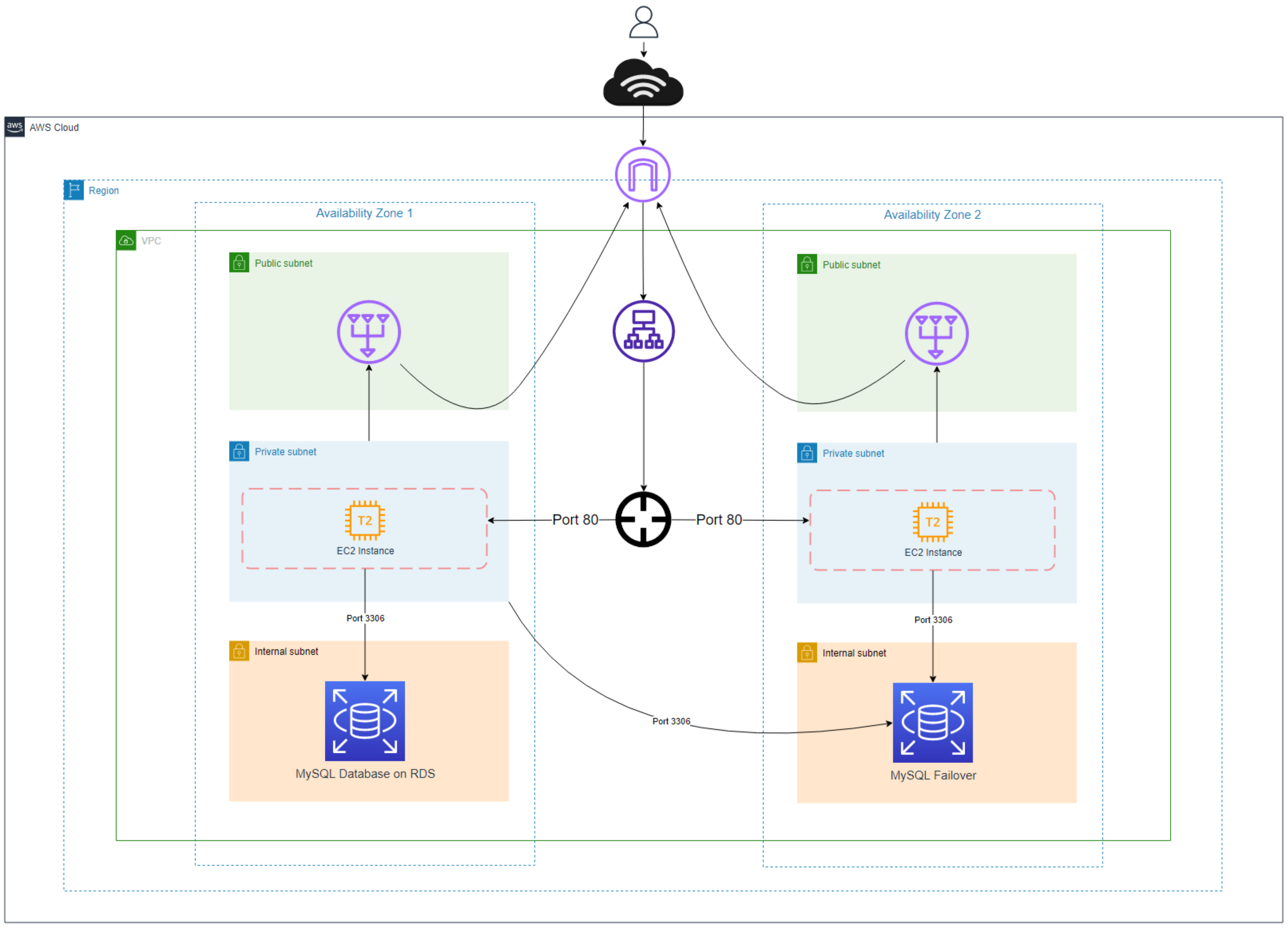
HashiCorp
Terraform

Deployment 09 - Terraform

For this deployment, the objective was to use Terraform to deploy resources on AWS Cloud. Terraform is an open source infrastructure as code tool created by Hashicorp. Terraform allows us to write and execute code to define. We can also deploy, update, and destroy infractures. The syntax that Terraform uses is called HashiCorp Configuration Language (HCL). This syntax strikes a balance between human readable, editable, and being machine-friendly.

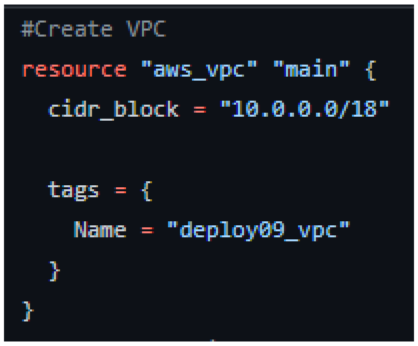


There are many benefits for Terraform. Terraform can make binary calls on your behalf to one or more providers such as AWS, Azure, GCP, etc. Terraform allows us to increase speed in development, decrease human error, reusability, scalable, self documentation, and be version controlled easily. Terraform also cuts down on ClickOps by allowing users to run terraform files from the command line to provision our resources.

We specifically had to architect a VPC that consisted of subnets, EC2, security Groups, RDS MySQL database, application load balancer, and a target group. The simple flow of traffic would be users would access the internet and it would hit an internet gateway. Traffic will then be sent to the load balancer that interacts with the target group. Traffic will be directed to an EC2 instance at a specific port. The EC2 instance will then be able to interact with the RDS MySQL database. For this deployment, the only thing that I have changed was the type of database to create. I use MySQL databases more than PostgreSQL.

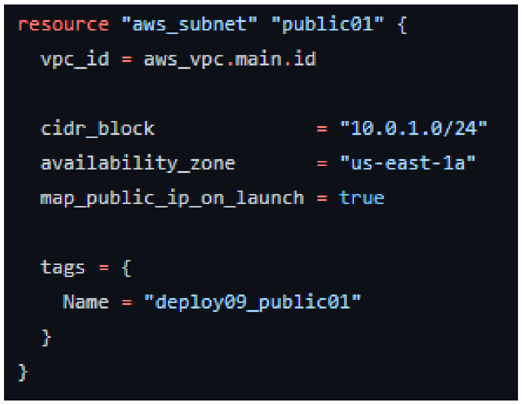
Part 1 - VPC

The first thing we need to create is the VPC. VPCs are virtual private clouds which are essentially isolated networks. Inside of my vpc.tf file, I am using the resource module called “aws_vpc_ and naming my resource “main”. I gave my vpc a tag called “deploy09_vpc” and used the cidr block 10.0.0.0/18.

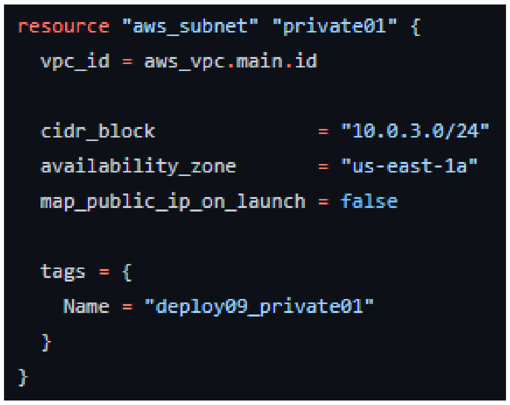


```
#Create VPC
resource "aws_vpc" "main" {
  cidr_block = "10.0.0.0/18"

  tags = {
    Name = "deploy09_vpc"
  }
}
```

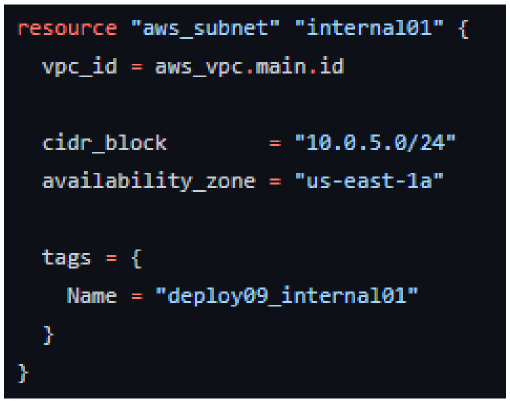
The next thing that we have to create are subnets. Subnets are simply a network inside a network. Companies use subnets to divide large networks into smaller, efficient subnetworks. This will help minimize traffic. For this assignment, we were tasked to create 2 public subnets, 1 private subnets, and 2 internal subnets. I chose to add one more private subnet for personal reasons as it would look better on my topology.



```
resource "aws_subnet" "public01" {
  vpc_id = aws_vpc.main.id

  cidr_block              = "10.0.1.0/24"
  availability_zone       = "us-east-1a"
  map_public_ip_on_launch = true

  tags = {
    Name = "deploy09_public01"
  }
}
```

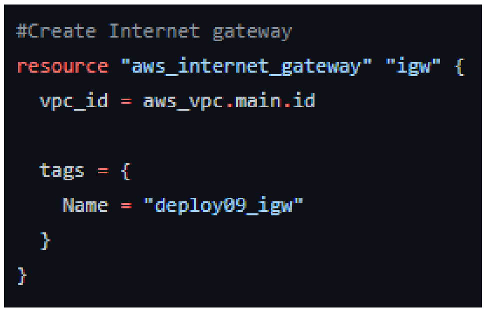


```
resource "aws_subnet" "private01" {
  vpc_id = aws_vpc.main.id

  cidr_block              = "10.0.3.0/24"
  availability_zone       = "us-east-1a"
  map_public_ip_on_launch = false

  tags = {
    Name = "deploy09_private01"
  }
}
```



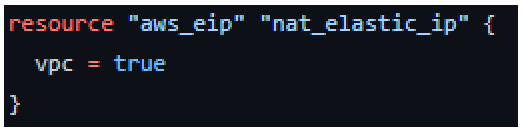
```
resource "aws_subnet" "internal01" {
  vpc_id = aws_vpc.main.id

  cidr_block        = "10.0.5.0/24"
  availability_zone = "us-east-1a"

  tags = {
    Name = "deploy09_internal01"
  }
}
```

While creating these subnets, I had to configure the vpc that each would use. I then had to give each subnet a unique cidr block and an availability zone which would be us-east-1a. The only thing that I saw a difference was the map_pllub_ip_on_launch option. This would allow the subnet to have a public ip address. For this case we would not want a public ip for the private subnet. The option was not added on the internal subnet because there is no need for a private or public ip.
We will now need to create an internet gateway which will allow traffic inside and outside of our vpc. We will simply need to just attach it to our VPC.



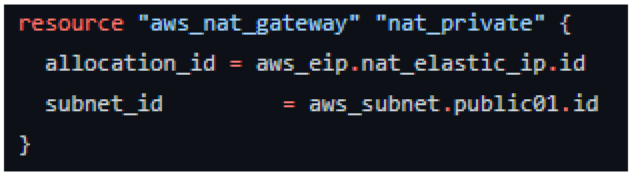
```
#Create Internet gateway
resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.main.id

  tags = {
    Name = "deploy09_igw"
  }
}
```

An elastic IP resource needs to be created so it can be attached to our NAT Gateway.



