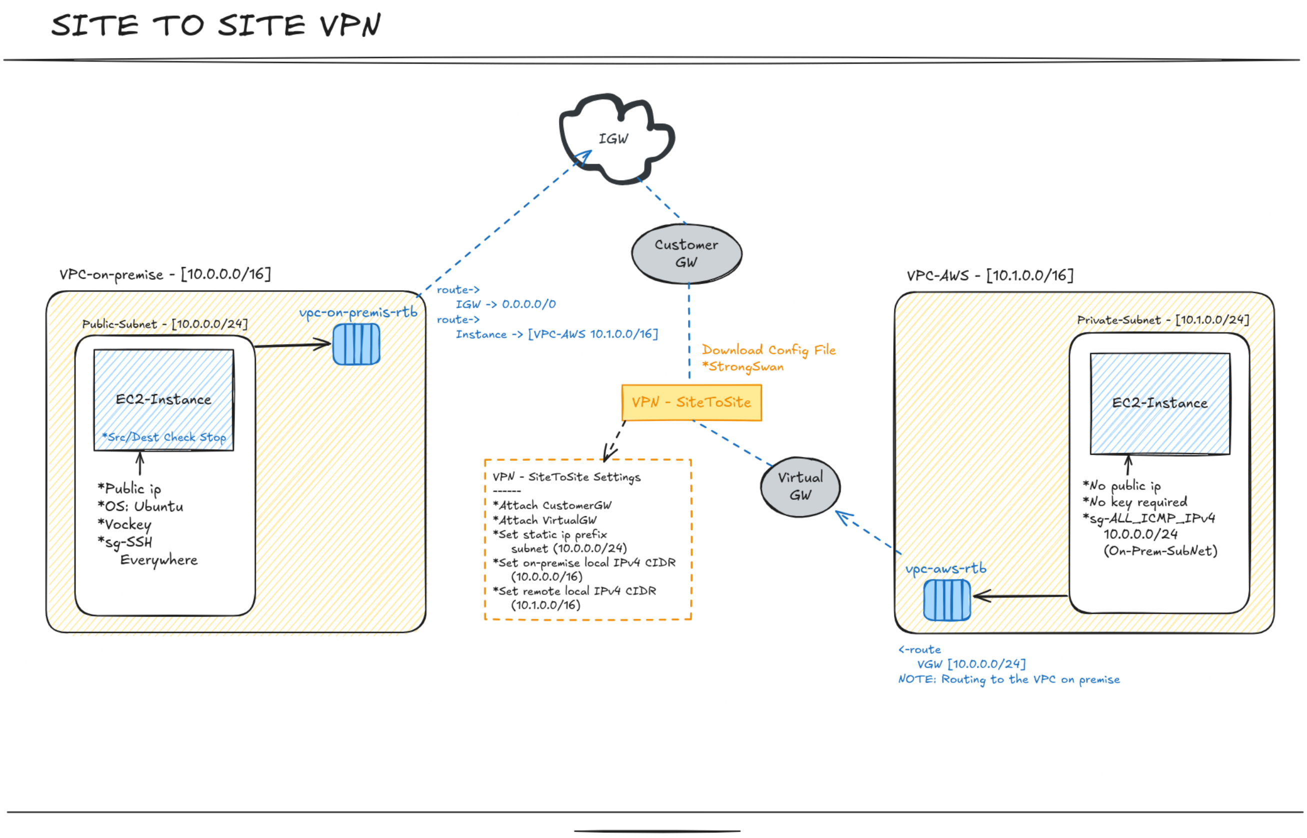
Omar BaGunaid

In this document, I will thoroughly go through the steps to implement a Site-to-Site Virtual Private Network (VPN) using an on-premises network simulated as a separate Virtual Private Cloud (VPC). And the other will be the VPC we connect to via the VPN.

Requirements

Topology



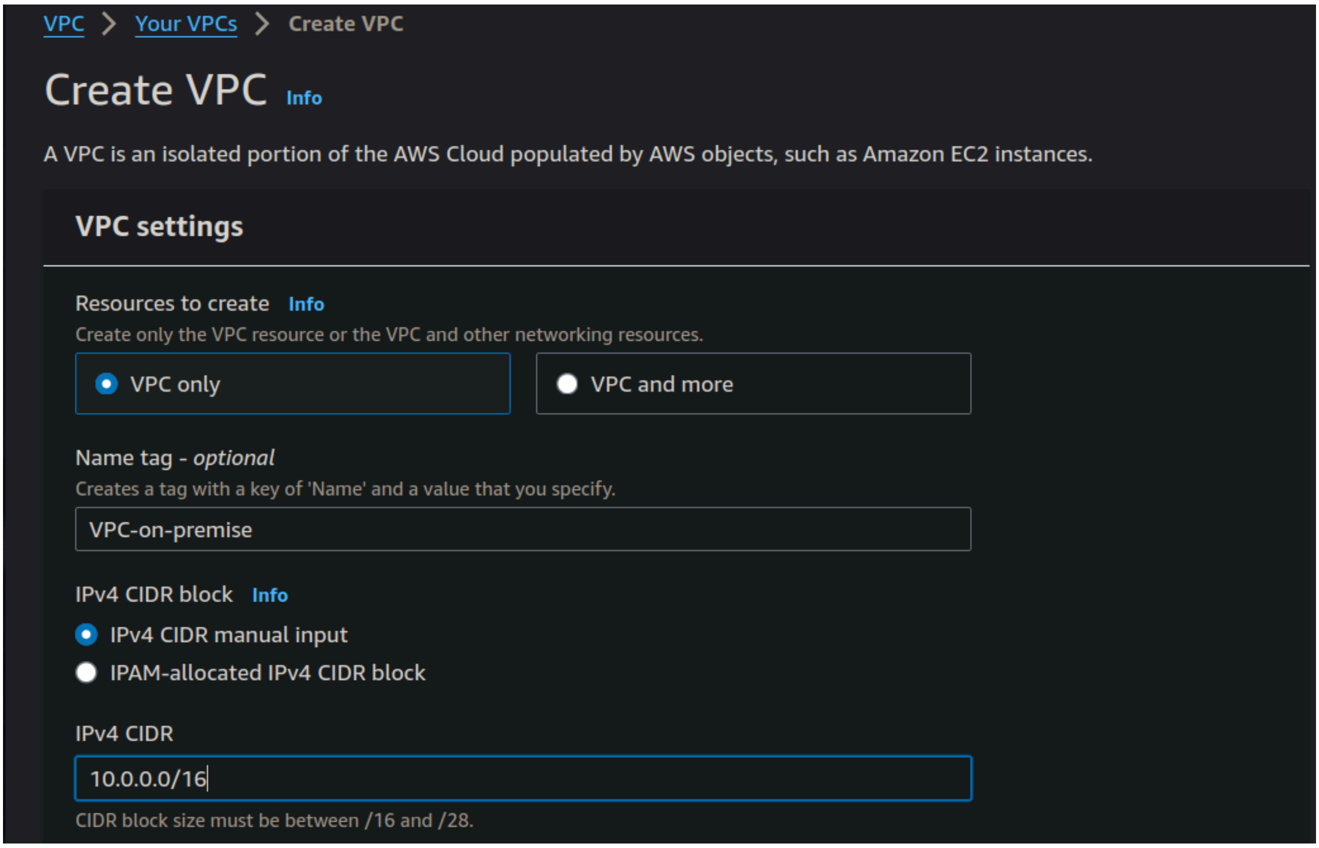
Resources

1. vpc-on-premise (10.0.0.0/16)
    1. public-subnet (10.0.0.0/24)
        1. vpc-on-premise-rtb
            1. To Internet Gateway (0.0.0.0/0)
            2. To Instance of the other side instance 10.1.0.0/16
        2. ec2-instance
            1. OS: Ubuntu
            2. Public IP
            3. VOCKEY

            4. SG-SSH - Everywhere.
    2. Internet Gateway
2. aws-vpc (10.1.0.0/16)
    1. private-subnet (10.1.0.0/24)
        1. vpc-aws-rtb
            1. To Virtual Gateway
        2. ec2-instance
            1. No public ip
            2. No key
            3. OS: Amazon Linux (default)
            4. SG-all icmp ipv4 from 10.0.0.0/24 (on-premise-public-subnet)
3. Customer Gateway (will use the public ip address of the on-premise-instance)
4. Internet Gateway (will be used in the on-premise-public-subnet)
5. Virtual Gateway (will be used on the aws-vpc private-subnet routing table)
6. Site to Site VPN Connection
    1. *Attach CustomerGW
    2. Attach VirtualGW
    3. Set static ip prefix subnet (10.0.0.0/24)
    4. Set on-premise local IPv4 CIDR (10.0.0.0/16)
    5. Set remote local IPv4 CIDR (10.1.0.0/16)
    6. In the static routes add the vpc-on-premise CIDR Block (10.0.0.0/16) or the public-subnet of the on-premise (10.0.0.0/24).

