Deploy a 3-tier architecture in AWS using Terraform

Scenario: A 3–tier architecture on AWS

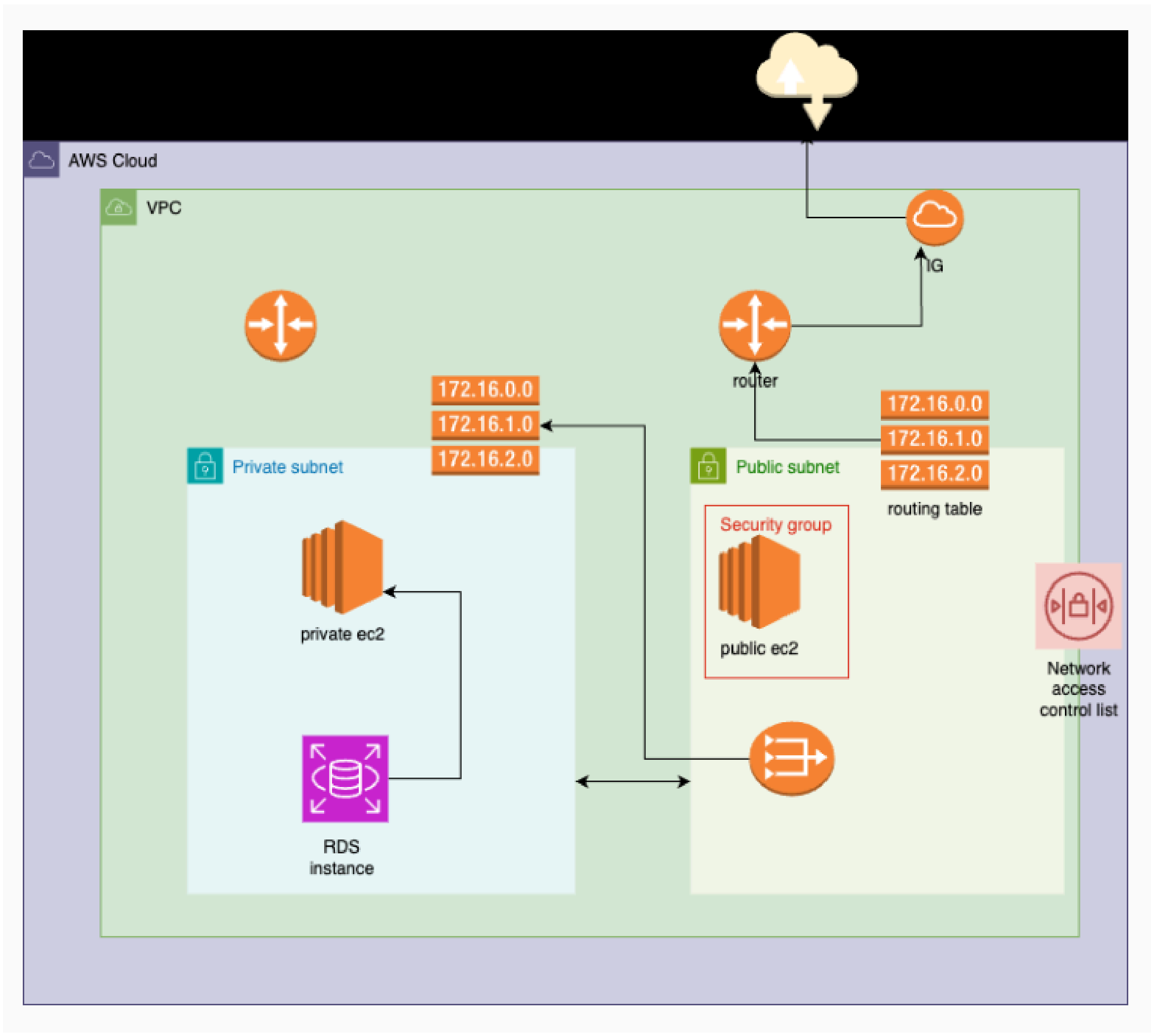
Creating a three–tier architecture on AWS involves setting up three layers: a Web layer, an Application layer, and a Database layer.

We’ll set up two EC2 instances: one in a public subnet with a public IP for internet access, and another in a private subnet without direct internet connectivity.

To enable software updates for the private instance, we’ll implement a NAT gateway in the public subnet with an Elastic IP. We’ll configure routes to allow the private instance to communicate through the NAT gateway.

Additionally, we’ll create an RDS instance running Postgres in the private subnet, accessible only from the private EC2 instance. This setup ensures a secure environment with controlled internet access and database isolation.

Architectural Diagram



Step 1: Set Up Your Networking

- Create a new VPC with CIDR block(10.0.0.0/16)
- Create a private and public subnet and respective routing tables.
- Create an Internet Gateway and associate it with VPC. Create a public route to allow internet access to your public subnet.
- Create a NAT gateway in your public subnet and a private route allowing outbound internet access to your private subnet via the public subnet.

Step 2: Set Up the Database Layer

- Create an Amazon RDS instance for your database in the private subnet.

- Configure the database security group to allow connections only from the application layer(EC2 instance on your private subnet).

Step 3: Set Up the Application and web Layer

- Create an EC2 instance in your private subnet for the application layer. This will not have a public IP, you can only access it from the web layer.
- Create an EC2 instance in your public subnet for the web layer. This VM will have a public IP, allowing connection from the internet.
- Ensure that the security group allows traffic from the Web layer and to the Database layer.

Step 4: Configure Security Groups and Network ACLs

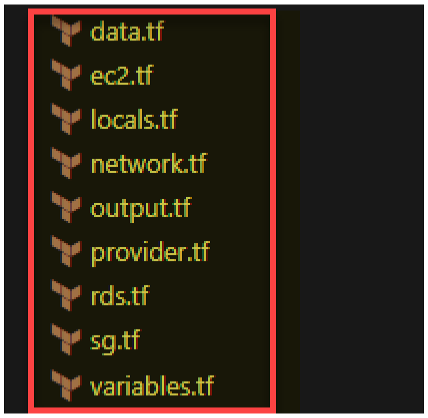
- Set up security groups to control inbound and outbound traffic at the instance level for both the application and web layer.
- Configure Network Access Control Lists (ACLs) to control inbound and outbound traffic at the subnet level.

Step 5: Testing Your Setup

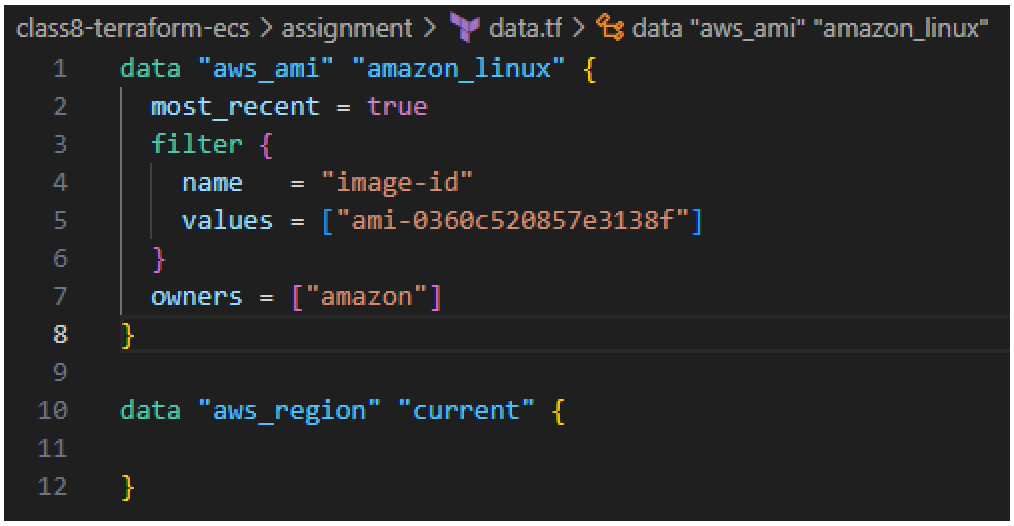
- Access your web server through its public IP address or domain name.
- Check the connectivity between the layers by trying to access the application layers from the web layer, and the database layer from the application level.

Step 6: Terraform Code

1. Directory Structure

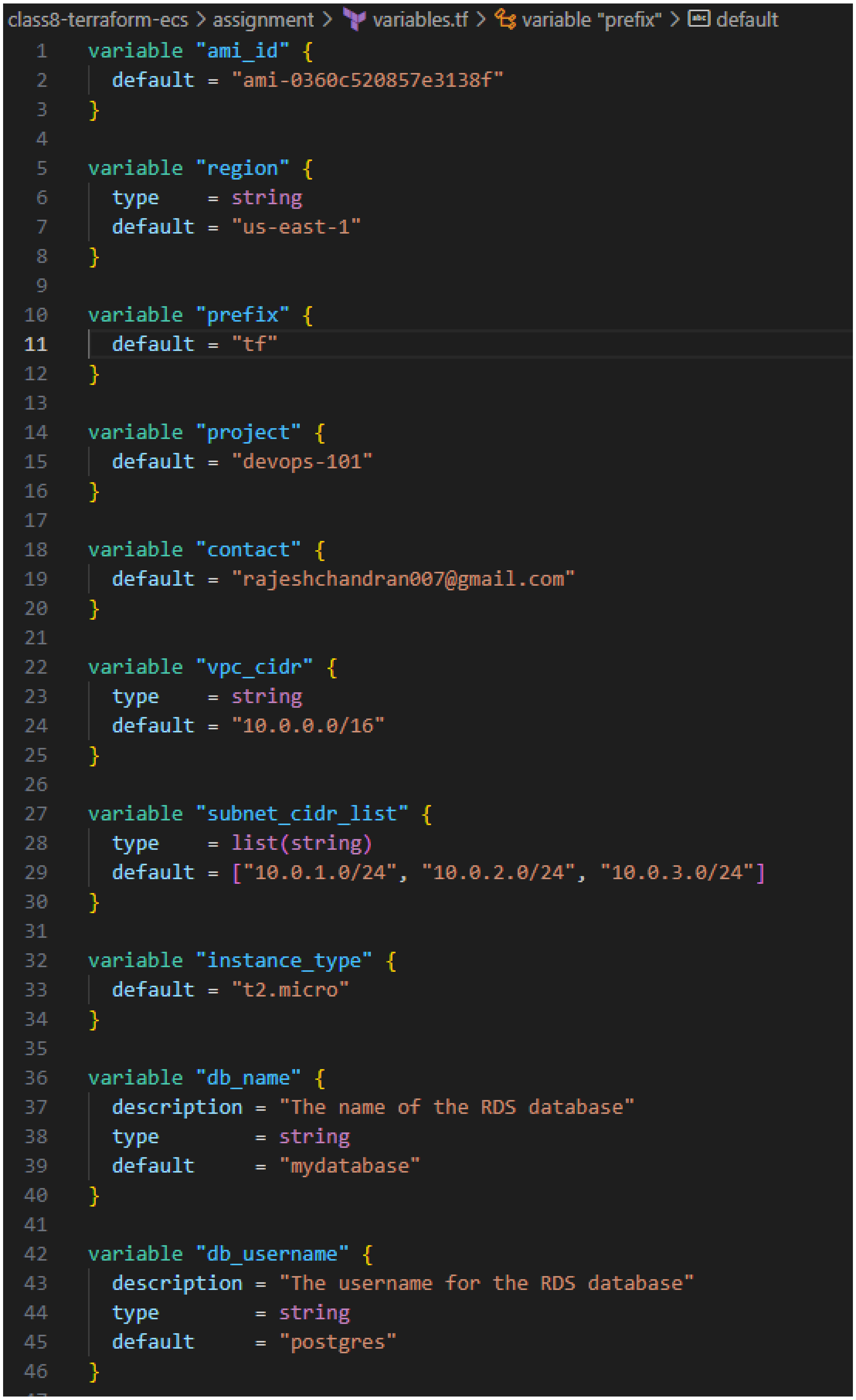

```
data.tf
ec2.tf
locals.tf
network.tf
output.tf
provider.tf
rds.tf
sg.tf
variables.tf
```

