**Set up an environment to experiment with VPC peering.**

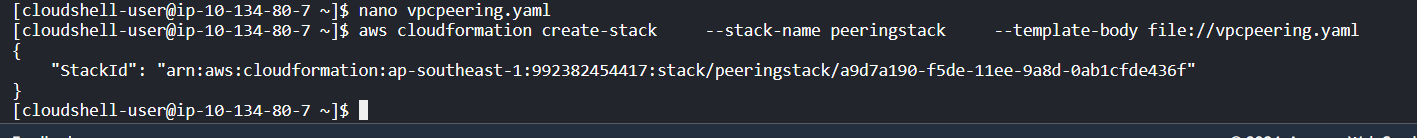
Below is a CloudFormation template for creating the VPC peering connection between VPC\_1 and VPC\_2 along with associated resources like subnets, route tables, and security groups.

|  |
| --- |
| 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 |

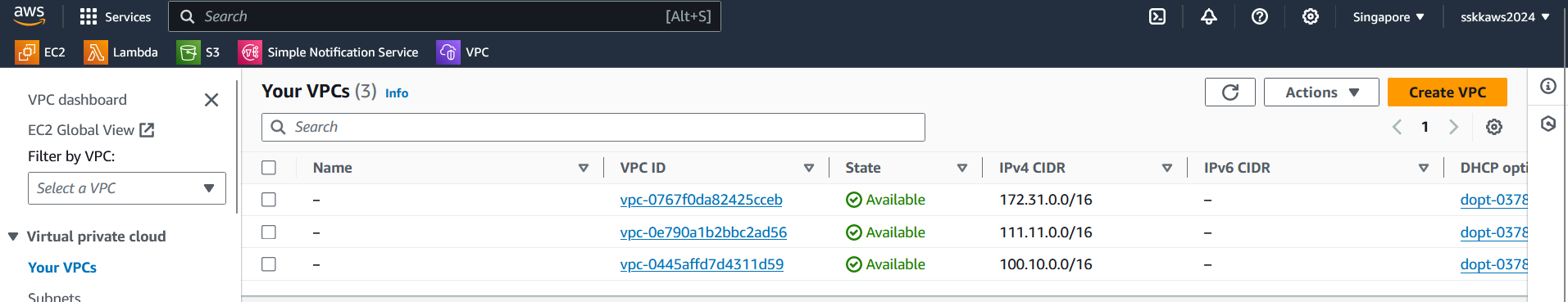
Once the template is ready, use cloudshell to place the code as a yaml file in the root directory. Upload the userdata.sh script the cloudshell, with which we can setup apache2 for the instances.

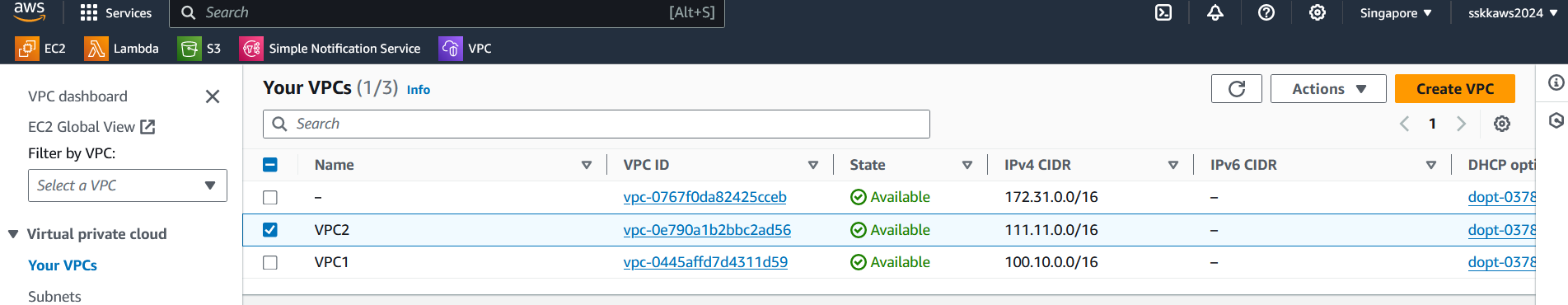
|  |
| --- |
| #!/bin/bash  apt-get update -y  apt-get install -y apache2  systemctl start apache2  systemctl enable apache2 |

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:** In the VPC dashboard, select "Peering Connections" from the left-hand menu, and then click on the "Create Peering Connection" button.
2. **Configure Peering Connection:**
   * Give your peering connection a name and select the VPC1 as the requester VPC and VPC2 as the accepter VPC.
   * Ensure that "Allow all VPCs to communicate with each other" is unchecked to restrict access from VPC2 to VPC1.
3. **Create Peering Connection:** Click on the "Create Peering Connection" button.
4. **Accept Peering Connection:** After creating the peering connection, it will appear in the list of peering connections with a status of "Pending Acceptance". Select the peering connection, and then click on the "Actions" dropdown and choose "Accept Peering Connection". This will send a request to the owner of the accepter VPC (in this case, VPC2) to accept the peering connection.
5. **Update Route Tables:** Once the peering connection is accepted, go to the route tables of both VPCs and update them to route traffic destined for the CIDR blocks of the remote VPCs through the peering connections. For VPC1, add a route for VPC2's CIDR block pointing to the peering connection, and vice versa for VPC2.

