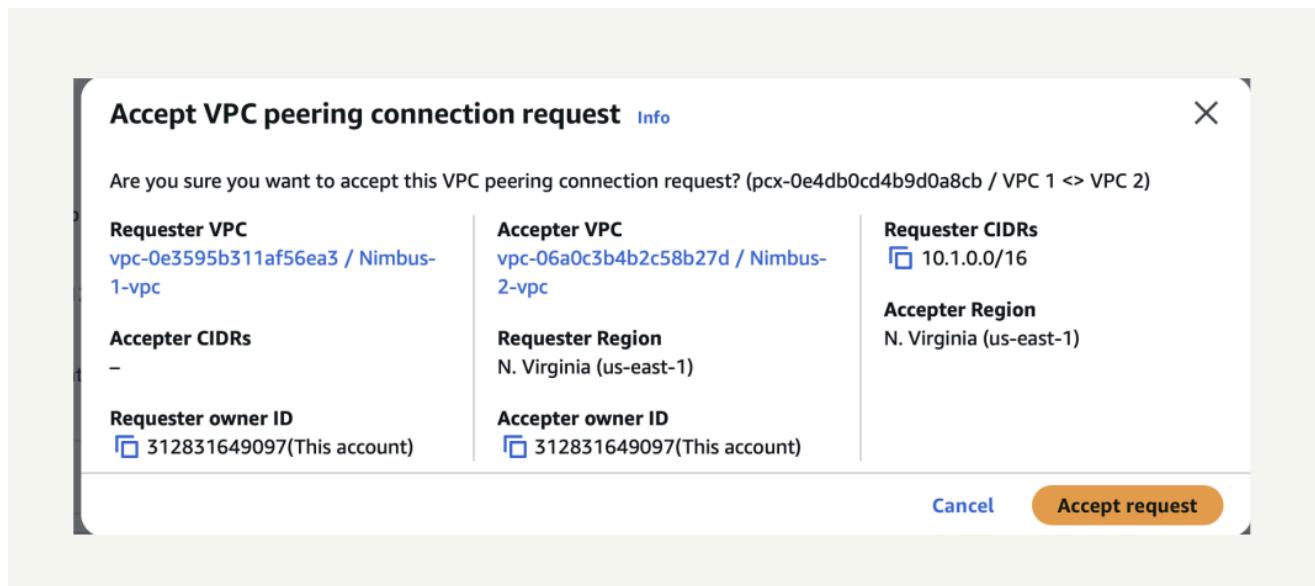


# VPC Peering

Erik Gonzalez



# Introducing Today's Project!

## How I used Amazon VPC in this project

In today's project, I used Amazon VPC to set up a multi-VPC architecture, did a peer test between two VPCs using ping command, and updated security groups to allow traffic.

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

One thing I didn't expect in this project was the amount of errors I had to remediate.

## This project took me...

This project took me about 1 hour and a half.

# In the first part of my project...

## Step 1 – Set up my VPC

In this step, I will use the launch wizard to create two VPCs in just minutes.

## Step 2 – Create a Peering Connection

In this step, I will set up a VPC peering connection which is a VPC component designed to directly connect to VPCs together.

## Step 3 – Update Route Tables

In this step, I will set up a way for traffic coming from VPC 1 to get to VPC 2 and vice versa.

## Step 4 – Launch EC2 Instances

In this step, I will launch an EC2 instance in each VPC, so we can use them to test your VPC peering connection later.

