AWS VPC

AWS-VPC

VPC

Imagine you want to set up a private, secure, and isolated area in the cloud where you can run your applications and store your data. This is where a VPC comes into play.

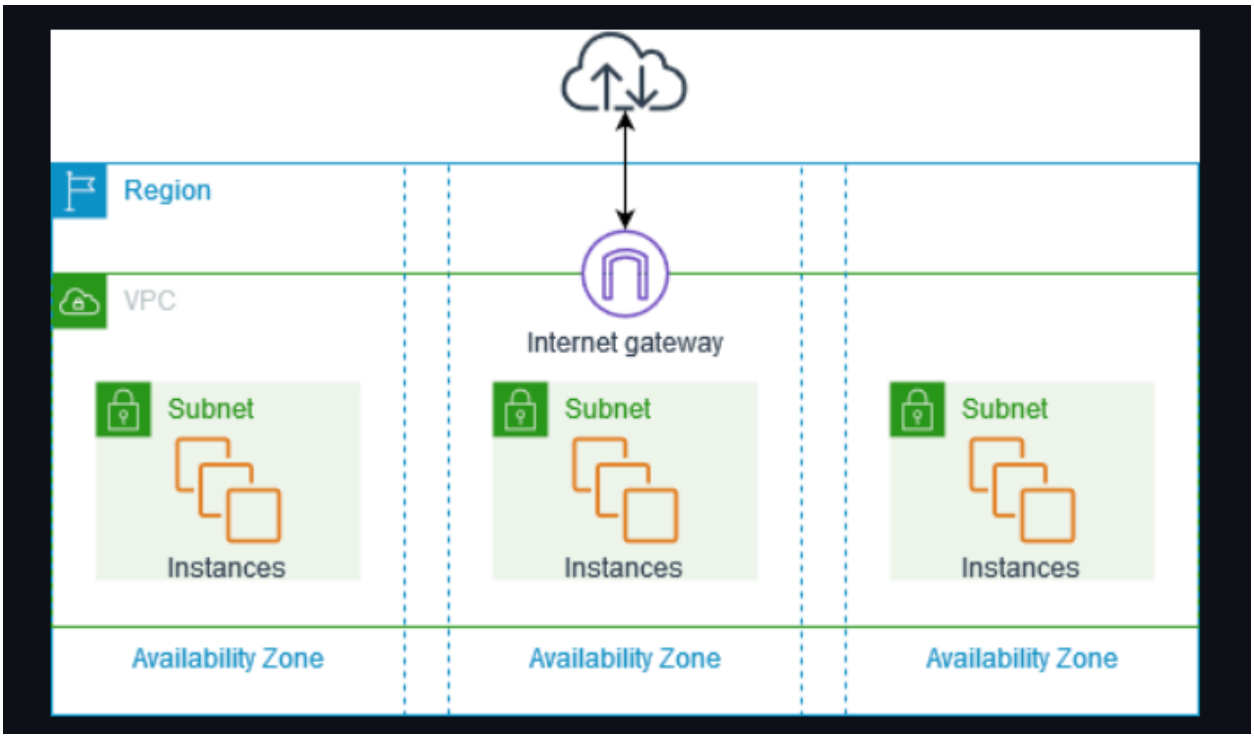
A VPC is a virtual network that you create in the cloud. It allows you to have your own private section of the internet, just like having your own network within a larger network. Within this VPC, you can create and manage various resources, such as servers, databases, and storage.

Think of it as having your own little "internet" within the bigger internet. This virtual network is completely isolated from other users' networks, so your data and applications are secure and protected.

Just like a physical network, a VPC has its own set of rules and configurations. You can define the IP address range for your VPC and create smaller subnetworks within it called subnets. These subnets help you organize your resources and control how they communicate with each other.

To connect your VPC to the internet or other networks, you can set up gateways or routers. These act as entry and exit points for traffic going in and out of your VPC. You can control the flow of traffic and set up security measures to protect your resources from unauthorized access.

With a VPC, you have control over your network environment. You can define access rules, set up firewalls, and configure security groups to regulate who can access your resources and how they can communicate.



By default, when you create an AWS account, AWS will create a default VPC for you but this default VPC is just to get started with AWS. You should create VPCs for applications or projects.

VPC components

The following features help you configure a VPC to provide the connectivity that your applications need:

Virtual private clouds (VPC)

A VPC is a virtual network that closely resembles a traditional network that you'd operate in your own data center. After you create a VPC, you can add subnets

Subnets:

A subnet is a range of IP addresses in your VPC. A subnet must reside in a single Availability Zone. After you add subnets, you can deploy AWS resources in your VPC.

IP addressing:

You can assign IP addresses, both IPv4 and IPv6, to your VPCs and subnets. You can also bring your public IPv4 and IPv6 GUA addresses to AWS and allocate them to resources in your VPC, such as EC2 instances, NAT gateways, and Network Load Balancers.

Network Access Control List (NACL):

A Network Access Control List is a stateless firewall that controls inbound and outbound traffic at the subnet level. It operates at the IP address level and can allow or deny traffic based on rules that you define. NACLs provide an additional layer of network security for your VPC.

Security Group:

A security group acts as a virtual firewall for instances (EC2 instances or other resources) within a VPC. It controls inbound and outbound traffic at the instance level. Security groups allow you to define rules that permit or restrict traffic based on protocols, ports, and IP addresses.

Routing:

Use route tables to determine where network traffic from your subnet or gateway is directed.

Gateways and endpoints:

A gateway connects your VPC to another network. For example, use an internet gateway to connect your VPC to the internet. Use a VPC endpoint to connect to AWS services privately, without the use of an internet gateway or NAT device.

Peering connections:

Use a VPC peering connection to route traffic between the resources in two VPCs.

Traffic Mirroring:

Copy network traffic from network interfaces and send it to security and monitoring appliances for deep packet inspection.

Transit gateways:

Use a transit gateway, which acts as a central hub, to route traffic between your VPCs, VPN connections, and AWS Direct Connect connections.

VPC Flow Logs:

A flow log captures information about the IP traffic going to and from network interfaces in your VPC.

VPN connections:

Connect your VPCs to your on-premises networks using AWS Virtual Private Network (AWS VPN).

Resources:

VPC with servers in private subnets and NAT

https://docs.aws.amazon.com/vpc/latest/userguide/vpc-example-private-subnets-nat.html

AWS VPC (Virtual Private Cloud):

Amazon VPC lets you create a logically isolated network in the AWS cloud where you can launch AWS resources (like EC2 instances, RDS databases, etc.) in a custom-defined virtual network.

