CSL 1: Assignment 9: **Terraform**

Create a folder for your Terraform files and put the files provided here. Download here in the folder. Navigate to the folder and run the commands:

```
terraform init
terraform apply
```

This will create a simple VPC with a subnet.

```
Plan: 2 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_vpc.main: Creating...
aws_vpc.main: Creation complete after 2s [id=vpc-010fbd50695cebac6]
aws_subnet.public: Creating...
aws_subnet.public: Creation complete after 1s [id=subnet-069eb31b7168ce7d6]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
```

| | | | |
|---|---|---|---|
| CYBERSECURITY_VPC | vpc-010fbd50695cebac6 | Available | 10.0.0.0/16 |

| | | |
|---|---|---|
| CYBERSECURITY_SUBNET_PUB | subnet-069eb31b7168ce7d6 | 10.0.1.0/24 |

**Task 1**: Modify the file **subnets.tf** inside the provided .zip file to create a private subnet along with the existing public subnet. Hint: The command *terraform apply* "uploads" new changes to AWS. Use this command to check if your "code" works properly.

- Modified subnets.tf file:

```
resource "aws_subnet" "public" {
  vpc_id            = aws_vpc.main.id
  cidr_block        = "10.0.1.0/24"
  availability_zone = "us-east-1b"
  tags = {
    Name = "CYBERSECURITY_SUBNET_PUB"
  }
}

resource "aws_subnet" "private" {
  vpc_id            = aws_vpc.main.id
  cidr_block        = "10.0.2.0/24"
  availability_zone = "us-east-1b"
  tags = {
    Name            = "CYBERSECURITY_SUBNET_PRIV"
    InstanceManager = "terraform-prasanna"
  }
}
```

- Changes done after "terraform apply": One new subnet has been created under our VPC.

```
Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_subnet.private: Creating...
aws_subnet.private: Creation complete after 1s [id=subnet-0c3

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

| | | |
|---|---|---|
| CYBERSECURITY_SUBNET_PRIV | subnet-0711720e0529199c1 | 10.0.2.0/24 |

- Prasanna Venkatesan Aravindan

**Task 2**: Create a file named **gateways.tf** and place the "code" needed to create an internet gateway that is associated with the VPC terraform creates.

- Contents of the gateways.tf file:

```
resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.main.id

  tags = {
    Name            = "CYBERSEC-IGW"
    InstanceManager = "terraform-prasanna"
  }
}
```

**Task 3**: Create a file named **route-tables.tf** and write the "code" needed to create the proper route-table for the internet-gateway as well as the subnet association for the public subnet.

- Contents of the route-tables.tf file:

```
resource "aws_route_table" "igw-rt" {
  vpc_id = aws_vpc.main.id
  route {
    cidr_block = "0.0.0.0/0" # connections to the internet
    gateway_id = aws_internet_gateway.igw.id
  }
  tags = {
    Name            = "IGW-RT"
    InstanceManager = "terraform-prasanna"
  }
}

resource "aws_route_table_association" "pub_snet_association" {
  route_table_id = aws_route_table.igw-rt.id
  subnet_id      = aws_subnet.public.id
}
```

**Task 4**: Create a file called **security-groups.tf** and write the "code" required to create security group(s) that allows inbound traffic for SSH, and for TCP for port 8081, and all outgoing traffic.

HINT: You should probably consider not putting all the rules inside one resource. It is a better idea to separate the groups so you can assign a subset of the groups to a public EC2 and another subset to a private EC2 without having groups with repeating rules. This means that instead of creating one group with all the rules, you can create one group for each rule.

- Inside security-groups.tf, I have created two blocks of code to create two separate security groups for public and private subnets.
- The Public Subnet resources will have open access. But the private subnet resources can be accessed only within same VPC & not have internet access.

CSL 1: Assignment 9: **Terraform**

Public Subnet Security Group:

```
resource "aws_security_group" "pub_sg" {
  name        = "CYBERSEC-PUB-SG"
  description = "SG for Public Subnet"
  vpc_id      = aws_vpc.main.id

  # Ingress Rule-1: SSH Access to Public Subnet
  ingress {
    description = "Allow SSH from anywhere"
    from_port   = 0 # any source port
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"] # any source

  }

  # Ingress Rule-2: Access to Public Subnet via port 8081
  ingress {
    description = "Allow 8081 from anywhere"
    from_port   = 0
    to_port     = 8081
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  # Egress Rule-1: Allow internet access
  egress {
    description = "Open outbound connection"
    from_port   = 0
    to_port     = 0
    protocol    = "-1" # Allow any protocol
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