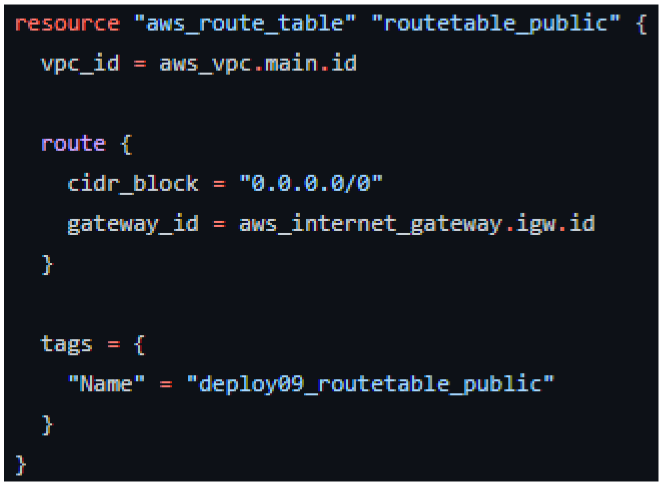
```
resource "aws_eip" "nat_elastic_ip" {
  vpc = true
}
```

While creating the NAT Gateway, we will need to link our elastic ip resource to it along with the public subnet. Nat Gateway allows our private resources to interact with the internet to get updates. Our private resources don't have direct access to the internet so it will talk to the nat gateway which is located inside the public subnet.

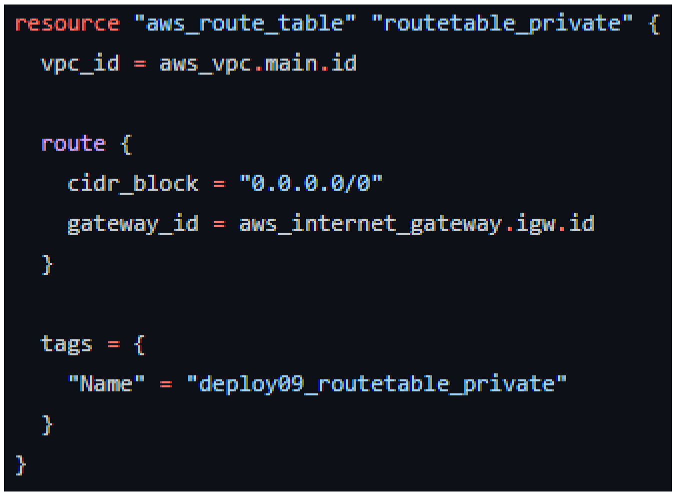


```
resource "aws_nat_gateway" "nat_private" {
  allocation_id = aws_eip.nat_elastic_ip.id
  subnet_id         = aws_subnet.public01.id
}
```

After the required resources were made, we would have to create a public and private route table. Route tables contain a set of rules that are used to determine where network traffic from your subnet is directed.



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

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

  tags = {
    "Name" = "deploy09_routetable_public"
  }
}
```

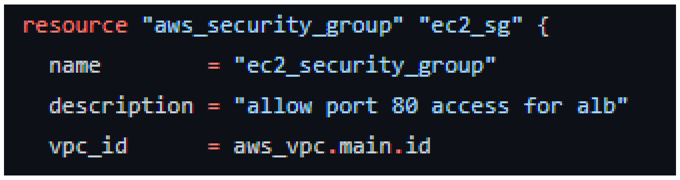


```
resource "aws_route_table" "routetable_private" {
  vpc_id = aws_vpc.main.id

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

  tags = {
    "Name" = "deploy09_routetable_private"
  }
}
```

Part 2 - EC2

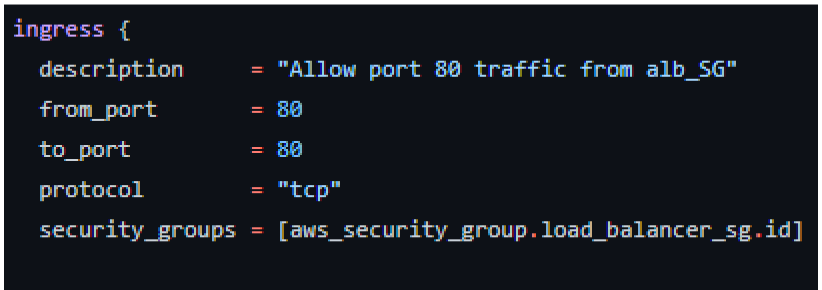
For this part, we are tasked to create an EC2 instance in a private subnet. While configuring the EC2, we have to select an Ubuntu AMI. The other configuration such as instance type, size, and tags are left up to our choosing. For security, we have to create a security group with certain ingress and egress rules.

The first thing that I decided to do is create a security group using the resource “aws_security_group”. Inside this resource, I named my security group, gave it a description, and linked it to my vpc.

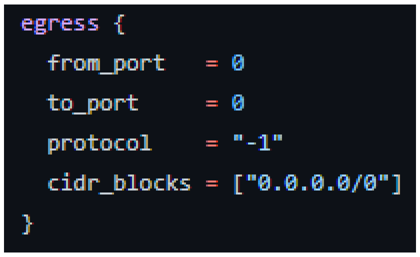


```
resource "aws_security_group" "ec2_sg" {
  name        = "ec2_security_group"
  description = "allow port 80 access for alb"
  vpc_id      = aws_vpc.main.id
```

For ingress and egress there were certain requirements we had to follow. Ingress had to allow port 80 traffic from the ALB security group. This security group is for an upcoming security group we had to make. I decided to proceed to creating the current security group. For egress rules, we had to allow all outbound traffic to any ipv4 traffic.

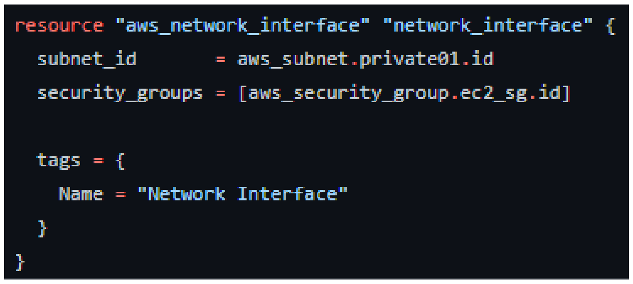


```
ingress {
  description     = "Allow port 80 traffic from alb_SG"
  from_port       = 80
  to_port         = 80
  protocol        = "tcp"
  security_groups = [aws_security_group.load_balancer_sg.id]
```



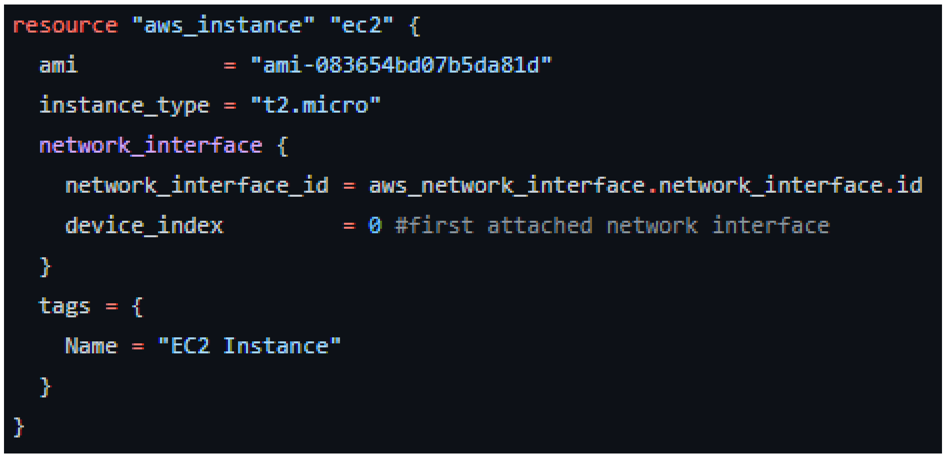
```
egress {
  from_port   = 0
  to_port     = 0
  protocol    = "-1"
  cidr_blocks = ["0.0.0.0/0"]
}
```

I had to find a way to attach this security group to the EC2 instance before creating it. I did some research and found out I had to create an aws_network_interface module. This will allow me to attach a security group and subnet. I can then specify the network card while creating the ec2.



```
resource "aws_network_interface" "network_interface" {
  subnet_id       = aws_subnet.private01.id
  security_groups = [aws_security_group.ec2_sg.id]

  tags = {
    Name = "Network Interface"
  }
}
```

Finally, I can create the EC2 instance using the aws_instance resource

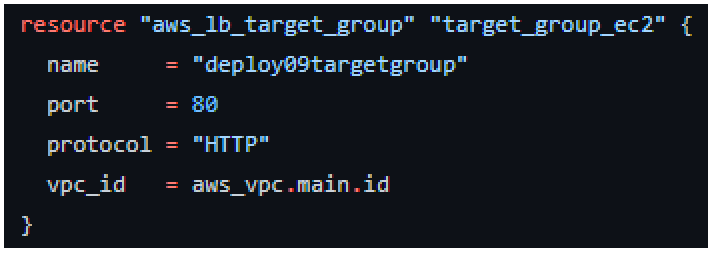