You define:

IP range (CIDR block)

Subnets (public/private)

Route tables

Internet and NAT gateways

Security groups and NACLs

> Think of VPC like your own private data center inside AWS

Key Features of VPC:

Feature	Description
Subnets	Divide the VPC into smaller IP ranges
Internet Gateway	Enables communication with the internet
NAT Gateway	Allows private subnet instances to access internet
Route Tables	Define where traffic goes
Security Groups	Virtual firewalls for EC2
Network ACLs	Optional stateless firewall at subnet level

VPC Components (with Example)

Component	Example
VPC CIDR block	`10.0.0.0/16`
Public Subnet	`10.0.1.0/24`
Private Subnet	`10.0.2.0/24`
Internet Gateway	`IGW` attached to VPC
Route Table	Routes `0.0.0.0/0` to IGW
NAT Gateway	Allows internet access for private subnet
EC2 instance	Launched in subnet

Hands-On: Create VPC (Console)

Step 1: Create VPC

1. Go to VPC Dashboard > Your VPCs > Create VPC

2. Name: `MyCustomVPC`

3. IPv4 CIDR block: `10.0.0.0/16`

4. Leave another options default, click Create

Step 2: Create Subnets

1. Public Subnet

   Name: `PublicSubnet1`

   CIDR: `10.0.1.0/24`

   Availability Zone: choose any

2. Private Subnet

   Name: `PrivateSubnet1`

   CIDR: `10.0.2.0/24`

Step 3: Create Internet Gateway (IGW)

1. Go to Internet Gateways > Create IGW
2. Name: `MyIGW`
3. Attach it to your VPC

Step 4: Create Route Table

1. Go to Route Tables > Create
2. Name: `PublicRouteTable`
3. Associate it with `MyCustomVPC`
4. Add a route:

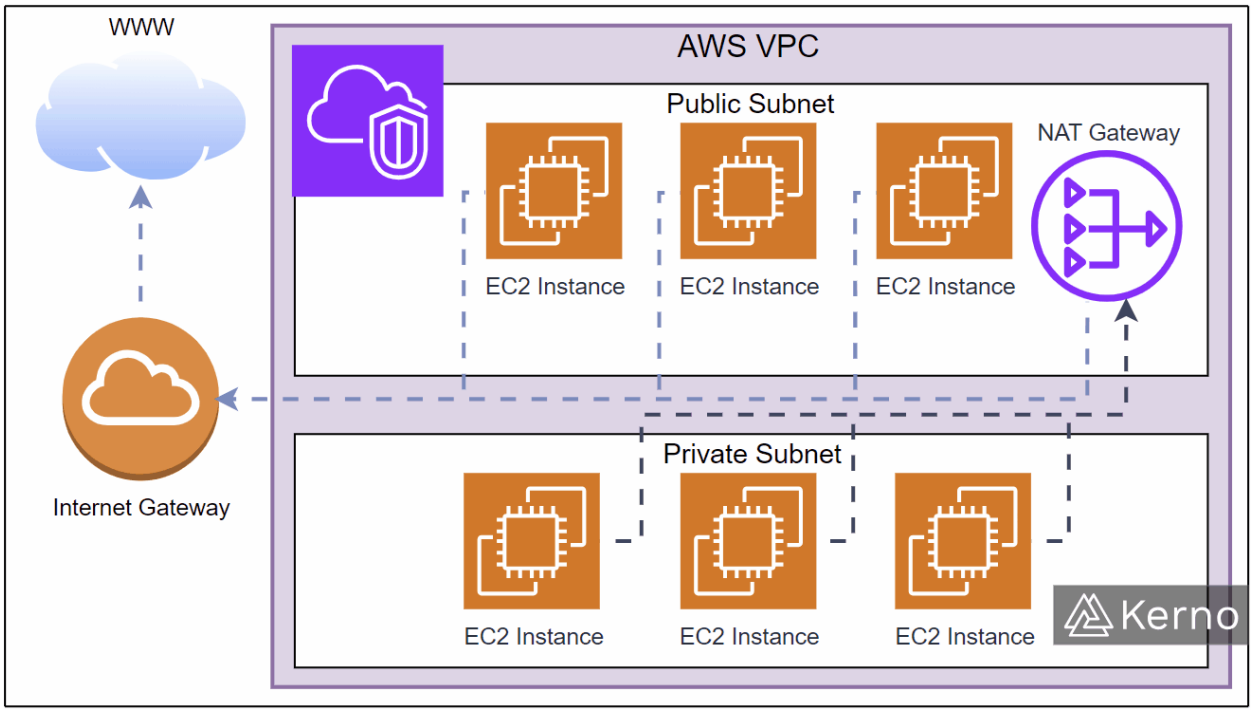
   Destination: `0.0.0.0/0`

   Target: `Internet Gateway (MyIGW)`
5. Associate it with `PublicSubnet1`

Step 5: Launch EC2 in Public Subnet

1. Launch a new EC2 instance
2. Choose subnet: `PublicSubnet1`
3. Enable `Auto-assign public IP`
4. Use security group to allow SSH/HTTP
5. Launch and connect — test internet access

Summary Diagram



Real-World Use Cases:

Use Case	VPC Setup
Public-facing web server	Public subnet + IGW
Private backend/database	Private subnet + no IGW
Private EC2 with internet	Private subnet + NAT Gateway
Full isolation (no internet)	Private subnets only

Real-Life Analogy: Office Building = VPC

Imagine you're setting up an office building for your company:

📦 The building is your VPC

🖥️ Rooms inside are Subnets

👷 Employees are EC2 Instances

🛡️ Security guards at the doors are Security Groups

📞 You install internet connection – that's your Internet Gateway

📬 Only specific rooms can receive deliveries – this is like NAT Gateway

🛣️ Hallways and signs telling where to go = Route Tables

Real-Life Production Example: E-commerce Website

Let’s say you’re building a real e-commerce web app like Amazon or Flipkart.

