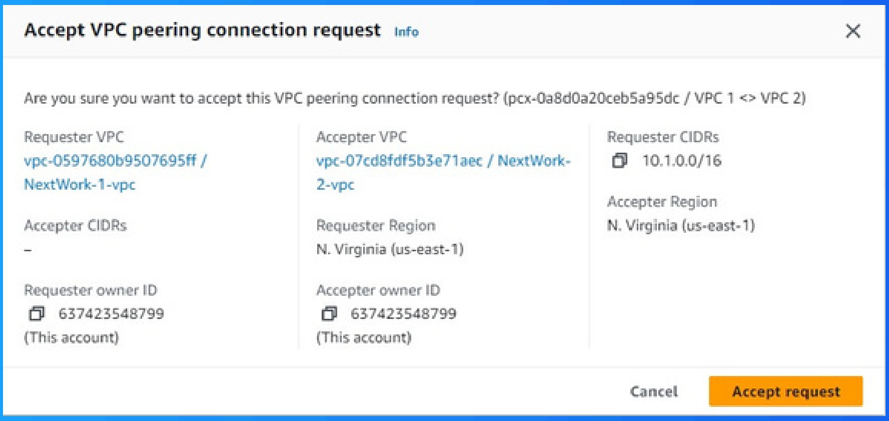
VPC Peering

Hassan Gachoka



Accept VPC peering connection request Info

Are you sure you want to accept this VPC peering connection request? (pcx-0a8d0a20ceb5a95dc / VPC 1 <> VPC 2)

Requester VPC
vpc-0597680b9507695ff / NextWork-1-vpc

Accepter CIDRs
–

Requester owner ID
637423548799
(This account)

Accepter VPC
vpc-07cd8fdf5b3e71aec / NextWork-2-vpc

Requester Region
N. Virginia (us-east-1)

Accepter owner ID
637423548799
(This account)

Requester CIDRs
10.1.0.0/16

Accepter Region
N. Virginia (us-east-1)

Introducing Today's Project!

What is Amazon VPC?

Amazon VPC (Virtual Private Cloud) allows you to create isolated networks within AWS. It offers customization, security controls, scalability, and cost-effectiveness, making it ideal for deploying secure applications in the cloud.

How I used Amazon VPC in this project

In today's project, I used Amazon VPC to create isolated networks for two EC2 instances. I set up VPC peering between them, allowing secure communication and testing connectivity through ping commands, ensuring proper traffic flow.

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

Even with experience, it's easy to overlook the little things like missing an inbound ICMP rule that can halt everything. This project was a great reminder that attention to detail, no matter how small, makes all the difference in network setups.

This project took me...

This project took a bit longer than expected—about 40 minutes—mainly due to troubleshooting the VPC peering connection and updating security rules. It was a great reminder of how crucial small details can be, even for those with experience!

In the first part of my project...

Step 1 Set up my VPC

I’m creating two VPCs from scratch using the VPC wizard in this step. This tool will help me quickly set up the VPCs with the necessary configurations, making the process faster and more efficient.

Step 2 Create a Peering Connection

In this step, I'll setup a peering connection between the two VPCs, allowing them to communicate with each other. This involves creating a connection link that bridges the VPCs for seamless data exchange.

Step 3 Update Route Tables

In this step, I'm configuring the route tables for both VPCs to allow traffic to flow between them through the peering connection, enabling communication and resource access.

Step 4 Launch EC2 Instances

In this step, I'm launching EC2 instances in both VPCs. These instances will serve as test resources, allowing me to verify that the VPC peering connection is functioning correctly and enabling communication between the two VPCs.

Hassan Gachoka
linkedin.com/gachokahassan

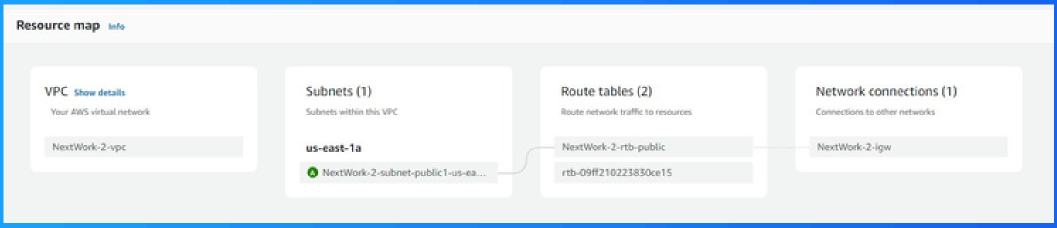
Multi-VPC Architecture

I started my project by launching 2 VPCs, each with 1 public subnet. No private subnets were created, keeping it straightforward with a single public subnet in both VPCs for todayś' project.