2. Data block for using the latest AMI for ubuntu system


class8-terraform-ecs > assignment > data.tf > data "aws_ami" "amazon_linux"

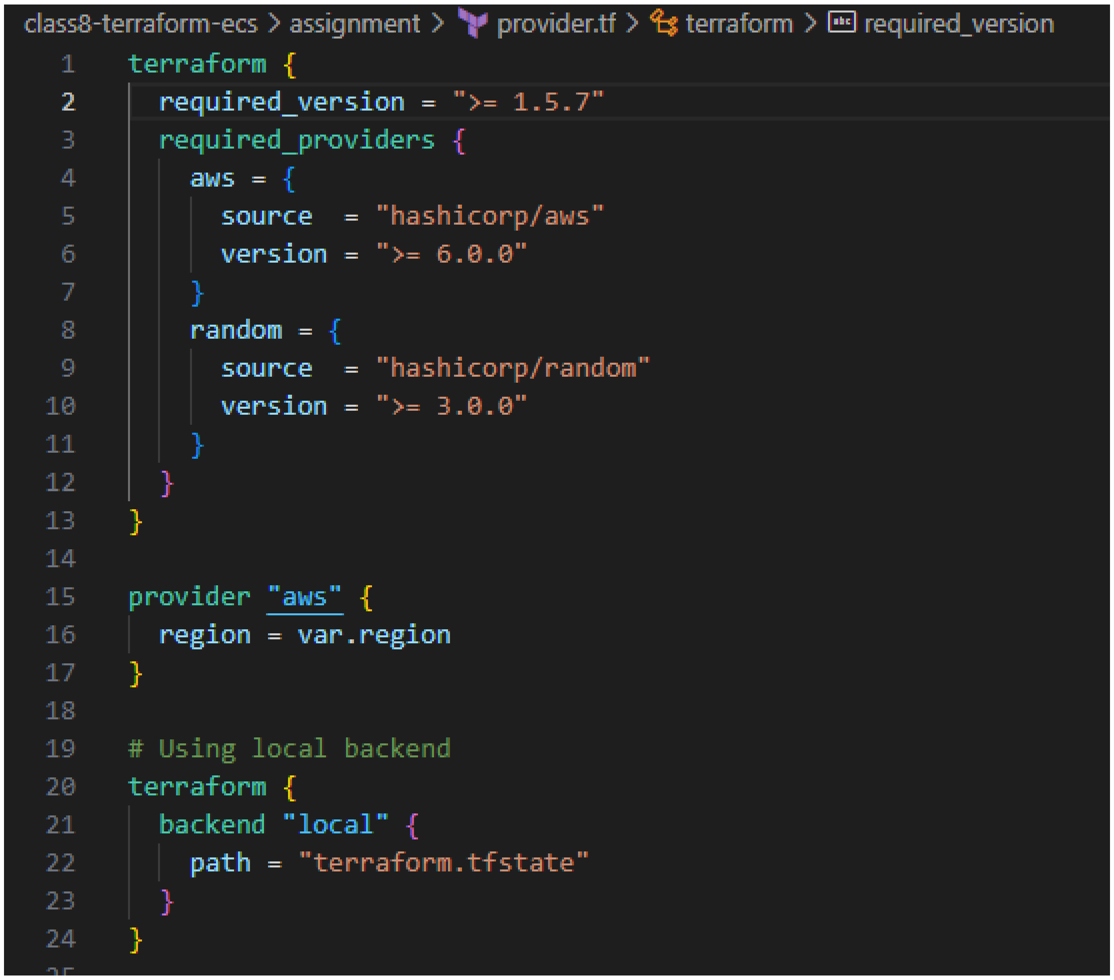
```
data "aws_ami" "amazon_linux" {
  most_recent = true
  filter {
    name   = "image-id"
    values = ["ami-0360c520857e3138f"]
  }
  owners = ["amazon"]
}

data "aws_region" "current" {

}
```

3. Variables and Local Variables


class8-terraform-ecs > assignment > variables.tf > variable "prefix" > default

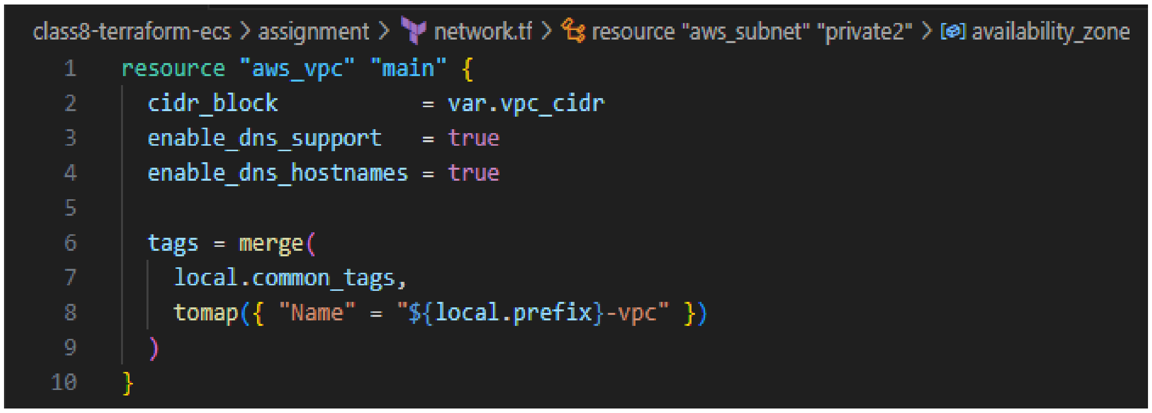
```
variable "ami_id" {
  default = "ami-0360c520857e3138f"
}

variable "region" {
  type    = string
  default = "us-east-1"
}

variable "prefix" {
  default = "tf"
}

variable "project" {
  default = "devops-101"
}

variable "contact" {
  default = "rajeshchandran007@gmail.com"
}

variable "vpc_cidr" {
  type    = string
  default = "10.0.0.0/16"
}

variable "subnet_cidr_list" {
  type    = list(string)
  default = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]
}

variable "instance_type" {
  default = "t2.micro"
}

variable "db_name" {
  description = "The name of the RDS database"
  type        = string
  default     = "mydatabase"
}

variable "db_username" {
  description = "The username for the RDS database"
  type        = string
  default     = "postgres"
}
```

4. Provider block

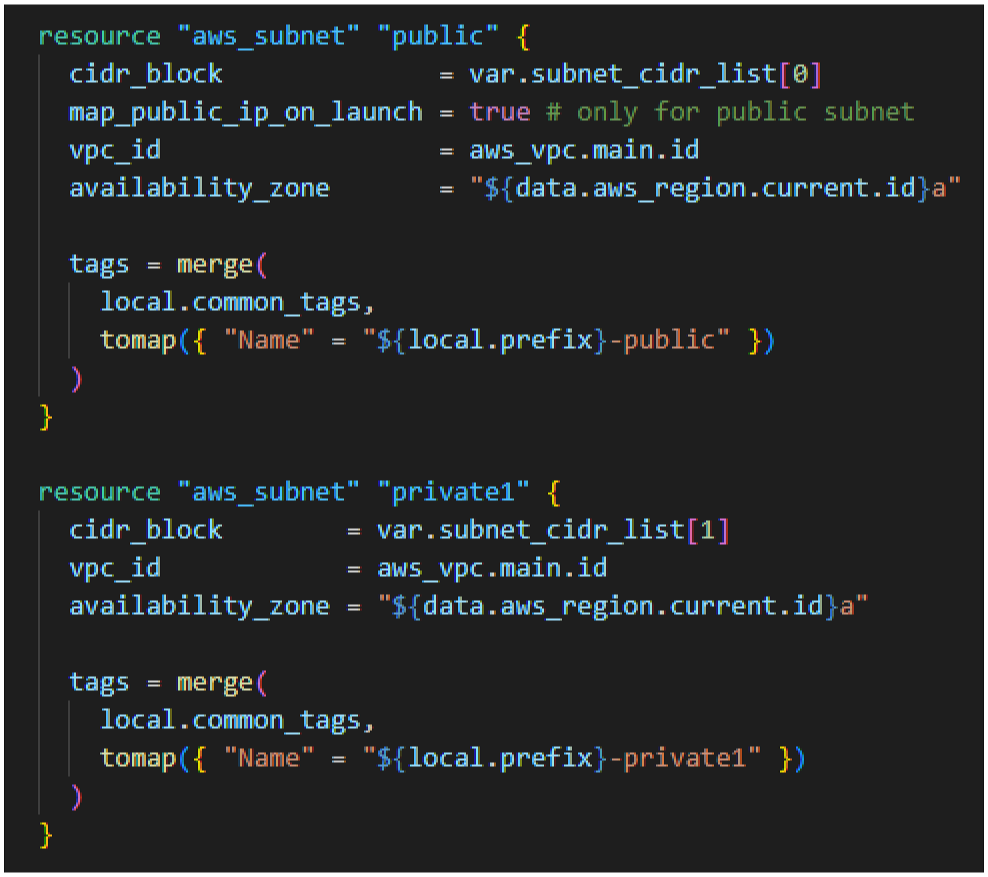

class8-terraform-ecs > assignment > provider.tf > terraform > required_version

```
terraform {
  required_version = ">= 1.5.7"
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = ">= 6.0.0"
    }
    random = {
      source  = "hashicorp/random"
      version = ">= 3.0.0"
    }
  }
}

provider "aws" {
  region = var.region
}

# Using local backend
terraform {
  backend "local" {
    path = "terraform.tfstate"
  }
}
```