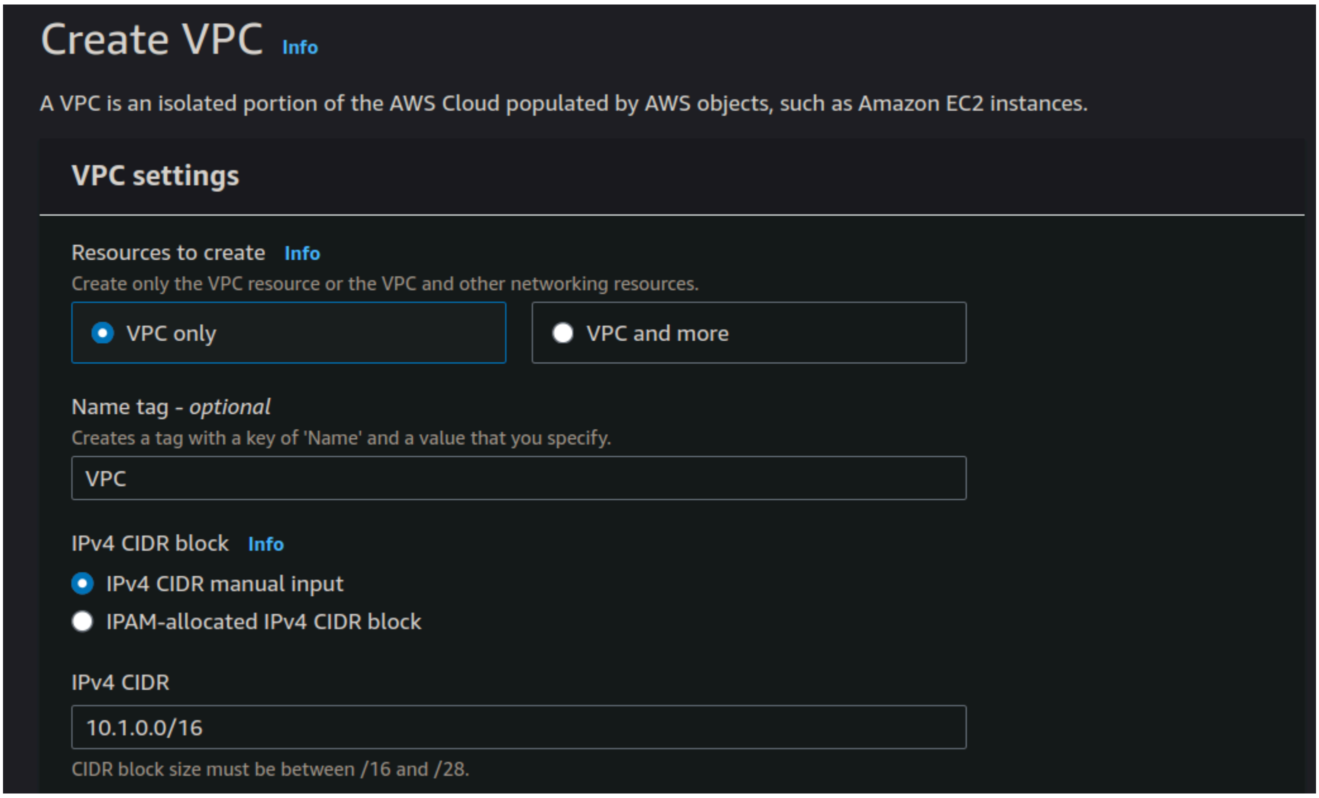
Lets Create VPCs and subnets

VPC-on-premise

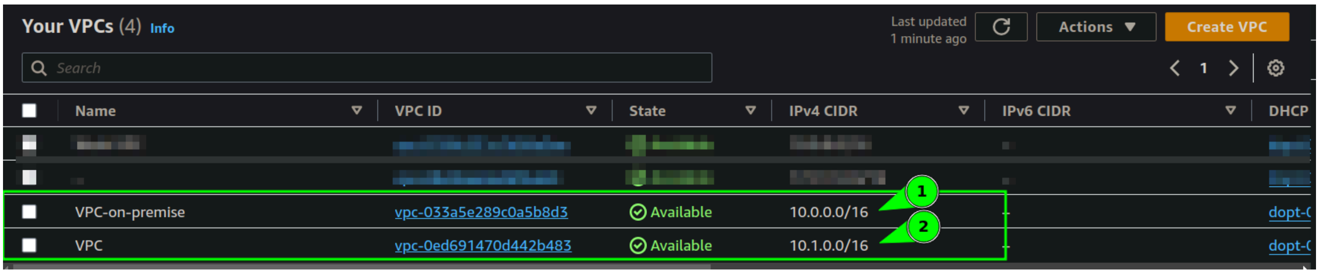


Setting the IPv4 CIDR at 10.0.0.0/16

Another VPC to be connected to from the VPC-on-premis



Setting the IPv4 CIDR at 10.1.0.0/16

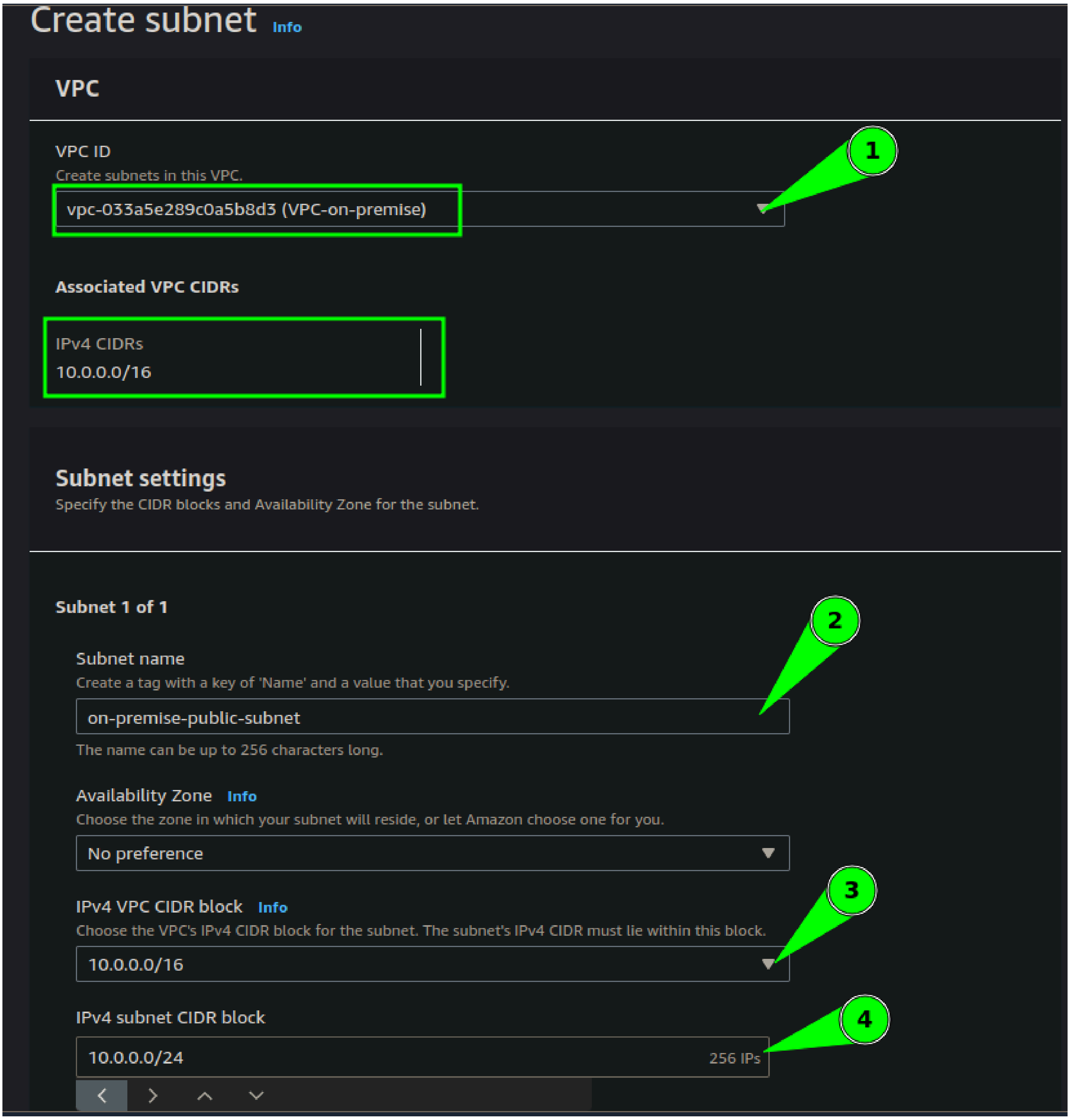


Lets create the subnets

On-Premise Public Subnet

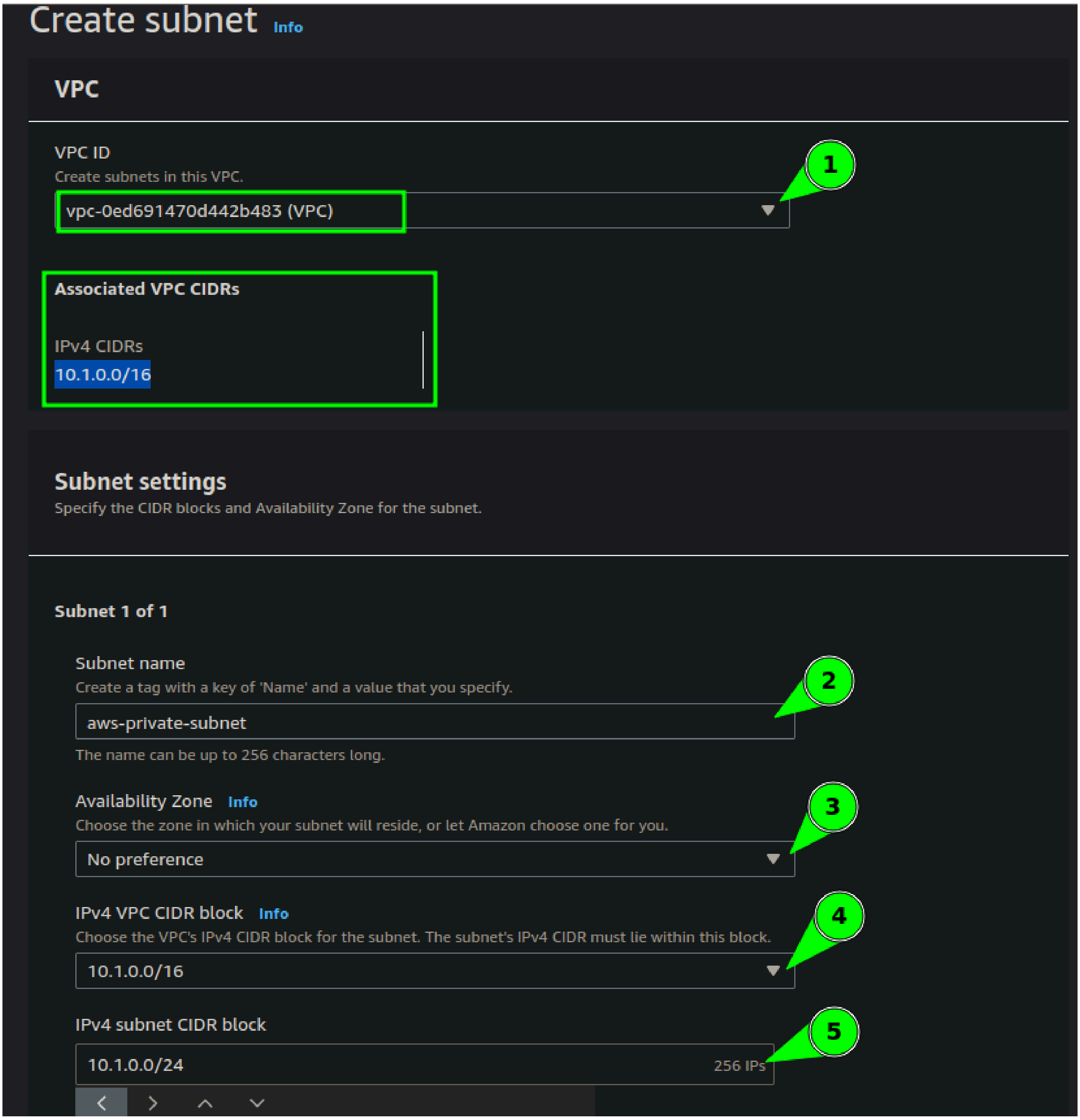
First subnet will be created for the VPC-on-premise. Its going to be named `on-premise-public-subnet`. It needs a routing table and also needs an Internet Gateway (IGW). The on premise subnet will use the VPC-on-premise at CIDR block `10.0.0.0/16` and this subnet, will use `10.0.0.0/24` of the vpc-on-premise CIDR block, this will give the subnet 256 hosts.

I will be using only one host, for the EC2 instance to connect to the VPN.



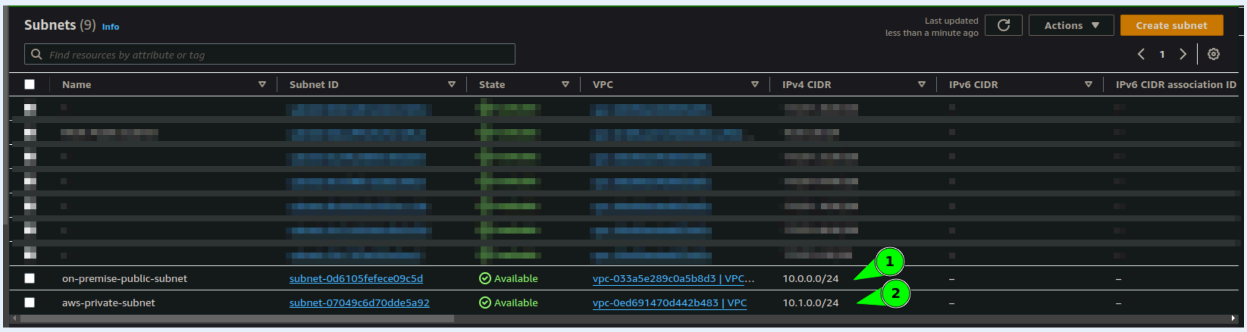
AWS VPC Private Subnet

Secondly, creating a subnet for the other VPC, `vpc`. For this one, I will name it `aws-private-subnet`. It needs a routing table as well. This subnet will use the VPC ip block `10.1.0.0/16` and this subnet, will use `10.1.0.0/24` from the given VPC block, this will give the subnet 256 hosts.



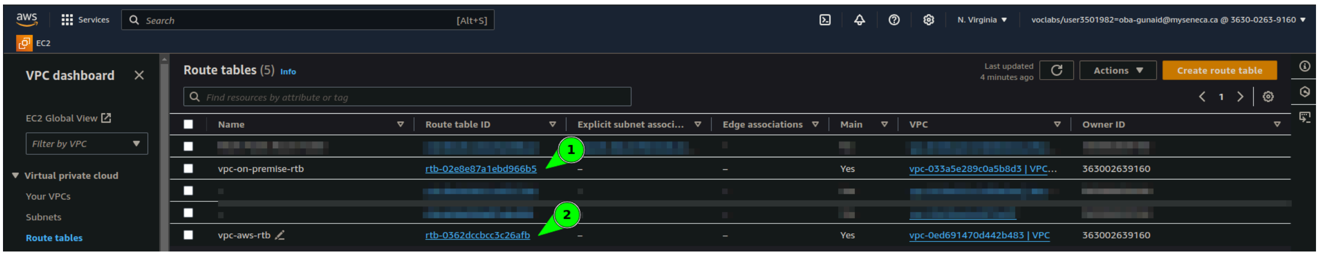
Total Subnets

Total of required subnets for this Site To Site VPN. Only two subnets, one will be used for the on-premise machine, and the other one will be used on other side, will be used to connect to the instances within the subnet using the tunnels by the Site To Site VPN.



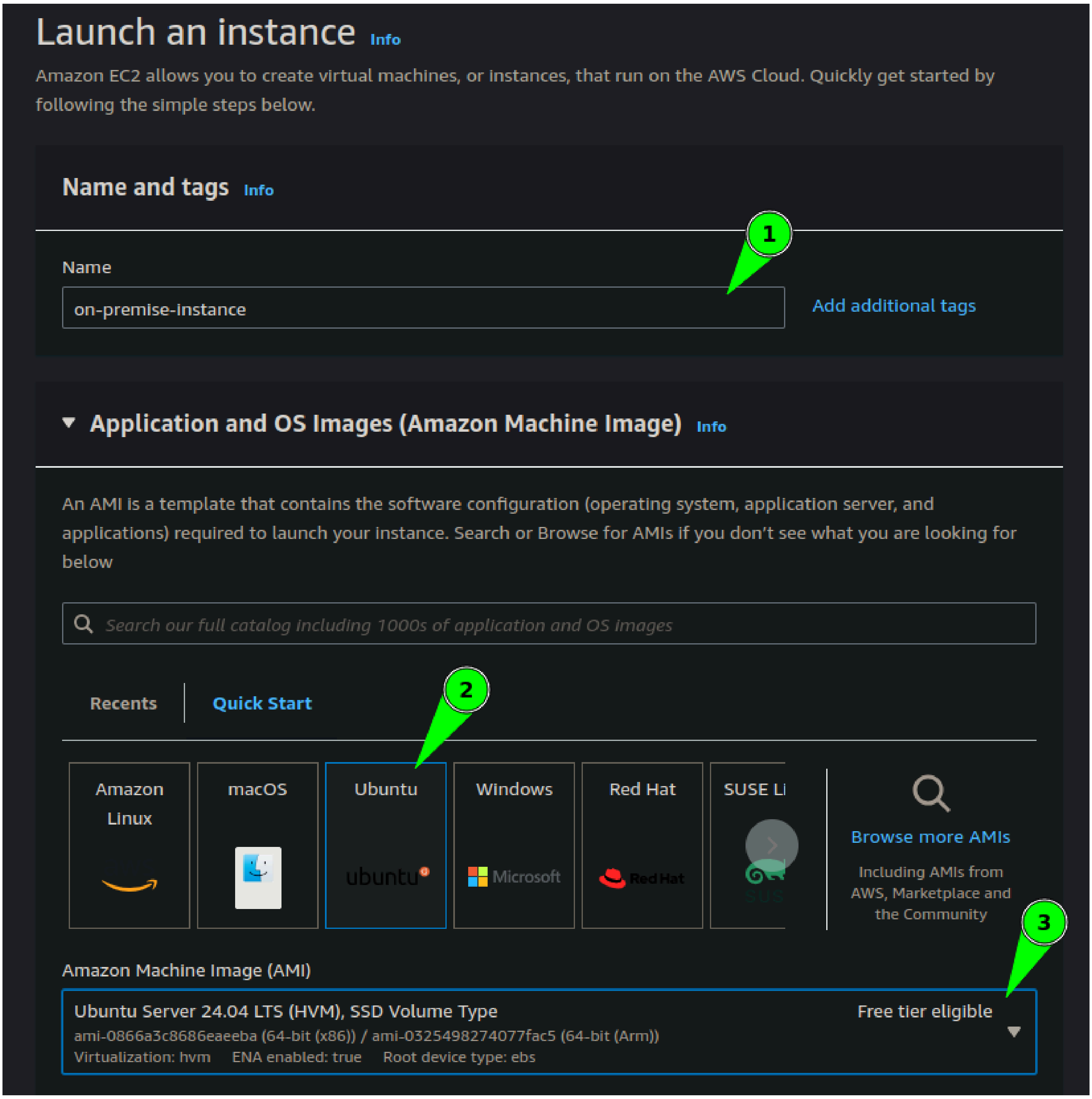
Lets edit or create routing table.

I will be using the routing tables that were created while creating the subnets. You can edit them and renamed them if you want or create new ones, thats totally up to you. I will go through editing their routings later.

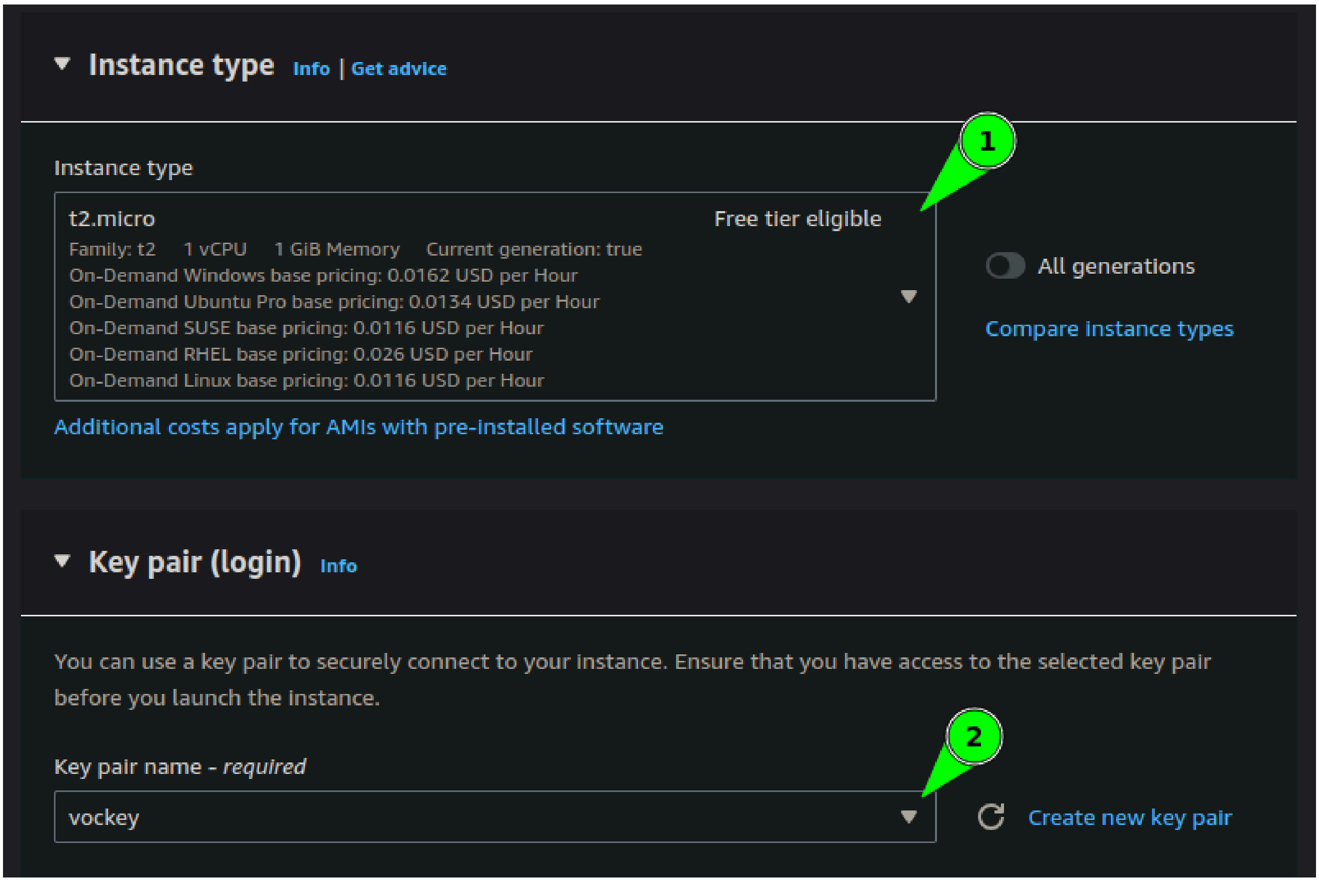


Lets create an EC2 instance for vpc-on-premise

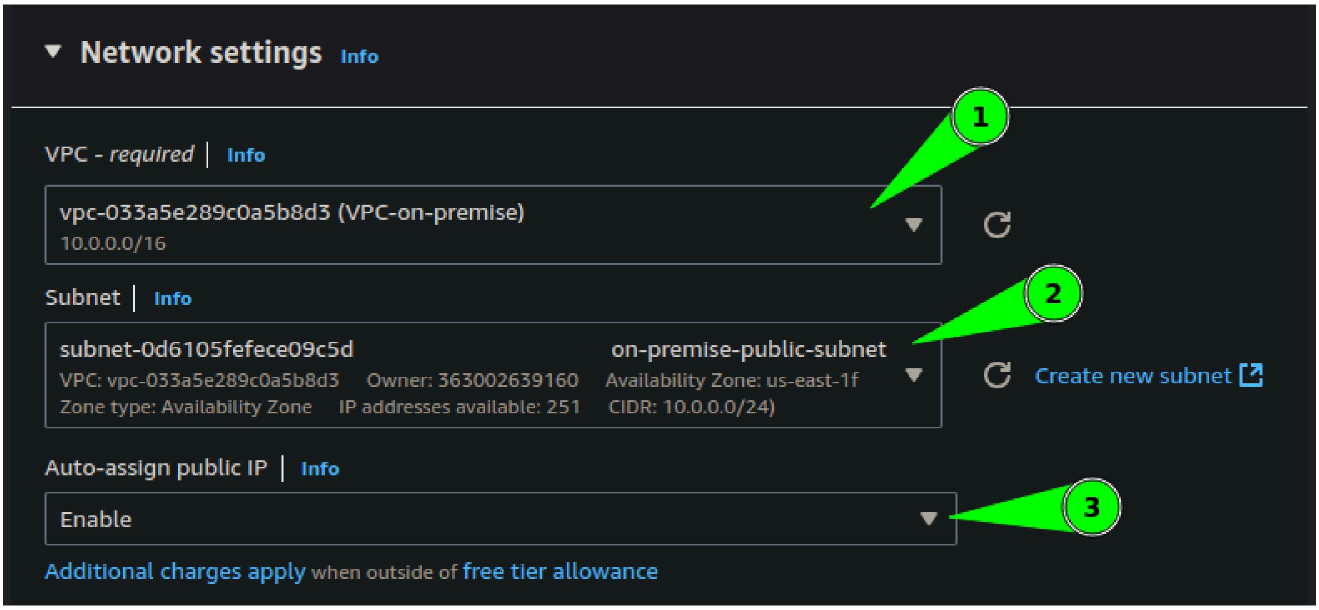
Its required since we need its public ip address for the *Customer Gateway* later. As well, I will be going through editing the routing table of the `on-premise-public-subnet`.



