Task 1 (11%): Modify the file `subnets.tf` inside the provided .zip file to create a private subnet along with the existing public subnet.

```
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-1c"
  tags = {
    Name = "CYBERSECURITY_SUBNET_PRIV"
  }
}
```

Task 2 (11%): Create a file named `gateways.tf` and place the "code" needed to create an internet gateway that is associated with the VPC terraform creates.

```
resource "aws_internet_gateway" "gw" {
  vpc_id = aws_vpc.main.id
  tags = {
    Name = "main"
  }
}
```

Task 3 (11%): 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.

```
resource "aws_route_table" "routetable" {
  vpc_id = aws_vpc.main.id
  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.gw.id
  }
  tags = {
    Name = "routetable"
  }
```

```
}
resource "aws_route_table_association" "a" {
  subnet_id      = aws_subnet.public.id
  route_table_id = aws_route_table.routetable.id
}
```

Task 4 (11%): 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 of 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.

```
resource "aws_security_group" "allow_ssh" {
  name        = "allow_ssh"
  description = "Allow SSH inbound traffic"
  vpc_id      = aws_vpc.main.id
  ingress {
    description      = "SSH from VPC"
    from_port        = 22
    to_port          = 22
    protocol         = "tcp"
    cidr_blocks      = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }
  tags = {
    Name = "allow_ssh"
  }
}
resource "aws_security_group" "allow_tcp_8081" {
  name        = "allow_tcp_8081"
  description = "Allow TCP 8081 inbound traffic"
  vpc_id      = aws_vpc.main.id
  ingress {
    description      = "TCP 8081 from VPC"
    from_port        = 8081
    to_port          = 8081
    protocol         = "tcp"
    cidr_blocks      = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }
```

```
  tags = {
    Name = "allow_tcp_8081"
  }
}
resource "aws_security_group" "allow_all_outgoing" {
  name        = "allow_all_outgoing"
  description = "Allow all outbound traffic"
  vpc_id      = aws_vpc.main.id
  egress {
    from_port        = 0
    to_port          = 0
    protocol         = "-1"
    cidr_blocks      = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }
  tags = {
    Name = "allow_all_outgoing"
  }
}
```

Task 5 (11%): Create a file called `ami.tf` and place the required "code" in order to create an image based on Ubuntu server 18.04 AMD 64. Your code should create an AMI from ubuntu's AMI.

```
data "aws_ami" "ubuntu" {
  most_recent      = true
  owners           = ["amazon"]
  filter {
    name   = "name"
    values = ["ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-server-
20220901"]
  }
}
resource "aws_ami_copy" "Ubuntu1804" {
  name              = "Ubuntu1804"
  description       = "A copy of Ubuntu server 18.04 AMD 64"
  source_ami_id     = data.aws_ami.ubuntu.id
  source_ami_region = "us-east-1"
  tags = {
    Name = "Ubuntu1804"
  }
}
```

```
aws_ami_copy.Ubuntu1804: Still creating... [4m0s elapsed]
aws_ami_copy.Ubuntu1804: Still creating... [4m10s elapsed]
aws_ami_copy.Ubuntu1804: Still creating... [4m20s elapsed]
aws_ami_copy.Ubuntu1804: Still creating... [4m30s elapsed]
aws_ami_copy.Ubuntu1804: Still creating... [4m40s elapsed]
aws_ami_copy.Ubuntu1804: Still creating... [4m50s elapsed]
aws_ami_copy.Ubuntu1804: Still creating... [5m0s elapsed]
aws_ami_copy.Ubuntu1804: Creation complete after 5m9s [id=ami-0f4e2c7ba4c584
8ec]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
PS D:\Course\22FALL-CMPT732\terraform>
```

Task 6 (11%): 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`.

```
resource "aws_instance" "ubuntu_public " {
  ami                         = aws_ami_copy.Ubuntu1804.id
  instance_type               = "t2.micro"
  subnet_id                   = aws_subnet.public.id
  vpc_security_group_ids      = [aws_security_group.allow_ssh.id,
aws_security_group.allow_tcp_8081.id,
aws_security_group.allow_all_outgoing.id]
  associate_public_ip_address = true
  key_name                    = "CYBERSECURITY_EC2_PUB"

  tags = {
    Name = "ubuntu_public"
  }
}
```

```
aws_instance.ubuntu: Creating...
aws_instance.ubuntu: Still creating... [10s elapsed]
aws_instance.ubuntu: Still creating... [20s elapsed]
aws_instance.ubuntu: Still creating... [30s elapsed]
aws_instance.ubuntu: Still creating... [40s elapsed]
aws_instance.ubuntu: Creation complete after 42s [id=i-0abb552a1d5a39503]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
PS D:\Course\22FALL-CMPT732\terraform>
```

Task 7 (11%): 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 have to specify a key so that you can SSH to that machine later (follow the same steps as in the **Task 6's HINT**). Also set the `associate_public_ip_address = false.`

```
resource "aws_instance" "ubuntu_public" {
  ami                         = aws_ami_copy.Ubuntu1804.id
  instance_type               = "t2.micro"
  subnet_id                   = aws_subnet.public.id
  vpc_security_group_ids      = [aws_security_group.allow_ssh.id,
aws_security_group.allow_tcp_8081.id,
aws_security_group.allow_all_outgoing.id]
  associate_public_ip_address = true
  key_name                    = "CYBERSECURITY_EC2_PUB"

  tags = {
    Name = "ubuntu_public"
  }
}

resource "aws_instance" "ubuntu_private" {
  ami                         = aws_ami_copy.Ubuntu1804.id
  instance_type               = "t2.micro"
  subnet_id                   = aws_subnet.private.id
  vpc_security_group_ids      = [aws_security_group.allow_ssh.id,
aws_security_group.allow_all_outgoing.id]
  associate_public_ip_address = false
  key_name                    = "CYBERSECURITY_EC2_PUB"

  tags = {
    Name = "ubuntu_private"
  }
}
```

Task 8 (11%): 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 actually ping www.google.com. Explain briefly why on the **EC2 private instance** you cannot ping google whereas on the **EC2 public instance you can**.

