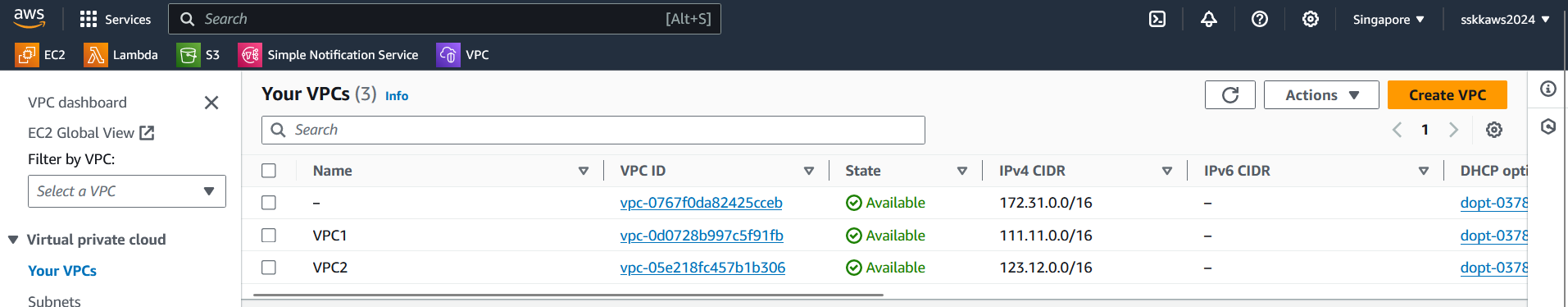
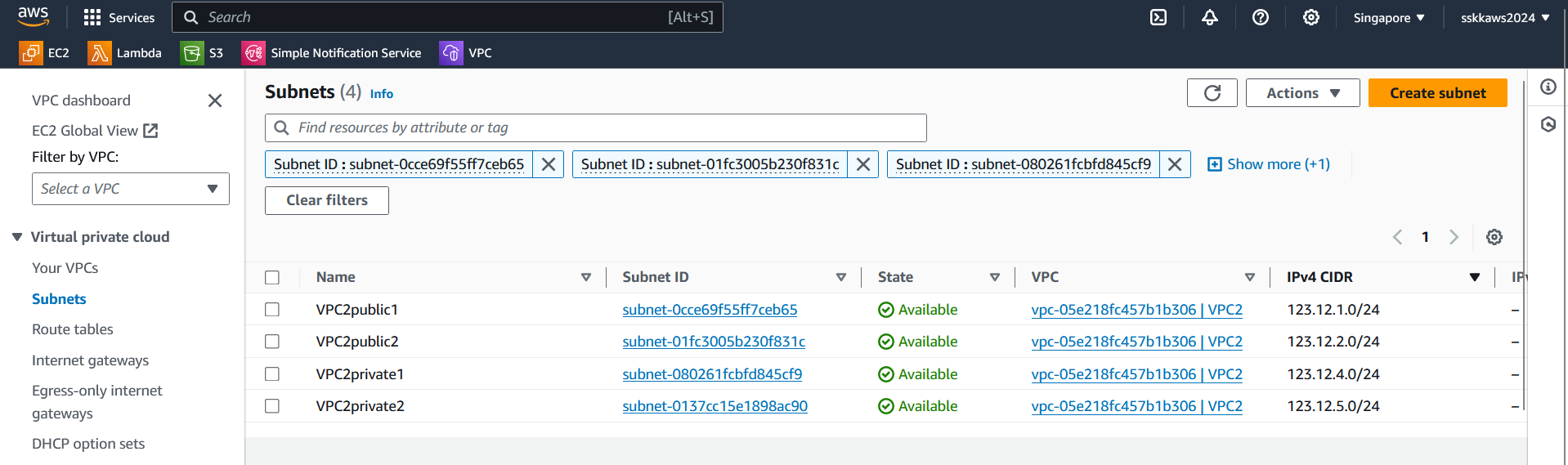
**Setting Up a Peering Connection - Capstone Project**

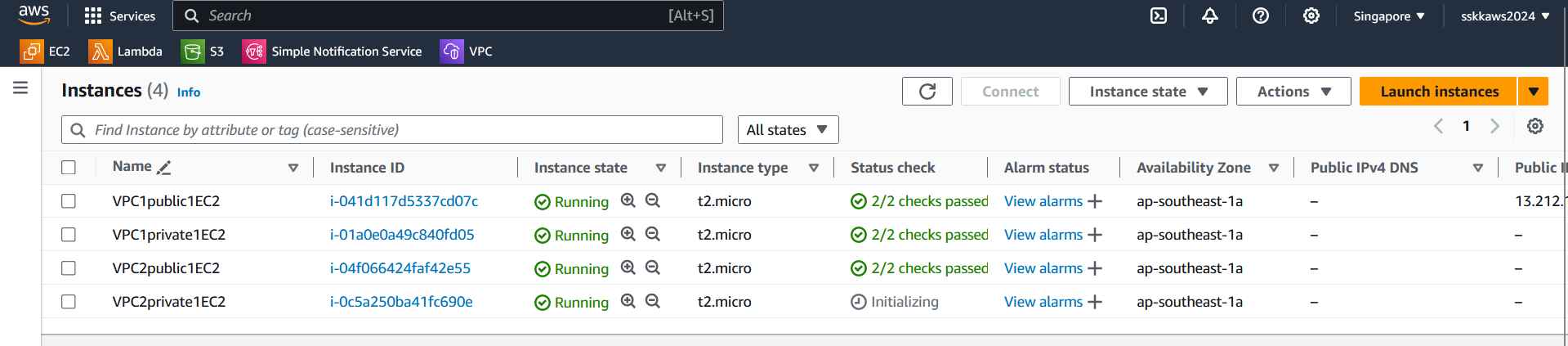
**Complete setup of Environment**

2 VPCs with public subnets and private subnets required for peering are to be configured with the following CIDR blocks. This test environment can be manually setup with configuring every single component. Alternative suggestion is to use a cloudformation template given in the appendix to create the test environment.

|  |  |  |
| --- | --- | --- |
|  | VPC1 | VPC2 |
| CIDR block | 111.11.0.0/16 | 123.12.0.0/16 |
| CIDR of public subnet | 111.11.1.0/24  111.11.2.0/24 | 123.12.1.0/24  123.12.2.0/24 |
| CIDR of private subnet | 111.11.4.0/24  111.11.5.0/24 | 123.12.4.0/24  123.12.5.0/24 |

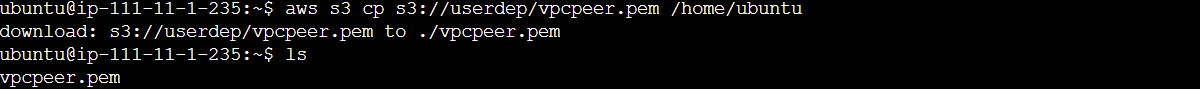


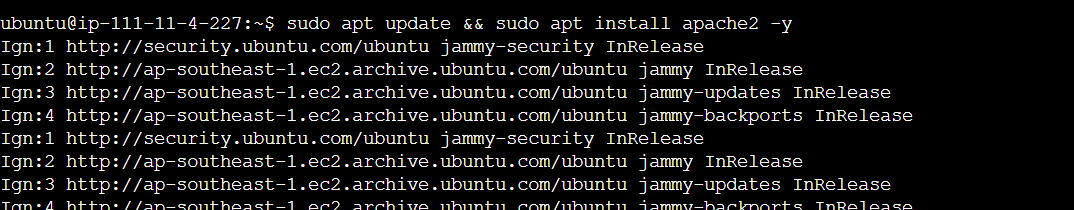
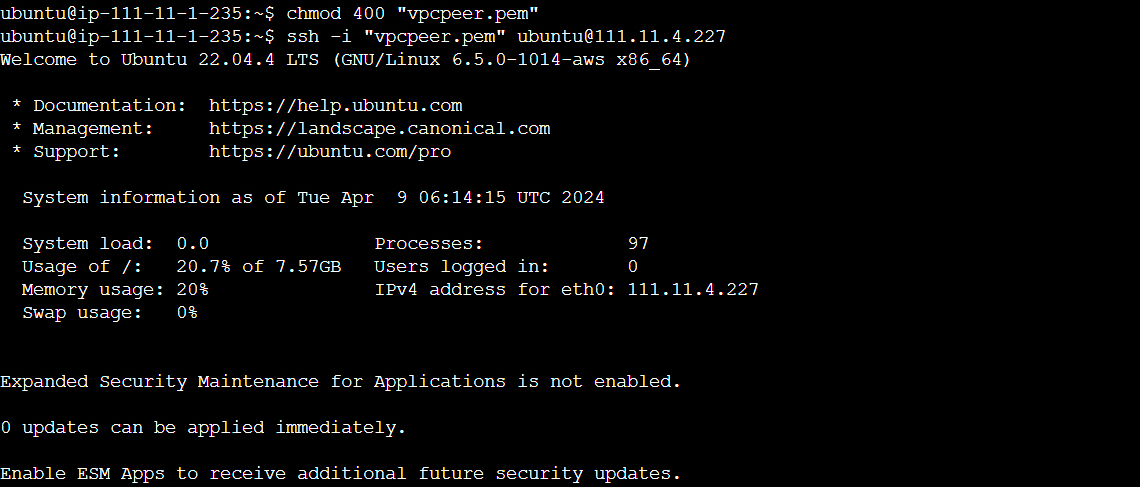


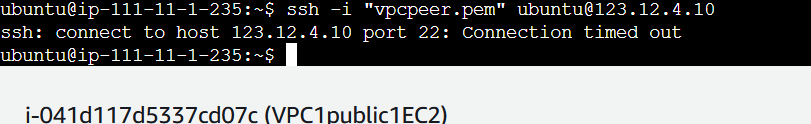


Initially verify that the instances in public subnet can access the instances in private subnets.

VPC1public1EC2 can acces the instance VPC1private1EC2, but cannot access VPC2private1EC2. Similarly resources in VPC1 cannot be accessed by resources in VPC2.