The CIDR blocks for VPCs 1 and 2 are 10.1.0.0/16 and 10.2.0.0/16. They must be unique to ensure proper routing in the route tables. Unique IP ranges and well-configured route tables and security groups are essential for effective traffic flow.

I also launched 2 EC2 instances

I didn't set up key pairs for these EC2 instances as I'm using EC2 Instance Connect, which allows me to SSH into my instances without managing key pairs, simplifying access and enhancing security.

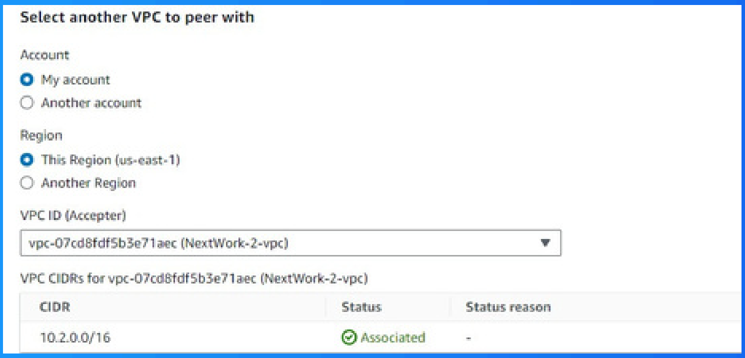


VPC Peering

A VPC peering connection is a networking connection that allows two AWS VPCs to communicate with each other as if they were within the same network. It enables resources in both VPCs to exchange traffic securely without using public internet routes.

VPCs use peering connections to enable direct communication between resources in different VPCs, allowing efficient data transfer and resource sharing without exposing traffic to the public internet.

The difference between a Requester and an Accepter in a peering connection is that the Requester initiates the peering request, while the Accepter accepts or declines the request to establish the connection between VPCs.



Select another VPC to peer with

Account
[✓ My account]
Another account

Region
[✓ This Region (us-east-1)]
Another Region

VPC ID (Accepter)
vpc-07cd8fdf5b3e71aec (NextWork-2-vpc)

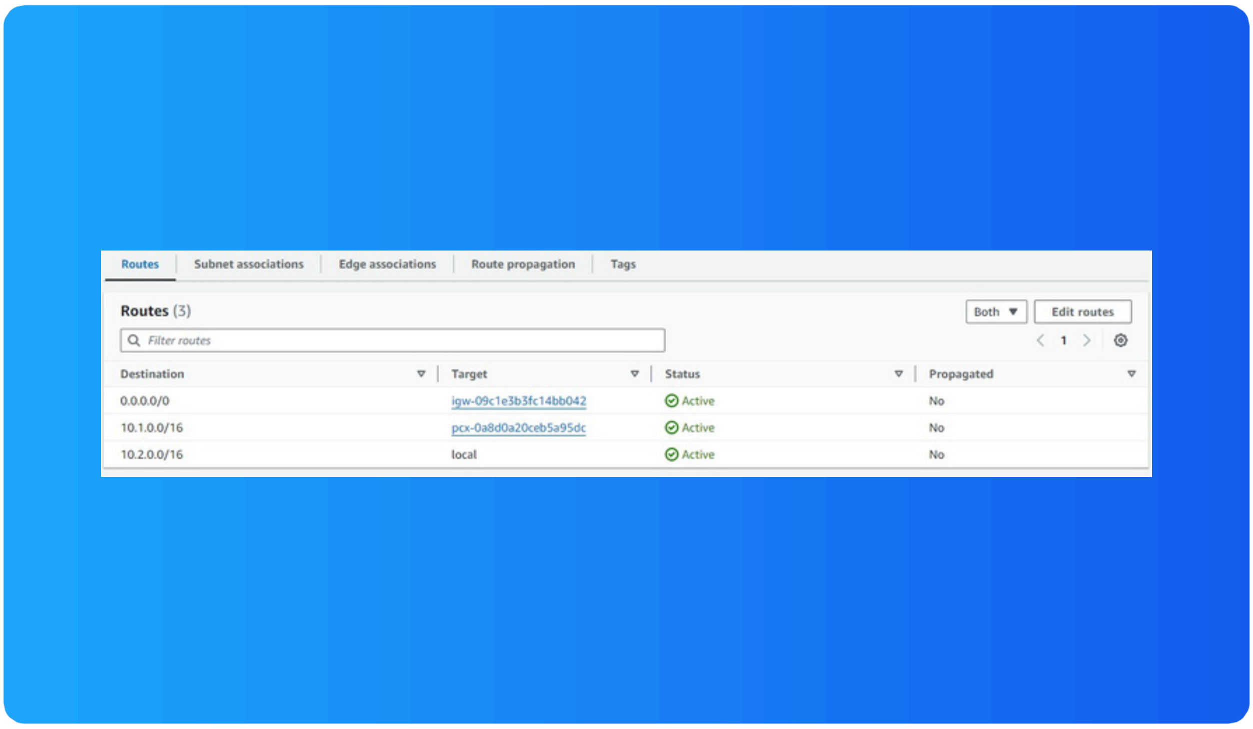
VPC CIDRs for vpc-07cd8fdf5b3e71aec (NextWork-2-vpc)