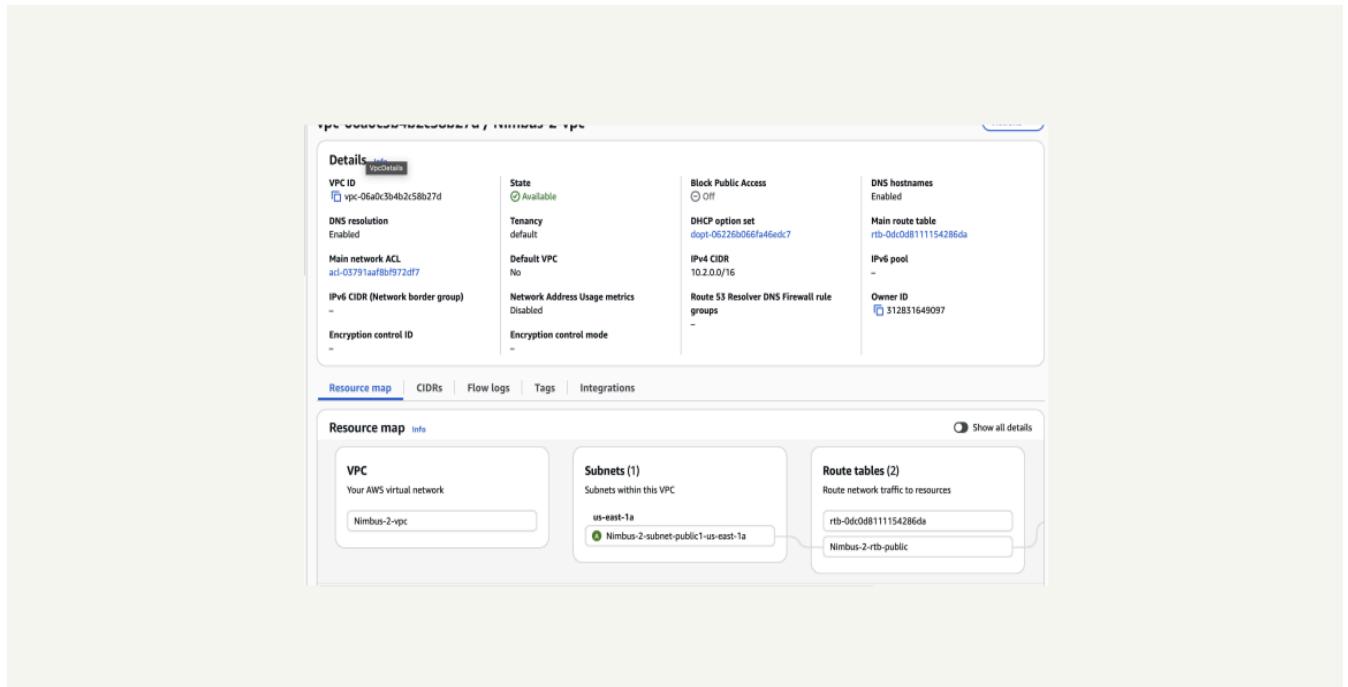
# Multi-VPC Architecture

I started my project by launching two VPCs - they have CIDR blocks and each have one public subnet.

The CIDR blocks for VPCs 1 and 2 are 10.1.0.0/16 and 10.2.0.0/16 respectively. They have to be unique because once you set up a VPC peering connection, route tables need unique addresses for correct routing across VPCs.

## I also launched 2 EC2 instances

I didn't set up key pairs for these EC2 instances because I am using EC2 Instance Connect to directly connect with our EC2 instance and later in this project, which handles key pair creation and management.



# VPC Peering

A VPC peering connection is a direct connection between two VPCs. A peering connection lets VPCs and their resources route traffic between them using their private IP addresses. This means data can now be transferred between VPCs without going through the public internet.

VPCs would use peering connections to communicate between servers/connections.

The difference between a Requester and an Acceptor in a peering connection is that the Requester is the VPC that initiates a peering connection and an Acceptor is the VPC that receives a peering connection request.

Select another VPC to peer with

**Account**

My account  
 Another account

**Region**

This Region (us-east-1)  
 Another Region

**VPC ID (Acceptor)**

vpc-06a0c3b4b2c58b27d (Nimbus-2-vpc) ▾

**VPC CIDRs for vpc-06a0c3b4b2c58b27d (Nimbus-2-vpc)**

CIDR	Status	Status reason
10.2.0.0/16	Associated	-

**Tags**  
A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.

Key Value - optional

Name   Value

You can add 49 more tags.

# Updating route tables

After accepting a peering connection, my VPCs' route tables need to be updated because the default route table doesn't have a route using the peer connection yet - this needs to be set up so that resources can be directed to the peering connection when trying to reach the other VPC.