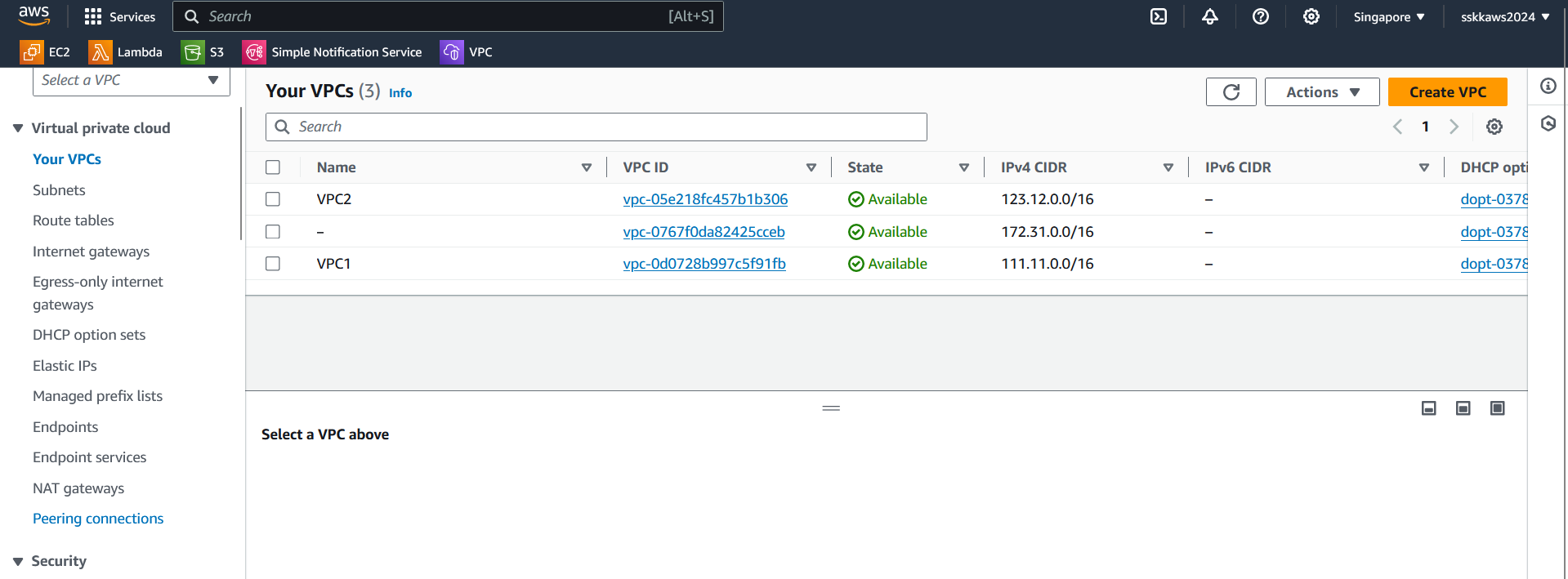


**VPC peering setup**

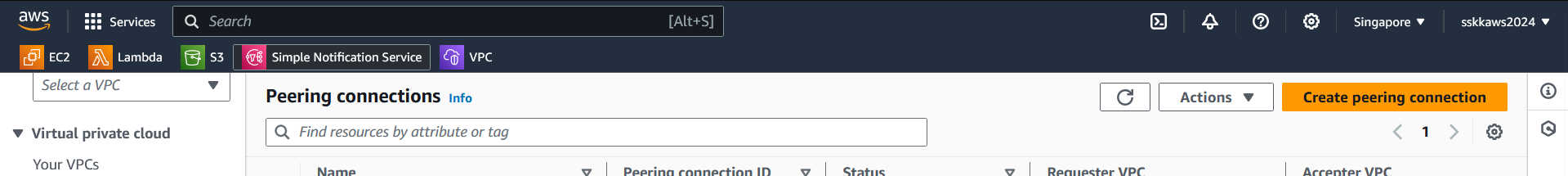
To establish a VPC peering connection between VPC\_1 and VPC\_2 and ensure that all subnets in VPC\_2 are accessible from VPC\_1, follow these steps:

1. **Create Peering Connection:**

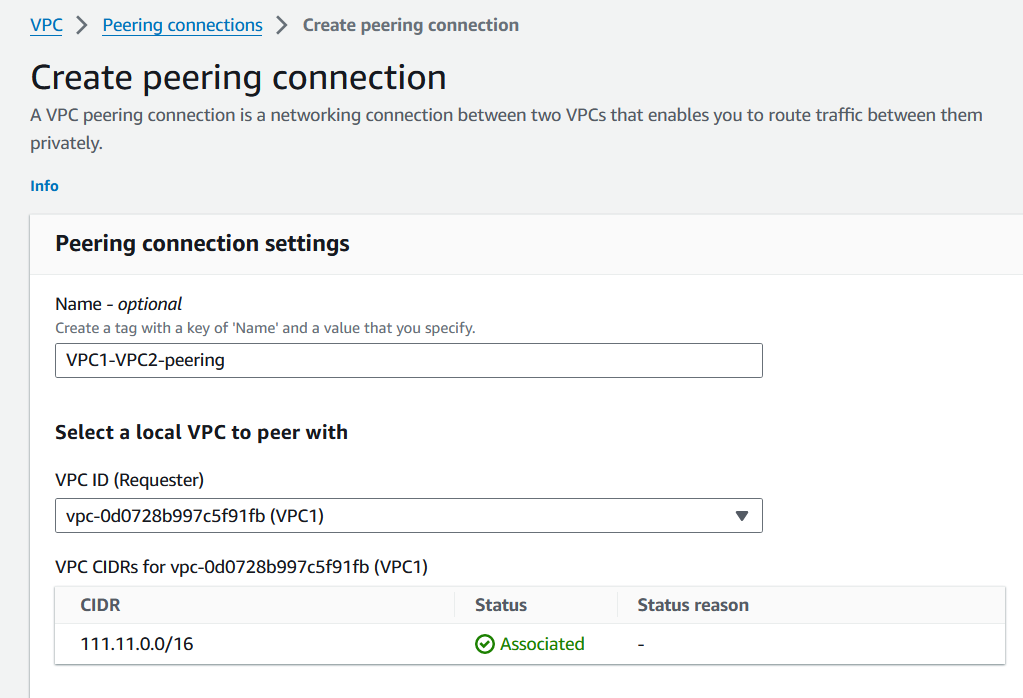
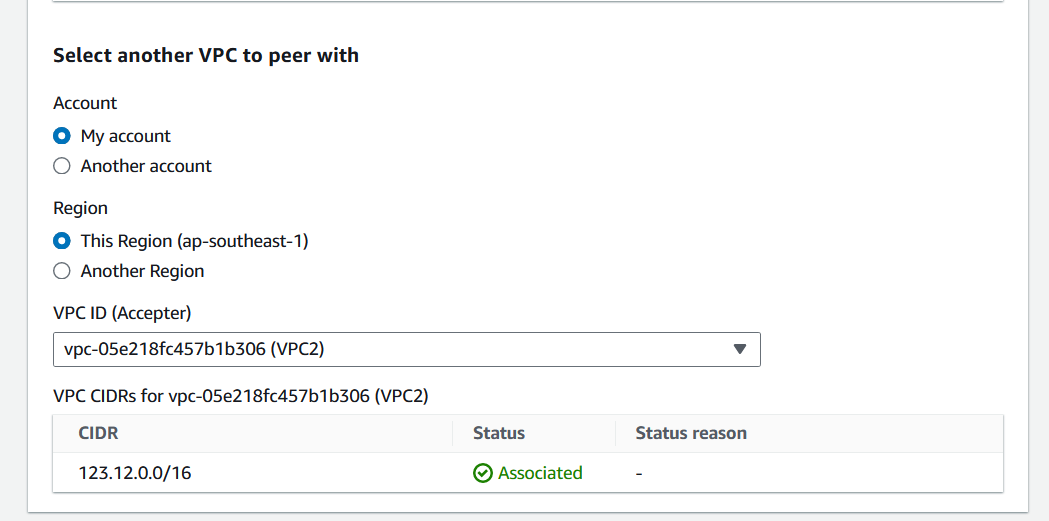
* In the VPC dashboard, locate the "Peering Connections" section.



* Click on "Create Peering Connection".



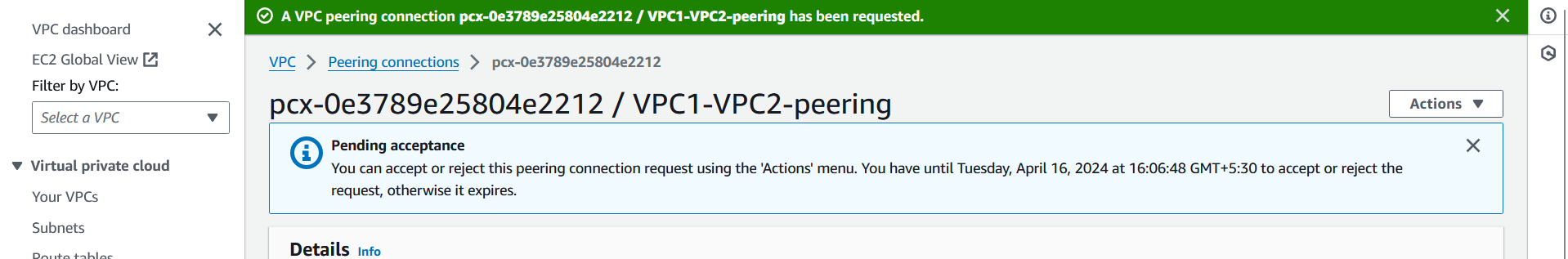
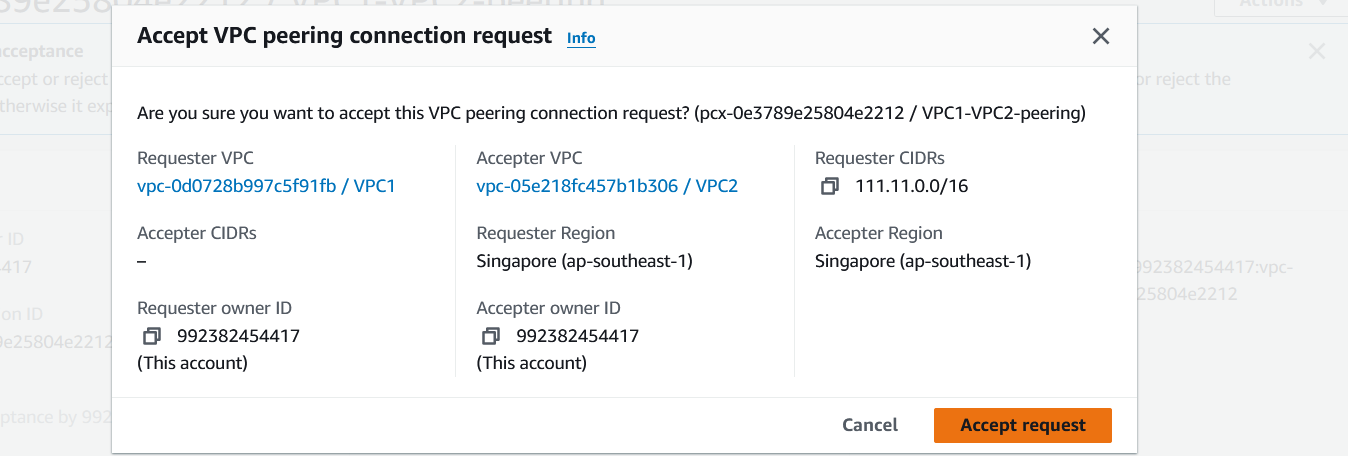
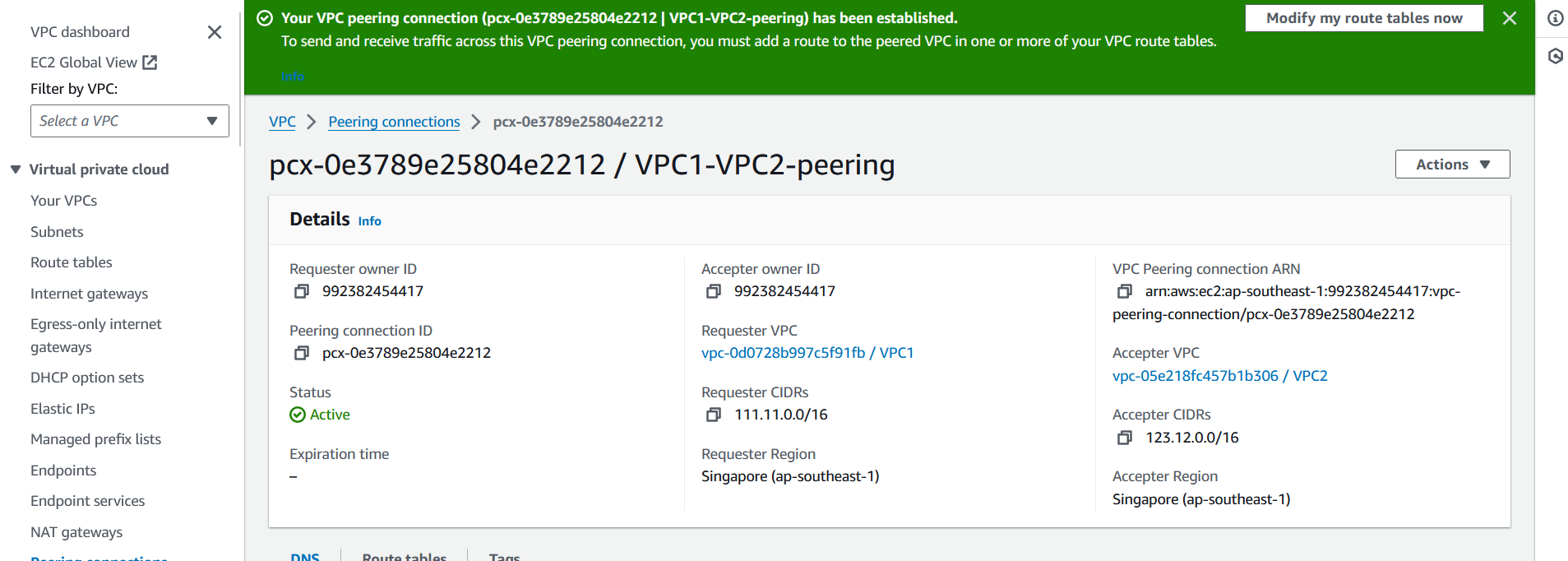
* Select VPC\_1 as the requester VPC and VPC\_2 as the peer VPC.

