

VPC Connectivity

By Haroon Zaman | December 2025

Introduction

In this project, I focused on understanding how different parts of my VPC communicate with each other and with the internet. After creating both public and private EC2 instances, this project helped me test real-world connectivity scenarios inside a cloud network.

The goal was to practice how traffic flows inside a VPC, how instances interact across subnets, and how public resources reach the internet. This is an important step in mastering AWS networking and troubleshooting cloud connectivity issues.

In this project, I completed the following tasks:

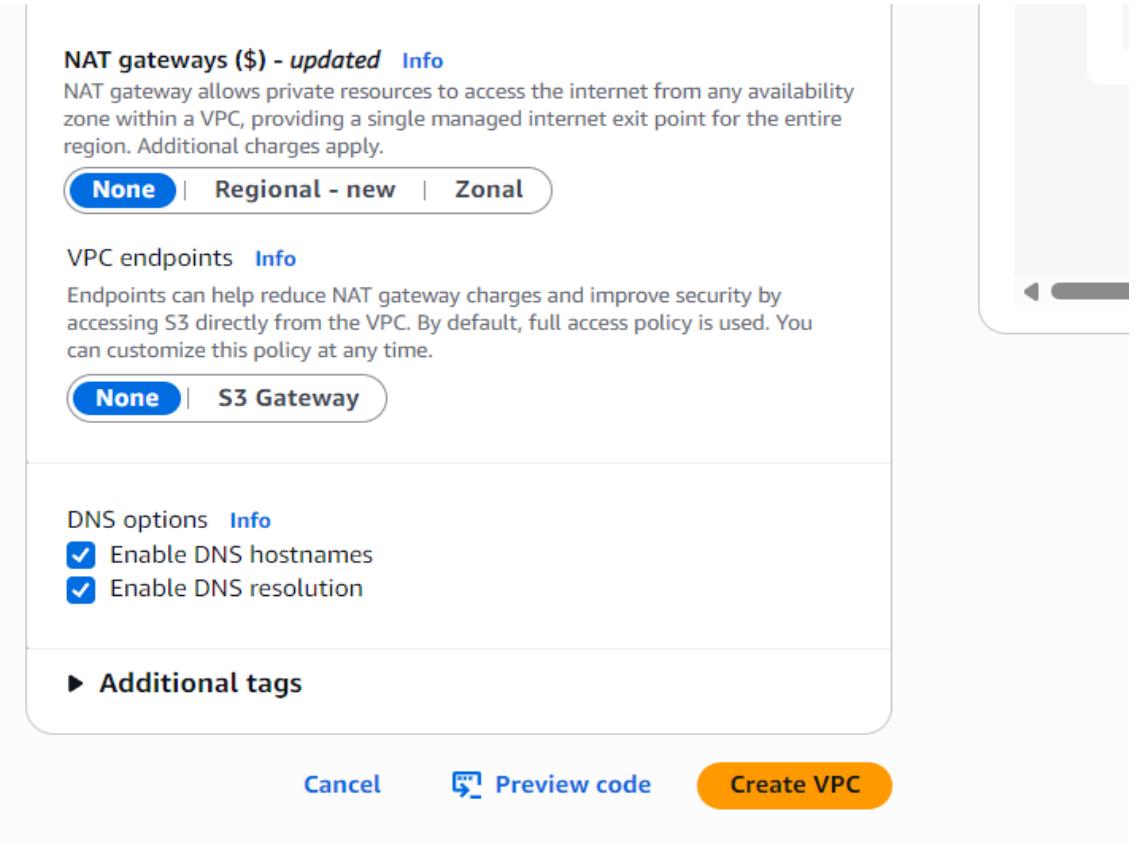
- **Connected to my Public EC2 Instance from the AWS Management Console** – to confirm that the instance has proper public access and my security group is configured correctly.
- **Tested Connectivity Between EC2 Instances** – to see how the public EC2 communicates with the private EC2 using internal IPs and security group rules.
- **Tested VPC Connectivity with the Internet** – to verify that internet-facing resources can reach the internet and respond properly.