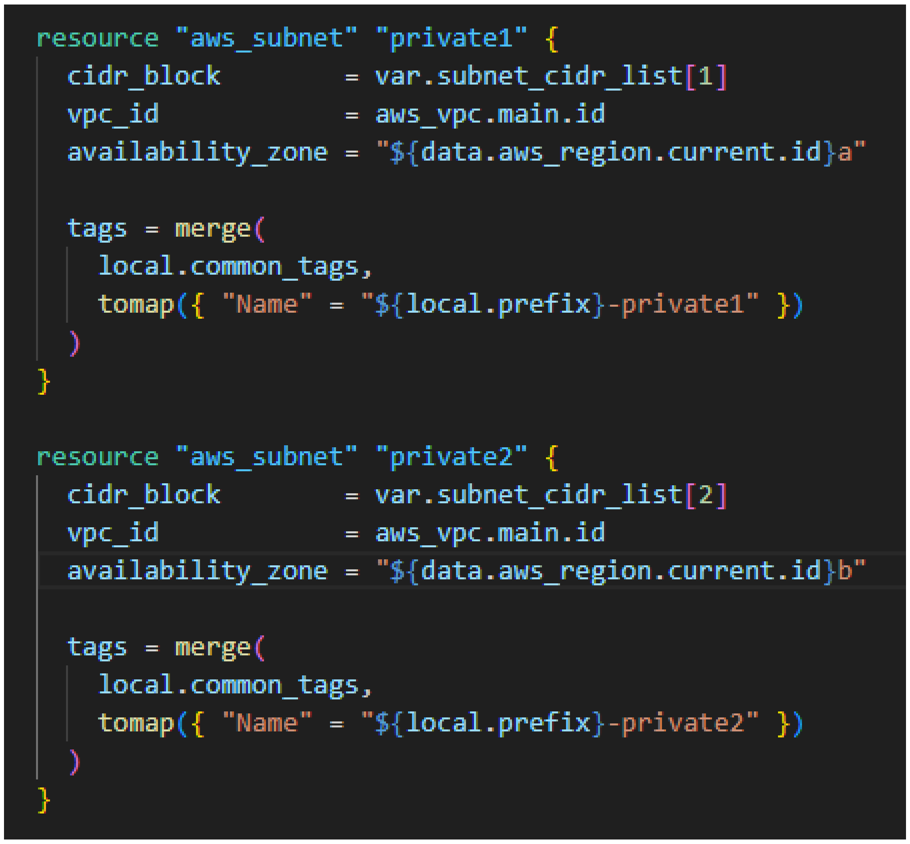
5. Network (VPC, Subnets, IGW, NAT, Route Tables)


class8-terraform-ecs > assignment > network.tf > resource "aws_subnet" "private2" > availability_zone

```
resource "aws_vpc" "main" {
  cidr_block           = var.vpc_cidr
  enable_dns_support   = true
  enable_dns_hostnames = true

  tags = merge(
    local.common_tags,
    tomap({ "Name" = "${local.prefix}-vpc" })
  )
}
```

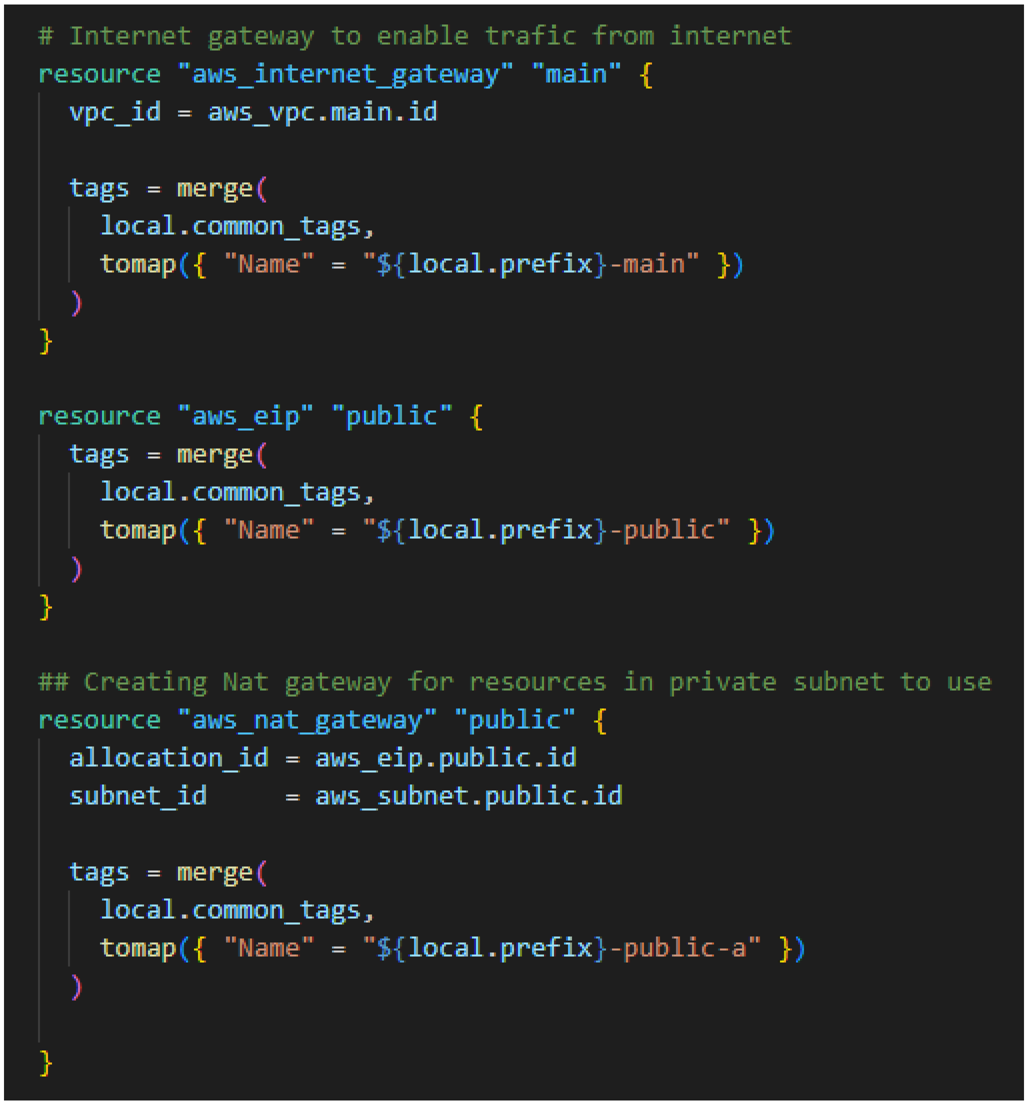


```
resource "aws_subnet" "public" {
  cidr_block              = var.subnet_cidr_list[0]
  map_public_ip_on_launch = true # only for public subnet
  vpc_id                  = aws_vpc.main.id
  availability_zone       = "${data.aws_region.current.id}a"

  tags = merge(
    local.common_tags,
    tomap({ "Name" = "${local.prefix}-public" })
  )
}

resource "aws_subnet" "private1" {
  cidr_block        = var.subnet_cidr_list[1]
  vpc_id            = aws_vpc.main.id
  availability_zone = "${data.aws_region.current.id}a"

  tags = merge(
    local.common_tags,
    tomap({ "Name" = "${local.prefix}-private1" })
  )
}
```

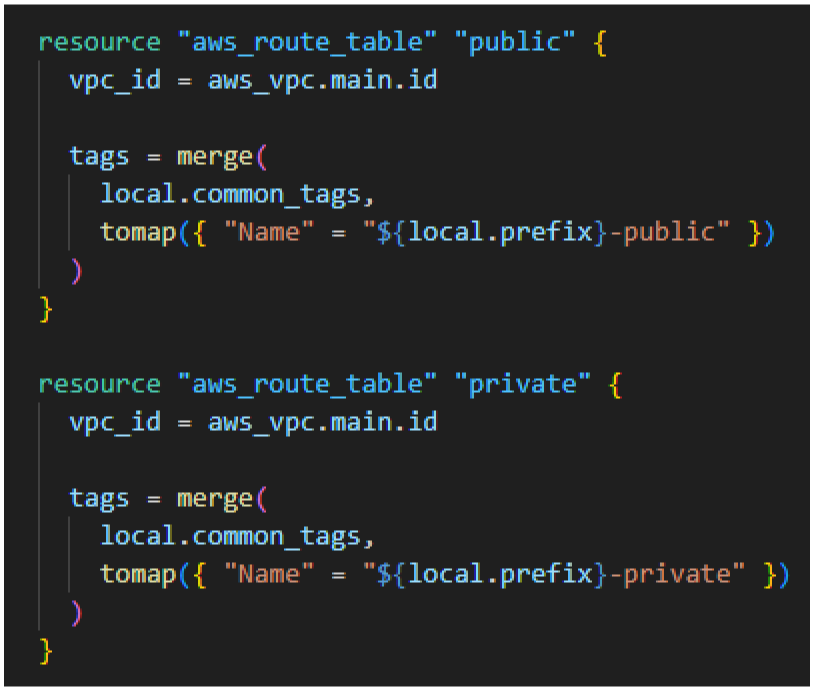


```
resource "aws_subnet" "private1" {
  cidr_block        = var.subnet_cidr_list[1]
  vpc_id            = aws_vpc.main.id
  availability_zone = "${data.aws_region.current.id}a"

  tags = merge(
    local.common_tags,
    tomap({ "Name" = "${local.prefix}-private1" })
  )
}
```

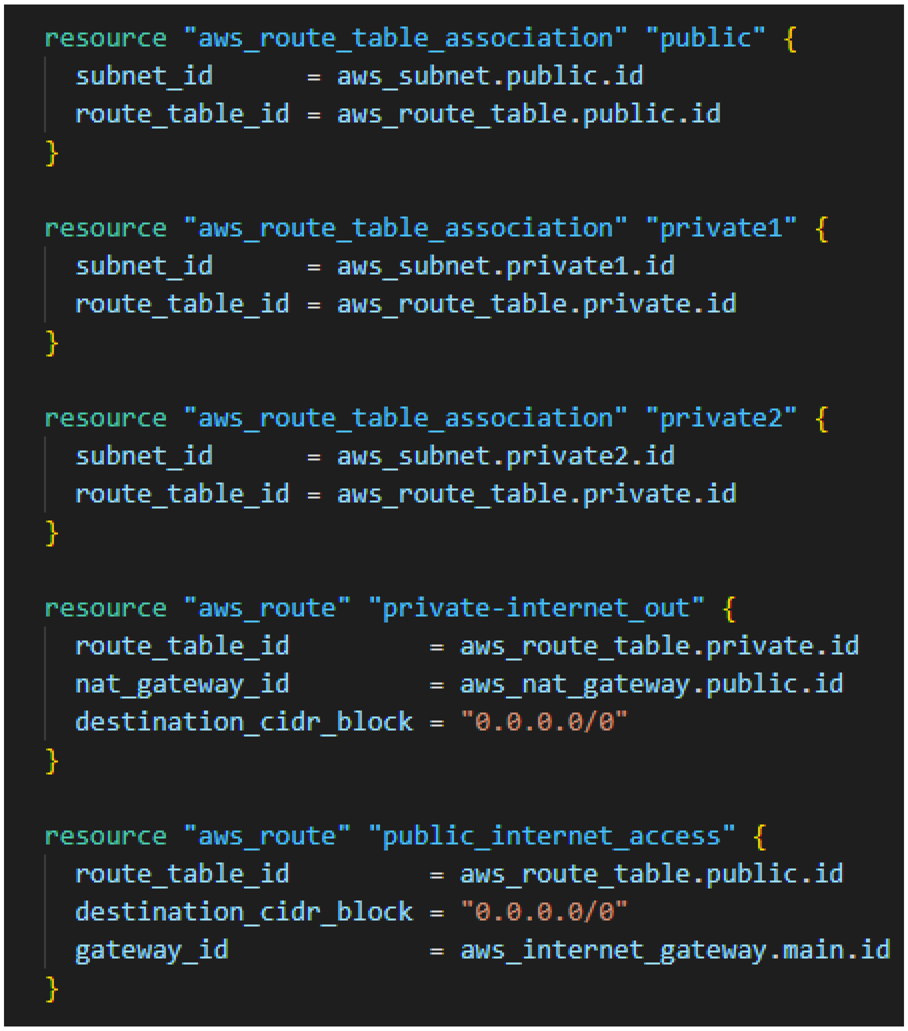
```
resource "aws_subnet" "private2" {
  cidr_block        = var.subnet_cidr_list[2]
  vpc_id            = aws_vpc.main.id
  availability_zone = "${data.aws_region.current.id}b"

  tags = merge(
    local.common_tags,
    tomap({ "Name" = "${local.prefix}-private2" })
  )
}
```