CIDR	Status	Status reason
10.2.0.0/16	✓ Associated	-

Updating route tables

After accepting a peering connection, my VPCs' route tables need to be updated because traffic from one VPC won't know how to reach resources in the other without specific routes directing it through the peering connection.

My VPCs' new routes have a destination of 10.1.0.0/16 for VPC 2 and 10.2.0.0/16 for VPC 1. The routes' target was the respective peering connection established between the two VPCs.



Destination	Target	Status	Propagated
0.0.0.0/0	igw-09c1e3b3fc14bb042	Active	No
10.1.0.0/16	pcx-0a8d0a20ceb5a95dc	Active	No
10.2.0.0/16	local	Active	No

In the second part of my project...

Step 5 Use EC2 Instance Connect

In this step, I'll connect to my first EC2 instance usingEC2 Instance Connect to test the VPC peering connection. If any connection errors arise, I’ll troubleshoot to ensure successful communication between the instances.

Step 6 Connect to EC2 Instance 1

In this step, I'll try to connect to EC2 Instance 1 using EC2 Instance Connect again. After resolving the previous error with an Elastic IP, I’ll ensure successful access to the instance and troubleshoot any new errors that may arise.

Step 7 Test VPC Peering

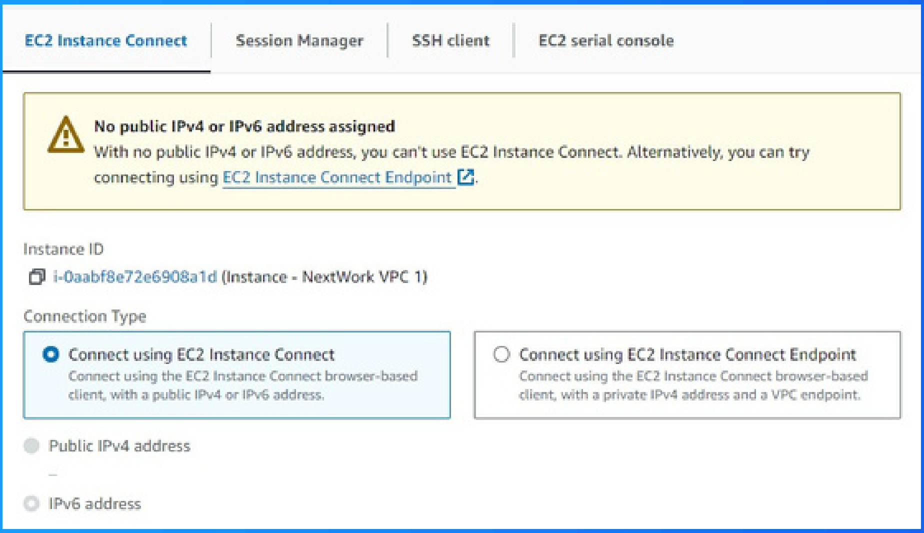
In this step, I'm testing VPC peering by connecting directly from the EC2 instance in NextWork VPC 1 to the EC2 instance in NextWork VPC 2. I'll send ping requests and troubleshoot any errors for proper communication.

