VPC subnets/security groups

1. Welcome back to the VPC setup in our Terraform and Ansible for AWS course on Linux Academy!
2. Now that we have setup our subnets and route tables, we need to do a few more things to get our VPC working properly.
3. First, we need to associate our subnets with the proper route tables. Since the private route table is the default, we don’t have to make any explicit associations, but we do need to make those associations for the public route table. The subnets we need to associate with the public are: the public (obviously), and the two private subnets. Why do we have to associate the private subnets with the public route table you ask? It is for the load balancer to be able to access the outside world as it will be placed in the same subnet as the private instances. This is why we ensured the private subnets do not assign public IP addresses to their instances. This limitation can be overcome by a NAT or other, more advanced configurations, but that is beyond the scope of this course. So, let’s walk through associating the public subnet with the public route table, then I will show you what the others should look like:

resource “aws\_route\_table\_association” “public\_assoc” {

subnet\_id = “${aws\_subnet.public.id}”

route\_table\_id = “${aws\_route\_table.public.id}”

}

As you can see, we have called the aws\_route\_table\_association resource and given it an ID of public\_assoc. I have then set the subnet\_id as the public subnet and the route\_table\_id as the public route table.

Now I have added the other associations we need. Private1 and private2. Each one has the same information, aside from the subnet\_id, as the public, so feel free to copy and paste then modify them accordingly.

1. Now we need to create an RDS subnet group. Amazon can do this automatically for us, but like I said before, I prefer to have complete control over what we do. It makes referencing these things much easier when you start to expand your infrastructure.

resource “aws\_db\_subnet\_group” “rds\_subnetgroup” {

name = “rds\_subnetgroup”

subnet\_ids = [“${aws\_subnet.rds1.id}”, “${aws\_subnet.rds2.id}”, “${aws\_subnet.rds3.id}”]

tags {

Name = “rds\_sng”

}

}

1. Alright, that was fun, now on to the next section of our VPC, the security groups!
   1. The first security group we will create is the Public security group. This group will allow access from anywhere on port 80 and from port 22 only from your IP address. You can certainly add more IP addresses, or open it up to the world, but I certainly don’t recommend having SSH open to the world to brute force.

resource “aws\_security\_group” “public” {

name = “sg\_public”

description = “Used for public instances”

vpc\_id = “${aws\_vpc.vpc.id}”

# SSH access from your local ip, $(curl canhazip.com) is one good way to find it. The reason we don’t want to automate this is because you may need access from another computer other than the one the script is running on.

ingress {

from\_port = 22

to\_port = 22

protocol = “tcp”

cidr\_blocks = [“${var.localip}”]

}

#HTTP access from anywhere. You can also restrict this just to your computer if you want.

ingress {

from\_port = 80

to\_port = 80

protocol = “tcp”

cidr\_blocks = [“0.0.0.0/0”]

}

#Outbound internet access

egress {

from\_port = 0

to\_port = 0

protocol = “-1”

cidr\_blocks = [“0.0.0.0/0”]

}

}

Let’s walk through this. First of all, we create our resource “aws\_security\_group”, then we give it an id of “public”. We then give it a name, a description (this is good for auditing security rules and I strongly recommended it, as will any security advisors you encounter.), them the VPC id.

Under that, we start our rules. We have ingress rules and egress rules. These are essentially technical terms for “incoming” and “outgoing”.

We first set the access for your local IP address. You can see the SSH port 22 listed. Obviously, if you change your SSH ports, this should be noted here, but since this server is only accessible to you, the need to change the port isn’t quite as dire. We then specify the TCP protocol, and call our localip variable. This variable will need to be added to the variables.tf file and the terraform.tfvars file. If you want the variable to ask you every time, you can skip the tfvars file and just add localip{} to your variables.tf file. Go ahead and pause the video and add the variable now so you don’t forget.

Next, we add HTTP access from the world. 0.0.0.0/0 specifies any ip address as always.

Then we have to give the resources contained in the security group outbound access. Using 0 for the ports and -1 for the protocol will allow this.

So, a point of clarity here, you may have noticed that we have brackets around the cidr\_blocks field. This is to indicate the ability to add a list of multiple values if you wish. You can typically expect to use brackets in a terraform script if the key is pluralized such as “cidr\_blocks” in this case. Keep that in mind as that small syntax error can certainly cause a script to crash.

As you can see, there is a decent amount to type for just a few rules, so I’m going to copy and paste the other required blocks here for you to copy yourself. The concepts are the same, just different names. Keep in mind that I have also added a rule to allow access from other security groups by just allowing the full vpc subnet.

Another thing I’d like to point out is the difference between the ingress in the private security group and the RDS security group. If you notice, I have allowed the subnet cidr\_block in the private and the name of the security groups in the RDS. This is due to the fact that Terraform can’t build both security groups simultaneously, so one will fail if it tries to reference a security group that doesn’t exist yet, so I just used the subnet for the private since it is created first, then the security groups for the RDS.