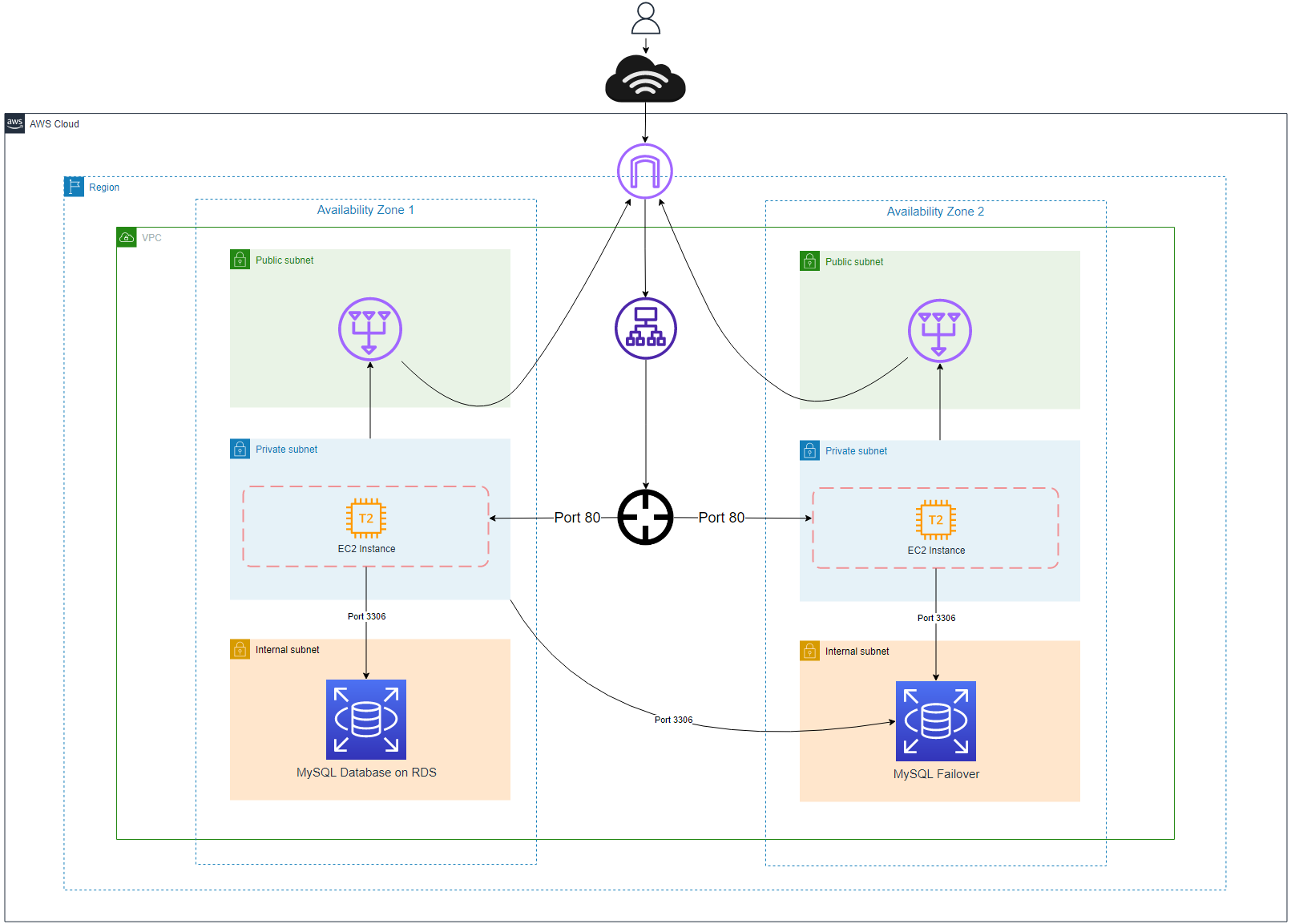
**Deployment 09 - Terraform**

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.

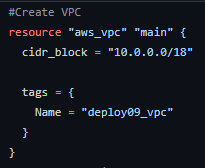


There are many benefits for Terraform. Terraform can make binary calls on your behalf to one or more providers such as AWS, Azure, GCP, etc. Terraform allows us to increase speed in development, decrease human error, reusability, scalable, self documentation, and be version controlled easily. Terraform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

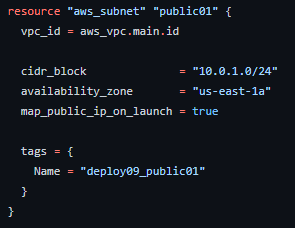
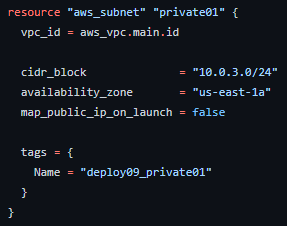
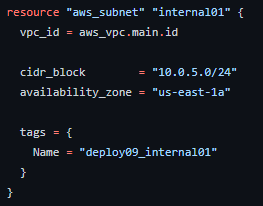
We specifically had to architect a VPC that consisted of subnets, EC2, security Groups, RDS MySQL database, application load balancer, and a target group. The simple flow of traffic would be users would access the internet and it would hit an internet gateway. Traffic will then be sent to the load balancer 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.

