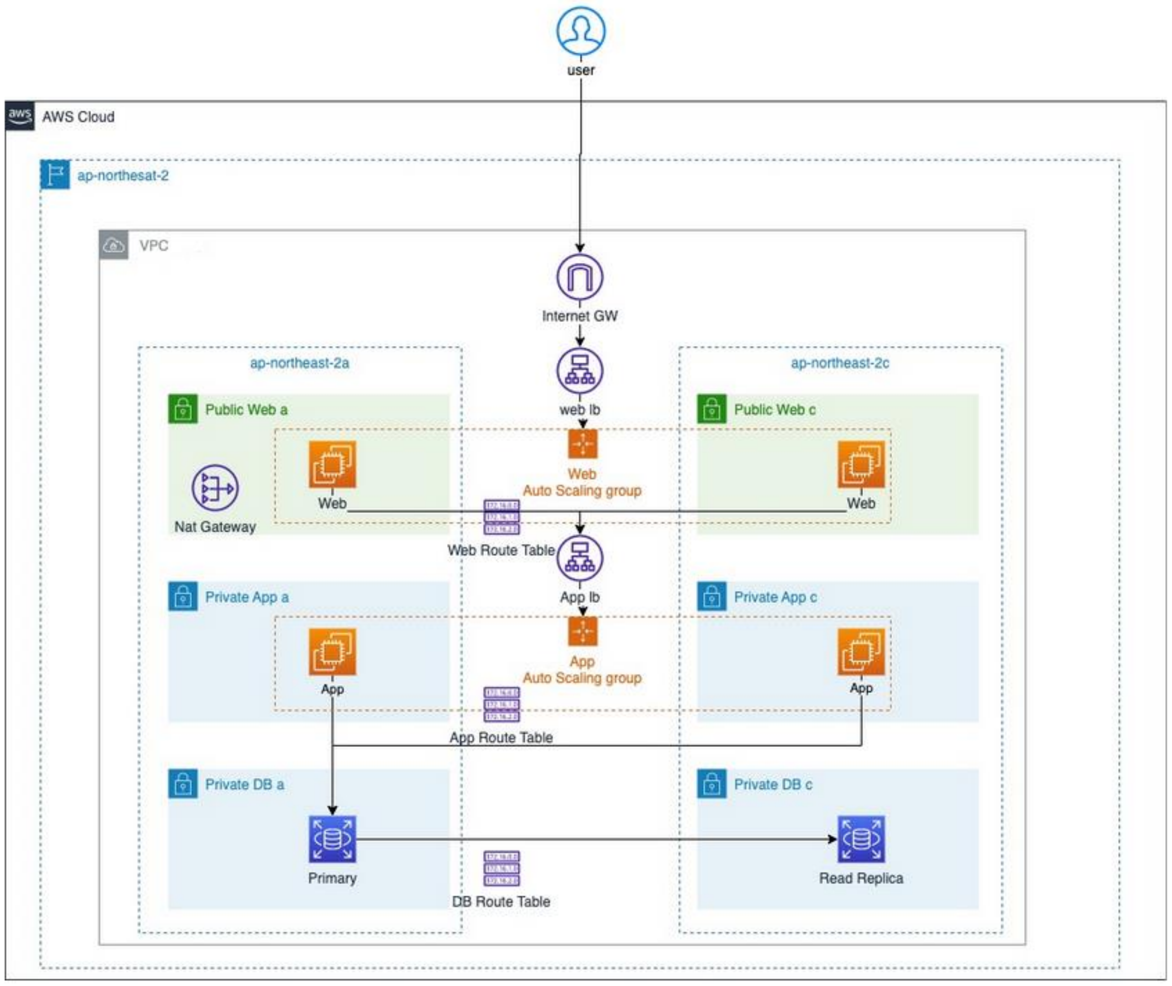
테라폼 프로젝트 보고서

작성자: 장민우

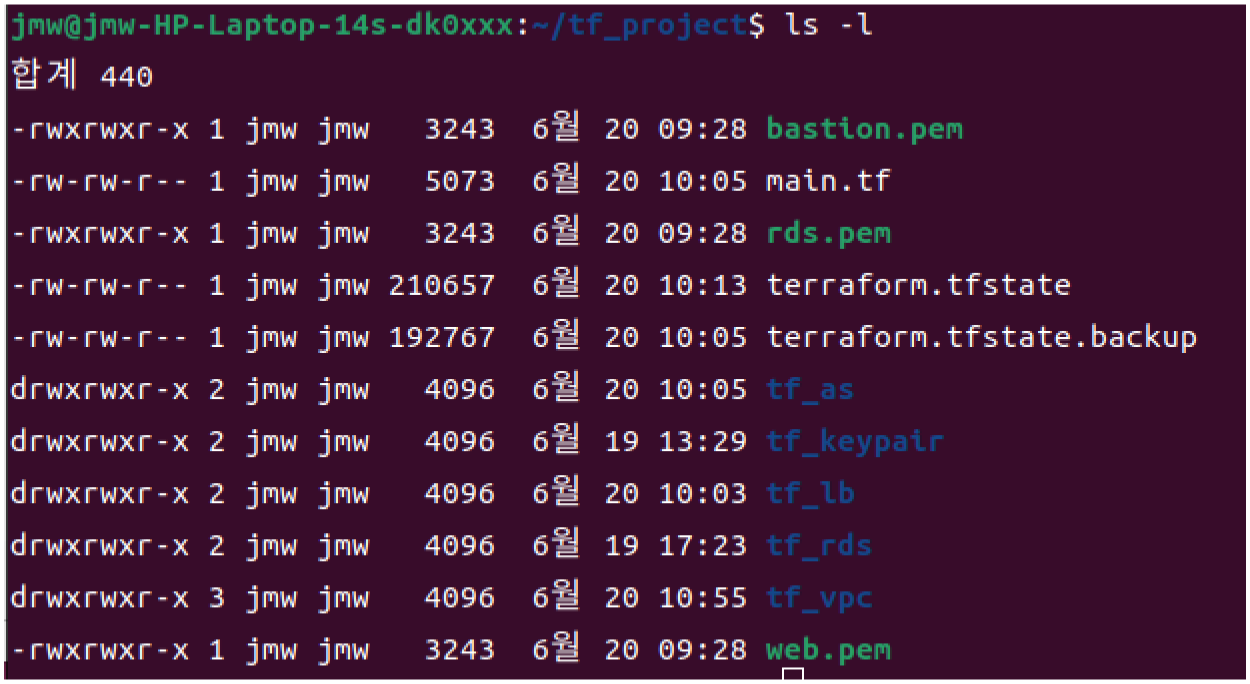
목차

1. 개요



aws 로 구성했던 인프라를 테라폼을 다시 구성해본다.

2. 구성



```
jmw@jmw-HP-Laptop-14s-dk0xxx:~/tf_project$ ls -l
합계 440
-rwxrwxr-x 1 jmw jmw   3243  6월 20 09:28 bastion.pem
-rw-rw-r-- 1 jmw jmw   5073  6월 20 10:05 main.tf
-rwxrwxr-x 1 jmw jmw   3243  6월 20 09:28 rds.pem
-rw-rw-r-- 1 jmw jmw 210657  6월 20 10:13 terraform.tfstate
-rw-rw-r-- 1 jmw jmw 192767  6월 20 10:05 terraform.tfstate.backup
drwxrwxr-x 2 jmw jmw   4096  6월 20 10:05 tf_as
drwxrwxr-x 2 jmw jmw   4096  6월 19 13:29 tf_keypair
drwxrwxr-x 2 jmw jmw   4096  6월 20 10:03 tf_lb
drwxrwxr-x 2 jmw jmw   4096  6월 19 17:23 tf_rds
drwxrwxr-x 3 jmw jmw   4096  6월 20 10:55 tf_vpc
-rwxrwxr-x 1 jmw jmw   3243  6월 20 09:28 web.pem
```

tf_project

...... main.tf (테라폼 실행.)

...... tf_vpc

    ..... .main.tf (vpc 모듈)

...... tf_keypair

    ...... main.tf (키페어 모듈)

...... tf_lb

    ...... main.tf (로드밸런싱 모듈)

...... tf_as

    ...... main.tf (오토스케일링 모듈)

...... tf_rds

…… main.tf (rds 모듈)

…… bastion.pem (키페어 모듈 실행 후. 생성)

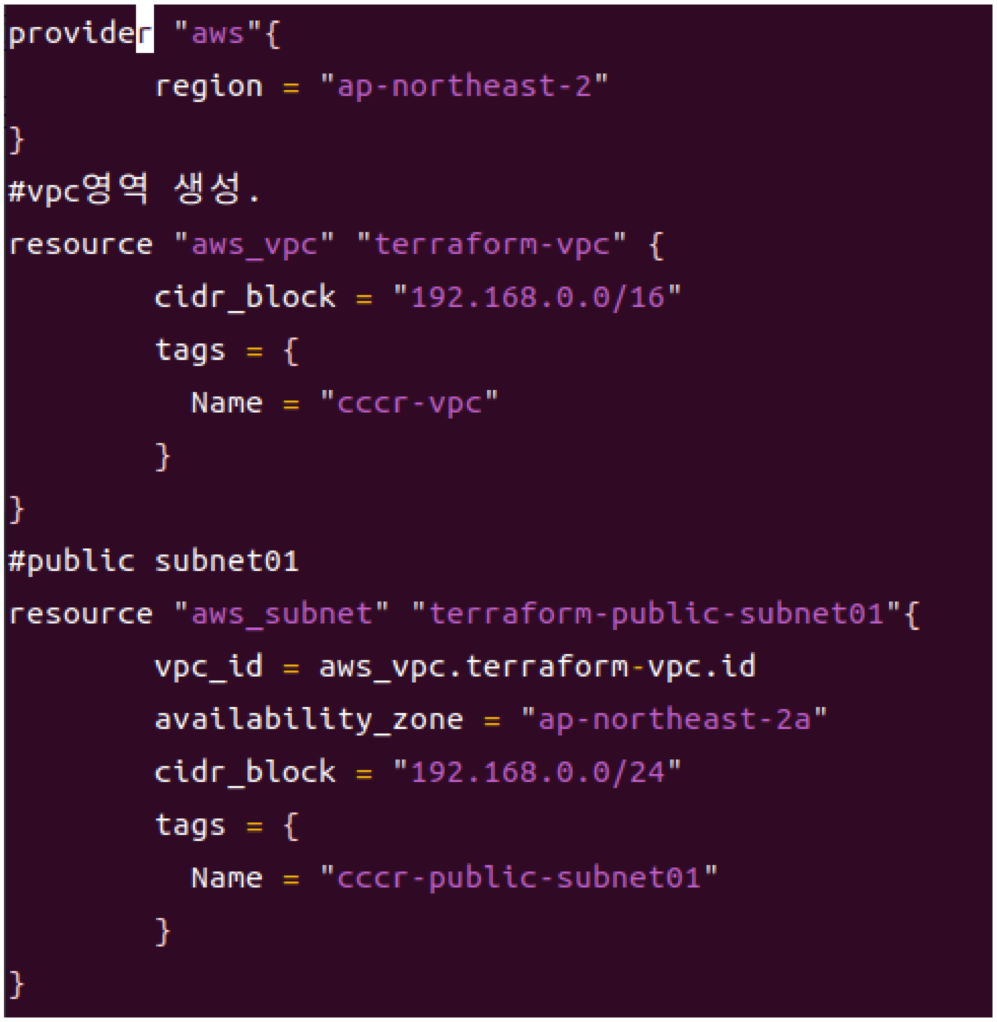
…… web.pem (키페어 모듈 실행 후. 생성)

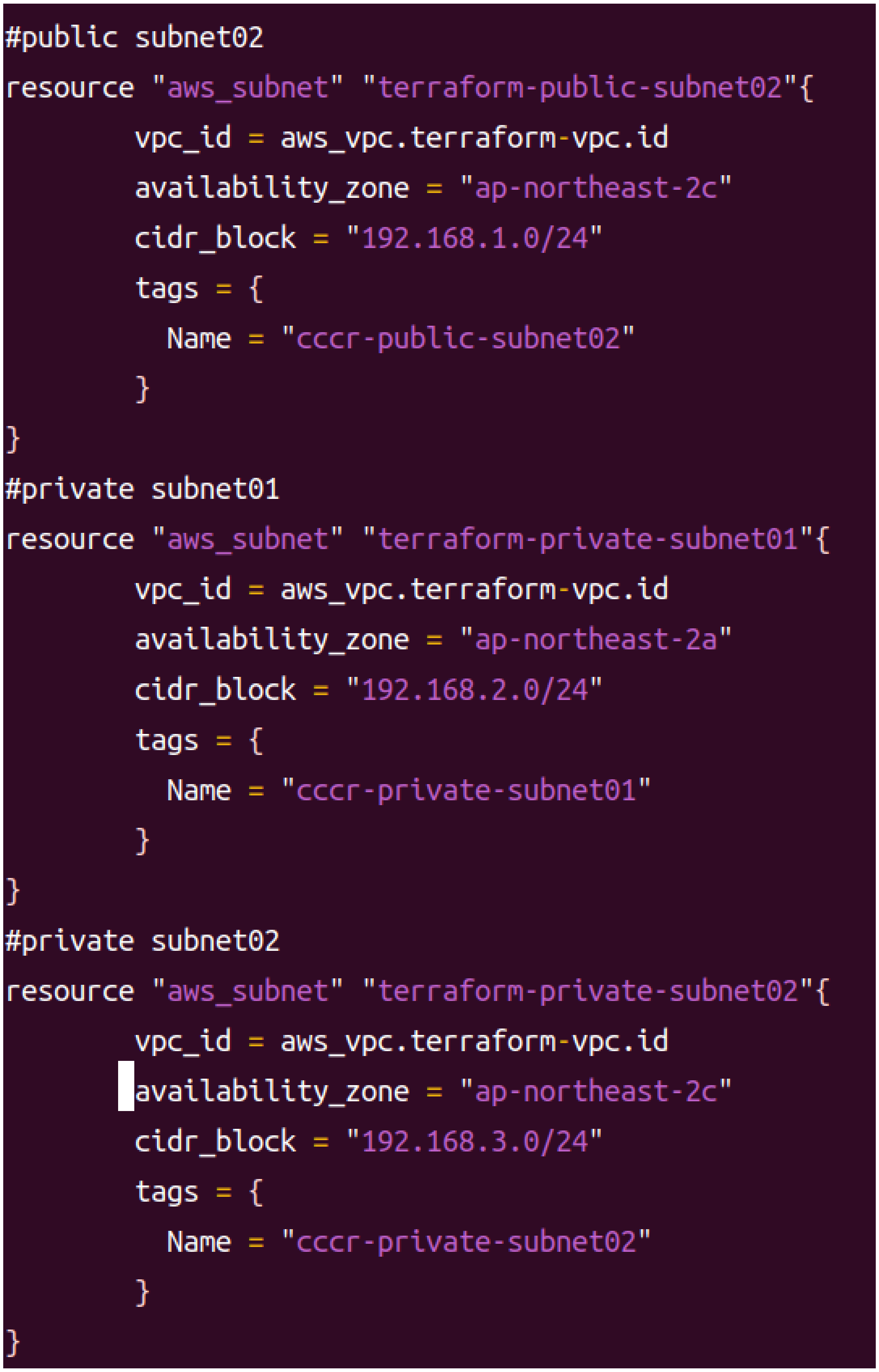
…… rds.pem (키페어 모듈 실행 후. 생성)

3. vpc 모듈.

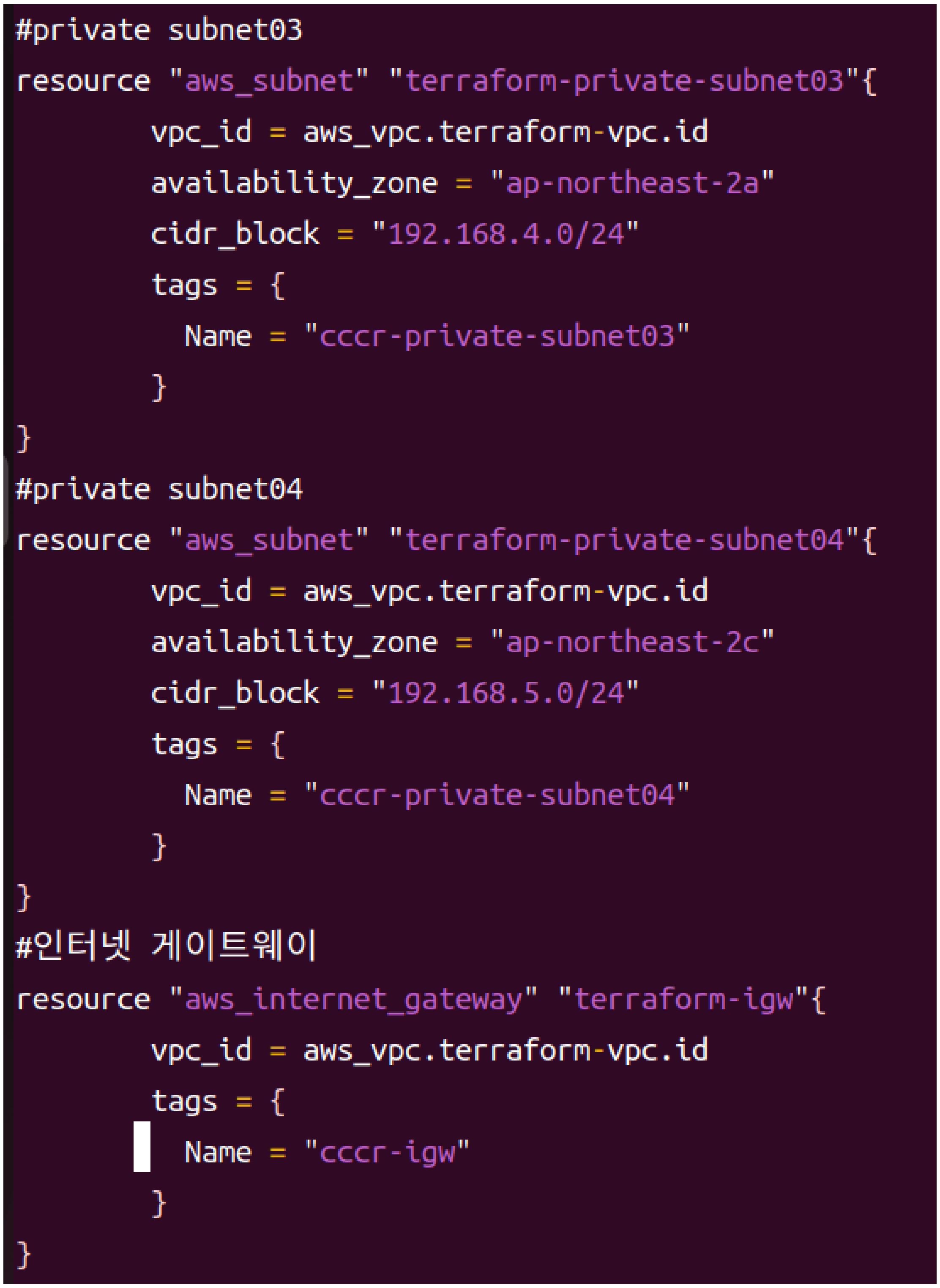
#vim tf_project/tf_vpc/tf.main

1. vpc 영역과 서브넷을 생성 해준다.


```
provider "aws"{
        region = "ap-northeast-2"
}
#vpc영역 생성.
resource "aws_vpc" "terraform-vpc" {
        cidr_block = "192.168.0.0/16"
        tags = {
          Name = "cccr-vpc"
        }
}
#public subnet01
resource "aws_subnet" "terraform-public-subnet01"{
        vpc_id = aws_vpc.terraform-vpc.id
        availability_zone = "ap-northeast-2a"
        cidr_block = "192.168.0.0/24"
        tags = {
          Name = "cccr-public-subnet01"
        }
}
```