```
resource "aws_instance" "ec2" {
  ami           = "ami-083654bd07b5da81d"
  instance_type = "t2.micro"
  network_interface {
    network_interface_id = aws_network_interface.network_interface.id
    device_index         = 0 #first attached network interface
  }
  tags = {
    Name = "EC2 Instance"
  }
}
```

Part 3 - ALB

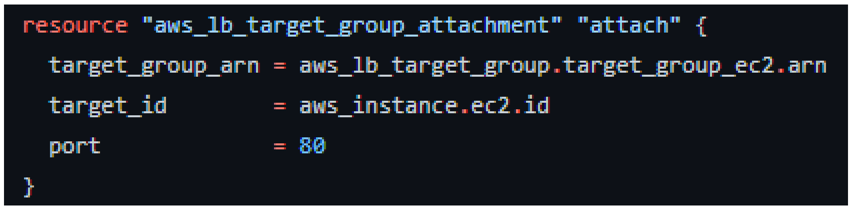
For this part, we had to create an application load balancer in 2 public subnets. The ALB needs a security group with certain required ingress and egress rules. After we create the security group, we have to develop a target group and link it to the ec2 instance we created in part 2. The ALB needs a listener that forwards traffic to the target group. While creating this, we have to accept HTTP traffic only.

I decided on creating the target group first since the ALB depends on it. I used the “aws_lb_target_group_ resource" and simply named it, linked the vpc, choose what type of port and protocol it should listen for.



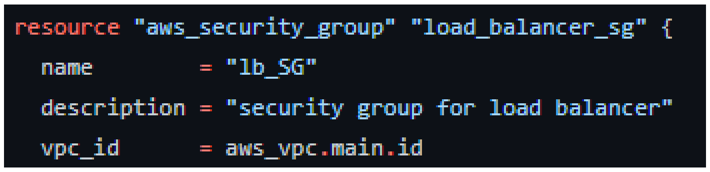
```
resource "aws_lb_target_group" "target_group_ec2" {
  name     = "deploy09targetgroup"
  port     = 80
  protocol = "HTTP"
  vpc_id   = aws_vpc.main.id
}
```

I then needed to attach the target group so I used the resource "aws_lb_target_group_attachment" which links the target group to the load balancer. You would then need to configure the ec2 that will be targeted and a specific port.



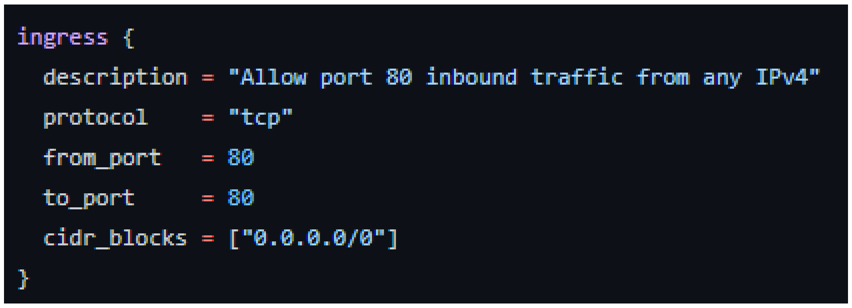
```
resource "aws_lb_target_group_attachment" "attach" {
  target_group_arn = aws_lb_target_group.target_group_ec2.arn
  target_id        = aws_instance.ec2.id
  port             = 80
}
```

We can then create the security group for the load balancer. We start off by naming the security group, giving a description, and linking the vpc.



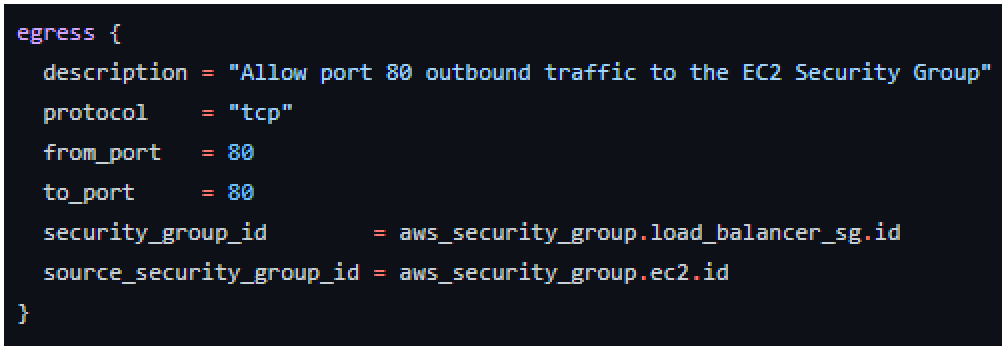
```
resource "aws_security_group" "load_balancer_sg" {
  name        = "lb_SG"
  description = "security group for load balancer"
  vpc_id      = aws_vpc.main.id
```

For the ingress rules, we are simply allowing only port 80 inbound from any ipv4.



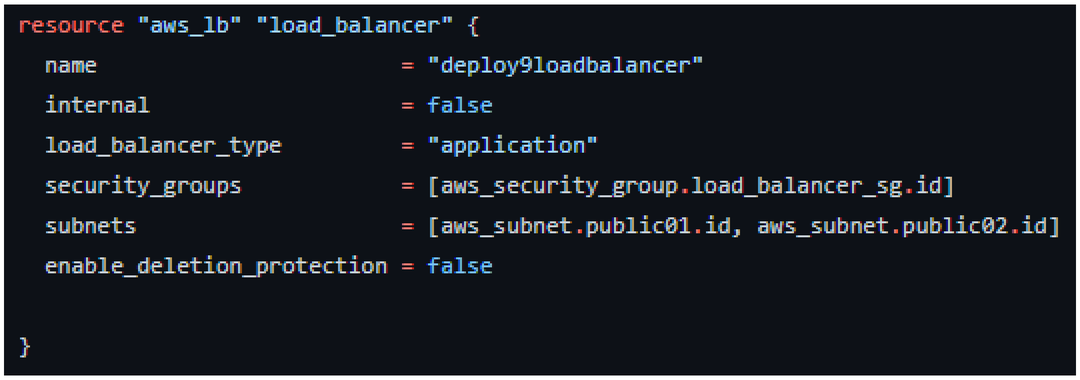
```
ingress {
  description = "Allow port 80 inbound traffic from any IPv4"
  protocol    = "tcp"
  from_port   = 80
  to_port     = 80
  cidr_blocks = ["0.0.0.0/0"]
}
```

For egress, we are allowing only port 80 outbound traffic to the EC2 security group. This will direct all http outbound traffic to the ec2 instance.



```
egress {
  description = "Allow port 80 outbound traffic to the EC2 Security Group"
  protocol    = "tcp"
  from_port   = 80
  to_port     = 80
  security_group_id        = aws_security_group.load_balancer_sg.id
  source_security_group_id = aws_security_group.ec2.id
}
```

Finally, we can create the load balancer using the resource “aws_lb”. While creating the load balancer, there are important configurations such as load_balancer_type which tells the load balancer what type it is. We need it to be an application load balancer so we will select that. We then need to attach the security group and subnet.

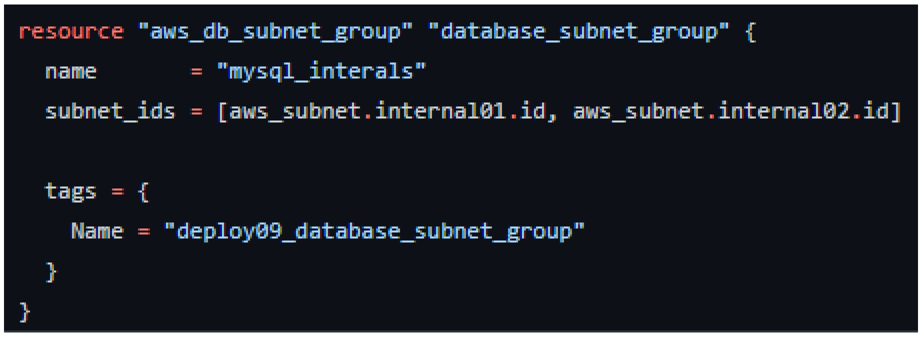


```
resource "aws_lb" "load_balancer" {
  name                       = "deploy9loadbalancer"
  internal                   = false
  load_balancer_type         = "application"
  security_groups            = [aws_security_group.load_balancer_sg.id]
  subnets                    = [aws_subnet.public01.id, aws_subnet.public02.id]
  enable_deletion_protection = false

}
```

Part 4 - RDS

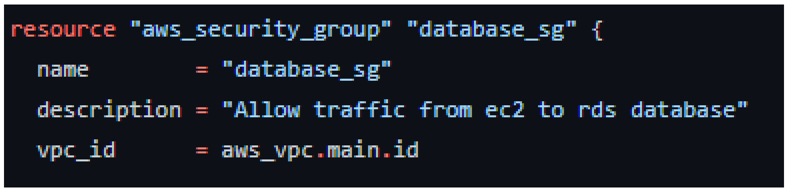
The last part is to create an RDS database. For this assignment we had to create a PostgresQL RDS database but I wanted to create a traditional MySQL database. For configurations, we had to make the database multi-az. The name, instance type, size, tags, database username, and database password were up to our desires. After creating the database, we had to create a security group for the RDS with certain ingress rules. We also had to create a subnet group for the RDS that had the 2 internal subnets from part 1 attached.

The first thing that I decided to tackle was creating the database subnet group. I used the resource “aws_db_subnet_group”. I simply had to name it, and attach the two internal subnets



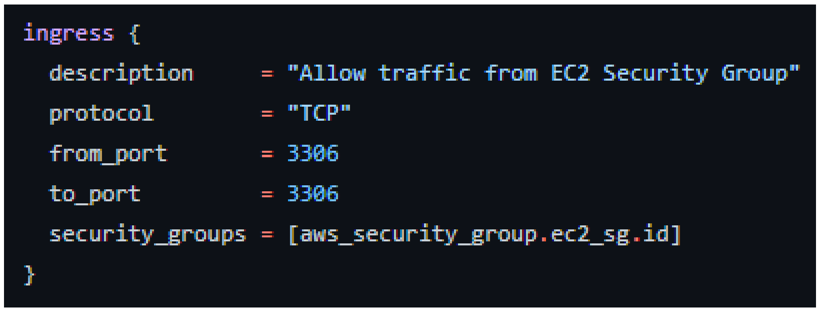
```
resource "aws_db_subnet_group" "database_subnet_group" {
  name       = "mysql_interals"
  subnet_ids = [aws_subnet.internal01.id, aws_subnet.internal02.id]

  tags = {
    Name = "deploy09_database_subnet_group"
  }
}
```

I then created the security group for the RDS database. I started off by naming the security group, giving a description, and linking the vpc.



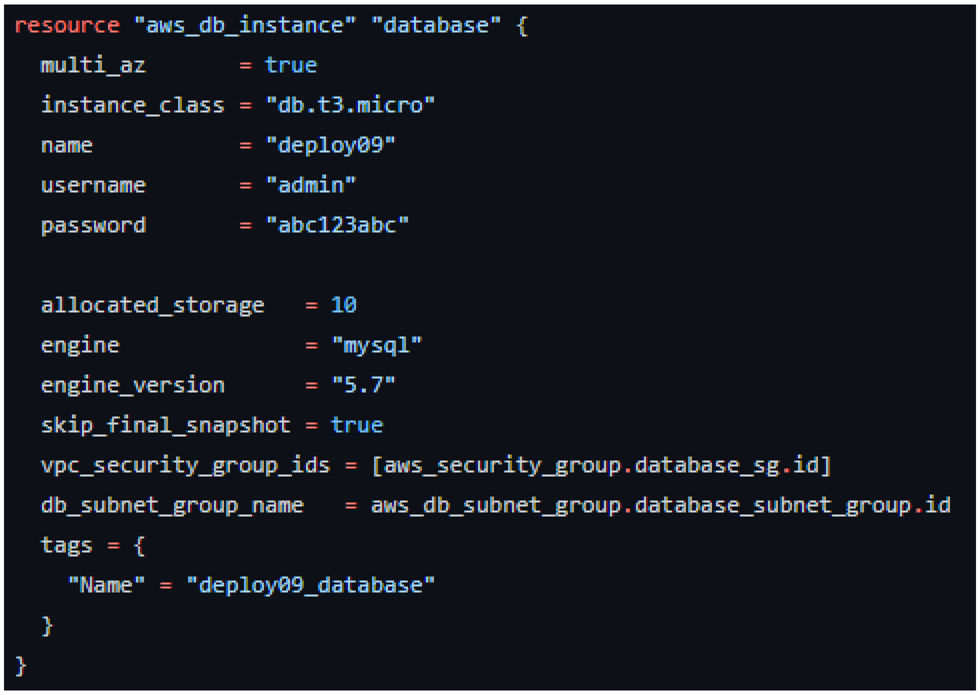
```
resource "aws_security_group" "database_sg" {
  name        = "database_sg"
  description = "Allow traffic from ec2 to rds database"
  vpc_id      = aws_vpc.main.id
```

For the ingress rules, we had to allow port 3306 traffic from the EC2 security group.