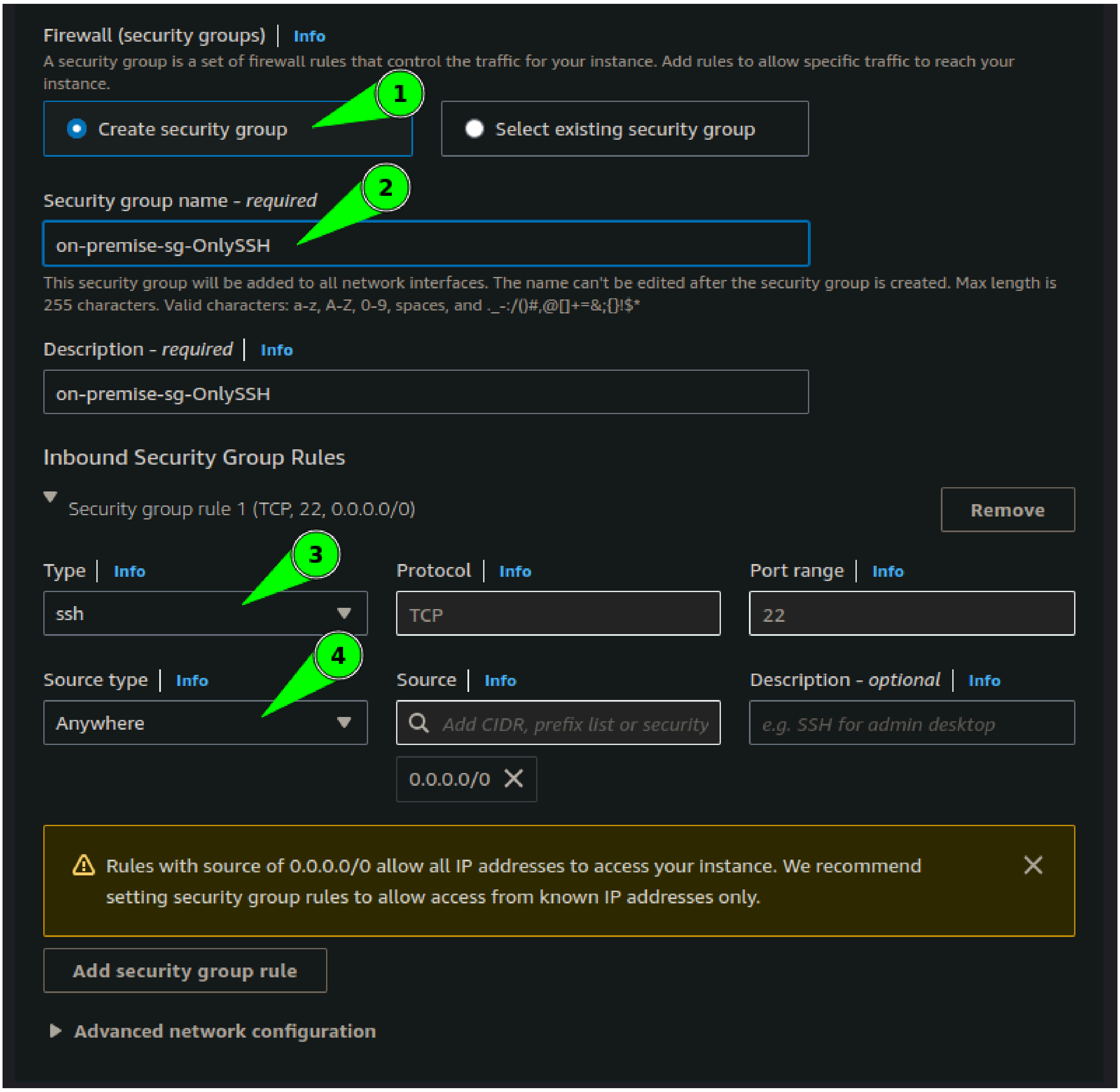
I will name this instance `on-premise-instance` and use Ubuntu, as it includes pre-built packages. I will also use **strongSwan** to establish the connection to the Site-to-Site VPN.



I am going to be using the **vockey** provided for me. You can create a pair, don't forget to download the private key, we will be needing that to login to the instance through SSH.



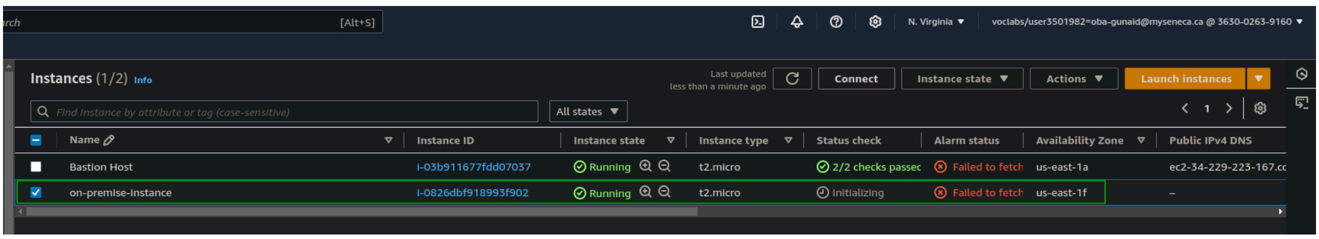
These are the network settings. Make sure to assign a public ip address for this instance.



Make sure to add SSH security group, we will want to connect to it later.

NOTE

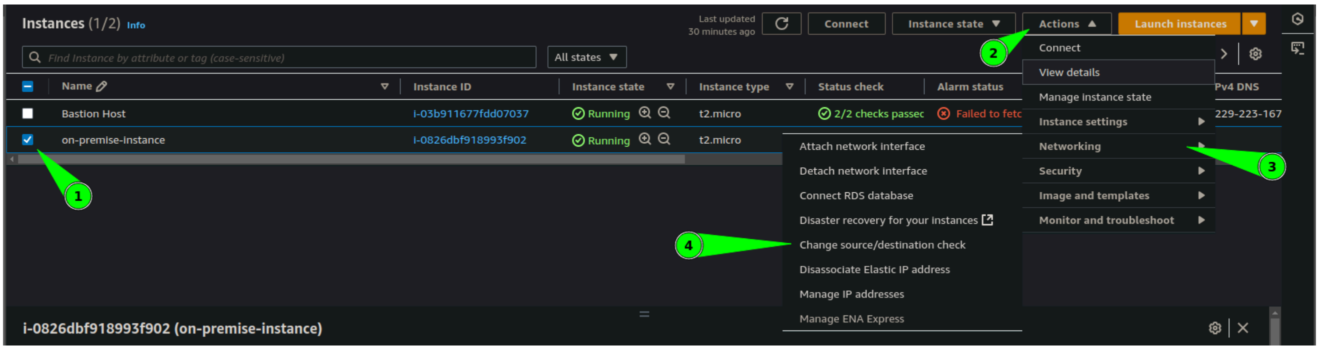
You can assign your public IP address to limit access to that instance for secure login. To do this, make sure to select *My IP* from *Source type*. This ensures that only your home public IP address is allowed to connect to the instance.

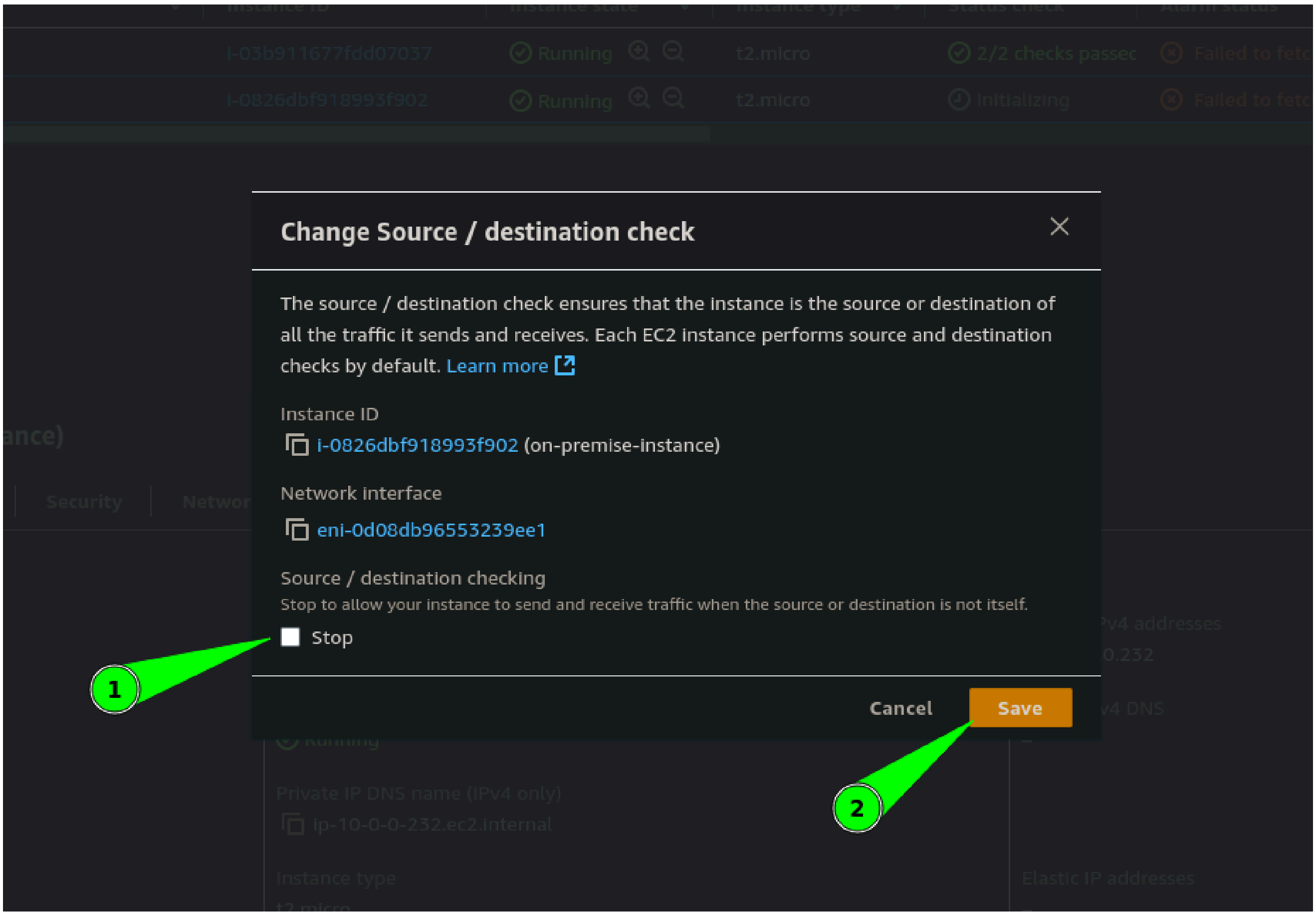


Denote public ip

> Copy the public ip address of the created instance, we will be needing that later for the *Customer Gateway* as well as to login to that instance.

Also, we will need to stop Source and Destination check from the instance.



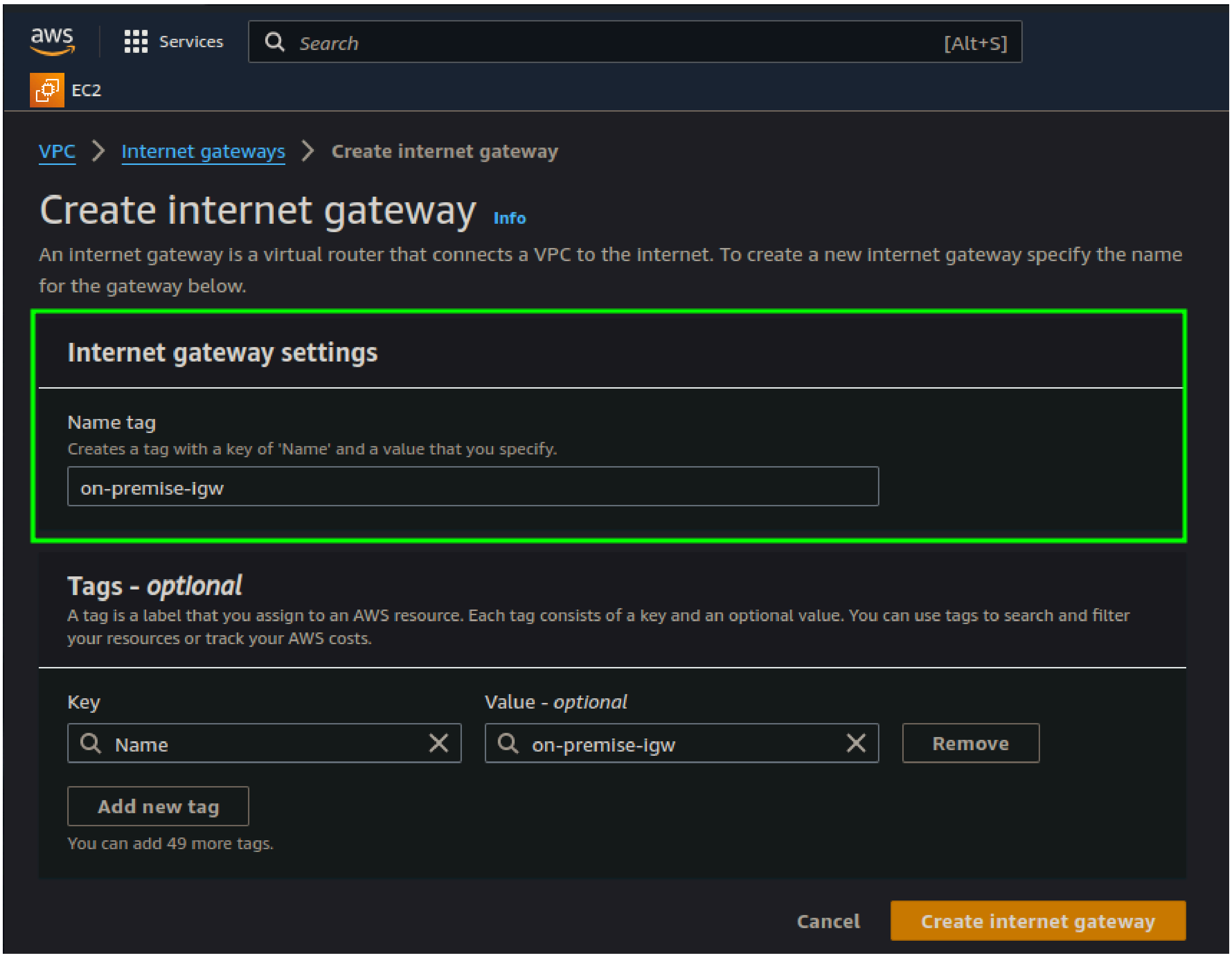


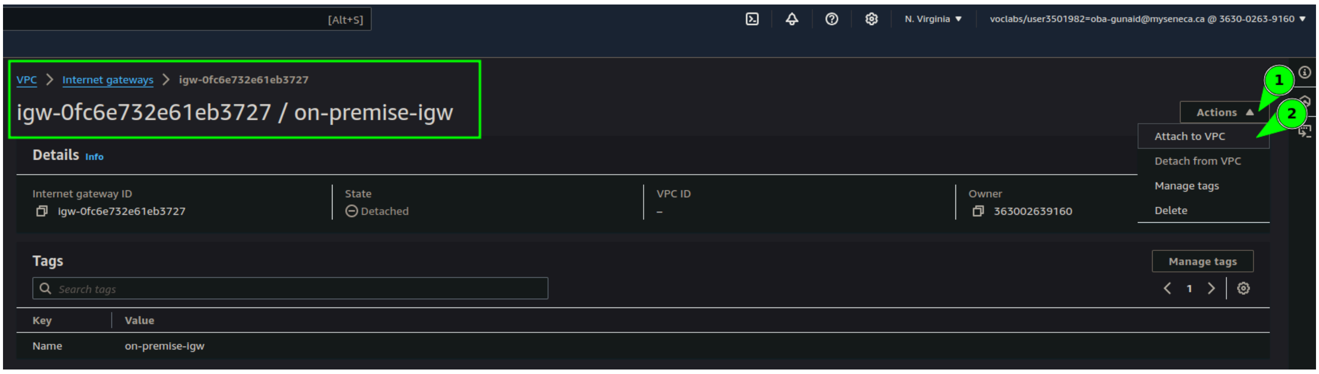
Lets create the required gateways

Internet Gateway

You will need to navigate to the **Internet Gateway** section and create a new Internet Gateway. I will name this `on-premise-igw`. Remember, we are only creating one Internet Gateway to be used for the `vpc-on-premises` VPC.