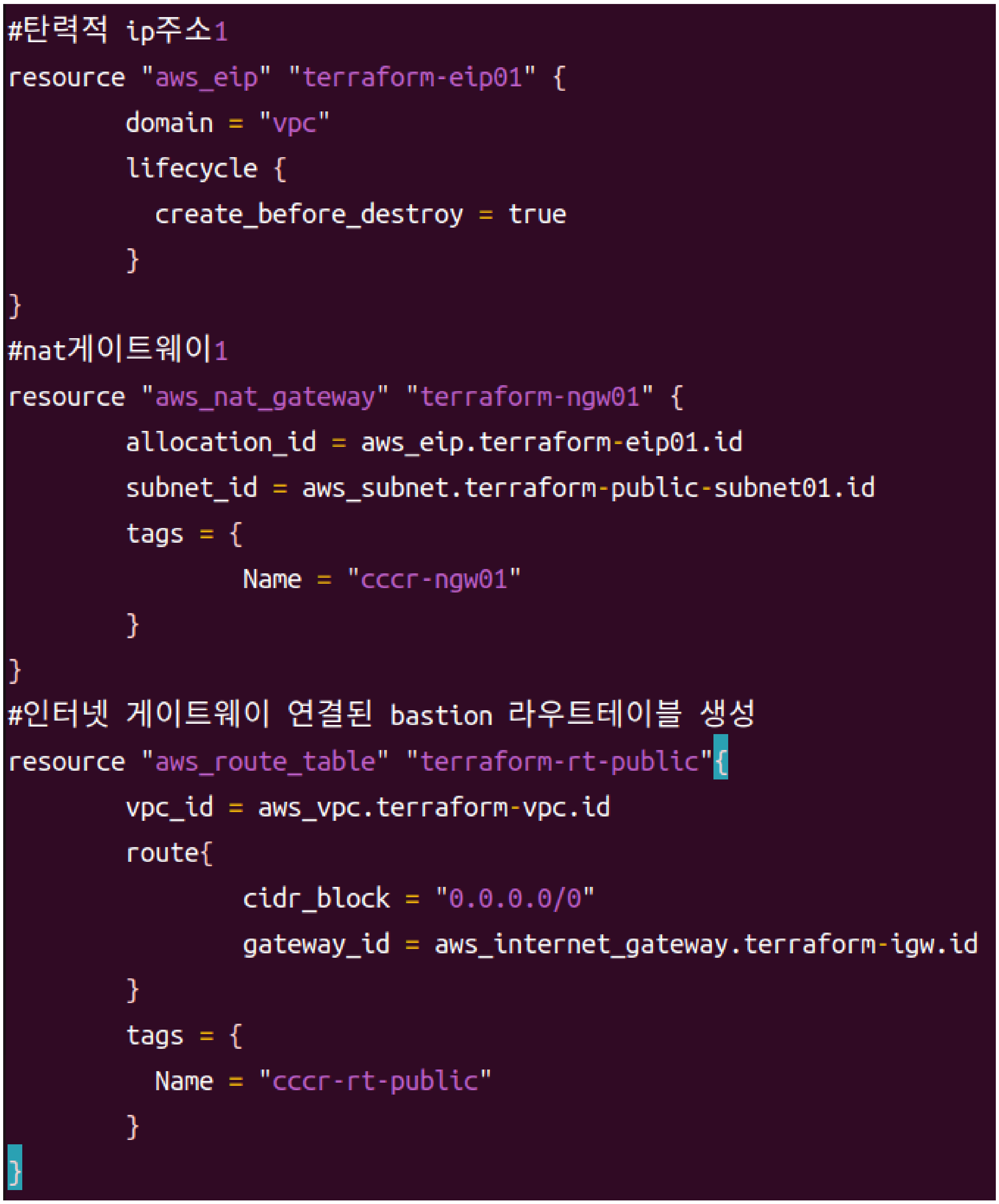

```
#public subnet02
resource "aws_subnet" "terraform-public-subnet02"{
        vpc_id = aws_vpc.terraform-vpc.id
        availability_zone = "ap-northeast-2c"
        cidr_block = "192.168.1.0/24"
        tags = {
          Name = "cccr-public-subnet02"
        }
}
#private subnet01
resource "aws_subnet" "terraform-private-subnet01"{
        vpc_id = aws_vpc.terraform-vpc.id
        availability_zone = "ap-northeast-2a"
        cidr_block = "192.168.2.0/24"
        tags = {
          Name = "cccr-private-subnet01"
        }
}
#private subnet02
resource "aws_subnet" "terraform-private-subnet02"{
        vpc_id = aws_vpc.terraform-vpc.id
        availability_zone = "ap-northeast-2c"
        cidr_block = "192.168.3.0/24"
        tags = {
          Name = "cccr-private-subnet02"
        }
}
```

2. 서브넷 생성 후. 인터넷 게이트웨이와 nat 게이트웨이를 생성해준다.

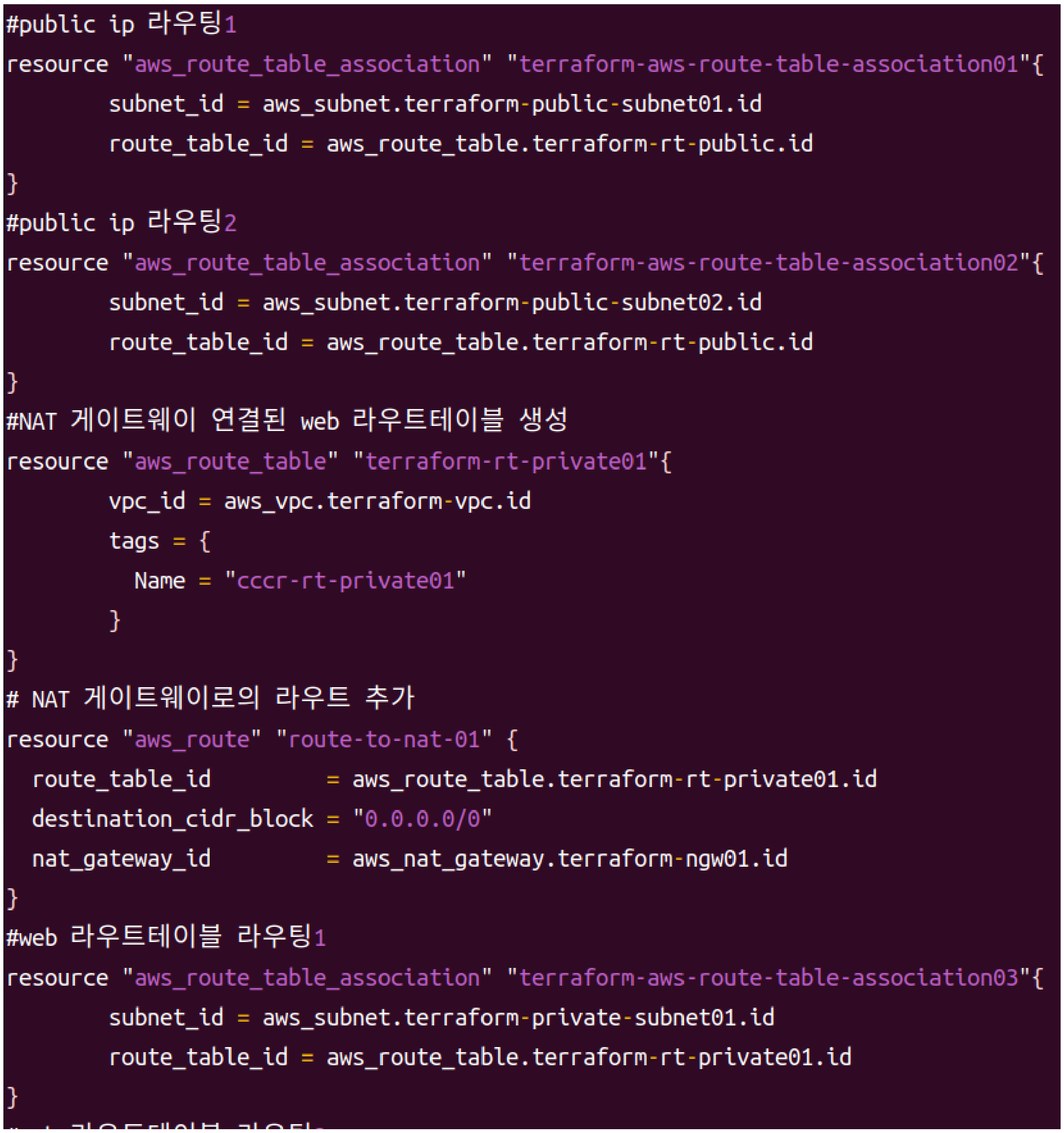


```
#private subnet03
resource "aws_subnet" "terraform-private-subnet03"{
        vpc_id = aws_vpc.terraform-vpc.id
        availability_zone = "ap-northeast-2a"
        cidr_block = "192.168.4.0/24"
        tags = {
          Name = "cccr-private-subnet03"
        }
}
#private subnet04
resource "aws_subnet" "terraform-private-subnet04"{
        vpc_id = aws_vpc.terraform-vpc.id
        availability_zone = "ap-northeast-2c"
        cidr_block = "192.168.5.0/24"
        tags = {
          Name = "cccr-private-subnet04"
        }
}
#인터넷 게이트웨이
resource "aws_internet_gateway" "terraform-igw"{
        vpc_id = aws_vpc.terraform-vpc.id
        tags = {
          Name = "cccr-igw"
        }
}
```

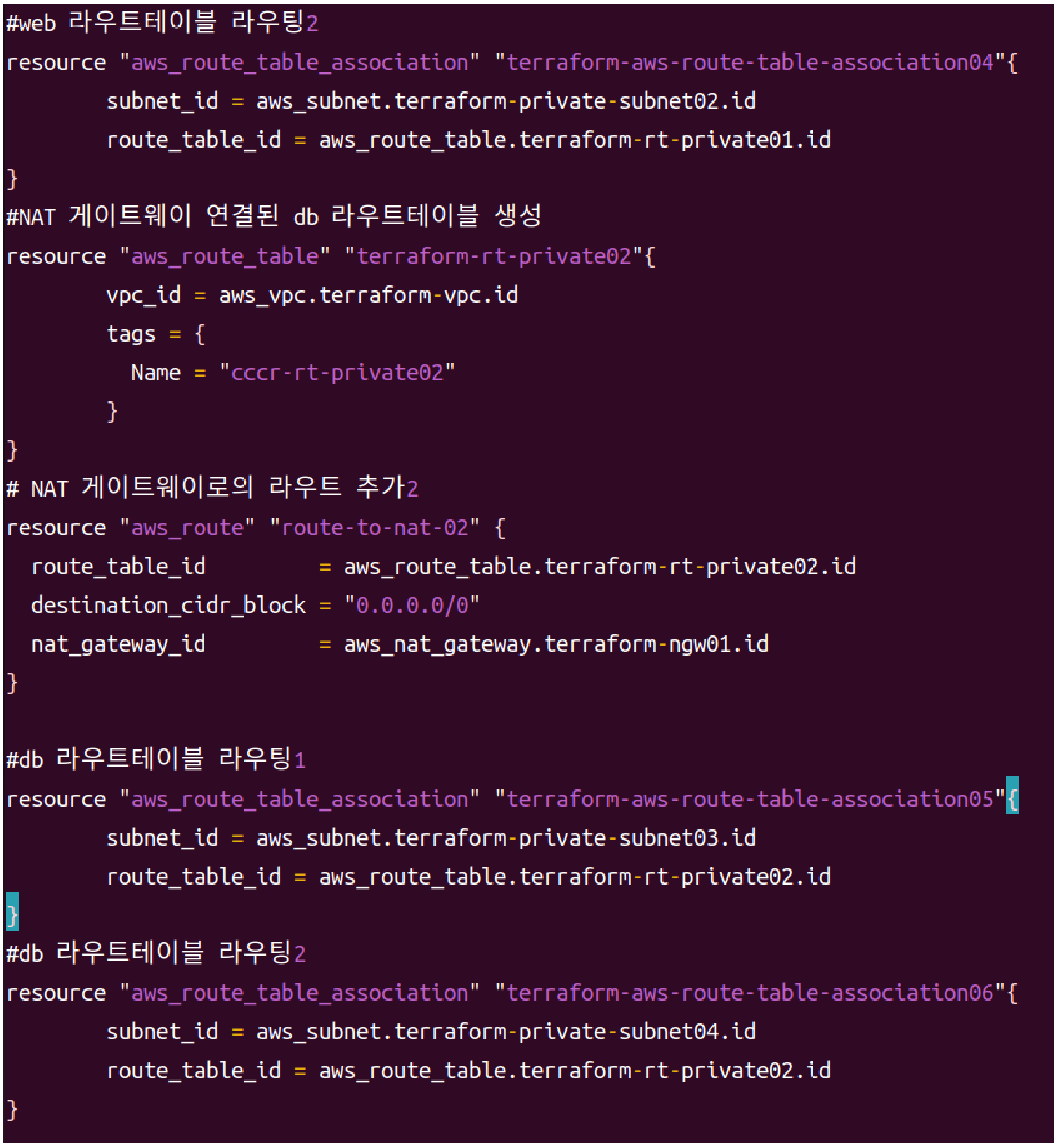
3. 라우팅 테이블 생성



```
#탄력적 ip주소1
resource "aws_eip" "terraform-eip01" {
        domain = "vpc"
        lifecycle {
          create_before_destroy = true
        }
}
#nat게이트웨이1
resource "aws_nat_gateway" "terraform-ngw01" {
        allocation_id = aws_eip.terraform-eip01.id
        subnet_id = aws_subnet.terraform-public-subnet01.id
        tags = {
                Name = "cccr-ngw01"
        }
}
#인터넷 게이트웨이 연결된 bastion 라우트테이블 생성
resource "aws_route_table" "terraform-rt-public"{
        vpc_id = aws_vpc.terraform-vpc.id
        route{
                cidr_block = "0.0.0.0/0"
                gateway_id = aws_internet_gateway.terraform-igw.id
        }
        tags = {
          Name = "cccr-rt-public"
        }
}
```

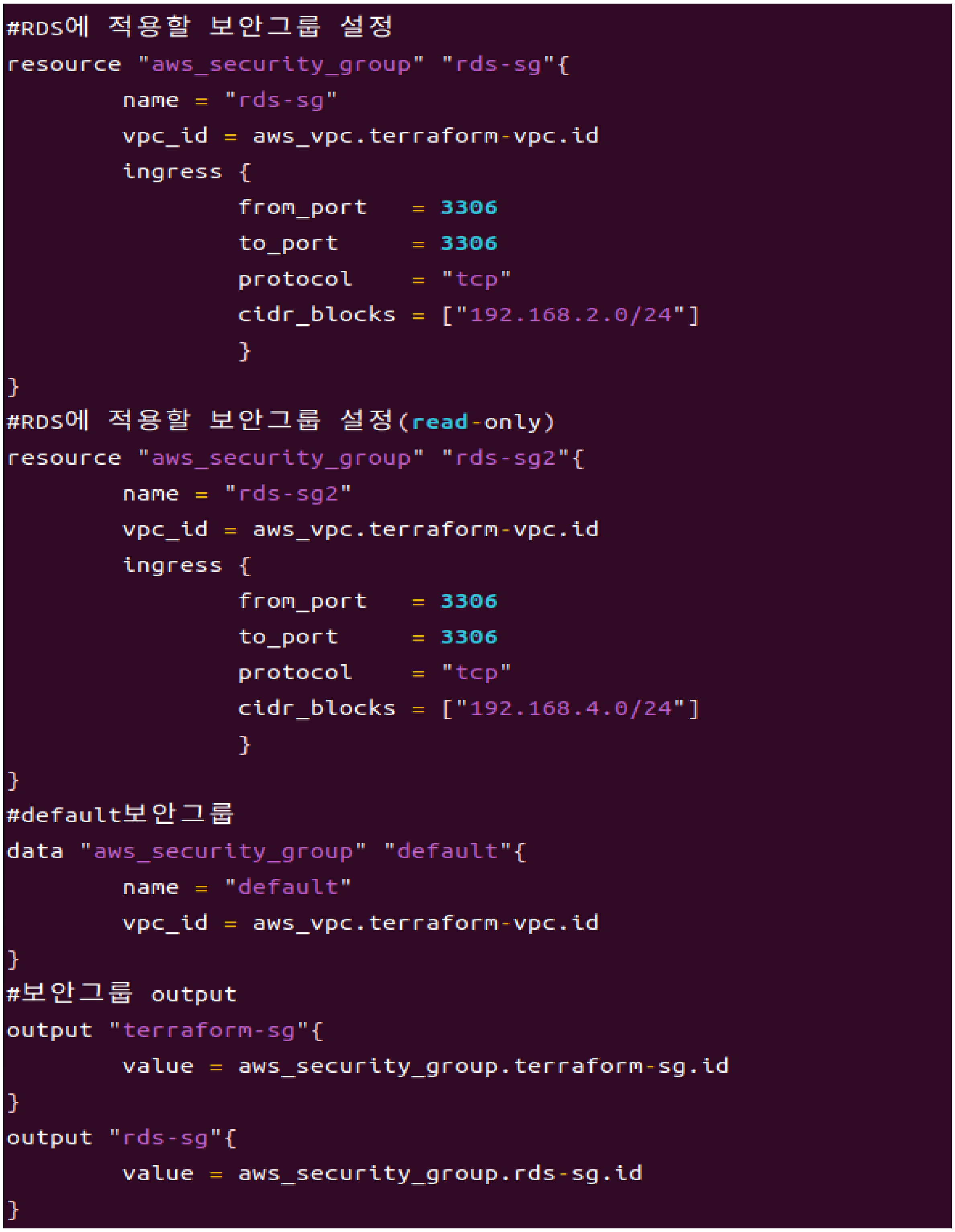


```
#public ip 라우팅1
resource "aws_route_table_association" "terraform-aws-route-table-association01"{
        subnet_id = aws_subnet.terraform-public-subnet01.id
        route_table_id = aws_route_table.terraform-rt-public.id
}
#public ip 라우팅2
resource "aws_route_table_association" "terraform-aws-route-table-association02"{
        subnet_id = aws_subnet.terraform-public-subnet02.id
        route_table_id = aws_route_table.terraform-rt-public.id
}
#NAT 게이트웨이 연결된 web 라우트테이블 생성
resource "aws_route_table" "terraform-rt-private01"{
        vpc_id = aws_vpc.terraform-vpc.id
        tags = {
          Name = "cccr-rt-private01"
        }
}
# NAT 게이트웨이로의 라우트 추가
resource "aws_route" "route-to-nat-01" {
  route_table_id         = aws_route_table.terraform-rt-private01.id
  destination_cidr_block = "0.0.0.0/0"
  nat_gateway_id         = aws_nat_gateway.terraform-ngw01.id
}
#web 라우트테이블 라우팅1
resource "aws_route_table_association" "terraform-aws-route-table-association03"{
        subnet_id = aws_subnet.terraform-private-subnet01.id
        route_table_id = aws_route_table.terraform-rt-private01.id
}
```