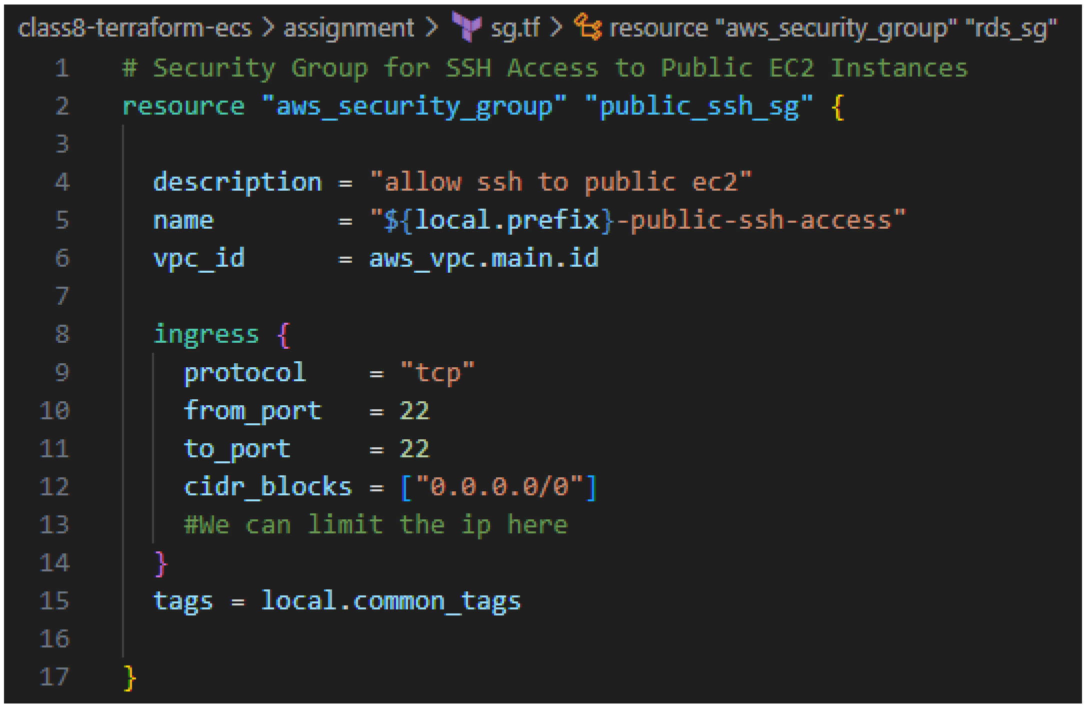
```
# Internet gateway to enable trafic from internet
resource "aws_internet_gateway" "main" {
  vpc_id = aws_vpc.main.id

  tags = merge(
    local.common_tags,
    tomap({ "Name" = "${local.prefix}-main" })
  )
}

resource "aws_eip" "public" {
  tags = merge(
    local.common_tags,
    tomap({ "Name" = "${local.prefix}-public" })
  )
}

## Creating Nat gateway for resources in private subnet to use
resource "aws_nat_gateway" "public" {
  allocation_id = aws_eip.public.id
  subnet_id     = aws_subnet.public.id

  tags = merge(
    local.common_tags,
    tomap({ "Name" = "${local.prefix}-public-a" })
  )

}
```



```
resource "aws_route_table" "public" {
  vpc_id = aws_vpc.main.id

  tags = merge(
    local.common_tags,
    tomap({ "Name" = "${local.prefix}-public" })
  )
}

resource "aws_route_table" "private" {
  vpc_id = aws_vpc.main.id

  tags = merge(
    local.common_tags,
    tomap({ "Name" = "${local.prefix}-private" })
  )
}
```

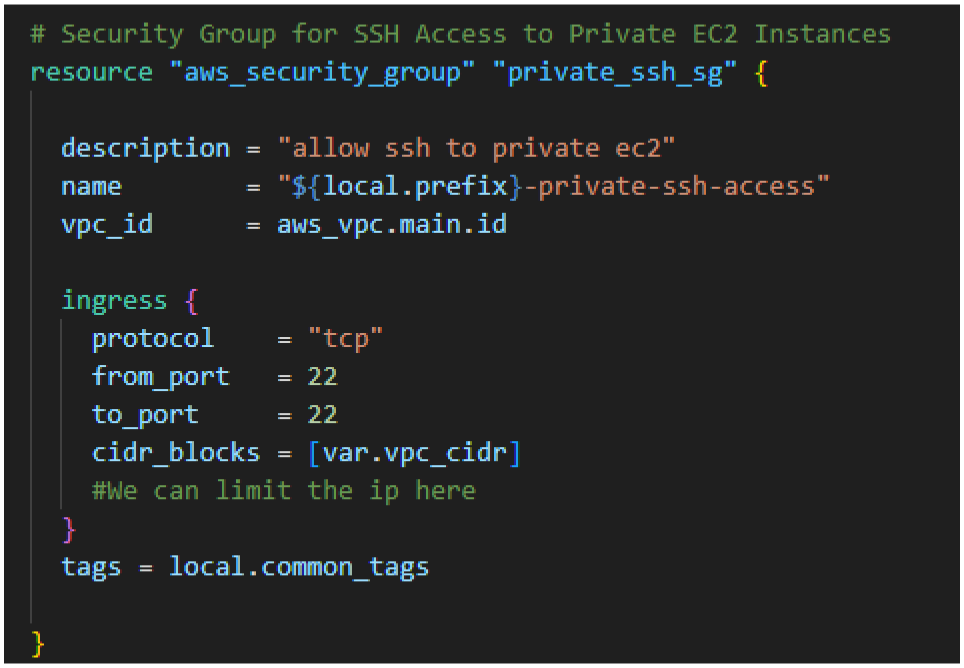

```
resource "aws_route_table_association" "public" {
  subnet_id      = aws_subnet.public.id
  route_table_id = aws_route_table.public.id
}

resource "aws_route_table_association" "private1" {
  subnet_id      = aws_subnet.private1.id
  route_table_id = aws_route_table.private.id
}

resource "aws_route_table_association" "private2" {
  subnet_id      = aws_subnet.private2.id
  route_table_id = aws_route_table.private.id
}

resource "aws_route" "private-internet_out" {
  route_table_id         = aws_route_table.private.id
  nat_gateway_id         = aws_nat_gateway.public.id
  destination_cidr_block = "0.0.0.0/0"
}

resource "aws_route" "public_internet_access" {
  route_table_id         = aws_route_table.public.id
  destination_cidr_block = "0.0.0.0/0"
  gateway_id             = aws_internet_gateway.main.id
}
```

6. Security Group

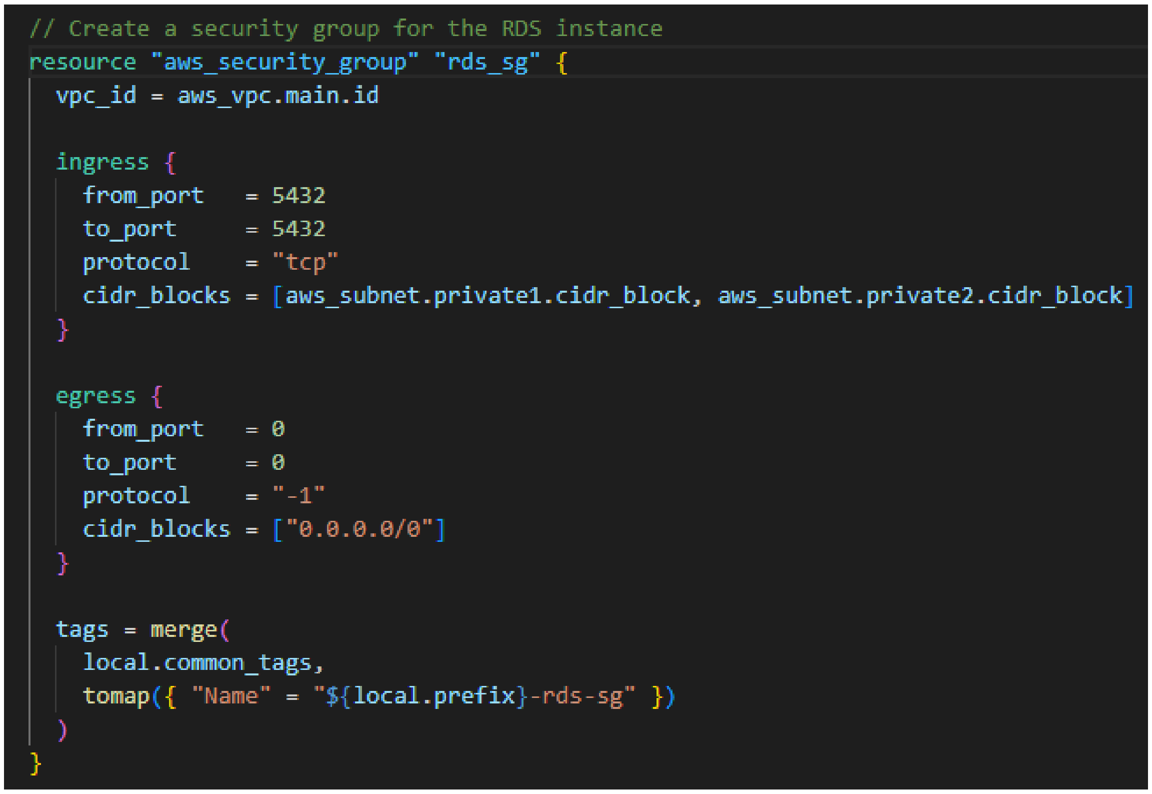


class8-terraform-ecs > assignment > sg.tf > resource "aws_security_group" "rds_sg"

```
# Security Group for SSH Access to Public EC2 Instances
resource "aws_security_group" "public_ssh_sg" {

  description = "allow ssh to public ec2"
  name        = "${local.prefix}-public-ssh-access"
  vpc_id      = aws_vpc.main.id

  ingress {
    protocol    = "tcp"
    from_port   = 22
    to_port     = 22
    cidr_blocks = ["0.0.0.0/0"]
    #We can limit the ip here
  }
  tags = local.common_tags

}
```