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

1. **Go to the VPC Dashboard:** Open the AWS Management Console, navigate to the VPC dashboard by selecting "Services" and then "VPC" under "Networking & Content Delivery".
2. **Select Route Tables:** In the VPC dashboard, select "Route Tables" from the left-hand menu.
3. **Update Route Table for VPC1:**
   * Find the route table associated with the subnets in VPC1.
   * Edit the route table and add a new route with the destination CIDR block of VPC2 pointing to the peering connection.
   * The peering connection ID can be found in the peering connection details or from the list of available resources.
4. **Update Route Table for VPC2:**
   * Similarly, find the route table associated with the subnets in VPC2.
   * Edit the route table and add a new route with the destination CIDR block of VPC1 pointing to the peering connection.
5. **Save Changes:** After adding the routes, save the changes to the route tables.

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:** Review the security group rules in both VPCs to ensure that the resources in VPC1 cannot be accessed by VPC2. You can do this by checking the inbound rules of security groups attached to resources in VPC1 and ensuring that they do not allow traffic from the CIDR block of VPC2.

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

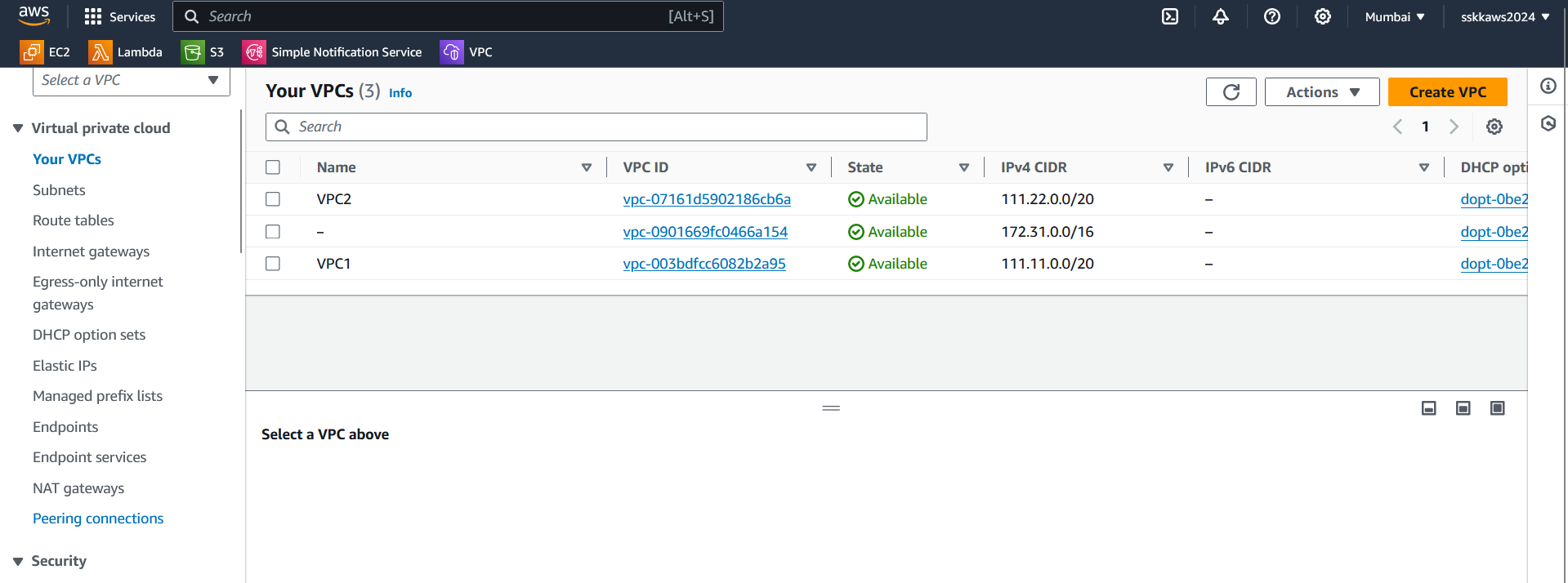
1. **Go to the VPC Dashboard:** Open the AWS Management Console, navigate to the VPC dashboard by selecting "Services" and then "VPC" under "Networking & Content Delivery".
2. **Select Security Groups:** In the VPC dashboard, select "Security Groups" from the left-hand menu.
3. **Review Security Groups for VPC1:**
   * Identify the security groups associated with resources (such as EC2 instances) in VPC1.
   * For each security group, review the inbound rules to ensure that there are no rules allowing traffic from the CIDR block of VPC2.
   * If such rules exist, remove them or modify them to restrict access as needed.
4. **Review Security Groups for VPC2:**
   * Similarly, identify the security groups associated with resources in VPC2.
   * Review the inbound rules to ensure that there are no rules allowing traffic from the CIDR block of VPC1.
   * Adjust or remove any rules that allow such access.
5. **Save Changes:** After reviewing and adjusting the security group rules, save the changes if any modifications were made.