```
 ingress {
   description     = "Allow traffic from EC2 Security Group"
   protocol        = "TCP"
   from_port       = 3306
   to_port         = 3306
   security_groups = [aws_security_group.ec2_sg.id]
 }
```

Once the security group was established, I could make the database using the resource “aws_db_instance” . While configuring this resource, the challenging part was making the database mysql. I found out I had to specify the engine to “mysql” and select the engine_version which was the mysql version.

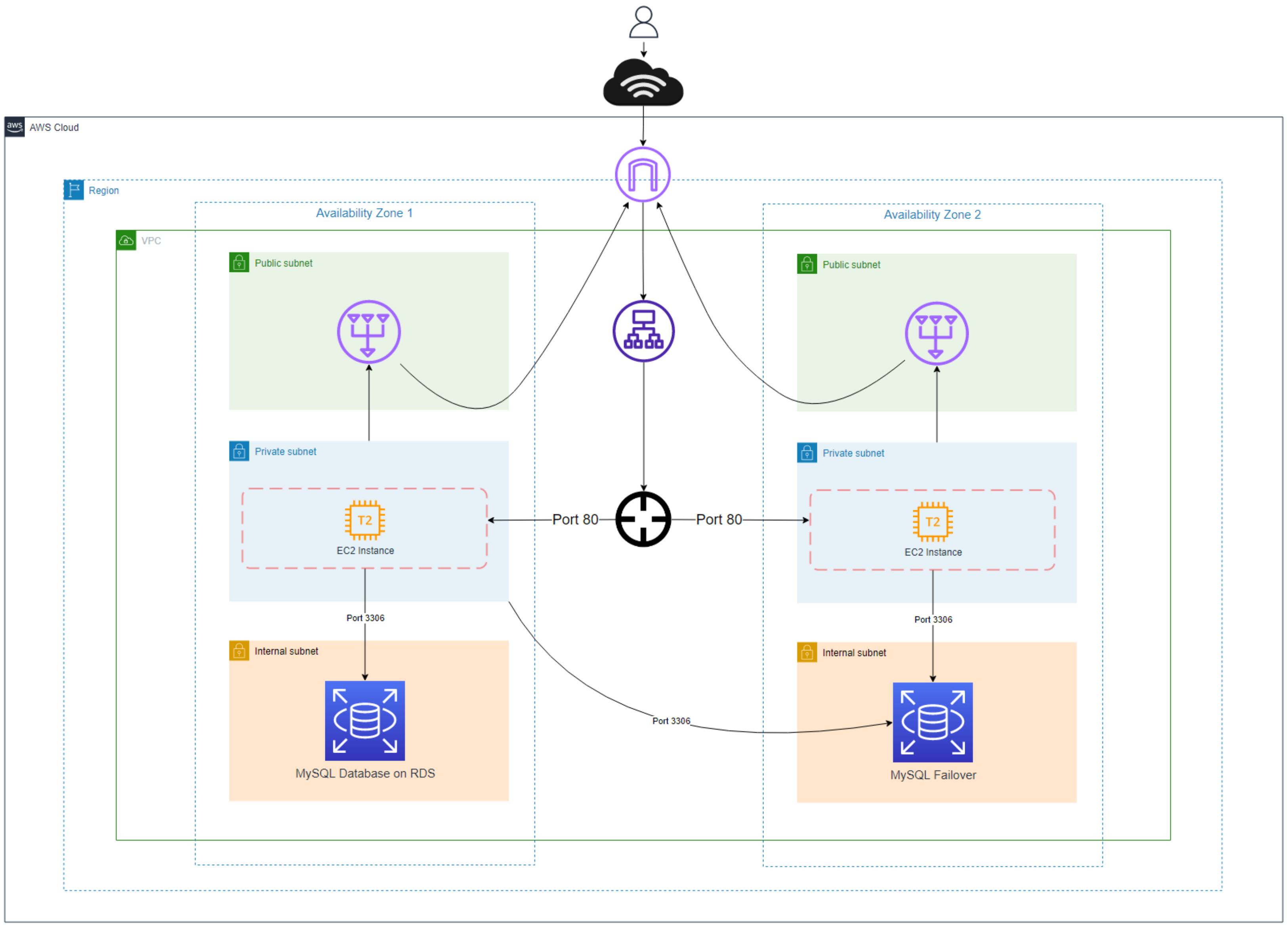


```
resource "aws_db_instance" "database" {
  multi_az       = true
  instance_class = "db.t3.micro"
  name           = "deploy09"
  username       = "admin"
  password       = "abc123abc"

  allocated_storage   = 10
  engine              = "mysql"
  engine_version      = "5.7"
  skip_final_snapshot = true
  vpc_security_group_ids = [aws_security_group.database_sg.id]
  db_subnet_group_name   = aws_db_subnet_group.database_subnet_group.id
  tags = {
    "Name" = "deploy09_database"
  }
}
```

Once everything was set up I ran a few helpful commands
git checkout -b rdeodutt/deployment09

terraform fmt -recursive
terraform init
terraform plan
terraform apply -auto-approve
terraform destroy -auto-approve

Topology

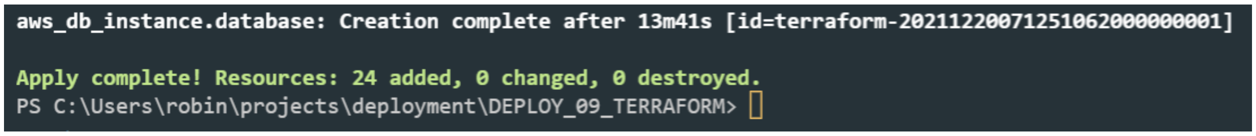


We are creating resources in the AWS Cloud. Inside the cloud we are hosting our VPC inside the US-EAST-1 region. VPCs are simply virtual private clouds. Think of it as an isolated network. Inside the vpc, we have two availability zones for resilience. In case one of our AZs goes down, we will have another up and running. Inside these AZs we were tasked with creating 3 subnets. Subnets are simply a network inside a network. Companies use subnets to divide large networks into smaller, efficient subnetworks. This will help minimize traffic. Public subnets route to the internet gateway. Internet gateways allow traffic in and out of the vpc. This allows our resources to interact with the internet. Private subnets route to the NAT gateways. NAT Gateways allow our private subnets to interact with the internet. Our private subnets do not have interactive access and applications need to be updated so the NAT gateway, which is hosted inside the public subnet, allows our private subnet to interact with the internet. The internal subnet does not associate any route tables but it takes a local / private network.

Inside the private subnet, we were tasked with creating an ec2 instance that has an Ubuntu AMI. The security group rules allow port 80 traffic from the load balancer only. The load balancer in this topology directs traffic from the internet gateway to the target group. The target group then directs traffic to a specific ec2 instance at a specific port. Finally, in the internal subnet, we created an RDS Database that is running MySQL. RDS stands for relational database. This database has a failover database inside availability zone 2. This was implemented to remove single points of failure. Thus by doing so gives our database resilience. The database allows inbound traffic from port 3306 from the EC2 instances only.

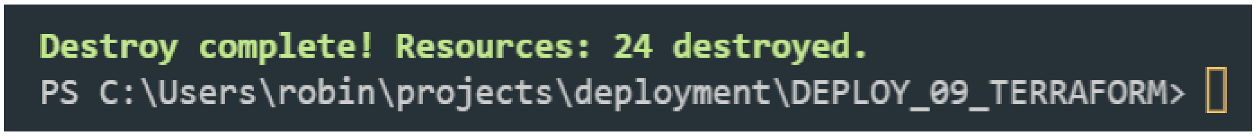
Screenshots

Successful creation of infrastructure



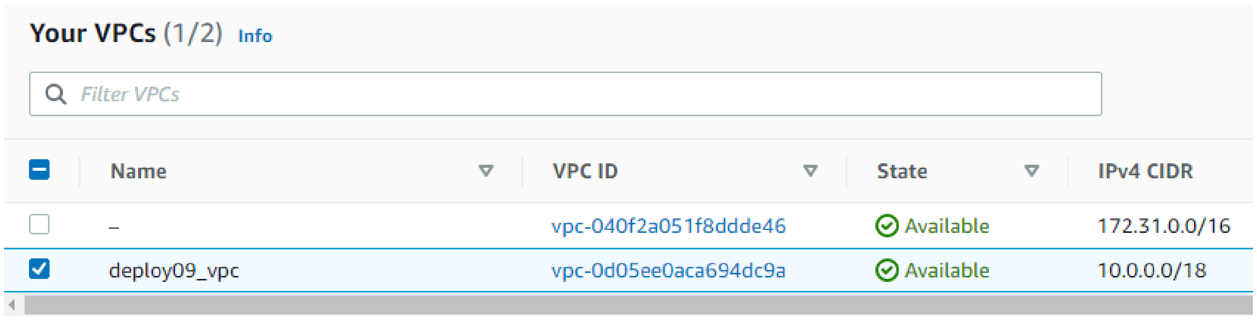
```
aws_db_instance.database: Creation complete after 13m41s [id=terraform-20211220071251062000000001]

Apply complete! Resources: 24 added, 0 changed, 0 destroyed.
PS C:\Users\robin\projects\deployment\DEPLOY_09_TERRAFORM>
```

Successful deletion of infrastructure