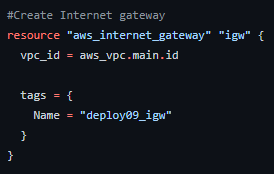


The next thing that we have to create are subnets. Subnets are simply a network inside a network. Companies use subnets to divide large networks into smaller, efficient subnetworks. This will help minimize traffic. For this assignment, we were tasked to create 2 public subnets, 1 private subnets, and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



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.

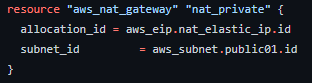
We will now need to create an internet gateway which will allow traffic inside and outside of our vpc. We will simply need to just attach it to our VPC.



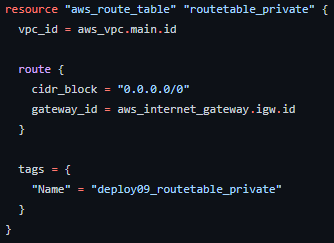
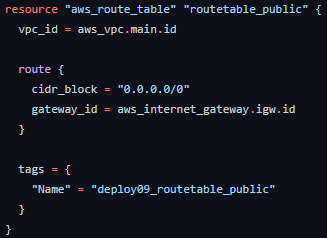
An elastic IP resource needs to be created so it can be attached to our NAT Gateway.



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.



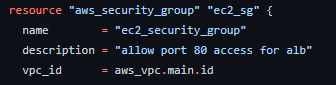
After the required resources were made, we would have to create a public and private route table. Route tables contain a set of rules that are used to determine where network traffic from your subnet is directed.



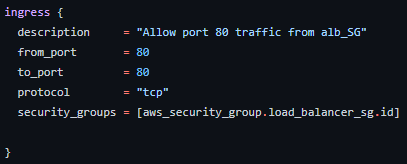
**Part 2 - EC2**

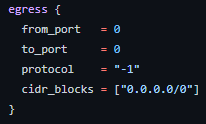
For this part, we are tasked to create an EC2 instance in a private subnet. While configuring the EC2, we have to select an Ubuntu AMI. The other configuration such as instance type, size, and tags are left up to our choosing. For security, we have to create a security group with certain ingress and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

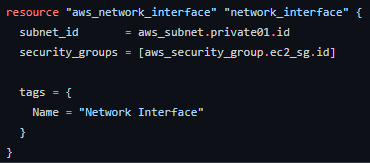


For ingress and egress there were certain requirements we had to follow. Ingress had to allow port 80 traffic from the ALB security group. This security group is for an upcoming security group we had to make. I decided to proceed to creating the current security group. For egress rules, we had to allow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

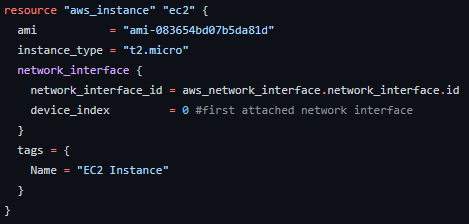




I had to find a way to attach this security group to the EC2 instance before creating it. I did some research and found out I had to create an aws\_network\_interface module. This will allow me to attach a security group and subnet. I can then specify the network card while creating the ec2.



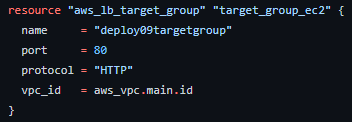
Finally, I can create the EC2 instance using the aws\_instance resource



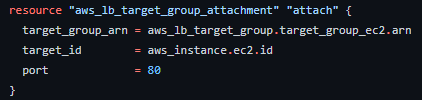
**Part 3 - ALB**

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.

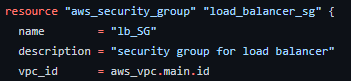
I decided on creating the target group first since the ALB depends on it. I used the “aws\_lb\_target\_group\_ resource'' and simply named it, linked the vpc, choose what type of port and protocol it should listen for.