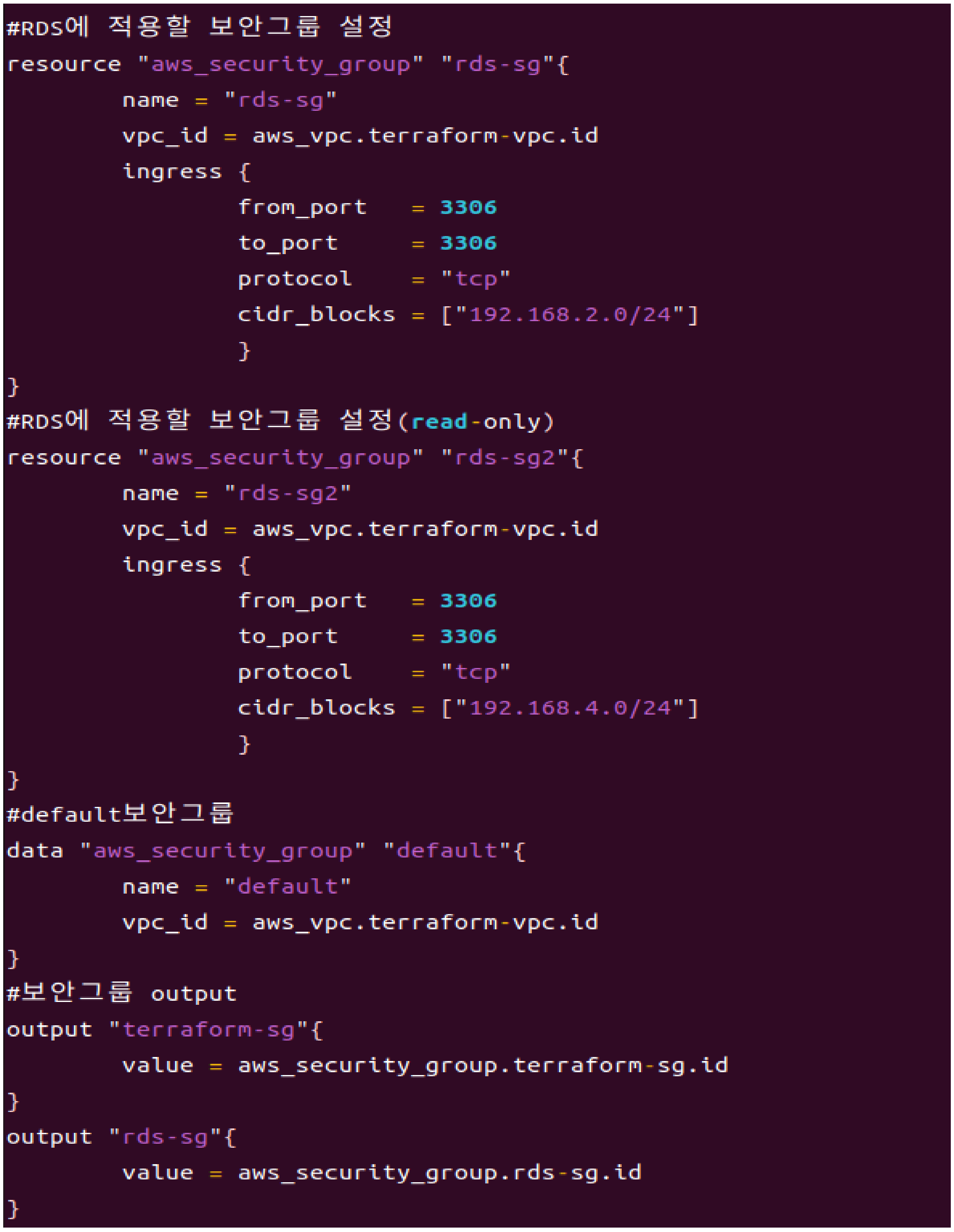
```
#web 라우트테이블 라우팅2
resource "aws_route_table_association" "terraform-aws-route-table-association04"{
        subnet_id = aws_subnet.terraform-private-subnet02.id
        route_table_id = aws_route_table.terraform-rt-private01.id
}
#NAT 게이트웨이 연결된 db 라우트테이블 생성
resource "aws_route_table" "terraform-rt-private02"{
        vpc_id = aws_vpc.terraform-vpc.id
        tags = {
          Name = "cccr-rt-private02"
        }
}
# NAT 게이트웨이로의 라우트 추가2
resource "aws_route" "route-to-nat-02" {
  route_table_id         = aws_route_table.terraform-rt-private02.id
  destination_cidr_block = "0.0.0.0/0"
  nat_gateway_id         = aws_nat_gateway.terraform-ngw01.id
}

#db 라우트테이블 라우팅1
resource "aws_route_table_association" "terraform-aws-route-table-association05"{
        subnet_id = aws_subnet.terraform-private-subnet03.id
        route_table_id = aws_route_table.terraform-rt-private02.id
}
#db 라우트테이블 라우팅2
resource "aws_route_table_association" "terraform-aws-route-table-association06"{
        subnet_id = aws_subnet.terraform-private-subnet04.id
        route_table_id = aws_route_table.terraform-rt-private02.id
}
```

4. 보안그룹 설정

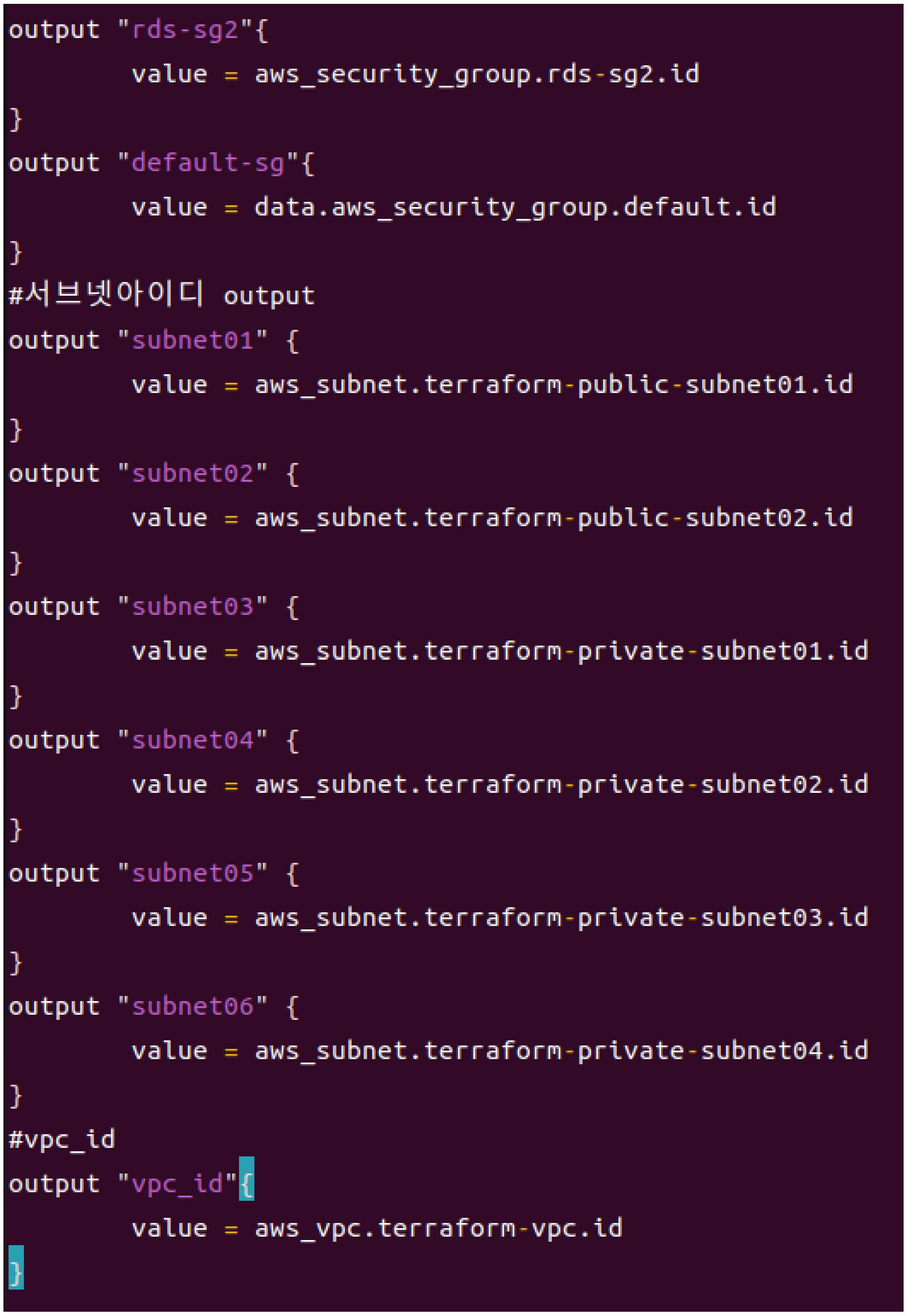


```
#RDS에 적용할 보안그룹 설정
resource "aws_security_group" "rds-sg"{
        name = "rds-sg"
        vpc_id = aws_vpc.terraform-vpc.id
        ingress {
                from_port    = 3306
                to_port      = 3306
                protocol     = "tcp"
                cidr_blocks = ["192.168.2.0/24"]
                }
}
#RDS에 적용할 보안그룹 설정(read-only)
resource "aws_security_group" "rds-sg2"{
        name = "rds-sg2"
        vpc_id = aws_vpc.terraform-vpc.id
        ingress {
                from_port    = 3306
                to_port      = 3306
                protocol     = "tcp"
                cidr_blocks = ["192.168.4.0/24"]
                }
}
#default보안그룹
data "aws_security_group" "default"{
        name = "default"
        vpc_id = aws_vpc.terraform-vpc.id
}
#보안그룹 output
output "terraform-sg"{
        value = aws_security_group.terraform-sg.id
}
output "rds-sg"{
        value = aws_security_group.rds-sg.id
}
```

5. vpc 생성 완료 후. tf_project/main.tf 에서 사용할 변수를 output 해준다.



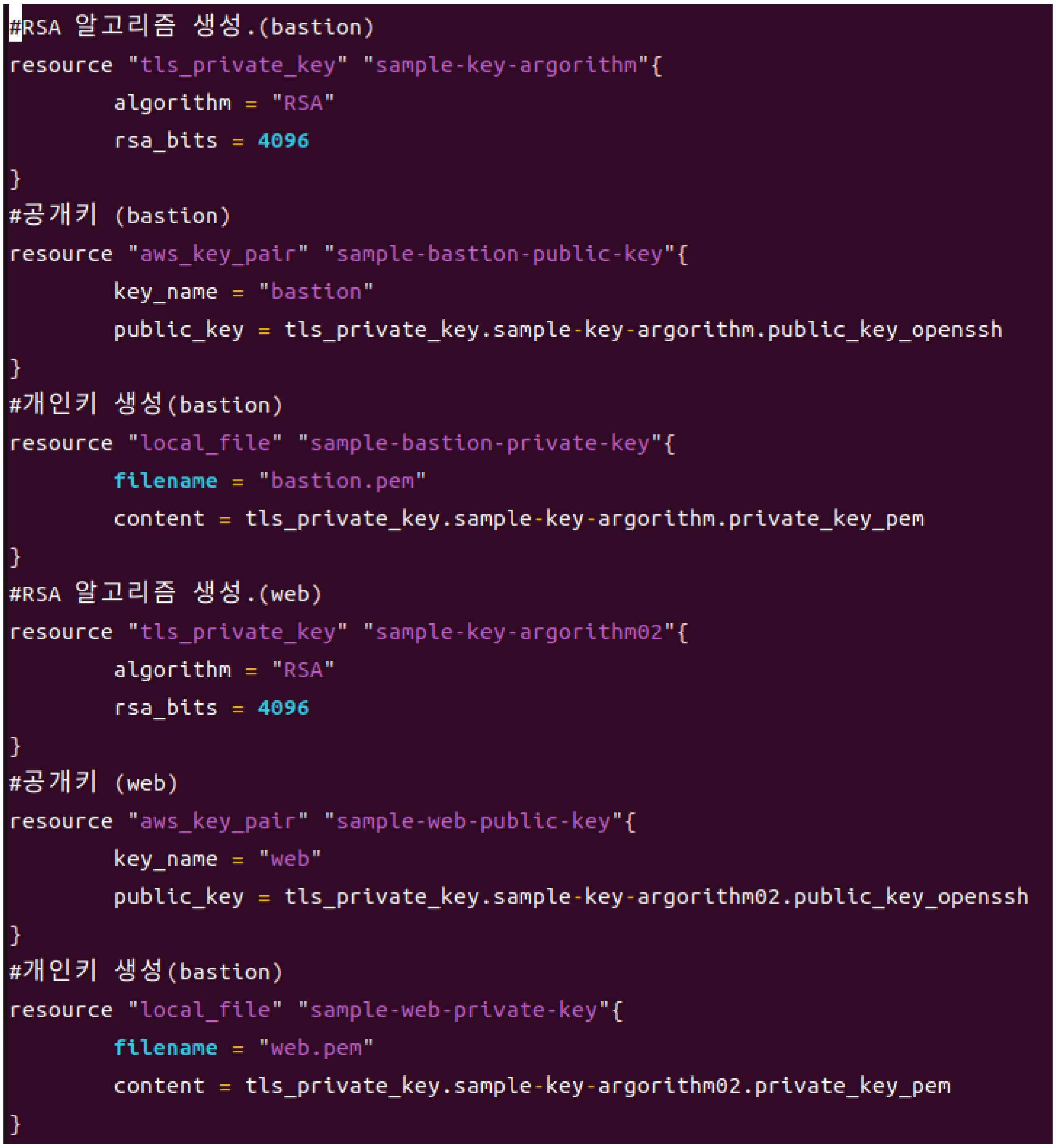
```
#RDS에 적용할 보안그룹 설정
resource "aws_security_group" "rds-sg"{
        name = "rds-sg"
        vpc_id = aws_vpc.terraform-vpc.id
        ingress {
                from_port    = 3306
                to_port      = 3306
                protocol     = "tcp"
                cidr_blocks = ["192.168.2.0/24"]
                }
}
#RDS에 적용할 보안그룹 설정(read-only)
resource "aws_security_group" "rds-sg2"{
        name = "rds-sg2"
        vpc_id = aws_vpc.terraform-vpc.id
        ingress {
                from_port    = 3306
                to_port      = 3306
                protocol     = "tcp"
                cidr_blocks = ["192.168.4.0/24"]
                }
}
#default보안그룹
data "aws_security_group" "default"{
        name = "default"
        vpc_id = aws_vpc.terraform-vpc.id
}
#보안그룹 output
output "terraform-sg"{
        value = aws_security_group.terraform-sg.id
}
output "rds-sg"{
        value = aws_security_group.rds-sg.id
}
```



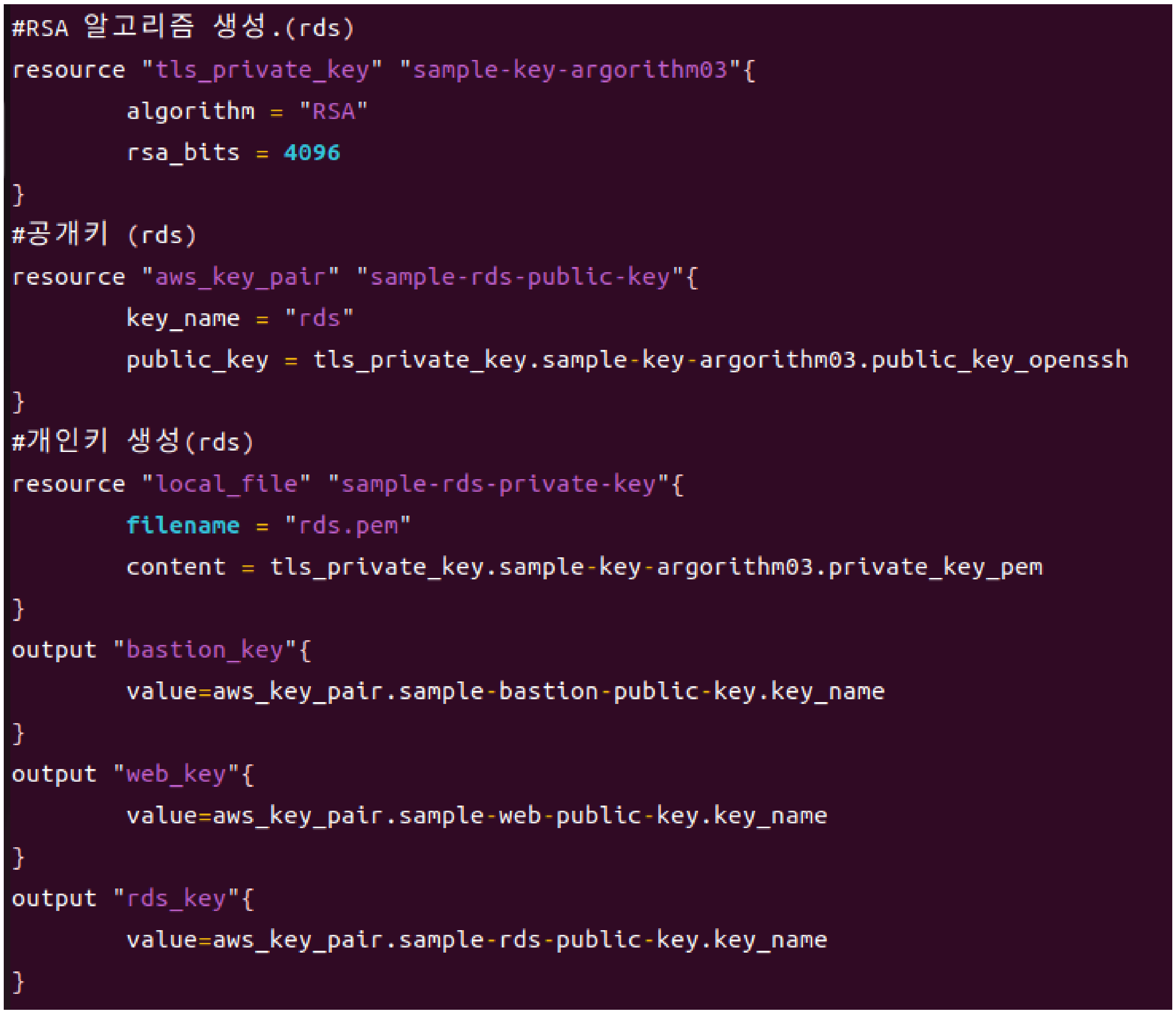
```
output "rds-sg2"{
        value = aws_security_group.rds-sg2.id
}
output "default-sg"{
        value = data.aws_security_group.default.id
}
#서브넷아이디 output
output "subnet01" {
        value = aws_subnet.terraform-public-subnet01.id
}
output "subnet02" {
        value = aws_subnet.terraform-public-subnet02.id
}
output "subnet03" {
        value = aws_subnet.terraform-private-subnet01.id
}
output "subnet04" {
        value = aws_subnet.terraform-private-subnet02.id
}
output "subnet05" {
        value = aws_subnet.terraform-private-subnet03.id
}
output "subnet06" {
        value = aws_subnet.terraform-private-subnet04.id
}
#vpc_id
output "vpc_id"{
        value = aws_vpc.terraform-vpc.id
}
```

4. keypair 모듈.