The screenshot shows the 'Create VPC' wizard. On the left, under 'VPC settings', the 'VPC and more' option is selected. It includes sections for 'Name tag auto-generation' (auto-generated name 'MY_Next_VPC'), 'IPv4 CIDR block' (CIDR 10.0.0.0/16), and 'IPv6 CIDR block' (No IPv6 CIDR block). On the right, the 'Preview' section shows the VPC structure: 'VPC' (with 'Hide details' and 'Your AWS virtual network' info), 'Subnets (2)' (listing 'eu-north-1a' with subnets 'MY_Next_VPC-subnet-public1-eu-1' and 'MY_Next_VPC-subnet-private1-eu-1'), and 'Route tables (2)' (listing 'MY_Next_VPC-rtb-public' and 'MY_Next_VPC-rtb-private1-eu-n'). Arrows point from the VPC and Subnets sections to their respective route tables.

Creating a VPC Using “VPC and More”

- I created a new VPC using the **VPC and More** option, which automatically generated **one public subnet** and **one private subnet**.
- The **public subnet** was routed to the internet through the Internet Gateway created by the wizard.
- I selected **No NAT Gateway** because my **private subnet should NOT access the internet**.

- I chose **No VPC Endpoints** since this project didn't require private access to AWS services like S3.



- The preview window showed the full network layout the wizard would create — including the VPC, public subnet, private subnet, route tables, and internet gateway.
- After the VPC was created, I went through each component **one by one** and updated their names manually.
- Renaming everything (VPC, subnets, route tables, etc.) helped keep my environment organized and easy to recognize during the testing steps.

Details	Status and alarms	Monitoring	Security	Networking	Storage	Tags
VPC ID vpc-0e69c4b80354b0e63 (My_Network_VPC)	Subnet ID subnet-024df28332ec19edc (My_Public_Subnet1)	Availability zone eu-north-1a				
Availability zone ID eun1-a21	Outpost ID -					
▼ IP addresses Info						
Public IPv4 address 13.61.6.152 open address	Private IPv4 addresses 10.0.0.34	IPv6 addresses -				
Secondary private IPv4 addresses -	Carrier IP addresses (ephemeral) -					
▼ Hostname and DNS Info						
Public DNS -	Private IP DNS name (IPv4 only) ip-10-0-0-34.eu-north-1.compute.internal	IPv4-only IP based name: A record only -				
Dualstack - IP based name: A and AAAA record -	IPv6-only - IP based name: AAAA record only -	Public hostname type -				
Private hostname type IP name: ip-10-0-0-34.eu-north-1.compute.internal	Use RBN as guest OS hostname Disabled	Answer RBN DNS hostname IPv4 Disabled				
Answer RBN DNS hostname IPv6 -	Answer private resource DNS name -					

Public EC2 Networking Overview

- After creating the public EC2 instance, I reviewed the networking panel.
- It was correctly connected to **My_Public_Subnet1**, which is the public subnet created by the VPC wizard.
- The instance was assigned a **Public IP: 13.6.16.152**, allowing it to be accessed from the internet.
- It also received a **Private IP: 10.0.0.34**, used for internal communication inside the VPC.
- This confirmed that the public EC2 was set up correctly for both internet and VPC connectivity.

Instance summary for i-05b6f1dfc79829a02 (My_Public_EC2) Info		Actions	
Updated less than a minute ago		Connect Instance state Actions	
Instance ID i-05b6f1dfc79829a02	Public IPv4 address 13.61.6.152 open address	Private IPv4 addresses 10.0.0.34	
IPv6 address -	Instance state Running	Public DNS -	
Hostname type IP name: ip-10-0-0-34.eu-north-1.compute.internal	Private IP DNS name (IPv4 only) ip-10-0-0-34.eu-north-1.compute.internal	Elastic IP addresses	
Answer private resource DNS name	Instance type		

Connecting to the Public EC2 Instance

- From the instance summary page, I clicked the **Connect** button in the top-right corner.

Connect [Info](#)
Connect to an instance using the browser-based client.