We will later add this to the routing table of the on-premise-vpc

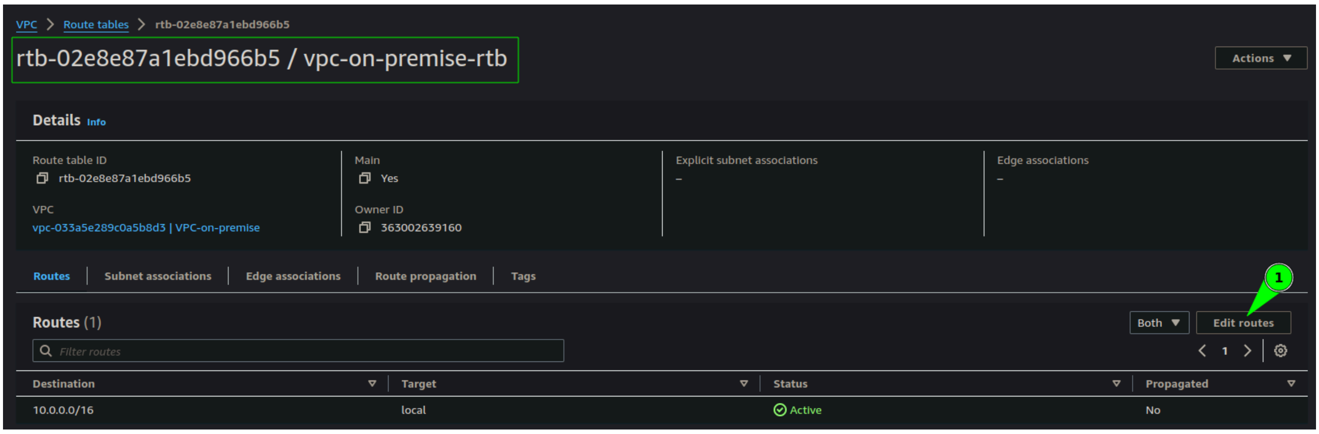


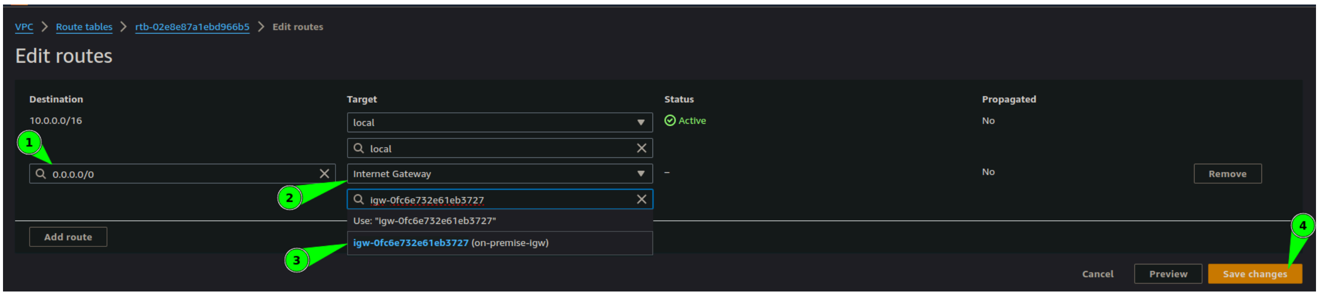


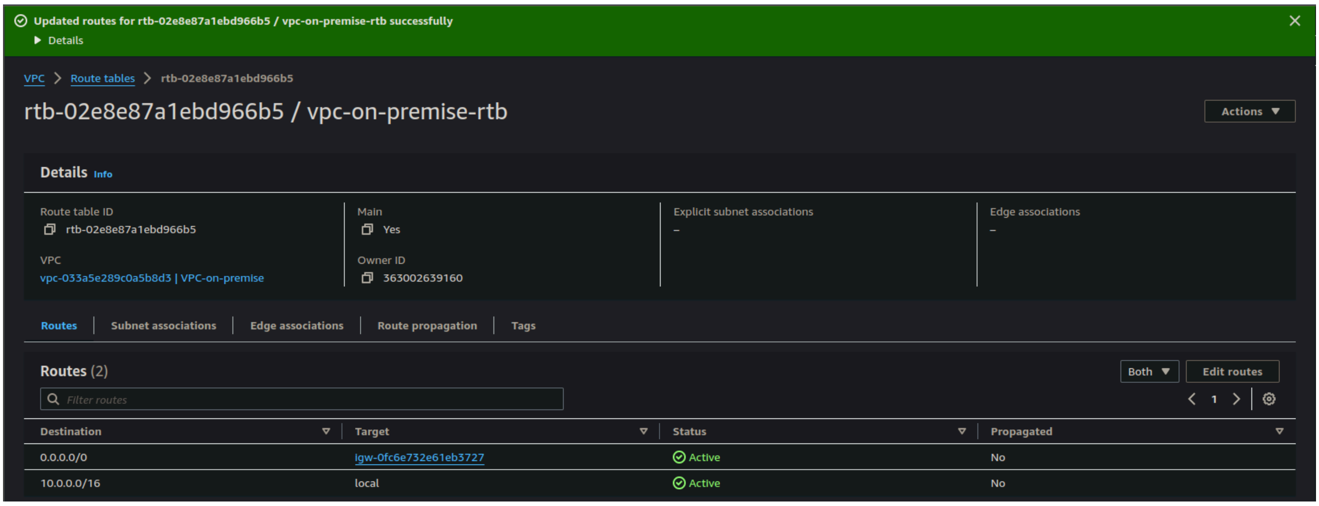
Don't forget to attach it to the `vpc-on-premise`

on-premise-public-subnet - Internet Access

Before we move on, lets take a look at the routing table of *on-premise-public-subnet*, we will need to add this **Internet Gateway** to it. This will ensure that all instances that are in this subnet, will have internet access.

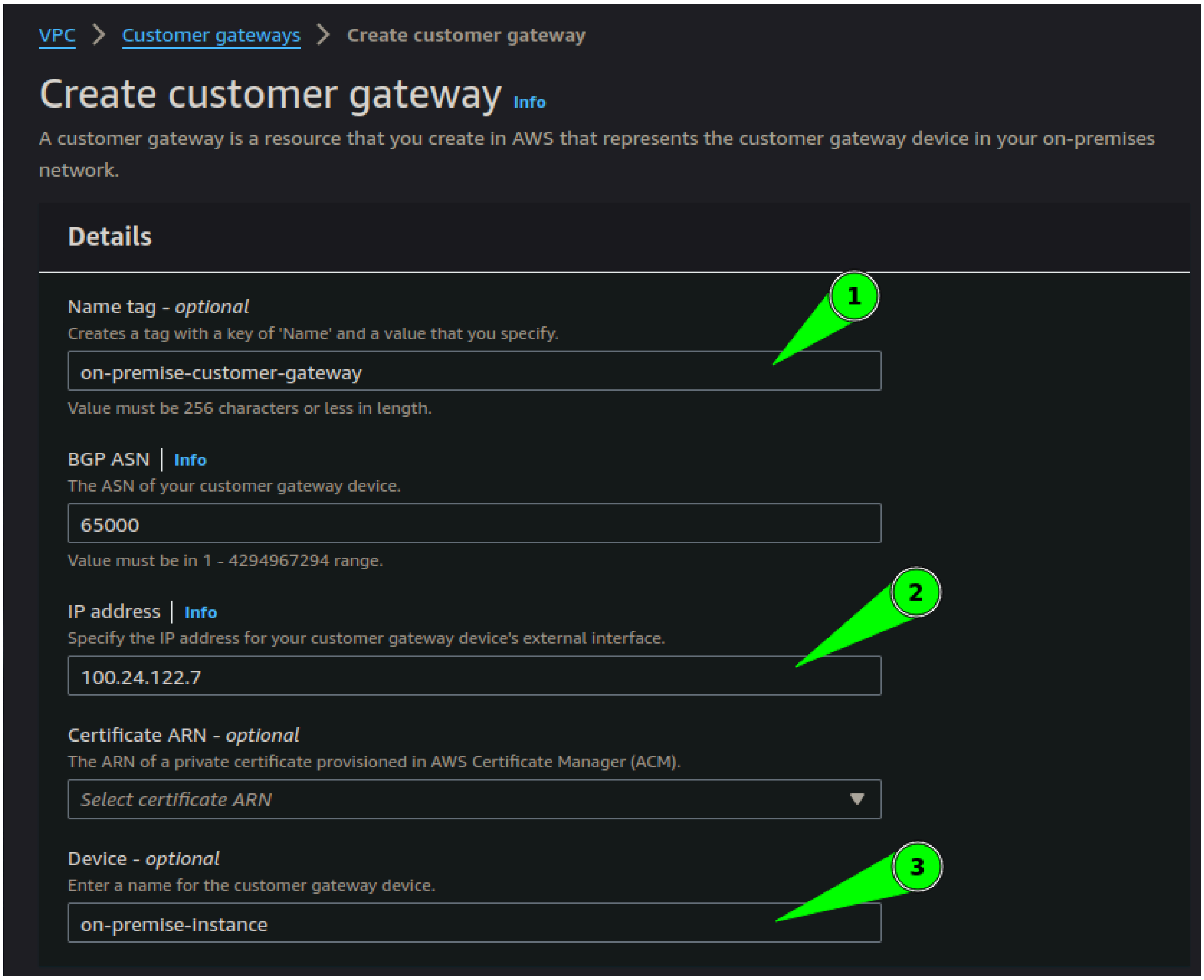




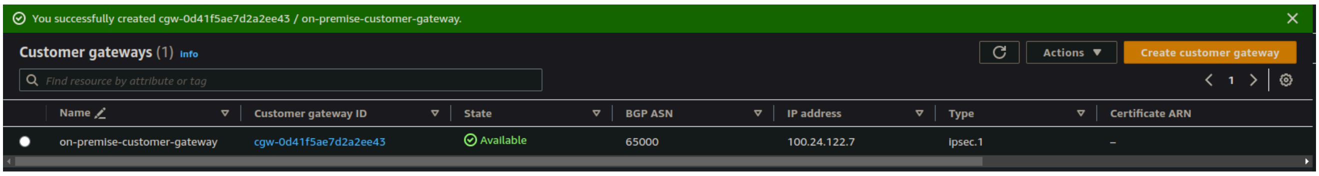


Customer Gateway

Under *Virtual Private Network* VPN section, navigate to *Customer Gateway*. Create a new one.



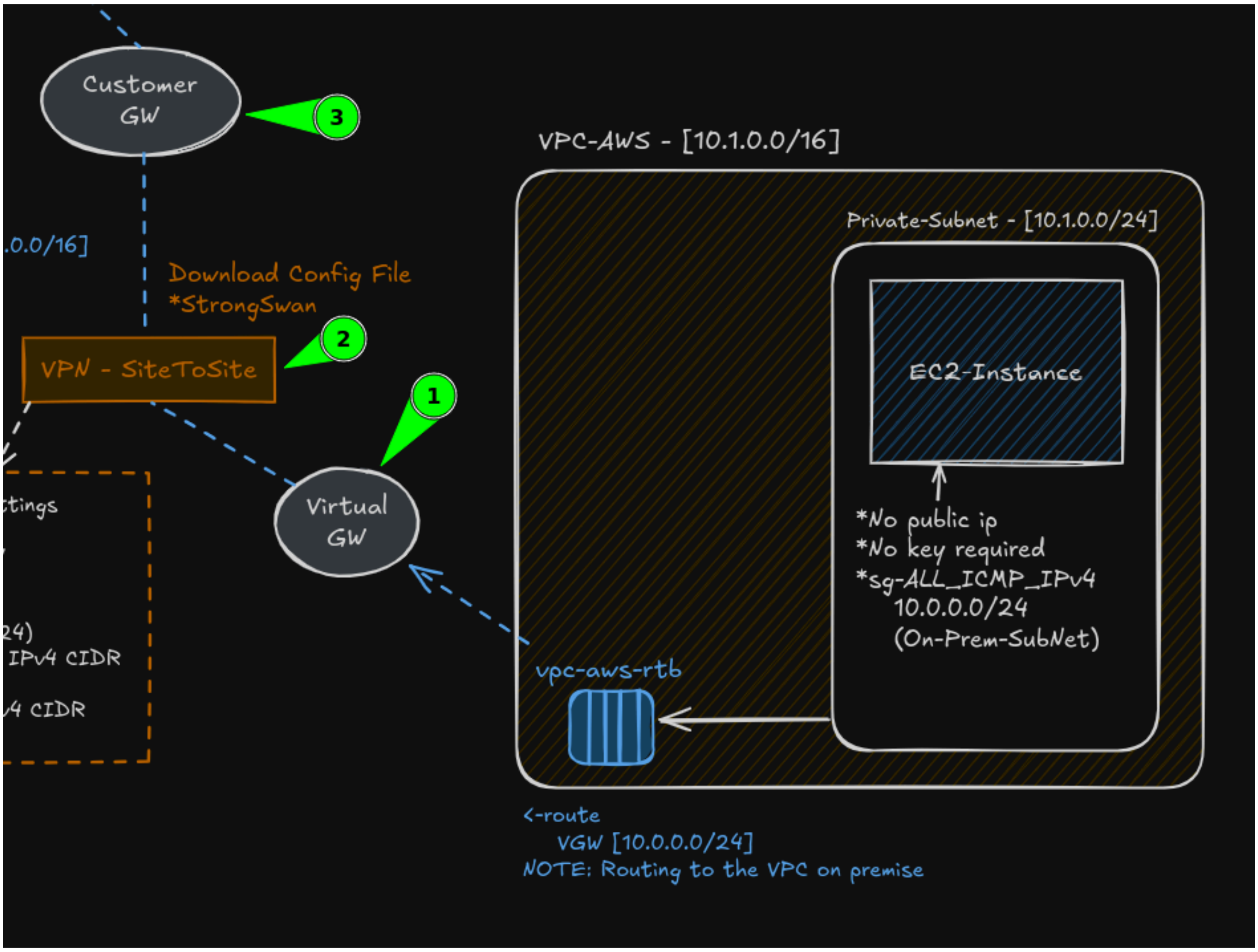
I used the public IP address of the `on-premise-instance` I created earlier. This instance will serve as the on-premises site.

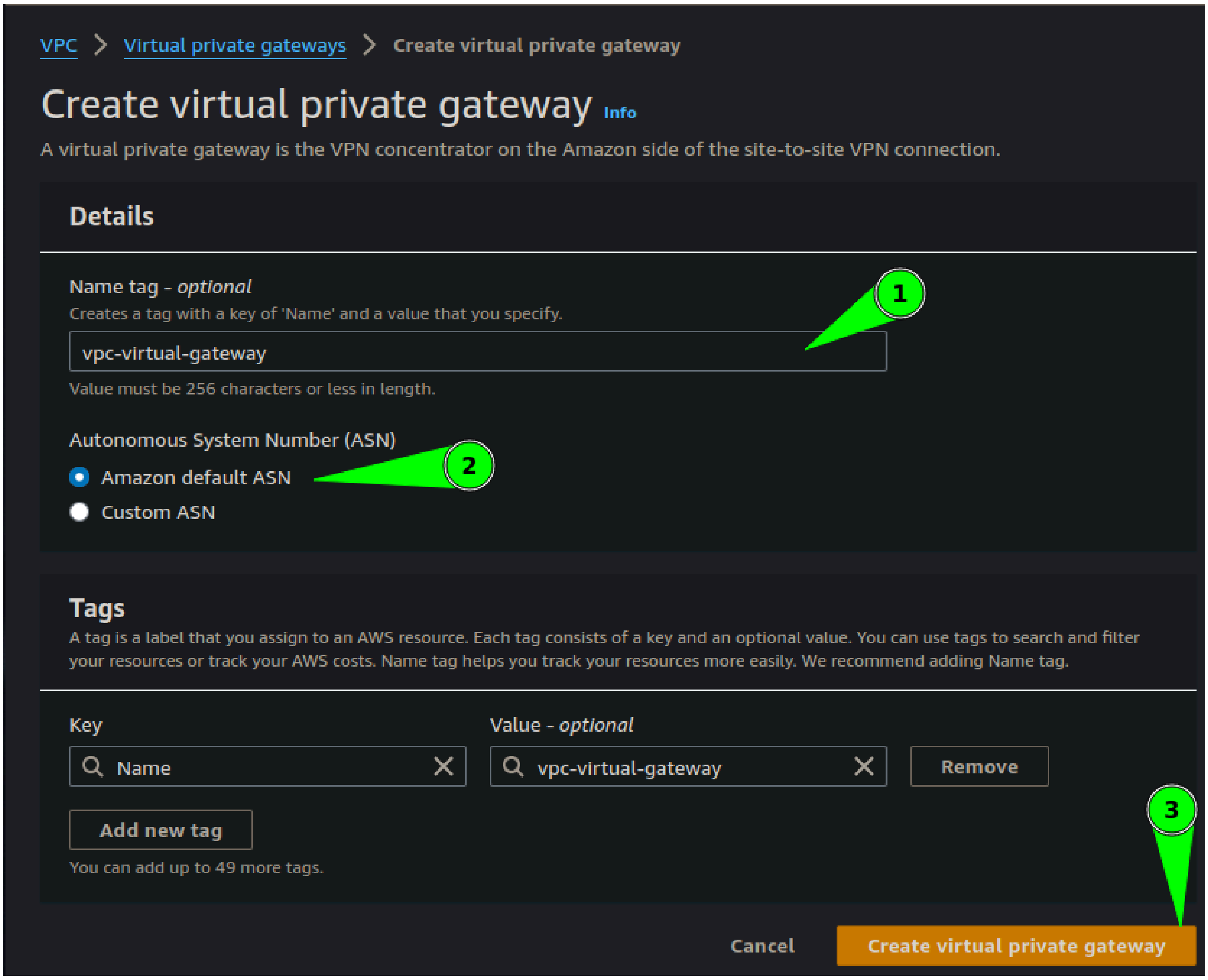


Virtual Private Gateway

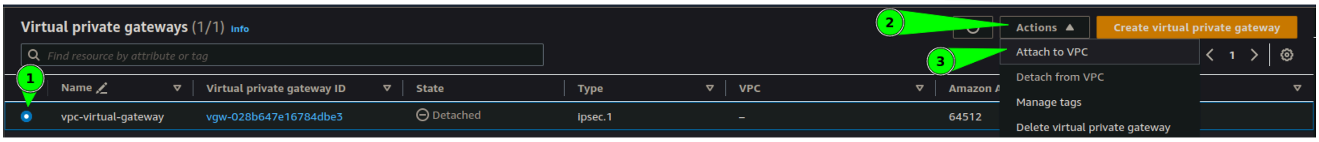
The virtual gateway is required to establish a Site-to-Site VPN connection. This virtual gateway will be used by the Site-to-Site VPN and the `vpc`.

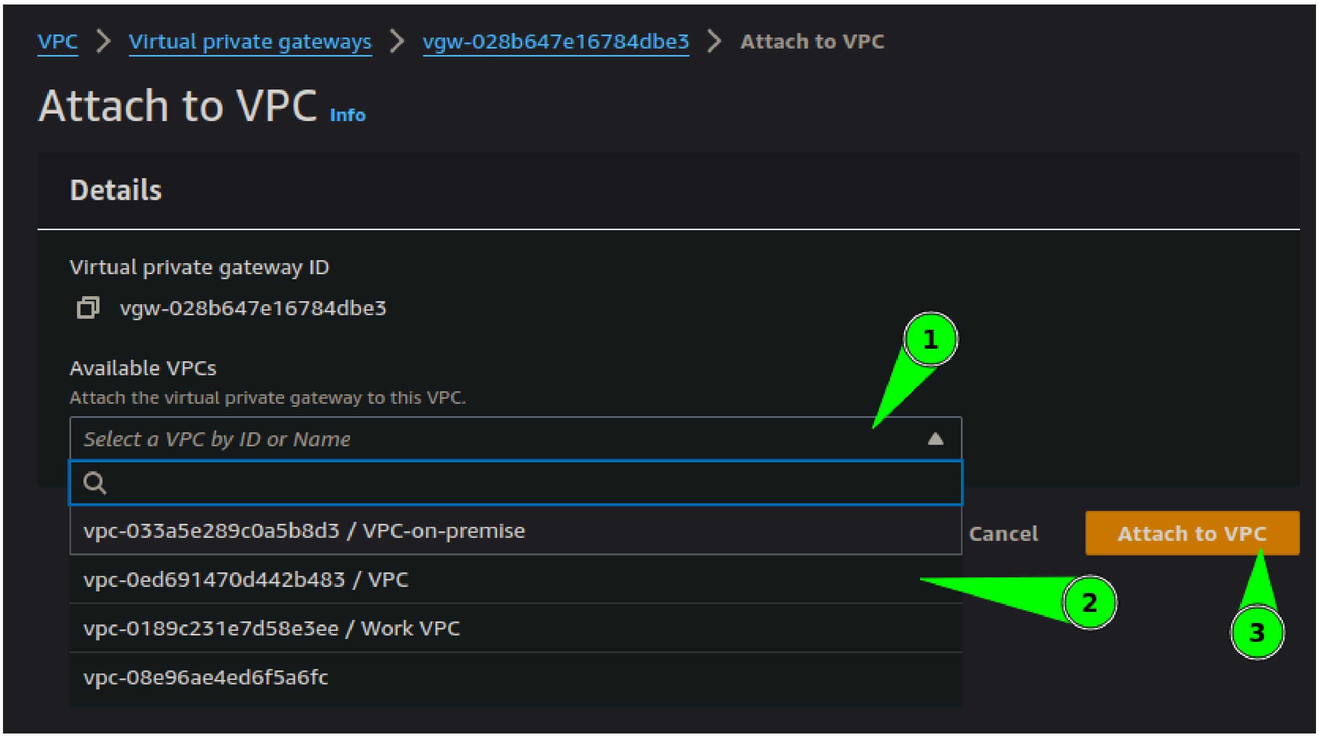
NOTE: *Look at the topology.*

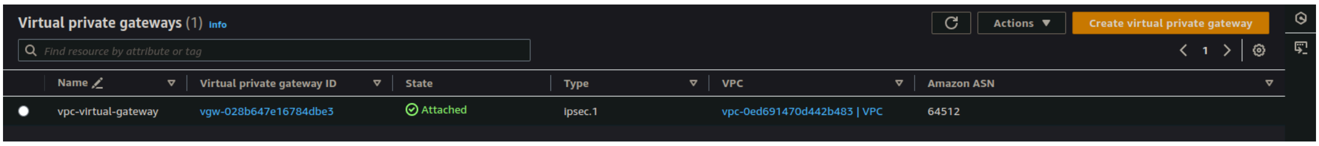




This *Virtual Private Gateway* needs to be attached to the *aws-vpc* `VPC` .