Hassan Gachoka
linkedin.com/gachokahassan

Troubleshooting Instance Connect

Next, I used EC2 Instance Connect to securely access my EC2 instance without needing SSH keys. This allows me to connect quickly and easily to my instance for testing and troubleshooting the VPC peering connection.

I was stopped from using EC2 Instance Connect as my instance didn't have a public IPv4 address assigned, which is necessary for the connection. This happened because I had disabled the auto-assign public IP option during the setup.



No public IPv4 or IPv6 address assigned
With no public IPv4 or IPv6 address, you can't use EC2 Instance Connect. Alternatively, you can try connecting using EC2 Instance Connect Endpoint.

Instance ID
i-0aabf8e72e6908a1d (Instance - NextWork VPC 1)

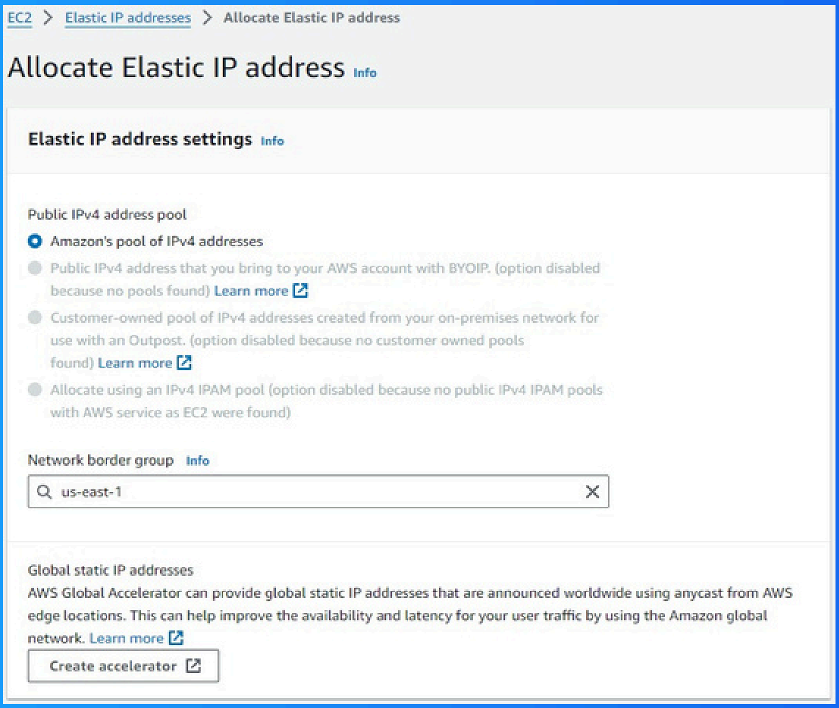
Connection Type
[✓ Connect using EC2 Instance Connect] Connect using the EC2 Instance Connect browser-based client, with a public IPv4 or IPv6 address.
Connect using EC2 Instance Connect Endpoint: Connect using the EC2 Instance Connect browser-based client, with a private IPv4 address and a VPC endpoint.

Public IPv4 address
–
IPv6 address

Elastic IP addresses

To resolve this error, I set up Elastic IP addresses. Elastic IP addresses are a static IPv4 addresses designed for dynamic cloud computing ensuring a consistent public IP even if the instance is restarted.

Associating an Elastic IP address resolved the error because it provided my EC2 instance with a public IP, allowing me to use EC2 Instance Connect.