#vim tf_project/tf_keypair/tf.main



```
#RSA 알고리즘 생성.(bastion)
resource "tls_private_key" "sample-key-argorithm"{
        algorithm = "RSA"
        rsa_bits = 4096
}
#공개키 (bastion)
resource "aws_key_pair" "sample-bastion-public-key"{
        key_name = "bastion"
        public_key = tls_private_key.sample-key-argorithm.public_key_openssh
}
#개인키 생성(bastion)
resource "local_file" "sample-bastion-private-key"{
        filename = "bastion.pem"
        content = tls_private_key.sample-key-argorithm.private_key_pem
}
#RSA 알고리즘 생성.(web)
resource "tls_private_key" "sample-key-argorithm02"{
        algorithm = "RSA"
        rsa_bits = 4096
}
#공개키 (web)
resource "aws_key_pair" "sample-web-public-key"{
        key_name = "web"
        public_key = tls_private_key.sample-key-argorithm02.public_key_openssh
}
#개인키 생성(bastion)
resource "local_file" "sample-web-private-key"{
        filename = "web.pem"
        content = tls_private_key.sample-key-argorithm02.private_key_pem
}
```



```
#RSA 알고리즘 생성.(rds)
resource "tls_private_key" "sample-key-argorithm03"{
        algorithm = "RSA"
        rsa_bits = 4096
}
#공개키 (rds)
resource "aws_key_pair" "sample-rds-public-key"{
        key_name = "rds"
        public_key = tls_private_key.sample-key-argorithm03.public_key_openssh
}
#개인키 생성(rds)
resource "local_file" "sample-rds-private-key"{
        filename = "rds.pem"
        content = tls_private_key.sample-key-argorithm03.private_key_pem
}
output "bastion_key"{
        value=aws_key_pair.sample-bastion-public-key.key_name
}
output "web_key"{
        value=aws_key_pair.sample-web-public-key.key_name
}
output "rds_key"{
        value=aws_key_pair.sample-rds-public-key.key_name
}
```

키페어 모듈은 RSA 알고리즘 생성 후. 개인키와 공개키를 생성 해주면 된다.

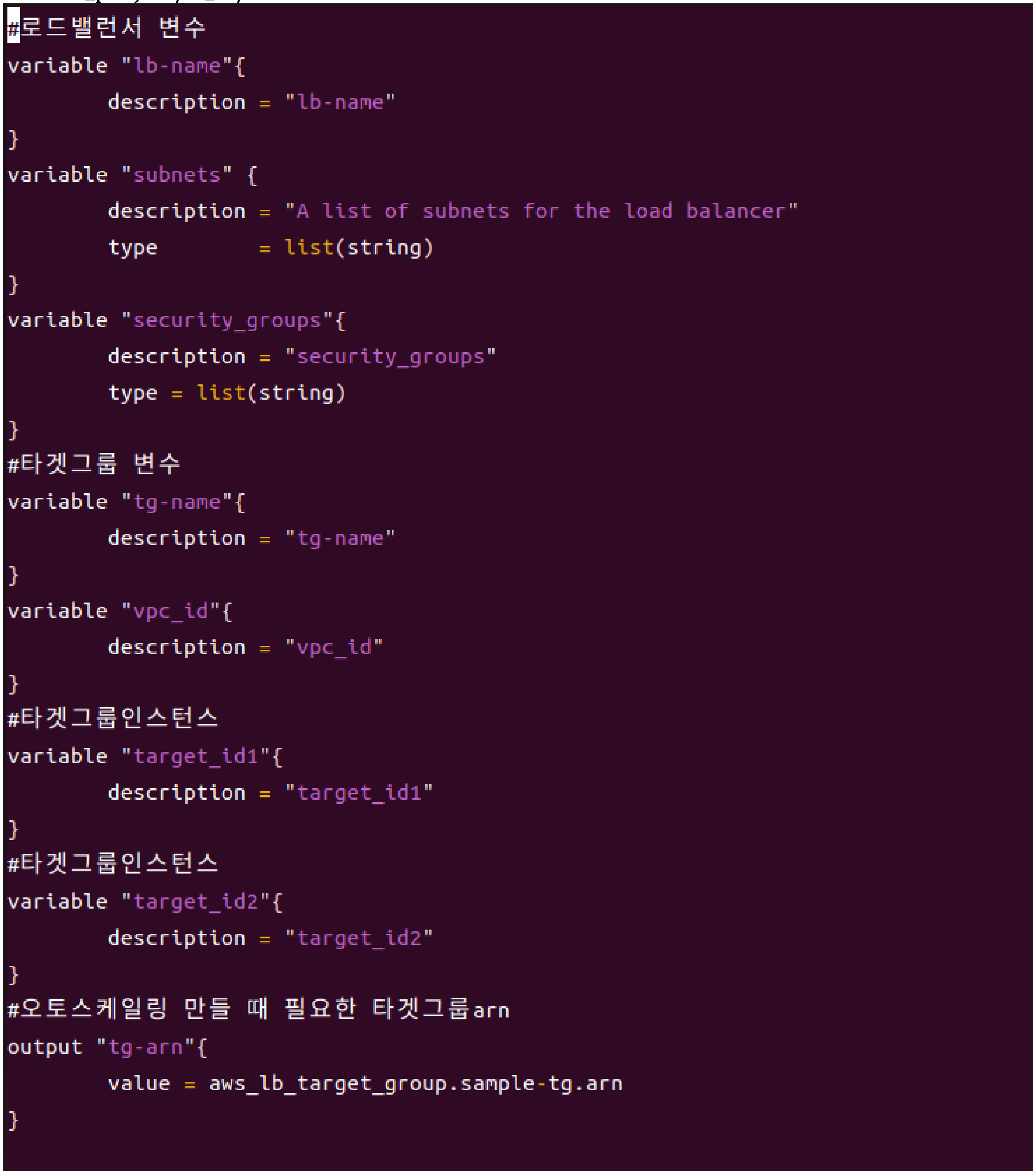
tf_project/main.tf 에서 테라폼을 실행하면 해당 디렉토리에

bastion.pem, web.pem, rds.pem 키가 생성된다.

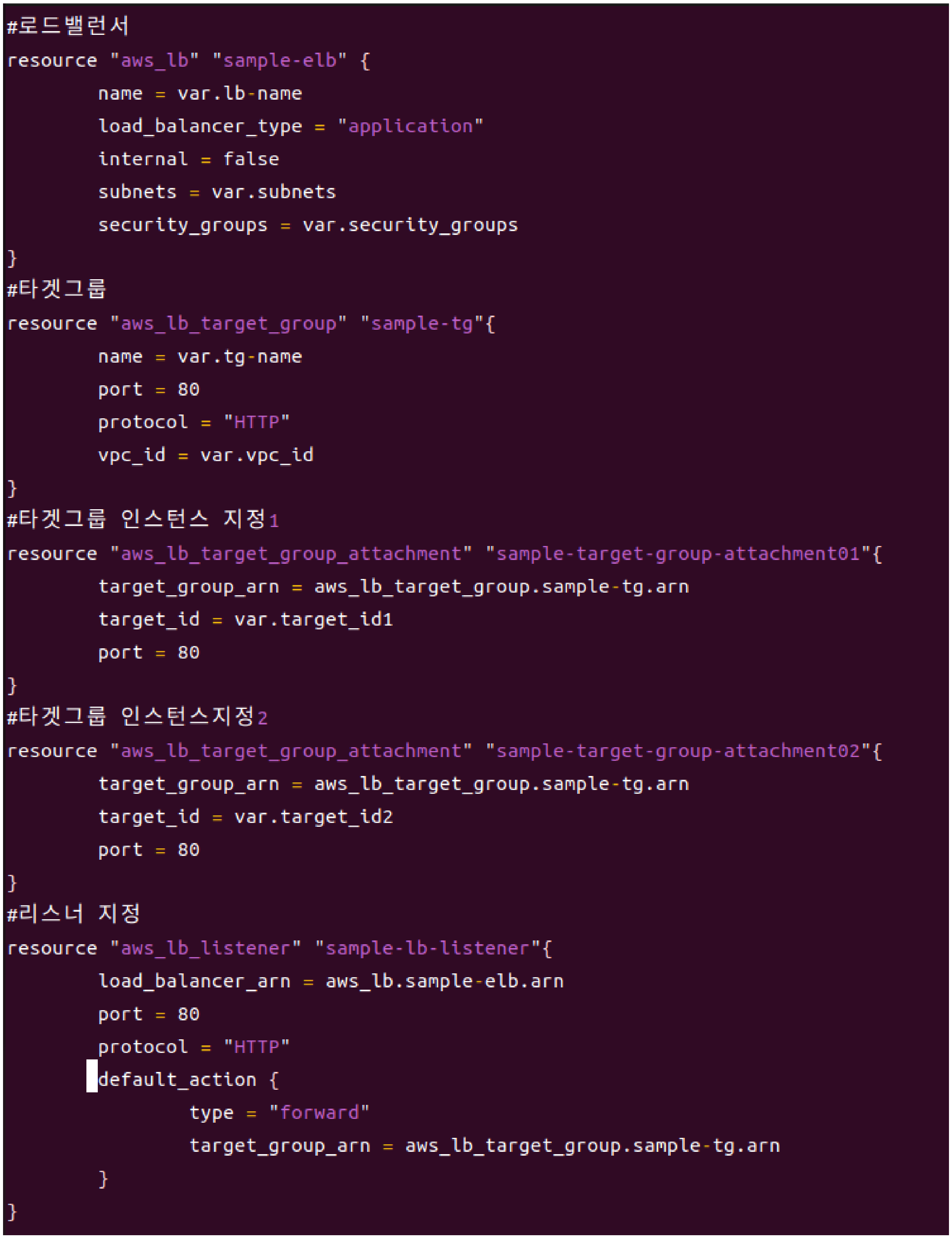
tf_project/main.tf 에서 사용할 키 변수들도 잊지말고 output 해준다.

5. 로드밸런서 모듈.

#vim tf_project/tf_lb/tf.main



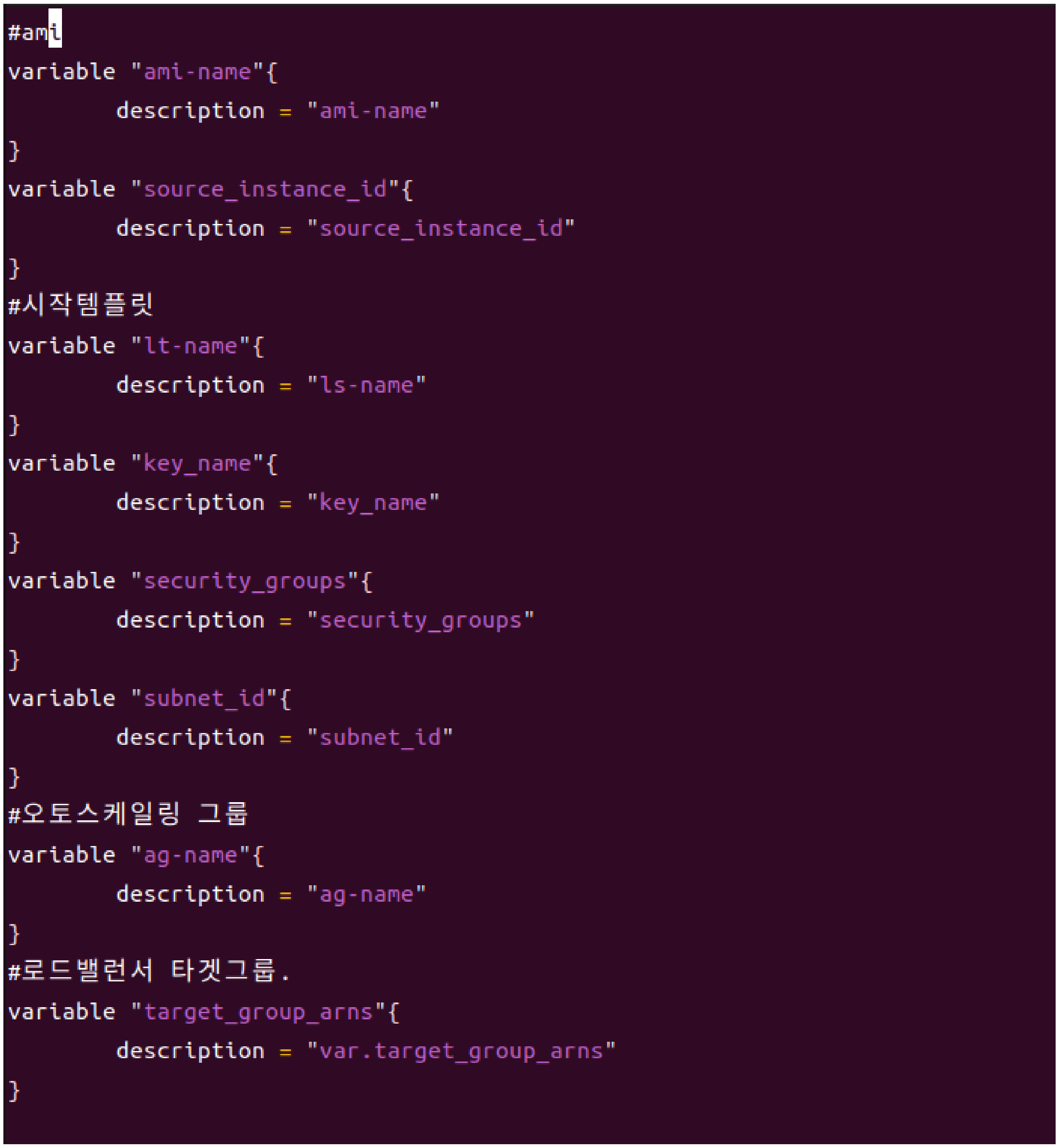
```
#로드밸런서 변수
variable "lb-name"{
        description = "lb-name"
}
variable "subnets" {
        description = "A list of subnets for the load balancer"
        type        = list(string)
}
variable "security_groups"{
        description = "security_groups"
        type = list(string)
}
#타겟그룹 변수
variable "tg-name"{
        description = "tg-name"
}
variable "vpc_id"{
        description = "vpc_id"
}
#타겟그룹인스턴스
variable "target_id1"{
        description = "target_id1"
}
#타겟그룹인스턴스
variable "target_id2"{
        description = "target_id2"
}
#오토스케일링 만들 때 필요한 타겟그룹arn
output "tg-arn"{
        value = aws_lb_target_group.sample-tg.arn
}
```



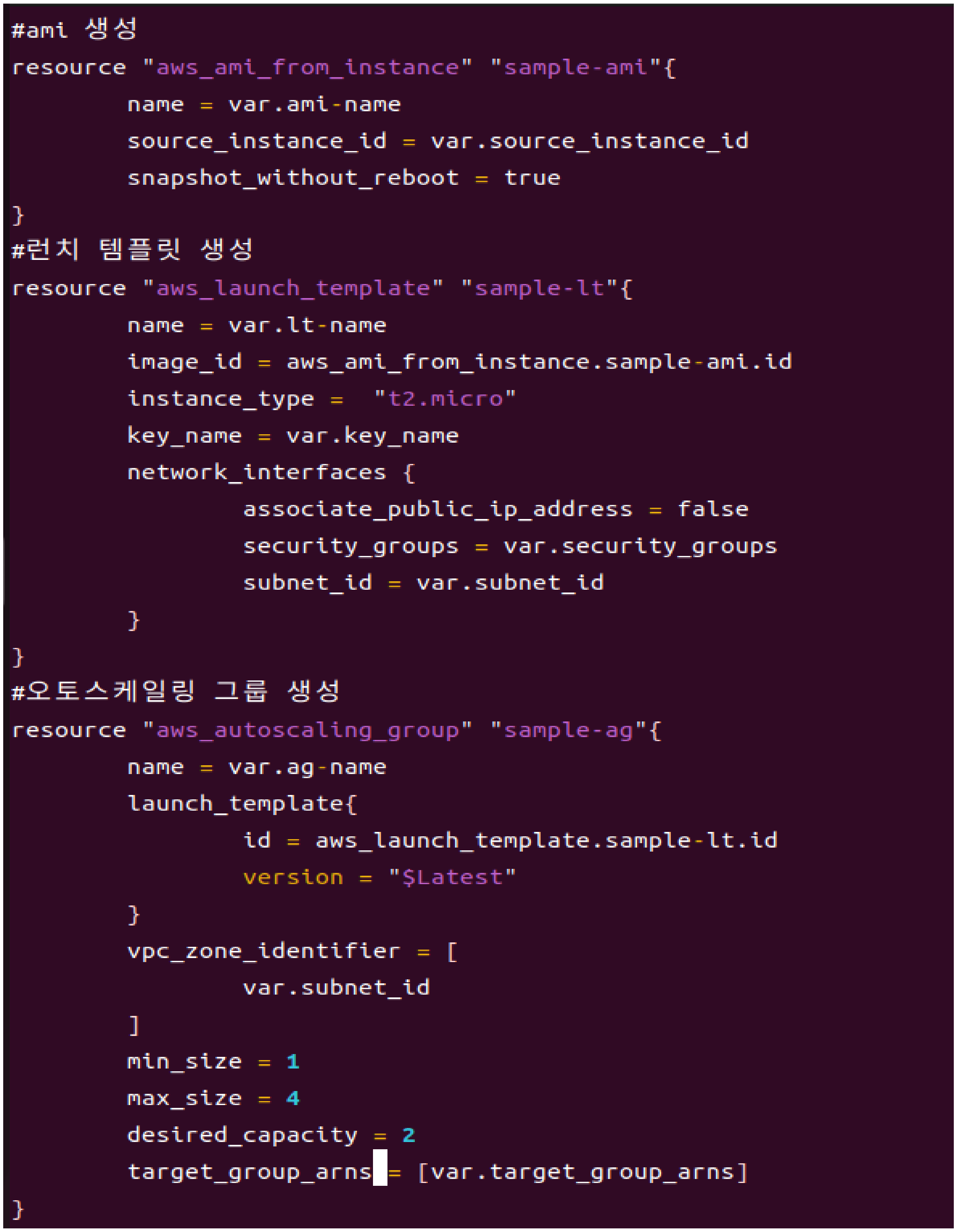
```
#로드밸런서
resource "aws_lb" "sample-elb" {
        name = var.lb-name
        load_balancer_type = "application"
        internal = false
        subnets = var.subnets
        security_groups = var.security_groups
}
#타겟그룹
resource "aws_lb_target_group" "sample-tg"{
        name = var.tg-name
        port = 80
        protocol = "HTTP"
        vpc_id = var.vpc_id
}
#타겟그룹 인스턴스 지정1
resource "aws_lb_target_group_attachment" "sample-target-group-attachment01"{
        target_group_arn = aws_lb_target_group.sample-tg.arn
        target_id = var.target_id1
        port = 80
}
#타겟그룹 인스턴스지정2
resource "aws_lb_target_group_attachment" "sample-target-group-attachment02"{
        target_group_arn = aws_lb_target_group.sample-tg.arn
        target_id = var.target_id2
        port = 80
}
#리스너 지정
resource "aws_lb_listener" "sample-lb-listener"{
        load_balancer_arn = aws_lb.sample-elb.arn
        port = 80
        protocol = "HTTP"
        default_action {
                type = "forward"
                target_group_arn = aws_lb_target_group.sample-tg.arn
        }
}
```

6. 오토스케일링 모듈.

