✅ Master Interview Questions for DevOps Networking (Cleaned & Categorized)

https://kirankumarvel.wordpress.com/2025/11/18/networking-for-devops-the-ultimate-guide/
https://kirankumarvel.wordpress.com/2025/11/18/networking-basics-for-devops/
https://kirankumarvel.wordpress.com/2025/11/18/ip-addressing-basics-for-devops/
https://kirankumarvel.wordpress.com/2025/11/18/subnets-explained-for-devops-public/
https://kirankumarvel.wordpress.com/2025/11/18/ports-and-protocols-for-devops-tcp-vs-udp/
https://kirankumarvel.wordpress.com/2025/11/18/dns-basics-for-devops/
https://kirankumarvel.wordpress.com/2025/11/18/firewalls-security-groups-for-devops-cloud-security-basics/
https://kirankumarvel.wordpress.com/2025/11/18/load-balancers-for-devops/
https://kirankumarvel.wordpress.com/2025/11/18/nat-routing-gateways-devops-guide/
https://kirankumarvel.wordpress.com/2025/11/18/http-https-devops-networking-guide/
https://kirankumarvel.wordpress.com/2025/11/18/docker-kubernetes-networking-basics-devops/
https://kirankumarvel.wordpress.com/2025/11/18/tools-every-devops-engineer-must-know-beginner-guide/
https://kirankumarvel.wordpress.com/2025/11/18/devops-tools-every-engineer-must-know-advanced-guide/
https://kirankumarvel.wordpress.com/2025/11/18/aws-cloud-networking-vpc-subnets-route-tables-security-groups/

1. Networking Basics — Detailed Interview Answers

1. What is networking in simple terms?

Networking is how computers **communicate** with each other to share information — like messages, files, videos, or websites.

- When you send a WhatsApp message, your phone sends data across a network to the WhatsApp server.
- When you open a website, your browser talks to a server on the internet.

Memory tip: Networking = computers talking.

2. What is an IP address?

An IP address is the **digital home address** of a device.
It helps other devices know **where to send data**.

Example:
Your laptop may have `192.168.1.5` at home.
Memory tip: IP = where the device lives.

3. Difference between public & private IP?

Public IP

- Assigned by ISP / Cloud provider.
- Reachable from the internet.
- Example: You can ping a server with a public IP.

Private IP

- Used inside local networks like home WiFi, office LAN, AWS VPC.
- Not reachable from internet without NAT or port forwarding.

Memory tip: Public = internet-facing.
Private = inside your home/office.

4. Examples of private IP ranges

These are special ranges reserved for internal networks:

- **10.0.0.0 – 10.255.255.255**
- **172.16.0.0 – 172.31.255.255**
- **192.168.0.0 – 192.168.255.255**

Memory tip: 10.x, 172.16–31, 192.168.x.

5. Difference between IP address and Port?

- **IP:** Identifies the machine
- **Port:** Identifies the service on that machine

Example:
`192.168.1.10:3306` → MySQL running on port 3306 on that device.
Memory tip: IP = building, Port = room number.

6. What is a port in networking?

A port is a **logical door** through which a service communicates.

- Port 22 = SSH
- Port 80 = Website (HTTP)
- Port 443 = Secure website (HTTPS)

Memory tip: Port = door to a service.

7. Why do services use specific ports?

Every service listens on a unique port so the OS knows **which application** should get the incoming data.
Examples:

- SSH → 22
- HTTP → 80
- HTTPS → 443

Memory tip: Ports avoid confusion when many services run on the same machine.

8. TCP vs UDP?

TCP

- Reliable
- Guarantees delivery
- Slower

- Used for: websites, SSH, emails

UDP

- No guarantee (best-effort)
- Faster
- Used for: video calls, gaming, streaming

Memory tip: TCP = careful delivery. UDP = fast delivery.

9. Common ports

- **22 → SSH**
- **80 → HTTP**
- **443 → HTTPS**
- **3306 → MySQL**

10. What is DNS and why is it important?

DNS is the **phonebook of the internet**.
It converts domain names → IP addresses.
Example:
`google.com` → `142.250.xx.xx`
Without DNS, you'd have to type IPs manually.
Memory tip: DNS = contacts list for websites.

11. What is an A record?

Maps a domain to an **IPv4 address**.
Example:
`example.com → 93.184.216.34`

12. What is a CNAME?

CNAME = **alias**.
It points one domain name to another domain.
Example:
`www.example.com → example.com`

13. What is an MX record?

MX stands for **Mail Exchange**.
It tells which server handles **email** for a domain.
Example:
`gmail.com` has MX entries like `smtp.google.com`

14. HTTP vs HTTPS

- **HTTP:** Not encrypted
- **HTTPS:** Encrypted using SSL/TLS

HTTPS protects:
✔ Passwords
✔ Credit card details
✔ Private data

15. What is a firewall?

A firewall is a **security filter** that decides:

- What traffic is allowed
- What traffic is blocked

It works based on IP, ports, and protocols.
Memory tip: Firewall = security guard.

16. Inbound vs Outbound rules

- **Inbound:** What can enter the server
- **Outbound:** What can leave the server

Example:
Inbound allow 443 → accept website traffic.

🔥 Intermediate Level

---

17. What is CIDR notation?

CIDR = **Classless Inter-Domain Routing**
It represents:

- The network
- How many IPs it contains

Example: `10.0.0.0/24`

18. What does 10.0.0.0/24 mean?

- `/24` = 24 bits for network
- Remaining 8 bits for hosts
- **Total IPs = 256**
- **Usable = 254** (1 for network, 1 for broadcast)

19. How many usable IPs in /24?

254 usable IPs.

20. Difference between /16, /24, /32

- **/16 → large network**
  (65,000+ IPs)
- **/24 → medium network**
  (256 IPs)
- **/32 → single IP**
  Used in firewall rules.

21. What is a subnet mask?

It specifies which part of the IP is:

- **Network portion**
- **Host portion**

Example:
`/24 = 255.255.255.0`

22. How does subnetting improve security?

Subnetting **isolates resources** so that unwanted traffic does not reach sensitive services.
Example:

- Web servers in public subnet
- Databases in private subnet
- DB subnet cannot be accessed from internet

23. What is DNS caching?

DNS records are stored temporarily to avoid repeating lookups.
This:

- Speeds up browsing
- Reduces DNS traffic

24. What is TTL?

TTL = **Time To Live**
It tells how long a DNS entry stays in cache.
Example:
TTL 300 = 5 minutes.

25. Why does DNS use TCP + UDP?

- **UDP:** Fast lookups
- **TCP:** Large responses (DNSSEC) + zone transfers

26. What happens if port 22 is blocked?

You **cannot SSH** into your server.
The connection will:

- Timeout
- Or get refused

27. How do firewalls allow/deny traffic?

Based on:

- **IP address**
- **Port**
- **Protocol (TCP/UDP)**
- **Direction (inbound/outbound)**

Example rule:
Allow TCP 443 from anywhere.

28. What protocol does HTTPS use and why?

HTTPS uses **TLS (SSL)**.
Purpose:

- **Encrypt** data
- **Protect** login/password
- **Verify** server identity
- **Secure** communications

🔥 Advanced Level

---

29. Why do we need IPv6?

IPv4 has **4.3 billion** addresses → **almost exhausted**.
IPv6 provides:

- Almost **unlimited addresses**
- Better routing
- Built-in security features
- Faster processing in some cases

30. What is a port scan and how do you detect it?

Port scan = someone checking which **ports are open** on your server.
Detection tools:

- **IDS** (Intrusion Detection Systems)
- **Firewall logs**
- **Fail2ban**
- **Cloud logs (AWS CloudWatch, VPC Flow Logs)**

31. How do you diagnose DNS latency issues?

Steps:

1. **dig +trace +stats** to measure timing
2. Change DNS resolver (e.g., 1.1.1.1 / 8.8.8.8)
3. Check TTL values
4. Test from different networks
5. Check packet loss using `ping`/`mtr`

32. How do you check if a firewall is blocking traffic?

Use:

1. telnet

- `telnet <IP> <PORT>`

2. netcat

- `nc -vz <IP> <PORT>`

3. Cloud logs

- AWS Security Groups
- NACL logs
- VPC Flow Logs

4. Packet capture

- `tcpdump -i eth0 port <PORT>`

If packets come but replies don't go, firewall is blocking.

