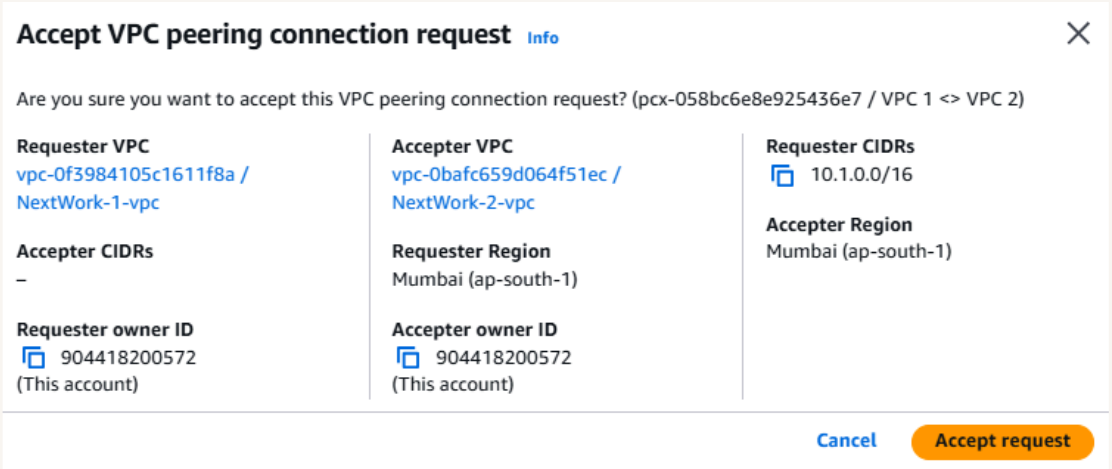
nextwork.org

# VPC Peering

RA **Ranjith D B**



RA
**Ranjith D B**
NextWork Student


# Introducing Today's Project!

## What is Amazon VPC?

Amazon VPC is a virtual network in AWS that allows you to create isolated environments for resources, control networking, and enhance security with subnets, route tables, and security groups.

## How I used Amazon VPC in this project

I used Amazon VPC to create two separate networks and establish a peering connection between them, allowing communication between EC2 instances in different VPCs.

## One thing I didn't expect in this project was...

One thing I didn't expect in this project was that ICMP traffic was blocked by default in the security group, preventing pings from working until I manually allowed it.

## This project took me...

This project took me approximately about an hour to complete, including troubleshooting connectivity issues and updating security rules.

RA
**Ranjith D B**
NextWork Student


# In the first part of my project...

## Step 1 - Set up my VPC

In this step, we are creating two VPCs from scratch using the VPC wizard. We'll leverage the visual VPC resource map to speed up the process, ensuring each VPC is properly configured for peering and future connectivity tests.

## Step 2 - Create a Peering Connection

In this step, we are creating a VPC peering connection to enable communication between our two VPCs. This allows resources in one VPC to securely talk to resources in the other without using the public internet.

## Step 3 - Update Route Tables

In this step, we are updating the route tables so that traffic from VPC 1 knows how to reach VPC 2 and vice versa. This ensures both VPCs can communicate using the newly created peering connection.

## Step 4 - Launch EC2 Instances

We're launching EC2 instances in each VPC to test the VPC peering connection. These instances will help verify if traffic can flow between the VPCs using private IPs instead of the public internet.