That's it! You have 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.

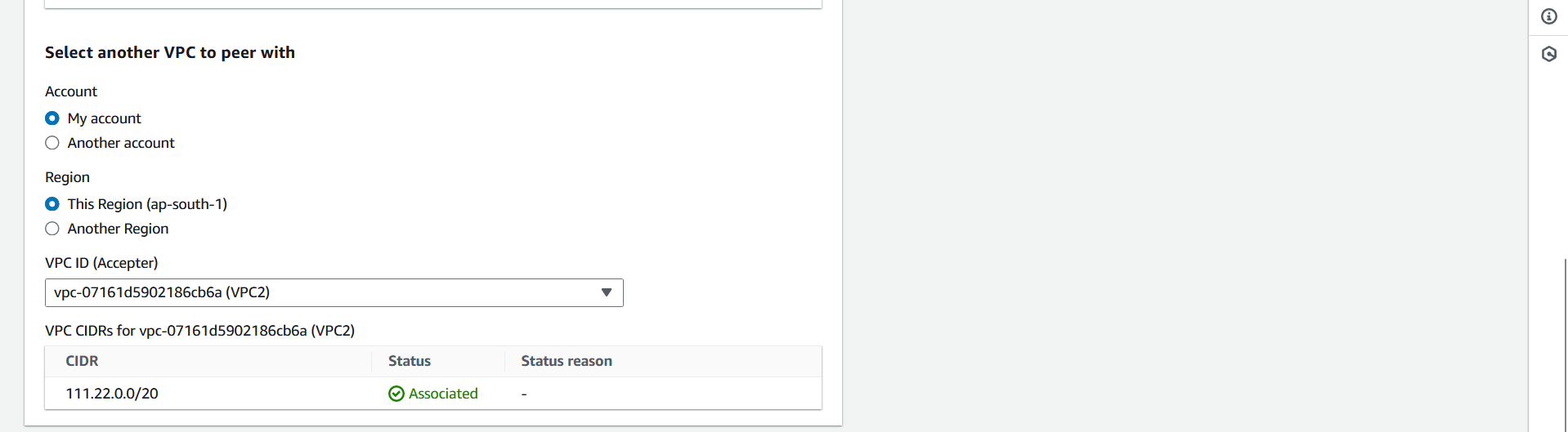
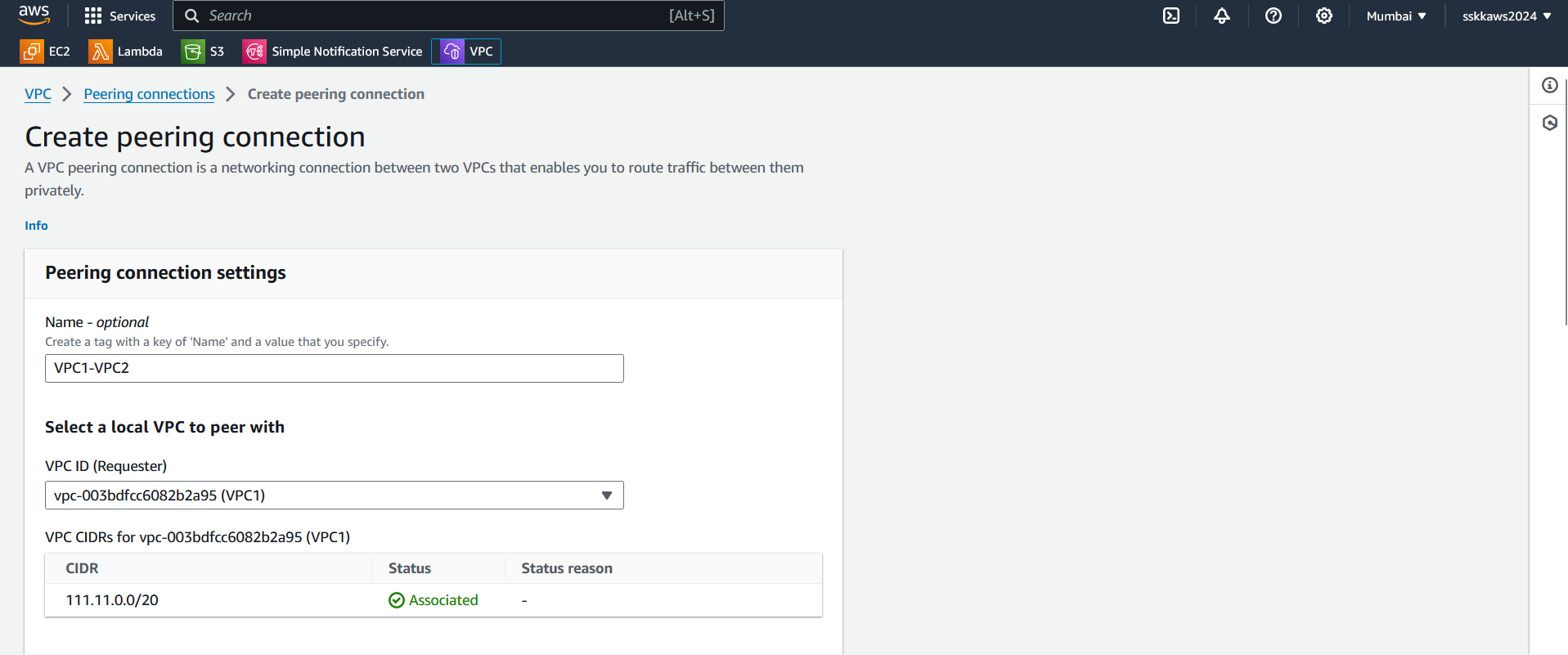
To establish a VPC peering connection between VPC1 and VPC2 and ensure that all subnets in VPC2 are accessible from VPC1 while implementing security controls, follow these steps:

**Step 1: Create a VPC Peering Connection:**

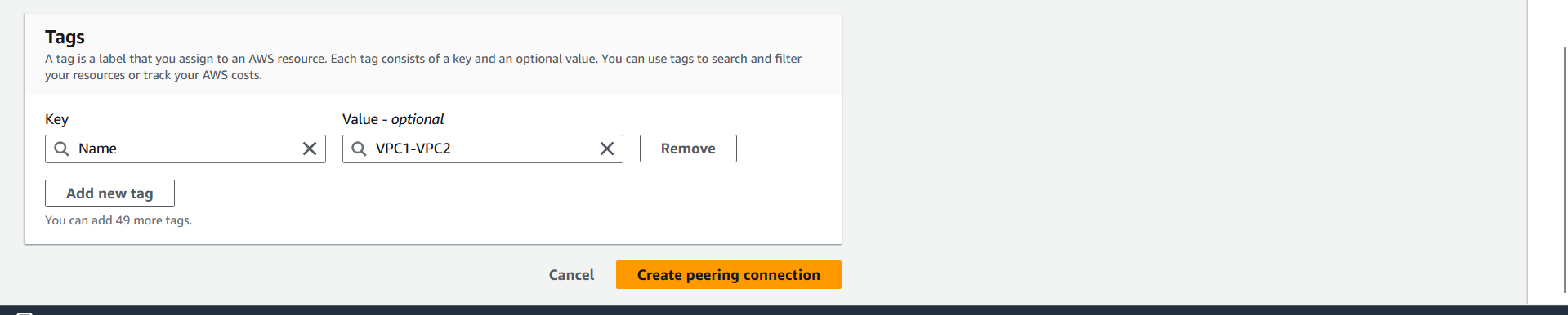
1. Go to the AWS Management Console.
2. Navigate to the VPC dashboard.
3. Select "Peering Connections" from the left-hand menu.