---

2. Cloud Networking (AWS) — Interview Questions

🔰 Beginner Level

---

34. What is a VPC and why do we use it?

A **VPC (Virtual Private Cloud)** is your own **private network** inside AWS — like your own mini-internet where YOU control everything.
Think of AWS as a huge city.
A VPC is your *private secured colony* inside that city.

Why do we use a VPC?

Because it gives you **complete control** over:
✔ **IP address range**
✔ **Subnets** (public/private)

✔ **Routing** (where traffic goes)
✔ **Security** (who can enter and leave)
✔ **Connectivity** (internet, VPN, on-prem connection)

Real-World Example:

If you run a website + database:

- You place the **web app** inside a public subnet (internet access).
- You place the **database** inside a private subnet (no internet, more secure).

Everything stays inside YOUR VPC → **safe, isolated, controlled.**

In one line for interviews:

"A VPC is an isolated network in AWS where you deploy apps securely with full control over IP ranges, subnets, routing, and firewalls."

35. What is a subnet?

A **subnet** is a smaller section inside a VPC.
You divide your VPC into subnets to organize and secure your resources.
Think of your VPC as a house.
Subnets are **rooms** inside the house.
Each room has a purpose.

Why do we use subnets?

💠 Better security
💠 Better isolation
💠 Better traffic control
💠 To follow cloud architecture best practices
💠 To separate internal vs external systems

Real AWS Example:

Public Subnet

- Has a route to **Internet Gateway (IGW)**
- Used for:
    - EC2 public servers
    - Load balancers
    - NAT Gateways

Private Subnet

- No direct route to internet
- Used for:
    - Databases (RDS)
    - Application backend servers
    - Internal microservices

Interview One-Liner:

"A subnet is a logical subdivision of a VPC used to place resources in separate public or private zones for security and organization."

-

36. Difference between public and private subnet?

Public Subnet	Private Subnet
Has route to Internet Gateway	No route to Internet Gateway
Can access internet directly	Cannot directly access internet
Used for web servers	Used for DB, internal services

37. What is an Internet Gateway?

An **Internet Gateway (IGW)** is the **bridge** that connects your VPC to the internet.
Think of it like:

- Your VPC = a private colony
- Internet Gateway = the **main gate** to the outside world

Without this gate, nothing in your VPC can go online.

Why is an Internet Gateway needed?

Because it allows:
✔ Outbound traffic (EC2 → Internet)
✔ Inbound traffic (Internet → EC2, if allowed)
✔ Public subnets to function
✔ Public IPs to work
If you create a public EC2 instance but don't attach an IGW to the VPC?
➡️ It WON'T reach the internet.
➡️ Even "ping google.com" will fail.
➡️ Even SSH won't work.

Where is the IGW attached?

- You attach an IGW to the **entire VPC** (not to subnets or EC2).

- Then you update the **route table** to send **0.0.0.0/0** traffic → IGW.

Interview One-Liner:

"An Internet Gateway is a VPC component that allows communication between AWS resources and the public internet."

38. Can private subnets access the internet? How?

Yes, private subnets *can* access the internet, but **indirectly**, using a **NAT Gateway**.

Why private subnets cannot access internet directly?

Because they do **NOT** have a route to the Internet Gateway.
Private subnet = no public gate.

How does a NAT Gateway help?

A **NAT Gateway** sits in a **public subnet** and acts like a **middleman**:

- It sends outbound traffic from private EC2 instances to the internet.
- It hides private IPs behind its **public IP**.
- It **does NOT allow** internet → private subnet (one-way traffic).

Flow (Very Simple):

```
Private EC2
   ↓
NAT Gateway (in Public Subnet)
   ↓
Internet Gateway
   ↓
Internet
```

When do you need this?

- Updating software
- Installing packages (`yum update`, `apt-get update`)
- Downloading logs
- Calling external APIs

Interview One-Liner:

"Private subnets access the internet using a NAT Gateway in a public subnet, which forwards outbound traffic while keeping the private subnet isolated."

⚙️ Intermediate Level

---

39. What is a Route Table?

A **Route Table** is like a **map** that tells your network where to send traffic.
Every subnet in AWS uses a Route Table to decide:

- Should traffic stay inside the VPC?
- Should it go to the internet?
- Should it go to another subnet?
- Should it go to a NAT Gateway?
- Should it go to a VPC Peering connection?

Why do we need a Route Table?

Because AWS networking doesn't automatically know:

- Where to send packets
- Whether traffic should go to the internet or stay private
- Which gateway to use

Without route tables, NOTHING communicates.

How does a Route Table work?

It has rules called **routes**.
Each route has two parts:

1. **Destination** → The target network (e.g., `0.0.0.0/0`)
2. **Target** → Where to send the traffic (e.g., IGW, NAT, local)

Example Routes (Very Common):

Destination	**Target**	**Meaning**
10.0.0.0/16	local	All internal VPC traffic stays inside
0.0.0.0/0	Internet Gateway	Allow traffic to the internet
0.0.0.0/0	NAT Gateway	Allow private subnet → internet
192.168.0.0/16	VPC Peering	Send traffic to another VPC

How AWS uses Route Tables

- A **public subnet** has a route:
  `0.0.0.0/0 → IGW`

- A **private subnet** has a route:
  `0.0.0.0/0 → NAT Gateway`
- An **isolated subnet** has NO route to the internet.

Interview One-Liner:

"A Route Table contains routing rules that control where traffic from a subnet goes—inside the VPC, to the internet via IGW, or to the internet via NAT Gateway."

40. Why do private subnets need a NAT Gateway?

Because **private subnets are not allowed to access the internet directly**, they need a NAT Gateway to reach external services **securely**.

Why can't private subnets connect directly?

- They **don't** have public IPs.
- They **don't** have a route to the Internet Gateway.
- AWS blocks direct inbound/outbound access for security.

Private subnet = **isolated by design**.

So what does the NAT Gateway do?

A NAT Gateway acts like a **safe middleman**:

- It sends traffic **outbound** to the internet.
- It receives responses back.
- It **never** allows inbound traffic from the internet.

What private EC2 instances use NAT for?

✔ Installing updates (yum, apt, pip)
✔ Downloading software
✔ Accessing external APIs
✔ Pushing logs
✔ Calling AWS services that require public endpoints

How NAT Gateway protects your private subnet

- Private EC2's IP is **hidden**
- NAT Gateway uses **its own public IP**
- Internet **cannot initiate a connection back**

This provides **strong security** while still enabling functionality.

Traffic Flow (Easy Diagram):

```
Private Subnet EC2

        ↓

     NAT Gateway

        ↓

   Internet Gateway

        ↓

      Internet
```

Where do you place NAT Gateway?

- NAT Gateway is placed **in a public subnet**
- Because it needs a **public IP** + access to IGW

Interview One-Liner:

“Private subnets need NAT Gateway because they cannot reach the internet directly. NAT provides secure outbound access for updates and APIs while still keeping the private subnet isolated from inbound traffic.”

41. Difference between Security Groups and NACL?

Security Group	**NACL**
Works at instance level	Works at subnet level
Stateful	Stateless
Return traffic automatically allowed	Need separate inbound + outbound rules
Easy to manage	For granular control

42. How does AWS decide if a subnet is public or private?

A subnet is **public** ONLY if:

1. **Its route table has a route to the Internet Gateway (IGW)**
   `0.0.0.0/0 → IGW`

If this route does **not** exist, the subnet is **private**.

Detailed Explanation (Easy to Remember)

AWS decides the type of subnet based on its **route table**, not on any special setting in the subnet itself.

Public Subnet = Has internet access

To be public, a subnet MUST have:
✔ A route to the Internet Gateway
✔ (Optional) EC2 inside it must have a public IP or Elastic IP
Without a route to IGW → No public access.

Private Subnet = No direct internet access

A subnet becomes private when:

- It **does n**
- It may have a route to NAT Gateway for outgoing access
  but NAT ≠ public.

So even if you put EC2 inside with a public IP,
without `0.0.0.0/0 → IGW`, it is still **private**.