```
Destroy complete! Resources: 24 destroyed.
PS C:\Users\robin\projects\deployment\DEPLOY_09_TERRAFORM>
```

VPC

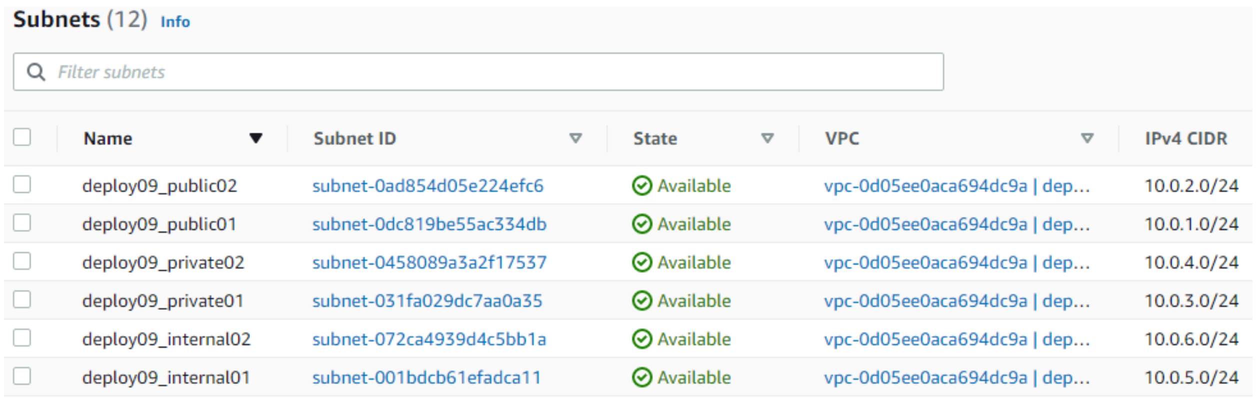


Your VPCs (1/2) Info

Filter VPCs

	Name	VPC ID	State	IPv4 CIDR
☐	–	vpc-040f2a051f8ddde46	Available	172.31.0.0/16
☑	deploy09_vpc	vpc-0d05ee0aca694dc9a	Available	10.0.0.0/18

Subnets

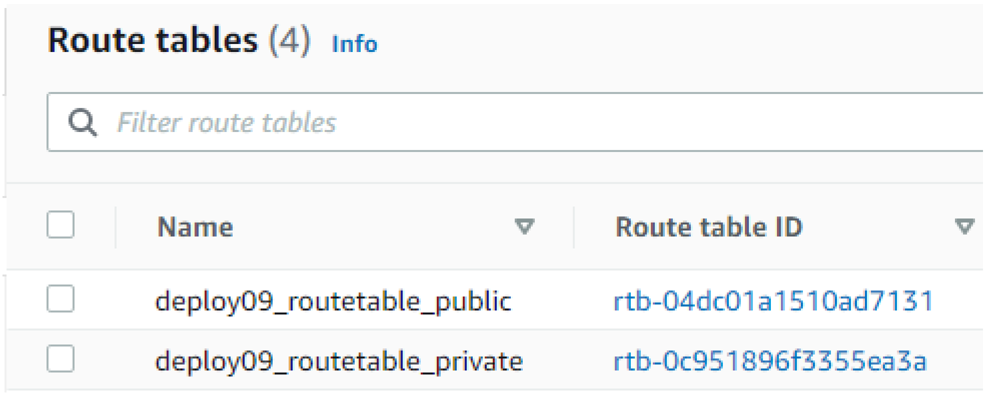


Subnets (12) Info

Filter subnets

	Name	Subnet ID	State	VPC	IPv4 CIDR
☐	deploy09_public02	subnet-0ad854d05e224efc6	✓ Available	vpc-0d05ee0aca694dc9a | dep...	10.0.2.0/24
☐	deploy09_public01	subnet-0dc819be55ac334db	✓ Available	vpc-0d05ee0aca694dc9a | dep...	10.0.1.0/24
☐	deploy09_private02	subnet-0458089a3a2f17537	✓ Available	vpc-0d05ee0aca694dc9a | dep...	10.0.4.0/24
☐	deploy09_private01	subnet-031fa029dc7aa0a35	✓ Available	vpc-0d05ee0aca694dc9a | dep...	10.0.3.0/24
☐	deploy09_internal02	subnet-072ca4939d4c5bb1a	✓ Available	vpc-0d05ee0aca694dc9a | dep...	10.0.6.0/24
☐	deploy09_internal01	subnet-001bdcb61efadca11	✓ Available	vpc-0d05ee0aca694dc9a | dep...	10.0.5.0/24

Route Tables

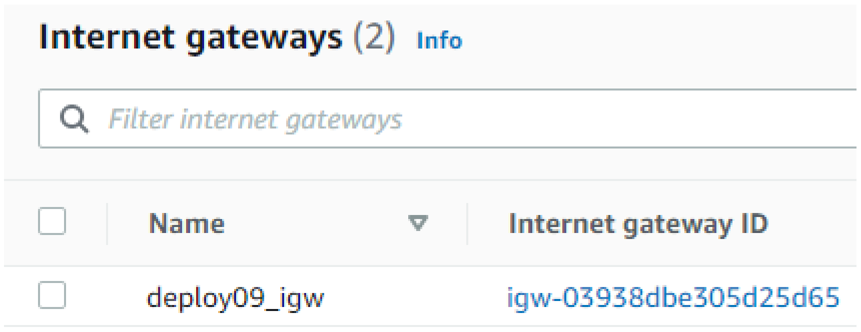


Route tables (4) Info

Filter route tables

	Name	Route table ID
☐	deploy09_routetable_public	rtb-04dc01a1510ad7131
☐	deploy09_routetable_private	rtb-0c951896f3355ea3a

Internet Gateway

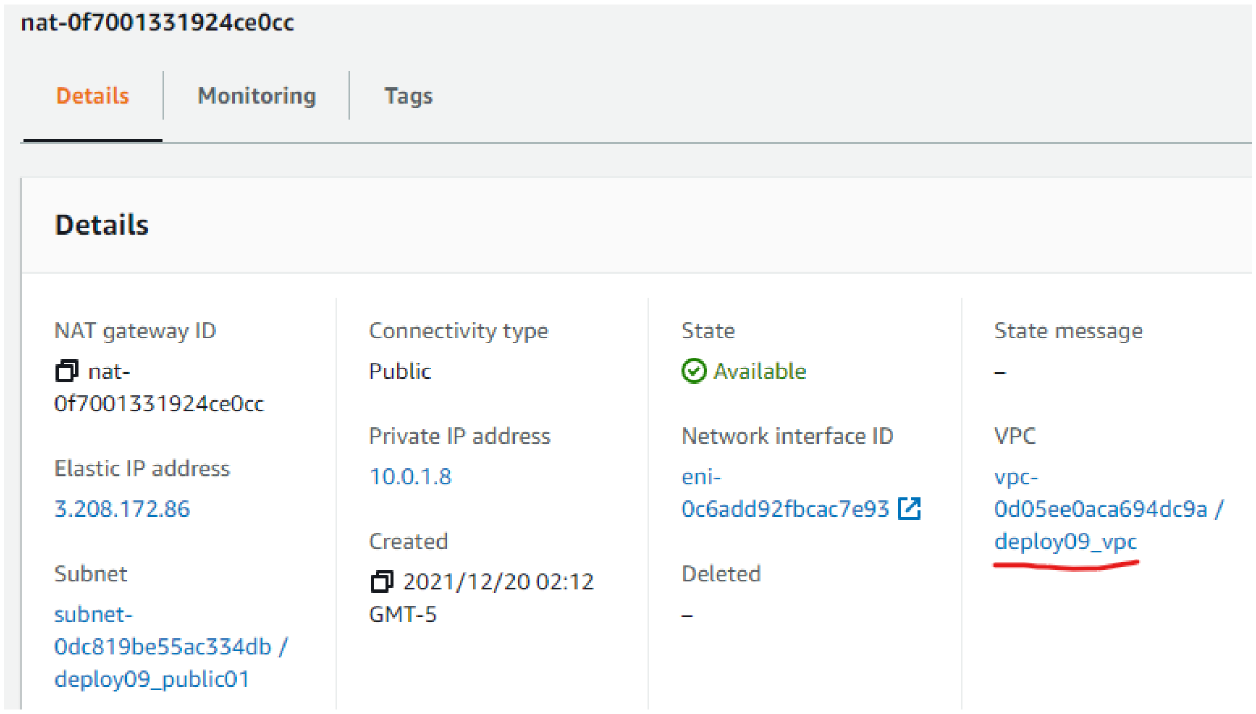


Internet gateways (2) Info

Filter internet gateways

	Name	Internet gateway ID
☐	deploy09_igw	igw-03938dbe305d25d65

NAT Gateway



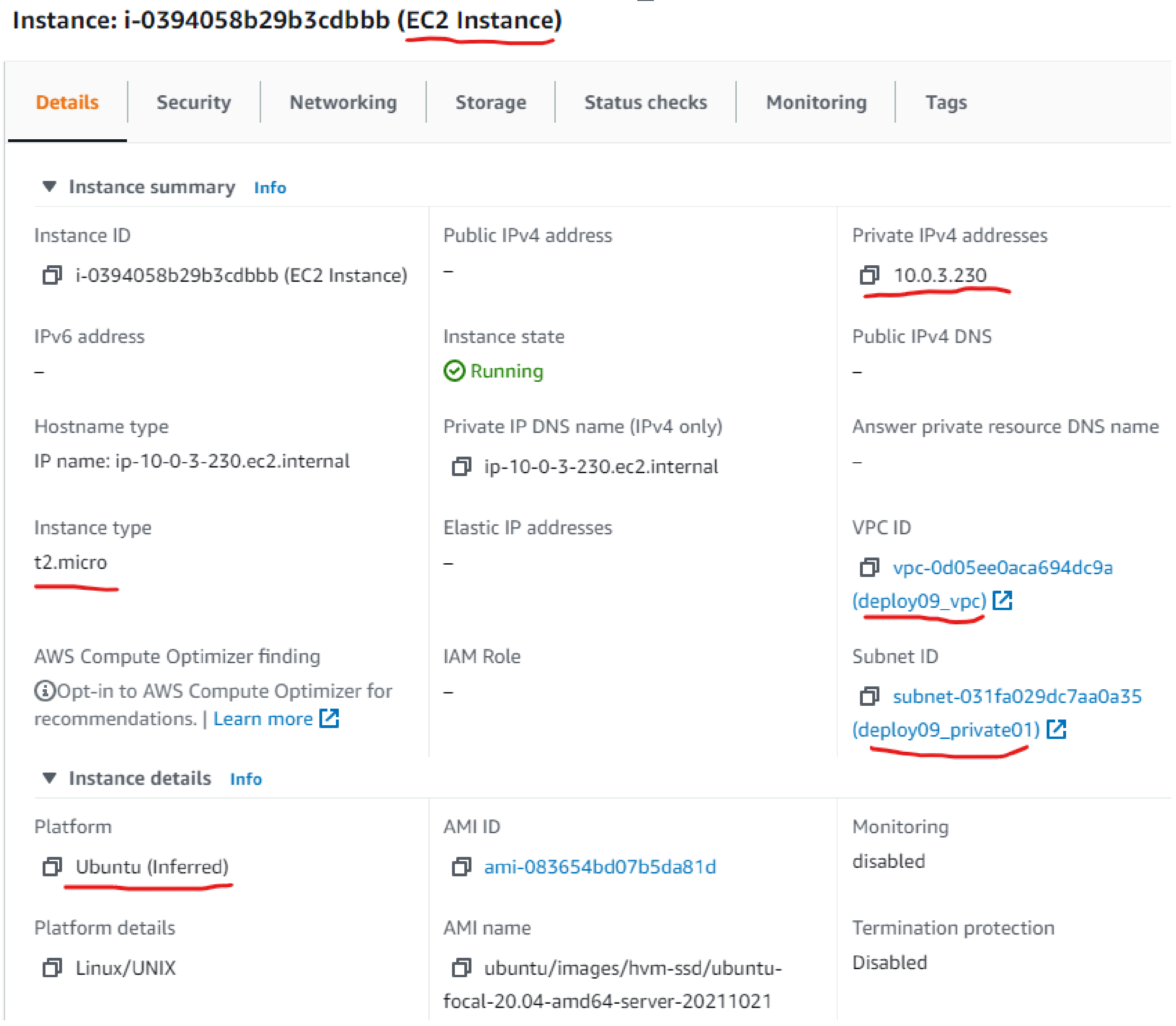
nat-0f7001331924ce0cc

Details | Monitoring | Tags

Details

NAT gateway ID	Connectivity type	State	State message
nat-0f7001331924ce0cc	Public	✓ Available	–
Elastic IP address	Private IP address	Network interface ID	VPC
3.208.172.86	10.0.1.8	eni-0c6add92fbcac7e93	vpc-0d05ee0aca694dc9a / deploy09_vpc
Subnet	Created	Deleted	
subnet-0dc819be55ac334db / deploy09_public01	2021/12/20 02:12 GMT-5	–	

EC2