* Provide a unique name for the peering connection.
* Review the settings and create the peering connection.

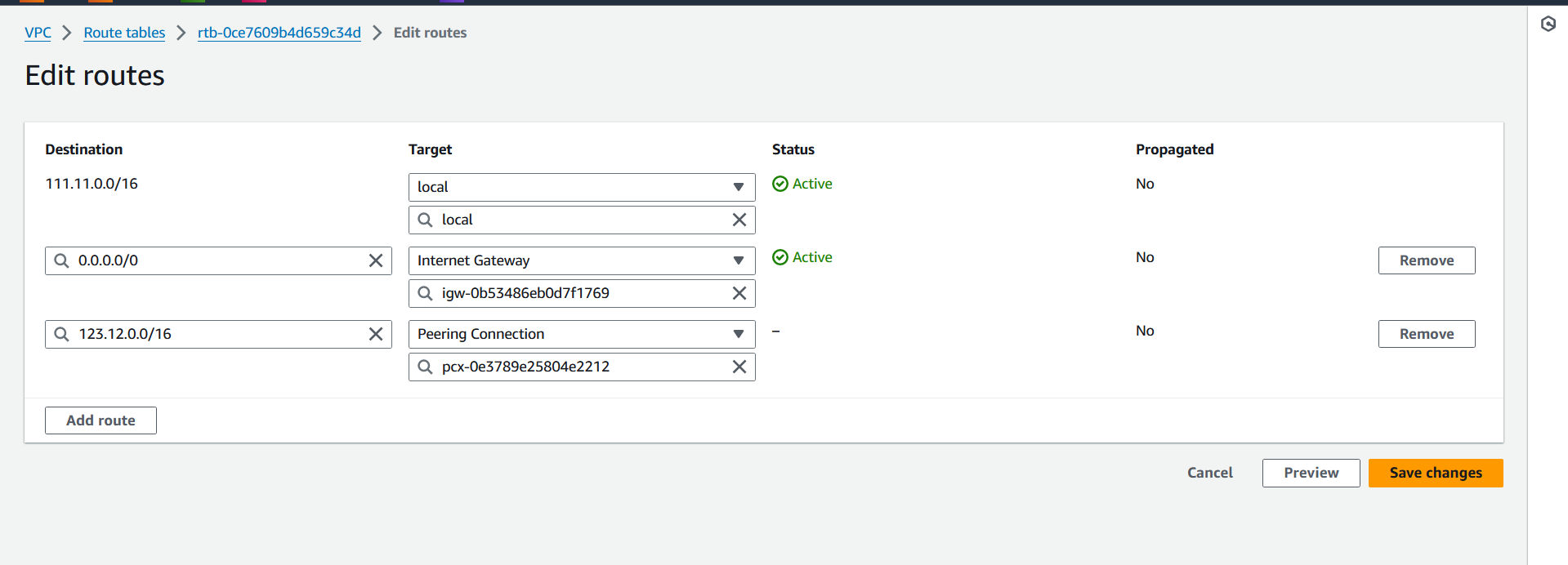
1. **Accept Peering Connection:**

* After creating the peering connection, switch to the VPC dashboard of VPC\_2.
* You should see a pending peering request. Accept the request.

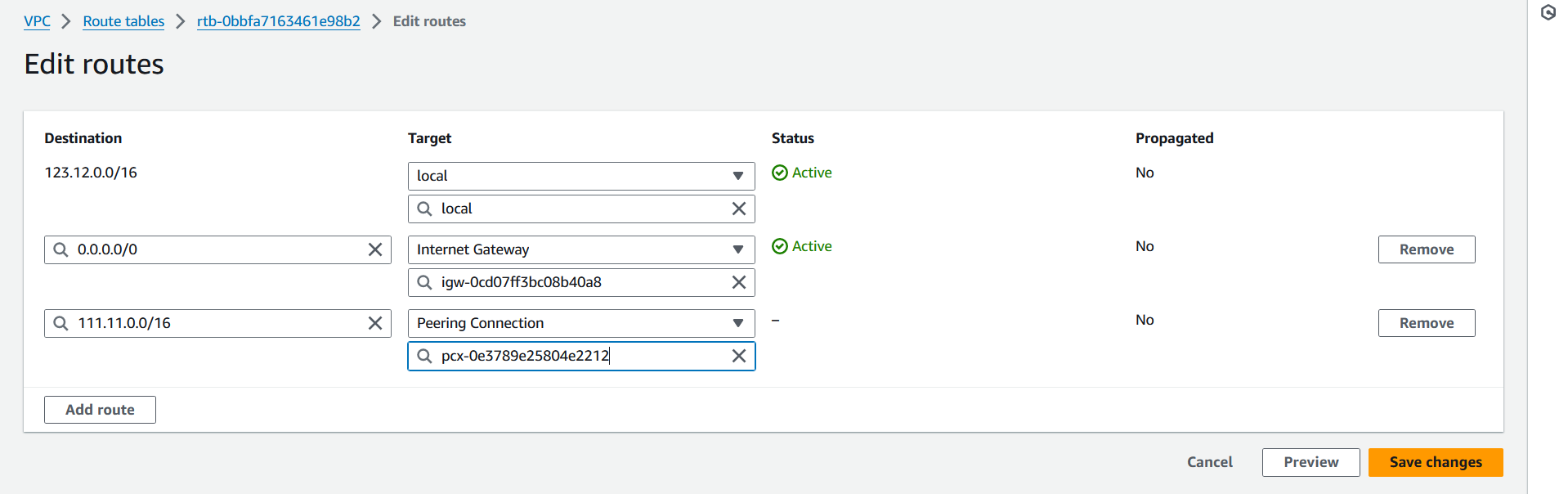
  

1. **Update Route Tables:**

* In the VPC dashboard, navigate to the route tables associated with the subnets in both VPCs.
* Edit the route table associated with the subnets in VPC\_1:
* Add a route for the CIDR block of VPC\_2, pointing to the peering connection.



* Edit the route table associated with the subnets in VPC\_2:
* Add a route for the CIDR block of VPC\_1, pointing to the peering connection.

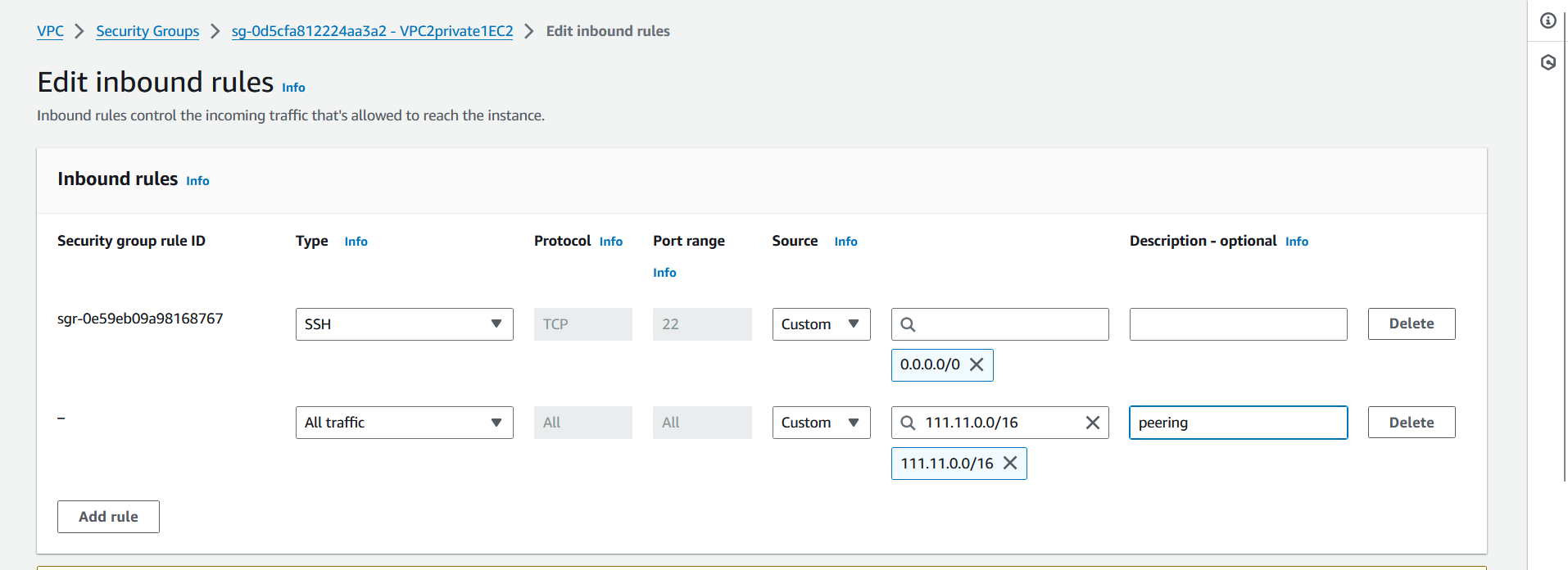
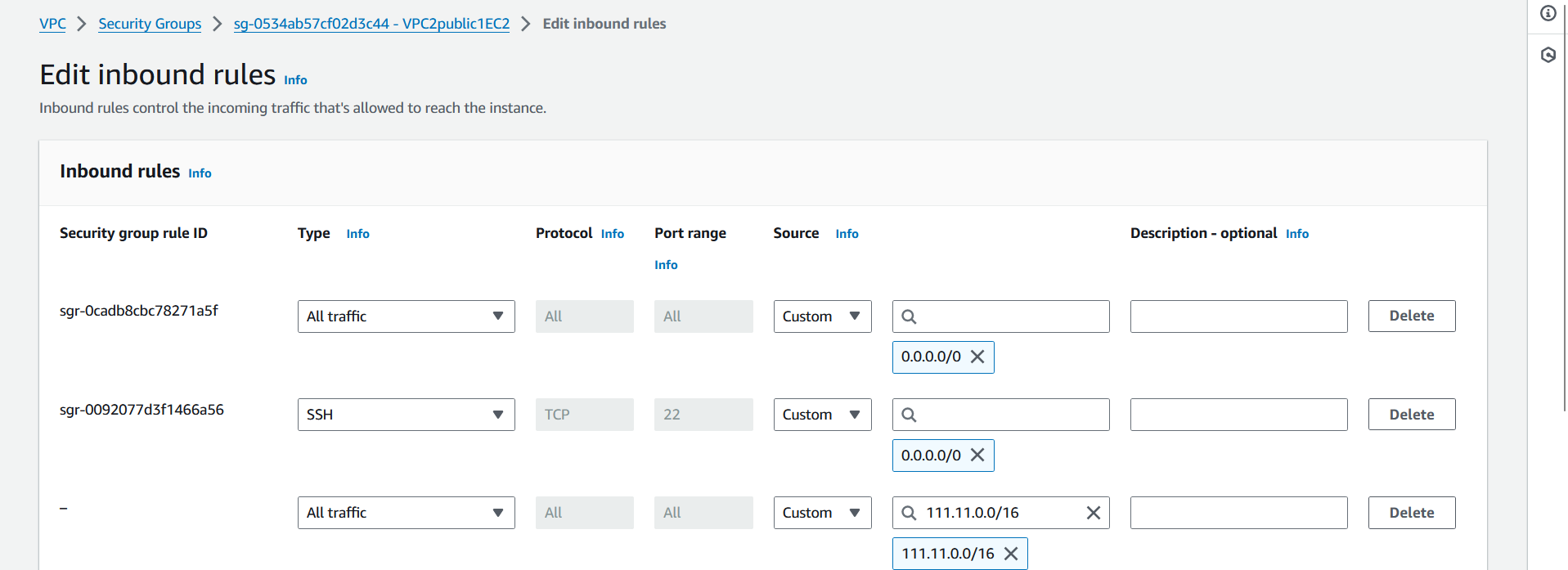