How AWS internally sees it

AWS checks the subnet’s route table:

Route to IGW exists?	**Subnet Type**
Yes	Public
No	Private

That’s it—simple!

Interview One-Liner:

"A subnet is public only when its route table sends internet-bound traffic to an Internet Gateway. Otherwise, AWS considers it private."

43. What happens if you put a database in a public subnet?

Putting a database in a public subnet exposes it to the internet, which is a **major security risk**.
Databases (like MySQL, PostgreSQL, MongoDB, RDS, etc.) are sensitive.
If placed in a public subnet:

Risk 1: Open to internet scanning

Hackers can:

- Scan the public IP
- Detect open ports (3306, 5432, 27017, etc.)
- Attempt brute-force login
- Try SQL injection or exploit vulnerabilities

Risk 2: Misconfigurations become dangerous

Even a small mistake like:

- Opening port 3306 to `0.0.0.0/0`
- Weak passwords
- Outdated DB versions

…can allow attackers to access your database.

Risk 3: Compliance Violations

Public databases often violate:

- PCI-DSS
- GDPR
- HIPAA security rules
- Company security policies

Risk 4: Easier target for DDoS

Publicly visible = attackers can flood it with traffic.

Best Practice (Always Follow This)

Databases should ALWAYS be in private subnets.

- No direct internet access
- Only app servers (in public subnet or other private subnets) can access them
- Security Groups strictly control traffic

How a safe architecture looks

- `Internet → ALB → App Server (Public Subnet) → Database (Private Subnet)`

Interview One-Liner:

"Placing a database in a public subnet exposes it to the internet and creates a huge security risk. Best practice is to always keep databases in private subnets with restricted access."

🚀 Advanced Level

---

44. Explain flow: ALB → EC2 → RDS.

This is **one of the most common architectures in AWS**, and interviewers expect you to explain it clearly.
Let's break it down step-by-step:

📌 **Step 1: User Sends a Request (Browser → Internet → AWS)**

A user opens a website:
`https://yourapp.com`
The request reaches AWS via the internet.
But it cannot directly reach EC2 or RDS.

📌 **Step 2: Request Hits the ALB (Public Subnet)**

The **Application Load Balancer (ALB)** lives in **public subnets**, meaning it HAS:
✔ public IP
✔ route to Internet Gateway (IGW)
The ALB's job:

- Receive traffic from users
- Distribute it across multiple EC2 instances
- Check which EC2 is healthy
- Scale automatically
- Terminate SSL/HTTPS

📌 **Step 3: ALB Forwards Traffic → EC2 (Private Subnet)**

EC2 instances hosting your app are placed in **private subnets** to keep them secure.
Why private?

- They don't need direct internet access
- Hackers cannot reach them
- Only the ALB can communicate with them

The ALB forwards requests based on:

- Port (example: 80 or 443)
- Target group rules
- Health checks

📌 **Step 4: EC2 Communicates with RDS (Private Subnet)**

Your application on EC2 needs a database.
RDS is also placed in a **private subnet** for maximum security.
Flow:

```
EC2 (App Logic)
     ↓
RDS (Database)
```

RDS often runs MySQL, PostgreSQL, MariaDB, etc.
EC2 and RDS talk using private IPs inside the VPC, through:
✔ Security Groups
✔ Private routing
✔ No internet exposure
This ensures **secure data communication**.

📌 **Step 5: The Response Travels Back**

After processing:

1. **RDS** → **EC2** returns the result (e.g., user profile, product list)
2. **EC2** → **ALB** sends final web response
3. **ALB** → **User** sends output back to browser

Flow summary:

```
User → ALB → EC2 → RDS → EC2 → ALB → User
```

📌 **Why This Architecture Is Used?**

✔ Security

- ALB is public
- EC2 + RDS are private
- Database is never exposed to internet

✔ Scalability

- ALB distributes traffic
- EC2 auto-scales
- RDS can scale vertically or with read replicas

✔ High Availability

- Multi-AZ ALB
- EC2 across multiple AZs
- RDS Multi-AZ failover

✔ Performance

- Internal traffic is private + fast
- ALB optimizes load distribution

This maintains **security + scalability**.
In AWS, the user request first hits the ALB in the public subnet. The ALB forwards it to EC2 servers running in private subnets. These EC2 instances communicate with an RDS database in private subnets. The response then flows back the same way: RDS → EC2 → ALB → User. This architecture ensures security, scalability, and proper isolation.

45. How would you troubleshoot “EC2 cannot access internet”?

Check in order:

1. Is EC2 in **private subnet** → needs NAT Gateway
2. Is NAT Gateway in a **public subnet** with route to IGW?
3. Route table:
   - Private subnet → NAT
   - Public subnet → IGW
4. Security Group outbound rules
5. NACL outbound rules
6. Is there a route conflict?

46. How do you secure a VPC end-to-end?

- Use private subnets for sensitive resources
- Security Groups for instance-level protection
- NACLs for subnet-level control
- Remove public IPs if not needed
- Use VPC Endpoints instead of internet
- Encrypt data in transit (TLS)
- Enable flow logs → monitor traffic
- Use WAF + Shield for public apps

47. How does VPC Peering differ from Transit Gateway

VPC Peering	Transit Gateway
Connects 2 VPCs	Connects many VPCs
One-to-one	Hub-and-spoke
No transitive routing	Supports transitive routing
Simple	Enterprise level

48. How do you design HA NAT Gateways across AZs?

A NAT Gateway lets private EC2 instances access the internet **without exposing them**.
But to make NAT highly available (HA), you must design it correctly **across multiple Availability Zones (AZs)**.

Why HA NAT design matters

- If NAT Gateway is deployed in **only one AZ**, and that AZ goes down →
  all private instances in every AZ lose internet access (e.g., no OS updates, no API calls).
- Also, if private subnets use a NAT Gateway in a *different* AZ →
  you get cross-AZ data transfer charges (₹₹).

The Correct (Best Practice) HA Design

1. Create ONE NAT Gateway per Availability Zone

Example:

AZ	NAT Gateway	Private Subnet
ap-south-1a	NAT-GW-A	Private-subnet-A
ap-south-1b	NAT-GW-B	Private-subnet-B
ap-south-1c	NAT-GW-C	Private-subnet-C

This ensures:

- Redundancy
- No AZ dependency
- No cross-zone routing

2. Each private subnet must route to the NAT Gateway in the SAME AZ

Example routing rule inside the **private route table** of AZ “1a”:

```
0.0.0.0/0 → NAT-GW-A
```

For AZ “1b”:

```
0.0.0.0/0 → NAT-GW-B
```

This keeps all traffic local to its AZ.

Best practice:

- Create **one NAT Gateway per AZ**
- Each private subnet uses the NAT Gateway in **the same AZ**
- Prevents cross-AZ charges
- Ensures high availability even if one AZ fails

“To design HA NAT Gateways, deploy one NAT Gateway per AZ and update the route table for each private subnet to use the NAT Gateway in the same AZ. This avoids cross-AZ routing, reduces cost, and ensures that if one AZ fails, the other AZs still have internet access."

4. Ports & Protocols — Interview Questions

Beginner Level

49. Why do services use ports?

Answer (simple & detailed):
A computer's IP address is like a building's street address — it gets traffic to the right building. A **port** is like a door number inside that building. Multiple services (web server, SSH, database) run on the same machine (same IP), so the OS uses ports to deliver incoming network traffic to the correct application.

- Ports are numbered **0–65535**.
  - **0–1023**: well-known system ports (HTTP 80, SSH 22).
  - **1024–49151**: registered ports.
  - **49152–65535**: ephemeral/dynamic ports used for outgoing connections.
- When your browser connects to `example.com:80`, it asks the server machine (IP) to deliver traffic to door 80 — where the web server (HTTP) listens.

Try it: On your machine, run `ss -tuln` or `netstat -tuln` to list listening ports and the services attached.
Memory tip: IP = house, port = room door.