My VPCs' new routes have a destination of the other VPC's CIDR block. The routes' target was the peering connection I set up.

Routes (3)					Edit routes	
Filter routes					Both	1
Destination	Target	Status	Propagated	Route Origin		
0.0.0.0/0	<a href="#">igw-0ac2ce33e17332d73</a>	Active	No	Create Route		
10.1.0.0/16	<a href="#">pcx-0e4db0cd4b9d0a8cb</a>	Active	No	Create Route		
10.2.0.0/16	local	Active	No	Create Route Table		

# In the second part of my project...

## Step 5 - Use EC2 Instance Connect

In this step, I will Use EC2 Instance Connect to connect to your first EC2 instance.

## Step 6 - Connect to EC2 Instance 1

In this step, I will Use EC2 Instance Connect to connect to Instance 1 (one more time)

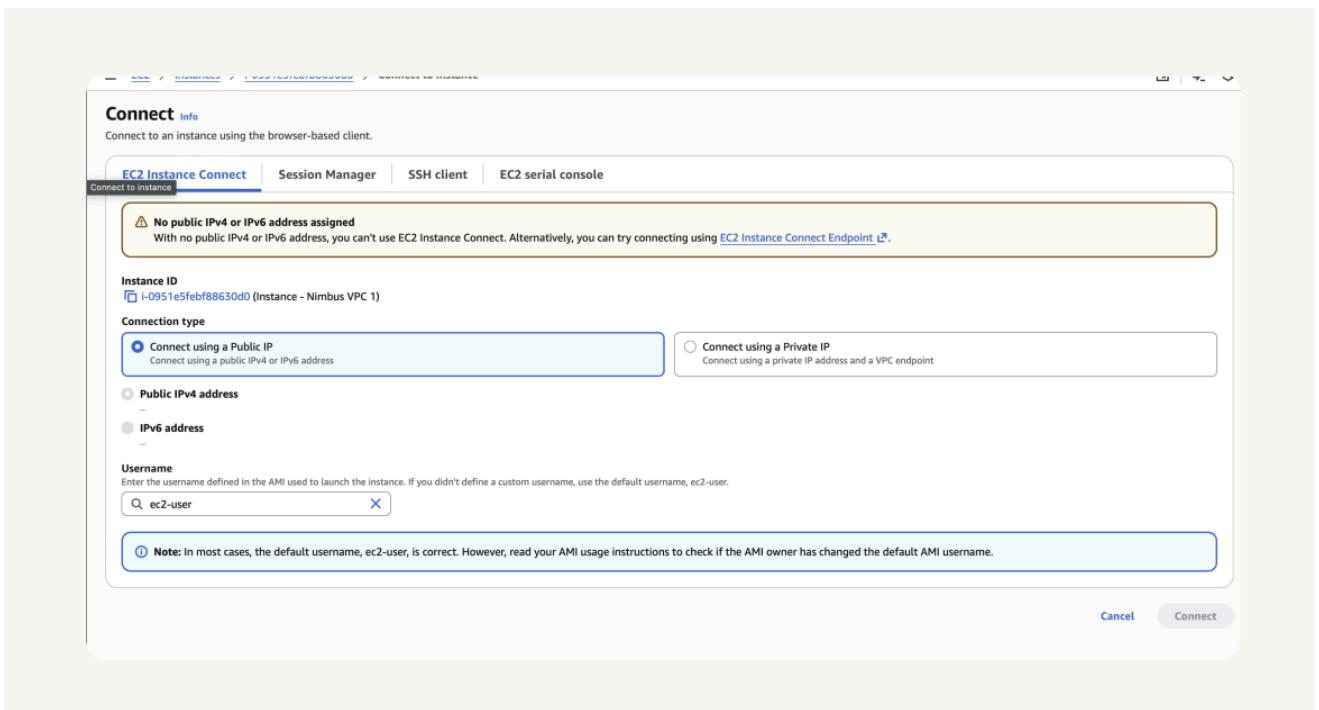
## Step 7 - Test VPC Peering

In this step, I will get Instance 1 to send test messages to Instance 2.