RA
**Ranjith D B**
NextWork Student



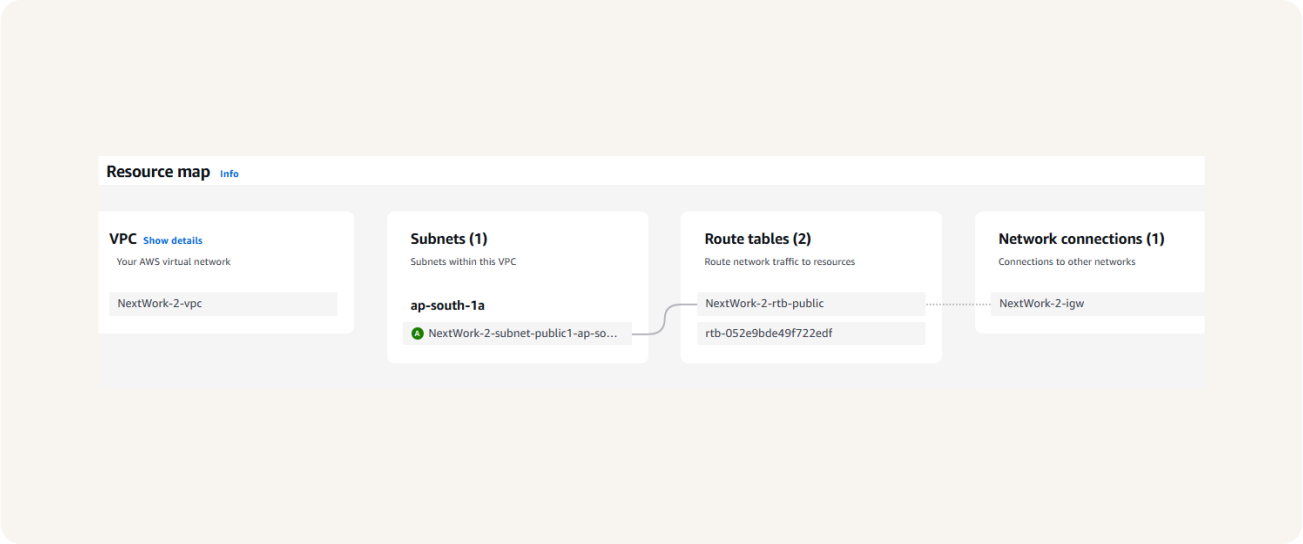
# Multi-VPC Architecture

I started my project by launching two VPCs: NextWork-1 and NextWork-2. Each VPC contains one public subnet, with no private subnets, NAT gateways, or VPC endpoints, keeping the setup simple for VPC peering.

The CIDR blocks for VPCs 1 and 2 are 10.1.0.0/16 and 10.2.0.0/16. They have to be unique because overlapping IP ranges would cause routing conflicts, making it impossible for the VPCs to communicate properly.

## I also launched 2 EC2 instances

I didn't set up key pairs for these EC2 instances as AWS EC2 Instance Connect manages key pairs for us, eliminating the need for manual key pair setup while still allowing secure SSH access when needed.



RA
**Ranjith D B**
NextWork Student

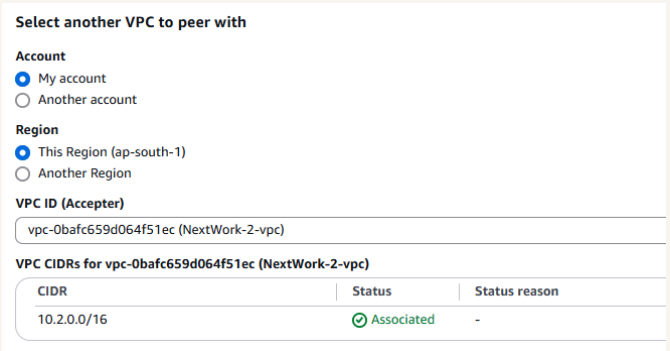


# VPC Peering

A VPC peering connection is a private network link between two VPCs, allowing them to communicate securely using private IP addresses without routing traffic over the public internet.

VPCs would use peering connections to enable direct, low-latency communication between resources in different VPCs, improving security and reducing costs by avoiding public internet exposure.

The difference between a Requester and an Accepter in a peering connection is that the Requester initiates the connection request, while the Accepter receives it and must approve it for the connection to be established.