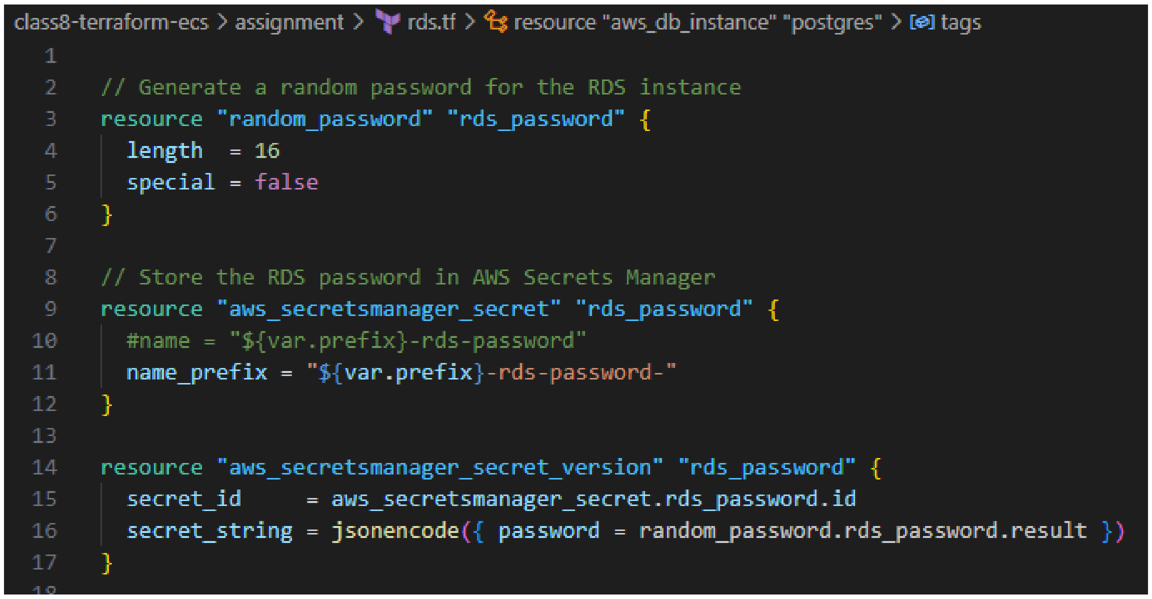


```
# Security Group for SSH Access to Private EC2 Instances
resource "aws_security_group" "private_ssh_sg" {

  description = "allow ssh to private ec2"
  name        = "${local.prefix}-private-ssh-access"
  vpc_id      = aws_vpc.main.id

  ingress {
    protocol    = "tcp"
    from_port   = 22
    to_port     = 22
    cidr_blocks = [var.vpc_cidr]
    #We can limit the ip here
  }
  tags = local.common_tags

}
```



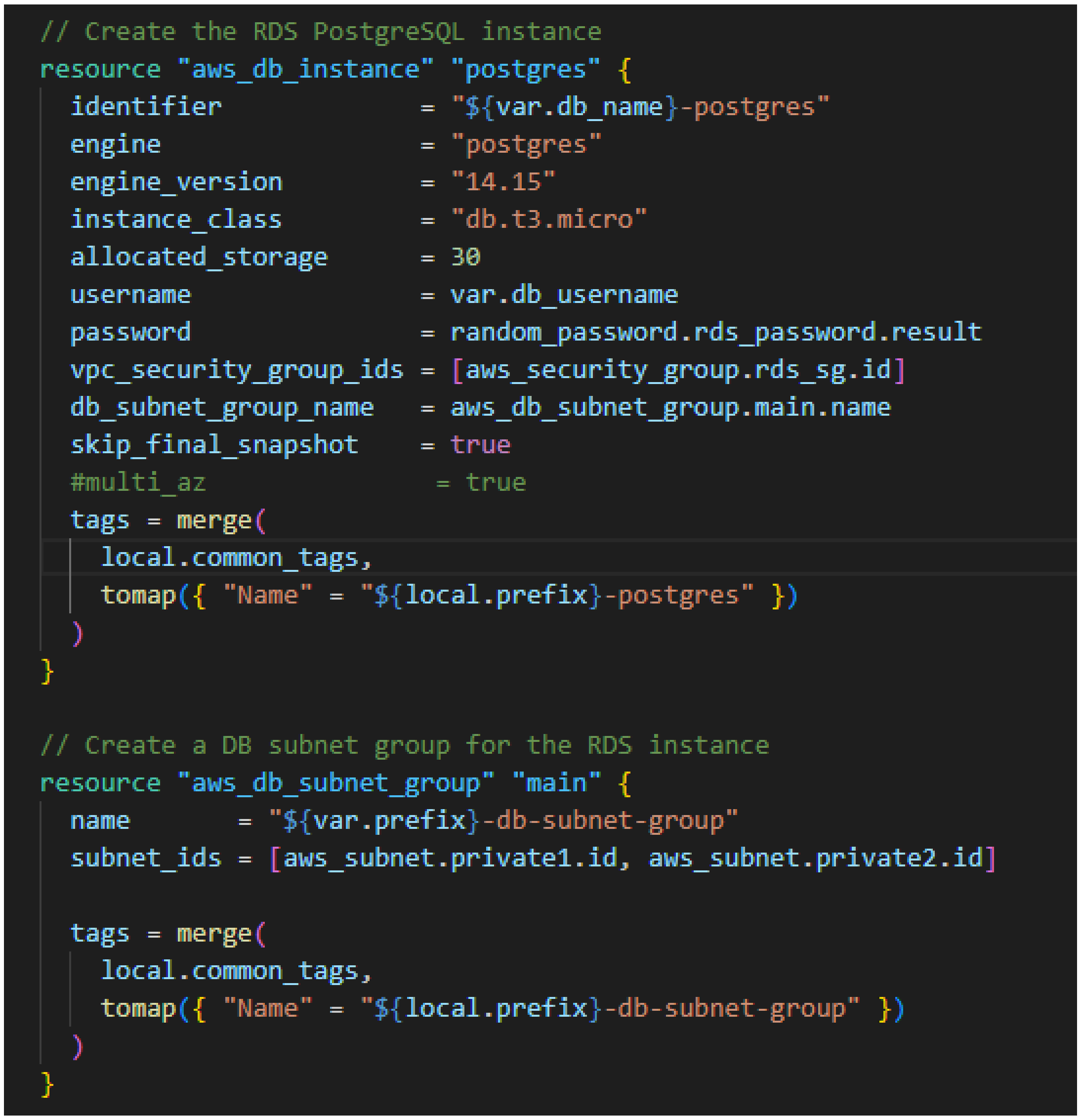
```
// Create a security group for the RDS instance
resource "aws_security_group" "rds_sg" {
  vpc_id = aws_vpc.main.id

  ingress {
    from_port   = 5432
    to_port     = 5432
    protocol    = "tcp"
    cidr_blocks = [aws_subnet.private1.cidr_block, aws_subnet.private2.cidr_block]
  }

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

  tags = merge(
    local.common_tags,
    tomap({ "Name" = "${local.prefix}-rds-sg" })
  )
}
```

7. RDS (DB Setup)



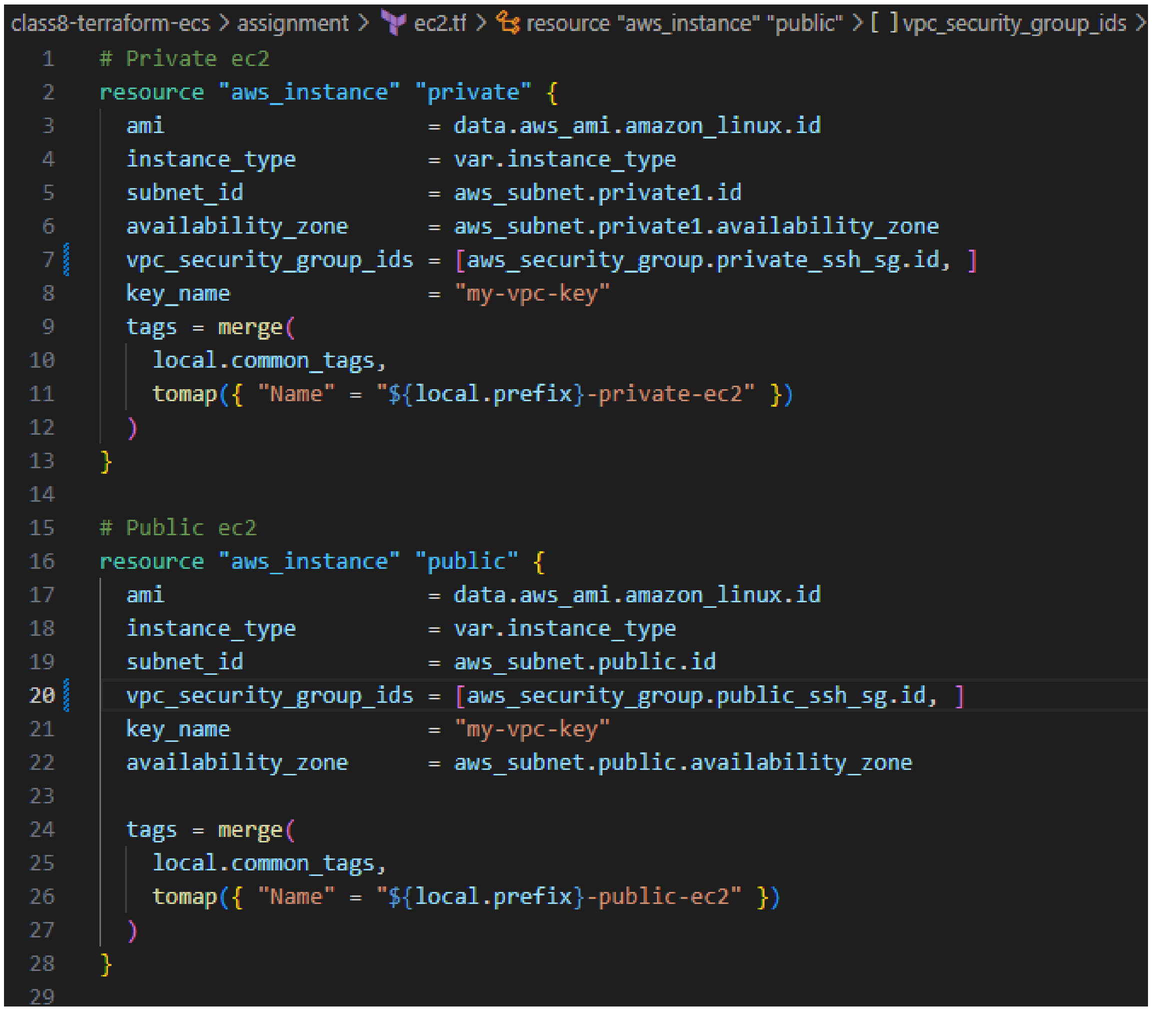
class8-terraform-ecs > assignment > rds.tf > resource "aws_db_instance" "postgres" > tags

```

// Generate a random password for the RDS instance
resource "random_password" "rds_password" {
  length  = 16
  special = false
}

// Store the RDS password in AWS Secrets Manager
resource "aws_secretsmanager_secret" "rds_password" {
  #name = "${var.prefix}-rds-password"
  name_prefix = "${var.prefix}-rds-password-"
}

resource "aws_secretsmanager_secret_version" "rds_password" {
  secret_id     = aws_secretsmanager_secret.rds_password.id
  secret_string = jsonencode({ password = random_password.rds_password.result })
}
```



```
// Create the RDS PostgreSQL instance
resource "aws_db_instance" "postgres" {
  identifier             = "${var.db_name}-postgres"
  engine                 = "postgres"
  engine_version         = "14.15"
  instance_class         = "db.t3.micro"
  allocated_storage      = 30
  username               = var.db_username
  password               = random_password.rds_password.result
  vpc_security_group_ids = [aws_security_group.rds_sg.id]
  db_subnet_group_name   = aws_db_subnet_group.main.name
  skip_final_snapshot    = true
  #multi_az               = true
  tags = merge(
    local.common_tags,
    tomap({ "Name" = "${local.prefix}-postgres" })
  )
}

// Create a DB subnet group for the RDS instance
resource "aws_db_subnet_group" "main" {
  name       = "${var.prefix}-db-subnet-group"
  subnet_ids = [aws_subnet.private1.id, aws_subnet.private2.id]

  tags = merge(
    local.common_tags,
    tomap({ "Name" = "${local.prefix}-db-subnet-group" })
  )
}
```