# Troubleshooting Instance Connect

Next, I used EC2 Instance Connect to directly connect with the instance - Nimbus VPC 1 just by using the AWS management console.

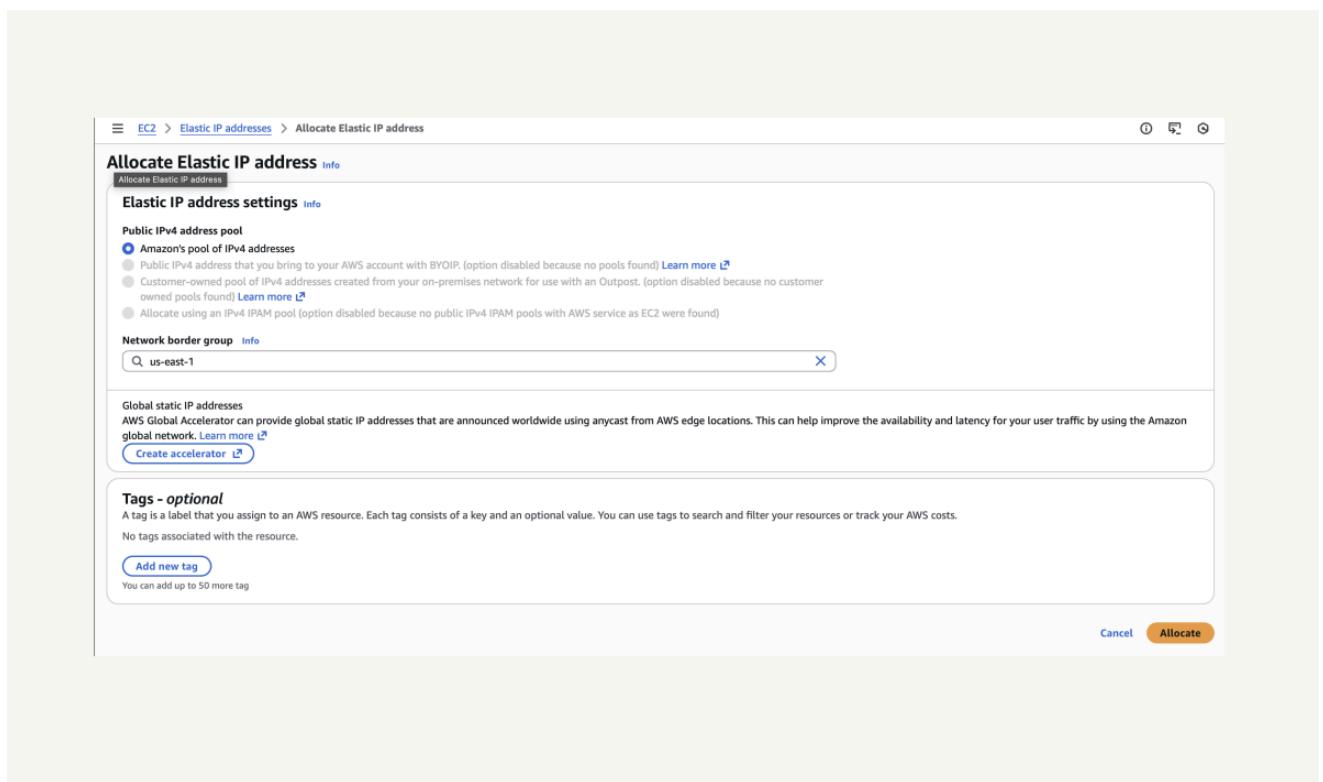
I was stopped from using EC2 Instance Connect as the instance did not have a public IPv4 address.



# Elastic IP addresses

To resolve this error, I set up Elastic IP addresses. Elastic IP addresses are static IPv4 addresses that get allocated to your AWS account, and are yours to delegate to an EC2 instance.

Associating an Elastic IP address resolved the error because it gives our instance a Public IP address. Fulfilling all the requirements for instance connect to work.