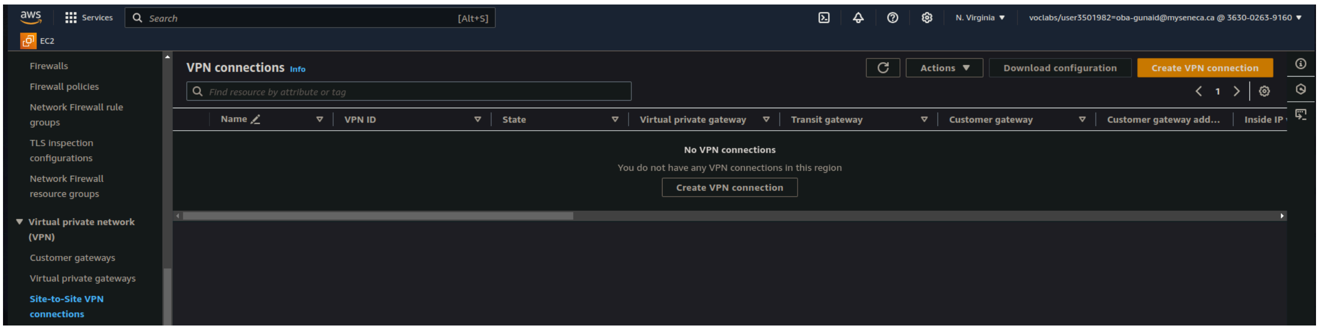




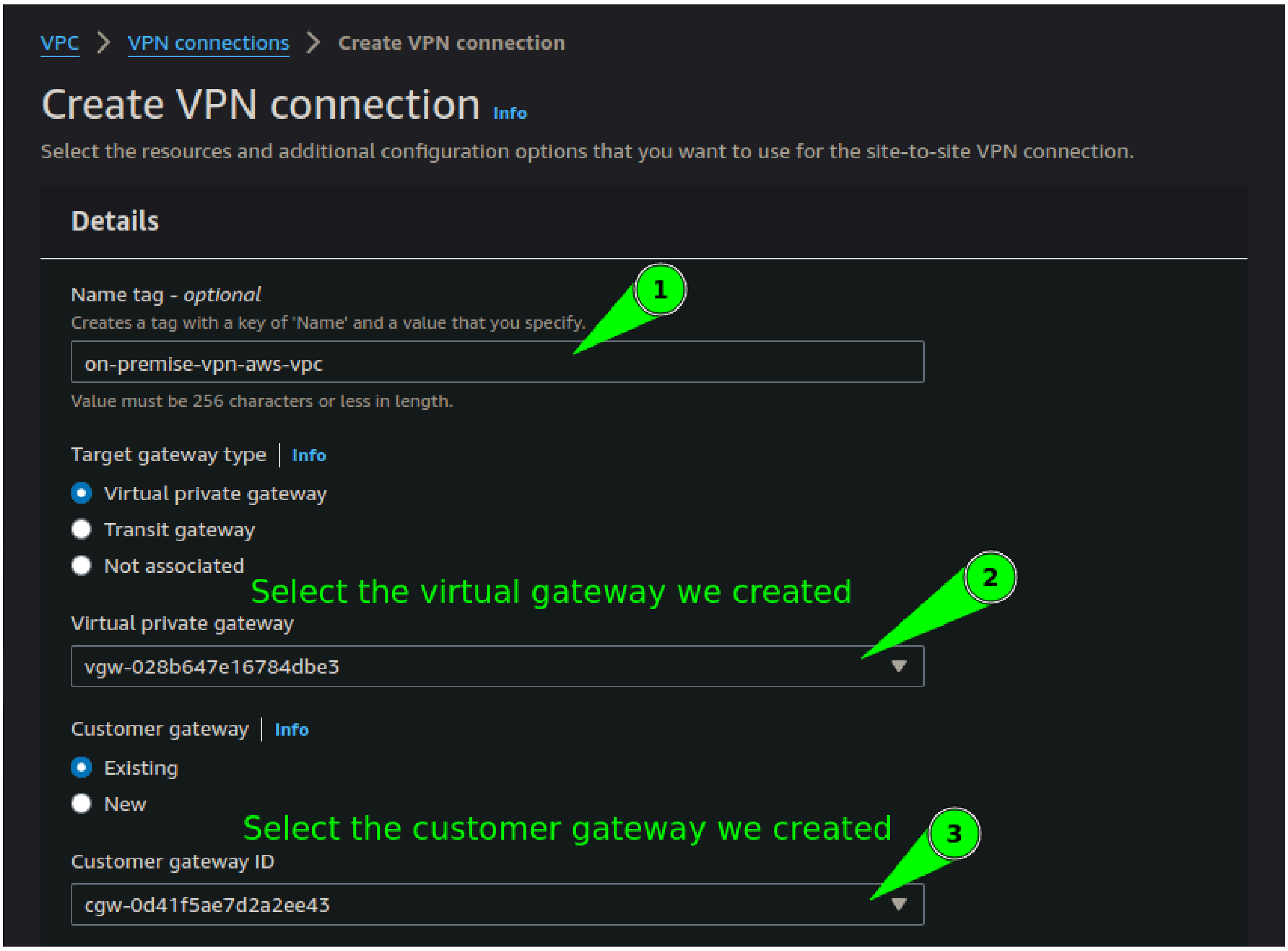
Now the VPC attached to the *Virtual Private Gateway*.

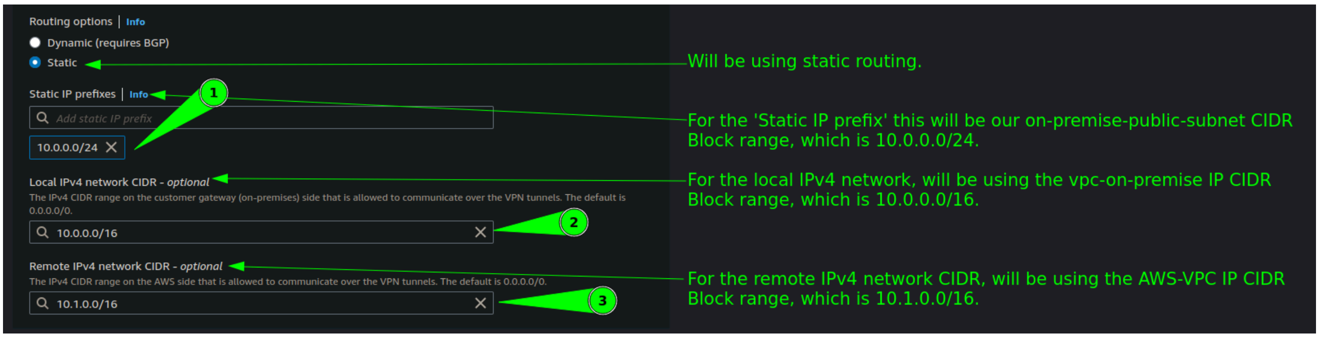
> **We are attaching the `VPC` not the `vpc-on-premise`**

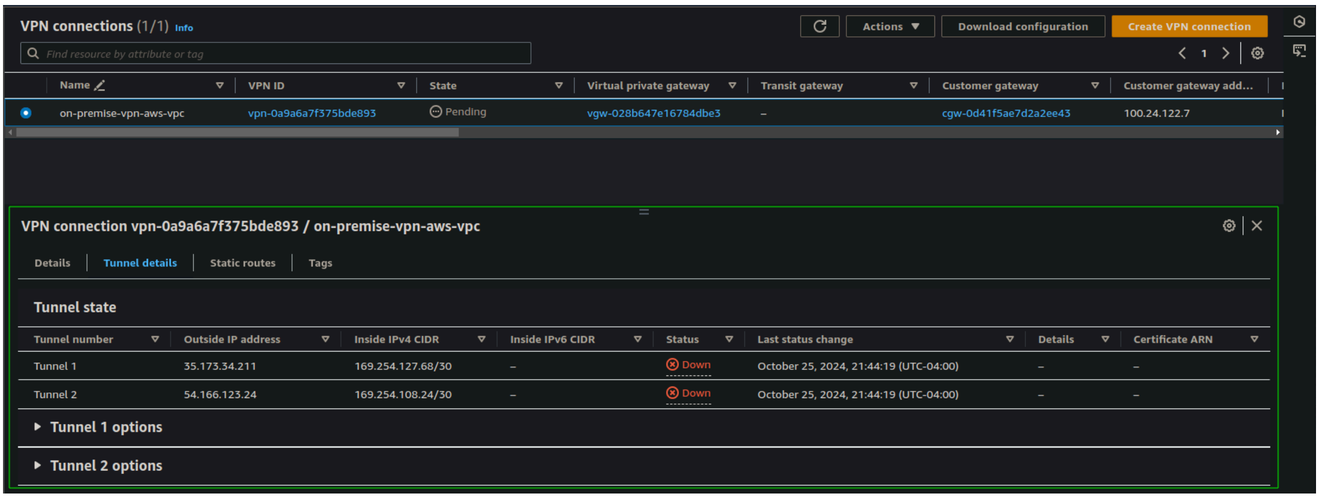
Site To Site VPN Connection



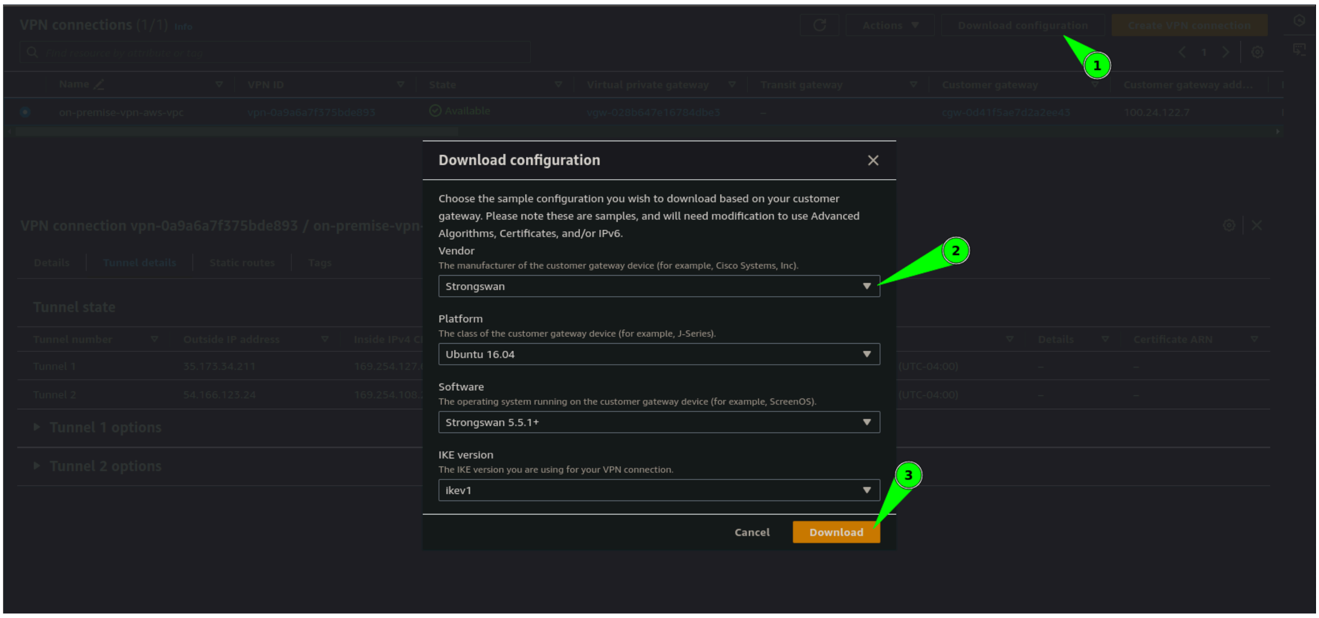
Now we will create a site to site vpn connection.