Requirements

1. Public-facing website (homepage, product list, etc.)
2. Secure backend APIs and admin tools
3. Database should not be publicly accessible
4. Some internal apps (CI/CD, monitoring)
5. Highly available, scalable setup

VPC Setup

Component	Setup
VPC CIDR	`10.0.0.0/16`
Public Subnet 1	`10.0.1.0/24` – contains EC2 with NGINX (Web)
Public Subnet 2	`10.0.2.0/24` – Load Balancer
Private Subnet 1	`10.0.3.0/24` – App server (Spring Boot/Django)
Private Subnet 2	`10.0.4.0/24` – Database (RDS, MySQL/Postgres)
Internet Gateway	Attached to VPC – for external web access
NAT Gateway	For private subnets to access the internet
Route Tables	Public subnet route 0.0.0.0/0 → IGW, Private → NAT
Security Groups	Only open necessary ports like 80/443/22
ACLs	Optional, subnet-level firewall rules

Workflow of the E-commerce Website

text

USER (browser)

|

ALB (in Public Subnet)

|

EC2 Web Server (in Public Subnet)

|

API Server (in Private Subnet)

|

Database (in Private Subnet - NO Internet access)

Communication Rules

Web server can talk to API server and ALB

API server can access the DB but no one from internet can directly access DB

Servers in private subnet use NAT Gateway for software updates or sending logs

Security Benefits

Only the web layer is exposed to the internet

Internal servers and database are isolated

Least privilege principle using security groups

Can create VPN/Direct Connect for secure on-premise link

In Short:

VPC is your custom data center in the cloud with walls, doors, locks, rules, and guards — you control everything about your environment's networking, security, and connectivity.

What is a CIDR Block?

CIDR stands for Classless Inter-Domain Routing.

A CIDR block defines a range of IP addresses using this format:

IP address / subnet mask

Example: 10.0.0.0/16

The IP address (like `10.0.0.0`) is the starting address.

The "/16" is the prefix length — it tells how many bits are fixed (network part) and how many are available for hosts.

CIDR Breakdown Example

CIDR: 10.0.0.0/16

This means:

First 16 bits (of 32) are fixed → `10.0`

Remaining 16 bits are for hosts

So, you can assign: 2^{16} - 2 = 65,534 IP addresses

(You subtract 2: 1 for network, 1 for broadcast)

CIDR Block Sizes Chart:

CIDR Block	Total IPs	Usable IPs	Subnet Mask
/8	16,777,216	16,777,214	255.0.0.0
/16	65,536	65,534	255.255.0.0
/24	256	254	255.255.255.0
/28	16	14	255.255.255.240
/32	1	0 (single IP)	255.255.255.255

You can use anything between `/16` to `/28` in AWS VPC/Subnets.

How to Calculate a CIDR Block

Let’s break this into 2 types:

1. Number of IPs from a CIDR

Formula:

Number of IPs = 2^(32 - subnet mask bits)

Usable IPs = Total - 2

Example:

CIDR: `10.0.1.0/24`

IPs: `2^(32 - 24) = 2^8 = 256`

Usable: `256 - 2 = 254`

2. Find Subnet Ranges Within a VPC

If your VPC is `10.0.0.0/16` and you want 4 subnets (`/18`), split as:

Subnet	CIDR	IP Range
Subnet A	10.0.0.0/18	10.0.0.0 – 10.0.63.255
Subnet B	10.0.64.0/18	10.0.64.0 – 10.0.127.255
Subnet C	10.0.128.0/18	10.0.128.0 – 10.0.191.255
Subnet D	10.0.192.0/18	10.0.192.0 – 10.0.255.255

So basically, you're carving up the space based on the bits you borrow.

Tools to Help

CIDR calculator online (like [SubnetOnline.com] (https://www.subnetonline.com))

AWS VPC wizard suggests CIDR automatically

Use Linux command:

ipcalc 10.0.1.0/24

AWS CIDR Restrictions

VPC CIDR: must be between `/16` to `/28`

Subnets cannot overlap

Cannot assign all IPs — 5 IPs reserved by AWS per subnet:

`x.x.x.0` → network address

`x.x.x.1` → VPC router

`x.x.x.2` → DNS

`x.x.x.3` → reserved

`x.x.x.255` → broadcast

Real-World Example: Subnet Planning in AWS

Goal: You have a `/16` VPC. You need:

2 public subnets

2 private subnets

Each in a different Availability Zone

Minimum 100 hosts per subnet

You can create 4 `/24` subnets:

Public-1: 10.0.1.0/24

Public-2: 10.0.2.0/24

Private-1: 10.0.3.0/24

Private-2: 10.0.4.0/24

Each has:

256 total IPs

251 usable (after AWS reserves)

Quick Exercise

You are given:

VPC CIDR: 192.168.0.0/16

Create 8 subnets each supporting at least 500 hosts.

Solution:

$2^9 = 512$ → So we need 9 host bits → `/23` subnets (32-9=23)

CIDR: `/23` = 512 IPs → 510 usable

Subnet CIDRs could be:

192.168.0.0/23

192.168.2.0/23

192.168.4.0/23

192.168.6.0/23

(you skip by 2 in the third octet)

Great question!

The CIDR block itself doesn't have bits per se — instead, it specifies how many bits are used for the network portion of an IP address.

So How Many Bits Is a CIDR Block?

Every IPv4 address is made of 32 bits.

A CIDR block uses this format:

<IP Address>/<prefix-length>

Example: 192.168.1.0/24

The `/24` means:

The first 24 bits are the network portion

The remaining 8 bits are for hosts

So:

`/8` → 8 bits for network, 24 for hosts

`/16` → 16 bits for network, 16 for hosts

`/32` → 32 bits for network, 0 for hosts (i.e., a single IP)

Visual Example: IP Address in Bits

Let's take `192.168.1.0/24`

Binary of 192.168.1.0:

11000000.10101000.00000001.00000000

| | |

8 bits + 8 bits + 8 bits = 24 network bits

So, `/24` = first 24 bits define the network, and 8 bits left for host IPs:

2^8 = 256 total IPs → 254 usable IPs

Summary:

CIDR	Network Bits	Host Bits	Total IPs	Usable IPs
/8	8	24	16,777,216	16,777,214
/16	16	16	65,536	65,534
/24	24	8	256	254
/32	32	0	1	0

So, the CIDR number (like `/24`) literally tells you how many bits of the 32-bit IPv4 address are used for the network portion.

CHEAT SHEET:

Absolutely! Here's your CIDR Cheat Sheet for IPv4 — it includes CIDR notation, subnet mask, number of IPs, and usable IPs.

Usable IPs = Total - 2, because AWS and networking protocols reserve:

1st: Network Address (e.g. `.0`)

Last: Broadcast Address (e.g. `.255`)

Quick Tips:

Use `/24` when you need \~250 hosts per subnet.

Use `/28` for small dev/test subnets.

AWS VPC CIDR must be between /16 to /28.

Want one IP (e.g., for EIP or security rules)? Use `/32`.