Private Subnet Security Group:

```
resource "aws_security_group" "priv_sg" {
  name        = "CYBERSEC-PRIV-SG"
  description = "SG for Private Subnet"
  vpc_id      = aws_vpc.main.id

  # Ingress Rule-1: SSH Access to Private Subnet
  ingress {
    description = "Allow SSH Access from same VPC"
    from_port   = 0
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = [aws_vpc.main.cidr_block] # connections from VPC
  }

  # Egress Rule-1: Allow internet access
  egress {
    description = "Open outbound connection"
    from_port   = 0
    to_port     = 0
    protocol    = "-1" # Allow any protocol
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

- Prasanna Venkatesan Aravindan

**Task 5**: Create a file called **ami.tf** and place the required "code" to create an image based on Ubuntu server 22.04 AMD 64. Your code should create an AMI from ubuntu's AMI.

```
resource "aws_ami_copy" "my-ami" {
  name              = "cybersec-ami"
  source_ami_id     = "ami-0fc5d935ebf8bc3bc"
  source_ami_region = var.region

  tags = {
    Name            = "CYBERSEC-AMI"
    InstanceManager = "terraform-prasanna"
  }
}
```

- This code creates a new AMI from the existing Ubuntu image in us-east-1 region.

**Task 6**: Create a file called **ec2.tf** and write the "code" to create an EC2 instance based on the AMI from Task 5 **inside the public subnet** the provided code creates. The new public instance adopts all the rules that are mentioned in Task 4. You should also assign a key to this instance so that in later tasks you can access it via SSH. Also set the option associate_public_ip_address = true.

- Pre-requisite: Creating an SSH Key pair locally

```
$ ssh-keygen.exe
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/prasa/.ssh/id_rsa): cybersec-ec2-kp
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in cybersec-ec2-kp
Your public key has been saved in cybersec-ec2-kp.pub
The key fingerprint is:
```

- Import this public key to AWS & use this key while creating the EC2 instance.

```
# Import Key Pair
resource "aws_key_pair" "ec2-kp" {
  key_name   = "CYBERSECURITY_EC2_PUB"
  public_key = file("cybersec-ec2-kp.pub")
}
```

```
# Create an EC2 Instance - Public Subnet
resource "aws_instance" "my-ec2-pub" {
  ami                         = aws_ami_copy.my-ami.id
  instance_type               = "t2.micro"
  associate_public_ip_address = true
  subnet_id                   = aws_subnet.public.id
  availability_zone           = aws_subnet.public.availability_zone
  key_name                    = aws_key_pair.ec2-kp.key_name
  vpc_security_group_ids      = [aws_security_group.pub_sg.id]

  tags = {
    Name            = "CYBERSEC-PUB-EC2"
    InstanceManager = "terraform-prasanna"
  }

  # Connect to Public EC2 to perform provisioning
  connection {
    user        = var.vm-user # ubuntu user
    private_key = file("cybersec-ec2-kp") # use this private key
    host        = self.public_ip
  }

  # Import Private Key to access Private EC2
  provisioner "file" {
    source      = "cybersec-ec2-kp"          # name of the private key
    destination = "/home/${var.vm-user}/cybersec-ec2-kp" # save in remote public ec2 home
  }

  provisioner "remote-exec" {

    inline = [ "chmod 600 cybersec-ec2-kp" ] # change permission of the pvt key

  }
}
```

- Note: As per the Private Subnet Security Group rule, the private instance can be accessed only within VPC (i.e., from Public Subnet). So, to connect to the private instance, we will need to import the private key inside the public instance. For that, I will need to export the private key from our local machine to the public instance. To do this, I have added the provisioner block (for file transfer) & connection block (we must connect to the instance to perform file transfer).

**Task 7**: In the ec2.tf file you created in Task 6, write the "code" to create an EC2 instance based on the AMI you created in Task 5. This new instance should be **inside the private subnet** you created in Task 1. It also should have outgoing traffic to everywhere and be accessible through SSH. Again, you must specify a key so that you can SSH to that machine later. Also set the associate_public_ip_address = false.

HINT: You can use some of the security groups you created in Task 4 for that, just keep in mind not to allow any inbound traffic other than SSH. In the case you created one security group in Task 4, you will probably need to create and attach another security group for this EC2 instance.