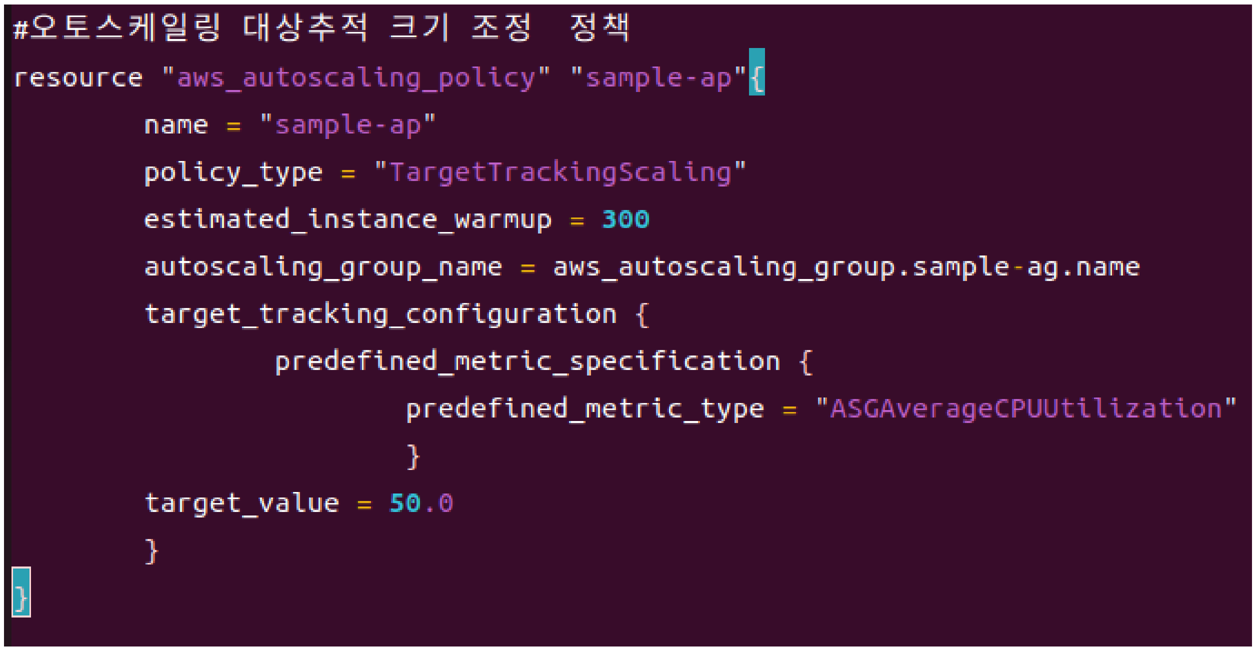
#vim tf_project/tf_as/tf.main



```
#ami
variable "ami-name"{
        description = "ami-name"
}
variable "source_instance_id"{
        description = "source_instance_id"
}
#시작템플릿
variable "lt-name"{
        description = "ls-name"
}
variable "key_name"{
        description = "key_name"
}
variable "security_groups"{
        description = "security_groups"
}
variable "subnet_id"{
        description = "subnet_id"
}
#오토스케일링 그룹
variable "ag-name"{
        description = "ag-name"
}
#로드밸런서 타겟그룹.
variable "target_group_arns"{
        description = "var.target_group_arns"
}
```



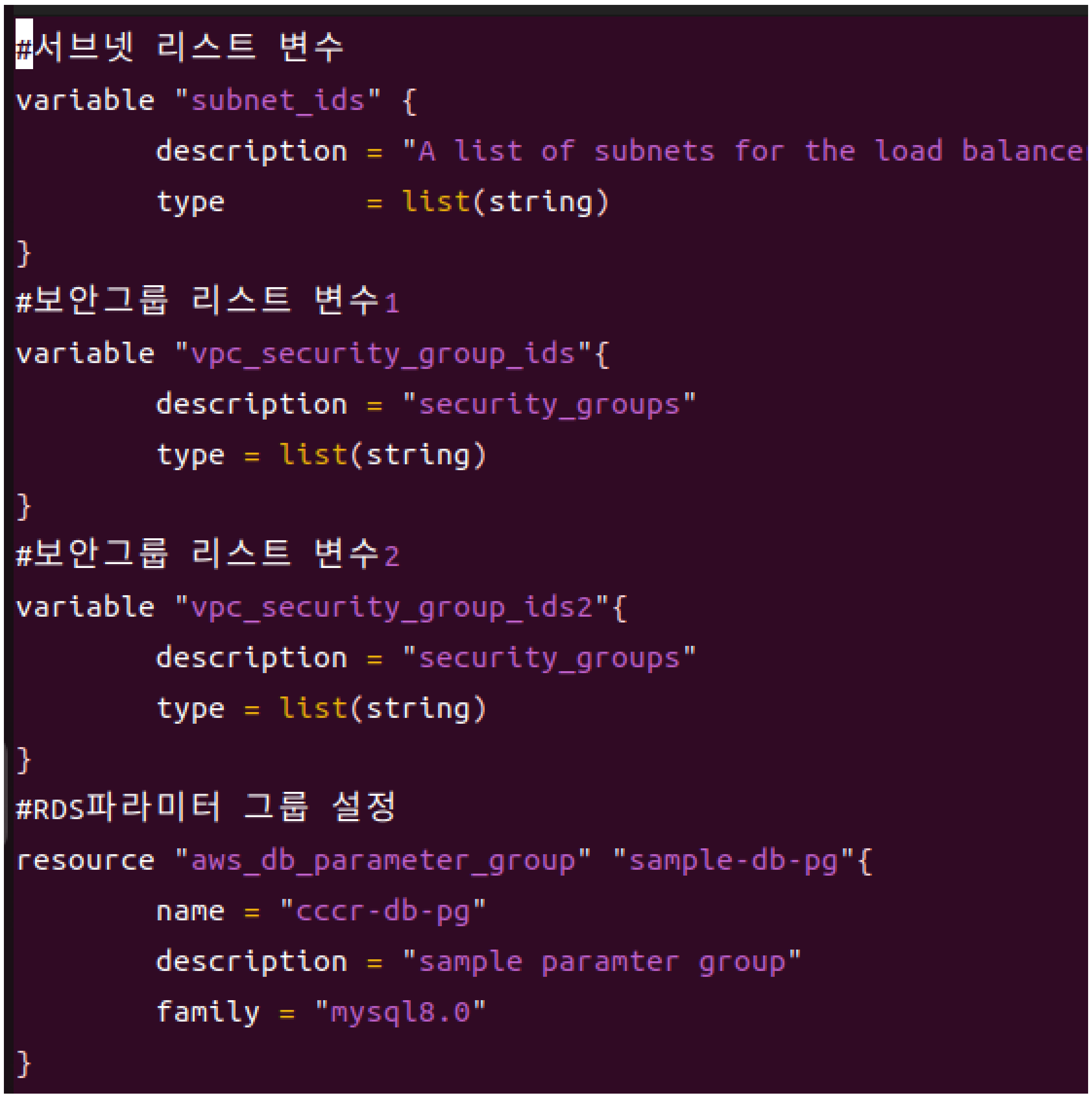
```
#ami 생성
resource "aws_ami_from_instance" "sample-ami"{
        name = var.ami-name
        source_instance_id = var.source_instance_id
        snapshot_without_reboot = true
}
#런치 템플릿 생성
resource "aws_launch_template" "sample-lt"{
        name = var.lt-name
        image_id = aws_ami_from_instance.sample-ami.id
        instance_type =  "t2.micro"
        key_name = var.key_name
        network_interfaces {
                associate_public_ip_address = false
                security_groups = var.security_groups
                subnet_id = var.subnet_id
        }
}
#오토스케일링 그룹 생성
resource "aws_autoscaling_group" "sample-ag"{
        name = var.ag-name
        launch_template{
                id = aws_launch_template.sample-lt.id
                version = "$Latest"
        }
        vpc_zone_identifier = [
                var.subnet_id
        ]
        min_size = 1
        max_size = 4
        desired_capacity = 2
        target_group_arns = [var.target_group_arns]
}
```



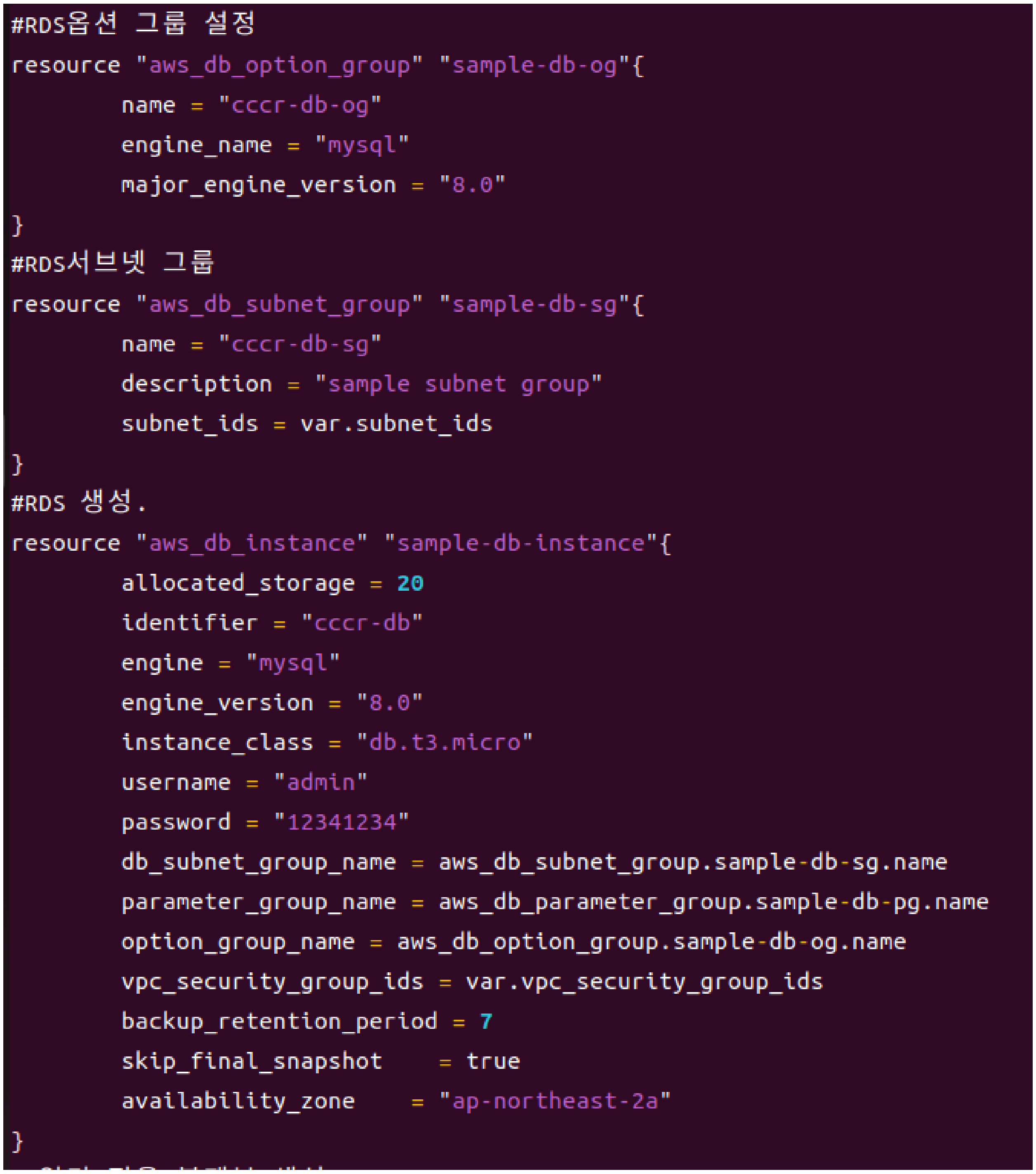
```
#오토스케일링 대상추적 크기 조정  정책
resource "aws_autoscaling_policy" "sample-ap"{
        name = "sample-ap"
        policy_type = "TargetTrackingScaling"
        estimated_instance_warmup = 300
        autoscaling_group_name = aws_autoscaling_group.sample-ag.name
        target_tracking_configuration {
                predefined_metric_specification {
                        predefined_metric_type = "ASGAverageCPUUtilization"
                        }
        target_value = 50.0
        }
}
```

7. RDS 모듈.

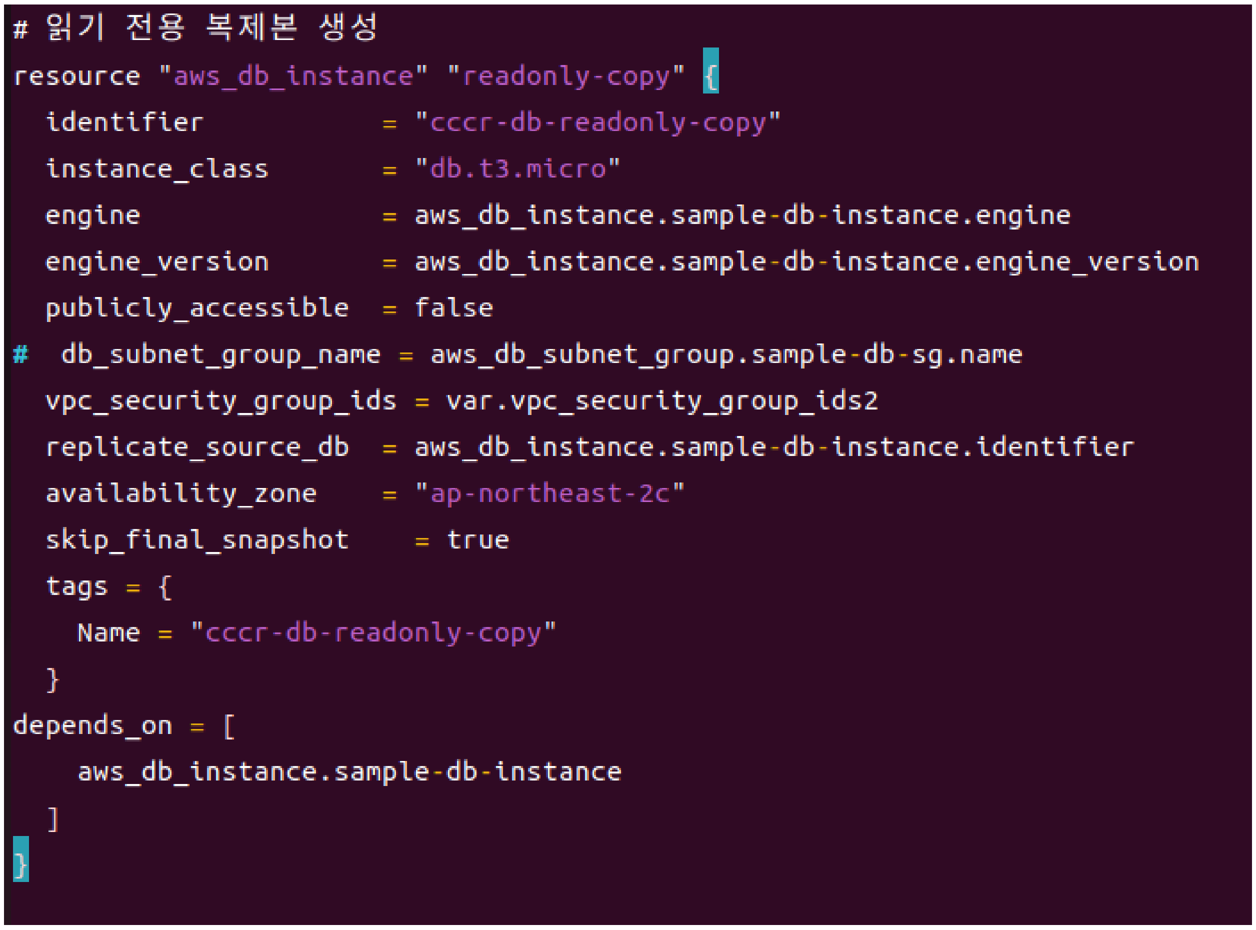
#vim tf_project/tf_rds/tf.main



```
#서브넷 리스트 변수
variable "subnet_ids" {
        description = "A list of subnets for the load balance
        type        = list(string)
}
#보안그룹 리스트 변수1
variable "vpc_security_group_ids"{
        description = "security_groups"
        type = list(string)
}
#보안그룹 리스트 변수2
variable "vpc_security_group_ids2"{
        description = "security_groups"
        type = list(string)
}
#RDS파라미터 그룹 설정
resource "aws_db_parameter_group" "sample-db-pg"{
        name = "cccr-db-pg"
        description = "sample paramter group"
        family = "mysql8.0"
}
```



```
#RDS옵션 그룹 설정
resource "aws_db_option_group" "sample-db-og"{
        name = "cccr-db-og"
        engine_name = "mysql"
        major_engine_version = "8.0"
}
#RDS서브넷 그룹
resource "aws_db_subnet_group" "sample-db-sg"{
        name = "cccr-db-sg"
        description = "sample subnet group"
        subnet_ids = var.subnet_ids
}
#RDS 생성.
resource "aws_db_instance" "sample-db-instance"{
        allocated_storage = 20
        identifier = "cccr-db"
        engine = "mysql"
        engine_version = "8.0"
        instance_class = "db.t3.micro"
        username = "admin"
        password = "12341234"
        db_subnet_group_name = aws_db_subnet_group.sample-db-sg.name
        parameter_group_name = aws_db_parameter_group.sample-db-pg.name
        option_group_name = aws_db_option_group.sample-db-og.name
        vpc_security_group_ids = var.vpc_security_group_ids
        backup_retention_period = 7
        skip_final_snapshot    = true
        availability_zone    = "ap-northeast-2a"
}
```