1. Click "Create Peering Connection."
2. Enter a name for the peering connection



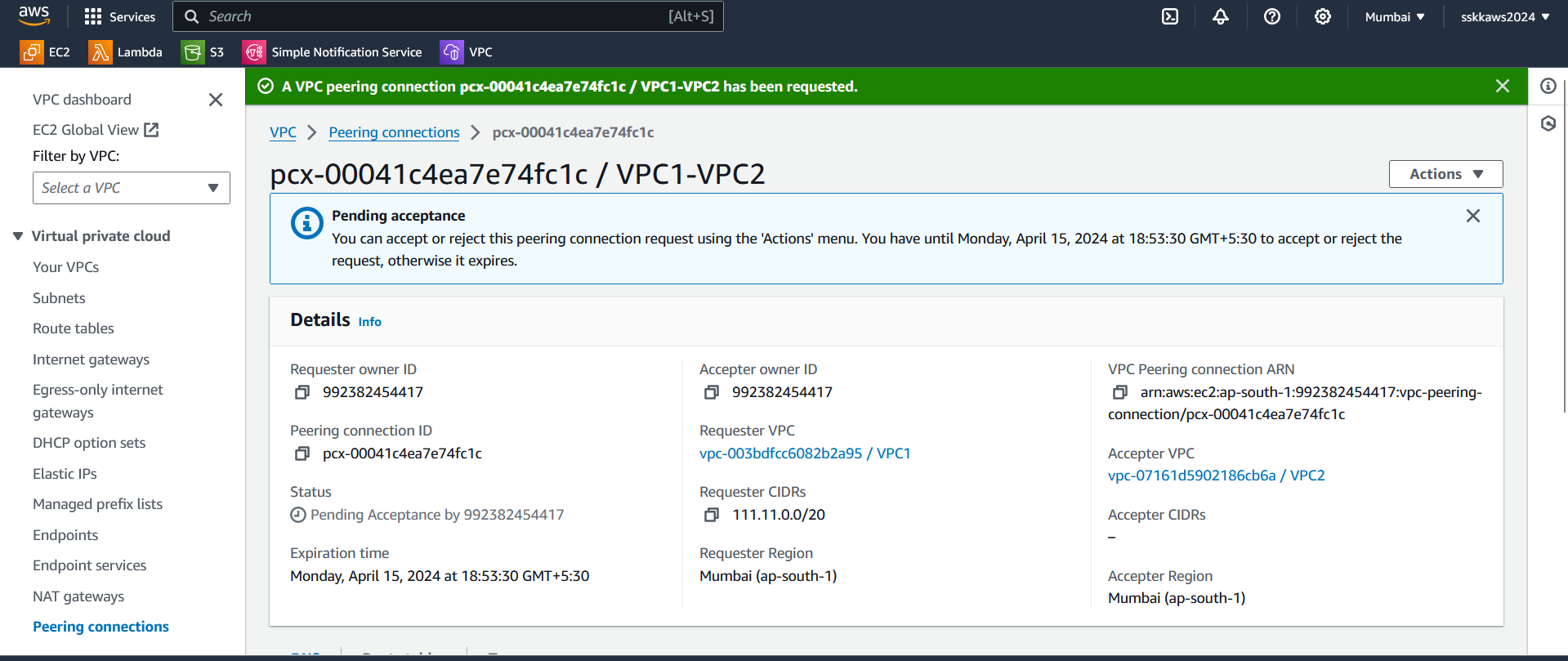
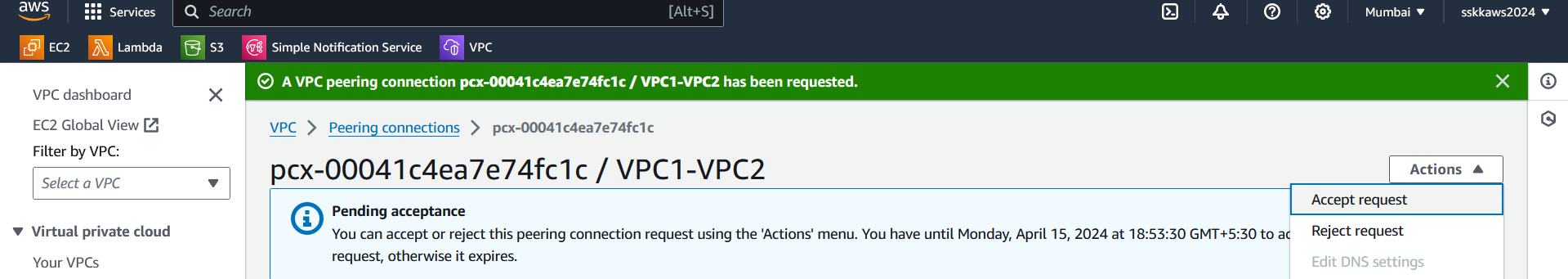
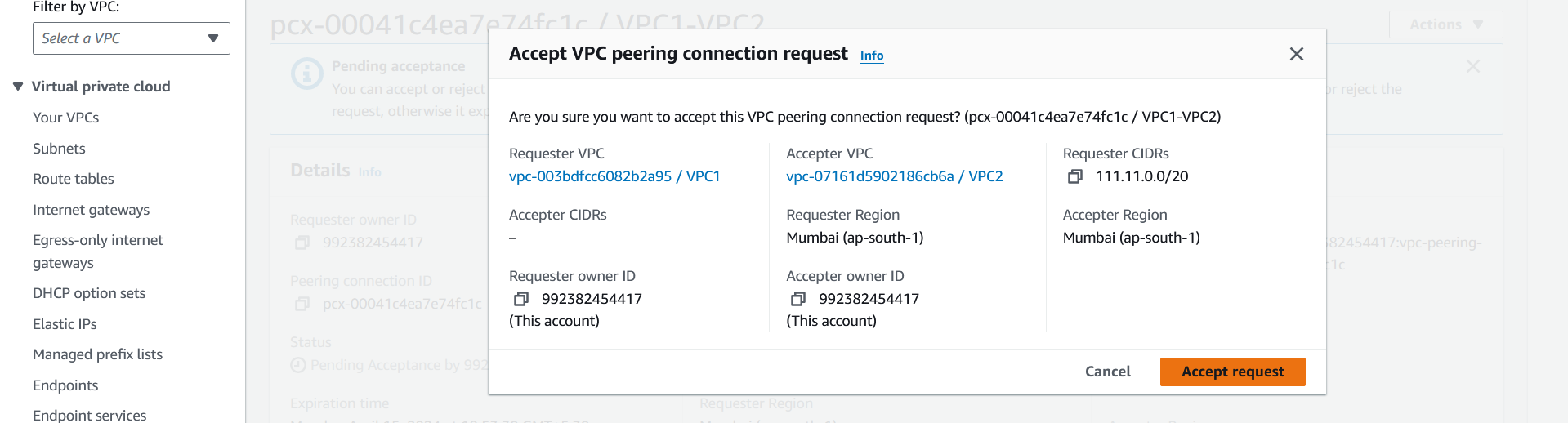
* **Requester VPC (VPC1):** This is the VPC that initiates the peering connection request. In this scenario, VPC1 has the public and private subnets and wants to connect with resources in VPC2.
* **Acceptor VPC (VPC2):** This is the VPC that receives and accepts the peering connection request. In this scenario, VPC2 also has public and private subnets and accepts the connection from VPC1.