```
# Create an EC2 Instance - Private Subnet
resource "aws_instance" "my-ec2-priv" {
  ami                         = aws_ami_copy.my-ami.id
  instance_type               = "t2.micro"
  associate_public_ip_address = false
  subnet_id                   = aws_subnet.private.id
  availability_zone           = aws_subnet.private.availability_zone
  key_name                    = aws_key_pair.ec2-kp.key_name
  vpc_security_group_ids      = [aws_security_group.priv_sg]

  tags = {
    Name            = "CYBERSEC-PRIV-EC2"
    InstanceManager = "terraform-prasanna"
  }
}
```

**Task 8**: If you have created the EC2 instances properly you should be able to SSH into the EC2 private instance created in Task 7 through the EC2 public instance created in Task 6. After you SSH into the private instance try to ping *www.google.com* & it should not work. If you try the same from the EC2 public instance you will see that you can ping www.google.com. Explain briefly why on the EC2 private instance you cannot ping google whereas on the EC2 public instance you can.

- Terraform Validate:

```
$ terraform validate
Success! The configuration is valid.
```

- Terraform Apply:

```
 Enter a value: yes

aws_internet_gateway.igw: Creating...
aws_key_pair.ec2-kp: Creating...
aws_ami_copy.my-ami: Creating...
aws_security_group.priv_sg: Creating...
aws_security_group.pub_sg: Creating...
aws_key_pair.ec2-kp: Creation complete after 1s [id=CYBERSECURITY_EC2_PUB]
aws_internet_gateway.igw: Creation complete after 1s [id=igw-0e059f9e156ec5c19]
aws_route_table.igw-rt: Creating...
aws_route_table.igw-rt: Creation complete after 1s [id=rtb-0204be3143b81fa13]
aws_route_table_association.pub_snet_association: Creating...
aws_route_table_association.pub_snet_association: Creation complete after 1s [id=
aws_security_group.priv_sg: Creation complete after 3s [id=sg-04b9cdb0acf6cbf5d]
aws_security_group.pub_sg: Creation complete after 3s [id=sg-05250f31503774369]
aws_ami_copy.my-ami: Still creating... [10s elapsed]
aws_ami_copy.my-ami: Still creating... [20s elapsed]
aws_ami_copy.my-ami: Still creating... [30s elapsed]
aws_ami_copy.my-ami: Still creating... [40s elapsed]
aws_ami_copy.my-ami: Still creating... [50s elapsed]
aws_ami_copy.my-ami: Still creating... [1m0s elapsed]
aws_ami_copy.my-ami: Creation complete after 1m6s [id=ami-0d5440f17d192dcaf]
aws_instance.my-ec2-priv: Creating...
aws_instance.my-ec2-pub: Creating...
aws_instance.my-ec2-priv: Still creating... [10s elapsed]
aws_instance.my-ec2-pub: Still creating... [10s elapsed]
aws_instance.my-ec2-pub: Still creating... [20s elapsed]
aws_instance.my-ec2-priv: Still creating... [20s elapsed]
aws_instance.my-ec2-pub: Still creating... [30s elapsed]
aws_instance.my-ec2-priv: Still creating... [30s elapsed]
aws_instance.my-ec2-pub: Provisioning with 'file'...
aws_instance.my-ec2-priv: Creation complete after 33s [id=i-0ac4aaefc8c8fa3a0]
```

- Internet connection status of public subnet instance: SUCCESS

```
ubuntu@ip-10-0-1-30:~$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=58 time=0.636 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=58 time=0.783 ms
^C
--- 8.8.8.8 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1017ms
rtt min/avg/max/mdev = 0.636/0.709/0.783/0.073 ms
```

- SSH to private instance (the private key has already been moved to the default folder by terraform)

```
ubuntu@ip-10-0-1-30:~$ ls
cybersec-ec2-kp          pvt key
ubuntu@ip-10-0-1-30:~$ ssh -i cybersec-ec2-kp ubuntu@10.0.2.66
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 6.2.0-1012-aws x86_64)
```

- Internet connectivity status of private EC2 instance: FAILURE

```
ubuntu@ip-10-0-2-66:~$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
^C
--- 8.8.8.8 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3066ms
```

- Why private instance cannot connect to the internet, although outbound connection was allowed?

This is because, as per the configured Route table, we haven't fixed the path for internet connection (0.0.0.0/0) & we only have the path for VPC. As no route is present, the packets are dropped by the instance. To resolve this, we need to create a NAT gateway in the public subnet, and associate the private subnet with a route table, so that connections to 0.0.0.0/0 are pointing to the NAT gateway. (Note: This NAT gateway should have an elastic IP associated)

**Task 9**: What would you change in your "code" to let the EC2 private instance to connect to the Internet (please consider using the best practices since this is an instance in a private subnet)?

- I have created another file, called "**nat-gateway.tf**" which performs the below tasks.
- Create Route Table for private subnet.
- Create an Elastic IP.
- Create a NAT gateway in the public subnet & use the Elastic IP.
- Add a route for internet connectivity (0.0.0.0/0) pointing to the NAT gateway.