8. EC2 Systems



class8-terraform-ecs > assignment > ec2.tf > resource "aws_instance" "public" > [ ] vpc_security_group_ids >

```
# Private ec2
resource "aws_instance" "private" {
  ami                    = data.aws_ami.amazon_linux.id
  instance_type          = var.instance_type
  subnet_id              = aws_subnet.private1.id
  availability_zone      = aws_subnet.private1.availability_zone
  vpc_security_group_ids = [aws_security_group.private_ssh_sg.id, ]
  key_name               = "my-vpc-key"
  tags = merge(
    local.common_tags,
    tomap({ "Name" = "${local.prefix}-private-ec2" })
  )
}

# Public ec2
resource "aws_instance" "public" {
  ami                    = data.aws_ami.amazon_linux.id
  instance_type          = var.instance_type
  subnet_id              = aws_subnet.public.id
  vpc_security_group_ids = [aws_security_group.public_ssh_sg.id, ]
  key_name               = "my-vpc-key"
  availability_zone      = aws_subnet.public.availability_zone

  tags = merge(
    local.common_tags,
    tomap({ "Name" = "${local.prefix}-public-ec2" })
  )
}

```

Step 7: Terraform Execution

terraform init

- Initializes the working directory, downloads provider plugins, and sets up the backend.

terraform plan

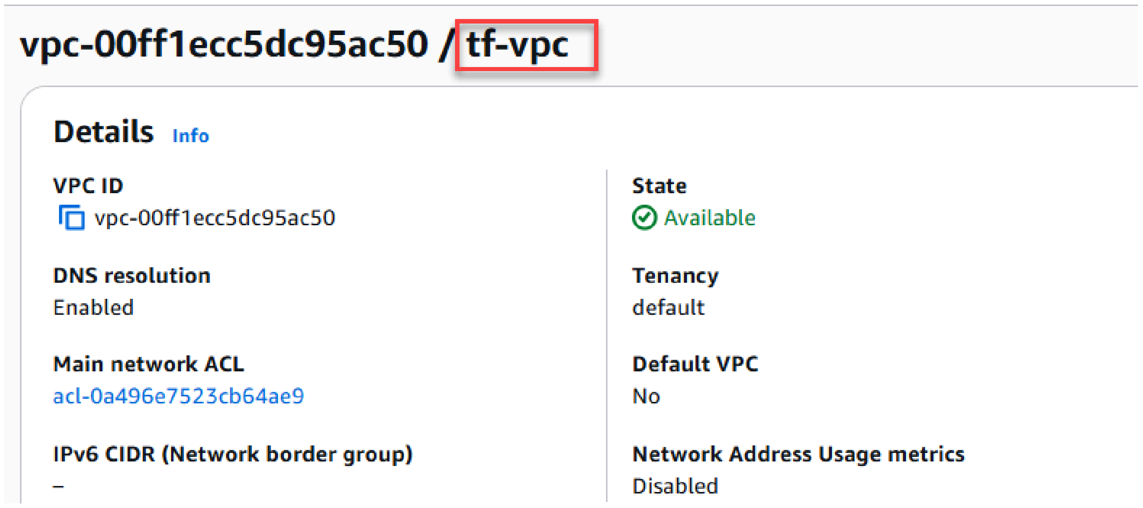
- Shows the execution plan, previewing what resources will be created, changed, or destroyed.

terraform apply –auto-approve

- Applies the changes without asking for interactive approval.

Step 8: AWS Resources Screenshots

VPC

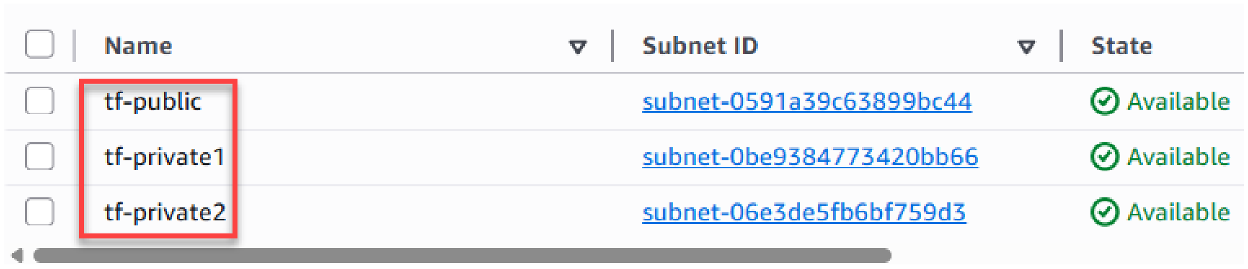


vpc-00ff1ecc5dc95ac50 / tf-vpc

Details Info

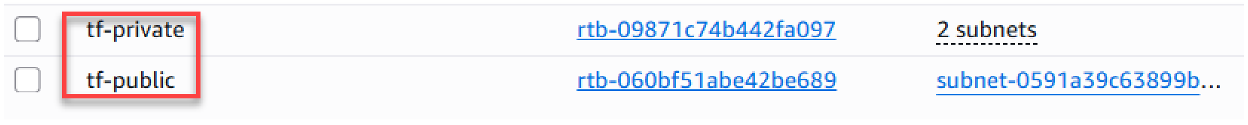
VPC ID vpc-00ff1ecc5dc95ac50	**State** Available
DNS resolution Enabled	**Tenancy** default
Main network ACL acl-0a496e7523cb64ae9	**Default VPC** No
IPv6 CIDR (Network border group) –	**Network Address Usage metrics** Disabled

Subnets



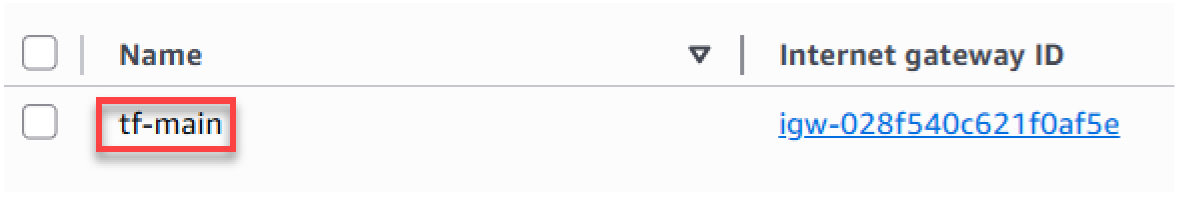
Name	Subnet ID	State
tf-public	subnet-0591a39c63899bc44	Available
tf-private1	subnet-0be9384773420bb66	Available
tf-private2	subnet-06e3de5fb6bf759d3	Available

Route Tables



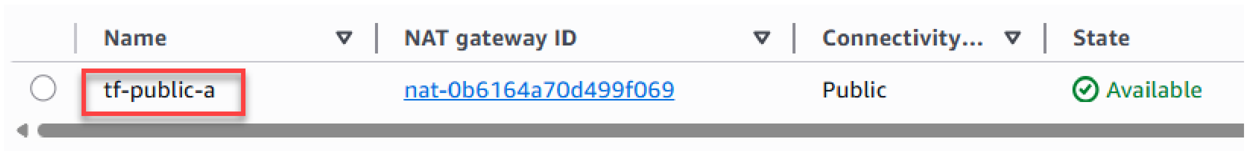
tf-private	rtb-09871c74b442fa097	2 subnets
tf-public	rtb-060bf51abe42be689	subnet-0591a39c63899b...

IGW



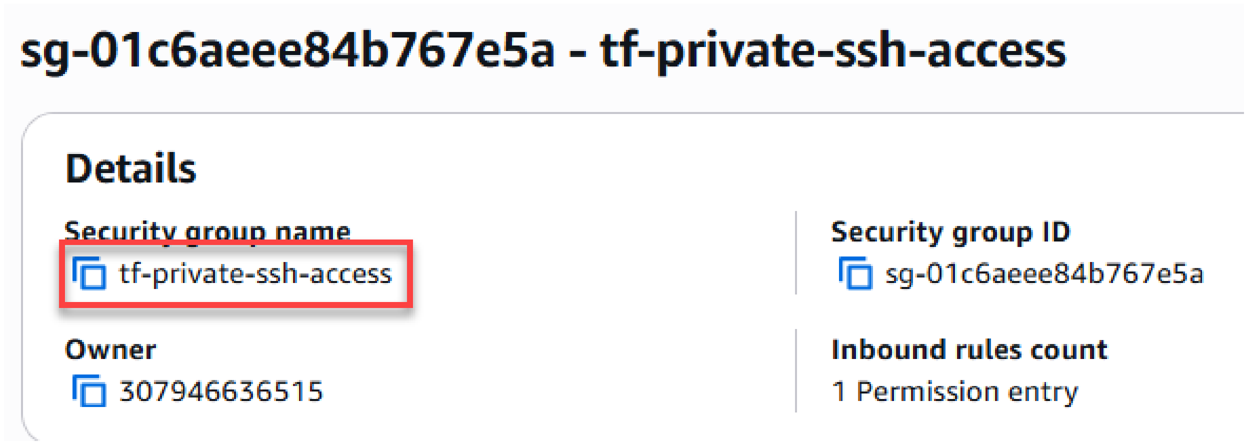
Name	Internet gateway ID
tf-main	igw-028f540c621f0af5e

NAT



	Name	NAT gateway ID	Connectivity...	State
○	tf-public-a	nat-0b6164a70d499f069	Public	✓ Available