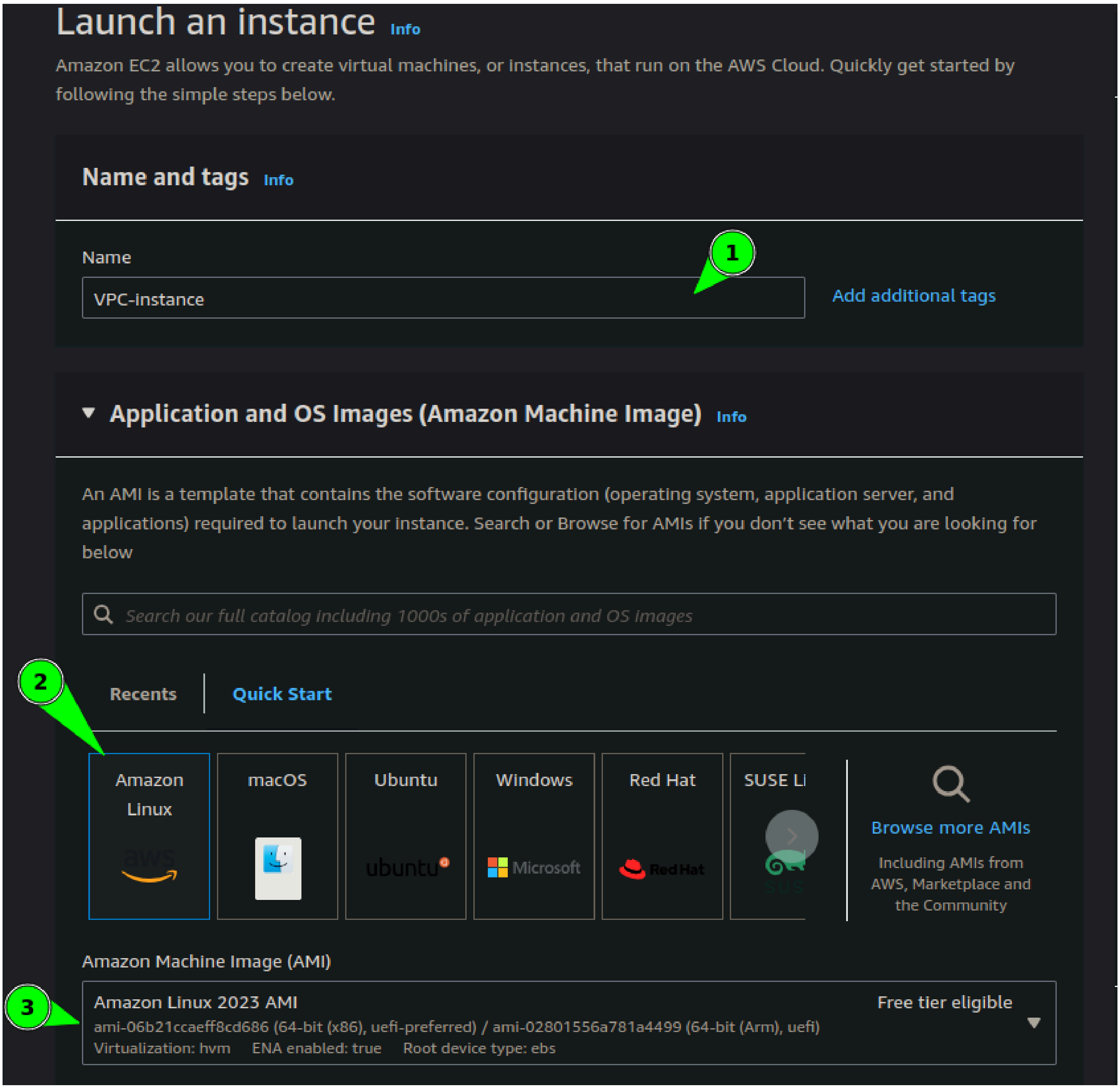


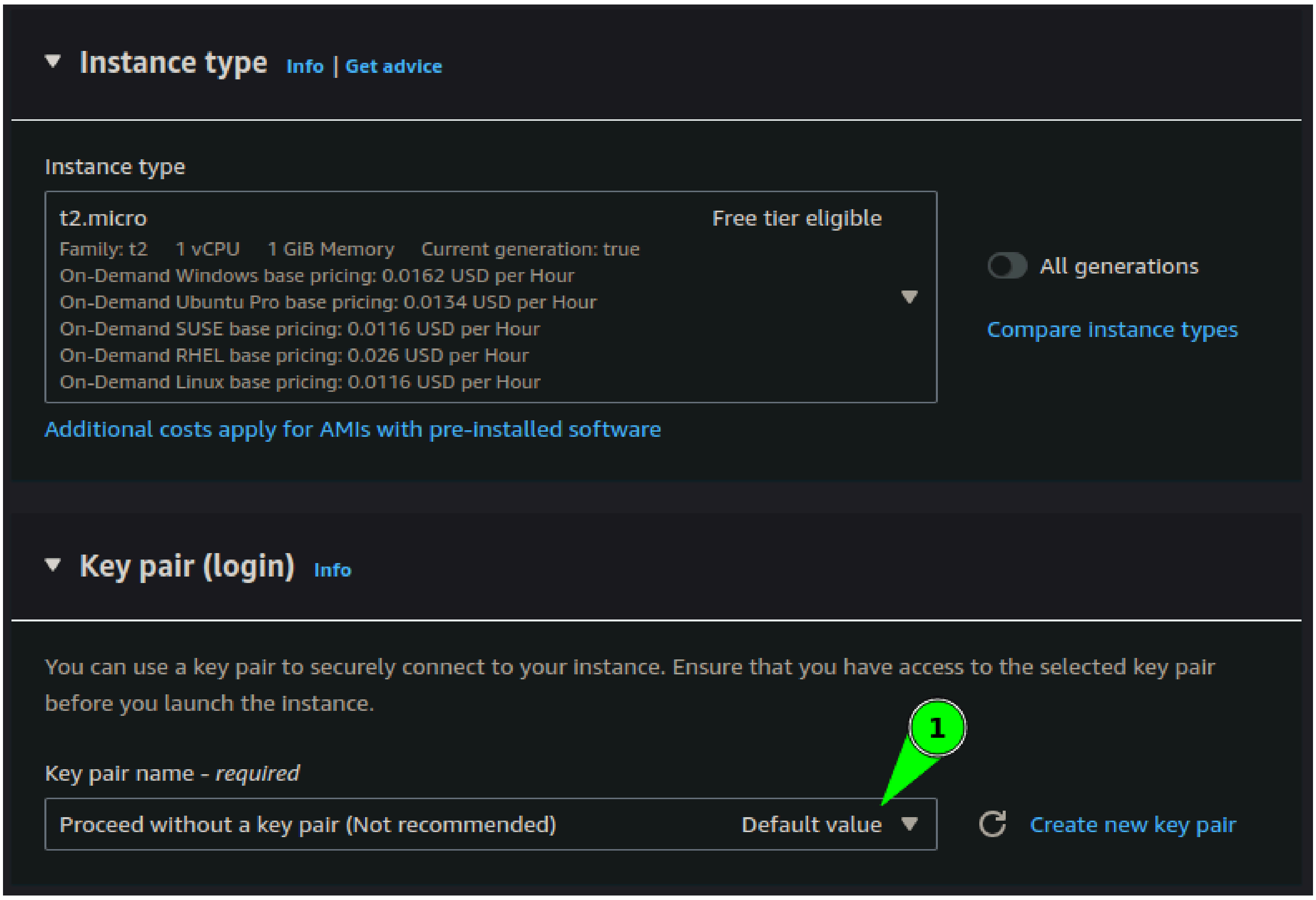
As you can see the tunnels are down. Lets download the configuration file for now. I will be using *strongSwan*.

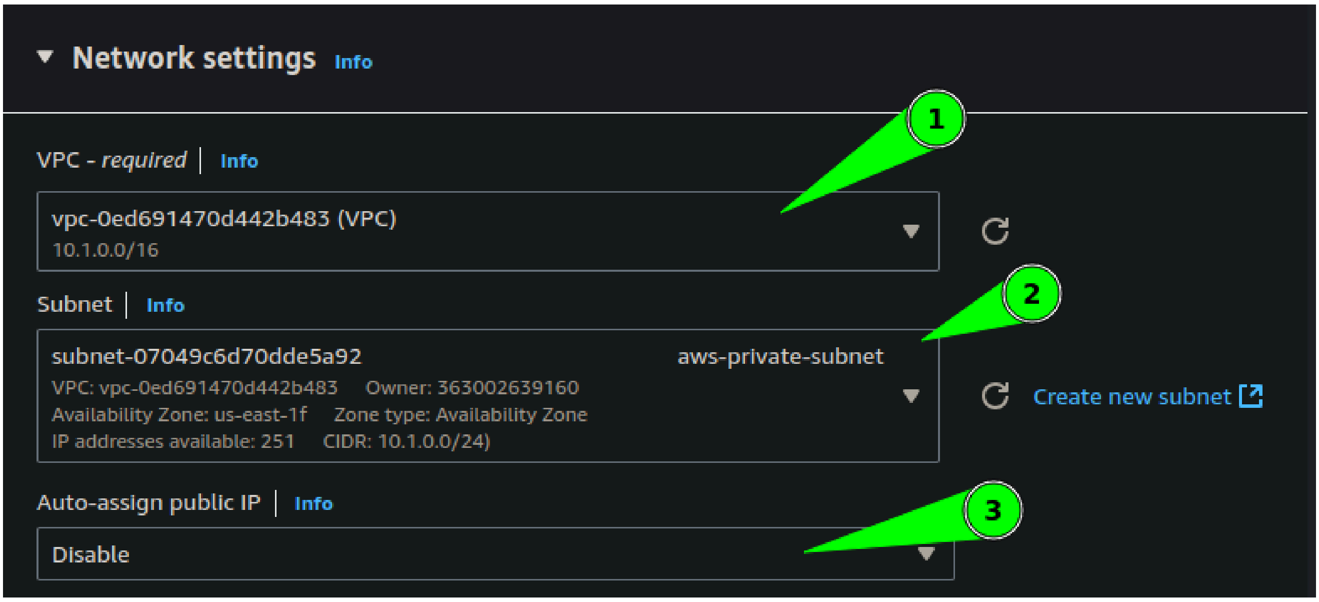


Launch an EC2 instance in `VPC` the non-premise VPC

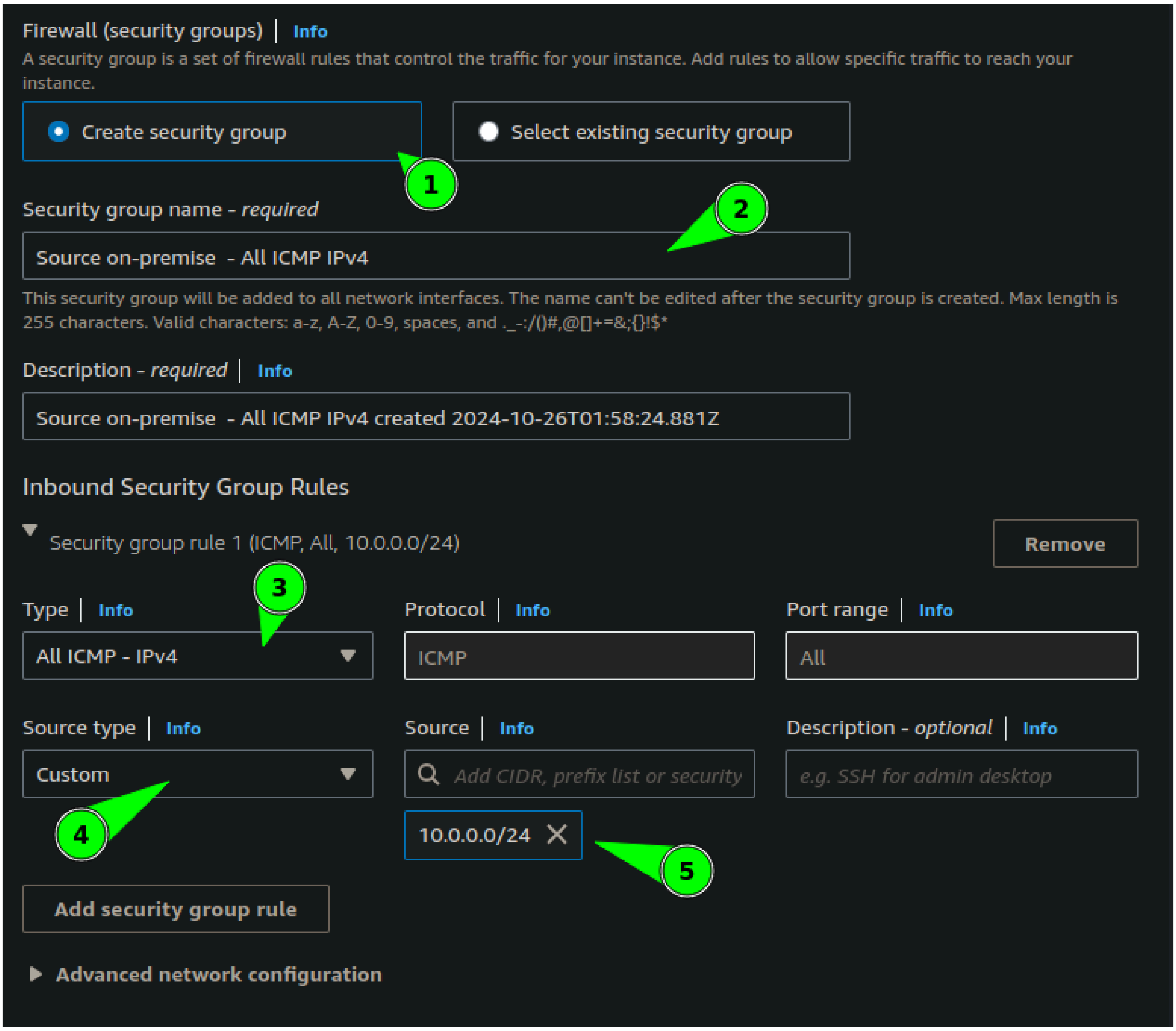
This instance won't have a public ip address, and no key is required. Just in the security group, we will need to allow `Allow All ICMP IPv4` from `10.0.0.0/24` which is the `on-premise-public-subnet`.


Launch an instance Info
Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.
Name and tags Info
Name
VPC-instance
1
Add additional tags
Application and OS Images (Amazon Machine Image) Info
An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below
Search our full catalog including 1000s of application and OS images
2
Recents
Quick Start
Amazon Linux
macOS
Ubuntu
Windows
Red Hat
SUSE Li
Browse more AMIs
Including AMIs from AWS, Marketplace and the Community
Amazon Machine Image (AMI)
3
Amazon Linux 2023 AMI
Free tier eligible
ami-06b21ccaeff8cd686 (64-bit (x86), uefi-preferred) / ami-02801556a781a4499 (64-bit (Arm), uefi)
Virtualization: hvm ENA enabled: true Root device type: ebs





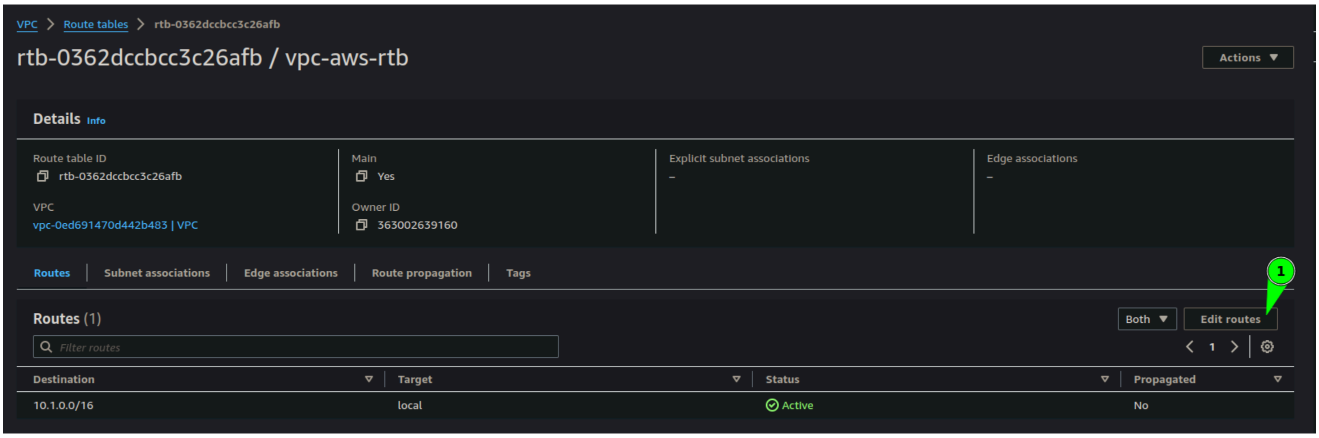
Now will need to ensure this instance accepts traffic from a specific IP CIDR block. *10.0.0.0/24*

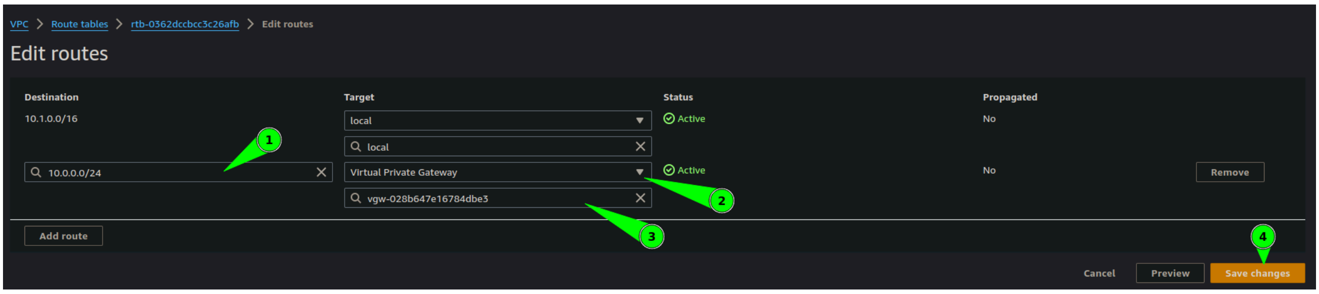


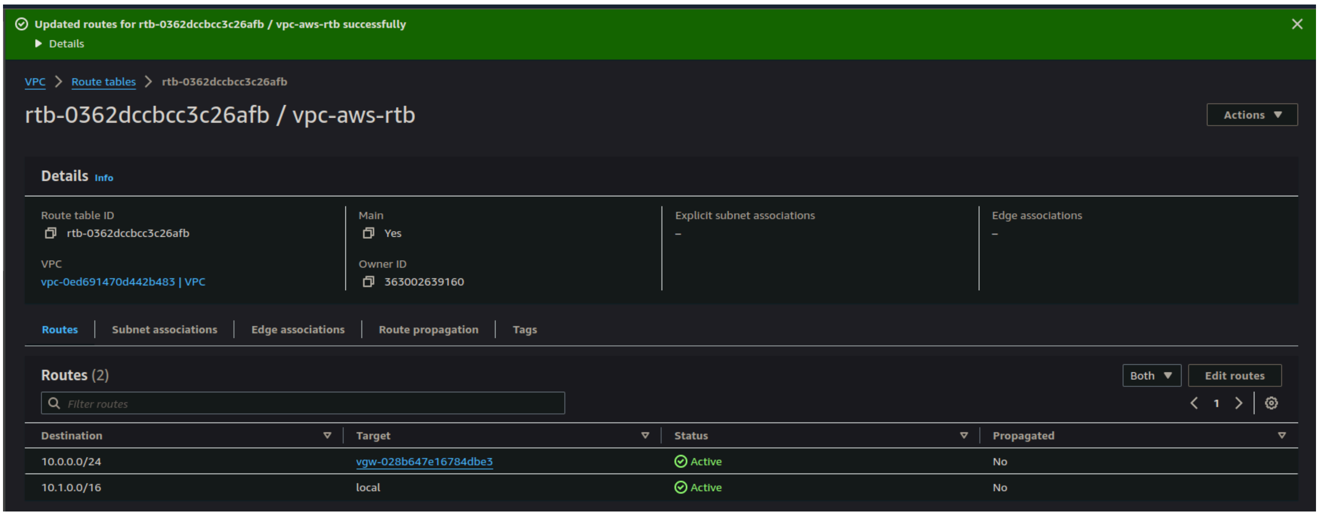
Lets go back to route tables now.

Before starting to configure the strongswan, lets make sure the route tables are set correctly, lets start with the AWS-VPC private-subnet. Since its closer to the **Virtual Private Gateway**. It needs to route traffic to the other on-premise-public-subnet. using the virtual private gateway.

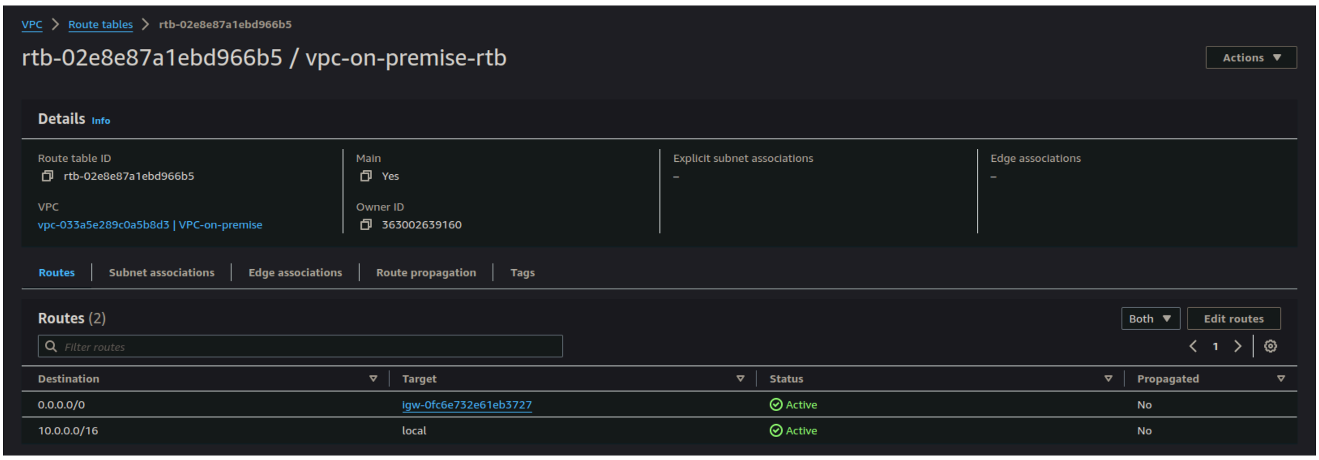
Vpc-aws routing table



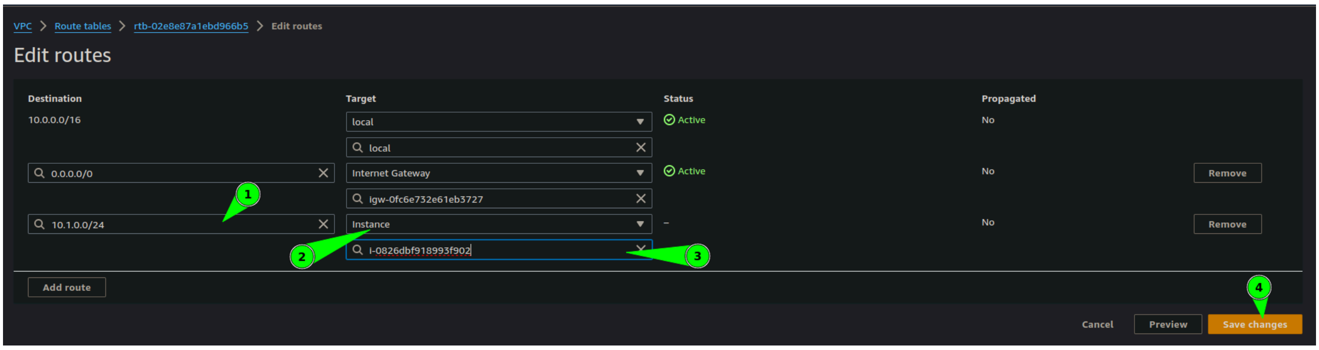




On-premise-public-subnet route table



We need to add the the instance of the on-premise to route to the `aws-private-subnet` CIDR Block 10.1.0.0/24. This connection is not peering, but since Site to Site tunnels are connected, it should have a path to the other VPC's subnet.



