" {
  name        = "allow_ssh_sg"
  description = "Allow SSH inbound connections"
  vpc_id = aws_vpc.my_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 = "allow_ssh_sg"
  }
}

resource "aws_instance" "my_instance" {
  ami           = "ami-0ac019f4fcb7cb7e6"
  instance_type = "t2.micro"
  key_name = "Lenovo T410"
  vpc_security_group_ids = [ aws_security_group.allow_ssh.id ]
  subnet_id = aws_subnet.public.id
  associate_public_ip_address = true

  tags = {
    Name = "My Instance"
  }
}

output "instance_public_ip" {
  value = "${aws_instance.my_instance.public_ip}"
}
```