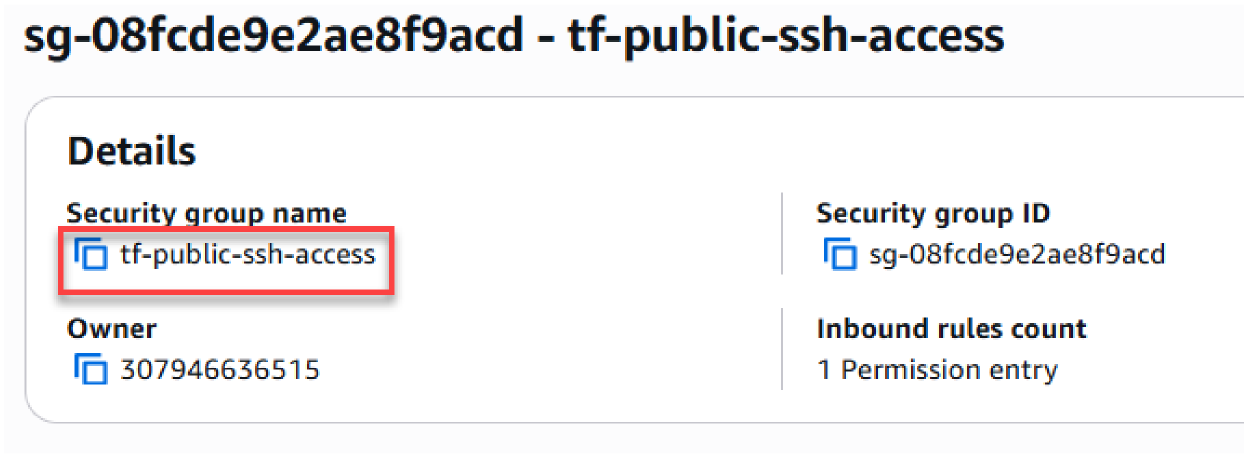
Security Groups (Private EC2, Public EC2, RDS)



sg-01c6aeee84b767e5a - tf-private-ssh-access

Details

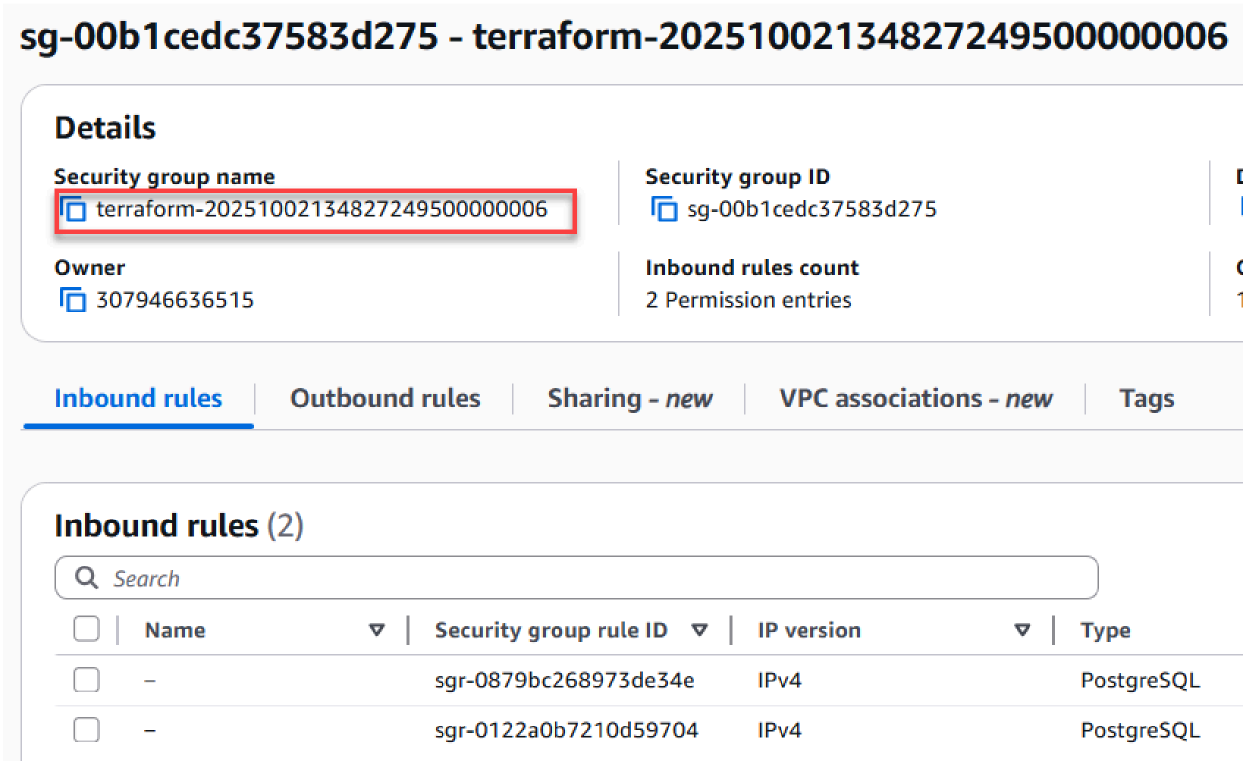
Security group name	Security group ID
tf-private-ssh-access	sg-01c6aeee84b767e5a
Owner	Inbound rules count
307946636515	1 Permission entry



sg-08fcde9e2ae8f9acd - tf-public-ssh-access

Details

Security group name	Security group ID
tf-public-ssh-access	sg-08fcde9e2ae8f9acd
Owner	Inbound rules count
307946636515	1 Permission entry


sg-00b1cedc37583d275 - terraform-20251002134827249500000006

Details

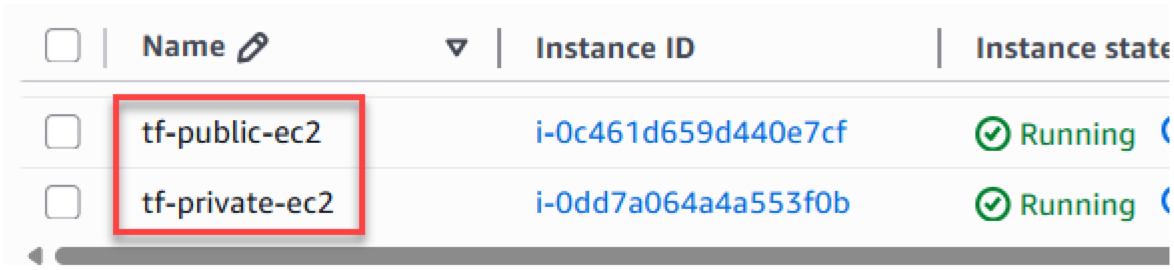
Security group name	Security group ID
terraform-20251002134827249500000006	sg-00b1cedc37583d275
Owner	**Inbound rules count**
307946636515	2 Permission entries

Inbound rules | Outbound rules | Sharing - *new* | VPC associations - *new* | Tags

Inbound rules (2)

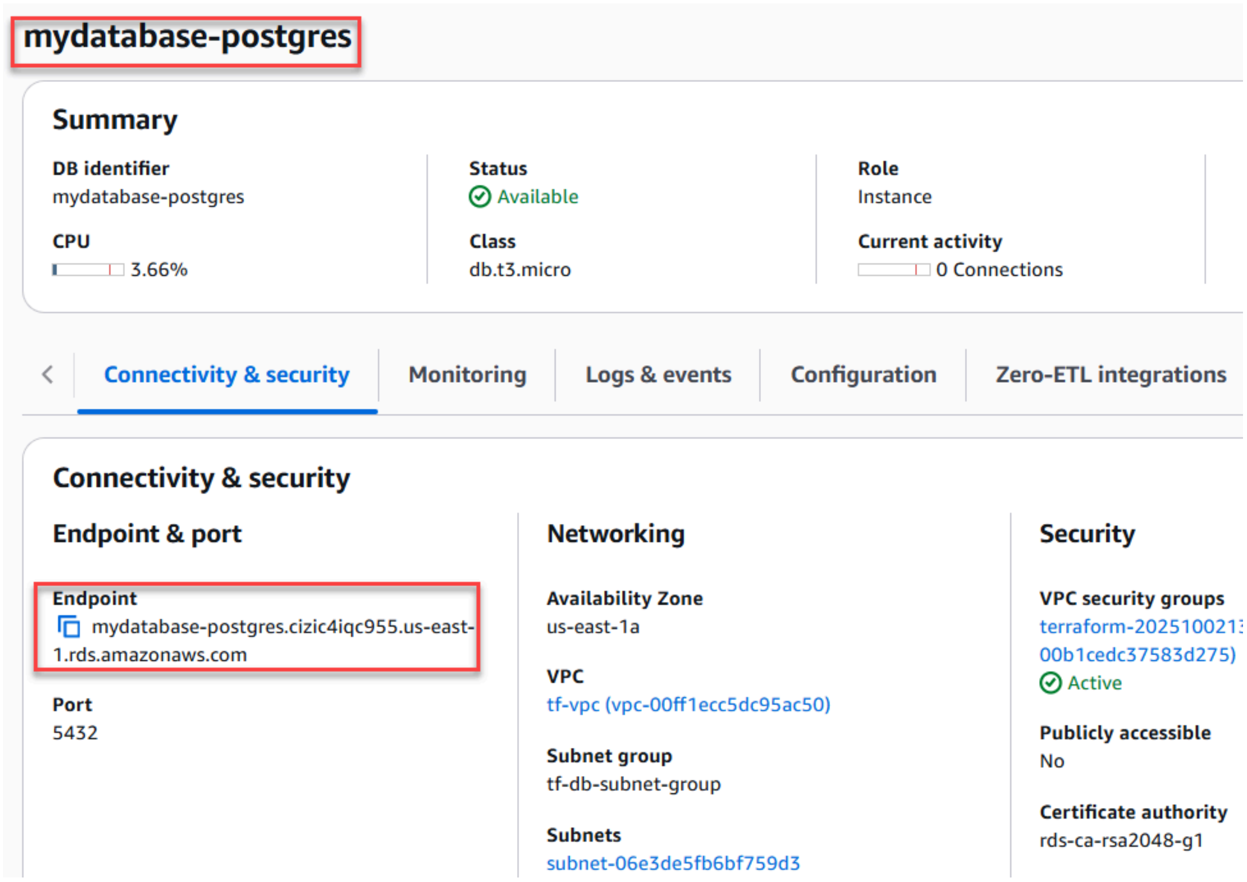
Name	Security group rule ID	IP version	Type
–	sgr-0879bc268973de34e	IPv4	PostgreSQL
–	sgr-0122a0b7210d59704	IPv4	PostgreSQL

EC2 Systems



Name	Instance ID	Instance state
tf-public-ec2	i-0c461d659d440e7cf	Running
tf-private-ec2	i-0dd7a064a4a553f0b	Running

RDS



mydatabase-postgres

Summary

DB identifier
mydatabase-postgres

CPU
3.66%

Status
Available

Class
db.t3.micro

Role
Instance

Current activity
0 Connections

Connectivity & security

Endpoint & port

Endpoint
mydatabase-postgres.cizic4iqc955.us-east-1.rds.amazonaws.com

Port
5432

Networking

Availability Zone
us-east-1a

VPC
tf-vpc (vpc-00ff1ecc5dc95ac50)