1. **Check Security Group and Network ACL Settings:**

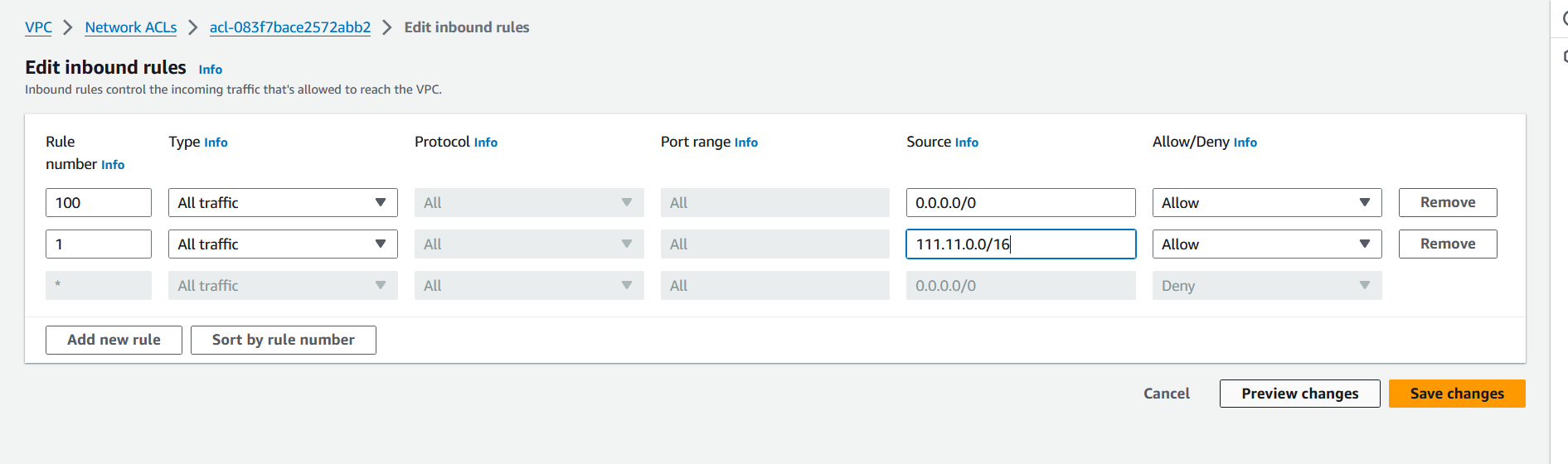
* Review the security group and network ACL settings in both VPCs to ensure they allow traffic between the peered subnets.
* Adjust security group rules and network ACLs if necessary to permit the desired traffic.

1. **Inbound Rules**:

* In the security groups associated with the instances in the subnets of VPC\_2:
* Allow inbound traffic from the CIDR block of VPC\_1 on the required ports and protocols. This will enable instances in VPC\_1 to communicate with instances in VPC\_2.

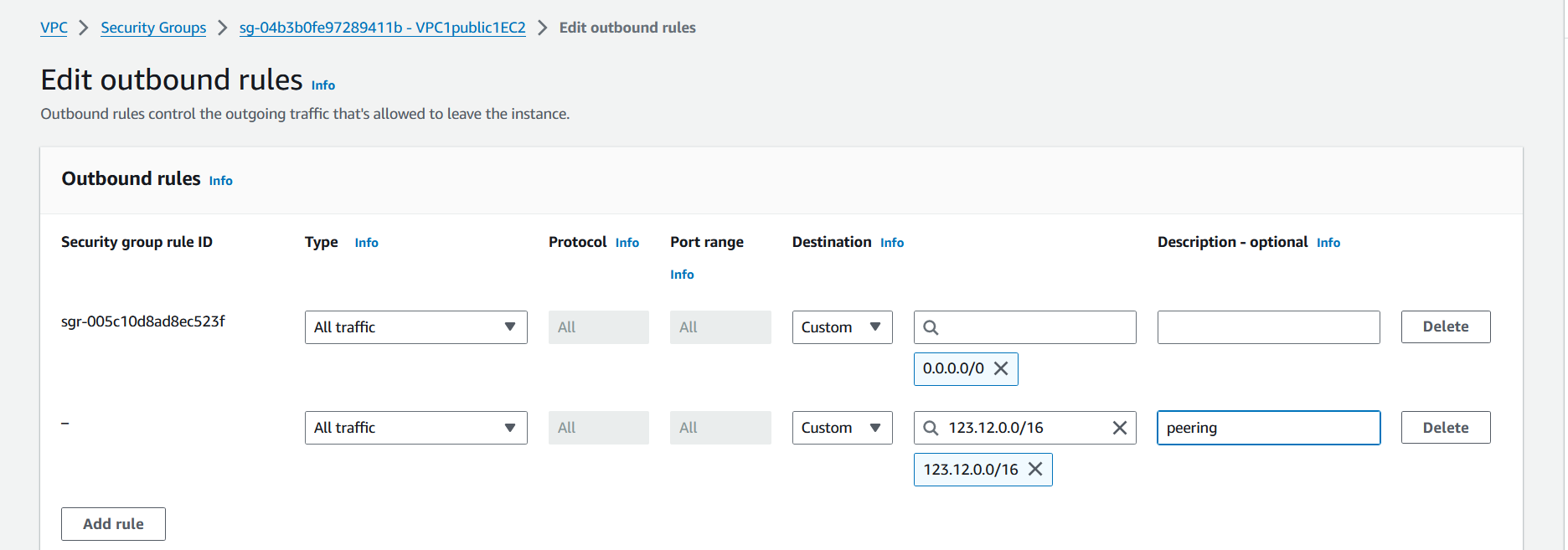
 

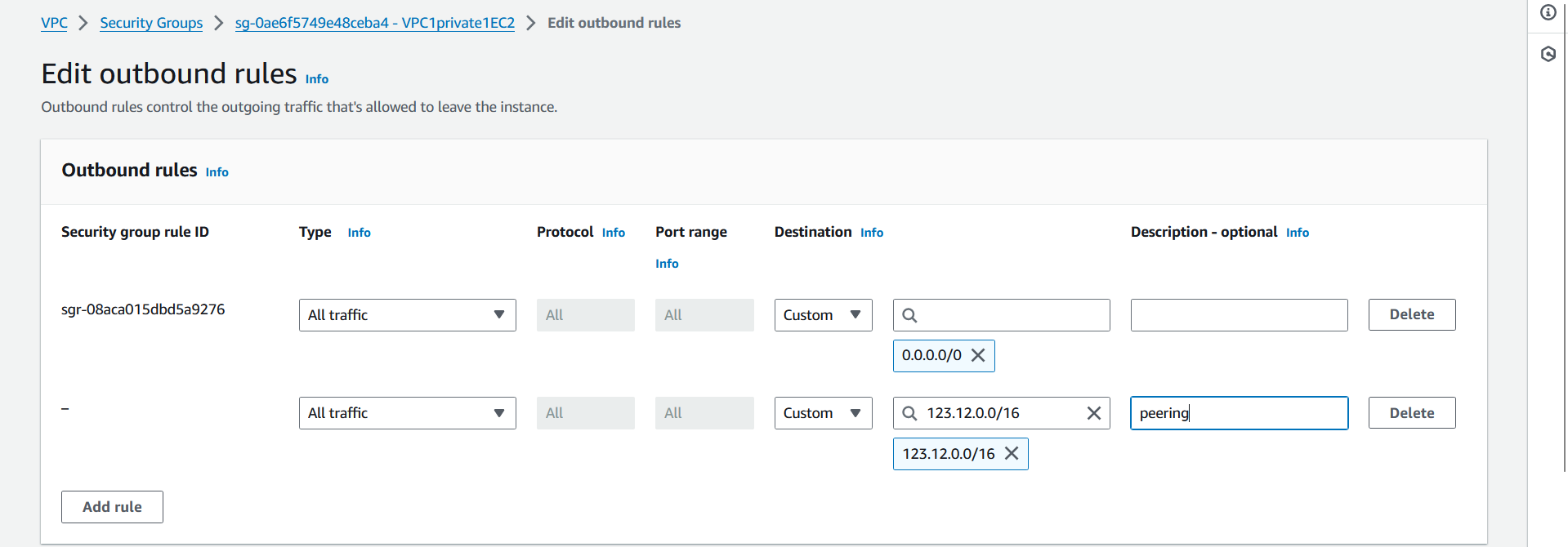
* In the network ACLs associated with the subnets in VPC\_2:
* Allow inbound traffic from the CIDR block of VPC\_1 on the required ports and protocols. Network ACLs are stateless, so both inbound and outbound rules need to be configured.