CIDR	Subnet Mask	Total IPs	Usable IPs	Host Bits
/8	255.0.0.0	16,777,216	16,777,214	24
/9	255.128.0.0	8,388,608	8,388,606	23
/10	255.192.0.0	4,194,304	4,194,302	22
/11	255.224.0.0	2,097,152	2,097,150	21
/12	255.240.0.0	1,048,576	1,048,574	20
/13	255.248.0.0	524,288	524,286	19
/14	255.252.0.0	262,144	262,142	18
/15	255.254.0.0	131,072	131,070	17
/16	255.255.0.0	65,536	65,534	16

/17	255.255.128.0	32,768	32,766	15
/18	255.255.192.0	16,384	16,382	14
/19	255.255.224.0	8,192	8,190	13
/20	255.255.240.0	4,096	4,094	12
/21	255.255.248.0	2,048	2,046	11
/22	255.255.252.0	1,024	1,022	10
/23	255.255.254.0	512	510	9
/24	255.255.255.0	256	254	8
/25	255.255.255.128	128	126	7
/26	255.255.255.192	64	62	6
/27	255.255.255.224	32	30	5
/28	255.255.255.240	16	14	4
/29	255.255.255.248	8	6	3
/30	255.255.255.252	4	2	2
/31	255.255.255.254	2	0 or 2 (point-to-point)	1
/32	255.255.255.255	1	0	0

What Are Subnets in AWS?

A subnet (sub-network) is a logical subdivision of a VPC's IP address range (CIDR block).

When you create a VPC, AWS asks you to divide it into smaller pieces — those are your subnets.

Each subnet:

Belongs to one Availability Zone (AZ).

Holds EC2 instances and other AWS resources.

Has a subset of the VPC's IP range.

Can be public or private.

Why Use Subnets?

Subnets help you:

Separate public-facing services from private/internal services.

Isolate workloads (e.g., frontend/backend).

Improve security using route tables and NACLs.

Control network flow using NAT, gateways, and firewalls.

Design for high availability across AZs.

Visual Example:

Let's say your VPC is:

CIDR: 10.0.0.0/16 (65,536 IPs)

You divide it into 4 subnets:

Subnet Name	CIDR Block	Type	AZ
public-sub-1	10.0.1.0/24	Public	us-east-1a
public-sub-2	10.0.2.0/24	Public	us-east-1b
private-sub-1	10.0.3.0/24	Private	us-east-1a
private-sub-2	10.0.4.0/24	Private	us-east-1b

Types of Subnets

1. Public Subnet

Can access the Internet directly.

Must have:

Route to Internet Gateway

EC2 instances with public IP

Used for: Web servers, NAT Gateways, bastion hosts.

2. Private Subnet

No direct Internet access

Used for backend DBs, app servers, Lambda, etc.

To connect out (e.g., install packages), use NAT Gateway

3. VPN/DB Subnet (optional)

Can be isolated for sensitive systems (e.g. database tier).

Subnet Components

Component	Description
CIDR Block	Defines the IP range of subnet
Availability Zone	Each subnet is tied to a single AZ
Route Table	Controls traffic routing rules
NACLs	Optional stateless firewall for subnet level
Auto-assign Public IP	Enabled for public subnets

How Subnets Work in AWS

When launching EC2, you choose a subnet.

Based on the route table, AWS decides whether that subnet:

Sends traffic to the Internet (via Internet Gateway)

Sends traffic to NAT Gateway (for private outbound)

Keeps traffic local (e.g., backend or database only)

Hands-On Example:

Create Subnets

Prerequisites:

A VPC with CIDR: `10.0.0.0/16`

2 AZs: `us-east-1a`, `us-east-1b`

Create 2 Public and 2 Private Subnets:

Name	CIDR	AZ	Type
public-subnet-a	10.0.1.0/24	us-east-1a	Public
public-subnet-b	10.0.2.0/24	us-east-1b	Public
private-subnet-a	10.0.3.0/24	us-east-1a	Private
private-subnet-b	10.0.4.0/24	us-east-1b	Private

In the AWS Console or CLI:

```
aws ec2 create-subnet --vpc-id <vpc-id> \
  --cidr-block 10.0.1.0/24 --availability-zone us-east-1a \
  --tag-specifications 'ResourceType=subnet,Tags=[{Key=Name,Value=public-subnet-a}]'
```

Repeat for others.

Subnet Design Best Practices

- At least 2 public + 2 private subnets in different AZs.
- Use `/24` or `/26` subnets for modular and scalable design.
- Avoid overlapping CIDRs.
- Reserve smaller subnets for internal or staging.

What is an Internet Gateway (IGW)?

An Internet Gateway (IGW) is a managed, horizontally scaled, redundant AWS component that allows instances in your VPC to:

- Send outbound traffic to the internet.
- Receive inbound traffic from the internet.

> Without an IGW, your VPC is completely private — even if your EC2 instance has a public IP.

Key Features of Internet Gateway

Feature	Description
Fully managed	AWS manages its availability, scaling, and durability
Horizontally scaled	Handles traffic without bandwidth bottlenecks
Redundant	Built to be highly available and fault tolerant
One per VPC	You can only attach **one IGW** per VPC
Used with public IPs	Required to reach internet-facing services from EC2

Real-Life Analogy

Think of a VPC as your office building and an IGW as your front gate:

If you don't have a gate, you can't go out or let others in.

But if you have a gate (IGW), only rooms (subnets) connected to it can communicate outside — provided they have permission (route table and public IP).

How Does IGW Work?

To enable internet access for an EC2 instance:

1. Attach IGW to your VPC.
2. Place instance in a subnet with route to IGW.
3. Assign a public IP to the instance.
4. Ensure Security Group & NACLs allow traffic (e.g. SSH on port 22, HTTP on port 80).

Traffic Flow Example:

EC2 (with public IP) → Route Table (with IGW route) → IGW → Internet

Required Components for IGW Access

Component	Purpose
IGW	Connects VPC to the internet
Public Subnet	Subnet with route to IGW
Route Table	Contains `0.0.0.0/0 → igw-xxxxxxxx`
Public IP	EC2 must have one for external access
Security Group	Must allow inbound/outbound traffic

Hands-On: Setup IGW

1. Create and Attach IGW

aws ec2 create-internet-gateway --tag-specifications \

'ResourceType=internet-gateway,Tags=[{Key=Name,Value=MyIGW}]'

Copy the `InternetGatewayId`.

aws ec2 attach-internet-gateway --vpc-id <vpc-id> \

--internet-gateway-id <igw-id>

2. Modify Route Table

Find the route table associated with your public subnet, then:

aws ec2 create-route \

  --route-table-id <rtb-id> \

  --destination-cidr-block 0.0.0.0/0 \

  --gateway-id <igw-id>

Security Concerns

While IGW provides internet access, it also opens your instances to public traffic. Secure it using:

Security Groups: Restrict inbound access (e.g., allow SSH only from your IP).

