NETWORKING FUNDAMENTALS FOR DEVOPS

WHAT IS AN IP ADDRESS?

Definition: IP Address (Internet Protocol Address) is used to generate or provide a **unique address** to a device connected to a network.

Purpose:

- Identifies devices on a network
- Enables communication between devices
- Routes data to correct destination

Analogy: Just like your home has a postal address so mail can reach you, every device on a network needs an IP address so data can reach it.

IP ADDRESS VERSIONS

Two types:

1. **IPv4** (Internet Protocol version 4)
2. **IPv6** (Internet Protocol version 6)

Why two versions? We can generate a huge number of unique addresses using these protocols to accommodate all devices worldwide.

IPv4 ADDRESS FORMAT

Structure:

A.B.C.D

Where:

- A, B, C, D are numbers
- Each can vary from **0 to 255**
- Each represents **1 byte (8 bits)**

Examples:

192.168.1.1

172.16.0.5

10.0.0.100

8.8.8.8 (Google DNS)

Total representation:

(0-255).(0-255).(0-255).(0-255)

This is the **IPv4 standard**.

WHY 0-255 RANGE?

Technical explanation:

Each section represents **8 bits (1 byte)**

Binary to Decimal:

- 8 bits can represent: 2^8 = 256 different values
- Range: 0 to 255 (256 total values including 0)

Example:

```
IP: 192.168.1.1


In Binary:

192 = 11000000 (8 bits)

168 = 10101000 (8 bits)

1   = 00000001 (8 bits)

1   = 00000001 (8 bits)


Total: 32 bits (4 bytes)
```

Computer understanding: Computers only understand bits (0s and 1s), so:

- Human readable: 192.168.1.1
- Computer readable: 11000000.10101000.00000001.00000001

IPv4 CAPACITY

Total possible IPv4 addresses:

2^32 = 4,294,967,296 addresses

(approximately 4.3 billion)

Problem: With billions of devices (phones, computers, IoT devices), we're running out of IPv4 addresses!

Solution: IPv6 was created with 2^128 addresses (340 undecillion addresses!)

IPv6 ADDRESS FORMAT

Structure:

xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx

Example:

2001:0db8:85a3:0000:0000:8a2e:0370:7334

Characteristics:

- 8 groups of 4 hexadecimal digits
- Each group represents 16 bits
- Total: 128 bits
- Provides 340,282,366,920,938,463,463,374,607,431,768,211,456 addresses!

Note: We'll focus on IPv4 as it's still most commonly used.

WHAT IS A SUBNET?

Subnet = Sub-network = A smaller network within a larger network

---

REAL-WORLD SCENARIO: SCHOOL NETWORK

Setup:

- You have a school network
- You have a VPC on AWS
- You request **65,000 IP addresses**

- Maximum people connecting to your network: **40,000**

Initial setup: All devices connected to **one single network** - everyone can access everyone.

THE SECURITY PROBLEM

Scenario:

1. One person accesses a **malicious website** written by a hacker
2. Hacker gains access to **that person's device**
3. Since all devices are on the **same network**, hacker now has access to **ALL devices**!

In an office network:

- Sensitive financial data
- Employee personal information
- Company secrets
- Customer data

Result: Hacker gets access to **EVERYTHING** 🚨

SOLUTION: SUBNETTING (SUB-NETWORKING)

Concept: Split your large network into **smaller, isolated networks** (subnets)

Example: Office Network Split

Network 1 (Finance Subnet):

- Secure network
- Only finance team access
- Contains sensitive financial data
- Isolated from other departments

Network 2 (General Subnet):

- Can be accessed by others
- General office use

- Internet access
- Less sensitive data

Key point: Network 1 and Network 2 are **subnets** of the main network.