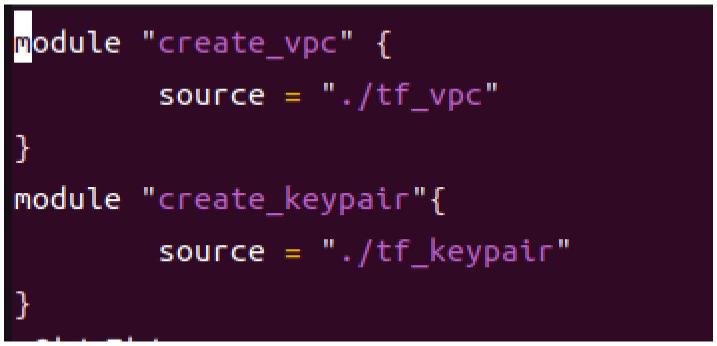


```
# 읽기 전용 복제본 생성
resource "aws_db_instance" "readonly-copy" {
  identifier           = "cccr-db-readonly-copy"
  instance_class       = "db.t3.micro"
  engine               = aws_db_instance.sample-db-instance.engine
  engine_version       = aws_db_instance.sample-db-instance.engine_version
  publicly_accessible  = false
#  db_subnet_group_name = aws_db_subnet_group.sample-db-sg.name
  vpc_security_group_ids = var.vpc_security_group_ids2
  replicate_source_db  = aws_db_instance.sample-db-instance.identifier
  availability_zone    = "ap-northeast-2c"
  skip_final_snapshot    = true
  tags = {
    Name = "cccr-db-readonly-copy"
  }
depends_on = [
    aws_db_instance.sample-db-instance
  ]
}
```

8. tf_project/main.tf

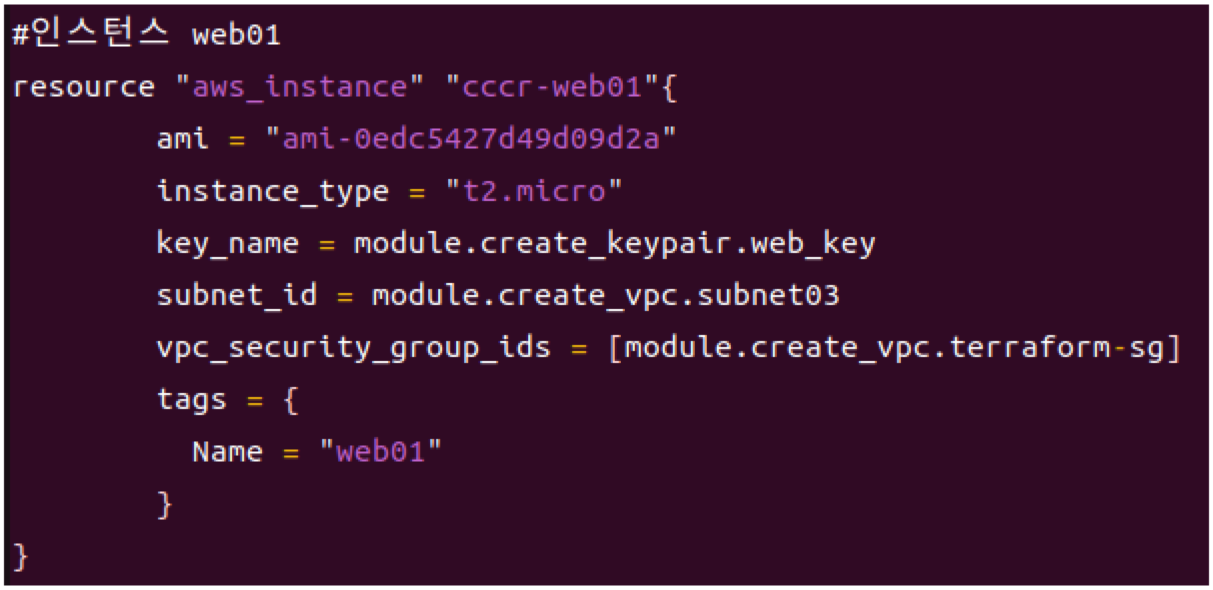
#vim tf_project/main.tf

1. vpc 와 keypair 모듈을 실행한다.

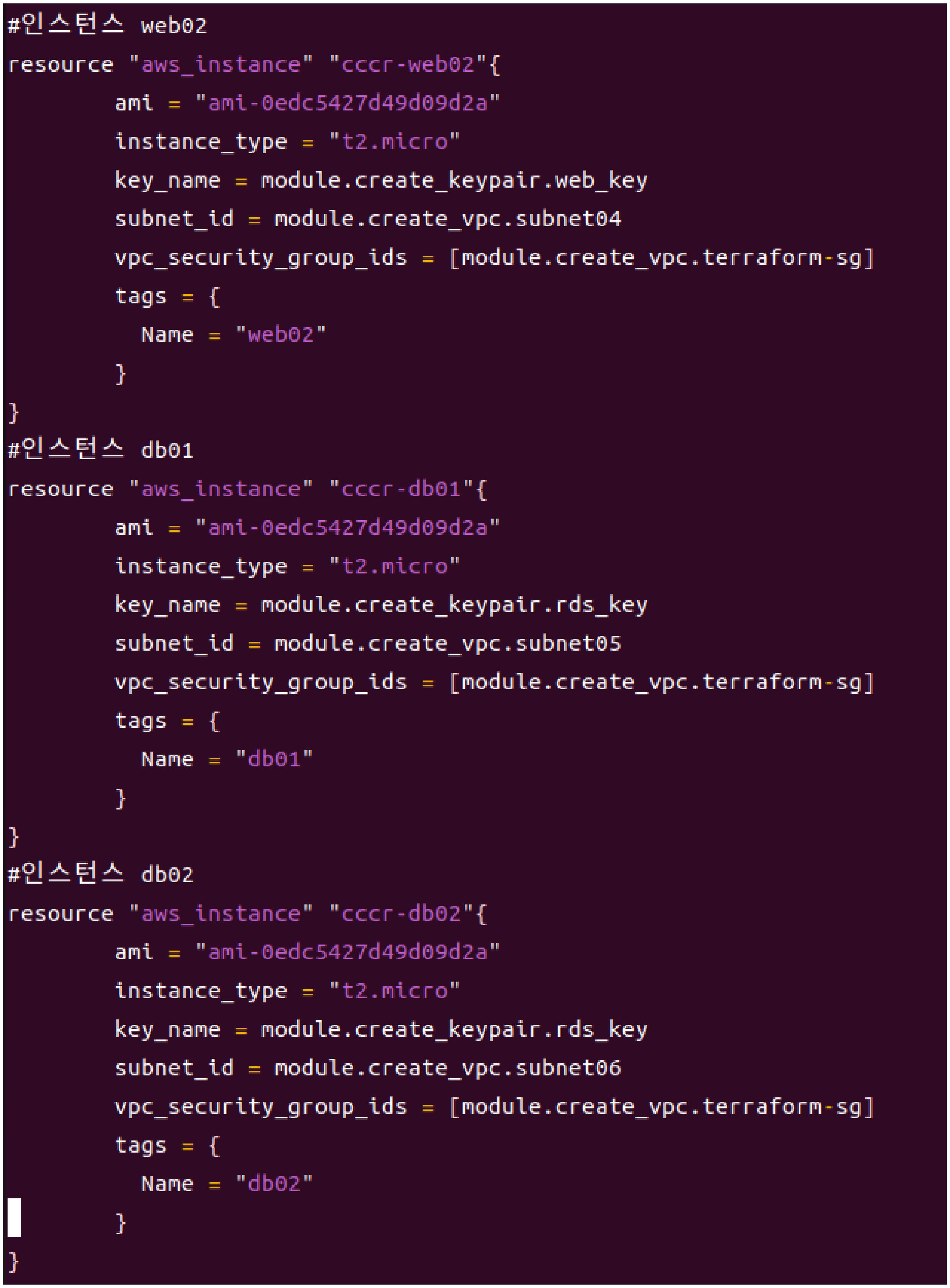


```
module "create_vpc" {
        source = "./tf_vpc"
}
module "create_keypair"{
        source = "./tf_keypair"
}
```

2. 인스턴스를 생성한다.

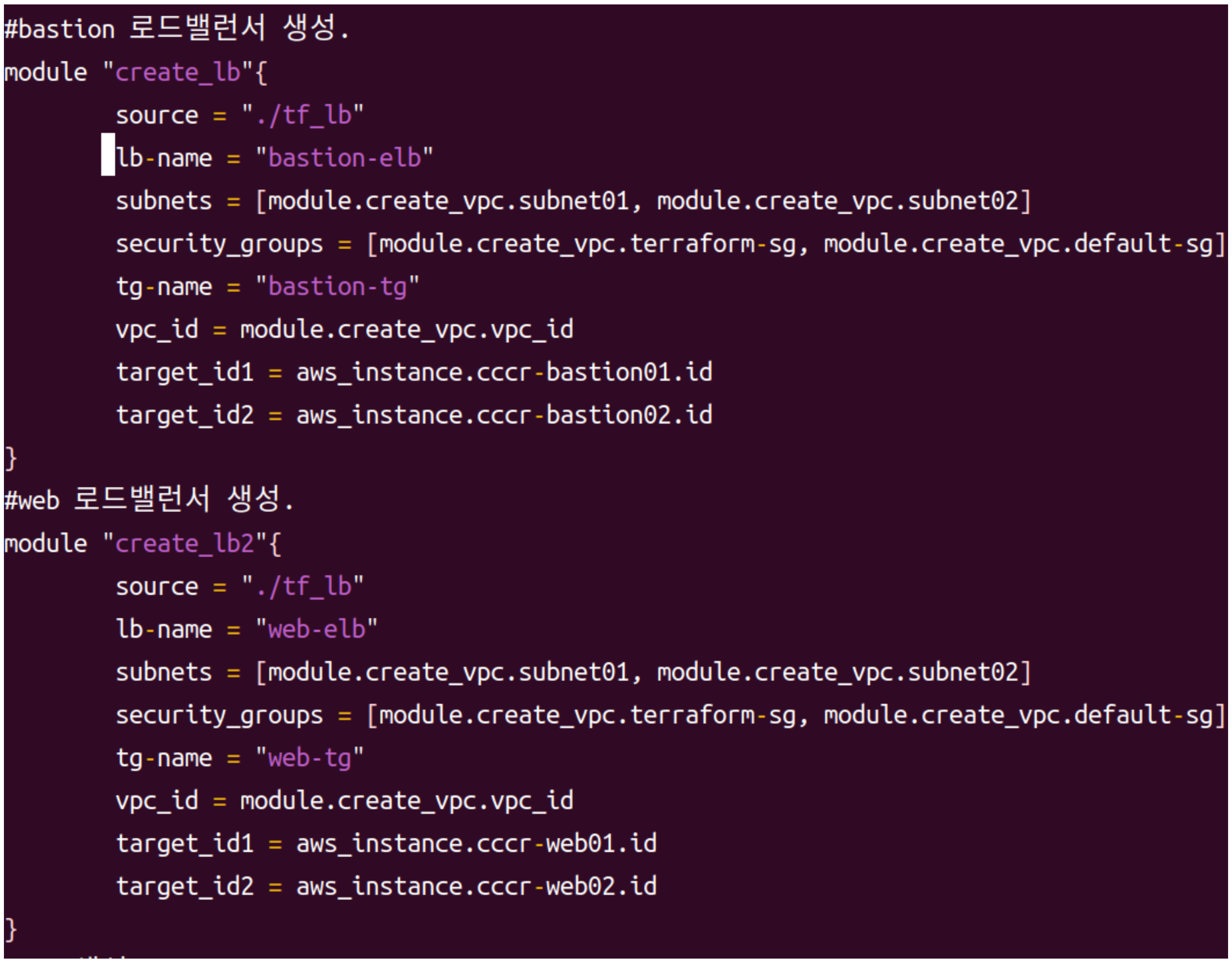


```
#인스턴스 web01
resource "aws_instance" "cccr-web01"{
        ami = "ami-0edc5427d49d09d2a"
        instance_type = "t2.micro"
        key_name = module.create_keypair.web_key
        subnet_id = module.create_vpc.subnet03
        vpc_security_group_ids = [module.create_vpc.terraform-sg]
        tags = {
          Name = "web01"
        }
}
```



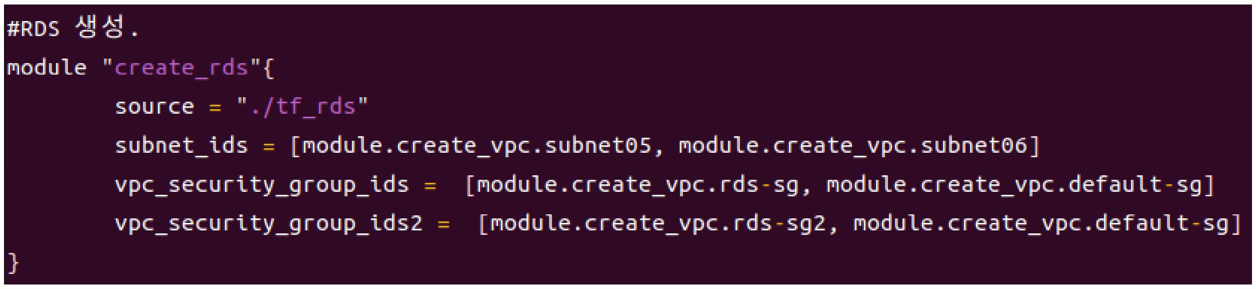
```
#인스턴스 web02
resource "aws_instance" "cccr-web02"{
        ami = "ami-0edc5427d49d09d2a"
        instance_type = "t2.micro"
        key_name = module.create_keypair.web_key
        subnet_id = module.create_vpc.subnet04
        vpc_security_group_ids = [module.create_vpc.terraform-sg]
        tags = {
          Name = "web02"
        }
}
#인스턴스 db01
resource "aws_instance" "cccr-db01"{
        ami = "ami-0edc5427d49d09d2a"
        instance_type = "t2.micro"
        key_name = module.create_keypair.rds_key
        subnet_id = module.create_vpc.subnet05
        vpc_security_group_ids = [module.create_vpc.terraform-sg]
        tags = {
          Name = "db01"
        }
}
#인스턴스 db02
resource "aws_instance" "cccr-db02"{
        ami = "ami-0edc5427d49d09d2a"
        instance_type = "t2.micro"
        key_name = module.create_keypair.rds_key
        subnet_id = module.create_vpc.subnet06
        vpc_security_group_ids = [module.create_vpc.terraform-sg]
        tags = {
          Name = "db02"
        }
}
```

3. 로드밸런서 생성.



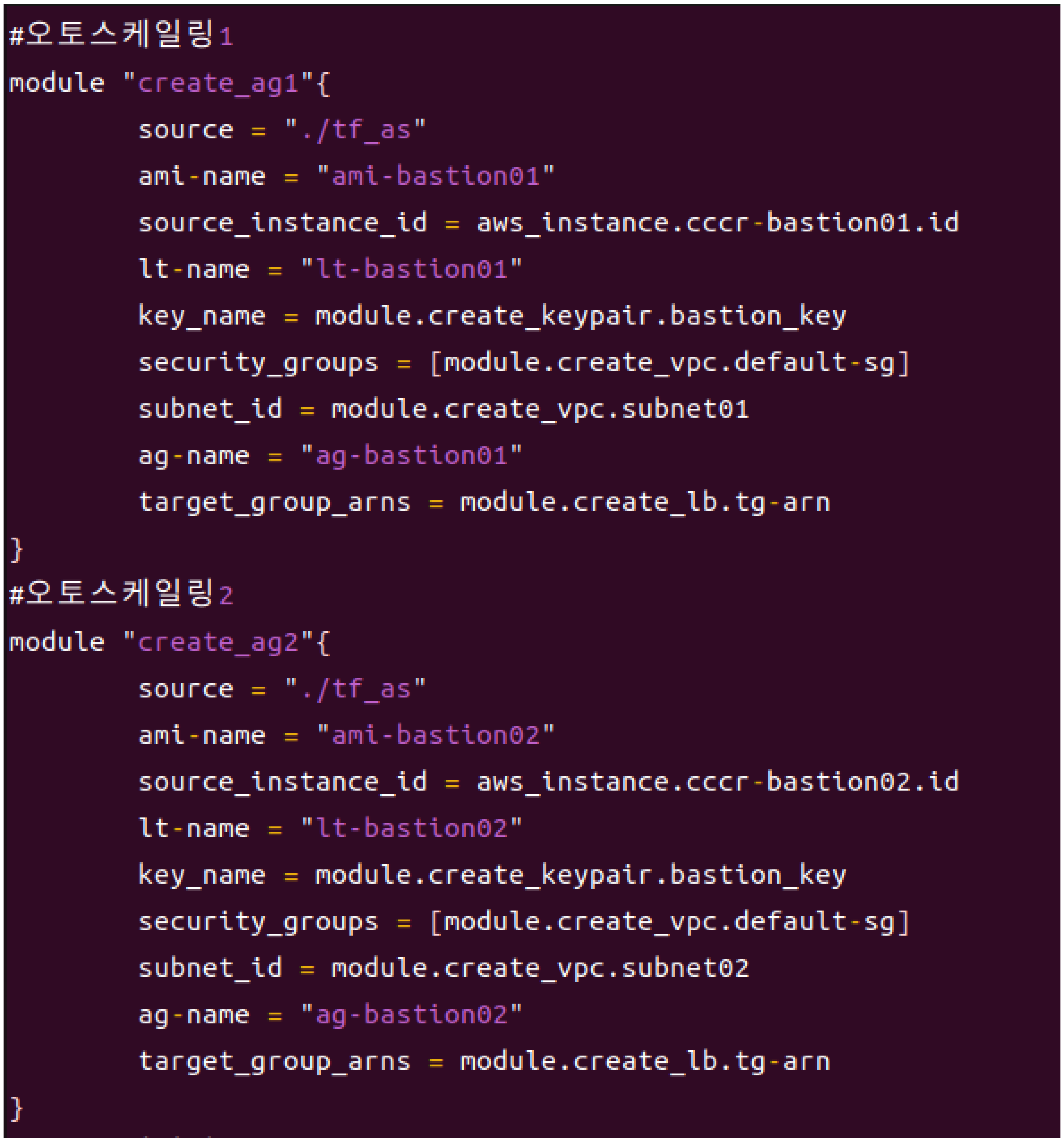
```
#bastion 로드밸런서 생성.
module "create_lb"{
        source = "./tf_lb"
        lb-name = "bastion-elb"
        subnets = [module.create_vpc.subnet01, module.create_vpc.subnet02]
        security_groups = [module.create_vpc.terraform-sg, module.create_vpc.default-sg]
        tg-name = "bastion-tg"
        vpc_id = module.create_vpc.vpc_id
        target_id1 = aws_instance.cccr-bastion01.id
        target_id2 = aws_instance.cccr-bastion02.id
}
#web 로드밸런서 생성.
module "create_lb2"{
        source = "./tf_lb"
        lb-name = "web-elb"
        subnets = [module.create_vpc.subnet01, module.create_vpc.subnet02]
        security_groups = [module.create_vpc.terraform-sg, module.create_vpc.default-sg]
        tg-name = "web-tg"
        vpc_id = module.create_vpc.vpc_id
        target_id1 = aws_instance.cccr-web01.id
        target_id2 = aws_instance.cccr-web02.id
}
```

4. rds 생성.