50. What is the difference between TCP and UDP?

Answer (simple & detailed):
TCP and UDP are transport-layer protocols that move data between machines. The main differences:

- **TCP (Transmission Control Protocol)**
  - **Reliable:** ensures data arrives and in order.
  - **Connection-oriented:** client and server establish a connection (handshake) before data flows.
  - **Retransmission & flow control:** lost packets are resent; receiver controls speed.
  - **Use cases:** web pages (HTTP/HTTPS), SSH, SMTP, databases — anywhere you need accuracy.
- **UDP (User Datagram Protocol)**
  - **Unreliable (best-effort):** sends packets without waiting for acknowledgements.
  - **Connectionless:** no handshake, lower overhead and latency.
  - **No retransmission by default:** application must handle loss if needed.

    - **Use cases:** DNS queries (short queries), video streaming, VoIP, online gaming — where speed matters more than perfect delivery.

Simple analogy: TCP = registered post (tracking, guaranteed delivery). UDP = postcard (fast, no guarantee).
Try it: `curl -I http://example.com` uses TCP (HTTP). `dig @8.8.8.8 example.com +tcp` forces DNS over TCP; without `+tcp` dig uses UDP by default.
Memory tip: TCP = careful & reliable, UDP = fast & simple.

51. What port does MySQL run on?

Answer :

- **Default port for MySQL: 3306 (TCP)**.
- MySQL server listens on TCP port 3306; clients connect to the server IP and port 3306 to perform queries.

Notes & best practice:

- In production, **do not expose 3306 to the internet**. Keep it in a private subnet or behind firewall rules / security groups. Use SSL/TLS for encryption if you must connect across networks. Consider using user accounts and strong passwords, and allow access only from specific application servers (by IP or security group).

Try it: From a machine allowed to reach the DB: `telnet <db-ip> 3306` or `nc -vz <db-ip> 3306`. If connection succeeds you'll see a response; otherwise it times out (blocked).
Memory tip: MySQL → 33**06** (think 33 = pair, 06 = db).

52. What happens if a required port is blocked?

Answer (simple & detailed):
If a port a service needs is blocked, the client cannot reach that service — the connection will fail or time out. Symptoms vary by application:

- **SSH (22) blocked:** you cannot log in remotely; connection tries then times out or is refused.
- **HTTP (80) blocked:** browser shows "connection refused" or "site can't be reached."
- **Database port blocked:** app can't query the DB and fails with connection errors.

Typical reasons a port is blocked:

- Host firewall (iptables/nftables/firewalld) denies incoming traffic.
- Cloud security groups or NACLs are not allowing the port.
- Network firewall / corporate policy blocks the port.
- Service is not running or listening on that port (so OS refuses connection).

How to check & fix:

1. **Is service up?** `ss -tuln | grep <port>` or `systemctl status <service>`.
2. **From client:** `telnet <ip> <port>` or `nc -vz <ip> <port>`.
    - If it connects, network allows it.
    - If it fails, check firewalls and security groups.
3. **On host:** `sudo iptables -L -n` or `sudo nft list ruleset`.

4. **Cloud:** check Security Group inbound/outbound rules and NACLs; ensure route tables/IGW if public.

Memory tip: Blocked port = closed door — you need permission (firewall rule) or the right door (service listening).

Intermediate Level

---

53. Why does DNS use both TCP and UDP?

Answer (simple & detailed):

DNS originally used **UDP** for most queries because UDP is faster and most DNS responses fit into a single UDP packet. However, DNS uses **TCP** in these cases:

1. **Zone transfers (AXFR):** when one DNS server copies an entire zone from another server (larger data), TCP is used for reliability.
2. **Large responses / truncation:** if a DNS response is too large for a single UDP packet (especially with DNSSEC or many records), the server sets the “TC” (truncated) flag; the client retries the query over **TCP** to get the full response reliably.
3. **DNS over TCP for TLS/HTTP upgrades:** modern secure transports also exist (DoT, DoH), but classic TCP fallback still matters.

Practical summary:

- **UDP** = default & fast for standard lookups.
- **TCP** = fallback for large/reliable transfers.

Try it: `dig example.com` (UDP) vs `dig +tcp example.com` (forces TCP).

Memory tip: UDP first for speed; TCP when the answer is too big or needs reliability.

54. What protocol does HTTPS use?

Answer (simple & detailed):

- **HTTPS = HTTP over TLS (Transport Layer Security)**. Historically called SSL, but modern secure web uses **TLS**.

- The connection sequence:
  1. **TCP** connection established (usually port **443**).
  2. **TLS handshake** runs over that TCP connection to negotiate encryption keys, verify certificates, and set up an encrypted channel.
  3. **HTTP** messages are exchanged **inside** this encrypted TLS tunnel.

Why TLS?

- Encrypts data in transit (prevents eavesdropping).
- Provides server identity via certificates (prevents impersonation).
  Can provide integrity checks (detect modification).

Try it: `openssl s_client -connect example.com:443` will show TLS handshake and server certificate details.

Memory tip: HTTPS = HTTP + TLS (like a private conversation inside a secure room).

55. How do firewalls use ports to allow traffic?

Answer (simple & detailed):
Firewalls control traffic by matching on **IP addresses, ports, and protocols (TCP/UDP/ICMP)** and applying rules to **allow** or **deny** packets.

- **Host firewall (iptables, nftables, ufw):** runs on the server and can allow/deny traffic for specific ports and IPs. Rules are applied top-to-bottom and can be stateless or stateful.
- **Network firewall / perimeter firewall:** placed on the network edge to enforce organization-wide rules.
- **Cloud firewall (Security Groups, NACLs):** virtual firewalls in cloud environments.

Rule examples:

- Allow SSH only from your office IP: `allow tcp port 22 from 203.0.113.5/32`.
- Allow web traffic from anywhere: `allow tcp port 443 from 0.0.0.0/0`.

Stateful vs Stateless:

- **Stateful** firewalls (like Security Groups) remember connections — if inbound is allowed, response outbound is automatically allowed.
- **Stateless** firewalls (like NACLs) evaluate each packet independently and require explicit allow for both directions.

How packet matching works (simplified):

1. Packet arrives with source IP, destination IP, destination port, and protocol.
2. Firewall checks rules: if a rule matches an allow condition, the packet is permitted; otherwise it is denied or dropped.

Try it: On Linux, list iptables rules: `sudo iptables -L -n --line-numbers`. For nftables: `sudo nft list ruleset`.
Memory tip: Firewalls are gatekeepers: IP = who, port = which door, protocol = how they knock.

Cloud Level

56. How do AWS Security Groups manage ports?

Answer (simple & detailed):
Security Groups (SGs) in AWS are **virtual stateful firewalls** attached to EC2 instances, ENIs, load balancers, RDS, etc.
Key points:

- **Instance-level:** SGs are associated with resources (not subnets).
- **Stateful behavior:** If an incoming connection is allowed, return traffic is automatically allowed — you don't need to create a separate outbound rule for responses.
- **Rules are permissive-only:** You add allow rules; deny is implicit (no explicit deny rules inside SGs).
- **Specify by port & source:** Each rule specifies protocol (TCP/UDP), port or port-range, and source (CIDR block, security group, or IP).
    - Example inbound rule: allow TCP port 22 from `203.0.113.0/32`.
    - Example inbound rule: allow TCP port 443 from `0.0.0.0/0` (public HTTPS).

Best practices:

- **Least privilege:** Open only needed ports and restrict source IPs/security groups.
- **Use SG references:** Allow access from another SG (e.g., app servers allow DB access from web server SG) instead of IPs, which is cleaner in autoscaling setups.
- **Multiple SGs:** You can attach multiple SGs to one instance—rules are unioned.

Try it: In AWS console, look at an EC2 instance's Security Groups → Inbound/Outbound rules.
Memory tip: Security Group = instance-level allow-list; stateful means replies are automatic.

57. Why restrict port 3306 in production?

Port **3306** is MySQL's default port. Exposing it publicly is dangerous:
Risks:

- **Brute-force attacks:** bots/attacker scripts try weak credentials.
- **Data leakage & exploitation:** unpatched DBs or misconfigurations can be exploited.
- **Automated scanners** constantly scan the internet for open DB ports.