# Troubleshooting Ping Issues

To test VPC peering, I ran the command 'ping'

A successful ping test would validate my VPC peering connection because the ping test would otherwise display communication between two VPCs. The fact that I got a ping response shows clean and clear communication.

I had to update my second EC2 instance's security group because It was not letting in ICMP traffic - which is the traffic type of a ping message. I added a new rule that allows ICMP traffic to come in from any resource and VPC too.

```
PING 10.2.4.201 (10.2.4.201) 56(84) bytes of data.  
64 bytes from 10.2.4.201: icmp_seq=543 ttl=127 time=0.538 ms  
64 bytes from 10.2.4.201: icmp_seq=544 ttl=127 time=0.488 ms  
64 bytes from 10.2.4.201: icmp_seq=545 ttl=127 time=1.08 ms  
64 bytes from 10.2.4.201: icmp_seq=546 ttl=127 time=0.420 ms  
64 bytes from 10.2.4.201: icmp_seq=547 ttl=127 time=0.476 ms  
64 bytes from 10.2.4.201: icmp_seq=548 ttl=127 time=0.463 ms  
64 bytes from 10.2.4.201: icmp_seq=549 ttl=127 time=0.416 ms  
64 bytes from 10.2.4.201: icmp_seq=550 ttl=127 time=0.441 ms  
64 bytes from 10.2.4.201: icmp_seq=551 ttl=127 time=0.422 ms  
64 bytes from 10.2.4.201: icmp_seq=552 ttl=127 time=0.435 ms  
64 bytes from 10.2.4.201: icmp_seq=553 ttl=127 time=0.435 ms  
64 bytes from 10.2.4.201: icmp_seq=554 ttl=127 time=0.441 ms  
64 bytes from 10.2.4.201: icmp_seq=555 ttl=127 time=0.469 ms  
64 bytes from 10.2.4.201: icmp_seq=556 ttl=127 time=0.423 ms  
64 bytes from 10.2.4.201: icmp_seq=557 ttl=127 time=0.422 ms  
64 bytes from 10.2.4.201: icmp_seq=558 ttl=127 time=0.493 ms  
64 bytes from 10.2.4.201: icmp_seq=559 ttl=127 time=0.412 ms  
64 bytes from 10.2.4.201: icmp_seq=560 ttl=127 time=0.505 ms  
64 bytes from 10.2.4.201: icmp_seq=561 ttl=127 time=0.448 ms  
64 bytes from 10.2.4.201: icmp_seq=562 ttl=127 time=0.451 ms  
64 bytes from 10.2.4.201: icmp_seq=563 ttl=127 time=0.425 ms  
64 bytes from 10.2.4.201: icmp_seq=564 ttl=127 time=0.455 ms  
64 bytes from 10.2.4.201: icmp_seq=565 ttl=127 time=0.424 ms  
64 bytes from 10.2.4.201: icmp_seq=566 ttl=127 time=0.507 ms  
64 bytes from 10.2.4.201: icmp_seq=567 ttl=127 time=0.454 ms  
64 bytes from 10.2.4.201: icmp_seq=568 ttl=127 time=0.499 ms  
64 bytes from 10.2.4.201: icmp_seq=569 ttl=127 time=0.793 ms  
64 bytes from 10.2.4.201: icmp_seq=570 ttl=127 time=0.502 ms  
64 bytes from 10.2.4.201: icmp_seq=571 ttl=127 time=1.04 ms  
64 bytes from 10.2.4.201: icmp_seq=572 ttl=127 time=0.474 ms  
64 bytes from 10.2.4.201: icmp_seq=573 ttl=127 time=0.490 ms  
64 bytes from 10.2.4.201: icmp_seq=574 ttl=127 time=0.564 ms  
64 bytes from 10.2.4.201: icmp_seq=575 ttl=127 time=0.411 ms  
64 bytes from 10.2.4.201: icmp_seq=576 ttl=127 time=0.440 ms  
64 bytes from 10.2.4.201: icmp_seq=577 ttl=127 time=0.487 ms
```