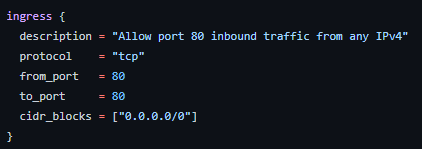
I then needed to attach the target group so I used the resource “aws\_lb\_target\_group\_attachment” which links the target group to the load balancer. You would then need to configure the ec2 that will be targeted and a specific port.



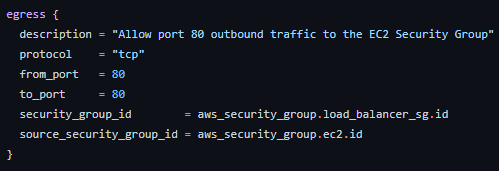
We can then create the security group for the load balancer. We start off by naming the security group, giving a description, and linking the vpc.



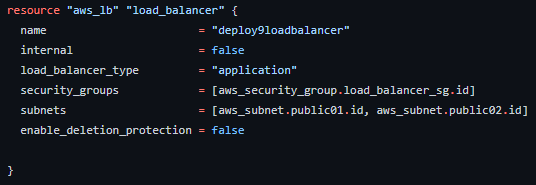
For the ingress rules, we are simply allowing only port 80 inbound from any ipv4.



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.



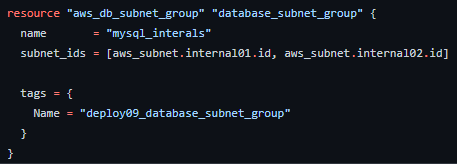
Finally, we can create the load balancer using the resource “aws\_lb”. While creating the load balancer, there are important configurations such as load\_balancer\_type which tells the load balancer what type it is. We need it to be an application load balancer so we will select that. We then need to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



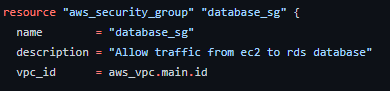
**Part 4 - RDS**

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.

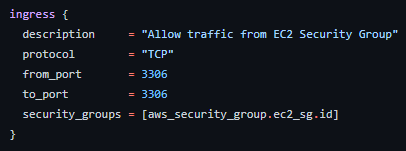
The first thing that I decided to tackle was creating the database subnet group. I used the resource “aws\_db\_subnet\_group”. I simply had to name it, and attach the two internal subnets



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.



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



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.