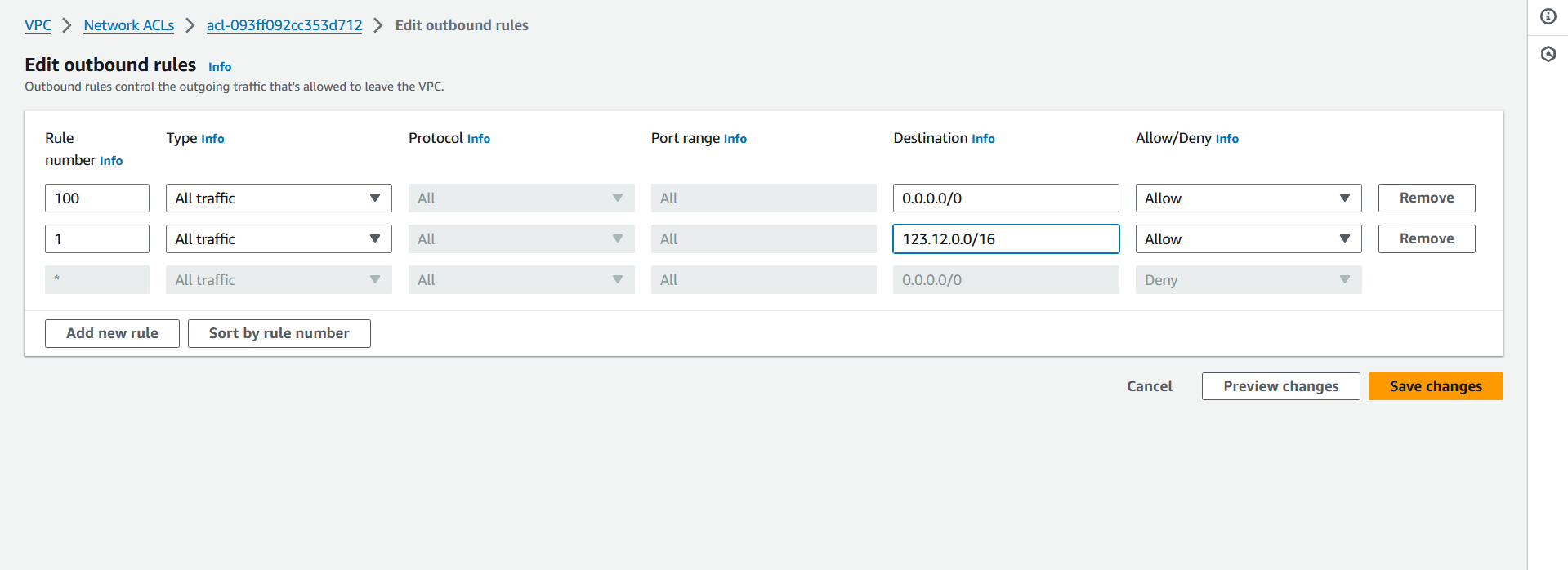
1. **Outbound Rules**:

* In the security groups associated with the instances in the subnets of VPC\_1:
* Allow outbound traffic to the CIDR block of VPC\_2 on the required ports and protocols. This will enable instances in VPC\_1 to communicate with instances in VPC\_2.





* In the network ACLs associated with the subnets in VPC\_1:
* Allow outbound traffic to the CIDR block of VPC\_2 on the required ports and protocols.

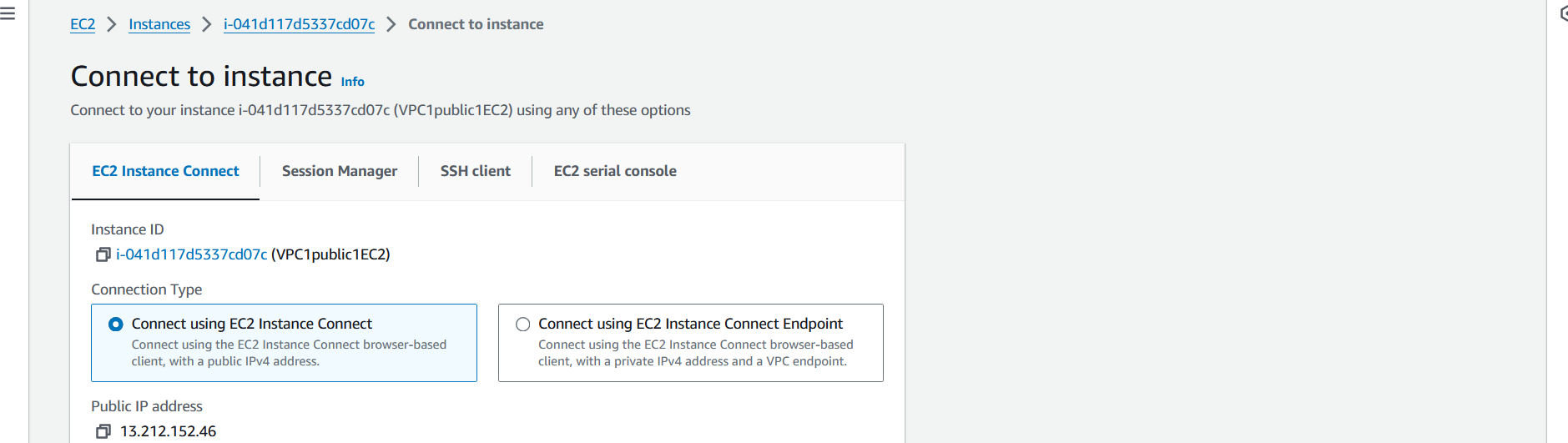
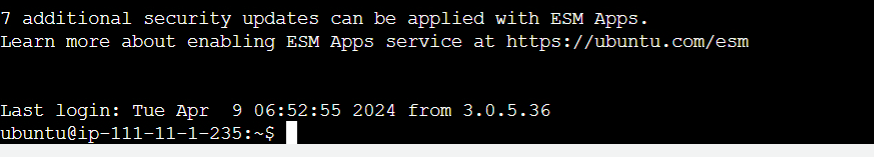


1. **Testing Connectivity:**

* Once the configuration is done, test connectivity between subnets in VPC\_1 and VPC\_2.

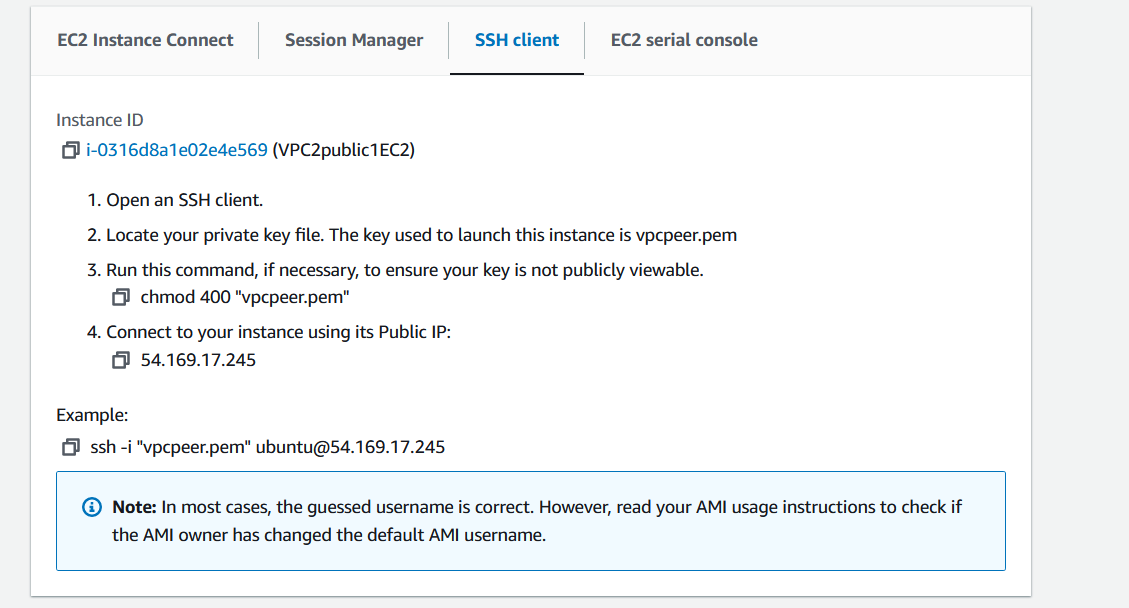
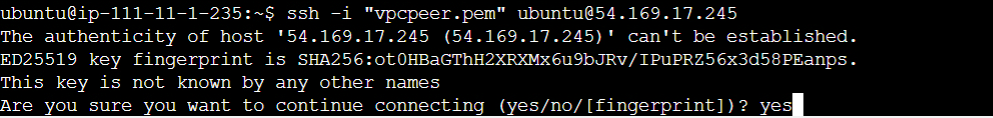
Acces the instances in VPC2 from instances in VPC1.

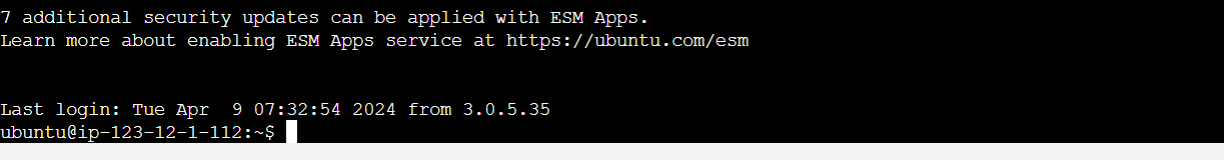
1. ssh into instance in public subnet of VPC1.

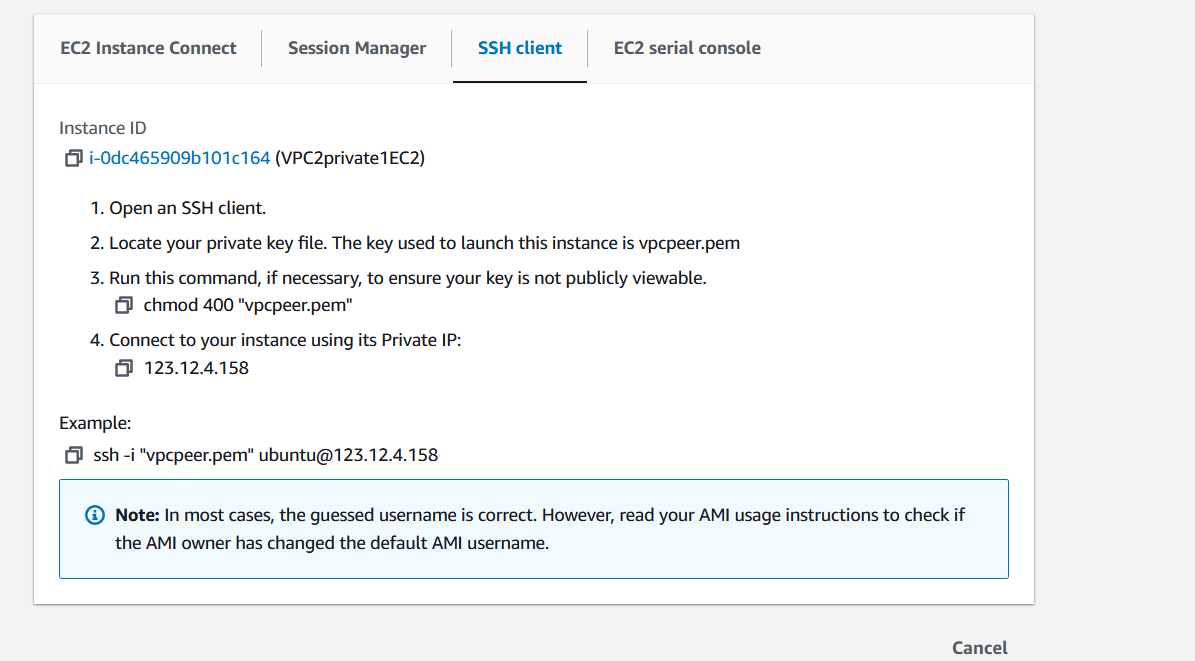
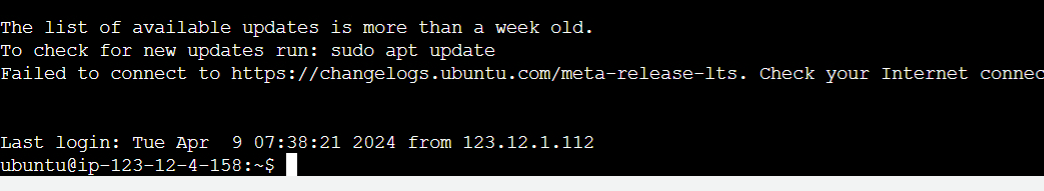
1. Connect to the instances in VPC2