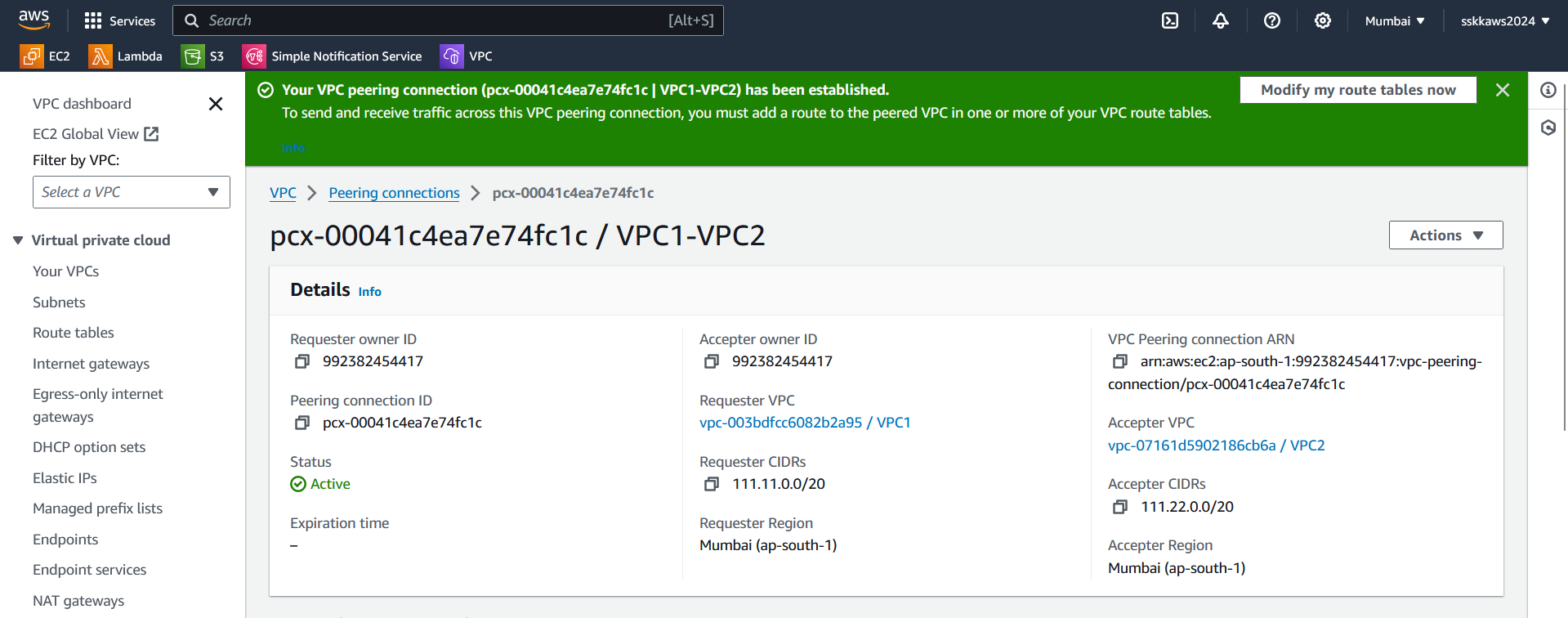


1. Click "Create Peering Connection."

**Step 2: Accept the Peering Connection:**

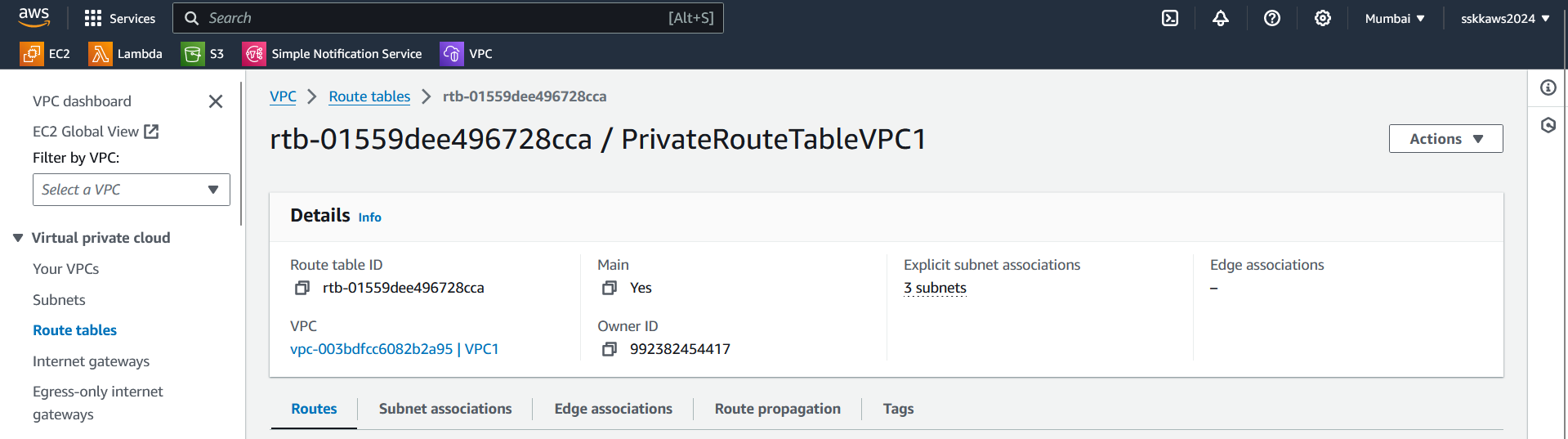
1. Find the peering connection request from VPC1.
2. Click "Actions" and then "Accept Request" to accept the peering connection.