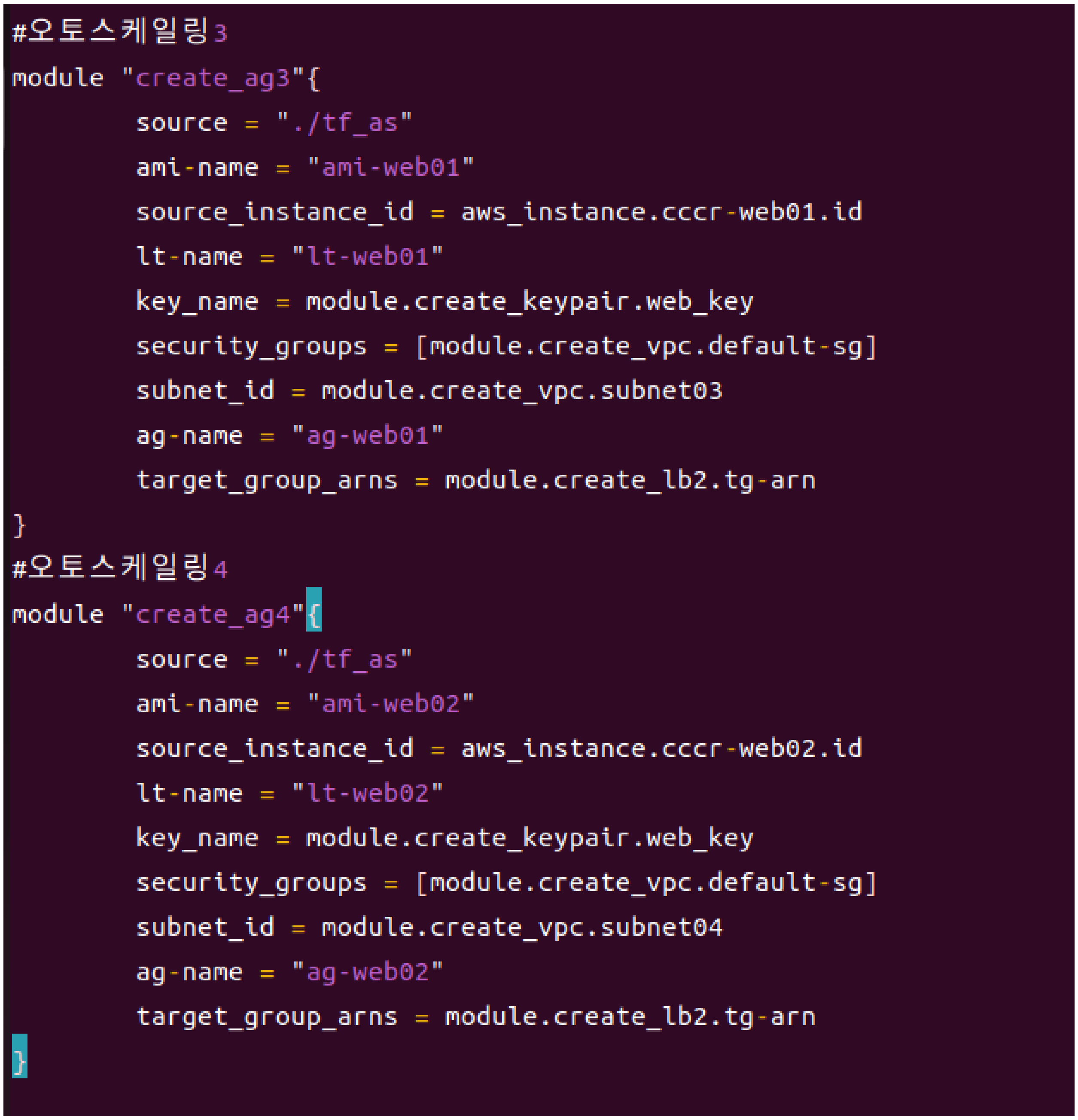


```
#RDS 생성.
module "create_rds"{
        source = "./tf_rds"
        subnet_ids = [module.create_vpc.subnet05, module.create_vpc.subnet06]
        vpc_security_group_ids =  [module.create_vpc.rds-sg, module.create_vpc.default-sg]
        vpc_security_group_ids2 =  [module.create_vpc.rds-sg2, module.create_vpc.default-sg]
}
```

5. 오토스케일링



```
#오토스케일링1
module "create_ag1"{
        source = "./tf_as"
        ami-name = "ami-bastion01"
        source_instance_id = aws_instance.cccr-bastion01.id
        lt-name = "lt-bastion01"
        key_name = module.create_keypair.bastion_key
        security_groups = [module.create_vpc.default-sg]
        subnet_id = module.create_vpc.subnet01
        ag-name = "ag-bastion01"
        target_group_arns = module.create_lb.tg-arn
}
#오토스케일링2
module "create_ag2"{
        source = "./tf_as"
        ami-name = "ami-bastion02"
        source_instance_id = aws_instance.cccr-bastion02.id
        lt-name = "lt-bastion02"
        key_name = module.create_keypair.bastion_key
        security_groups = [module.create_vpc.default-sg]
        subnet_id = module.create_vpc.subnet02
        ag-name = "ag-bastion02"
        target_group_arns = module.create_lb.tg-arn
}
```



```
#오토스케일링3
module "create_ag3"{
        source = "./tf_as"
        ami-name = "ami-web01"
        source_instance_id = aws_instance.cccr-web01.id
        lt-name = "lt-web01"
        key_name = module.create_keypair.web_key
        security_groups = [module.create_vpc.default-sg]
        subnet_id = module.create_vpc.subnet03
        ag-name = "ag-web01"
        target_group_arns = module.create_lb2.tg-arn
}
#오토스케일링4
module "create_ag4"{
        source = "./tf_as"
        ami-name = "ami-web02"
        source_instance_id = aws_instance.cccr-web02.id
        lt-name = "lt-web02"
        key_name = module.create_keypair.web_key
        security_groups = [module.create_vpc.default-sg]
        subnet_id = module.create_vpc.subnet04
        ag-name = "ag-web02"
        target_group_arns = module.create_lb2.tg-arn
}
```

6. terraform.tfstate 수정

“skip_final_snapshot” : true 로 수정.

(수정 하지 않으면 rds 삭제할 때 오류 발생.)

```
"skip_final_snapshot": true,
```

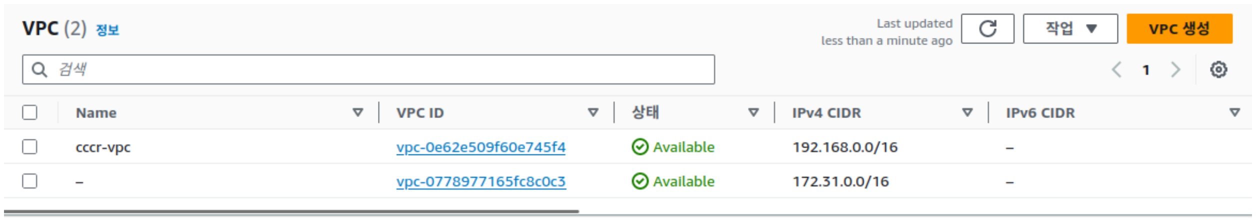
9. 결과

#terraform init

#terraform plan

#terraform apply

1. vpc

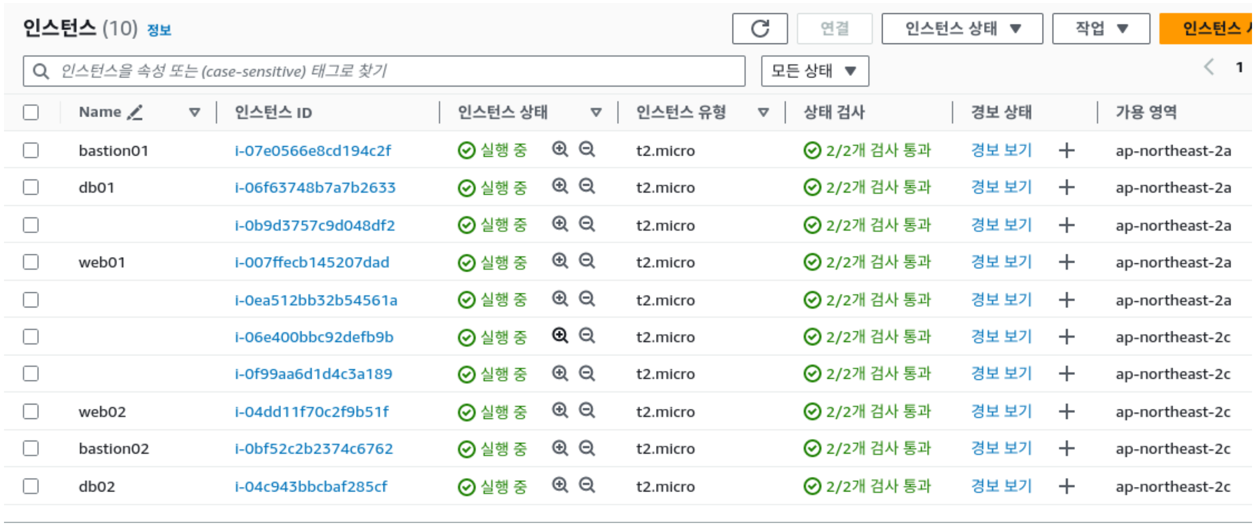


VPC (2) 정보

Last updated less than a minute ago

Name	VPC ID	상태	IPv4 CIDR	IPv6 CIDR
cccr-vpc	vpc-0e62e509f60e745f4	Available	192.168.0.0/16	–
–	vpc-0778977165fc8c0c3	Available	172.31.0.0/16	–

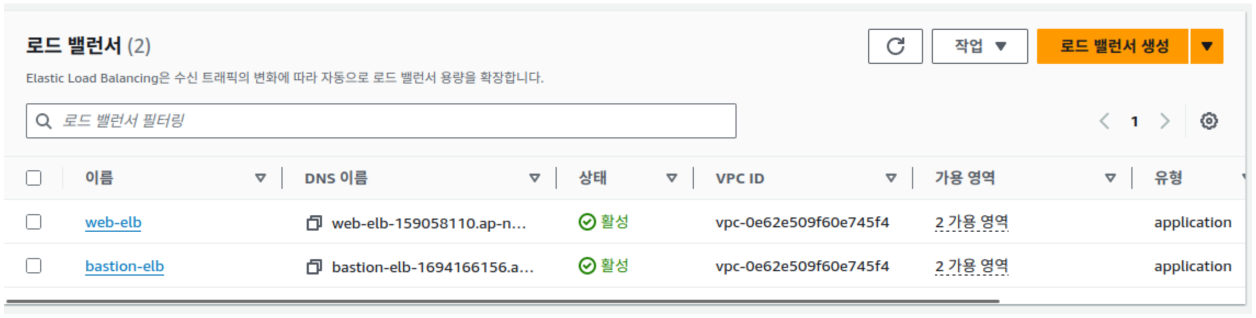
2. 인스턴스



인스턴스 (10) 정보

Name	인스턴스 ID	인스턴스 상태	인스턴스 유형	상태 검사	경보 상태	가용 영역
bastion01	i-07e0566e8cd194c2f	실행 중	t2.micro	2/2개 검사 통과	경보 보기	ap-northeast-2a
db01	i-06f63748b7a7b2633	실행 중	t2.micro	2/2개 검사 통과	경보 보기	ap-northeast-2a
	i-0b9d3757c9d048df2	실행 중	t2.micro	2/2개 검사 통과	경보 보기	ap-northeast-2a
web01	i-007ffecb145207dad	실행 중	t2.micro	2/2개 검사 통과	경보 보기	ap-northeast-2a
	i-0ea512bb32b54561a	실행 중	t2.micro	2/2개 검사 통과	경보 보기	ap-northeast-2a
	i-06e400bbc92defb9b	실행 중	t2.micro	2/2개 검사 통과	경보 보기	ap-northeast-2c
	i-0f99aa6d1d4c3a189	실행 중	t2.micro	2/2개 검사 통과	경보 보기	ap-northeast-2c
web02	i-04dd11f70c2f9b51f	실행 중	t2.micro	2/2개 검사 통과	경보 보기	ap-northeast-2c
bastion02	i-0bf52c2b2374c6762	실행 중	t2.micro	2/2개 검사 통과	경보 보기	ap-northeast-2c
db02	i-04c943bbcbaf285cf	실행 중	t2.micro	2/2개 검사 통과	경보 보기	ap-northeast-2c

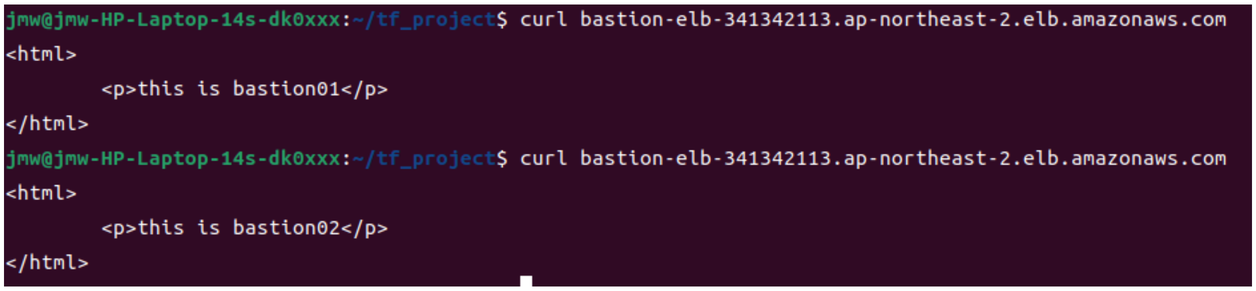
3. 로드 밸런싱



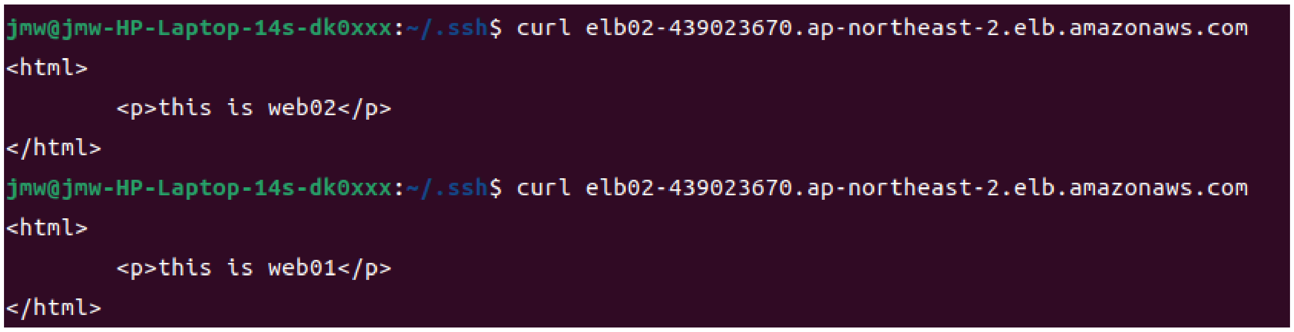
로드 밸런서 (2)

Elastic Load Balancing은 수신 트래픽의 변화에 따라 자동으로 로드 밸런서 용량을 확장합니다.