* Access to public instance in VPC2 via public instance in VPC2

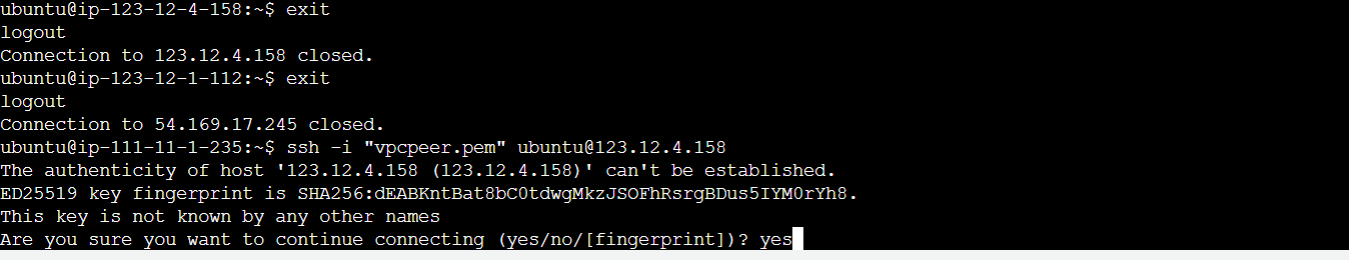
 

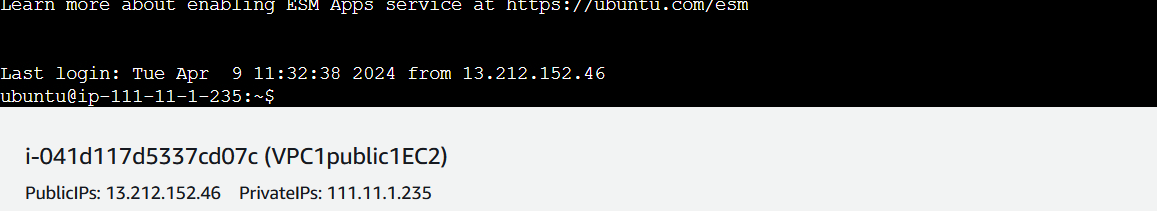


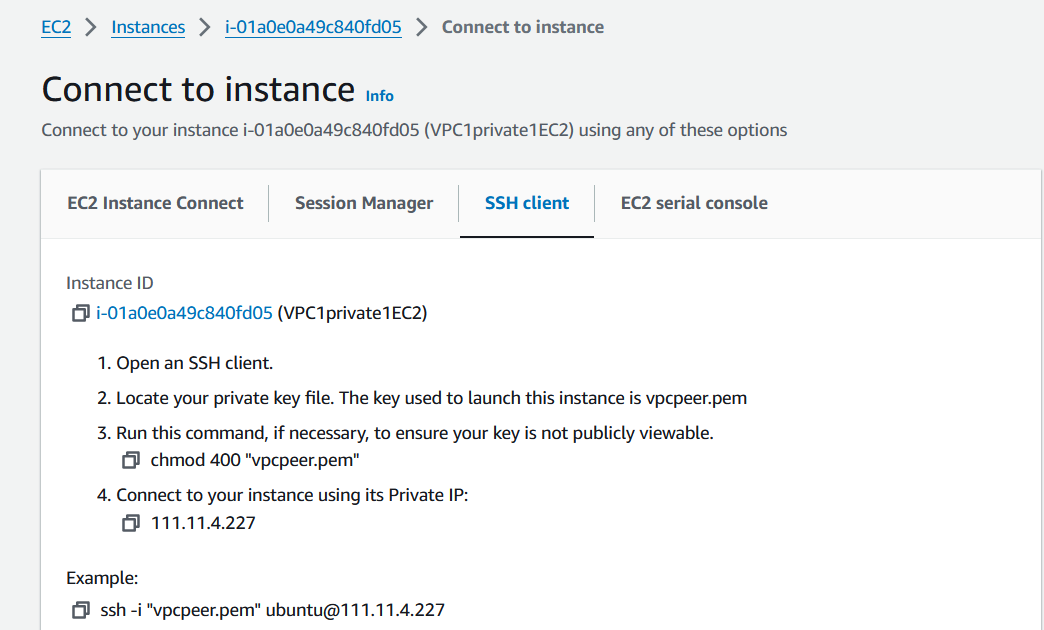
* Indirect access to private instance in VPC2 via public instance in VPC2

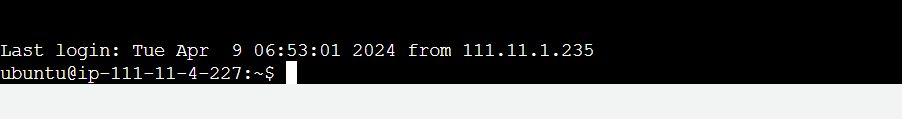
 

* Direct acces to instance in private subnet in VPC2.





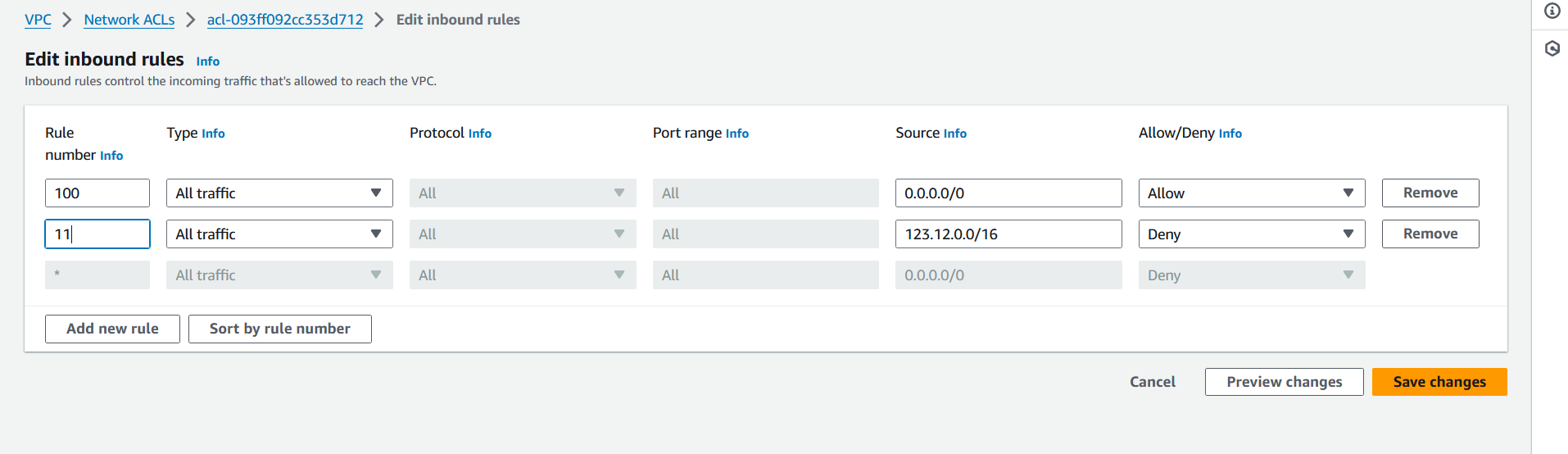




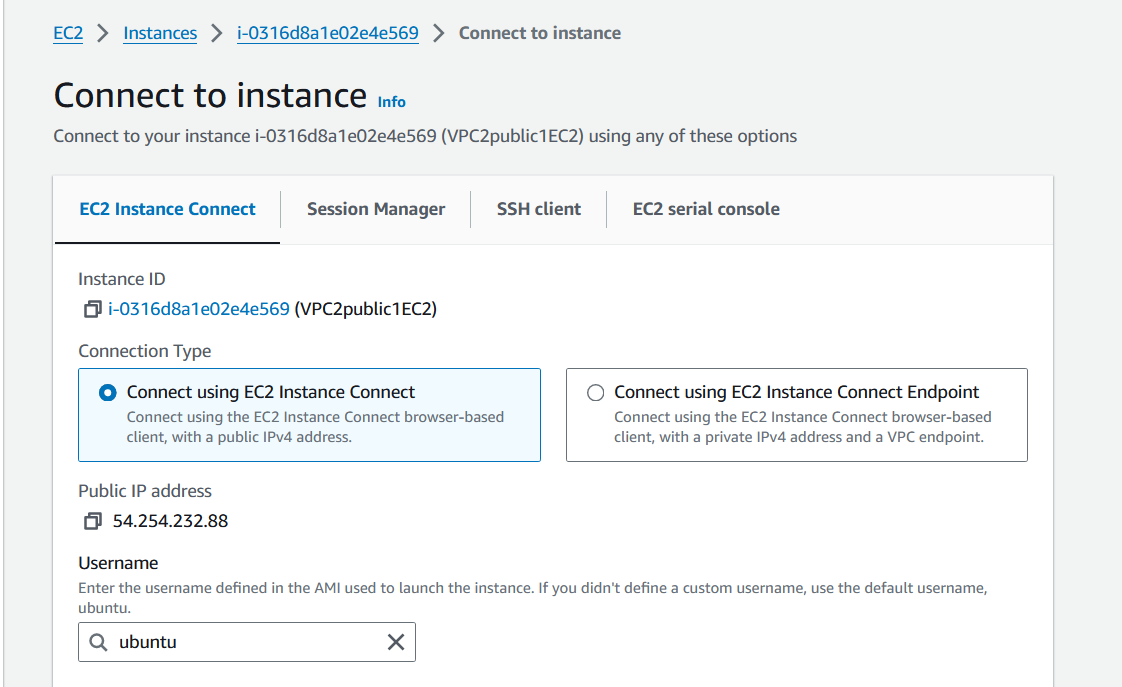
1. **Restrict access to VPC1 from VPC2**

We can utilise the operations of ACL to restrict the access of VPC1 subnets by VPC2 resources.

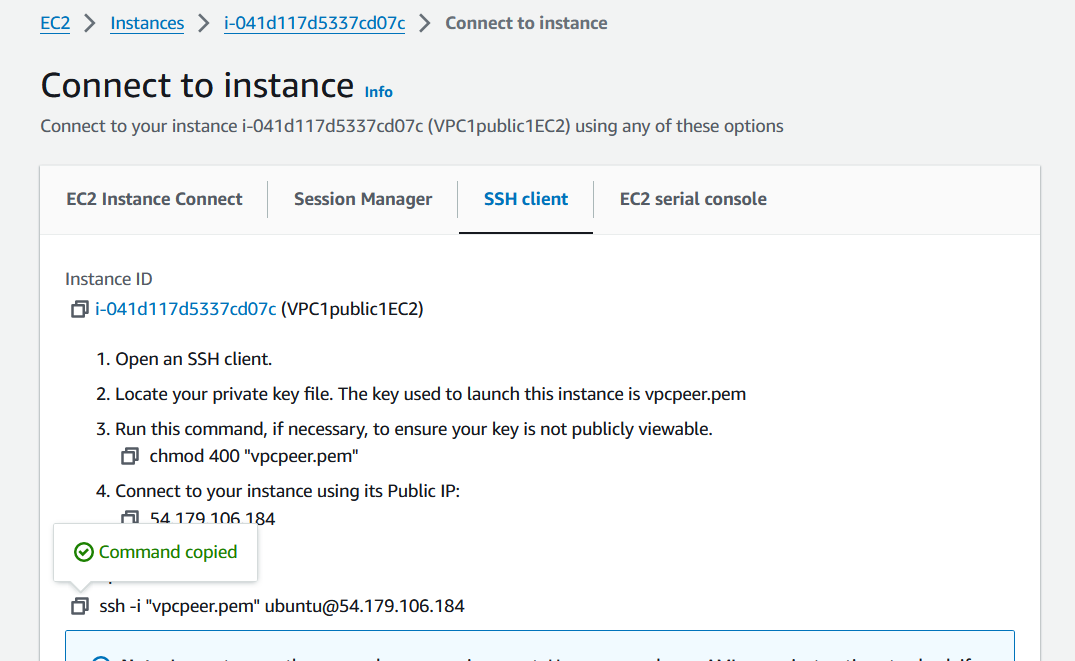
* The ACL of VPC1 is edited to deny the inbound traffic from VPC2.