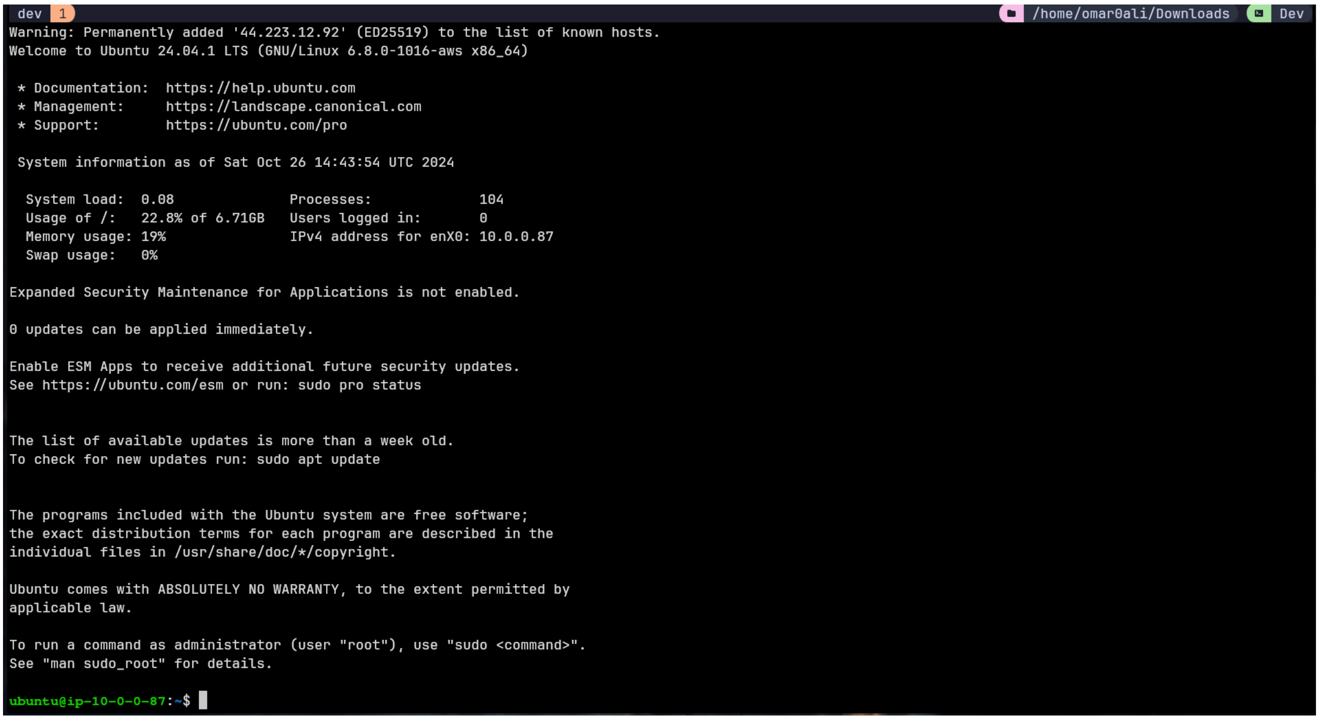
Lets configure the on-premise-instance

First, we need to login to our `on-premise-instance` to do that, we need to download our key from aws if you haven't. I did, and the first thing is to ensure the key can be only read. So using the linux command prompt type:

```
chmod 400 path/labsuser
```

Then we will need to connect to the instance, for example:

```
ssh -i labsuser.pem ubuntu@123.222.24.2
```



Configuration

Lets install `strongswan`

```
ubuntu@ip-10-0-0-87:~$ sudo apt install strongswan
```

Open `/etc/sysctl.conf`

> **Editor**
>
> I am using Vim to edit these files; you can use any editor, such as `nano`, which is also available.
> Please refer to the documentation for either Vim or Nano.

> **Check Your Configuration File**
>
> Check your configuration file downloaded from the Site To Site VPN Connection Page. You can follow the steps provided in the file. I am going through the steps used for strongswan.

```
sudo vim /etc/sysctl.conf
```

```
#
# /etc/sysctl.conf - Configuration file for setting system variables
# See /etc/sysctl.d/ for additional system variables.
# See sysctl.conf (5) for information.
#

#kernel.domainname = example.com

# Uncomment the following to stop low-level messages on console
#kernel.printk = 3 4 1 3

##############################################################
# Functions previously found in netbase
#

# Uncomment the next two lines to enable Spoof protection (reverse-path filter
)
# Turn on Source Address Verification in all interfaces to
# prevent some spoofing attacks
#net.ipv4.conf.default.rp_filter=1
#net.ipv4.conf.all.rp_filter=1

# Uncomment the next line to enable TCP/IP SYN cookies
# See http://lwn.net/Articles/277146/
# Note: This may impact IPv6 TCP sessions too
#net.ipv4.tcp_syncookies=1

# Uncomment the next line to enable packet forwarding for IPv4
net.ipv4.ip_forward=1          1

# Uncomment the next line to enable packet forwarding for IPv6
#  Enabling this option disables Stateless Address Autoconfiguration
#  based on Router Advertisements for this host
#net.ipv6.conf.all.forwarding=1


"/etc/sysctl.conf" 64L, 2208B written                    28,1          Top
```

Uncomment `net.ipv4.ip_forward=1`

Now we need to apply the changes:

```
sudo sysctl -p
```

Next, open `etc/ipsec.conf`

```
sudo vim /etc/ipsec.conf
```

```
dev  1                                   /home/omar0ali/Downloads   Dev
# ipsec.conf - strongSwan IPsec configuration file

# basic configuration

config setup
        # strictcrlpolicy=yes
        uniqueids = no
```

Uncomment `uniqueids = no`

And copy the following from your config file. And paste it at the end of `/etc/ipsec.conf` file.

```
# AWS VPN will also support AES256 and SHA256 for the "ike" (Phase 1) and "esp" (Phase 2) entries below.
# For Phase 1, AWS VPN supports DH groups 2, 14-18, 22, 23, 24. Phase 2 supports DH groups 2, 5, 14-18, 22, 23, 24
# To see Strongswan's syntax for these different values, please refer to https://wiki.strongswan.org/projects/strongswan/wiki/IKEv1CipherSuites

conn Tunnel1
        auto=start
        left=%defaultroute
        leftid=44.223.12.92
        right=3.216.254.210
        type=tunnel
        leftauth=psk
        rightauth=psk
        keyexchange=ikev1
        ike=aes128-sha1-modp1024
        ikelifetime=8h
        esp=aes128-sha1-modp1024
        lifetime=1h
        keyingtries=%forever
        leftsubnet=0.0.0.0/0
        rightsubnet=0.0.0.0/0
        dpddelay=10s
        dpdtimeout=30s
        dpdaction=restart
        ## Please note the following line assumes you only have two tunnels in your Strongswan configuration file. This "mark" value must be unique and may need to be changed based on other entries in your configuration file.
        mark=100
        ## Uncomment the following line to utilize the script from the "Automated Tunnel Healhcheck and Failover" section. Ensure that the integer after "-m" matches the "mark" value above, and <VPC CIDR> is replaced with the CIDR of your VPC
        ## (e.g. 192.168.1.0/24)
        #leftupdown="/etc/ipsec.d/aws-updown.sh -ln Tunnel1 -ll 169.254.198.158/30 -lr 169.254.198.157/30 -m 100 -r <VPC CIDR>"
```

We will need to modify the last line. Uncomment the last line that starts with `leftupdown=...` and update the `<VPC CIDR>` This CIDR range is the AWS `VPC` not the `on-premise-vpc`. Which is `10.1.0.0/16`

```
conn Tunnel1
        auto=start
        left=%defaultroute
        leftid=44.223.12.92
        right=3.216.254.210
        type=tunnel
        leftauth=psk
        rightauth=psk
        keyexchange=ikev1
        ike=aes128-sha1-modp1024
        ikelifetime=8h
        esp=aes128-sha1-modp1024
        lifetime=1h
        keyingtries=%forever
        leftsubnet=0.0.0.0/0
        rightsubnet=0.0.0.0/0
        dpddelay=10s
        dpdtimeout=30s
        dpdaction=restart
        ## Please note the following line assumes you only have two tunnels in
 your Strongswan configuration file. This "mark" value must be unique and may
need to be changed based on other entries in your configuration file.
        mark=100
        ## Uncomment the following line to utilize the script from the "Automa
ted Tunnel Healhcheck and Failover" section. Ensure that the integer after "-m
" matches the "mark" value above, and <VPC CIDR> is replaced with the CIDR of
your VPC
        ## (e.g. 192.168.1.0/24)
        leftupdown="/etc/ipsec.d/aws-updown.sh -ln Tunnel1 -ll 169.254.198.158
/30 -lr 169.254.198.157/30 -m 100 -r 10.1.0.0/16"
```

This is **tunnel 1**, we will do the same thing for **tunnel 2**. First, find it from your configuration file.

```
conn Tunnel2
        auto=start
        left=%defaultroute
        leftid=44.223.12.92
        right=34.236.97.73
        type=tunnel
        leftauth=psk
        rightauth=psk
        keyexchange=ikev1
        ike=aes128-sha1-modp1024
        ikelifetime=8h
        esp=aes128-sha1-modp1024
        lifetime=1h
        keyingtries=%forever
        leftsubnet=0.0.0.0/0
        rightsubnet=0.0.0.0/0
        dpddelay=10s
        dpdtimeout=30s
        dpdaction=restart
        ## Please note the following line assumes you only have two tunnels in your Strongswan
configuration file. This "mark" value must be unique and may need to be changed based on other entries in
your configuration file.
        mark=200
        ## Uncomment the following line to utilize the script from the "Automated Tunnel Healhcheck and
Failover" section. Ensure that the integer after "-m" matches the "mark" value above, and <VPC CIDR> is
replaced with the CIDR of your VPC
        ## (e.g. 192.168.1.0/24)
        #leftupdown="/etc/ipsec.d/aws-updown.sh -ln Tunnel2 -ll 169.254.246.50/30 -lr 169.254.246.49/30 -
m 200 -r <VPC CIDR>"
```

Then copy it and uncommmment the last line that starts with `leftupdown=...` and update the

`<VPC CIDR>` This CIDR range is the `AWS-VPC` not the `on-premise-vpc`. Which is `10.1.0.0/16`

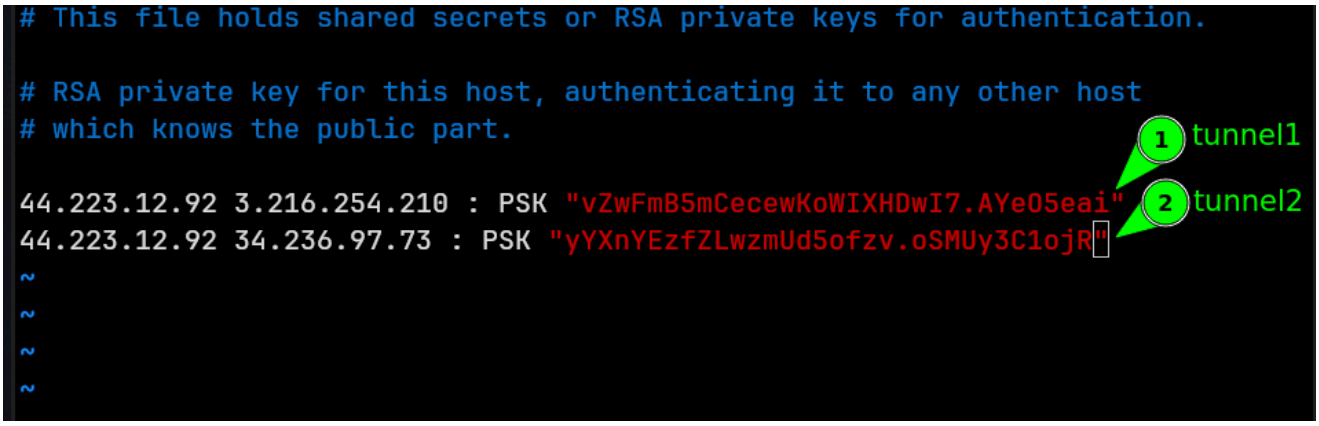
Next, open `/etc/ipsec.secrets`

```
sudo vim /etc/ipsec.secrets
```

> 4) Create a new file at /etc/ipsec.secrets if it doesn't already exist, and append this line to the file (be mindful of the spacing!). This value authenticates the tunnel endpoints:
> 44.223.12.92 3.216.254.210 : PSK "vZwFmB5mCecewKoWIXHDwI7.AYeO5eai"

This is the shared secret for the tunnel 1, there is also another shared secret for tunnel 2.

4) Create a new file at /etc/ipsec.secrets if it doesn't already exist, and append this line to the file (be mindful of the spacing!). This value authenticates the tunnel endpoints:
44.223.12.92 34.236.97.73 : PSK "yYXnYEzfZLwzmUd5ofzv.oSMUy3C1ojR"



```
# This file holds shared secrets or RSA private keys for authentication.

# RSA private key for this host, authenticating it to any other host
# which knows the public part.

44.223.12.92 3.216.254.210 : PSK "vZwFmB5mCecewKoWIXHDwI7.AYeO5eai"
44.223.12.92 34.236.97.73 : PSK "yYXnYEzfZLwzmUd5ofzv.oSMUy3C1ojR"
~
~
~
~
```


Copy them and add them to the file.

Next, we will create a new file. At `/etc/ipsec.d/aws-updown` and copy the following code from your configuration file.

```
sudo vim /etc/ipsec.d/aws-updown.sh
```

```
=== HOW-TO ===
1) Create a new file at /etc/ipsec.d/aws-updown.sh if it doesn't already exist, and append the following
script to the file:

#!/bin/bash

while [[ $# > 1 ]]; do
        case ${1} in
                -ln|--link-name)
                        TUNNEL_NAME="${2}"
                        TUNNEL_PHY_INTERFACE="${PLUTO_INTERFACE}"
                        shift
                        ;;
                -ll|--link-local)
                        TUNNEL_LOCAL_ADDRESS="${2}"
                        TUNNEL_LOCAL_ENDPOINT="${PLUTO_ME}"
                        shift
                        ;;
                -lr|--link-remote)
                        TUNNEL_REMOTE_ADDRESS="${2}"
                        TUNNEL_REMOTE_ENDPOINT="${PLUTO_PEER}"
                        shift
                        ;;
                -m|--mark)
                        TUNNEL_MARK="${2}"
                        shift
                        ;;
                -r|--static-route)
                        TUNNEL_STATIC_ROUTE="${2}"
```

Copy the whole code

There is also a minor tweak required on this code. There is a function called `add_route()`

```
add_route() {
        IFS=',' read -ra route <<< "${TUNNEL_STATIC_ROUTE}"
        for i in "${route[@]}"; do
            ip route add ${i} dev ${TUNNEL_NAME} metric ${TUNNEL_MARK} src 10.0.0.87
        done
        iptables -t mangle -A FORWARD -o ${TUNNEL_NAME} -p tcp --tcp-flags SYN,RST SYN -j TCPMSS --clamp-mss-to-pmtu
        iptables -t mangle -A INPUT -p esp -s ${TUNNEL_REMOTE_ENDPOINT} -d ${TUNNEL_LOCAL_ENDPOINT} -j MARK --set-xmark ${TUNNEL_MARK}
        ip route flush table 220
}
```

That is my local private ip address on the `on-premise-instance`. After saving the files.

Run `sudo chmod 744 /etc/ipsec.d/aws-updown.sh`

We will need to restart the `ipsec` with the following command.

```
sudo ipsec restart
```

```
ubuntu@ip-10-0-0-87:~$ sudo chmod 744 /etc/ipsec.d/aws-updown.sh
ubuntu@ip-10-0-0-87:~$ sudo ipsec restart
Stopping strongSwan IPsec...
Starting strongSwan 5.9.13 IPsec [starter]...
ubuntu@ip-10-0-0-87:~$
```

Lets check the status of the `ipsec`

```
sudo ipsec status
```

```
ubuntu@ip-10-0-0-87:~$ sudo ipsec status
Security Associations (2 up, 0 connecting):
     Tunnel2[2]: ESTABLISHED 91 seconds ago, 10.0.0.87[44.223.12.92]...34.236.97.73[34.236.97.
73]
     Tunnel2{1}:  INSTALLED, TUNNEL, reqid 1, ESP in UDP SPIs: cbfe0d3c_i cd5f38cb_o
     Tunnel2{1}:   10.0.0.0/16 === 10.1.0.0/16
     Tunnel1[1]: ESTABLISHED 91 seconds ago, 10.0.0.87[44.223.12.92]...3.216.254.210[3.216.254
.210]
     Tunnel1{2}:  INSTALLED, TUNNEL, reqid 2, ESP in UDP SPIs: caa9f1a2_i cb012ac7_o
     Tunnel1{2}:   10.0.0.0/16 === 10.1.0.0/16
ubuntu@ip-10-0-0-87:~$
```

All the tunnels are established and installed.