EC2 > Elastic IP addresses > Allocate Elastic IP address

Allocate Elastic IP address Info

Elastic IP address settings Info

Public IPv4 address pool

- [✓ Amazon's pool of IPv4 addresses]
- Public IPv4 address that you bring to your AWS account with BYOIP. (option disabled because no pools found) Learn more
- Customer-owned pool of IPv4 addresses created from your on-premises network for use with an Outpost. (option disabled because no customer owned pools found) Learn more
- Allocate using an IPv4 IPAM pool (option disabled because no public IPv4 IPAM pools with AWS service as EC2 were found)

Network border group Info

us-east-1

Global static IP addresses
AWS Global Accelerator can provide global static IP addresses that are announced worldwide using anycast from AWS edge locations. This can help improve the availability and latency for your user traffic by using the Amazon global network. Learn more

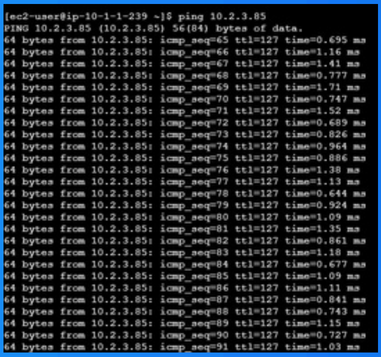
Hassan Gachoka
linkedin.com/gachokahassan

Troubleshooting ping issues

To test VPC peering, I ran the command ping 10.2.3.85 in the terminal to check connectivity between the EC2 instance in VPC 1 and the EC2 instance in VPC 2.

A successful ping test would validate my VPC peering connection because it indicates that the EC2 instance in VPC 1 can communicate with the EC2 instance in VPC 2, confirming proper routing and security group settings between the two VPCs.

I had to update my second EC2 instance's security group because it didn't allow inbound ICMP traffic. I added a new rule that permitted ICMP from the NextWork VPC 1 public subnet, enabling successful communication between the two instances.



```
[ec2-user@ip-10-1-1-239 ~]$ ping 10.2.3.85
PING 10.2.3.85 (10.2.3.85) 56(84) bytes of data.
64 bytes from 10.2.3.85: icmp_seq=65 ttl=127 time=0.695 ms
64 bytes from 10.2.3.85: icmp_seq=66 ttl=127 time=1.16 ms
64 bytes from 10.2.3.85: icmp_seq=67 ttl=127 time=1.41 ms
64 bytes from 10.2.3.85: icmp_seq=68 ttl=127 time=0.777 ms
64 bytes from 10.2.3.85: icmp_seq=69 ttl=127 time=1.71 ms
64 bytes from 10.2.3.85: icmp_seq=70 ttl=127 time=0.747 ms
64 bytes from 10.2.3.85: icmp_seq=71 ttl=127 time=1.52 ms
64 bytes from 10.2.3.85: icmp_seq=72 ttl=127 time=0.689 ms
64 bytes from 10.2.3.85: icmp_seq=73 ttl=127 time=0.826 ms
64 bytes from 10.2.3.85: icmp_seq=74 ttl=127 time=0.964 ms
64 bytes from 10.2.3.85: icmp_seq=75 ttl=127 time=0.886 ms
64 bytes from 10.2.3.85: icmp_seq=76 ttl=127 time=1.38 ms
64 bytes from 10.2.3.85: icmp_seq=77 ttl=127 time=1.13 ms
64 bytes from 10.2.3.85: icmp_seq=78 ttl=127 time=0.644 ms
64 bytes from 10.2.3.85: icmp_seq=79 ttl=127 time=0.924 ms
64 bytes from 10.2.3.85: icmp_seq=80 ttl=127 time=1.09 ms
64 bytes from 10.2.3.85: icmp_seq=81 ttl=127 time=1.35 ms
64 bytes from 10.2.3.85: icmp_seq=82 ttl=127 time=0.861 ms
64 bytes from 10.2.3.85: icmp_seq=83 ttl=127 time=1.18 ms
64 bytes from 10.2.3.85: icmp_seq=84 ttl=127 time=0.677 ms
64 bytes from 10.2.3.85: icmp_seq=85 ttl=127 time=1.09 ms
64 bytes from 10.2.3.85: icmp_seq=86 ttl=127 time=1.11 ms
64 bytes from 10.2.3.85: icmp_seq=87 ttl=127 time=0.841 ms
64 bytes from 10.2.3.85: icmp_seq=88 ttl=127 time=0.743 ms
64 bytes from 10.2.3.85: icmp_seq=89 ttl=127 time=1.15 ms
64 bytes from 10.2.3.85: icmp_seq=90 ttl=127 time=0.727 ms
64 bytes from 10.2.3.85: icmp_seq=91 ttl=127 time=1.03 ms
```