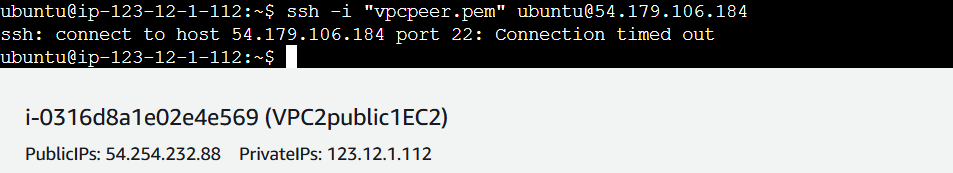


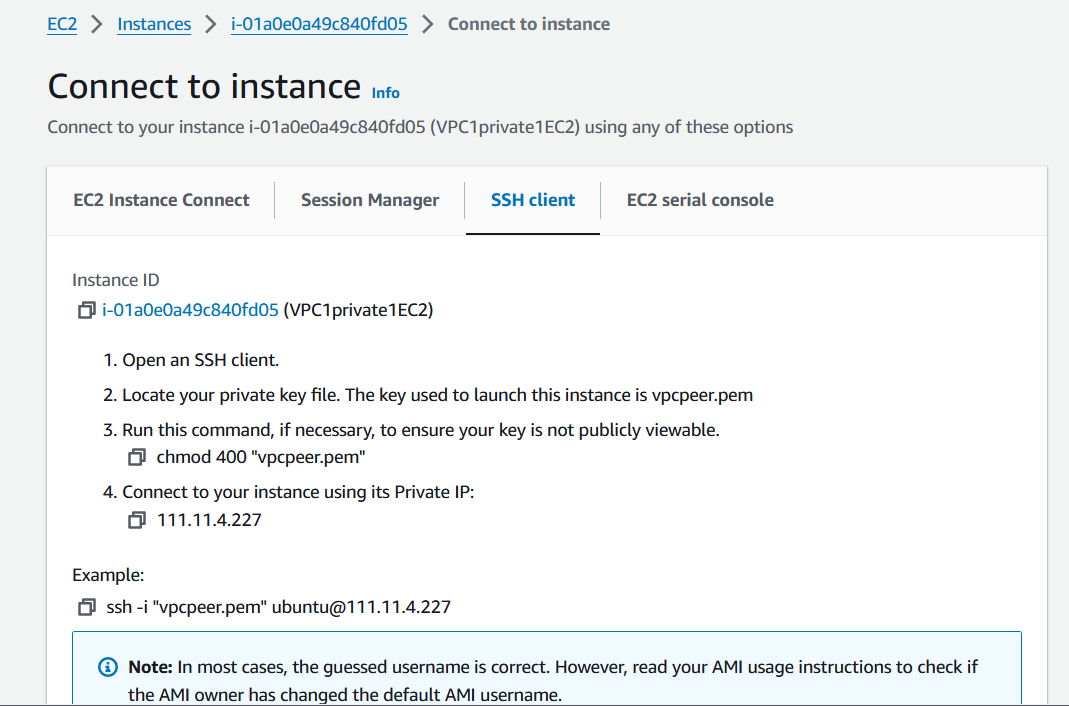
* ssh into instance in public subnet of VPC2.

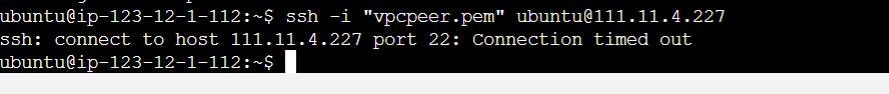


* Try connecting to instance in public subnet of VPC1. We can observe that VPC1 is not accessed from VPC2.









**Setting up the environment for VPC peering using cloudformation.**

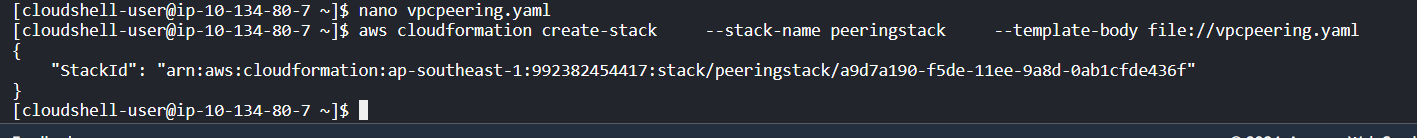
Below is a CloudFormation template for creating the VPC peering connection between VPC\_1 and VPC\_2 along with associated resources like subnets, route tables, and security groups.

|  |
| --- |
| AWSTemplateFormatVersion: '2010-09-09'  Description: CloudFormation template for creating VPC1 and VPC2 with specified subnets, security groups, route tables, and internet gateway  Parameters:  VpcCidrBlock1:  Description: CIDR block for VPC1  Type: String  Default: "100.10.0.0/16"  PublicSubnet1CidrBlock1:  Description: CIDR block for public subnet 1 in VPC1  Type: String  Default: "100.10.1.0/24"  PublicSubnet2CidrBlock1:  Description: CIDR block for public subnet 2 in VPC1  Type: String  Default: "100.10.2.0/24"  PrivateSubnet1CidrBlock1:  Description: CIDR block for private subnet 1 in VPC1  Type: String  Default: "100.10.4.0/24"  PrivateSubnet2CidrBlock1:  Description: CIDR block for private subnet 2 in VPC1  Type: String  Default: "100.10.5.0/24"  VpcCidrBlock2:  Description: CIDR block for VPC2  Type: String  Default: "111.11.0.0/16"  PublicSubnet1CidrBlock2:  Description: CIDR block for public subnet 1 in VPC2  Type: String  Default: "111.11.1.0/24"  PublicSubnet2CidrBlock2:  Description: CIDR block for public subnet 2 in VPC2  Type: String  Default: "111.11.2.0/24"  PrivateSubnet1CidrBlock2:  Description: CIDR block for private subnet 1 in VPC2  Type: String  Default: "111.11.4.0/24"  PrivateSubnet2CidrBlock2:  Description: CIDR block for private subnet 2 in VPC2  Type: String  Default: "111.11.5.0/24"  Resources:  VPC1:  Type: AWS::EC2::VPC  Properties:  CidrBlock: !Ref VpcCidrBlock1  EnableDnsSupport: true  EnableDnsHostnames: true  PublicSubnet1VPC1:  Type: AWS::EC2::Subnet  Properties:  VpcId: !Ref VPC1  CidrBlock: !Ref PublicSubnet1CidrBlock1  MapPublicIpOnLaunch: true  PublicSubnet2VPC1:  Type: AWS::EC2::Subnet  Properties:  VpcId: !Ref VPC1  CidrBlock: !Ref PublicSubnet2CidrBlock1  MapPublicIpOnLaunch: true  PrivateSubnet1VPC1:  Type: AWS::EC2::Subnet  Properties:  VpcId: !Ref VPC1  CidrBlock: !Ref PrivateSubnet1CidrBlock1  PrivateSubnet2VPC1:  Type: AWS::EC2::Subnet  Properties:  VpcId: !Ref VPC1  CidrBlock: !Ref PrivateSubnet2CidrBlock1  InternetGatewayVPC1:  Type: AWS::EC2::InternetGateway  AttachGatewayVPC1:  Type: AWS::EC2::VPCGatewayAttachment  Properties:  VpcId: !Ref VPC1  InternetGatewayId: !Ref InternetGatewayVPC1  PublicRouteTableVPC1:  Type: AWS::EC2::RouteTable  Properties:  VpcId: !Ref VPC1  PublicRoute1VPC1:  Type: AWS::EC2::Route  Properties:  RouteTableId: !Ref PublicRouteTableVPC1  DestinationCidrBlock: "0.0.0.0/0"  GatewayId: !Ref InternetGatewayVPC1  DependsOn: AttachGatewayVPC1  SecurityGroup1VPC1:  Type: AWS::EC2::SecurityGroup  Properties:  GroupDescription: Security group for VPC1  VpcId: !Ref VPC1  SecurityGroup2VPC1:  Type: AWS::EC2::SecurityGroup  Properties:  GroupDescription: Security group for VPC1  VpcId: !Ref VPC1  VPC2:  Type: AWS::EC2::VPC  Properties:  CidrBlock: !Ref VpcCidrBlock2  EnableDnsSupport: true  EnableDnsHostnames: true  PublicSubnet1VPC2:  Type: AWS::EC2::Subnet  Properties:  VpcId: !Ref VPC2  CidrBlock: !Ref PublicSubnet1CidrBlock2  MapPublicIpOnLaunch: true  PublicSubnet2VPC2:  Type: AWS::EC2::Subnet  Properties:  VpcId: !Ref VPC2  CidrBlock: !Ref PublicSubnet2CidrBlock2  MapPublicIpOnLaunch: true  PrivateSubnet1VPC2:  Type: AWS::EC2::Subnet  Properties:  VpcId: !Ref VPC2  CidrBlock: !Ref PrivateSubnet1CidrBlock2  PrivateSubnet2VPC2:  Type: AWS::EC2::Subnet  Properties:  VpcId: !Ref VPC2  CidrBlock: !Ref PrivateSubnet2CidrBlock2  InternetGatewayVPC2:  Type: AWS::EC2::InternetGateway  AttachGatewayVPC2:  Type: AWS::EC2::VPCGatewayAttachment  Properties:  VpcId: !Ref VPC2  InternetGatewayId: !Ref InternetGatewayVPC2  PublicRouteTableVPC2:  Type: AWS::EC2::RouteTable  Properties:  VpcId: !Ref VPC2  PublicRoute1VPC2:  Type: AWS::EC2::Route  Properties:  RouteTableId: !Ref PublicRouteTableVPC2  DestinationCidrBlock: "0.0.0.0/0"  GatewayId: !Ref InternetGatewayVPC2  DependsOn: AttachGatewayVPC2  SecurityGroup1VPC2:  Type: AWS::EC2::SecurityGroup  Properties:  GroupDescription: Security group for VPC2  VpcId: !Ref VPC2  SecurityGroup2VPC2:  Type: AWS::EC2::SecurityGroup  Properties:  GroupDescription: Security group for VPC2  VpcId: !Ref VPC2  Outputs:  Vpc1Id:  Description: VPC1 ID  Value: !Ref VPC1  PublicSubnet1IdVPC1:  Description: Public Subnet 1 ID in VPC1  Value: !Ref PublicSubnet1VPC1  PublicSubnet2IdVPC1:  Description: Public Subnet 2 ID in VPC1  Value: !Ref PublicSubnet2VPC1  PrivateSubnet1IdVPC1:  Description: Private Subnet 1 ID in VPC1  Value: !Ref PrivateSubnet1VPC1  PrivateSubnet2IdVPC1:  Description: Private Subnet 2 ID in VPC1  Value: !Ref PrivateSubnet2VPC1  Vpc2Id:  Description: VPC2 ID  Value: !Ref VPC2  PublicSubnet1IdVPC2:  Description: Public Subnet 1 ID in VPC2  Value: !Ref PublicSubnet1VPC2  PublicSubnet2IdVPC2:  Description: Public Subnet 2 ID in VPC2  Value: !Ref PublicSubnet2VPC2  PrivateSubnet1IdVPC2:  Description: Private Subnet 1 ID in VPC2  Value: !Ref PrivateSubnet1VPC2  PrivateSubnet2IdVPC2:  Description: Private Subnet 2 ID in VPC2  Value: !Ref PrivateSubnet2VPC2 |