EC2 Instance Connect	Session Manager	SSH client	EC2 serial console
Instance ID i-05b6f1dfc79829a02 (My_Public_EC2)			
Connection type			
<input checked="" type="radio"/> Connect using a Public IP 13.61.6.152	<input type="radio"/> Connect using a Private IP Connect using a private IP address and a VPC endpoint		
<input checked="" type="radio"/> Public IPv4 address 13.61.6.152	<input type="radio"/> IPv6 address -		
Username Enter the username defined in the AMI used to launch the instance. If you didn't define a custom username, use the default username, ec2-user.	<input type="text" value="ec2-user"/> X		
Note: In most cases, the default username, ec2-user, is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.			
		Cancel	Connect

- A new window opened showing different connection options.
- I selected **Connect using Public IP**, since this instance is in a public subnet and has a public address.
- After selecting the option, I clicked **Connect**, and the browser-based terminal opened successfully.



Initial Connection Issue

- At first, the instance did **not connect** through the browser terminal.
- This meant something in the networking or security configuration was blocking access.
- I had to **troubleshoot the issue** to find out what was preventing the connection.

A screenshot of the AWS Route Table configuration page. The top navigation bar includes 'Flow logs', 'Route table' (which is selected), 'Network ACL', 'CIDR reservations', 'Sharing', and 'Tags'. Below the navigation is a section titled 'Route table: rtb-05b130f2f43879211 / My_Public_RTable'. Underneath, there's a table titled 'Routes (2)' with columns for 'Destination' and 'Target'. The first row shows '10.0.0.0/16' as the destination and 'local' as the target. The second row shows '0.0.0.0/0' as the destination and 'igw-0ba7aa3860f1c6392' as the target.

Troubleshooting the Connection

- The first thing I checked was the **route table** associated with my public subnet.
- I confirmed that the route to the **Internet Gateway** was correctly configured.
- Everything in the route table looked good, so the issue was not related to routing.

A screenshot of the AWS Network ACL configuration page. The top navigation bar includes 'Flow logs', 'Route table', 'Network ACL' (selected), 'CIDR reservations', 'Sharing', and 'Tags'. Below the navigation is a section titled 'Network ACL: acl-0724c115c5b04687a / My_Public_ACL'. Underneath, there are two tables: 'Inbound rules (2)' and 'Outbound rules (2)'. Both tables have columns for 'Rule number', 'Type', 'Protocol', 'Port range', 'Source', 'Allow/Deny', and a 'Delete' icon. In the 'Inbound rules' table, rule 100 allows all traffic from 0.0.0.0/0. Rule * denies all traffic from 0.0.0.0/0. In the 'Outbound rules' table, rule 100 allows all traffic to 0.0.0.0/0. Rule * denies all traffic to 0.0.0.0/0.

Checking the Network ACL

- Next, I checked the **Network ACL** attached to my public subnet.
- Both **inbound and outbound rules** allowed all traffic, which confirmed that the ACL was not blocking the connection.
- Since the ACL was fully open, this wasn't the cause of the connection issue.

Edit inbound rules [Info](#)

Inbound rules control the incoming traffic that's allowed to reach the instance.

Inbound rules [Info](#)

Security group rule ID	Type	Info	Protocol	Info	Port range	Info	Source	Info	Description - optional	Info
sgr-07a51b6a9c7cc671	HTTP		TCP		80		Custom		<input type="text" value="0.0.0.0/0"/> X	Delete
-	SSH		TCP		22		Anywhere		<input type="text" value="0.0.0.0/0"/> X	Delete

[Add rule](#)

⚠ Rules with source of 0.0.0.0/0 or ::/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

[Cancel](#) [Preview changes](#) [Save rules](#)