Subnet group
tf-db-subnet-group

Subnets
subnet-06e3de5fb6bf759d3

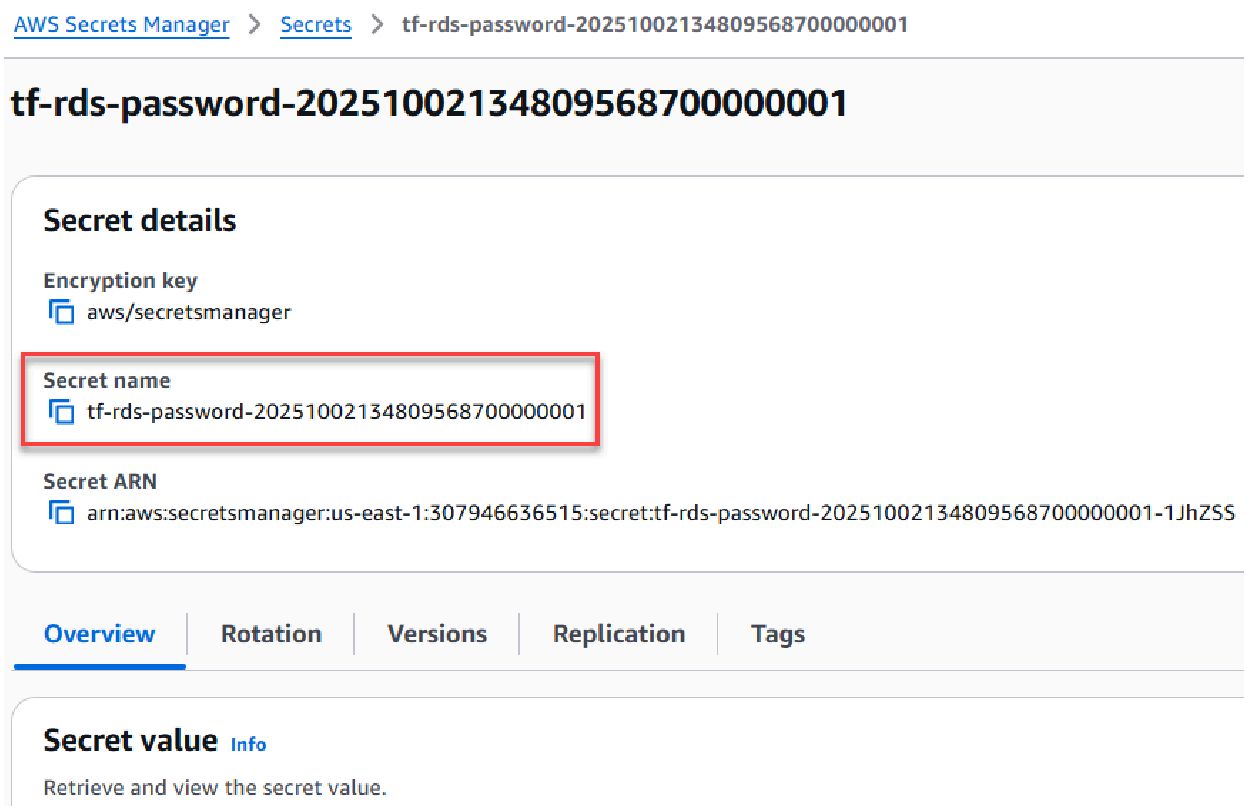
Security

VPC security groups
terraform-20251002
00b1cedc37583d275)
Active

Publicly accessible
No

Certificate authority
rds-ca-rsa2048-g1

Secret Manager



AWS Secrets Manager > Secrets > tf-rds-password-20251002134809568700000001

tf-rds-password-20251002134809568700000001

Secret details

Encryption key
aws/secretsmanager

Secret name
tf-rds-password-20251002134809568700000001

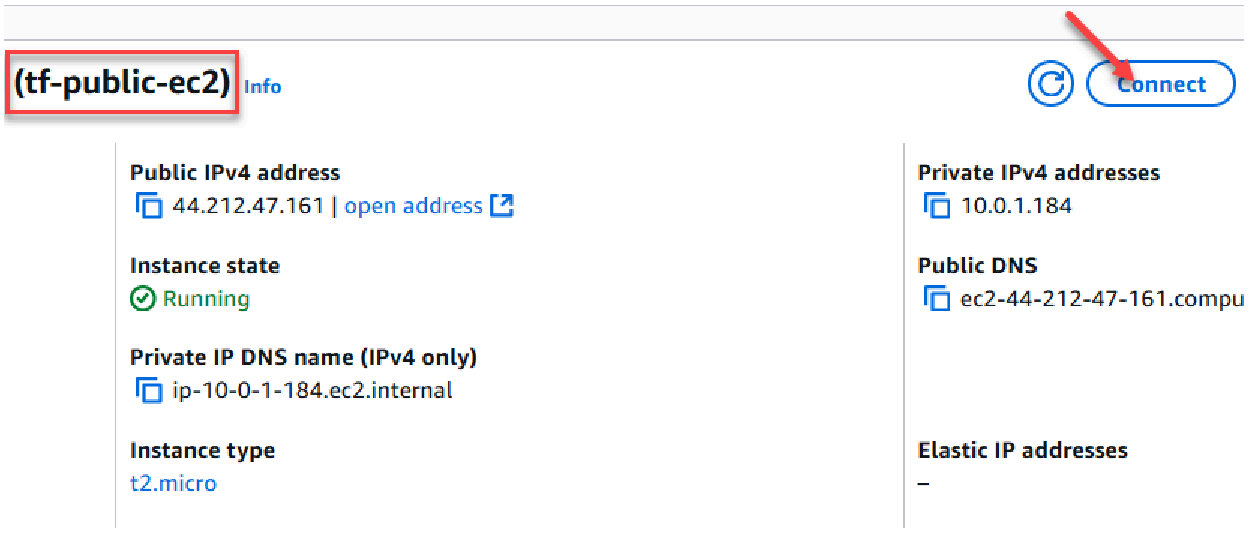
Secret ARN
arn:aws:secretsmanager:us-east-1:307946636515:secret:tf-rds-password-20251002134809568700000001-1JhZSS

Secret value Info

Retrieve and view the secret value.

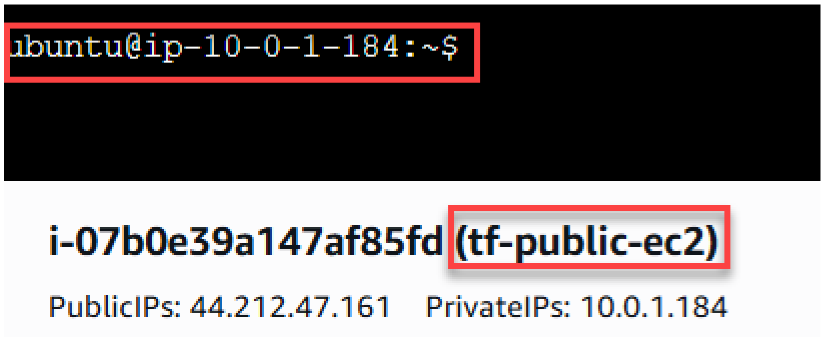
Step 9: Connectivity Testing

Public EC2 (Connection Success from Outside)



(tf-public-ec2) Info

Public IPv4 address 44.212.47.161 | open address	Private IPv4 addresses 10.0.1.184
Instance state Running	Public DNS ec2-44-212-47-161.compu
Private IP DNS name (IPv4 only) ip-10-0-1-184.ec2.internal	
Instance type t2.micro	Elastic IP addresses –

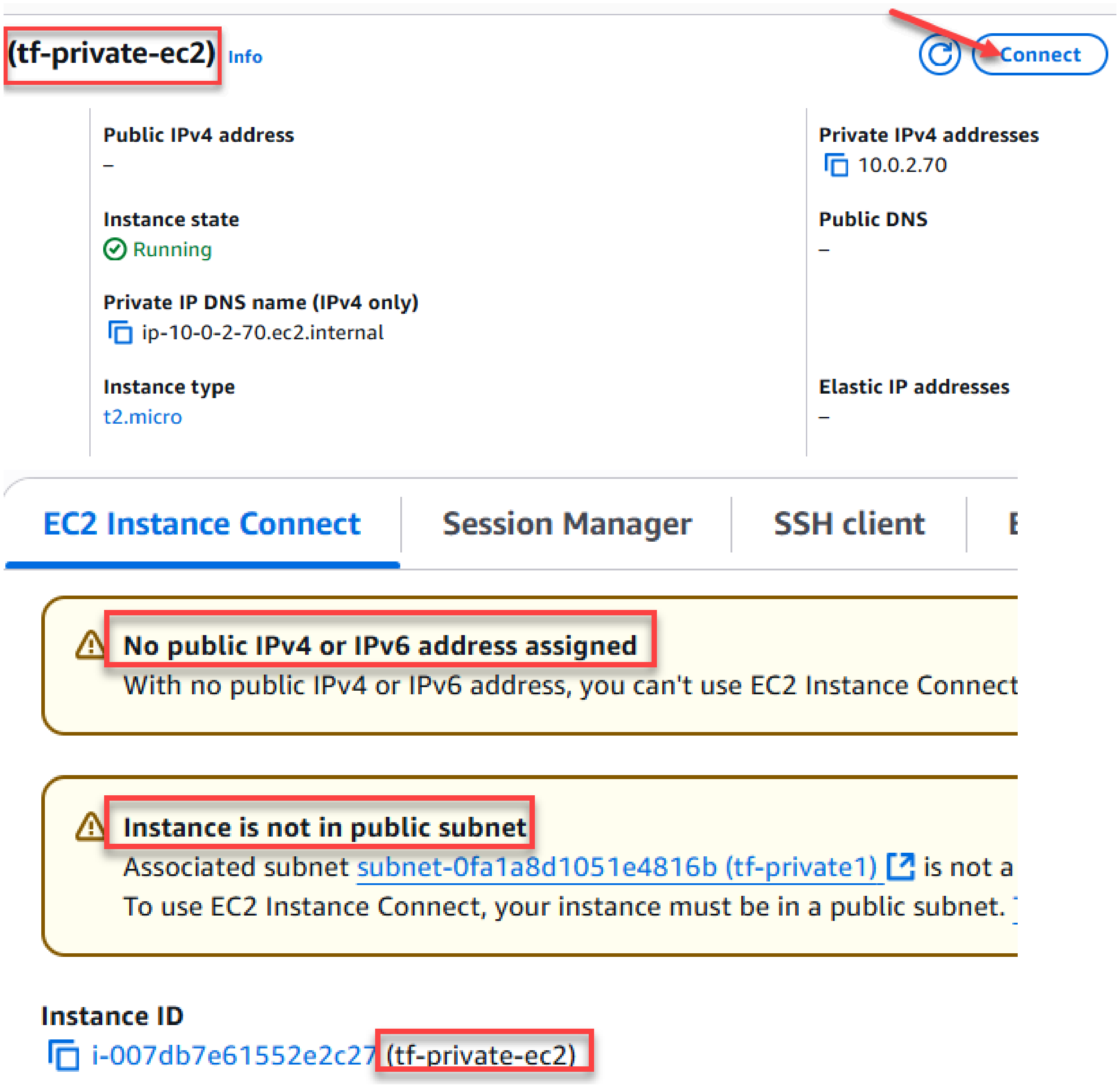


```
ubuntu@ip-10-0-1-184:~$
```

i-07b0e39a147af85fd (tf-public-ec2)

PublicIPs: 44.212.47.161   PrivateIPs: 10.0.1.184

Private EC2 (Connection Failure from Outside)



(tf-private-ec2) Info

Public IPv4 address
–

Private IPv4 addresses
10.0.2.70

Instance state
Running

Public DNS
–

Private IP DNS name (IPv4 only)
ip-10-0-2-70.ec2.internal

Instance type
t2.micro

Elastic IP addresses
–

No public IPv4 or IPv6 address assigned
With no public IPv4 or IPv6 address, you can't use EC2 Instance Connect

Instance is not in public subnet
Associated subnet subnet-0fa1a8d1051e4816b (tf-private1) is not a
To use EC2 Instance Connect, your instance must be in a public subnet.

Instance ID
i-007db7e61552e2c27 (tf-private-ec2)