Best practices (how to restrict):

1. **Don't give DB public IPs.** Place DBs in a **private subnet**.
2. **Security Groups:** allow inbound **3306** only from app server SG or specific internal IPs.
3. **Use TLS** for DB connections across networks.
4. **Use database users + least privilege** (no shared root/passwords).
5. **Use bastion/jump hosts** or port forwarding for admin access (only from admin IPs).
6. **Enable auditing & monitoring.**

Try it: From outside your VPC, try `nc -vz <db-public-ip> 3306`. If properly restricted, it should time out or refuse.
Memory tip: DBs are internal tools — keep their doors closed to the internet.

58. How do load balancers forward traffic based on ports?

Load balancers accept client traffic on **listener ports** and forward it to registered backend instances/targets on **target ports**. There are two common concepts:

1. **Listener:** The port and protocol on which the load balancer accepts connections (e.g., port 80 or 443).
2. **Target/Instance port:** The port on the backend server where the application listens (e.g., port 8080 or 3306).

Flow examples:

- **Simple web case:** ALB listener on **443 (HTTPS)** → forwards to target group on **port 80** (HTTP) on EC2s (ALB can also do TLS termination).
- **Port translation:** You can have ALB/NLB receive on port 443 and forward to instance port 8443 or 8080.
- **Multiple listeners:** LB can have multiple listeners (80 → forward to service A, 443 → service B).
- **Health checks:** LB regularly checks target ports to ensure instance is healthy before sending traffic.

Types of load balancers:

- **Layer 4 (NLB):** forwards based on IP & TCP/UDP ports, very fast — doesn't inspect HTTP.
- **Layer 7 (ALB):** can route based on HTTP details (host, path) and still forwards on ports.

AWS specifics:

- **Target groups** contain targets (instances, IPs, lambda) with specified port(s).
- Listener rules decide how to route (host/path conditions).
- You can use **sticky sessions** (session affinity) to bind a client to a target (useful for stateful apps).

Try it: In AWS, create a simple target group with instances on port 8080, create ALB listener on 80 and see traffic forwarded to target port 8080. Or locally, run `nginx` on port 8080 and use an Nginx reverse proxy as a simple LB listening on 80 forwarding to 8080.
Memory tip: LB = receptionist: listens on a public door (listener port) and sends guests to the correct office room (target port).

---

5. DNS — Interview Questions

Beginner Level

---

59. What is DNS and why do we need it?

DNS (Domain Name System) is the **phonebook of the internet**.

Why do we need it?

Because humans remember names like **google.com**, but computers need IPs like **142.250.x.x**.
DNS converts:

```
Name → IP address
google.com → 142.250.193.xxx
```

Without DNS:

- You would enter IPs manually
- Websites would become hard to access
- Apps, APIs, and microservices would break instantly

60. What is an A, AAAA, CNAME, MX record?

Type	Descriptions	Example
A Record	Maps a domain → **IPv4 address**	`example.com → 192.168.10.5`
AAAA Record	Maps a domain → IPv6 address	`example.com → 2001:db8::1`
CNAME Record	An alias. It points one domain to another domain, not an IP.	`www.example.com → example.com`
MX Record	Used for email routing. Tells where emails for the domain should go.	

Intermediate Level

61. Difference between A and AAAA?

Record	Purpose	Example
A	IPv4	192.168.1.10
AAAA	IPv6	2001:db8::1

IPv6 is newer, supports more addresses, and is the future.

62. When to use CNAME?

Use a CNAME when:

- You want multiple domains to point to **the same destination**
- You want **one place** to update the target domain

Example:

```
shop.example.com → example.com
blog.example.com → example.com
```

If you change example.com's IP, the others follow automatically.

63. What is the TXT record used for?

TXT records store **text-based verification info**, including:

- Google domain verification
- DKIM(DomainKeys Identified Mail)
- SPF(sender policy framework)
- Email authentication
- Security policies

Example:

```
v=spf1 include:_spf.google.com ~all
```

64. What is TTL?

TTL = **Time To Live**
It tells how long the DNS record should be cached before querying again.
Example:

```
TTL = 300 seconds (5 minutes)
```

Low TTL → fast changes, but more DNS traffic
High TTL → slower changes, but faster performance

65. How does DNS caching work?

When you visit a website:

1. Your browser checks its cache
2. Your OS checks DNS cache
3. Your router checks
4. ISP checks
5. Finally goes to global DNS servers

Each layer respects the record's **TTL**.
This is why DNS changes take time to propagate.

Cloud Level

---

66. Why is DNS important in microservices?

Microservices communicate using **service names**, not IPs.
Because IPs change often in:

- Kubernetes pods
- Auto-scaling groups
- Containers

DNS ensures services find each other automatically.
Example:

```
auth-service → auth-service.cluster.local
```

If auth-service pod dies and restarts → DNS updates automatically.

67. How do CDNs integrate with DNS?

CDNs (CloudFront, Akamai, Cloudflare) use DNS to send users to the **nearest edge location**.
Flow:

```
User → DNS → Nearest CDN server → Origin
```

Benefits:

- Low latency
- Faster content delivery
- Load balancing across locations
- Built-in DDoS protection

68. What is Route53 used for?

Route53 is AWS's **DNS service**.
You use it for:

- Managing domains
- Hosting DNS records
- Health checks
- Traffic routing
- Failover routing
- Latency-based routing
- Weighted routing (A/B testing)

It is extremely reliable and globally distributed.

69. What happens if DNS fails?

If DNS breaks:

- Websites become unreachable
- Apps cannot call their APIs
- Microservices stop talking
- Emails fail
- CDNs don't route
- Users see "Server not found"

Even if servers are perfectly healthy, **without DNS, nothing can find them.**
DNS failure = complete shutdown of internet communication.

---

6. Firewalls & Security Groups — Interview Questions

Beginner Level

---

70. What is a firewall?

A **firewall** is like a **security guard** for your network.
It decides:

- **What traffic is allowed in**
- **What traffic is allowed out**

Think of it as the "border control" for your computer or server.
A firewall protects you from:

- Hackers
- Unwanted traffic
- Malware
- Unauthorized access

Firewalls can be:

- Software firewalls (on your OS / cloud VM)
- Hardware firewalls (router, security appliance)
- Cloud firewalls (AWS Security Groups, NACLs)

71. What are inbound and outbound rules?

Inbound Rules → What can ENTER

These rules control **incoming traffic**.
Example:
Allow port **443** → Users can access your HTTPS website
Block port **22** → No one can SSH into your server

Outbound Rules → What can LEAVE

These rules control **outgoing traffic**.
Example:
Allow all outbound → Your server can access internet
Block outbound port 25 → Prevents sending spam emails
Inbound = door to enter
Outbound = door to exit
Simple memory trick:
➡️ **INbound = INcoming**
➡️ **OUTbound = OUTgoing**

72. Why should port 22 be restricted?

Port **22** is used for **SSH**, a direct way to log into your server.
If port 22 is open to the world:

- Hackers will attempt brute-force attacks
- Bots will try default usernames/passwords
- Server compromise becomes very easy

Best practice:

- Allow **SSH only from your IP** (e.g., your home network)
- Or use a bastion host
- Or use AWS SSM Session Manager (no port 22 needed)

This is one of the **first things interviewers check** in DevOps security.

73. What port is needed for HTTPS?

HTTPS uses **port 443**.
Why?
Because HTTPS = HTTP + SSL/TLS encryption
Port 443 ensures:

- Data is encrypted
- Login credentials are safe
- Payment info is protected
- Browsers show the “secure lock”

Common ports to remember:

- **80** → HTTP

- **443** → HTTPS
- **22** → SSH
- **3306** → MySQL

Intermediate Level

---

74. Difference between firewall and security group?

Feature	Firewall	Security Group (AWS)
Location	Can exist on devices, networks, routers	Exists only inside AWS
Direction	Controls inbound + outbound	Controls inbound + outbound
Layer	Works at network level	Works at instance level
Behavior	Can be **stateful or stateless**	Always **stateful**
Typical Use	Protect networks	Protect EC2 instances

Easy memory:
Firewall = city gate
Security Group = security guard for each building

75. Are Security Groups stateful or stateless?

Security Groups are **stateful**.
Meaning:

- If you allow **inbound** traffic, the **response is automatically allowed back**.
- You don't need to write an outbound rule for the reply

Example:
Allow inbound port 443 → response leaves automatically.