Fixing the Security Group

- Then I checked the **Security Group** attached to my public EC2 instance.
 - I noticed that only **HTTP (port 80)** was allowed, which meant SSH connections were being blocked.
 - To fix this, I added a new **inbound rule** allowing **SSH (port 22)**.
 - I set the source to **0.0.0.0/0** since this is a public instance and needs to accept SSH connections from anywhere.
 - After saving the rule, the instance became reachable.

```
'          #
~\_\_###\_      Amazon Linux 2023
~~ \###\|
~~ \###|      https://aws.amazon.com/linux/amazon-linux-2023
~~     \#/   __>
~~     V~'   '->
~~~      /
~~ .-.
~~ / , /-
~/m/ , /-
[ec2-user@ip-10-0-0-34 ~]§
```

Connection Successful

- After updating the security group, I retried the connection.
 - This time, the browser terminal connected successfully to my public EC2 instance.
 - The issue was fully resolved, and the connection was established without any problems.
 - This completed **Step 1** of the project.

Testing Connectivity between Public and Private EC2

- Next, I tested the connectivity between my **public EC2** and **private EC2**.
 - Since both instances are in the same VPC, my **private EC2 should be able to talk to my public EC2** using its private IP.
 - This test helps confirm that internal communication inside the VPC is working and that the security group rules allow the connection.

The screenshot shows the AWS CloudWatch Metrics interface. At the top, there's a search bar and several filter buttons: 'All states', 'Actions', and 'Launch instances'. Below the search bar is a table of EC2 instances:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public
My_Private_EC2	i-0a76a5771420e3576	Running	t3.micro	3/3 checks passed	View alarms	eu-north-1a	-	-
My_Public_EC2	i-05b6f1dfc79829a02	Running	t3.micro	3/3 checks passed	View alarms	eu-north-1a	-	13.61.6

Below the table, a specific instance is selected: 'i-0a76a5771420e3576 (My_Private_EC2)'. The 'Networking' tab is active. On the left, there are sections for VPC ID, Availability zone ID, and IP addresses. The 'IP addresses' section shows a 'Private IPv4 addresses' field containing '10.0.1.174', which is highlighted with a red box. Other fields like 'Public IPv4 address' and 'Secondary private IPv4 addresses' are shown as empty.

Preparing to Test Internal Connectivity

- First, I copied the **private IPv4 address** of my Private EC2 instance.
- This private IP is only reachable inside the VPC, so it's required for any internal communication tests.
- Then I went to my **Public EC2**, which was already running and connected through the browser terminal.
- From there, I planned to ping the private IP to check if the two instances could communicate internally.

```

      #
~\ _###_#_          Amazon Linux 2023
~~ \####\ \
~~ \###|_
~~ \#/ _-->
~~ V~'-'-
~~ . / \
~~ . / /
~~ /m/ '
[ec2-user@ip-10-0-0-34 ~]$ ping 10.0.1.174
PING 10.0.1.174 (10.0.1.174) 56(84) bytes of data.

```

Ping Attempt and Issue Found

- I tried to ping the private EC2 from my public EC2 using its private IP address.
- Only **one ping was sent**, and it stopped immediately afterward.
- This showed that something was blocking the communication between the two instances.
- I had to **troubleshoot the issue** to find out why the internal connectivity wasn't working.

The screenshot shows the Network ACL configuration for a subnet. It displays two sections: Inbound rules (1) and Outbound rules (1). Both sections have a header with columns: Rule number, Type, Protocol, Port range, Source, and Allow/Deny. The Inbound rule is for All traffic (Type: All traffic, Protocol: All, Port range: All, Source: 0.0.0.0/0, Allow/Deny: Deny). The Outbound rule is also for All traffic (Type: All traffic, Protocol: All, Port range: All, Destination: 0.0.0.0/0, Allow/Deny: Deny).

Checking the Private Subnet's ACL

- I checked the **Network ACL** attached to my Private Subnet.
- I noticed that **no traffic at all was allowed** — both inbound and outbound rules were set to *deny*.
- Because of this, the private EC2 couldn't receive or reply to any ping requests.
- This confirmed that the ACL was the reason internal communication wasn't working.

The screenshot shows the 'Edit inbound rules' dialog. It lists one rule: Rule number 100, Type: All ICMP - IPv4, Protocol: ICMP (1), Port range: All, Source: 10.0.0.0/24, Allow/Deny: Allow. Below the table are buttons for 'Add new rule' and 'Sort by rule number'. At the bottom are 'Cancel', 'Preview changes', and a highlighted 'Save changes' button.

Fixing the Private Subnet's ACL

- To allow internal communication, I added new rules to the Private NACL.
- In the **Inbound Rules**, I allowed **ICMP** (used for ping) with **ICMP type: 100** and **source: 10.0.0.0/24**.
- In the **Outbound Rules**, I added the same ICMP rule with **destination: 10.0.0.0/24**.
- These rules allowed ping traffic between the public and private EC2 instances inside the VPC.

The screenshot shows the 'Edit inbound rules' dialog for a specific security group. It lists two rules under the 'Inbound rules' section. The first rule is for SSH (Type: SSH, Protocol: TCP, Port range: 22, Source: Custom, Source value: sg-0b82e411cbe7af96c, Allow/Deny: Allow). The second rule is for ICMP (Type: All ICMP - IPv4, Protocol: ICMP, Port range: All, Source: Custom, Source value: sg-0fa8a96d6e3f6da2a, Allow/Deny: Allow). A red box highlights the second ICMP rule. At the bottom are 'Cancel', 'Preview changes', and a highlighted 'Save rules' button.

Updating the Private Security Group

- After fixing the NACL, I checked the **Private Security Group** attached to my private EC2.
- I noticed that it only allowed **SSH**, and there were **no ICMP rules** for ping traffic.
- To allow the private EC2 to respond to pings, I added **ICMP** in both **Inbound** and **Outbound** rules.

- This ensured that ping requests coming from the public EC2 could reach the private EC2, and the private EC2 could send replies back.