Instance: i-0394058b29b3cdbbb (EC2 Instance)

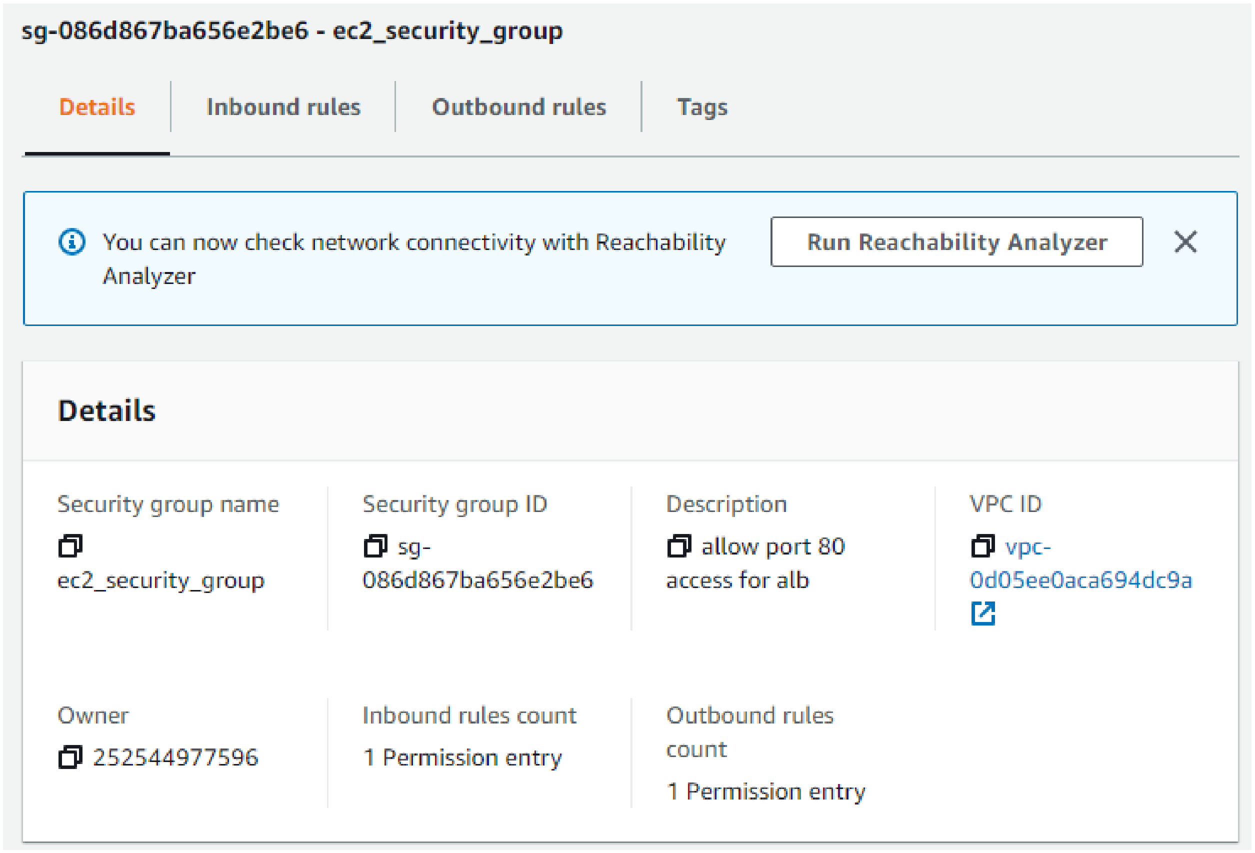
▼ Instance summary Info

Instance ID	Public IPv4 address	Private IPv4 addresses
i-0394058b29b3cdbbb (EC2 Instance)	–	10.0.3.230
IPv6 address	Instance state	Public IPv4 DNS
–	Running	–
Hostname type	Private IP DNS name (IPv4 only)	Answer private resource DNS name
IP name: ip-10-0-3-230.ec2.internal	ip-10-0-3-230.ec2.internal	–
Instance type	Elastic IP addresses	VPC ID
t2.micro	–	vpc-0d05ee0aca694dc9a (deploy09_vpc)
AWS Compute Optimizer finding	IAM Role	Subnet ID
Opt-in to AWS Compute Optimizer for recommendations. | Learn more	–	subnet-031fa029dc7aa0a35 (deploy09_private01)

▼ Instance details Info

Platform	AMI ID	Monitoring
Ubuntu (Inferred)	ami-083654bd07b5da81d	disabled
Platform details	AMI name	Termination protection
Linux/UNIX	ubuntu/images/hvm-ssd/ubuntu-focal-20.04-amd64-server-20211021	Disabled

Security Group



sg-086d867ba656e2be6 - ec2_security_group

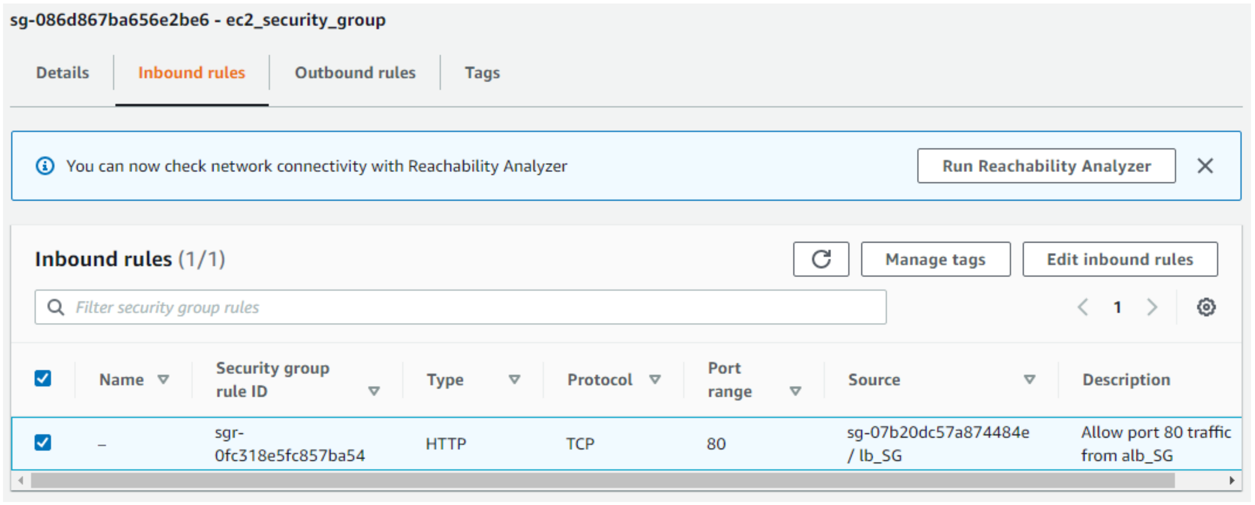
You can now check network connectivity with Reachability Analyzer

Details

Security group name	Security group ID	Description	VPC ID
ec2_security_group	sg-086d867ba656e2be6	allow port 80 access for alb	vpc-0d05ee0aca694dc9a

Owner	Inbound rules count	Outbound rules count
252544977596	1 Permission entry	1 Permission entry

Ingress/Inbound Security Group



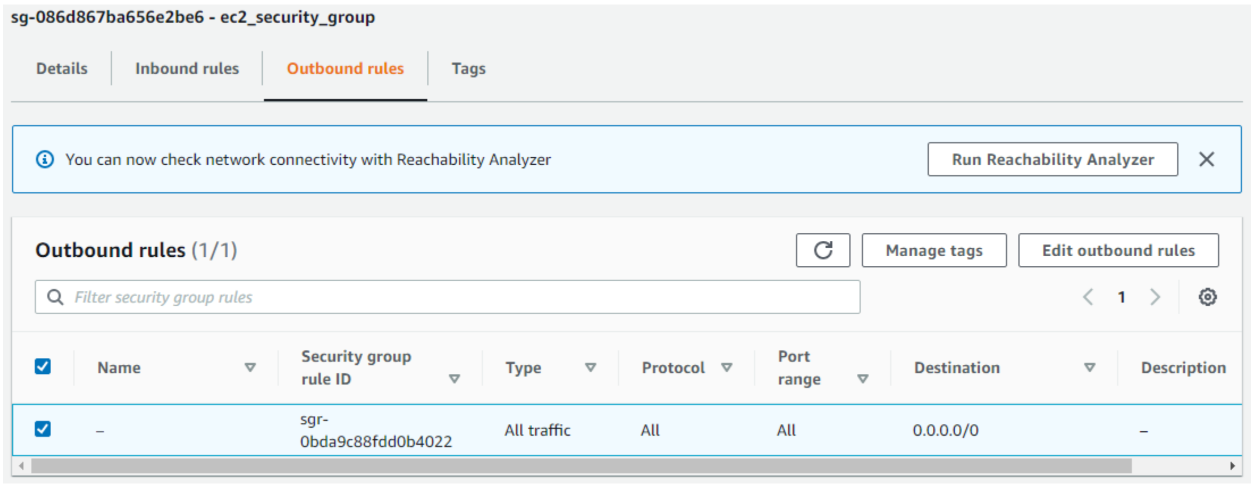
sg-086d867ba656e2be6 - ec2_security_group

You can now check network connectivity with Reachability Analyzer

Inbound rules (1/1)

☑	Name	Security group rule ID	Type	Protocol	Port range	Source	Description
☑	–	sgr-0fc318e5fc857ba54	HTTP	TCP	80	sg-07b20dc57a874484e / lb_SG	Allow port 80 traffic from alb_SG

Egress/Outbound Security Group



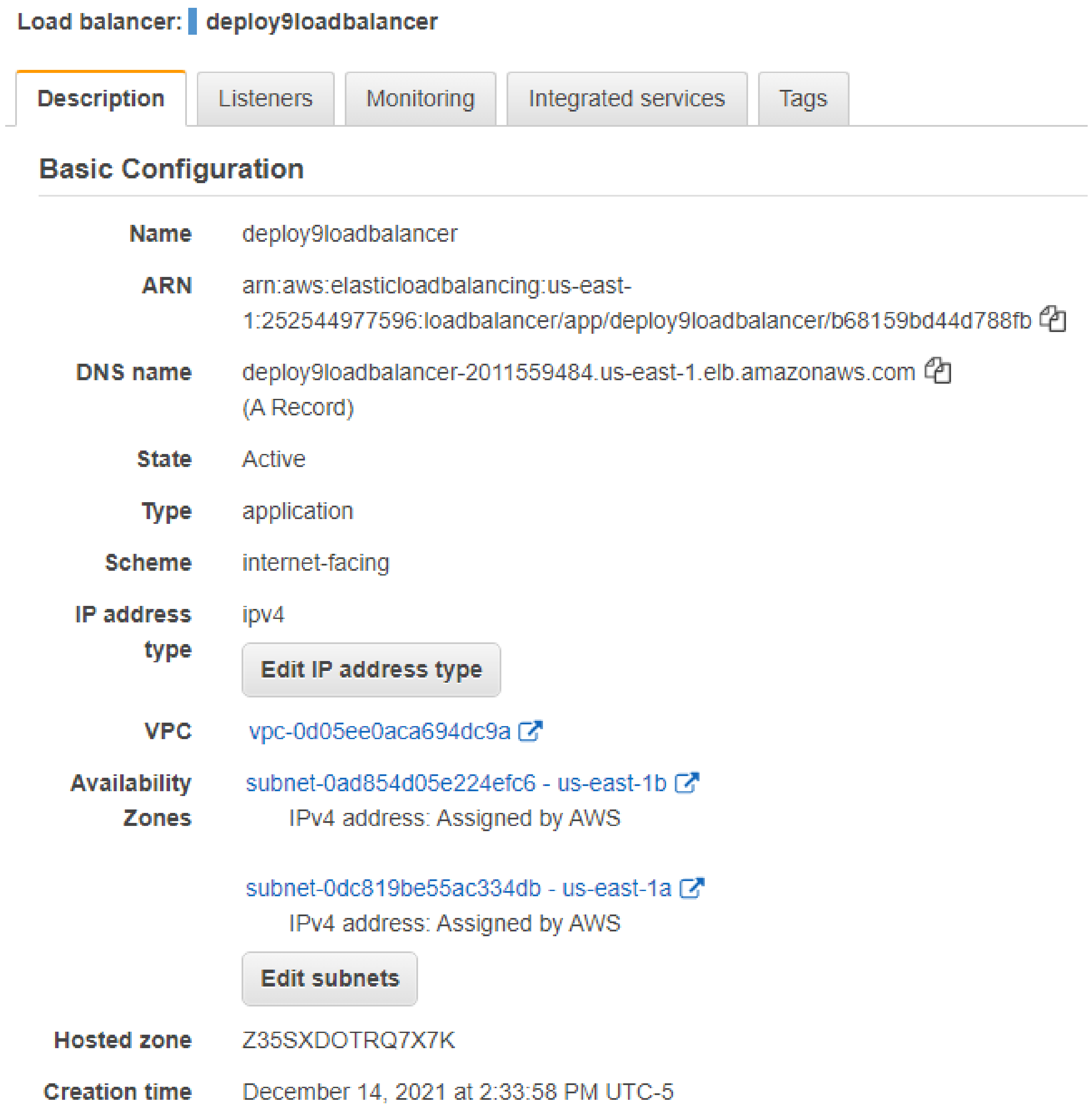
sg-086d867ba656e2be6 - ec2_security_group

Details
Inbound rules
Outbound rules
Tags
You can now check network connectivity with Reachability Analyzer