NACLs: Optional subnet-level firewall rules.

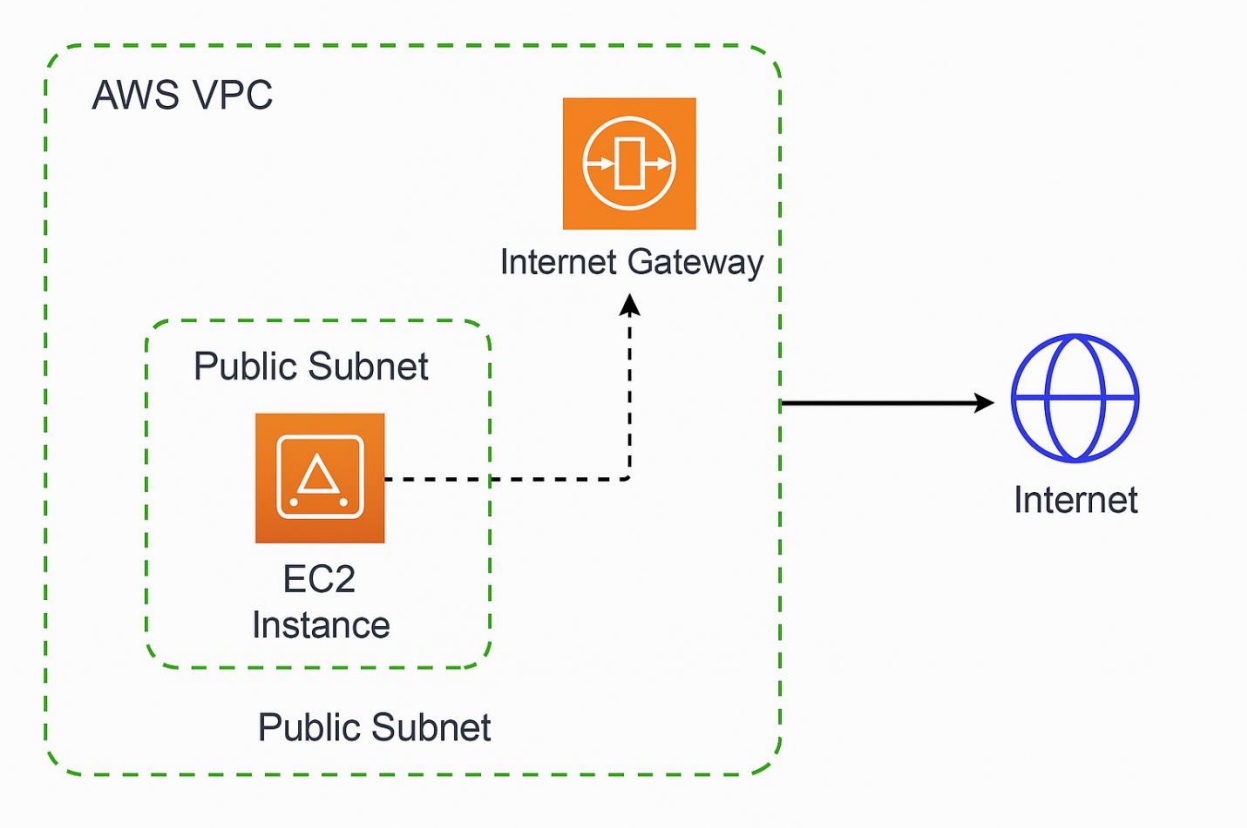
Bastion Host: Use this instead of giving all instances public IPs.

Best Practices:

Best Practice	Why?
Use IGW only with **public subnets**	Avoid unnecessary exposure
Use **NAT Gateway** for private subnets	Allows outbound only internet access
Secure EC2 with **least privilege SGs**	Prevent unwanted access
Tag resources properly	Easier management

Common Mistakes

Mistake	Result
EC2 in public subnet but no public IP	Cannot access internet
Route table missing IGW route	No internet even if public IP is present
IGW not attached to VPC	Route to IGW won’t work



NAT Gateway (and NAT Instance) — which allows private subnets to access the internet securely without being exposed.

What is a NAT Gateway / NAT Instance?

NAT stands for Network Address Translation.

A NAT Gateway (or NAT Instance) allows instances in private subnets to:

Access the internet for updates or external services.

Remain inaccessible from the internet (no inbound access allowed).

Outbound only internet access for private subnets.

Why NAT is Needed?

Let's say you have an EC2 instance in a private subnet (no route to Internet Gateway), but you want it to:

Download OS updates.

Install packages via `yum` or `apt`.

Send data to an external API.

You don't want to give it a public IP, so you use a NAT Gateway or NAT Instance to route outbound traffic securely.

How Traffic Flows with NAT Gateway

Private EC2 → Route Table → NAT Gateway (in public subnet) → IGW → Internet

The return traffic flows back through the NAT Gateway.

NAT Gateway vs NAT Instance

Feature	NAT Gateway	NAT Instance
Managed by AWS	Yes	No (you manage EC2 manually)
High Availability	Automatic in AZ	You must configure manually
Performance	Up to 45 Gbps	Limited to EC2 instance type
Setup Effort	Simple	Complex
Auto Scaling	Built-in	Needs custom solution
Cost	Higher	Lower (for small workloads)
Security Groups	Not needed	Needed (like any EC2)

Use NAT Gateway for production.

Use NAT Instance for cost-saving in testing/dev.

Hands-On: NAT Gateway Setup (Step-by-Step)

Prerequisites:

A VPC with CIDR `10.0.0.0/16`

2 subnets:

Public Subnet: `10.0.1.0/24`

Private Subnet: `10.0.2.0/24`

IGW already attached

1. Create NAT Gateway

aws ec2 allocate-address --domain vpc

Take note of the `AllocationId`.

```
aws ec2 create-nat-gateway \
  --subnet-id <public-subnet-id> \
  --allocation-id <eip-alloc-id> \
  --tag-specifications 'ResourceType=natgateway,Tags=[{Key=Name,Value=MyNATGW}]'
```

2. Modify Route Table of Private Subnet

Find the route table associated with the private subnet, and add:

aws ec2 create-route \

--route-table-id <rtb-private-id> \

--destination-cidr-block 0.0.0.0/0 \

--nat-gateway-id <natgw-id>

NAT Instance Setup (Alternate Approach)

1. Launch an EC2 instance (Amazon Linux 2 or NAT AMI).

2. Place it in a public subnet with:

Public IP

Route to IGW

3. Enable IP forwarding:

echo 1 > /proc/sys/net/ipv4/ip_forward

4. Modify security group to allow:

Inbound from private subnet

Outbound to anywhere

5. Update route table of private subnet:

aws ec2 create-route \

--route-table-id <rtb-private-id> \

--destination-cidr-block 0.0.0.0/0 \

--instance-id <nat-ec2-instance-id>

Requires manual management and scaling.