76. Why restrict access by IP?

Because restricting access by IP:

- Reduces attack surface
- Blocks unknown devices
- Only trusted networks can connect
- Prevents brute-force attacks

Examples:

- Allow SSH (22) **only from your office or home IP**
- Allow MySQL (3306) **only from app servers**

This is basic **network hygiene** in DevOps.

77. Why databases shouldn't have open inbound rules?

Databases must never be exposed to the internet because:

- They store sensitive data (users, passwords, payments)
- Hackers can run SQL injection or brute-force attacks
- Botnets constantly scan port **3306**, **5432**, etc.

Best practice:
DB inbound rules should allow **only private subnets** or **only specific application servers**.
Never open DB ports globally (0.0.0.0/0).

Cloud Level

---

78. SG vs NACL?

Feature	Security Group	NACL
Level	Instance-level	Subnet-level
Stateful	✔ Yes	✘ No (Stateless)
Rule Direction	Inbound + Outbound	Inbound + Outbound
Default Behavior	Deny everything	Allow everything
Use Case	Protect specific EC2, RDS	Control full subnet boundaries

Simple memory:
SG = Guard for each house
NACL = Fence around the colony

79. How to secure a public-facing EC2?

Follow this checklist:

Network Level

✔ Place EC2 in a **public subnet**
✔ Attach **internet gateway**
✔ Assign **elastic IP**

Security Level

✔ Allow only required ports (80/443)
✔ Restrict SSH (22) to your IP or disable 22 fully
✔ Attach IAM role instead of storing credentials
✔ Enable Security Group + NACL filtering
✔ Disable root login
✔ Use SSM Session Manager instead of SSH
✔ Keep OS updated

Monitoring

✔ Enable CloudWatch logs
✔ Enable GuardDuty (threat detection)

This is a **high-value interview answer**.

80. What rules for a production database?

Your production DB should follow **zero-trust** principles.

Inbound rules	Outbound rules
✔ Allow traffic **only from specific app servers** ✔ Deny from internet (0.0.0.0/0) ✔ Allow only DB port (e.g., 3306 or 5432)	✔ Allow only necessary internal services ✔ Block unknown external calls

Extra steps

✔ Place DB in **private subnet**
✔ Use **encryption at rest + in transit**
✔ Enable automatic backups
✔ Enable audit logs

81. What is least-privilege security model?

Least privilege = give only the minimum access required.

This applies to:

- Ports
- IPs
- Users
- IAM roles
- Applications
- Networking routes

Example:

- If app only needs DB port 3306 → allow only 3306
- If user only needs read access → don't give write access
- If EC2 only needs S3 → give S3-only IAM permissions

Goal:
Reduce risk by removing unnecessary access.

7. Load Balancers — Interview Questions

Beginner Level

82. What is a load balancer?

A load balancer is a system that distributes incoming traffic across multiple servers to ensure no single server gets overloaded. It improves availability, performance, and fault-tolerance.

83. Why do we need load balancer?

We need a load balancer because:

- It prevents a single server from becoming a bottleneck.
- It increases uptime—if one server fails, traffic shifts automatically to healthy ones.
- It allows horizontal scaling (adding more servers easily).
- It improves performance by sharing traffic efficiently.

84. ALB vs NLB basics

Feature	ALB (Application Load Balancer)	NLB (Network Load Balancer)
Layer	Layer 7 (Application)	Layer 4 (Transport)
Traffic Type	HTTP, HTTPS	TCP, UDP
Use Case	Intelligent routing, microservices	Extreme performance, low latency
Routing	Path-based, host-based	No advanced routing
SSL Termination	Yes	No* (can pass-through TLS)
Health Checks	Application-level	Network-level

Summary:
ALB = Smart, content-aware routing.
NLB = Ultra-fast, connection-level load balancing.

---

85. What is round-robin?

Round-robin is a load balancing algorithm that sends each new request to the next server in sequence.
Example: **Server1 → Server2 → Server3 → then repeats**.
It's simple and works well when all servers have similar capacity.

Intermediate Level

---

86. Difference between Layer 4 & Layer 7 Load Balancer?

Layer 4 (Transport Layer LB):	Layer 7 (Application Layer LB):
● Balances based on IP and port. ● Faster, lightweight. ● Does not inspect HTTP content. Used for TCP/UDP traffic	● Understands HTTP/HTTPS protocols. ● Can inspect headers, cookies, URLs. ● Supports intelligent routing (path/host-based). ● More flexible but slightly heavier.

In short:
L4 = Speed.
L7 = Intelligence.

87. What is path-based routing?

Path-based routing sends traffic to different target groups based on the URL path.
Examples:

- `/api/*` → API servers
- `/images/*` → Image servers
- `/admin/*` → Admin backend

This is essential when you have microservices behind a single load balancer.

88. Can a load balancer perform SSL termination?

Yes.
Load balancers like ALB can perform **SSL termination**, meaning:

- The LB decrypts HTTPS traffic.
- Backend servers receive plain HTTP traffic.

Benefits:

- Reduces CPU load on backend servers.
- Central place to manage certificates.
- Improves performance.

89. What is health checking?

A load balancer uses health checks to confirm whether a backend instance is healthy.
It does this by pinging or sending HTTP requests to a defined endpoint (e.g., `/health`).
If an instance is unhealthy:

- It is removed from the rotation.
- Traffic is sent only to healthy instances.

This ensures high availability and reliability.

Cloud Level

---

90. How does an ALB route to target groups?

An Application Load Balancer evaluates incoming HTTP/HTTPS requests and forwards them to **target groups** based on routing rules.
How it works:

1. Client sends a request to ALB.
2. ALB checks its rule set (host-based, path-based, header-based, method-based).
3. It matches the request with the appropriate target group.
4. ALB forwards to a healthy target (EC2, ECS, Lambda, IP).

Example:

- `/api/*` → API target group
- `/static/*` → Static content target group

ALB always picks a **healthy** instance and uses algorithms like round-robin or least outstanding requests.

91. When to use NLB over ALB?

Use an NLB when you need:

✓ Ultra-low latency

NLB works at Layer 4 and is extremely fast.

✓ Millions of requests per second

Great for real-time systems and high throughput.

✓ TCP/UDP traffic

DB connections, IoT, gaming, VoIP.

✓ Static IP support

NLB can attach Elastic IPs and be whitelisted.

✓ TLS pass-through

If you want backend servers to handle encryption/decryption.

Short answer:
Use NLB when performance, network-level traffic, or speed is the priority over intelligent routing.

92. Auto Scaling + Load Balancer Integration?

Auto Scaling Groups (ASG) integrate tightly with ALB/NLB to ensure your application scales automatically.

How they work together:

- ASG launches new instances when load increases.
- These new instances **register automatically** with the load balancer's target group.
- When ASG terminates instances, they are **deregistered gracefully** from the LB.
- LB's health checks help ASG replace unhealthy instances.

Result:
Continuous availability + automatic capacity management + healthy traffic distribution.

93. What are sticky sessions?

Sticky sessions (session affinity) allow a load balancer to send **the same user** to the **same backend server** for all requests during their session.

Useful when:

- Session data is stored locally on servers (not in Redis/DB).
- Applications need user-specific state maintained.

Examples:

- Shopping carts
- Login sessions without distributed caching

Types:

- ALB uses **cookies** for stickiness.
- NLB uses **source IP affinity**.

Downside:
Can create uneven load if many users get "stuck" to fewer servers.

94. How does NGINX act as a reverse proxy?

NGINX can sit between clients and backend servers, forwarding requests on their behalf.

How it acts as a reverse proxy:

1. Client sends request to NGINX.
2. NGINX forwards request to backend server(s).
3. Backend responds to NGINX.
4. NGINX sends response back to the client.

What it adds:

- Load balancing
- Caching
- SSL termination
- Rate limiting
- Security filtering
- Hiding backend server IPs

Why it's used:
It improves performance, security, and scalability while simplifying backend infrastructure.

---

8. NAT, Routing & Gateways — Interview Questions

Basic Level

---

95. What is NAT and why is it used?

NAT (Network Address Translation) allows private network devices to access the internet using **one public IP**.

Why it's used:

- Private IPs **cannot** reach the internet directly.
- NAT translates private → public IP for outbound traffic.
- It hides internal servers from the internet (**security benefit**).
- Saves public IP usage.