Once everything was set up I ran a few helpful commands

git checkout -b rdeodutt/deployment09

terraform fmt -recursive

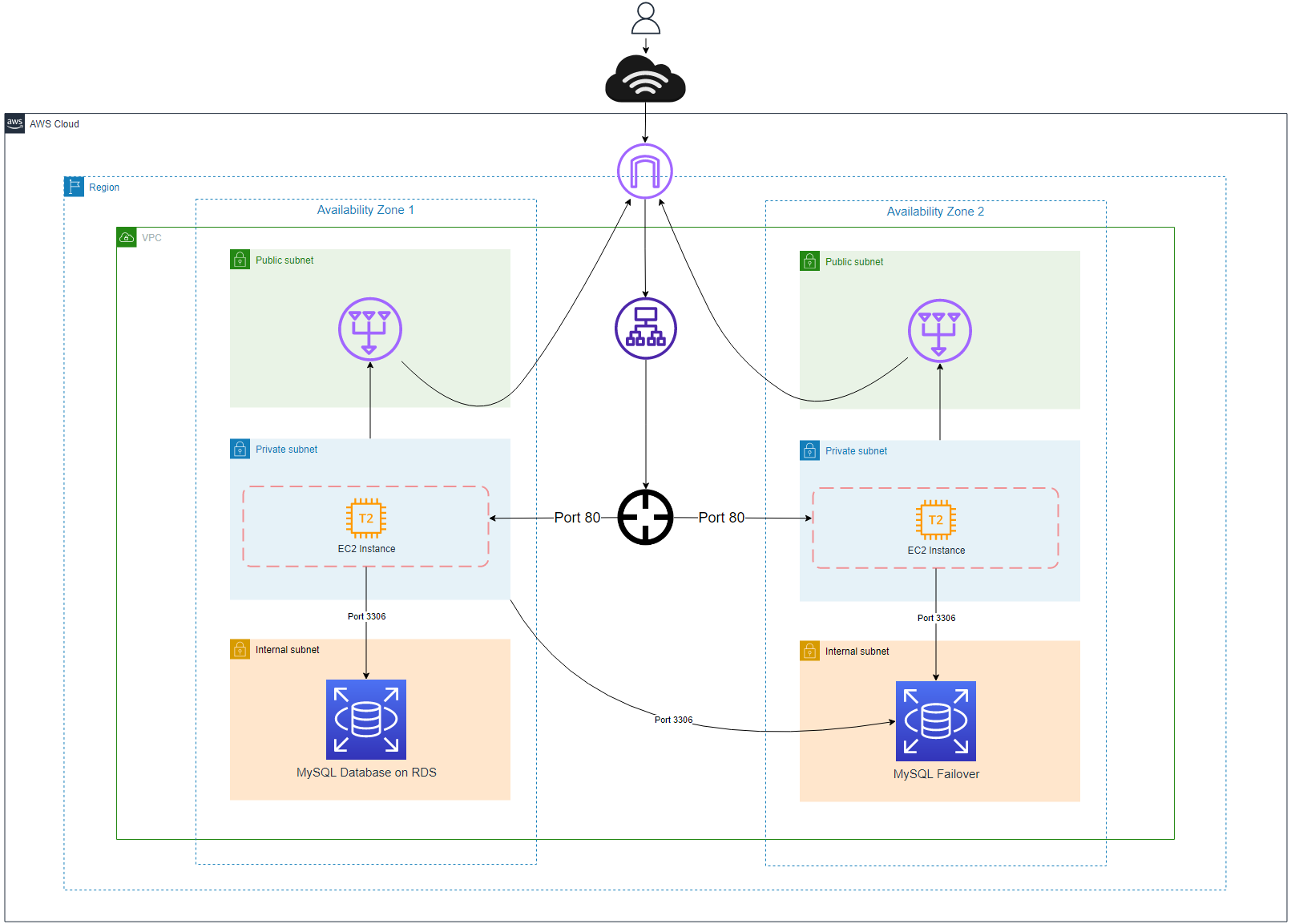
terraform init

terraform plan

terraform apply -auto-approve

terraform destroy -auto-approve

**Topology**

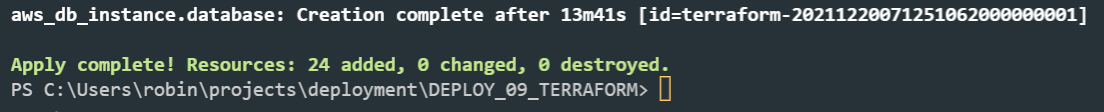


We are creating resources in the AWS Cloud. Inside the cloud we are hosting our VPC inside the US-EAST-1 region. VPCs are simply virtual private clouds. Think of it as an isolated network. Inside the vpc, we have two availability zones for resilience. In case one of our AZs goes down, we will have another 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.

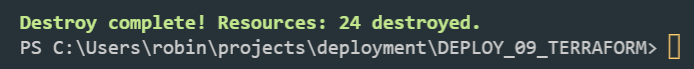
Inside the private subnet, we were tasked with creating an ec2 instance that has an Ubuntu AMI. The security group rules allow port 80 traffic from the load balancer only. The load balancer in this topology directs traffic from the internet gateway to the target group. The target group then directs traffic 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**

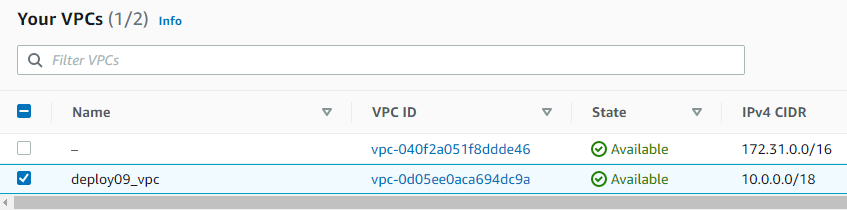
**Successful creation of infrastructure**



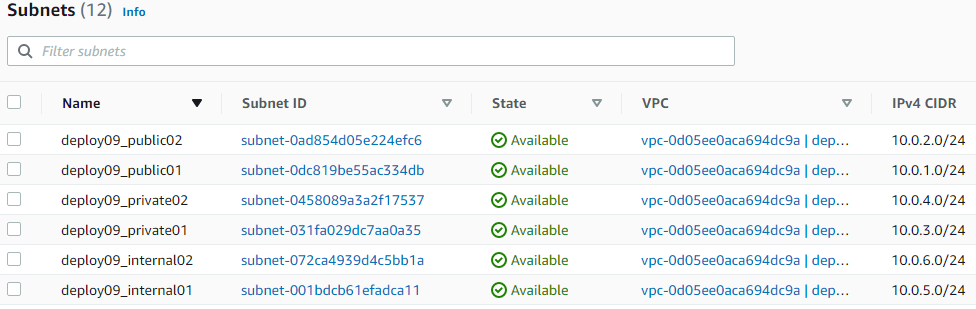
**Successful deletion of infrastructure**