Best Practices

Use 1 NAT Gateway per AZ for high availability.

Don’t put NAT Gateway in a private subnet.

Set detailed CloudWatch alarms for traffic or failures.

Prefer NAT Gateway unless you must cut costs.

Summary Cheat:

Subnet Type	Public IP?	IGW Access	NAT Needed?	Use Case
Public	Yes	Yes	No	Web servers, Bastions
Private	No	No	Yes	DBs, app servers, Lambda

Awesome! Let's now deep dive into the Route Table— one of the most critical components in AWS VPC networking that controls how traffic flows within your VPC and to the internet.

What is a Route Table in AWS?

A Route Table in AWS VPC is like a GPS for your subnet. It tells the traffic:

Where to go (destination IP)

How to get there (target: IGW, NAT, local, etc.)

Each subnet must be associated with one route table, and it uses that to determine where to send traffic.

Anatomy of a Route Table

A route table is made up of routes, and each route has:

Field	Description
Destination	The IP range of the destination (CIDR block)
Target	The next hop or gateway to reach that destination

Example Route Table

Destination	Target	Description
10.0.0.0/16	local	Traffic within the VPC
0.0.0.0/0	igw-xxxxxxx	Public internet traffic
0.0.0.0/0	nat-xxxxxxx	Private subnet to internet (via NAT)
172.31.0.0/16	pcx-xxxxxxx	Peered VPC

Key Concepts

1. Local Route

Automatically created when you create a VPC.

Enables communication between subnets inside the VPC.

2. Internet Gateway Route

Required in public subnets to send/receive traffic from internet.

Example:

Destination: 0.0.0.0/0

Target: igw-xxxxxxxx

3. NAT Gateway Route

Allows private subnets to access the internet for outbound only.

Example:

Destination: 0.0.0.0/0

Target: **nat-xxxxxxxx**

4. VPC Peering Route

Enables traffic between two VPCs (peered).

Example:

Destination: 10.1.0.0/16

Target: pcx-xxxxxxxx

Route Table Configuration Options

Use Case	Route Table Entry Example
Internet access from public subnet	`0.0.0.0/0 → igw-xxxxx`
Private subnet access internet	`0.0.0.0/0 → nat-xxxxx`
VPC-to-VPC communication (peering)	`10.1.0.0/16 → pcx-xxxxx`
VPN traffic	`192.168.0.0/16 → vpn-xxxxx`

Hands-On CLI Example: Route Table for Public Subnet

Step 1: Create Route Table

```
aws ec2 create-route-table \
  --vpc-id <vpc-id> \
  --tag-specifications 'ResourceType=route-table,Tags=[{Key=Name,Value=PublicRT}]'
```

Step 2: Add Route to IGW

```
aws ec2 create-route \
  --route-table-id <rtb-id> \
  --destination-cidr-block 0.0.0.0/0 \
  --gateway-id <igw-id>
```

Step 3: Associate with Subnet

```
aws ec2 associate-route-table \
  --route-table-id <rtb-id> \
  --subnet-id <public-subnet-id>
```

Important Notes

Concept	Notes
Route Table ≠ Subnet	You can have one route table for **multiple subnets**
Explicit Association	If not associated, the subnet uses **main route table**
Main Route Table	Default table attached to all subnets (can be edited)
Longest Prefix Match	AWS uses **most specific match** when multiple routes

Best Practices

Name your route tables properly (e.g., `public-rt`, `private-rt`)

Use separate route tables for public and private subnets

Don’t modify main route table blindly — clone and use custom tables

Always test routes using:

curl ifconfig.me

Troubleshooting Tips

Problem	Reason
EC2 has public IP but no internet	No route to IGW
Private EC2 cannot update packages	No NAT route or NAT misconfigured
Cross-VPC not working	Missing peering route

Real-Life Scenario Example

You create:

VPC: `10.0.0.0/16`

Public Subnet: `10.0.1.0/24`

Private Subnet: `10.0.2.0/24`

IGW attached

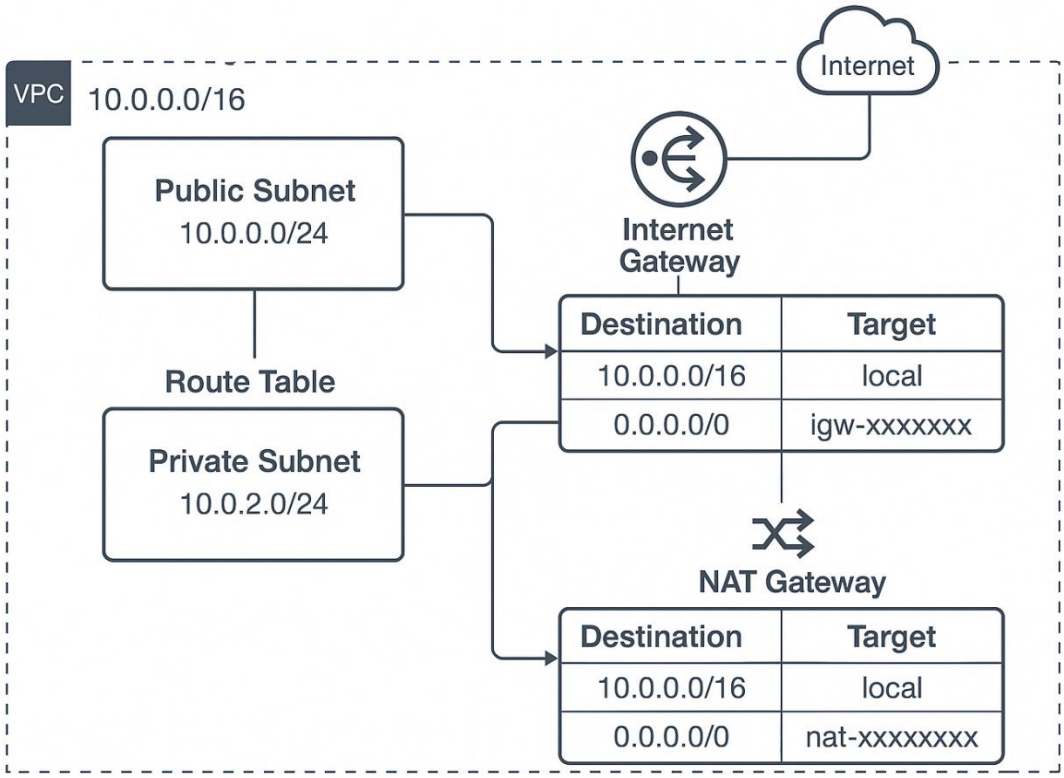
NAT Gateway in public subnet

Route Table 1: For Public Subnet

Destination	Target
10.0.0.0/16	local
0.0.0.0/0	igw-xxxxx

Route Table 2: For Private Subnet

Destination	Target
10.0.0.0/16	local
0.0.0.0/0	nat-xxxxx



What is a Security Group?