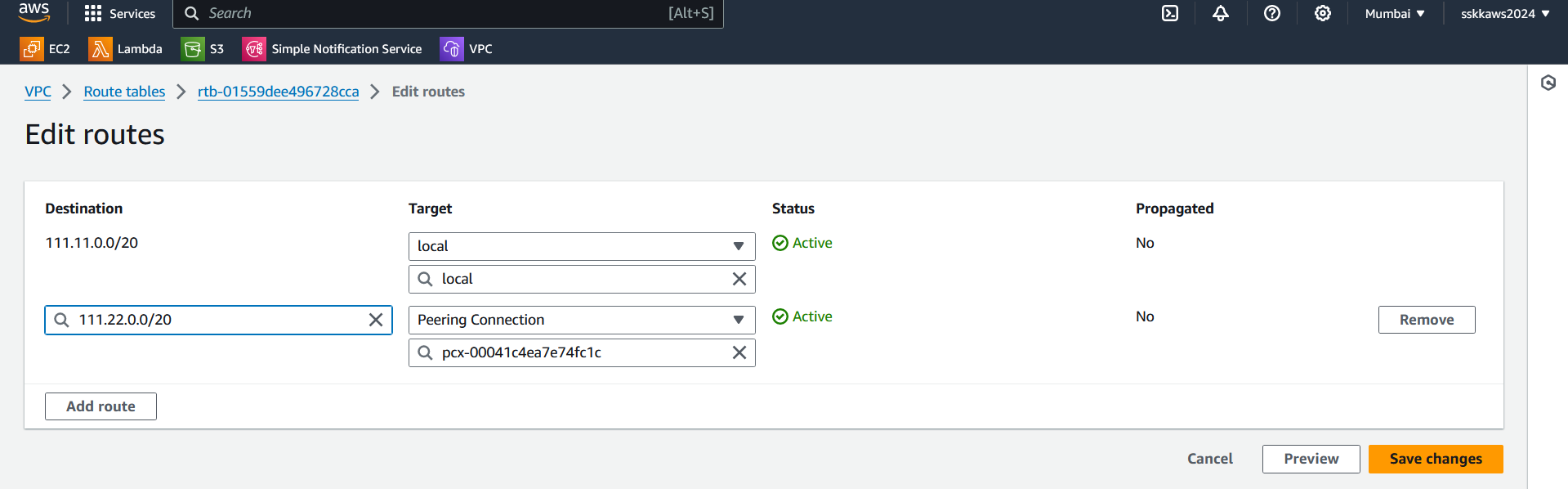


**Step 3: Update Route Tables:**

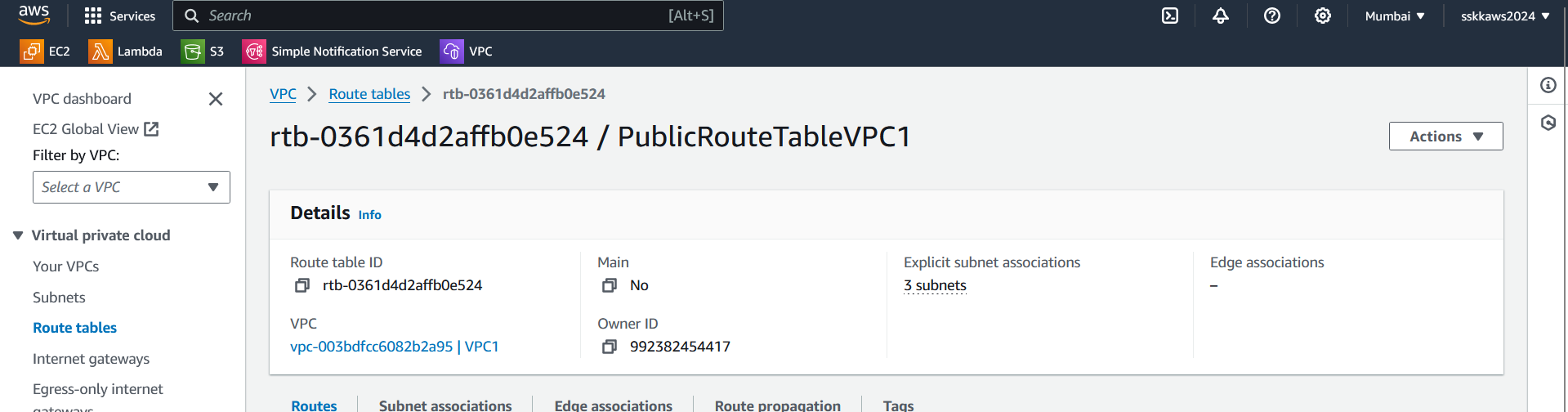
1. In the VPC dashboard for VPC1, select "Route Tables"
2. Edit the route table associated with the private subnet in VPC1.

