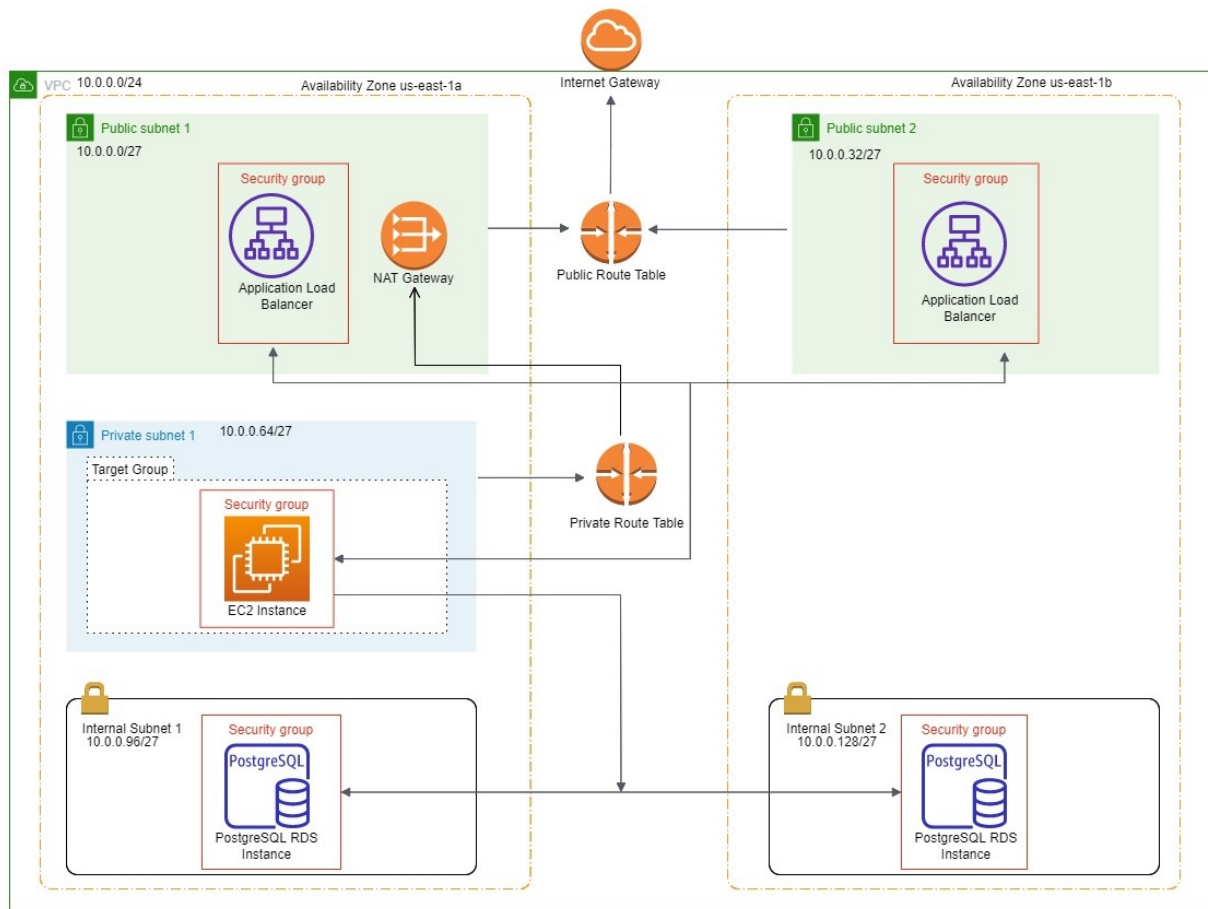


Denzel Gustave - Terraform

Deployment 9:

This deployment was largely documented in the terraform file itself in the form of comments explaining each resource provisioned. I followed a very precise naming scheme that helped me keep things organized, well structured and quicker to reference. There was a total of twenty-six resources provisioned in this deployment.

To help visualize what is being provisioned here, and better understand the infrastructure, I thought it would be interesting to draw out the architecture for this deployment.



- #1. Create VPC
- #2. Create Internet Gateway
- #3. Create public subnet 1
- #4. Create public subnet 2
- #5. Create private subnet 1
- #6. Create Custom Route Table for public subnets
- #7. Create Custom Route Table for private subnet
- #8. Associate public01 Subnet with public Route Table
- #9. Associate public02 Subnet with public Route Table
- #10. Associate private01 Subnet with private Route Table
- #11. Allocate Elastic IP to NAT Gateway
- #12. Create NAT Gateway
- #13. Create Ubuntu instance in private subnet
- #14. Create Security Group for private instance to allow port 80
- #15. Creating 1 application load balancer in the 2 public subnets
- #16. Create Security Group for ALB to allow port 80
- #17. Create a target group
- #18. Attach private instance to target group
- #19. Application load balancer listener to forward traffic to target group
- #20. Create postgresql rds instance
- #21. Create Security Group for database instance to allow port 80
- #FIXES
- #Creating ingress security group rule for deploy9-private-security group
- #Creating egress security group rule for deploy9-alb-security group
- #Create internal subnet 1
- #Create internal subnet 2
- #Create subnet group for internal subnets

Summary:

The VPC contains five subnets: two public subnets, one private subnet, and two internal subnets. The public subnets will have access to the internet via a route table that routes to the internet gateway. Whereas the private subnet will gain access to the internet via a route table that routes to a NAT gateway in the public subnet which then routes to the internet gateway. The two internal subnets do not have internet access but instead is limited to local resources.