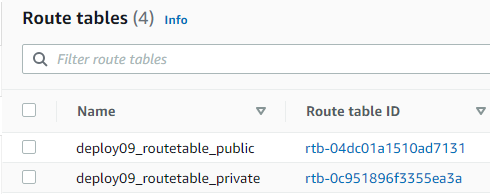
**VPC**



**Subnets**



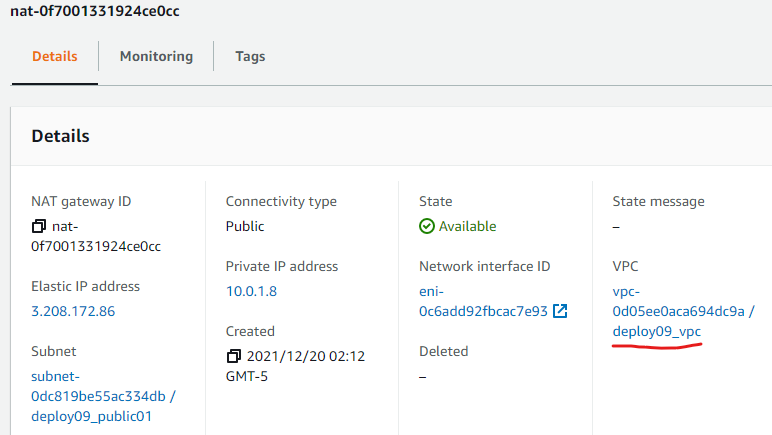
**Route Tables**



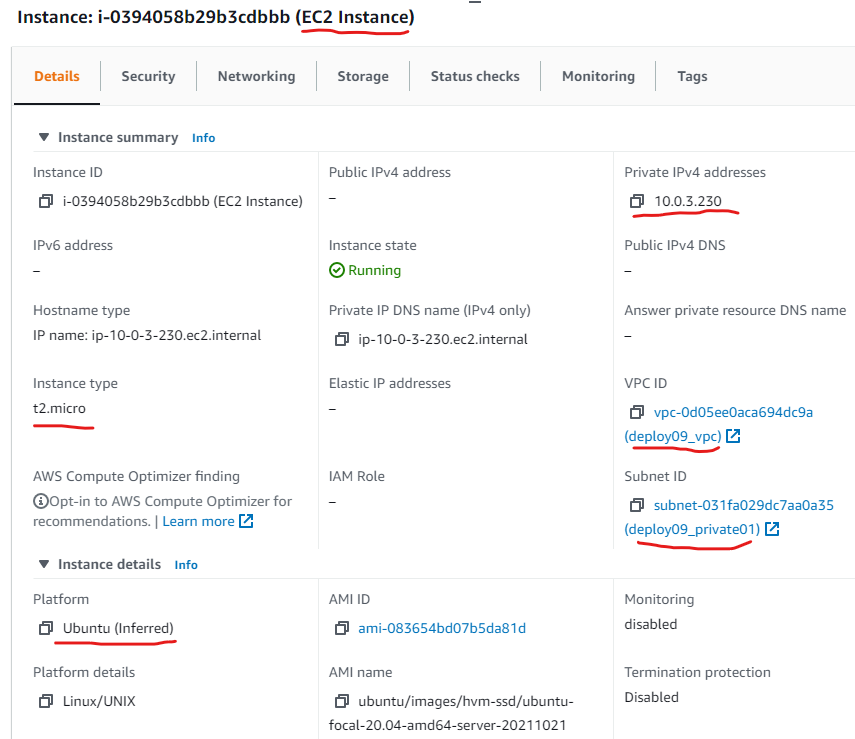
**Internet Gateway**



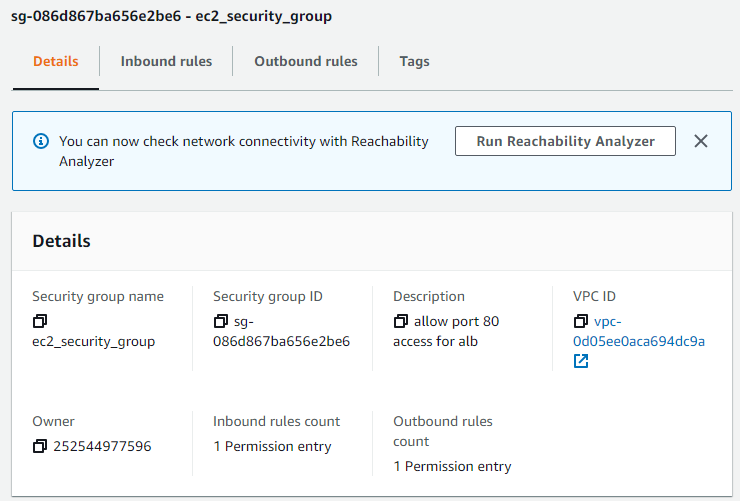