I used the Way 1:

```
ubuntu@ip-10-0-1-83:~$ ssh -o 'IdentitiesOnly yes' -i CYBERSECURITY_EC2_PUB.pem ubu
ntu@10.0.2.5
Welcome to Ubuntu 18.04.6 LTS (GNU/Linux 5.4.0-1084-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Wed Nov 16 02:28:46 UTC 2022

  System load:  0.0               Processes:           93
  Usage of /:   16.1% of 7.57GB   Users logged in:     0
  Memory usage: 18%               IP address for eth0: 10.0.2.5
  Swap usage:   0%

0 updates can be applied immediately.



The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-10-0-2-5:~$
```

```
ubuntu@ip-10-0-2-5:~$ ping google.com
PING google.com (142.251.163.113) 56(84) bytes of data.
^C
--- google.com ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4102ms

ubuntu@ip-10-0-2-5:~$ exit
logout
Connection to 10.0.2.5 closed.
ubuntu@ip-10-0-1-83:~$ ping google.com
PING google.com (142.250.31.101) 56(84) bytes of data.
64 bytes from bj-in-f101.1e100.net (142.250.31.101): icmp_seq=1 ttl=49 time=1.15 ms
64 bytes from bj-in-f101.1e100.net (142.250.31.101): icmp_seq=2 ttl=49 time=1.26 ms
^C
--- google.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 1.158/1.213/1.268/0.055 ms
ubuntu@ip-10-0-1-83:~$
```

The route table for the public subnet has a route that sends all internet-bound IPv4 traffic to the internet gateway. So the instances in the public which have public IP addresses or Elastic IP addresses can communicate with the Internet over the internet gateway.
For comparison, the route table of the private subnet does not have a route to the internet gateway. So instances in the private subnet can't communicate with the internet over the internet gateway, even if they have public IP addresses.

Task 9 (12%): What would you change in your "code" in order 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 added EIP & NAT gateway in gateway.tf and added NAT route table and private-subnet association in the route-tables.tf.

```
aws_eip.elastic_ip: Creating...
aws_eip.elastic_ip: Creation complete after 0s [id=eipalloc-01bce01ecebb88b82]
aws_nat_gateway.nat_gateway: Creating...
aws_nat_gateway.nat_gateway: Still creating... [10s elapsed]
aws_nat_gateway.nat_gateway: Still creating... [20s elapsed]
aws_nat_gateway.nat_gateway: Still creating... [30s elapsed]
aws_nat_gateway.nat_gateway: Still creating... [40s elapsed]
aws_nat_gateway.nat_gateway: Still creating... [50s elapsed]
aws_nat_gateway.nat_gateway: Still creating... [1m0s elapsed]
aws_nat_gateway.nat_gateway: Still creating... [1m10s elapsed]
aws_nat_gateway.nat_gateway: Still creating... [1m20s elapsed]
aws_nat_gateway.nat_gateway: Still creating... [1m30s elapsed]
aws_nat_gateway.nat_gateway: Still creating... [1m40s elapsed]
aws_nat_gateway.nat_gateway: Creation complete after 1m46s [id=nat-0a9994a1b3eed215c]
aws_route_table.NAT_route_table: Creating...
aws_route_table.NAT_route_table: Creation complete after 2s [id=rtb-009b8fbdf4080fee9]
aws_route_table_association.associate_routetable_to_private_subnet: Creating...
aws_route_table_association.associate_routetable_to_private_subnet: Creation complete after 0s [id=rtbassoc-0a3bbe66b59daa
dcc]

Apply complete! Resources: 4 added, 0 changed, 0 destroyed.
PS D:\Course\22FALL-CMPT732\terraform>
```

```
# elastic ip
resource "aws_eip" "elastic_ip" {
  vpc      = true
}

# NAT gateway
resource "aws_nat_gateway" "nat_gateway" {
  depends_on = [
    aws_subnet.public,
    aws_eip.elastic_ip,
  ]
  allocation_id = aws_eip.elastic_ip.id
  subnet_id     = aws_subnet.public.id

  tags = {
    Name = "nat-gateway"
  }
}

# route table with target as NAT gateway
resource "aws_route_table" "NAT_route_table" {
  depends_on = [
    aws_vpc.main,
    aws_nat_gateway.nat_gateway,
  ]

  vpc_id = aws_vpc.main.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_nat_gateway.nat_gateway.id
  }
```

```
  tags = {
    Name = "NAT-route-table"
  }
}

# associate route table to private subnet
resource "aws_route_table_association"
"associate_routetable_to_private_subnet" {
  depends_on = [
    aws_subnet.private,
    aws_route_table.NAT_route_table,
  ]
  subnet_id      = aws_subnet.private.id
  route_table_id = aws_route_table.NAT_route_table.id
}
```

Ping:

```
ubuntu@ip-10-0-2-5:~$ ping google.com
PING google.com (142.250.65.78) 56(84) bytes of data.
64 bytes from iad23s91-in-f14.1e100.net (142.250.65.78): icmp_seq=1 ttl=104 time=2.13 ms
64 bytes from iad23s91-in-f14.1e100.net (142.250.65.78): icmp_seq=2 ttl=104 time=1.47 ms
^C
--- google.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 1.478/1.806/2.135/0.331 ms
```