Run Reachability Analyzer
Manage tags
Edit outbound rules
Filter security group rules

Outbound rules (1/1)

Name	Security group rule ID	Type	Protocol	Port range	Destination	Description
–	sgr-0bda9c88fdd0b4022	All traffic	All	All	0.0.0.0/0	–

Application Load Balancer (ALB)

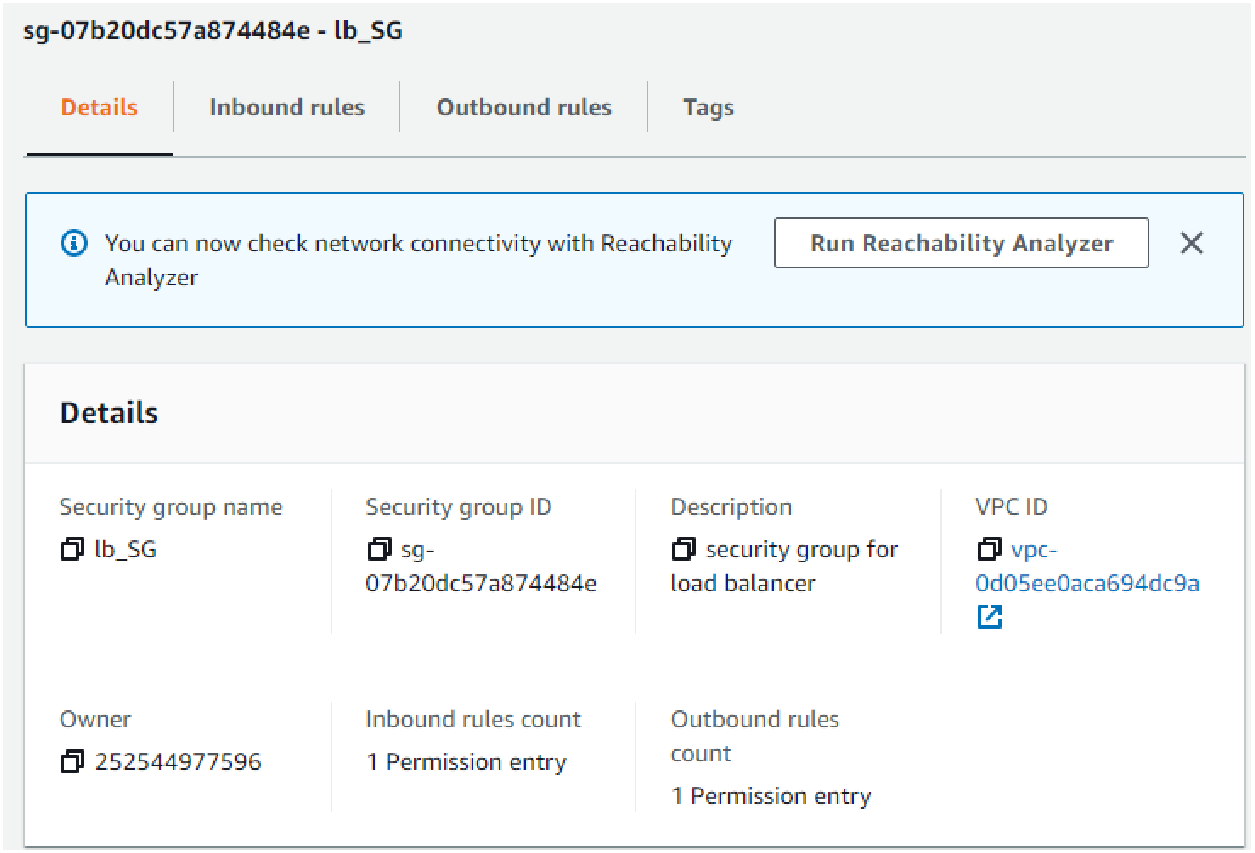


Load balancer: deploy9loadbalancer

Basic Configuration

Name	deploy9loadbalancer
ARN	arn:aws:elasticloadbalancing:us-east-1:252544977596:loadbalancer/app/deploy9loadbalancer/b68159bd44d788fb
DNS name	deploy9loadbalancer-2011559484.us-east-1.elb.amazonaws.com (A Record)
State	Active
Type	application
Scheme	internet-facing
IP address type	ipv4
VPC	vpc-0d05ee0aca694dc9a
Availability Zones	subnet-0ad854d05e224efc6 - us-east-1b IPv4 address: Assigned by AWS subnet-0dc819be55ac334db - us-east-1a IPv4 address: Assigned by AWS
Hosted zone	Z35SXDOTRQ7X7K
Creation time	December 14, 2021 at 2:33:58 PM UTC-5

ALB Security Group



sg-07b20dc57a874484e - lb_SG

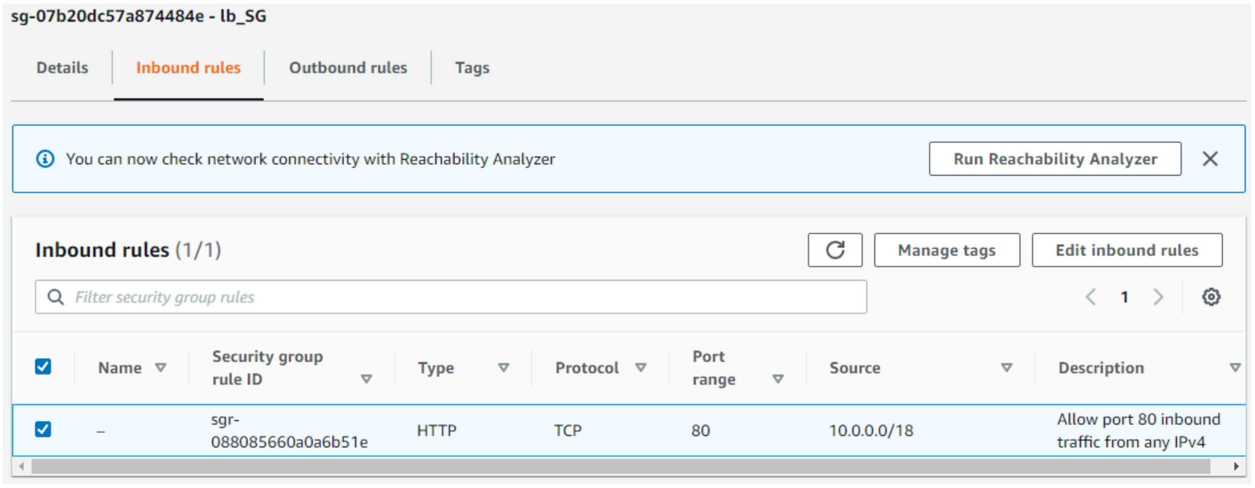
You can now check network connectivity with Reachability Analyzer

Details

Security group name	Security group ID	Description	VPC ID
lb_SG	sg-07b20dc57a874484e	security group for load balancer	vpc-0d05ee0aca694dc9a

Owner	Inbound rules count	Outbound rules count
252544977596	1 Permission entry	1 Permission entry

ALB Security Group Ingress



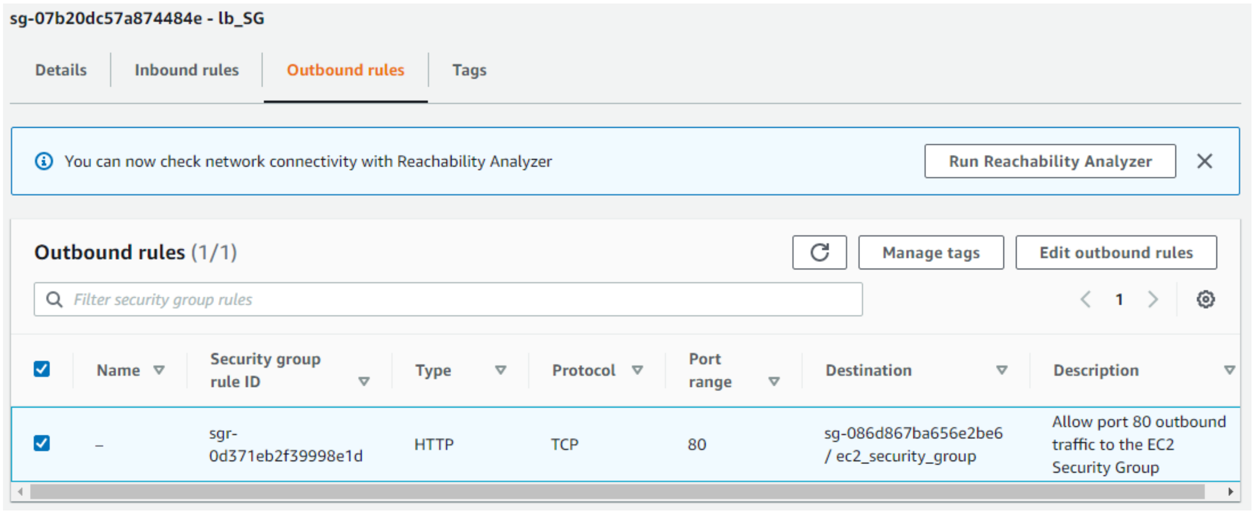
sg-07b20dc57a874484e - lb_SG

You can now check network connectivity with Reachability Analyzer

Inbound rules (1/1)

Name	Security group rule ID	Type	Protocol	Port range	Source	Description
–	sgr-088085660a0a6b51e	HTTP	TCP	80	10.0.0.0/18	Allow port 80 inbound traffic from any IPv4

ALB Security Group Egress



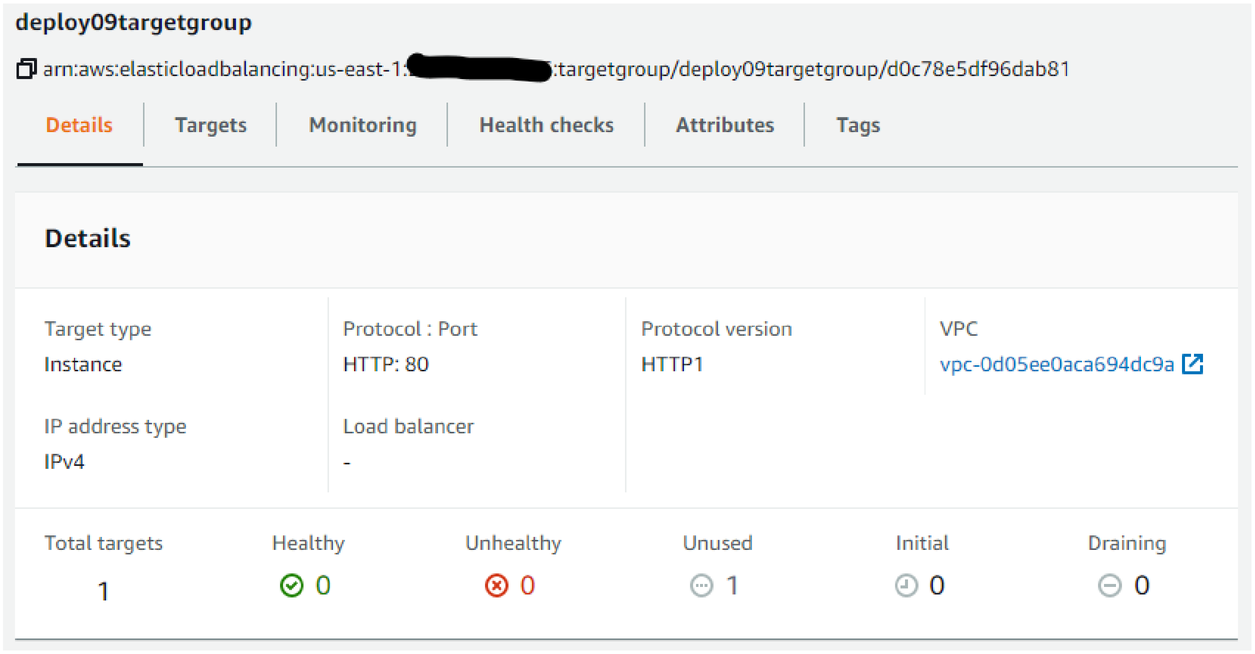
sg-07b20dc57a874484e - lb_SG

You can now check network connectivity with Reachability Analyzer

Outbound rules (1/1)

	Name	Security group rule ID	Type	Protocol	Port range	Destination	Description