```
4 bytes from 10.0.1.174: icmp_seq=928 ttl=127 time=0.175 ms
64 bytes from 10.0.1.174: icmp_seq=929 ttl=127 time=0.180 ms
64 bytes from 10.0.1.174: icmp_seq=930 ttl=127 time=0.185 ms
64 bytes from 10.0.1.174: icmp_seq=931 ttl=127 time=0.178 ms
64 bytes from 10.0.1.174: icmp_seq=932 ttl=127 time=0.176 ms
64 bytes from 10.0.1.174: icmp_seq=933 ttl=127 time=0.188 ms
64 bytes from 10.0.1.174: icmp_seq=934 ttl=127 time=0.178 ms
64 bytes from 10.0.1.174: icmp_seq=935 ttl=127 time=0.190 ms
64 bytes from 10.0.1.174: icmp_seq=936 ttl=127 time=0.220 ms
64 bytes from 10.0.1.174: icmp_seq=937 ttl=127 time=0.207 ms
64 bytes from 10.0.1.174: icmp_seq=938 ttl=127 time=0.215 ms
64 bytes from 10.0.1.174: icmp_seq=939 ttl=127 time=0.173 ms
64 bytes from 10.0.1.174: icmp_seq=940 ttl=127 time=0.193 ms
64 bytes from 10.0.1.174: icmp_seq=941 ttl=127 time=0.220 ms
64 bytes from 10.0.1.174: icmp_seq=942 ttl=127 time=0.190 ms
64 bytes from 10.0.1.174: icmp_seq=943 ttl=127 time=0.185 ms
64 bytes from 10.0.1.174: icmp_seq=944 ttl=127 time=0.177 ms
64 bytes from 10.0.1.174: icmp_seq=945 ttl=127 time=0.179 ms
64 bytes from 10.0.1.174: icmp_seq=946 ttl=127 time=0.172 ms
64 bytes from 10.0.1.174: icmp_seq=947 ttl=127 time=0.187 ms
64 bytes from 10.0.1.174: icmp_seq=948 ttl=127 time=0.205 ms
64 bytes from 10.0.1.174: icmp_seq=949 ttl=127 time=0.176 ms
64 bytes from 10.0.1.174: icmp_seq=950 ttl=127 time=0.186 ms
64 bytes from 10.0.1.174: icmp_seq=951 ttl=127 time=0.170 ms
64 bytes from 10.0.1.174: icmp_seq=952 ttl=127 time=0.192 ms
64 bytes from 10.0.1.174: icmp_seq=953 ttl=127 time=0.208 ms
64 bytes from 10.0.1.174: icmp_seq=954 ttl=127 time=0.169 ms
64 bytes from 10.0.1.174: icmp_seq=955 ttl=127 time=0.193 ms
64 bytes from 10.0.1.174: icmp_seq=956 ttl=127 time=0.194 ms
64 bytes from 10.0.1.174: icmp_seq=957 ttl=127 time=0.180 ms
64 bytes from 10.0.1.174: icmp_seq=958 ttl=127 time=0.180 ms
64 bytes from 10.0.1.174: icmp_seq=959 ttl=127 time=0.183 ms
```

i-05b6f1dfc79829a02 (My_Public_EC2)

Public IPs: 13.61.6.152 Private IPs: 10.0.0.34

Successful Internal Connectivity

- After updating the NACL and the security group, I went back to the public EC2 and tested the ping again.
- This time, the pings were **responding continuously**, without stopping.
- This confirmed that my **private EC2 instance was successfully connected** to my public EC2 through internal communication inside the VPC.

Testing VPC Connectivity With the Internet

- Next, I tested whether my VPC had proper internet connectivity using my **public EC2 instance**.
- Since the public EC2 is in a public subnet and has a public IP, it should be able to reach the internet through the Internet Gateway.
- Using the browser terminal on the public EC2, I ran basic commands to test external connectivity.