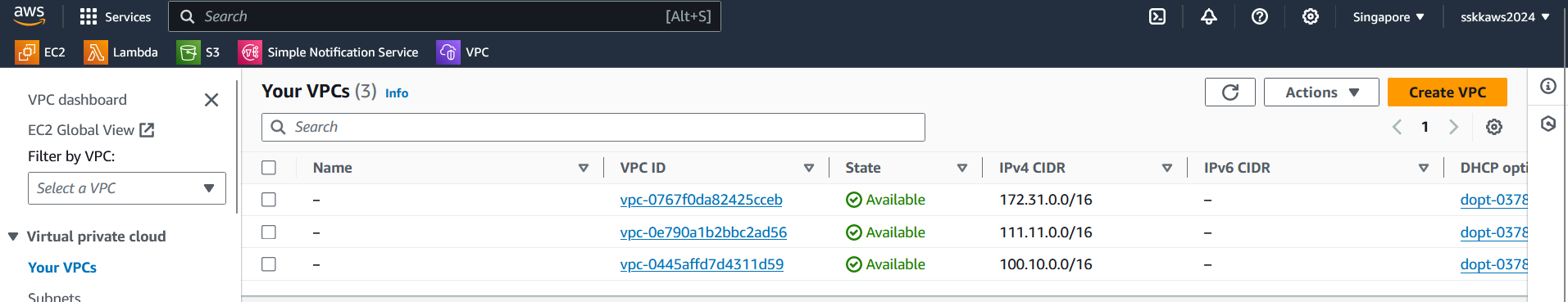
Once the template is ready, use cloudshell to place the code as a yaml file in the root directory. Upload the userdata.sh script the cloudshell, with which we can setup apache2 for the instances.

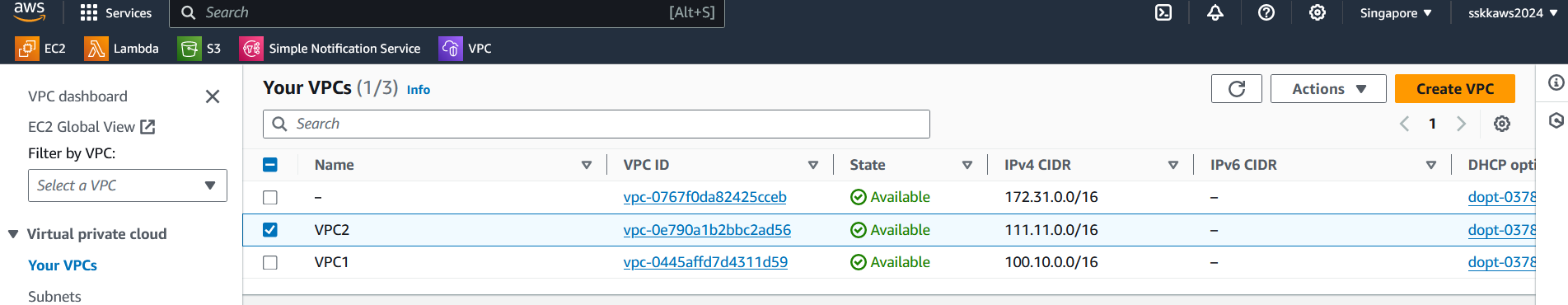
|  |
| --- |
| #!/bin/bash  apt-get update -y  apt-get install -y apache2  systemctl start apache2  systemctl enable apache2 |

Use create -stack command to initiate the stack formation from the stored yaml file.



Verify the VPC resources and create instances before proceeding with peering





Create one ec2 instance in the public subnet and ec2 instance in private subnet in each of the VPCs. So the final test environment would be 2 VPCs with 1 public subnet and 1 private subnet, with a total of 4 ec2 instances. Now the instances in VPC1 is to be setup to communicate with ec2 instances in VPC2.

To manually set up VPC peering between VPC1 and VPC2 in the AWS Management Console, follow these steps:

1. Create VPC Peering Connection
2. Configure Peering Connection
3. Create Peering Connection.
4. Accept Peering Connection
5. Update Route Tables

To update the route tables for both VPCs after the peering connection is accepted, follow these steps:

1. Go to the VPC Dashboard
2. Select Route Tables
3. Update Route Table for VPC1
4. Update Route Table for VPC2
5. Save Changes

By adding these routes, you instruct the VPC's routing tables to send traffic destined for the CIDR blocks of the remote VPCs through the peering connections. This allows communication between instances in the two VPCs via the peering connection.

1. Review Security Group Rules

To review the security group rules in both VPCs and ensure that resources in VPC1 cannot be accessed by VPC2, follow these steps:

1. Go to the VPC Dashboard
2. Select Security Groups
3. Review Security Groups for VPC1
4. Review Security Groups for VPC2
5. Save Changes

We can now manually set up a VPC peering connection between VPC1 and VPC2 and ensured that all subnets in VPC2 are accessible from VPC1 while restricting access from VPC2 to VPC1.

**Setting up the environment and VPC peering using cloudformation.**

Instead of setting up the vpc peering manually, we can automate the peering process also using cloudformation as given below.

|  |
| --- |
| AWSTemplateFormatVersion: '2010-09-09'  Description: CloudFormation template for creating VPC1 and VPC2 with public and private subnets, security groups, route tables, and internet gateway  Parameters:  Vpc1CIDR:  Description: CIDR block for VPC1  Type: String  Default: "111.11.0.0/16"  Vpc2CIDR:  Description: CIDR block for VPC2  Type: String  Default: "123.12.0.0/16"  Resources:  VPC1:  Type: AWS::EC2::VPC  Properties:  CidrBlock: !Ref Vpc1CIDR  EnableDnsSupport: true  EnableDnsHostnames: true  VPC2:  Type: AWS::EC2::VPC  Properties:  CidrBlock: !Ref Vpc2CIDR  EnableDnsSupport: true  EnableDnsHostnames: true  InternetGatewayVPC1:  Type: AWS::EC2::InternetGateway  InternetGatewayAttachmentVPC1:  Type: AWS::EC2::VPCGatewayAttachment  Properties:  VpcId: !Ref VPC1  InternetGatewayId: !Ref InternetGatewayVPC1  PublicRouteTableVPC1:  Type: AWS::EC2::RouteTable  Properties:  VpcId: !Ref VPC1  DefaultPublicRouteVPC1:  Type: AWS::EC2::Route  DependsOn: InternetGatewayAttachmentVPC1  Properties:  RouteTableId: !Ref PublicRouteTableVPC1  DestinationCidrBlock: "0.0.0.0/0"  GatewayId: !Ref InternetGatewayVPC1  PublicSubnet1VPC1:  Type: AWS::EC2::Subnet  Properties:  VpcId: !Ref VPC1  CidrBlock: "111.11.0.0/24"  AvailabilityZone: !Select [0, !GetAZs ""]  MapPublicIpOnLaunch: true  PublicSubnet1RouteTableAssociationVPC1:  Type: AWS::EC2::SubnetRouteTableAssociation  Properties:  SubnetId: !Ref PublicSubnet1VPC1  RouteTableId: !Ref PublicRouteTableVPC1  PrivateRouteTableVPC1:  Type: AWS::EC2::RouteTable  Properties:  VpcId: !Ref VPC1  PrivateSubnet1VPC1:  Type: AWS::EC2::Subnet  Properties:  VpcId: !Ref VPC1  CidrBlock: "111.11.1.0/24"  AvailabilityZone: !Select [0, !GetAZs ""]  PrivateSubnet1RouteTableAssociationVPC1:  Type: AWS::EC2::SubnetRouteTableAssociation  Properties:  SubnetId: !Ref PrivateSubnet1VPC1  RouteTableId: !Ref PrivateRouteTableVPC1  SecurityGroupWebServerVPC1:  Type: AWS::EC2::SecurityGroup  Properties:  GroupDescription: Allow HTTP and SSH inbound traffic  VpcId: !Ref VPC1  SecurityGroupIngress:  - IpProtocol: tcp  FromPort: 22  ToPort: 22  CidrIp: 0.0.0.0/0  - IpProtocol: tcp  FromPort: 80  ToPort: 80  CidrIp: 0.0.0.0/0  VPCPeeringConnection:  Type: AWS::EC2::VPCPeeringConnection  Properties:  VpcId: !Ref VPC1  PeerVpcId: !Ref VPC2  PeerRegion: !Ref "AWS::Region"  PeerOwnerId: !GetAtt VPC2.OwnerId  Tags:  - Key: Name  Value: VPC1-to-VPC2-Peering  VPCPeeringConnectionAccepter:  Type: AWS::EC2::VPCPeeringConnection  Properties:  VpcId: !Ref VPC2  PeerVpcId: !Ref VPC1  PeerRegion: !Ref "AWS::Region"  PeerOwnerId: !GetAtt VPC1.OwnerId  Tags:  - Key: Name  Value: VPC2-to-VPC1-Peering  PrivateRouteTableVPC2:  Type: AWS::EC2::RouteTable  Properties:  VpcId: !Ref VPC2  PrivateRouteToPeerVPC1:  Type: AWS::EC2::Route  Properties:  RouteTableId: !Ref PrivateRouteTableVPC2  DestinationCidrBlock: !Ref Vpc1CIDR  VpcPeeringConnectionId: !Ref VPCPeeringConnection  Outputs:  VPC1ID:  Description: ID of VPC1  Value: !Ref VPC1  VPC2ID:  Description: ID of VPC2  Value: !Ref VPC2  PublicSubnet1VPC1ID:  Description: ID of the public subnet in VPC1  Value: !Ref PublicSubnet1VPC1  PrivateSubnet1VPC1ID:  Description: ID of the private subnet in VPC1  Value: !Ref PrivateSubnet1VPC1  InternetGatewayVPC1ID:  Description: ID of the internet gateway in VPC1  Value: !Ref InternetGatewayVPC1  SecurityGroupWebServerVPC1ID:  Description: ID of the security group for web servers in VPC1  Value: !Ref SecurityGroupWebServerVPC1 |

This template creates:

* An internet gateway and attaches it to VPC1.
* Public and private route tables for VPC1, associating the public route table with the public subnet and the private route table with the private subnet.
* A security group allowing HTTP and SSH inbound traffic for web servers in VPC1.
* A route table (PrivateRouteTableVPC2) is created for VPC2.
* A route (PrivateRouteToPeerVPC1) is added to this route table, pointing to the VPC peering connection with VPC1. This allows traffic from VPC2 to VPC1.
* However, the route table of VPC1 doesn't have a route to the CIDR block of VPC2. By default, VPC peering allows traffic in both directions unless explicitly restricted. Therefore, there is no need to modify the route table of VPC1 to restrict traffic from VPC1 to VPC2.