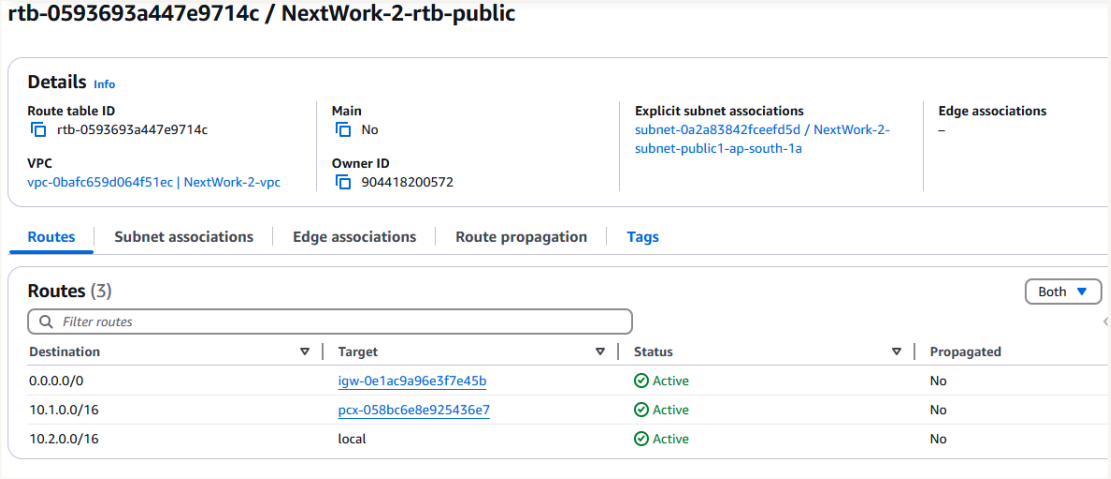
RA
**Ranjith D B**
NextWork Student



# Updating route tables

After accepting a peering connection, my VPCs' route tables need to be updated because without routes, traffic in one VPC wouldn't know how to reach the other, preventing communication between the two networks.

My VPCs' new routes have a destination of 10.2.0.0/16 for VPC 1 and 10.1.0.0/16 for VPC 2. The routes' target was the VPC peering connection (VPC 1 <> VPC 2).



RA
**Ranjith D B**
NextWork Student


# In the second part of my project...

## Step 5 - Use EC2 Instance Connect

In this step, we're connecting to the first EC2 instance using EC2 Instance Connect to test our VPC peering setup. We'll verify connectivity and troubleshoot any issues that arise during the process.

## Step 6 - Connect to EC2 Instance 1

We're retrying the connection to Instance 1 using EC2 Instance Connect after assigning an Elastic IP.

## Step 7 - Test VPC Peering

We're testing VPC peering by sending messages from Instance 1 to Instance 2. If any connection issues arise, we'll troubleshoot and resolve them to ensure successful communication between both instances.