```
[ec2-user@ip-10-0-0-34 ~]$ curl example.com
<!DOCTYPE html><html lang="en"><head><title>Example Domain</title></head><body><meta name="viewport" content="width=device-width, initial-scale=1"><style>body{background:#eee; width:600px; margin:15vh auto; font-family: Arial, sans-serif}</style><h1>Example Domain</h1><p>This domain is for use in documentation examples without needing permission. Avoid use in operations.<p><a href="https://iana.org/domains/example">Learn more</a></p></body></html>
[ec2-user@ip-10-0-0-34 ~]$ 
[ec2-user@ip-10-0-0-34 ~]$ 
[ec2-user@ip-10-0-0-34 ~]$ 
[ec2-user@ip-10-0-0-34 ~]$ 
[ec2-user@ip-10-0-0-34 ~]$ 
[ec2-user@ip-10-0-0-34 ~]$ 
[ec2-user@ip-10-0-0-34 ~]$ curl nextwork.org
Redirecting...[ec2-user@ip-10-0-0-34 ~]$ 
[ec2-user@ip-10-0-0-34 ~]$ 
[ec2-user@ip-10-0-0-34 ~]$ 
[ec2-user@ip-10-0-0-34 ~]$ curl google.com
<HTML><HEAD><meta http-equiv="content-type" content="text/html; charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
<P>The document has moved<br/>to <A href="http://www.google.com/">here</A>.
</BODY></HTML>
[ec2-user@ip-10-0-0-34 ~]$ 
[ec2-user@ip-10-0-0-34 ~]$ 
[ec2-user@ip-10-0-0-34 ~]$ 
[ec2-user@ip-10-0-0-34 ~]$ 
[ec2-user@ip-10-0-0-34 ~]$ 
[ec2-user@ip-10-0-0-34 ~]$
```

i-05b6f1dfc79829a02 (My_Public_EC2)
PublicIPs: 13.61.6.152 PrivateIPs: 10.0.0.34



Testing Internet Access Using cURL

- To test internet connectivity, I used the **curl** command on my public EC2 instance.
- I used curl because it sends a request to a website and returns a response, which is a clear way to confirm if the instance can reach the internet.
- First, I ran:
curl www.example.com
This is a standard test website used for connectivity checks.
- Then I tested:
curl nextwork.org
- Finally, I ran:
curl google.com
- All three commands returned successful responses, which confirmed that my **public EC2 instance—and therefore my VPC—was connected to the external internet** through the Internet Gateway.

Conclusion

Through this project, I was able to fully test and understand how connectivity works inside a VPC. I successfully connected to my public EC2 instance, fixed security and routing issues, and validated internal communication between my public and private EC2 instances. I also confirmed that my VPC had proper internet access by testing external connectivity through the public EC2.

This project helped me understand how routing tables, NACLs, and security groups work together, and how each layer affects network communication. Overall, it strengthened my knowledge of AWS networking, troubleshooting, and secure cloud design.