이름	DNS 이름	상태	VPC ID	가용 영역	유형
web-elb	web-elb-159058110.ap-n...	활성	vpc-0e62e509f60e745f4	2 가용 영역	application
bastion-elb	bastion-elb-1694166156.a...	활성	vpc-0e62e509f60e745f4	2 가용 영역	application

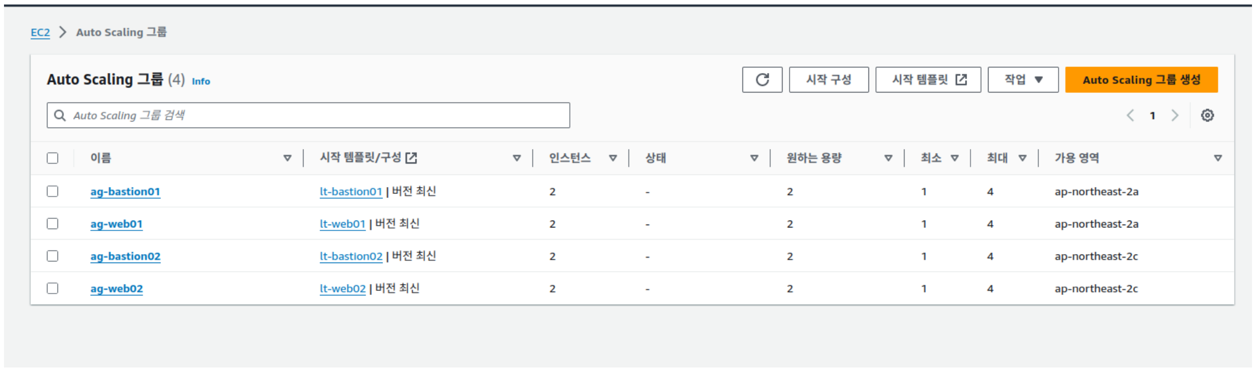


```
jmw@jmw-HP-Laptop-14s-dk0xxx:~/tf_project$ curl bastion-elb-341342113.ap-northeast-2.elb.amazonaws.com
<html>
        <p>this is bastion01</p>
</html>
jmw@jmw-HP-Laptop-14s-dk0xxx:~/tf_project$ curl bastion-elb-341342113.ap-northeast-2.elb.amazonaws.com
<html>
        <p>this is bastion02</p>
</html>
```



```
jmw@jmw-HP-Laptop-14s-dk0xxx:~/.ssh$ curl elb02-439023670.ap-northeast-2.elb.amazonaws.com
<html>
        <p>this is web02</p>
</html>
jmw@jmw-HP-Laptop-14s-dk0xxx:~/.ssh$ curl elb02-439023670.ap-northeast-2.elb.amazonaws.com
<html>
        <p>this is web01</p>
</html>
```

4. 오토스케일링

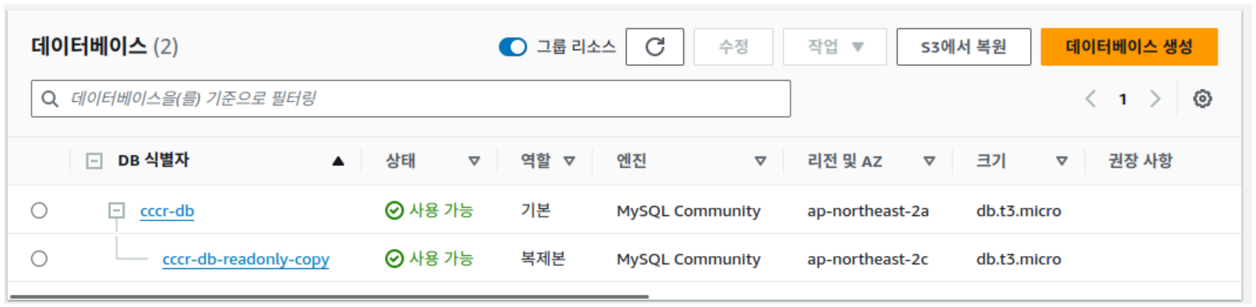


EC2 > Auto Scaling 그룹

Auto Scaling 그룹 (4) Info

이름	시작 템플릿/구성	인스턴스	상태	원하는 용량	최소	최대	가용 영역
ag-bastion01	lt-bastion01 | 버전 최신	2	-	2	1	4	ap-northeast-2a
ag-web01	lt-web01 | 버전 최신	2	-	2	1	4	ap-northeast-2a
ag-bastion02	lt-bastion02 | 버전 최신	2	-	2	1	4	ap-northeast-2c
ag-web02	lt-web02 | 버전 최신	2	-	2	1	4	ap-northeast-2c

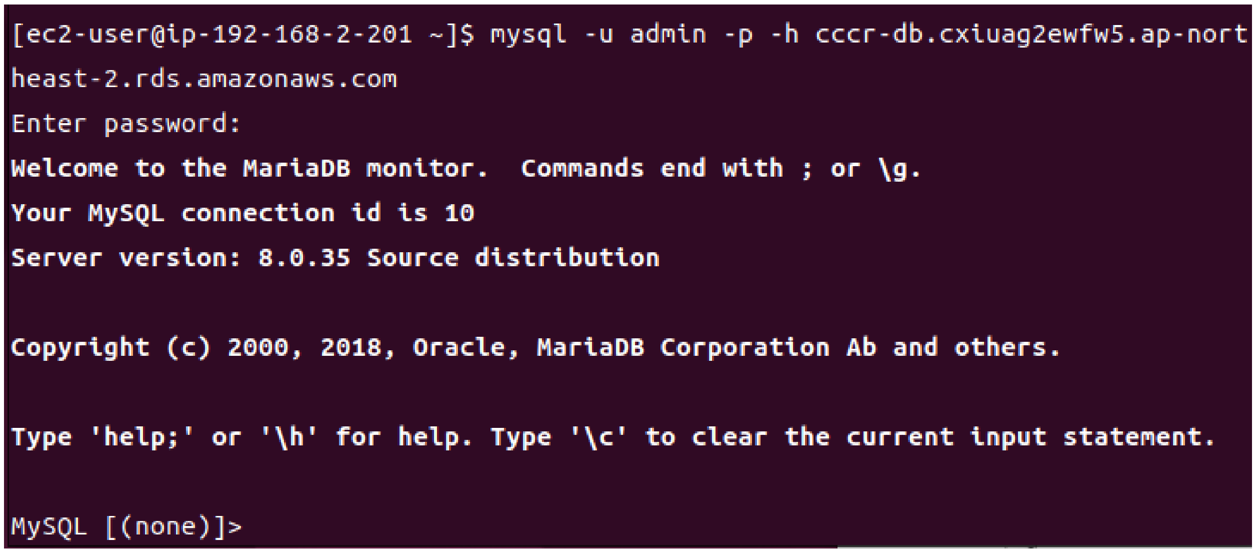
5. RDS



데이터베이스 (2)

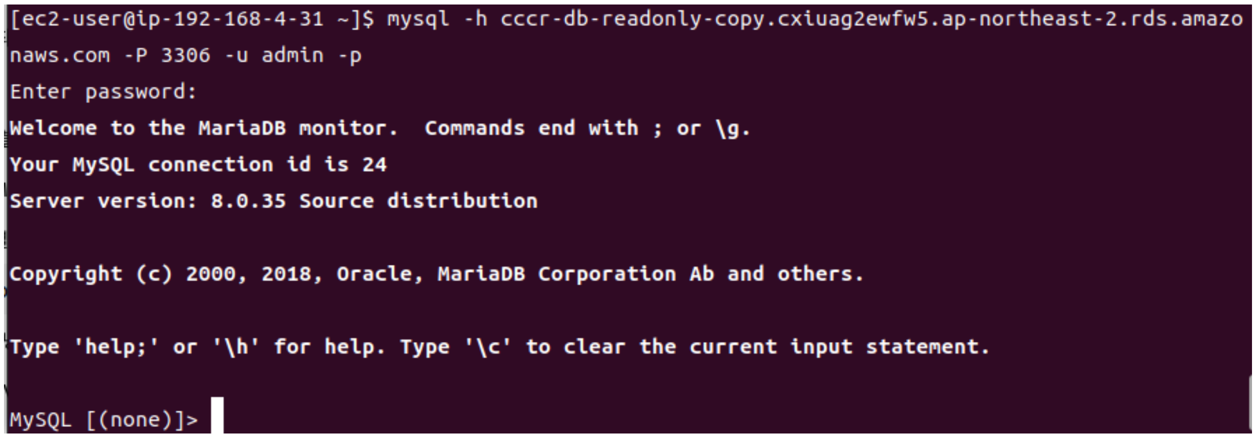
DB 식별자	상태	역할	엔진	리전 및 AZ	크기	권장 사항
cccr-db	사용 가능	기본	MySQL Community	ap-northeast-2a	db.t3.micro	
cccr-db-readonly-copy	사용 가능	복제본	MySQL Community	ap-northeast-2c	db.t3.micro	

web01 에서 rds 접속.



```
[ec2-user@ip-192-168-2-201 ~]$ mysql -u admin -p -h cccr-db.cxiuag2ewfw5.ap-nort
heast-2.rds.amazonaws.com
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 10
Server version: 8.0.35 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]>
```

db01 에서 Read Replica 접속.



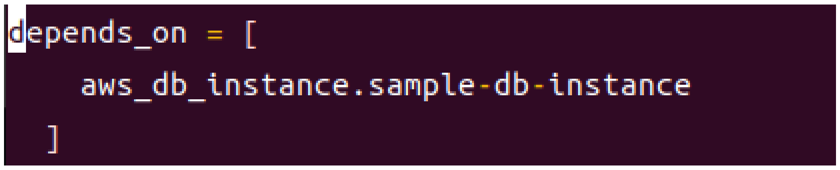
```
[ec2-user@ip-192-168-4-31 ~]$ mysql -h cccr-db-readonly-copy.cxiuag2ewfw5.ap-northeast-2.rds.amazo
naws.com -P 3306 -u admin -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 24
Server version: 8.0.35 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]>
```

10. 문제 및 해결

문제는 대부분 Read Replica 를 생성할 때 발생했다.

1. 의존성 문제

Replica 를 생성하기 전에 RDS 가 먼저 생성되어야 하는데 테라폼은 RDS 생성유무를 판단하지 않고 바로 Replica 실행에 들어가서 생기는 에러가 발생했다.



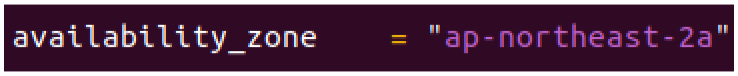
```
depends_on = [
    aws_db_instance.sample-db-instance
  ]
```

depends_on 을 추가하여 RDS 가 생성된 후에 Replica 를 생성하게 만들었다.

2. 가용영역 문제.

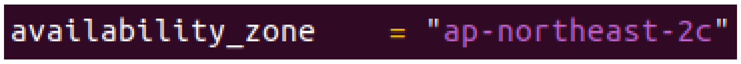
RDS 와 Replica 는 같은 서브넷영역을 공유하기 때문에 테라폼을 실행하면 같은 가용영역을 공유한다는 에러가 발생했다.

RDS 에 가용영역 선택.



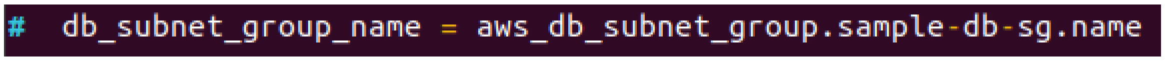
```
availability_zone     = "ap-northeast-2a"
```

Replica 에 가용영역 선택.



```
availability_zone     = "ap-northeast-2c"
```

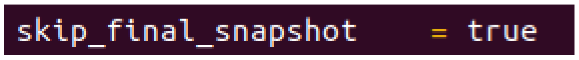
Replica 서브넷 그룹 주석처리.



```
#  db_subnet_group_name = aws_db_subnet_group.sample-db-sg.name
```

3. terraform 수정 후. 다시 apply 하면 스냅샷을 생성하지 못했다는 오류가 발생.

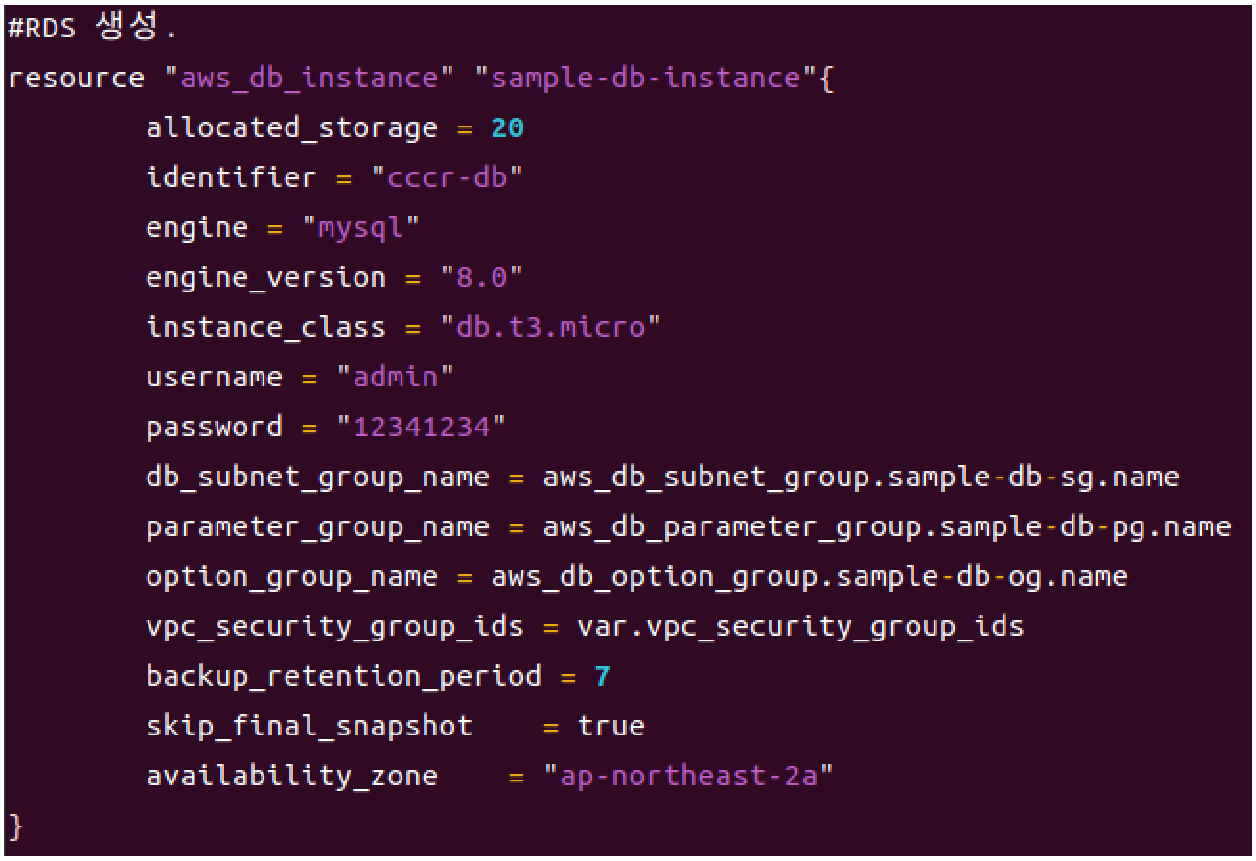
RDS 에 스냅샷 생성을 스킵하는 명령어 추가.



```
skip_final_snapshot     = true
```

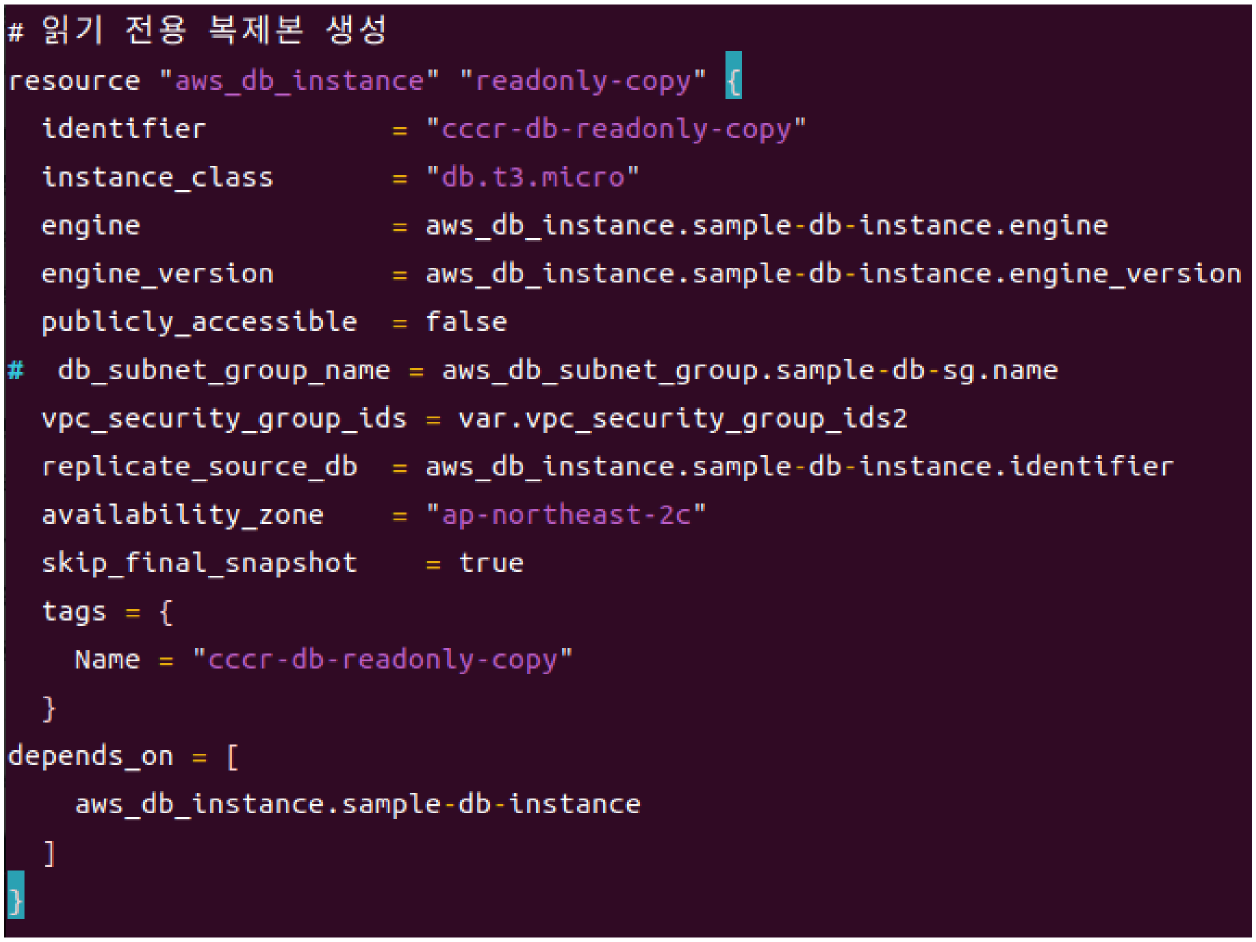
4. 완성본.

RDS



```
#RDS 생성.
resource "aws_db_instance" "sample-db-instance"{
        allocated_storage = 20
        identifier = "cccr-db"
        engine = "mysql"
        engine_version = "8.0"
        instance_class = "db.t3.micro"
        username = "admin"
        password = "12341234"
        db_subnet_group_name = aws_db_subnet_group.sample-db-sg.name
        parameter_group_name = aws_db_parameter_group.sample-db-pg.name
        option_group_name = aws_db_option_group.sample-db-og.name
        vpc_security_group_ids = var.vpc_security_group_ids
        backup_retention_period = 7
        skip_final_snapshot    = true
        availability_zone    = "ap-northeast-2a"
}
```

Replica



```
# 읽기 전용 복제본 생성
resource "aws_db_instance" "readonly-copy" {
  identifier             = "cccr-db-readonly-copy"
  instance_class         = "db.t3.micro"
  engine                 = aws_db_instance.sample-db-instance.engine
  engine_version         = aws_db_instance.sample-db-instance.engine_version
  publicly_accessible    = false
#  db_subnet_group_name  = aws_db_subnet_group.sample-db-sg.name
  vpc_security_group_ids = var.vpc_security_group_ids2
  replicate_source_db    = aws_db_instance.sample-db-instance.identifier
  availability_zone      = "ap-northeast-2c"
  skip_final_snapshot      = true
  tags = {
    Name = "cccr-db-readonly-copy"
  }
depends_on = [
    aws_db_instance.sample-db-instance
  ]
}
```