Lets check `ifconfig` but first we need to install `net-tools`

```
sudo apt install net-tools
```

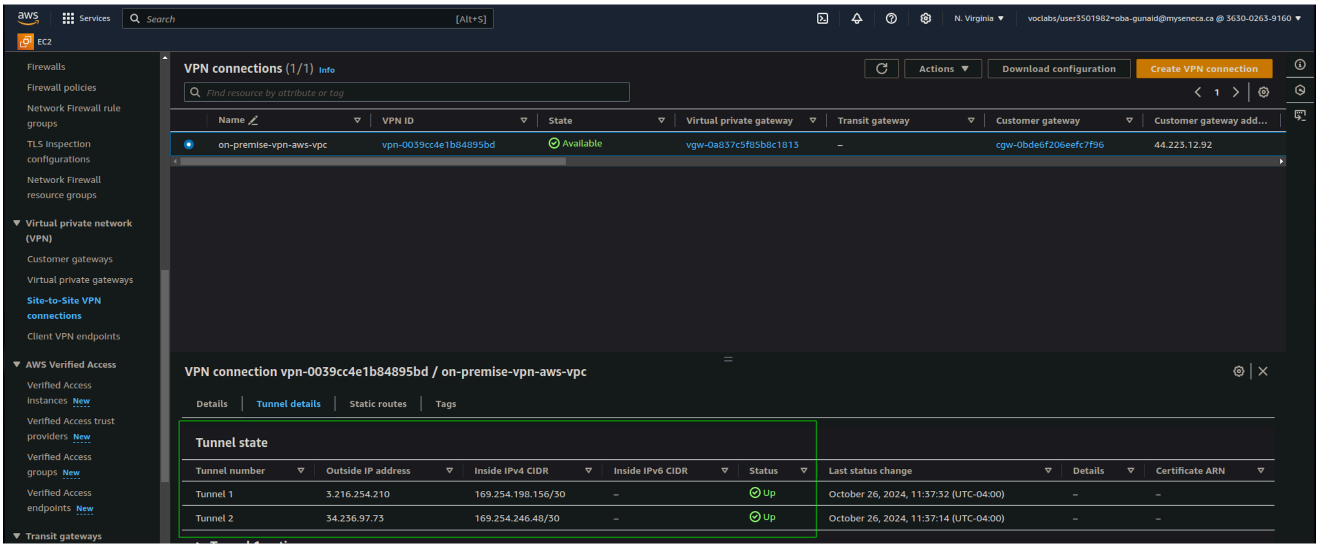
Then execute `ifconfig`

```
ubuntu@ip-10-0-0-87:~$ ifconfig
Tunnel1: flags=209<UP,POINTOPOINT,RUNNING,NOARP>  mtu 1419
        inet 169.254.198.158  netmask 255.255.255.252  destination 169.254.198.157
        inet6 fe80::5efe:a00:57  prefixlen 64  scopeid 0x20<link>
        tunnel   txqueuelen 1000  (IPIP Tunnel)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

Tunnel2: flags=209<UP,POINTOPOINT,RUNNING,NOARP>  mtu 1419
        inet 169.254.246.50  netmask 255.255.255.252  destination 169.254.246.49
        inet6 fe80::5efe:a00:57  prefixlen 64  scopeid 0x20<link>
        tunnel   txqueuelen 1000  (IPIP Tunnel)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

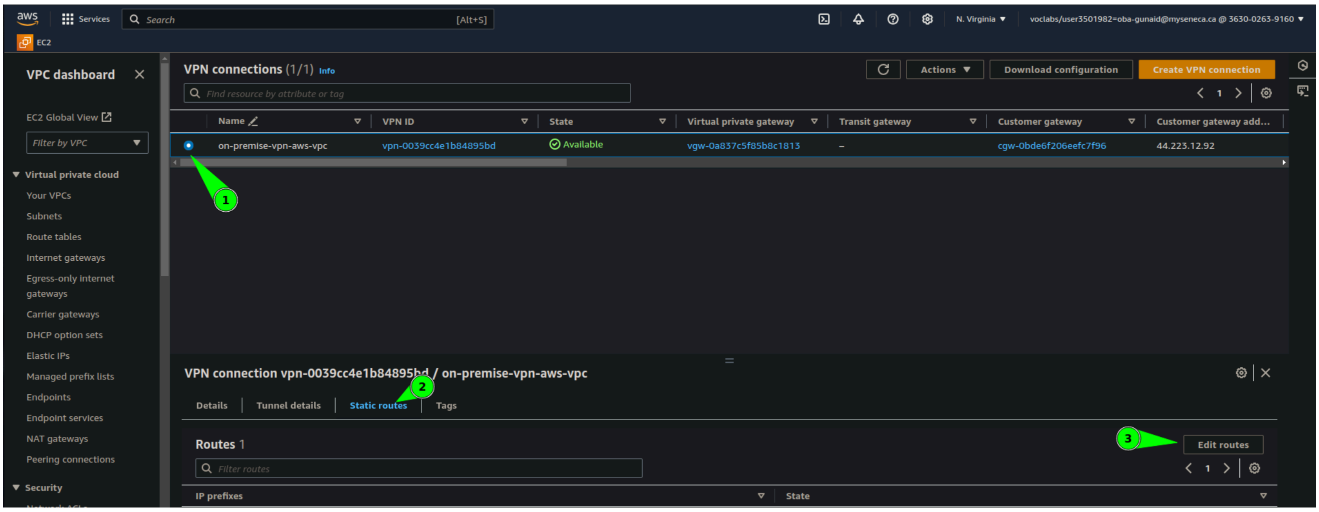
You can see, all the tunnels are up.

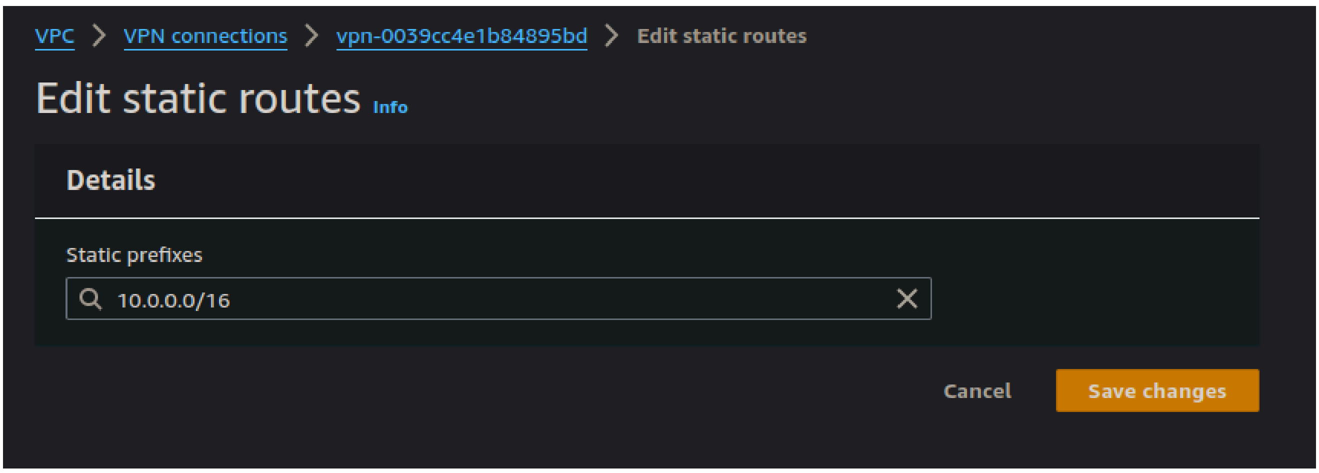
Lets check our **Site To Site VPN** and confirm that the tunnels are working.



The last part (static route & ping)

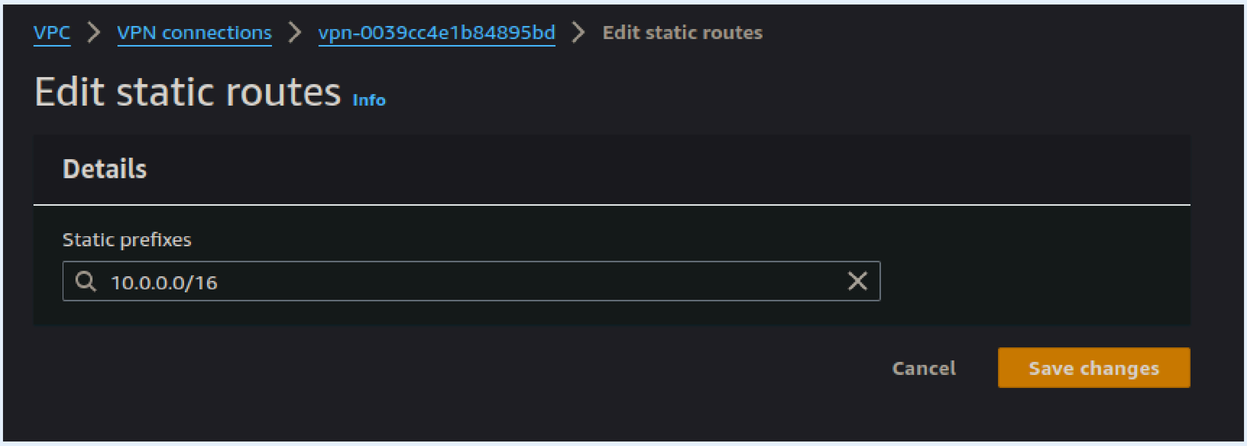
Before we ping the private aws-instance using its private ip address, we will need to add the `on-premise-instance` private IP address or its VPC CIDR block, in the static routes of the Site To Site VPN we created.





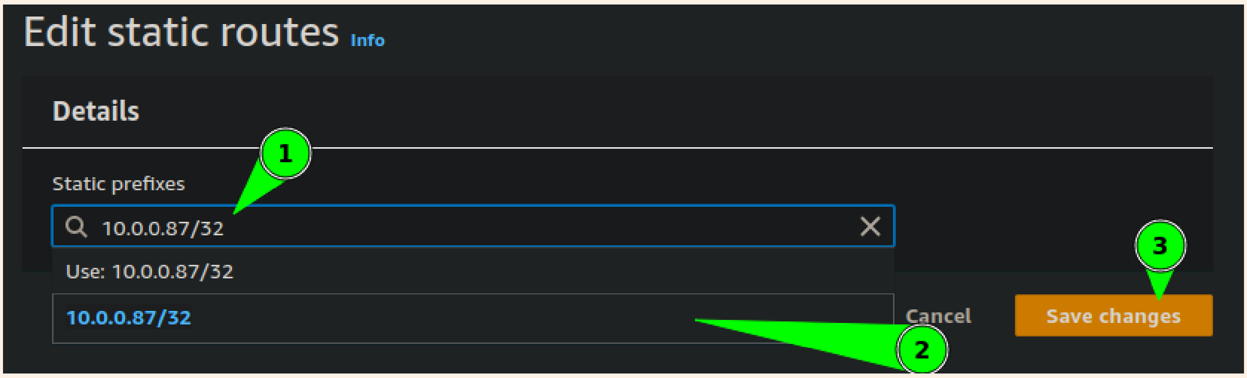
NOTE

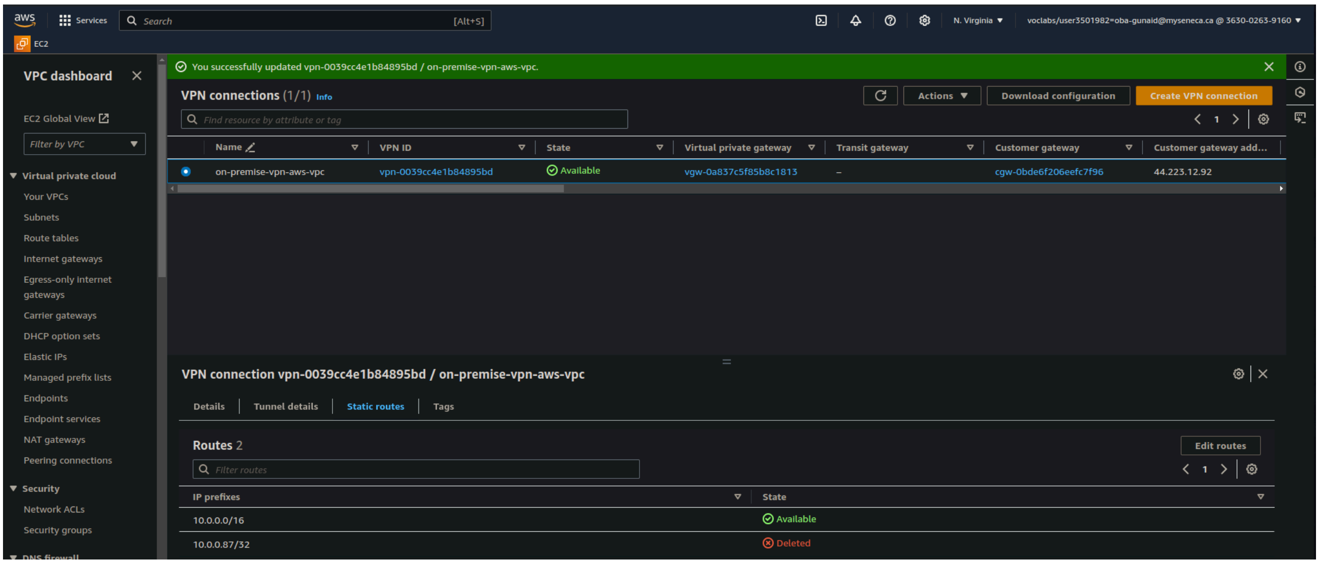
You can set the whole range of the on-premise-vpc, or just a specific IP address i.e the on-premise-instance.



You can add one of these two.

⚠ **Don't forget to click on it *from the drop-down list* and save changes.**

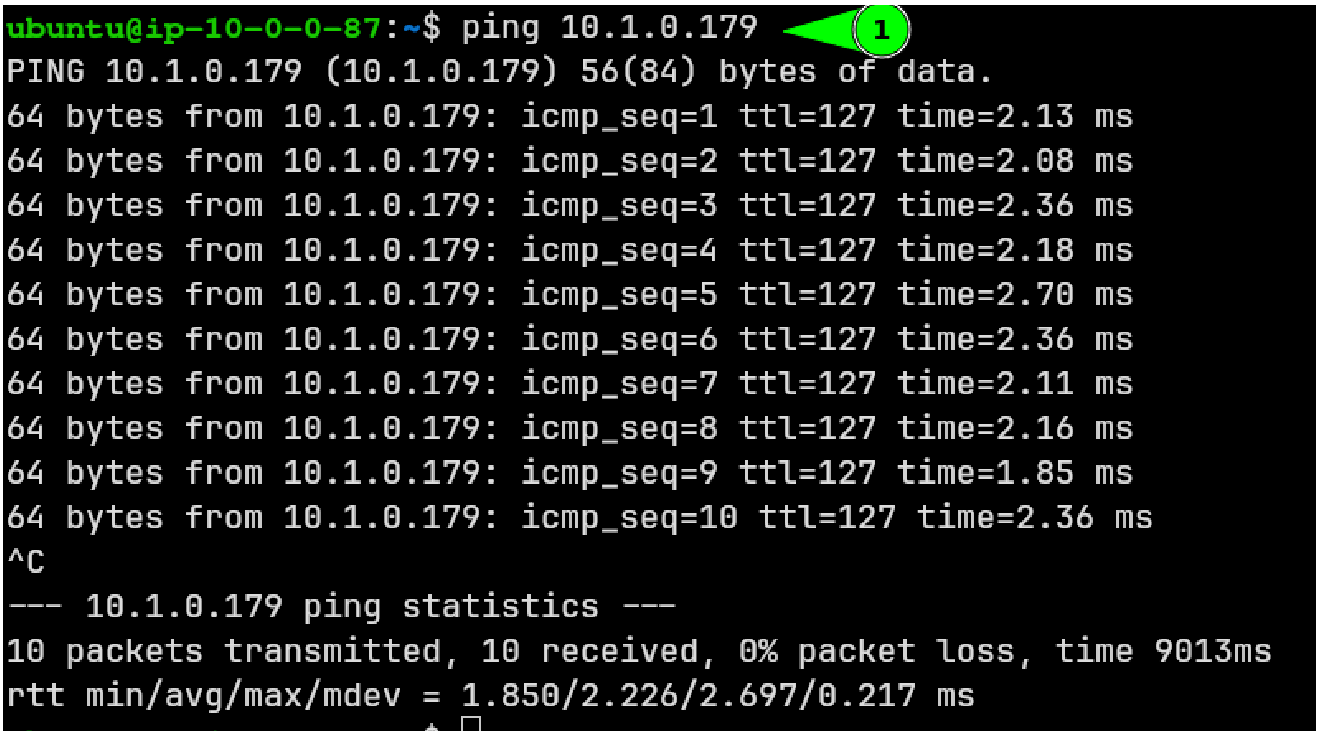




For testing

I have tested it in both ways, from a specific range i.e 10.0.0.0/16 (vpc-on-premise), and from a specific private ip address of an instance 10.0.0.87/32 (on-premise-instance).

Ping from 10.0.0.87 (on-premise-instance) to 10.1.0.179 (aws-instance)



With a few additional modifications to the AWS VPC, we can add more public and private subnets and possibly another bastion host or multiple instances. We can also SSH into any AWS instance using only its private IP address, but we must enable SSH access in the security groups.

Additionally, it’s important to note that, since we included the entire on-premises CIDR range in the Site-to-Site VPN configuration *static IPs*, any instance within that on-premises CIDR block can securely access resources within the AWS VPC using their private ip addresses, as long as appropriate routing and security group rules are configured.

Thank you,
Omar BaGunaid