WHY SUBNETTING IS IMPORTANT

Three main reasons:

1. Security:

- Isolate sensitive data
- Limit blast radius of security breach
- Control access between departments

2. Privacy:

- Finance can't see HR data
- HR can't see Engineering data
- Each department isolated

3. Isolation:

- If one subnet is compromised, others remain safe
- Problems in one subnet don't affect others
- Better network management

SUBNETTING IN ANY NETWORK

We can create subnets in any type of network:

- Home network
- Office network
- School/University network
- Cloud network (AWS VPC, Azure VNet, GCP VPC)
- Data center network

TYPES OF SUBNETS

Two main types:

1. Private Subnet:

- **No direct access to internet**
- Internal communication only
- More secure
- Example: Database servers, internal applications

2. Public Subnet:

- **Has access to internet**
- Can send and receive data from internet
- Example: Web servers, load balancers, API gateways

EXAMPLE: OFFICE NETWORK ARCHITECTURE

Main Office Network (65,000 IP addresses)

```
   |
   |--- Finance Subnet (Private) - 256 IPs
   |     - No internet access
   |     - Only finance team
   |     - Sensitive data
   |
   |--- General Subnet (Public) - 64,744 IPs
         - Internet access
         - All employees
         - General office work
```

THE IP ADDRESS ALLOCATION QUESTION

Problem:

- Finance subnet needs only **256 IP addresses**
- General subnet needs the **rest (~64,744 IP addresses)**

Question: How do you specify this when creating subnets?

Answer: Using **CIDR (Classless Inter-Domain Routing) range**

WHAT IS CIDR?

CIDR = Classless Inter-Domain Routing

Purpose: Method to allocate IP addresses and specify how many IPs a subnet should have.

Format:

IP_ADDRESS/PREFIX_LENGTH

Example:

192.168.1.0/24

172.16.0.0/16

10.0.0.0/8

HOW CIDR WORKS - STEP BY STEP

Step 1: Pick a range of IP addresses

Let's pick:

172.16.0.0 to 172.16.255.255

This gives us **65,536 IP addresses** (256 × 256)

Step 2: Understanding Octet Format

Any IP address can be represented in octet format:

```
172   .   16   .   0   .   0
|        |        |        |
8 bits    8 bits    8 bits    8 bits
(1 byte)  (1 byte)  (1 byte)  (1 byte)


Total: 32 bits (4 bytes)
```

Step 3: Finance Subnet Needs 256 IP Addresses

Observation: We need only **256 IP addresses** for finance department.

Key insight: 256 = 2^8

We can vary only the **last 8 bits** (last octet) to get 256 different addresses.

This means:

- First 3 octets: **FIXED** (same for all devices)
- Last octet: **VARIES** (0 to 255)

Step 4: Assigning IP Range to Finance Subnet

Multiple ways to represent:

Option 1: Range format

172.16.3.0 - 172.16.3.255

Option 2: Another range

172.16.4.0 - 172.16.4.255

Option 3: CIDR notation (PREFERRED)

172.16.3.0/24

All represent the same thing: 256 IP addresses where first 24 bits are fixed.

UNDERSTANDING CIDR NOTATION

Format:

IP_ADDRESS/PREFIX_LENGTH

Example:

172.16.3.0/24

Breaking it down:

172.16.3.0: Starting IP address (network address)

/24: Prefix length = **24 bits are fixed** (network portion)

Calculation:

- Total bits in IPv4: 32
- Fixed bits (network): 24
- Variable bits (host): 32 - 24 = 8
- Number of addresses: 2^8 = 256

CIDR NOTATION EXAMPLES

Example 1: /24 (256 addresses)

```
172.16.3.0/24

Fixed: 172.16.3.___
Variable: Last octet (0-255)
Total IPs: 2^(32-24) = 2^8 = 256
```

Example 2: /31 (2 addresses)

```
172.16.3.0/31

Fixed: First 31 bits
Variable: Last 1 bit
Total IPs: 2^(32-31) = 2^1 = 2
```

Why only 2 IPs? If we want only **2 IP addresses** for financial subnet:

- 2 = 2^1
- We need to vary only 1 bit
- So prefix length = 32 - 1 = 31
- CIDR: 172.16.3.0/31

CIDR CALCULATION FORMULA

Formula:

```
Number of IP addresses = 2^(32 - PREFIX_LENGTH)
```

Examples:

CIDR	Prefix	Variable Bits	Number of IPs	Use Case
/32	32	0	2^0 = 1	Single host
/31	31	1	2^1 = 2	Point-to-point links
/30	30	2	2^2 = 4	Small subnet
/29	29	3	2^3 = 8	Very small subnet
/28	28	4	2^4 = 16	Small office
/27	27	5	2^5 = 32	Small office
/26	26	6	2^6 = 64	Medium office
/25	25	7	2^7 = 128	Medium office
/24	24	8	2^8 = 256	Standard subnet
/23	23	9	2^9 = 512	Large subnet
/22	22	10	2^10 = 1,024	Large subnet
/21	21	11	2^11 = 2,048	Very large subnet
/20	20	12	2^12 = 4,096	Very large subnet
/16	16	16	2^16 = 65,536	Class B network
/8	8	24	2^24 = 16,777,216	Class A network

PRACTICAL CIDR EXAMPLES

Example 1: Finance Department (256 IPs)

CIDR: 172.16.3.0/24

IP Range:

172.16.3.0 (Network address - not usable)

172.16.3.1 (First usable)

172.16.3.2

...

172.16.3.254 (Last usable)

172.16.3.255 (Broadcast address - not usable)

Usable IPs: 254 (256 - 2)

Why subtract 2?

- First IP: Network address (reserved)
- Last IP: Broadcast address (reserved)

Example 2: General Department (64,000 IPs)

CIDR: 172.16.0.0/16

IP Range:

172.16.0.0 - 172.16.255.255

Total IPs: 2^(32-16) = 2^16 = 65,536

Usable IPs: 65,534

Example 3: Very Small Subnet (4 IPs)

CIDR: 192.168.1.0/30

IP Range:

192.168.1.0 (Network address)

192.168.1.1 (Usable)

192.168.1.2 (Usable)

192.168.1.3 (Broadcast address)

Total: 4 IPs

Usable: 2 IPs

PRIVATE IP ADDRESS RANGES

Important observation: Whenever we create **private subnets**, we see the first value mostly as:

- **192**
- **172**
- **10**

Why?

These are the numbers **reserved for private networks** according to RFC 1918.

STANDARD PRIVATE IP RANGES

Class A Private Range:

10.0.0.0 - 10.255.255.255

CIDR: 10.0.0.0/8

Total IPs: 16,777,216

Used for: Very large private networks

Class B Private Range:

172.16.0.0 - 172.31.255.255

CIDR: 172.16.0.0/12

Total IPs: 1,048,576

Used for: Medium to large private networks (AWS default VPC uses this)

Class C Private Range:

192.168.0.0 - 192.168.255.255

CIDR: 192.168.0.0/16

Total IPs: 65,536

Used for: Small private networks (home networks, small offices)

WHY THESE SPECIFIC RANGES?

Public vs Private IPs:

Public IPs:

- Routable on internet
- Must be unique globally
- Assigned by ISPs
- Cost money

Private IPs:

- NOT routable on internet
- Can be reused in different private networks
- Free to use
- Must use NAT gateway to access internet

The ranges 10.x.x.x, 172.16-31.x.x, and 192.168.x.x are reserved for private use only!