```
# Create a route-table for Private subnet
resource "aws_route_table" "priv-rt" {
  vpc_id = aws_vpc.main.id
  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_nat_gateway.nat-gw.id
  }
  tags = {
    Name = "Priv-RT"
    InstanceManager = "terraform-prasanna"
  }
}

# Create an elastic IP for NAT Gateway
resource "aws_eip" "elastic-ip" {
  domain = "standard" # any public ip
}

# Create a NAT Gateway
resource "aws_nat_gateway" "nat-gw" {
  subnet_id = aws_subnet.public.id
  allocation_id = aws_eip.elastic-ip.id
}


# Associate Private subnet with Private RT
resource "aws_route_table_association" "priv_snet_assoc" {
  route_table_id = aws_route_table.priv-rt.id
  subnet_id = aws_subnet.private.id
}
```

Terraform apply:

```
Plan: 4 to add, 1 to change, 0 to destroy.
```

- After the NAT gateway is setup, the private VM instance connects to the internet with the NAT gateway IP.

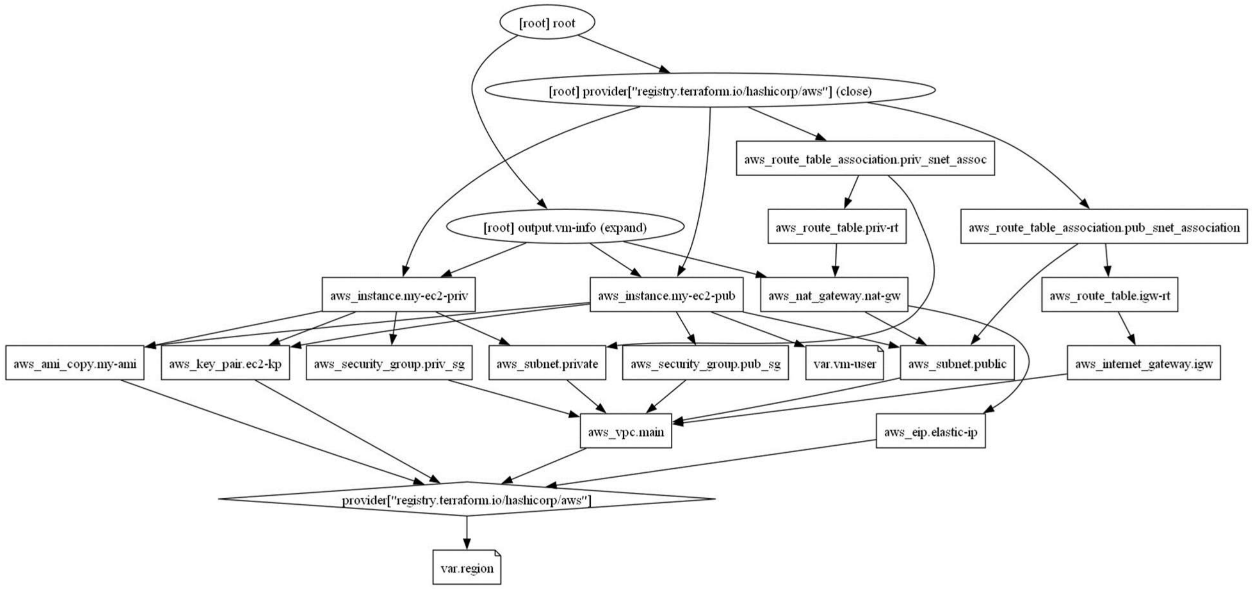
```
$ terraform output
vm-info = <<EOT
VM Username: ubuntu
Public VM IP: 54.210.13.228
Private VM IP: 10.0.2.66
NAT Gateway IP: 54.211.161.42

EOT
```

```
ubuntu@ip-10-0-2-66:~$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=115 time=2.08 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=1.57 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=115 time=1.57 ms
^C
--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 1.567/1.740/2.082/0.241 ms
ubuntu@ip-10-0-2-66:~$ curl ifconfig.io
54.211.161.42
ubuntu@ip-10-0-2-66:~$
```

NAT GW IP

Extra Note: Below is the Terraform code Resource dependency graph.




- Prasanna Venkatesan Aravindan