**NAT Gateway**



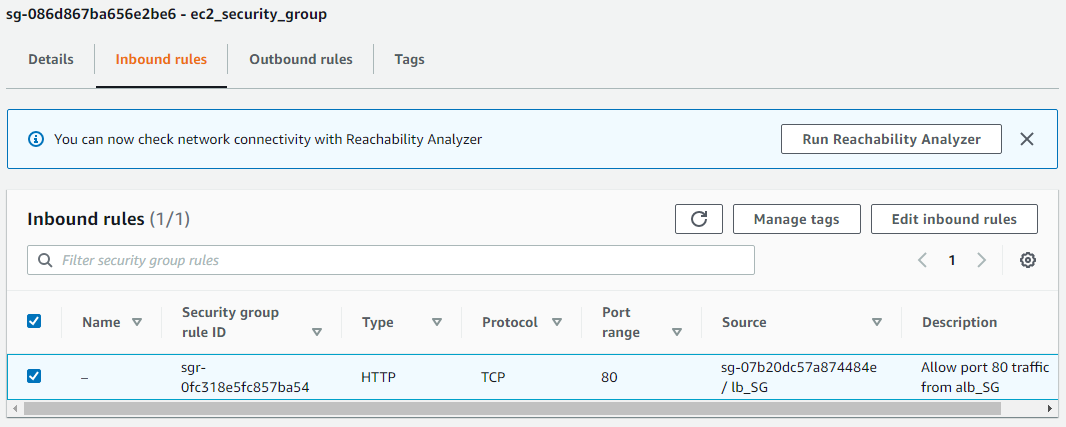
**EC2**



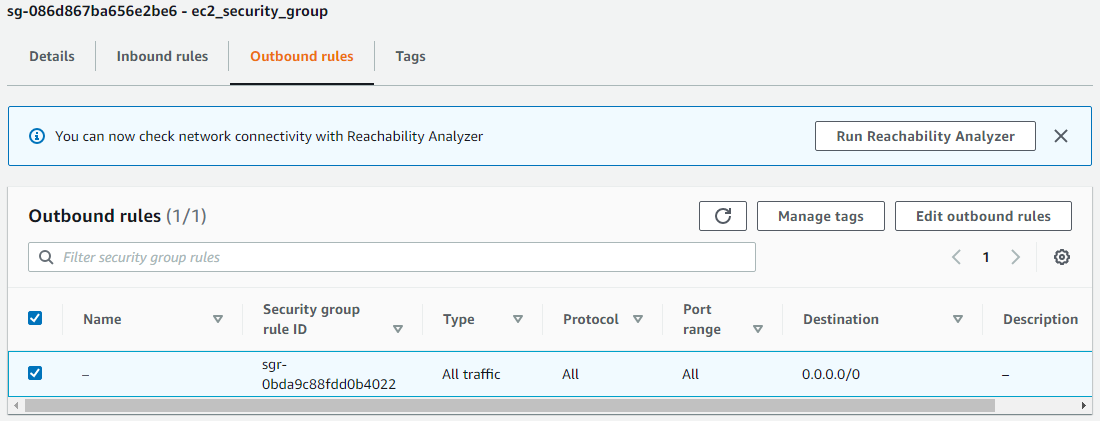
**Security Group**



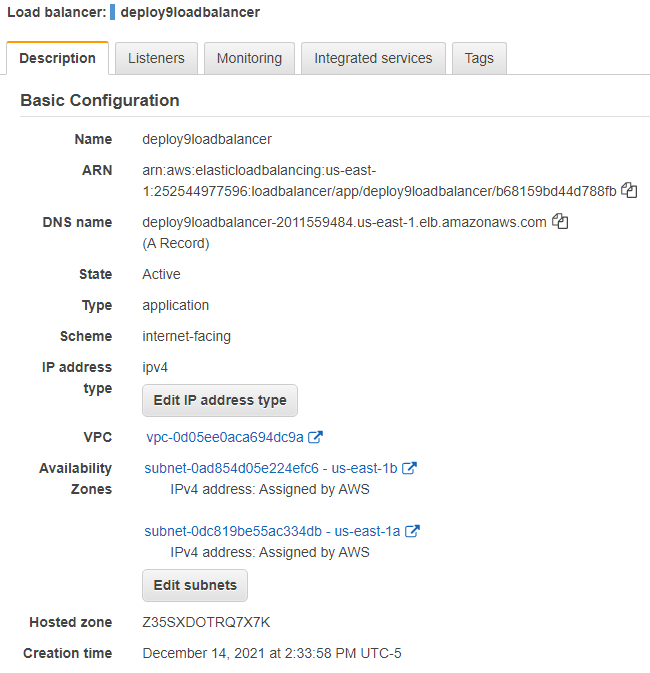
**Ingress/Inbound Security Group**



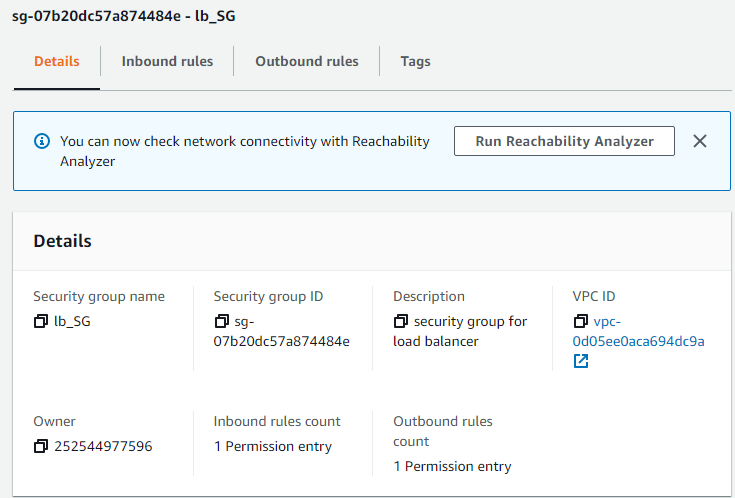