Example:

Your EC2 in a private subnet needs to download updates → NAT Gateway translates the request to a public IP → internet.
Short definition:
NAT allows private servers to access the internet **without exposing them**.

96. What is a routing table?

A **routing table** defines **where traffic should go** based on IP destinations.

What it contains:

- Destination CIDR (example: `0.0.0.0/0`)
- Target (IGW, NAT, local, VPC Endpoint)
- Subnet associations

Why it matters:

Without correct routes, resources **cannot talk** to each other or to the internet.
Simple line:
Routing table = GPS map for network traffic.

---

97. What is a default gateway?

The **default gateway** is the path used when the destination is **outside your network**.

In AWS:

- For **public subnets**, the default gateway = **Internet Gateway (IGW)**
- For **private subnets**, default gateway = **NAT Gateway**

Purpose:

When traffic doesn't match any specific route, it's forwarded through the default gateway to reach external networks.
One-liner:
Default gateway = "Use this route for anything not inside my network."

Intermediate Level

---

98. Can NAT work both directions?

No. NAT works only one way — outbound.

Meaning:

- Servers in **private subnet** → **internet** (YES)
- Internet → private subnet (NO)

NAT only allows **outgoing connections**, not incoming ones.
Why? It's designed for **security**, so private resources stay hidden.
One-liner: NAT = outbound-only internet access.

99. NAT Gateway vs Internet Gateway?

NAT Gateway	Internet Gateway (IGW)
• Used by **private subnets** • Allows **outbound** internet access • No inbound traffic allowed • Charges per GB • Fully managed by AWS	• Used by **public subnets** • Allows **inbound AND outbound** internet access • Needed for public-facing EC2, ALB, etc. • Free service

Simple difference:
NAT = private → internet
IGW = public ↔ internet (two-way)

100. Why do private subnets need NAT Gateway?

Because private subnets **don't have a direct route** to the internet.

Private EC2 needs internet for:

- OS updates

- Package downloads
- Calling external APIs
- License validations

By using NAT:

- EC2 gets **outbound** access
- But stays **unreachable** from the internet

One-liner:
Private subnets need NAT so they can download things without being exposed.

Advanced Level

---

101. Traffic flow from private EC2 → Internet

Here's the exact sequence:

1. EC2 (private subnet) sends request → default route
2. Route table sends traffic → **NAT Gateway**
3. NAT Gateway translates private IP → public IP
4. NAT sends traffic → Internet Gateway
5. Internet Gateway forwards → internet
6. Return traffic follows same path
7. NAT remembers the connection → sends response back to the correct EC2

Key idea:
NAT is the middleman that hides your private EC2.

102. NAT Gateway vs NAT Instance vs Internet Gateway

NAT Gateway	NAT Instance	Internet Gateway (IGW)
• Fully managed • Scales automatically • High availability • Recommended for production	• EC2 machine acting as NAT • Needs manual patching, scaling, HA • Used rarely today	• For **public subnets** • Allows direct public internet access • Supports inbound traffic

Comparison summary:
NAT Gateway → private outbound
NAT Instance → old method
IGW → public inbound/outbound

103. What is 0.0.0.0/0?

It means **"all IPv4 addresses in the world."**

Used for:

- Default routes
- Allowing all outbound internet

- Security rules (dangerous if used inbound!)

Example:
`0.0.0.0/0 → IGW`
Means: any destination → internet.
One-liner:
0.0.0.0/0 = "everywhere."

104. How routing priority works?

Routing follows **longest-prefix match** rule:

- More specific route = higher priority
- Less specific route = fallback

Example:

Route 1: `10.0.0.0/16 → local`
Route 2: `10.0.1.0/24 → NAT`
Traffic to `10.0.1.5` matches both,
but **/24** is more specific → wins.
If no match is found → use **default route (0.0.0.0/0)**.
Simple rule:
More specific CIDR > general CIDR.

---

9. Docker & Kubernetes Networking — Interview Questions

🟦 Docker – Beginner Level

---

105. What is a Docker bridge network?

A **bridge network** is Docker's default virtual network that allows containers on the **same host** to communicate with each other.

Simple view:

- Like a small private LAN inside your machine
- Containers get their **own IP addresses**
- They can talk to each other using container names

Use case:
App container ↔ Database container on same host.

106. What is the default Docker network?

The default network is named **bridge**.
When you run:

```
docker run nginx
```

Docker connects it automatically to the **default bridge network** unless you specify a custom network.
Why important? That's where containers get internal connectivity.

107. Bridge vs Host network?

Bridge Network	Host Network
• Containers get **their own private IP** • Communication via NAT • Need port mapping to access container from outside • More isolation, more secure	• Container uses **host's network directly** • No port mapping required • No isolation → better performance

Simple difference:
Bridge = isolated
Host = same network as host machine

108. Why do we need overlay network?

Overlay networks allow containers running on **different machines** to communicate as if they are on the same local network.

Use case:

- Multi-node Docker Swarm
- Kubernetes cluster network
- Distributed microservices

One-liner:
Overlay = network that spans multiple nodes in a cluster.

🟦 Docker – Intermediate Level

---

109. How does Docker map ports?

Docker uses **port mapping** to expose a container's port to the host machine.
Example:

```
docker run -p 8080:80 nginx
```

Meaning:

- Host port **8080** → Container port **80**

Docker sets up:

- NAT rules
- Forwarding rules

So traffic hitting 8080 goes to the container.

One-liner: Port mapping forwards host ports → container ports.

---

110. Why is overlay network used for multi-node clusters?

Because containers on different machines **cannot reach each other** using a normal bridge network.
Overlay network:

- Creates a **virtual network across all nodes**
- Encapsulates traffic (VXLAN)
- Ensures service discovery
- Ensures encrypted cross-node communication

Simple answer: Overlay allows multi-node container communication.

---

111. What are Docker network drivers?

Network drivers decide **how Docker creates and manages container networks**.

Common Docker network drivers:

Driver	Purpose
bridge	Default local container network
host	Use host network directly
overlay	Multi-node communication
macvlan	Containers get their own MAC + appear like real machines
none	No networking

One-liner:
Network drivers = plugins that define how container networking works.

Kubernetes – Beginner

---

112. Does each Pod get its own IP?

Yes.
Every Pod gets a **unique IP address** inside the cluster.

Why?

So containers inside different Pods can communicate **directly**, without port conflicts.

Golden rule: "Each Pod has its own IP, and Pods can talk to each other directly."

113. What is ClusterIP?

ClusterIP is the **default Kubernetes Service type**.
It exposes a service **inside the cluster only**.

Use case:

- Pod-to-Pod communication

- Internal microservices
- Databases, internal APIs

Not accessible from the internet.

114. What is NodePort?

NodePort exposes a service on a **port on every worker node**.
Example:
NodePort might open something like:

```
<nodeIP>:30080
```

This allows **external access**, but:

- Fixed port range (30000–32767)
- Not ideal for production

115. What is LoadBalancer?

LoadBalancer creates a **cloud provider load balancer** (AWS ELB, GCP LB, Azure LB) and routes external traffic → service → pods.

Good for:

- Public web apps
- Production workloads

116. What is Ingress?

Ingress is a **smart Layer-7 router** for HTTP/HTTPS.

It allows:

- Domain-based routing (api.example.com)
- Path-based routing (/login → service A, /cart → service B)
- SSL termination

Ingress is preferred over multiple LoadBalancers.

Kubernetes – Intermediate

---

117. How do pods communicate without NAT?

Kubernetes networking rule:

"All Pods must be able to reach each other directly without NAT."

This is done by:

- CNI plugins (Calico, Flannel, Weave, Cilium)
- Routing rules
- Overlay / BGP networks

Each Pod gets a **real, routable cluster IP**, so NAT is not needed.

118. What is kube-proxy?

`kube-proxy` handles **service networking**.

It manages:

- ClusterIP routing
- Load balancing across Pods
- NAT rules & iptables/IPVS

Simple: kube-proxy makes Services work.

119. Why are Pod IPs not stable?

Because Pods are **ephemeral** (temporary).
Reasons a Pod IP changes:

- Pod restarts
- New deployment rollout
- Scaling events
- Node failure

That's why **Services** exist — they give a stable virtual IP.