✓	–	sgr-0d371eb2f39998e1d	HTTP	TCP	80	sg-086d867ba656e2be6 / ec2_security_group	Allow port 80 outbound traffic to the EC2 Security Group

Target Group



deploy09targetgroup

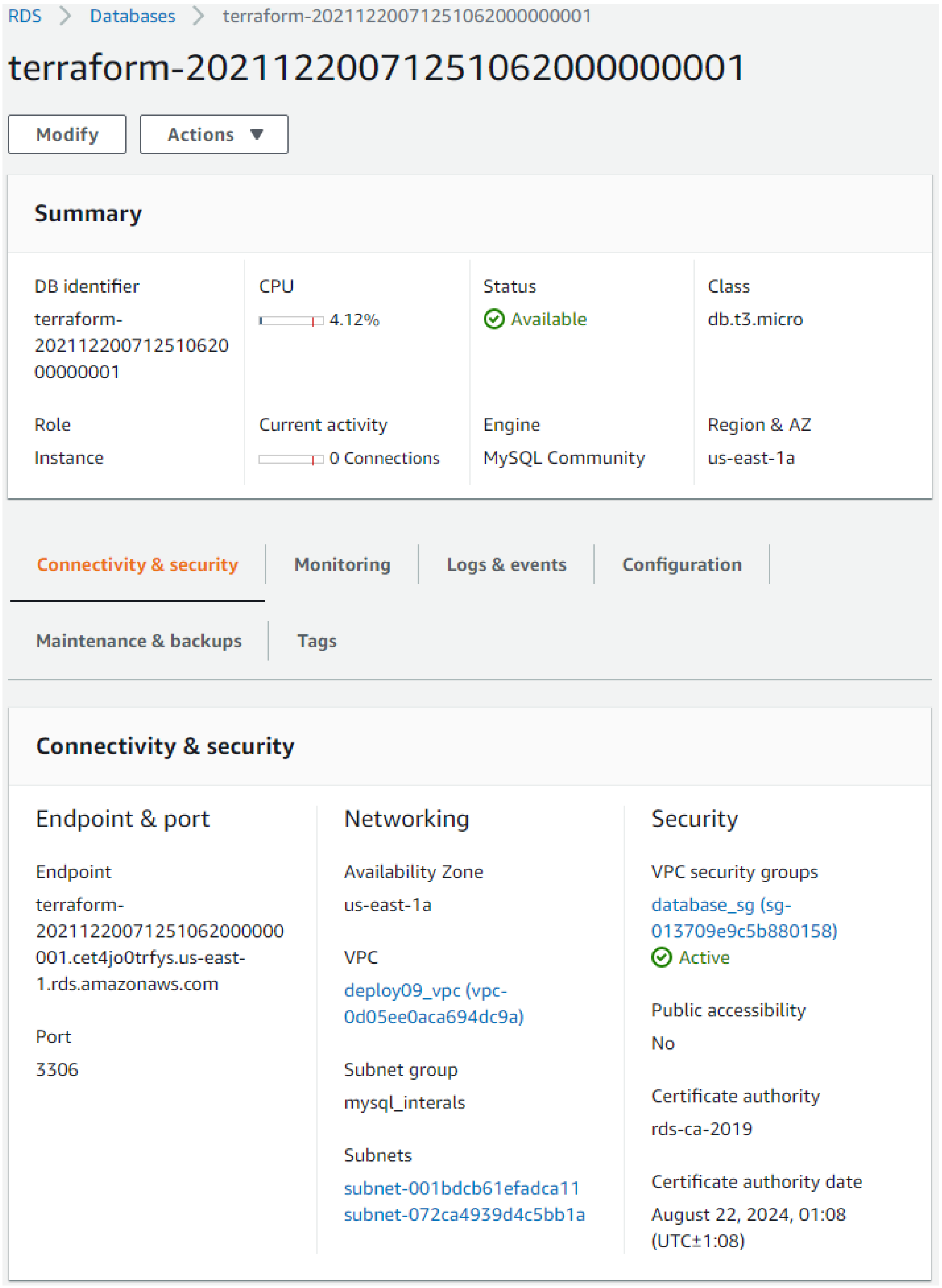
arn:aws:elasticloadbalancing:us-east-1:[redacted]:targetgroup/deploy09targetgroup/d0c78e5df96dab81

Details

Target type	Protocol : Port	Protocol version	VPC
Instance	HTTP: 80	HTTP1	vpc-0d05ee0aca694dc9a
IP address type	Load balancer		
IPv4	-		

Total targets	Healthy	Unhealthy	Unused	Initial	Draining
1	0	0	1	0	0

RDS Database



RDS > Databases > terraform-20211220071251062000000001

terraform-20211220071251062000000001

Modify | Actions ▼

Summary

DB identifier	CPU	Status	Class
terraform-20211220071251062000000001	4.12%	✓ Available	db.t3.micro
Role	Current activity	Engine	Region & AZ
Instance	0 Connections	MySQL Community	us-east-1a

Connectivity & security | Monitoring | Logs & events | Configuration | Maintenance & backups | Tags

Connectivity & security

Endpoint & port

Endpoint
terraform-20211220071251062000000001.cet4jo0trfys.us-east-1.rds.amazonaws.com

Port
3306

Networking

Availability Zone
us-east-1a

VPC
deploy09_vpc (vpc-0d05ee0aca694dc9a)

Subnet group
mysql_interals

Subnets
subnet-001bdcb61efadca11
subnet-072ca4939d4c5bb1a

Security

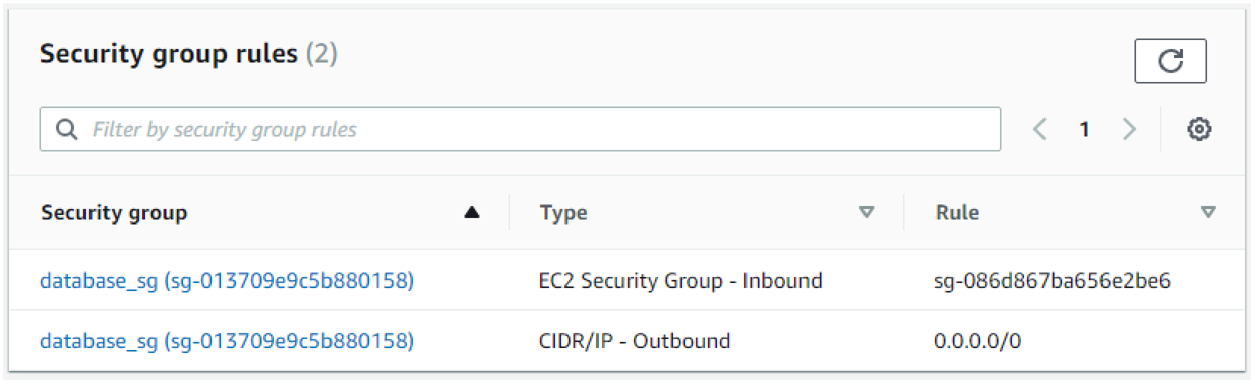
VPC security groups
database_sg (sg-013709e9c5b880158)
✓ Active

Public accessibility
No

Certificate authority
rds-ca-2019

Certificate authority date
August 22, 2024, 01:08 (UTC±1:08)

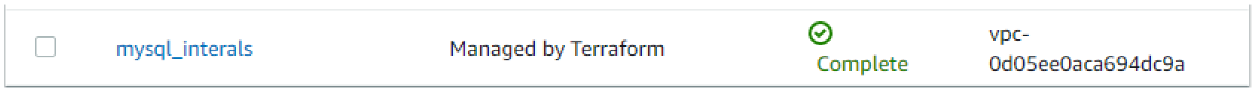
RDS Database Security Group



Security group rules (2)

Security group	Type	Rule
database_sg (sg-013709e9c5b880158)	EC2 Security Group - Inbound	sg-086d867ba656e2be6
database_sg (sg-013709e9c5b880158)	CIDR/IP - Outbound	0.0.0.0/0

RDS Database Subnet Group



☐	mysql_interals	Managed by Terraform	✓ Complete	vpc-0d05ee0aca694dc9a

Ricardo Deodutt