**Egress/Outbound Security Group**



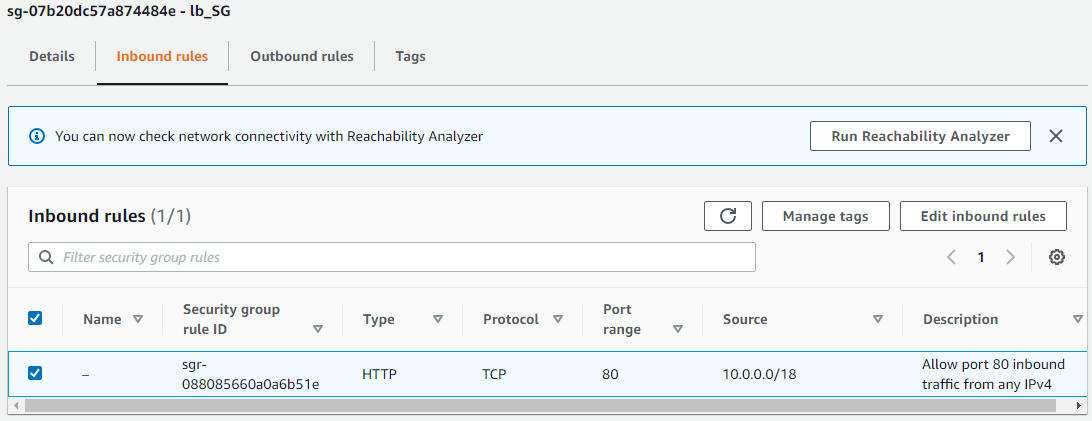
**Application Load Balancer (ALB)**



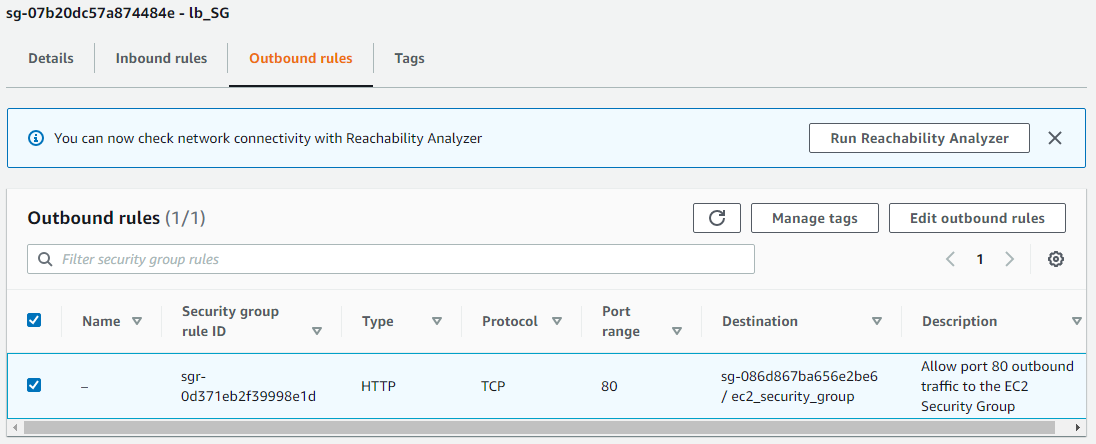
**ALB Security Group**



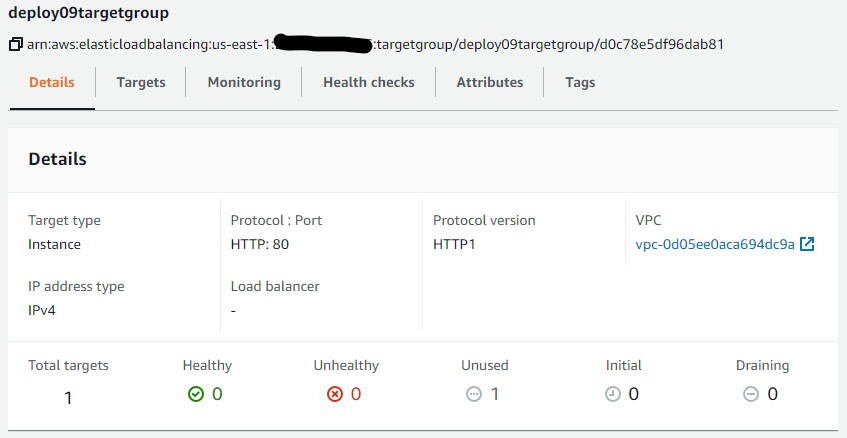
**ALB Security Group Ingress**



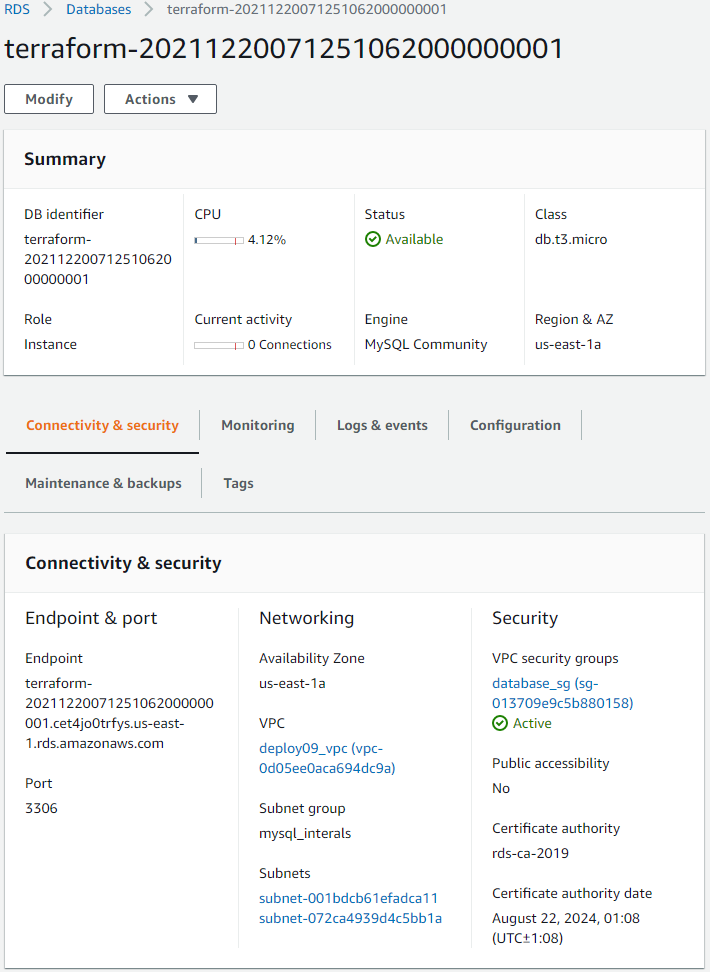
**ALB Security Group Egress**



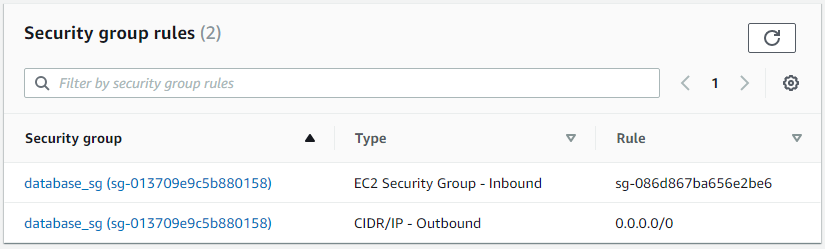
**Target Group**



**RDS Database**

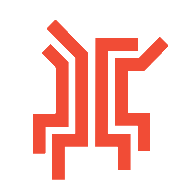


**RDS Database Security Group**



**RDS Database Subnet Group**





Ricardo Deodutt