120. NodePort vs LoadBalancer

Feature	NodePort	LoadBalancer
Access	Opens port on each node	Creates cloud LB
Cost	Free	Costs money
Best for	Testing	Production
Scaling	Manual	Auto + cloud-managed

121. Why Ingress when LoadBalancer exists?

Because LoadBalancer only forwards **all traffic to one service**.
Ingress allows:

- Multiple domains on one LoadBalancer
- Path-based routing
- SSL termination
- Rate limiting / rewriting / auth

One LoadBalancer + Ingress = smarter + cheaper.

Kubernetes – Advanced

---

122. Traffic flow: Internet → Ingress → Pod

Full flow:

1. User hits domain (e.g., example.com)
2. DNS points to cloud LoadBalancer
3. LoadBalancer forwards to Ingress Controller

4. Ingress checks rules (path/domain)
5. Ingress sends traffic to Service (ClusterIP)
6. Service load balances across Pods
7. Pod receives the request
8. Response follows the same path back

123. What is a CNI plugin?

CNI = **Container Network Interface**
A CNI plugin is responsible for:

- Assigning Pod IPs
- Setting up routing
- Connecting Pods across nodes
- Ensuring Pod-to-Pod communication

Without a CNI, Pods cannot talk to each other.

124. How do CNI plugins (Calico/Flannel/Cilium) work?

Calico

- Uses **BGP routing**
- Very fast
  Great for enterprises

Flannel

- Uses **overlay networks (VXLAN)**
- Simple and lightweight

Cilium

- Uses **eBPF**
- High performance
- Deep security control

Summary:
Each CNI decides *how* Pod networks are created and routed.

125. Pod networking vs Service networking

Pod Networking	Service Networking
● Pod gets IP ● Pod-to-Pod communication ● Handled by CNI	● Provides a stable virtual IP (ClusterIP) ● Load balances traffic across Pods ● Handled by kube-proxy

Pod IP = changes
Service IP = stable

126. How does Kubernetes handle internal service discovery?

Through:

1. **ClusterIP Services**
2. **Kube-DNS / CoreDNS**

Example:

`my-service.my-namespace.svc.cluster.local`

DNS automatically resolves the Service name → ClusterIP
ClusterIP → distributes traffic across Pods.
Simple: Kubernetes uses DNS + Services for discovery.

---

10. Tools Every DevOps Engineer Must Know — Interview Questions

---

Basic

---

127. What does `ping` do?

`ping` checks if a machine is **reachable** on the network.

What it tests:

- Network connectivity
- DNS resolution
- Round-trip latency (how long packets take)

How it works:

Ping sends **ICMP echo requests** → **waits for echo replies**.

If ping works → machine is reachable.

If not → network, firewall, or DNS issues.

128. How do you check DNS records (`nslookup` / `dig`)?

These tools help you verify **domain → IP mappings**.

`nslookup example.com`

Shows:

- A record
- DNS server used

`dig example.com`

Shows detailed DNS data:

- A / AAAA records
- CNAME
- TTL
- Query time

Why used?

- Troubleshoot website issues
- Check if DNS propagated
- Verify domain/IP mapping

129. Difference between `ping` and `traceroute`?

Tool	Purpose
ping	Tests if a host is reachable + measures latency
traceroute	Shows the full **path** (hops) packets take

Key idea:

- Ping tells **"Can I reach it?"**
- Traceroute tells **"How do I reach it?"**

Traceroute helps find:

- Network delays
- ISP routing issues
- Where packets get dropped

130. How to check which service uses a port?

On Linux:

```
sudo lsof -i :<port>
sudo netstat -tulpn | grep <port>
ss -tulpn | grep <port>
```

Example:

```
sudo lsof -i :3306   → MySQL
sudo lsof -i :22     → SSH
```

Helps in:

- Debugging port conflicts
- Checking if a service is running
- Security audits

131. What does `curl` test?

`curl` tests **HTTP/HTTPS endpoints**.

What it can check:

- Whether a website/API is reachable
- HTTP status codes (200, 404, 500)
- API responses
- SSL certificates
- Load balancer / reverse proxy issues

Examples:

```
curl https://google.com
curl -I https://example.com   # headers only
curl -v https://api.test.com  # verbose debug
```

Why essential in DevOps?

Because APIs, microservices, reverse proxies, and load balancers all rely on HTTP — and curl is the fastest way to test them.

132. How to test if port 3306 is open remotely?

Port **3306 = MySQL**.
To check if it's reachable from another machine:

Method 1: Using `nc` (netcat)

```
nc -vz <SERVER_IP> 3306
```

Success → port open
Failure → blocked by firewall/Security Group

Method 2: Using `telnet`

```
telnet <SERVER_IP> 3306
```

Method 3: Using `nmap`

```
nmap -p 3306 <SERVER_IP>
```

What this tells you:

- Is MySQL reachable?
- Is the port blocked by SG, NACL, firewall, VPC rules?
- Is the DB in a **private subnet** (common in AWS)?

133. What is `tcpdump`?

`tcpdump` is a **packet capture** and **network debugging** tool used in Linux.

What it does:

- Captures real-time network packets
- Shows source/destination IPs
- Shows protocols, ports, flags
- Helps debug traffic problems

Example:

```
sudo tcpdump -i eth0
sudo tcpdump port 443
sudo tcpdump host 10.0.1.5
```

Why DevOps uses it:

- Diagnose connection failures
- Check if packets reach a server
- Debug load balancer/ingress issues
- Analyze DNS or SSL problems

---

134. Wireshark vs tcpdump?

Feature	Wireshark	tcpdump
Type	Graphical tool	CLI tool
Usage	Deep packet analysis	Quick packet capture
Output	Visual diagrams, filters	Text output
Best for	Desktop visualization	Servers, SSH debugging

Simple explanation:

- **tcpdump** → lightweight, used on Linux servers
- **Wireshark** → rich UI, used on laptops for deep debugging

135. What does `netcat` do?

`netcat (nc)` is the **Swiss Army knife of networking**.

What it can do:

- Test open ports

  ```
  nc -vz <IP> <PORT>
  ```

- Create TCP/UDP connections
- Transfer files
- Act as a simple server
- Debug networking routes

Why DevOps uses it:

- Check port connectivity
- Simulate services
- Debug firewall/SG issues
- Ping using TCP (when ICMP is blocked)

136. How to debug APIs using `curl`?

1. Check if API is reachable

```
curl https://api.example.com
```

2. View only headers

```
curl -I https://api.example.com
```

3. Debug SSL issues

```
curl -v https://api.example.com
```

4. Send POST data

```
curl -X POST -d '{"name":"kumar"}' -H "Content-Type: application/json"
https://api.example.com/create
```

5. Check response time

```
curl -w "@curl-format.txt" -o /dev/null -s https://api.example.com
```

6. Check if Load Balancer is responding

```
curl -I http://LB-DNS
```

Why important in DevOps?

- Debug microservices
- Validate deployments
- Check LB/Ingress routing
- Validate API Gateway responses
- Test internal/private APIs

Advanced

137. Capture only HTTPS packets with tcpdump?

Q: How do you capture only HTTPS traffic?
A:

```
tcpdump -i any port 443 -w https.pcap
```

Filters packets on port 443 and saves them for analysis.

138. Explain TCP handshake via Wireshark.

Q: What does the 3-way handshake look like?
A:

- **SYN** → Client requests connection
- **SYN-ACK** → Server acknowledges
- **ACK** → Client confirms
  Wireshark clearly shows this as three sequential packets.

139. Inspect packets inside Kubernetes node.

Q: How do you capture traffic inside a K8s node?
A:
Use tcpdump on node interface:

```
tcpdump -i eth0
```

Or capture pod traffic via container PID namespace:

```
nsenter -t <pid> -n tcpdump -i any
```

---

140. Diagnose DNS latency issues.

Q: How to investigate slow DNS?
A:

Measure DNS resolution time:

```
dig google.com +stats
```

Check network hops for delay:

```
traceroute <dns-server>
```

Capture DNS packets:

```
tcpdump port 53
```

Look for high response times or retries.