A Security Group (SG) acts as a virtual firewall for your EC2 instances to control inbound and outbound traffic at the instance level.

Key Characteristics

Feature	Description
Stateful	If you allow inbound, the response is automatically allowed outbound
Instance-Level	Applied directly to EC2 instances (unlike NACLs which are subnet-level)
Default Deny	Denies all inbound traffic by default, allows all outbound
Rules are Permissive	You can **only allow** traffic; there is **no deny** rule
Multiple SGs	You can attach multiple SGs to one instance
Regional	Security Groups are specific to a VPC in a Region

Components of a Security Group Rule

Component	Description
Type	e.g. SSH, HTTP, HTTPS, Custom TCP
Protocol	TCP, UDP, ICMP, etc.
Port Range	22 for SSH, 80 for HTTP, etc.
Source/Dest	IP range (CIDR), SG, or `Anywhere`
Description	Optional description of the rule

Inbound vs Outbound

Inbound Rules: Define what traffic is allowed into the instance.

Outbound Rules: Define what traffic is allowed to leave the instance.

Common Use Case Examples

Use Case	Rule
SSH from personal IP	Inbound: TCP 22 from `203.x.x.x/32`
Public web server	Inbound: TCP 80, 443 from `0.0.0.0/0`
App to DB communication	Inbound: TCP 3306 from App SG
EC2 to Internet	Outbound: `0.0.0.0/0` (default rule)

Hands-On via AWS Console

Create a Security Group:

1. Go to VPC > Security Groups

2. Click Create Security Group

3. Add a name, description, and select VPC

Add Rules:

Inbound Rule:

Type = HTTP | Protocol = TCP | Port = 80 | Source = `0.0.0.0/0`

Outbound Rule:

Leave default (`All traffic → 0.0.0.0/0`)

Attach to EC2:

While launching, go to "Configure Security Group"

Choose existing or create a new SG

Common Mistakes

Mistake	Fix
No inbound rule for SSH	Add TCP 22 rule for your IP
Wrong source CIDR	Use correct CIDR (like `/32` for single IP)
Confused SG with NACL	SGs are for instances; NACLs are for subnets
Forgot to open app port	Add correct inbound rule (e.g., TCP 8080)

Best Practices

Restrict SSH access to known IPs only

Group rules logically (e.g., web-sg, db-sg)

Use security group references (e.g., app-sg → db-sg)

Monitor and clean unused SGs regularly

Real-World Example

Imagine a 3-tier architecture:

Tier	Port	Source
Web Server	80/443	`0.0.0.0/0`
App Server	8080	Web SG
DB Server	3306	App SG

You create 3 SGs: `web-sg`, `app-sg`, `db-sg` and reference each other accordingly.

What is a NACL (Network Access Control List)?

A Network ACL is a stateless firewall at the subnet level in AWS.

It controls inbound and outbound traffic to/from resources (like EC2) in the subnet.

Key Differences: NACL vs Security Group

Feature	Security Group	NACL
Scope	Instance-level	Subnet-level
Stateful?	✅ Yes (automatically allows responses)	❌ No (return traffic must be explicitly allowed)
Rules Type	Only **Allow**	Both **Allow and Deny**
Rule Evaluation Order	All rules evaluated	Rules evaluated in **numbered order**
Applied To	EC2 instances	All resources in the subnet
Default Behavior	All inbound denied, all outbound allowed	All inbound and outbound denied (except default NACL)

NACL Rule Structure

Field	Description
Rule #	Priority (lowest number wins)
Type	e.g. HTTP, SSH, Custom TCP
Protocol	e.g. TCP, UDP, ICMP
Port Range	e.g. 22, 80, 1024-65535
Source/Destination	IP CIDR (e.g. `0.0.0.0/0`, `10.0.0.0/16`)
Allow/Deny	Action to perform

Inbound + Outbound Rules

Both directions must be explicitly allowed for two-way communication.

Example:

To SSH into EC2:

Inbound Rule: Allow TCP 22 from `YOUR_IP/32`

Outbound Rule: Allow TCP 1024-65535 to `YOUR_IP/32`

(SSH response comes from high-numbered ephemeral ports)

Default NACL vs Custom NACL:

NACL Type	Behavior
Default	Allows all inbound and outbound traffic
Custom	Denies all traffic by default

Example NACL Rule Set (Public Subnet)

Inbound Rules

Rule #	Type	Port	Source	Action
100	HTTP	80	0.0.0.0/0	Allow
110	SSH	22	Your IP/32	Allow
	All	All	All	Deny

Outbound Rules

Rule #	Type	Port	Destination	Action
100	All TCP	1024–65535	0.0.0.0/0	Allow
	All	All	All	Deny

How to Create NACL (Console)

1. Go to VPC > Network ACLs
2. Click Create Network ACL
3. Select VPC, give it a name
4. Add Inbound and Outbound Rules
5. Associate with a Subnet

CLI Commands

Create NACL

aws ec2 create-network-acl --vpc-id <vpc-id> --tag-specifications 'ResourceType=network-acl,Tags=[{Key=Name,Value=MyNACL}]'

Add Inbound Rule

```
aws ec2 create-network-acl-entry \
  --network-acl-id <nacl-id> \
  --rule-number 100 \
  --protocol tcp \
  --rule-action allow \
  --egress false \
  --cidr-block 0.0.0.0/0 \
  --port-range From=80,To=80

# Associate with subnet
aws ec2 associate-network-acl \
  --subnet-id <subnet-id> \
  --network-acl-id <nacl-id
```

Best Practices

Use DENY rules for blocking specific IPs (not possible with SGs).

Always add outbound + inbound rules for traffic to work.

Put DENY rules at a lower number if you want to block before ALLOW.

Use NACL for additional layer of subnet security, especially public/private boundaries.