EXAMPLE: AWS VPC WITH SUBNETS

Creating VPC on AWS:

Step 1: Create VPC

VPC CIDR: 172.16.0.0/16

Total IPs: 65,536

Step 2: Create Subnets

Finance Subnet (Private):

CIDR: 172.16.1.0/24

IPs: 256

Internet: No

General Subnet (Public):

CIDR: 172.16.2.0/24

IPs: 256

Internet: Yes

Database Subnet (Private):

CIDR: 172.16.3.0/24

IPs: 256

Internet: No

Web Subnet (Public):

CIDR: 172.16.4.0/24

IPs: 256

Internet: Yes

VISUAL REPRESENTATION

VPC: 172.16.0.0/16 (65,536 IPs)

├── Finance Subnet: 172.16.1.0/24 (256 IPs) [Private]

├── General Subnet: 172.16.2.0/24 (256 IPs) [Public]

├── Database Subnet: 172.16.3.0/24 (256 IPs) [Private]

└── Web Subnet: 172.16.4.0/24 (256 IPs) [Public]

WHAT ARE PORTS?

Definition: A **port** is a number that binds our application to a unique identifier associated with that application.

Purpose: Using ports, we can distinguish **which request should be forwarded to which application**.

WHY DO WE NEED PORTS?

Scenario: One server (one IP address) running multiple applications:

- Web server
- Database server
- Email server

- FTP server

Problem: If all applications use the same IP, how does the server know which application should handle an incoming request?

Solution: Each application listens on a **different port number**.

IP ADDRESS + PORT COMBINATION

Format:

IP_ADDRESS:PORT

Examples:

```
192.168.1.100:80    (Web server)
192.168.1.100:443   (HTTPS web server)
192.168.1.100:3306  (MySQL database)
192.168.1.100:22    (SSH)
192.168.1.100:25    (Email server)
```

Same IP, different ports = different applications!

HOW PORTS WORK

Analogy:

Think of an IP address as an **apartment building address**.

Think of a port as an **apartment number**.

```
Building: 123 Main Street (IP: 192.168.1.100)
 ├── Apartment 80: Web Server
 ├── Apartment 443: HTTPS Server
 ├── Apartment 22: SSH Server
└── Apartment 3306: Database Server
```

When data arrives at 192.168.1.100:80, it knows to go to the web server apartment.

PORT NUMBER RANGE

Total available ports:

0 - 65535 (2^16 = 65,536 ports)

Port categories:

1. Well-Known Ports (0-1023):

- Reserved for system/common services
- Require admin/root privileges
- Examples: HTTP (80), HTTPS (443), SSH (22)

2. Registered Ports (1024-49151):

- Used by specific applications
- Examples: MySQL (3306), PostgreSQL (5432)

3. Dynamic/Private Ports (49152-65535):

- Used for temporary connections
- Automatically assigned by OS

COMMON PORT NUMBERS

Port	Protocol	Service	Description
20	FTP	FTP Data	File transfer data
21	FTP	FTP Control	File transfer control
22	SSH	Secure Shell	Remote login, secure file transfer
23	Telnet	Telnet	Unencrypted remote login
25	SMTP	Email	Sending emails
53	DNS	Domain Name System	Resolving domain names
80	HTTP	Web Server	Unencrypted web traffic
110	POP3	Email	Receiving emails
143	IMAP	Email	Receiving emails
443	HTTPS	Secure Web	Encrypted web traffic

Port	Protocol	Service	Description
3306	MySQL	Database	MySQL database
3389	RDP	Remote Desktop	Windows remote desktop
5432	PostgreSQL	Database	PostgreSQL database
6379	Redis	Cache	Redis cache server
8080	HTTP	Alt Web	Alternative web port
27017	MongoDB	Database	MongoDB database

PRACTICAL EXAMPLE: WEB SERVER

Scenario: You have a server at IP: 54.123.45.67

Multiple applications running:

1. Main website:

http://54.123.45.67:80

Browser automatically uses port 80 for HTTP

2. Admin panel:

https://54.123.45.67:443

Browser automatically uses port 443 for HTTPS

3. SSH access:

ssh ubuntu@54.123.45.67:22

SSH client uses port 22

4. Database:

mysql://54.123.45.67:3306

MySQL client connects to port 3306

HOW PORT FORWARDING WORKS

Step 1: Request arrives at server

Incoming request: 54.123.45.67:80

Step 2: Server checks port number

Port 80? → Forward to Web Server application

Step 3: Application processes request

Web Server receives request → Sends response

Step 4: Response sent back

Response sent from: 54.123.45.67:80

REAL-WORLD DEVOPS EXAMPLE

AWS Security Groups:

When you create an EC2 instance, you configure security group rules:

Inbound Rules:

Allow:

- Port 22 (SSH) from 0.0.0.0/0

- Port 80 (HTTP) from 0.0.0.0/0

- Port 443 (HTTPS) from 0.0.0.0/0

- Port 3306 (MySQL) from 172.16.1.0/24 (only from finance subnet)

This means:

- Anyone can access web server (port 80, 443)
- Anyone can SSH (port 22) - should restrict this!
- Only finance subnet can access database (port 3306)

PORT + SUBNET SECURITY EXAMPLE

Architecture:

Public Subnet (172.16.1.0/24):

├── Web Server: 172.16.1.10

│   └── Ports: 80 (HTTP), 443 (HTTPS) - Open to internet

│

Private Subnet (172.16.2.0/24):

```
 ├── Database Server: 172.16.2.20
│   └── Port: 3306 (MySQL) - Only accessible from public subnet
│
 ├── Finance App: 172.16.2.30
   └── Port: 8080 - Only accessible from finance team IPs
```

Security rules:

1. Internet can reach 172.16.1.10:80 and :443
2. Internet CANNOT reach 172.16.2.20:3306
3. Web server CAN reach database on :3306
4. Only finance subnet IPs can reach :8080

CHECKING WHICH PORTS ARE OPEN

On Linux:

Command 1: netstat

bash

```
netstat -tuln
```

Shows all listening ports

Command 2: ss

bash

```
ss -tuln
```

Modern replacement for netstat

Command 3: lsof

bash

```
sudo lsof -i -P -n | grep LISTEN
```

` ` `

Lists all listening ports with processes

Example output:

` ` `

```
Proto  Local Address        State
tcp   0.0.0.0:22          LISTEN  (SSH)
tcp   0.0.0.0:80          LISTEN  (Apache)
tcp   127.0.0.1:3306       LISTEN  (MySQL)
```

CHECKING FROM OUTSIDE (PORT SCANNING)

Command: nmap

bash

```
nmap 54.123.45.67
```

` ` `

Example output:

` ` `

```
PORT    STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http
443/tcp  open  https
3306/tcp closed mysql
```

Note: Port scanning without permission is illegal!

TESTING IF PORT IS OPEN

Command: telnet

bash

```
telnet 54.123.45.67 80
```

If connection succeeds, port is open.

Command: nc (netcat)

bash

nc -zv 54.123.45.67 80

Command: curl

bash

curl -v telnet://54.123.45.67:80

` ` `

PORT IN URLs

Full URL format:

` ` `

protocol://hostname:port/path

` ` `

Examples:

With explicit port:

` ` `

http://example.com:8080/api/users

https://192.168.1.100:443/admin

` ` `

Without explicit port (uses default):

` ` `

http://example.com → http://example.com:80

https://example.com → https://example.com:443

` ` `

Default ports:

- HTTP → 80

- HTTPS → 443

- FTP → 21

- SSH → 22

DEVOPS USE CASE: APPLICATION DEPLOYMENT

Scenario:

Deploy a web application

Components:

1. **Frontend (React):** Port 3000

2. **Backend (Node.js):** Port 5000

3. **Database (PostgreSQL):** Port 5432

4. **Redis Cache:** Port 6379

5. **Nginx Reverse Proxy:** Port 80

Network flow:

```
Internet → Port 80 (Nginx) → Port 3000 (React) or Port 5000 (Node.js)

                      ↓

                 Port 5432 (PostgreSQL)

                      ↓

                 Port 6379 (Redis)
```

Security:

- Only port 80 exposed to internet
- All other ports only accessible internally