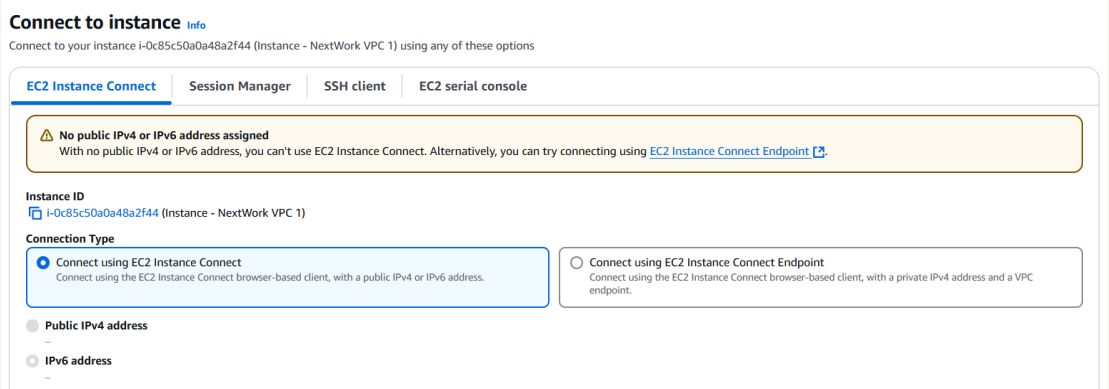
RA
**Ranjith D B**
NextWork Student



# Troubleshooting Instance Connect

Next, I used EC2 Instance Connect to access my EC2 instance without needing an SSH key pair, allowing me to run commands directly from the AWS console.

I was stopped from using EC2 Instance Connect as my instance had no public IPv4 address, preventing internet-based connections.



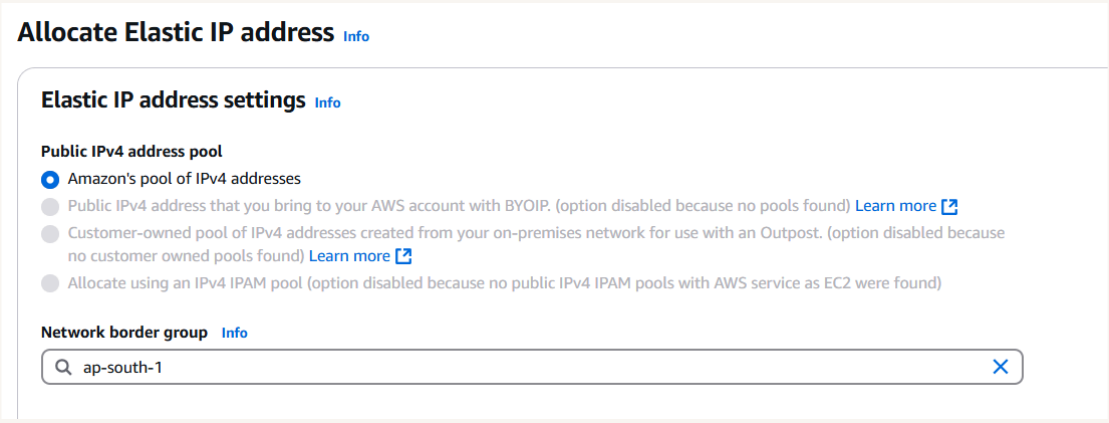
RA
**Ranjith D B**
NextWork Student



# Elastic IP addresses

To resolve this error, I set up Elastic IP addresses. Elastic IP addresses are static public IPv4 addresses that can be assigned to AWS resources, ensuring consistent external accessibility.

Associating an Elastic IP address resolved the error because it provided my EC2 instance with a public IP, enabling EC2 Instance Connect to establish a connection.



RA
**Ranjith D B**
NextWork Student

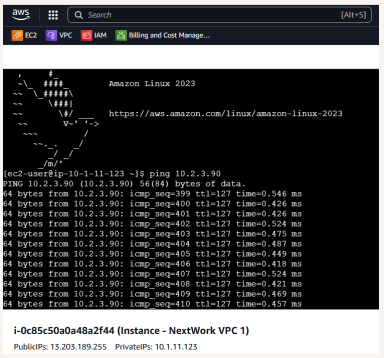


# Troubleshooting ping issues

To test VPC peering, I ran the command ping [Private IPv4 of Instance 2] from Instance 1. This sent ICMP echo requests to verify connectivity between the two instances.

A successful ping test would validate my VPC peering connection because it confirms that Instance 1 can reach Instance 2, meaning the routing, security groups, and network ACLs are correctly configured.

I had to update my second EC2 instance's security group because it was blocking ICMP traffic. I added a new rule that allowed All ICMP - IPv4 from VPC 1 (10.1.0.0/16) to enable successful ping responses.





# Everyone should be in a job they love.

Check out nextwork.org for more projects