Real-Life Scenario

You want a public subnet where EC2 can:

Accept HTTP traffic from anywhere

SSH only from your IP

Go out to the internet

You configure:

Inbound: Allow TCP 80 from `0.0.0.0/0`, TCP 22 from `your IP/32`

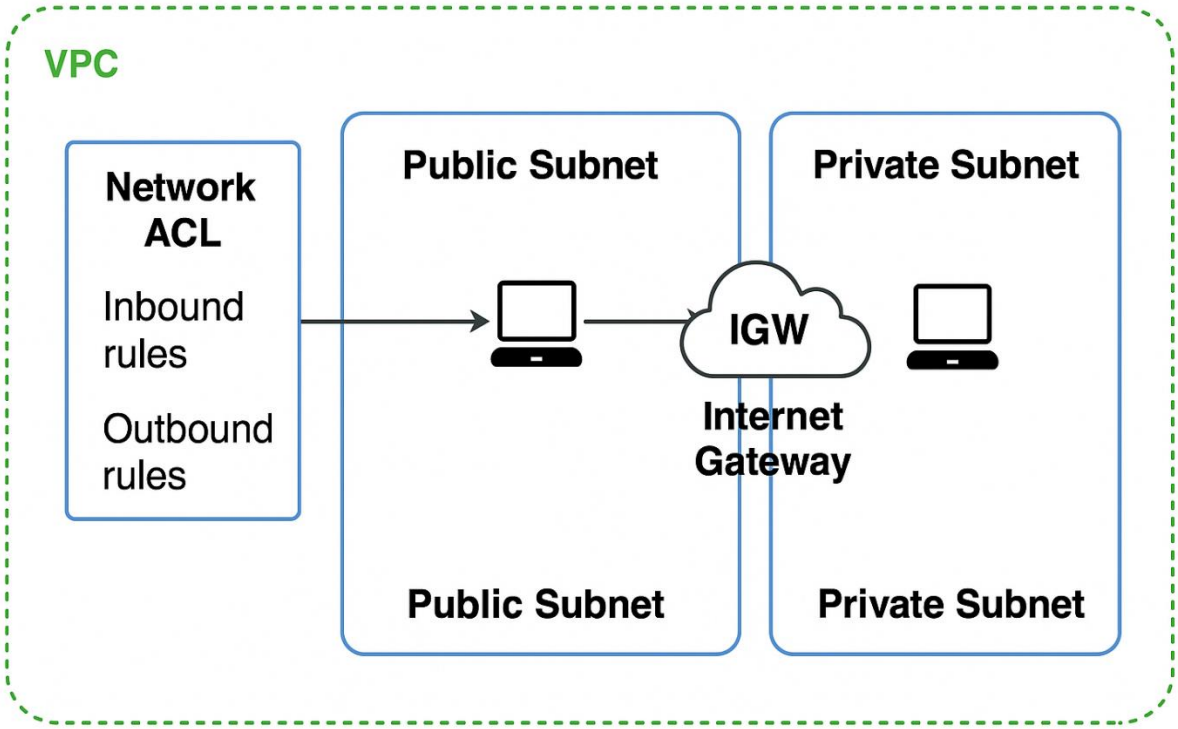
Outbound: Allow TCP 1024–65535 to `0.0.0.0/0`

Troubleshooting Tips

Symptom	Possible Issue
Cannot SSH	Missing outbound rule or NACL denies
HTTP works one way only	Missing return path in outbound
Security Group correct but no access	NACL blocking traffic

NACL vs Security Group: When to Use

Use Case	Recommended Tool
Instance-level access control	**Security Group**
Subnet-wide blacklisting	**NACL**
Logging blocked IPs	**VPC Flow Logs** + **NACL**
Defense-in-depth	Both SG + NACL


VPC
Network ACL
Inbound rules
Outbound rules
Public Subnet
Private Subnet
IGW
Internet Gateway
Public Subnet
Private Subnet

VPC Peering:

VPC Peering — a powerful feature in AWS networking that allows private connectivity between two VPCs.

VPC Peering is a network connection between two Virtual Private Clouds (VPCs) that enables you to route traffic between them using private IPs. It can be used within a region or across regions.

No need for Internet Gateway, NAT, or VPN for traffic

Peering is not transitive (VPC A ↔ B and B ↔ C doesn't mean A ↔ C)

Use Cases

Microservices spread across different VPCs

VPC isolation by business unit or project

Cross-region backup or disaster recovery

Shared services VPC (e.g., central logging, DNS)

VPC Peering Architecture

Example:

| VPC A (10.0.0.0/16) | <---> | VPC B (192.168.0.0/16) |

Communication happens over AWS backbone

No NAT or IGW needed

Each VPC updates its route table to direct traffic to the peer

Rules & Limitations

Rule / Limit	Details
IP Overlap	CIDR blocks **must not overlap**
Peering is not transitive	You need a direct peering for each pair
Max Peering Connections	125 per VPC (can be increased)
Security Group consideration	Must allow inbound/outbound traffic from the peer VPC CIDR
Cannot use for DNS resolution (by default)	Unless **DNS resolution** is enabled manually via AWS CLI

How VPC Peering Works (Steps)

Step 1: Create Peering Request

From VPC A to VPC B

```
aws ec2 create-vpc-peering-connection \
  --vpc-id vpc-11111111 \
  --peer-vpc-id vpc-22222222 \
  --peer-region us-west-2 \
  --tag-specifications 'ResourceType=vpc-peering-connection,
Tags=[{Key=Name,Value=MyPeering}]'
```

Step 2: Accept Peering Request

```
aws ec2 accept-vpc-peering-connection \
  --vpc-peering-connection-id pcx-abc123
```

Step 3: Add Route to Route Tables

In VPC A Route Table:

Destination: 192.168.0.0/16 → Target: pcx-abc123

In VPC B Route Table:

Destination: 10.0.0.0/16 → Target: pcx-abc123

Step 4: Modify Security Groups

Allow traffic from the peer VPC's CIDR range

Cross-Region Peering

Yes, VPC peering supports cross-region communication.

Slightly higher latency

No data transfer over public internet

Cost: Data transfer charges apply (e.g., \$0.01–0.02/GB)

Security Best Practices

Least privilege Security Group rules

Use NACLs to segment traffic further if needed

Monitor traffic via VPC Flow Logs

Use resource tags to track peering connections

Troubleshooting

Issue	Check
Ping/SSH not working	Security Group / NACL blocking traffic
Connection not appearing	Check that both VPCs are in "available" state
DNS not working across VPCs	DNS resolution not enabled
Still no route	Route table not updated for destination CIDR

VPC Peering vs Transit Gateway

Feature	VPC Peering	Transit Gateway
Transitive routing	Not supported	Supported
Scalability	Limited (125)	Highly scalable
Centralized routing	No	Yes
Cost	Cheaper for few VPCs	Better for many VPCs