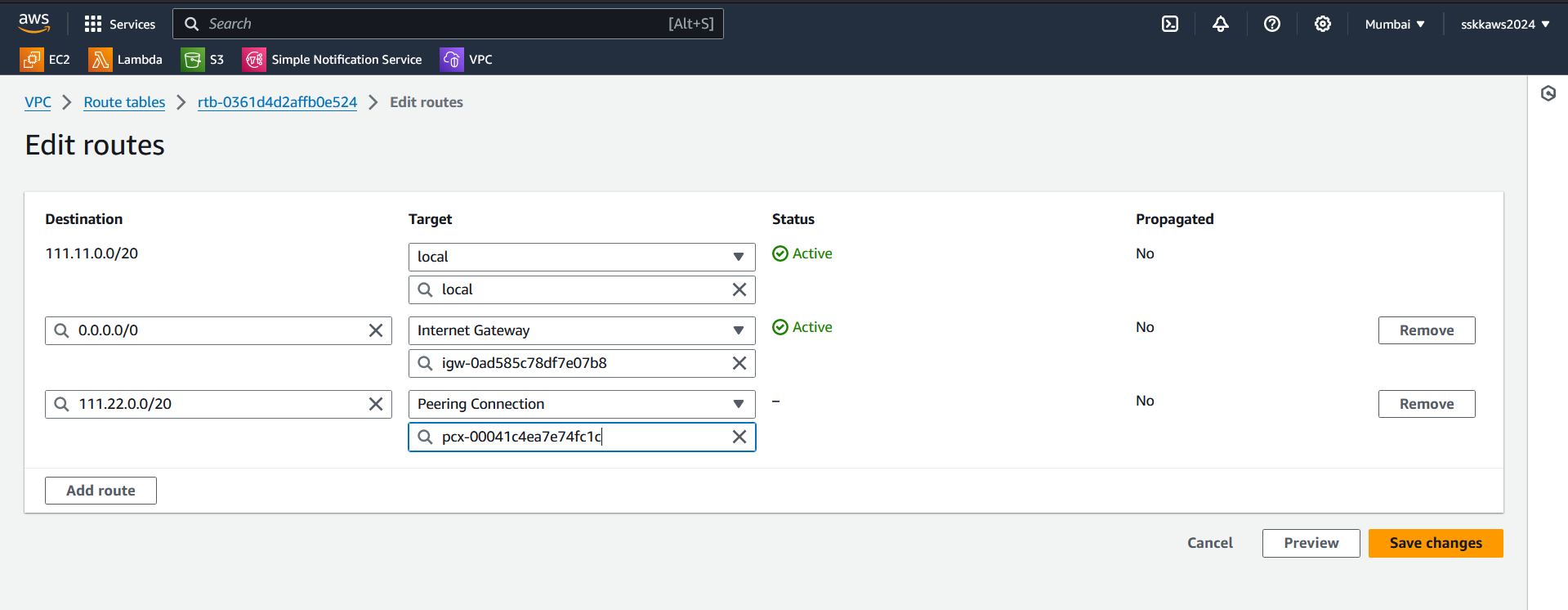


1. Add a route for the CIDR block of the private subnet in VPC2, with the peering connection as the target.



1. Repeat the same steps for the route table associated with the public subnet in VPC1.





**Step 4: Implement Security Controls:**

1. Select the security group of the PublicInstanceVPC1
2. Create or edit the security group associated with the resources in VPC1 that need access to VPC2.
3. Add inbound rules to allow traffic from the private subnet in VPC2 to the necessary ports on resources in VPC1.

**Step 5: Test Connectivity:**

1. Launch instances or resources in both VPCs.
2. Verify that resources in VPC1 can access resources in VPC2 and vice versa.
3. Ensure that access is restricted to the necessary resources only based on the implemented security controls.

**Step 6: Document the Peering Configuration:**

1. Create a document outlining the peering connection setup, including VPC IDs, subnet configurations, route table updates, and security group rules.
2. Include diagrams illustrating the network architecture and the flow of traffic between VPC1 and VPC2.
3. Document any security controls implemented and rationale behind them.

By following these steps, you'll establish a VPC peering connection between VPC1 and VPC2, ensure accessibility of all subnets in VPC2 from VPC1, and implement security controls to restrict access appropriately.