The infrastructure has some level of resilience since it is provisioned across two availability zones. The application load balancers are each provisioned in a public subnet, the ec2 instance is provisioned in a private subnet and the PostgreSQL instances are in a multi-AZ deployment across each internal subnet.

The security group of the application load balancers outbound traffic only to the security group of the ec2 instance, and the security group of that ec2 instance only allow traffic from the security group of the application load balancers. The security group of the PostgreSQL instance allows both inbound and outbound traffic with only the ec2 instance.

Challenge and Solution:

I encountered and troubleshot a Terraform Cycle Error during this deployment.

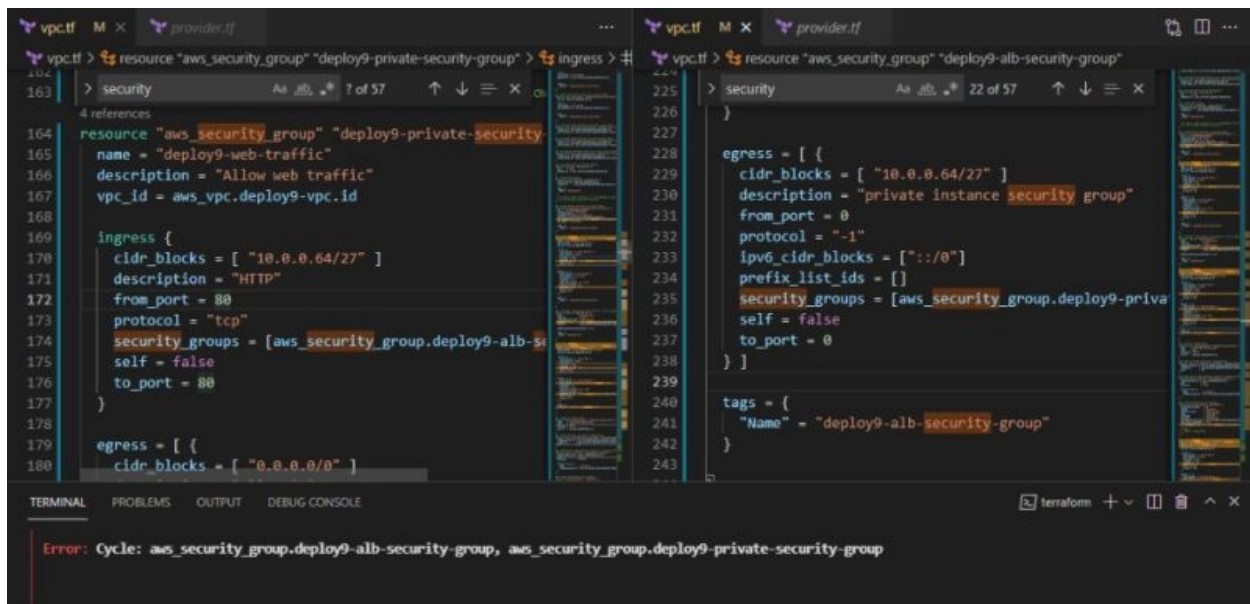
Terraform Cycle Errors are instances where resource dependencies rely on each other, therefore causing an infinite loop. Terraform is intelligent enough to realize this circular logic and terminates at that point.

It's able to realize this error because Terraform analyzes resource dependencies of your infrastructure configuration to determine the order of operations.

A cycle error logic looks like this:

(Sec1 -> Sec2) but (Sec2->Sec1)

I resolved this error by implementing security group rule resources which allowed me to abstract the policies from the security groups and storing them elsewhere. Therefore, allowing Terraform to create both security groups first and associating the policies after.



The screenshot shows a Terraform IDE with two files open: `vpctf` and `provider.tf`. The `vpctf` file contains two Terraform resources: `aws_security_group.deploy9-private-security-group` and `aws_security_group.deploy9-alb-security-group`. The `deploy9-private-security-group` resource has an `ingress` rule that depends on `aws_security_group.deploy9-alb-security-group`. The `deploy9-alb-security-group` resource has an `egress` rule that depends on `aws_security_group.deploy9-private-security-group`. This creates a circular dependency. The terminal at the bottom displays the error message: `Error: Cycle: aws_security_group.deploy9-alb-security-group, aws_security_group.deploy9-private-security-group`.

```
163 > security
164 resource "aws_security_group" "deploy9-private-security-group" {
165   name = "deploy9-web-traffic"
166   description = "Allow web traffic"
167   vpc_id = aws_vpc.deploy9-vpc.id
168
169   ingress {
170     cidr_blocks = [ "10.0.0.0/27" ]
171     description = "HTTP"
172     from_port = 80
173     protocol = "tcp"
174     security_groups = [ aws_security_group.deploy9-alb-security-group ]
175     self = false
176     to_port = 80
177   }
178
179   egress = [ {
180     cidr_blocks = [ "0.0.0.0/0" ]
181   } ]
182 }

225 > security
226
227
228
229 egress = [ {
230   cidr_blocks = [ "10.0.0.0/27" ]
231   description = "private instance security group"
232   from_port = 0
233   protocol = "-1"
234   ipv6_cidr_blocks = [ ":::/0" ]
235   prefix_list_ids = []
236   security_groups = [ aws_security_group.deploy9-private-security-group ]
237   self = false
238   to_port = 0
239 } ]
240
241 tags = {
242   "Name" = "deploy9-alb-security-group"
243 }
244 }
```

Terminal Output:

```
Error: Cycle: aws_security_group.deploy9-alb-security-group, aws_security_group.deploy9-private-security-group
```