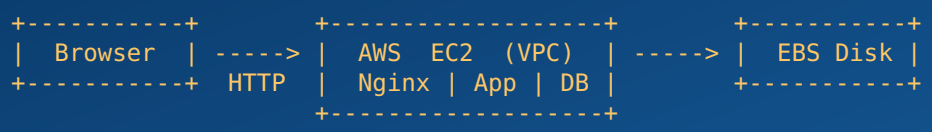
STUDY MATERIAL • HANDS-ON LAB GUIDE

AWS Cloud Computing: Hands-On Handbook

Networking, Linux, EC2, Web Servers, Application Stacks & EBS — explained step by step for engineering students

- Networking Fundamentals • OSI & TCP/IP Models
- Linux Basic & Advanced Commands
- Amazon EC2 • Nginx & Apache
- Static Website Hosting • Configuring Nginx & Apache
- Domain Names, DNS & Subdomains • Dynamic Website Hosting
- WordPress, Flask, Express • LAMP / LEMP / MEAN / MERN
- Amazon EBS Volumes & Snapshots
- EC2 Operations • Elastic IP, AMI & Vertical Scaling



PREPARED BY

Ravindra Bagale

Cloud Trainer

in linkedin.com/in/ravindra-bagale

Amazon Linux 2023 • Ubuntu 22.04 / 24.04 • CentOS Stream 9 | Copy-paste ready commands, diagrams, lab exercises and quick-revision summaries

Table of Contents

A Note from Your Trainer

Namaskar mitranno! (Hello, friends!)

Welcome! I have written this handbook the same way I teach in the classroom — slowly, step by step, and always starting with the question "Why do we even need this?" before we touch a single command.

Many of you know some programming already — C, Java, Python or JavaScript — but the words cloud, Linux server, security group or reverse proxy may still feel new. Kalji karu naka (don't worry) — that is perfectly fine. By the time you finish this book, you will launch your own servers on AWS, secure them, host static and dynamic websites, deploy complete application stacks and manage storage — exactly the kind of work cloud and DevOps engineers do every day.

A few requests from my side (dhyan do, these matter):

- **Type the commands yourself.** Copy-paste is allowed (every command box is copy-paste ready), but read each line and ask yourself what it does. Understanding is more important than speed. Lakshat theva (keep this in mind).
- **Break things in your lab.** Delete a config line, stop a service, block a port — then fix it. Troubleshooting is the real skill, and you learn it only by making mistakes in a safe environment.
- **Don't memorise, understand.** In interviews, nobody asks you to recite a command. They ask why it is needed and what happens when it fails. That is why every chapter ends with common mistakes and interview questions.
- **Switch off what you start.** Stop or terminate your lab instances when you finish. A forgotten server is the most common way students get a surprise bill.

Every topic follows the same simple pattern that I use in my sessions — ekdum simple aahe:

Step	Question it answers	Example
Why	Why do we need this? What problem does it solve?	Why use EC2 instead of buying a physical server?
What	What exactly is it? What are its parts?	What is an AMI, a key pair, a security group?
How	How do I actually do it, step by step?	Commands to launch, connect and configure

So open your laptop, keep a terminal ready, and let's learn cloud computing together. Chala, suru karuya! (Come on, let's begin!)

All the best, mitranno!

Ravindra Bagale
Cloud Trainer
linkedin.com/in/ravindra-bagale

Before we begin: conventions used in this book

- Commands you type are shown in grey boxes. They are **copy-paste ready**. Lines beginning with `#` inside a code block are comments — they are ignored by the shell.
- `$` at the start of a line in an example means "normal user prompt"; do **not** type the `$`. In most code blocks we have omitted the prompt so you can paste directly.
- Words in `<ANGLE_BRACKETS>` or `YOUR_...` must be replaced with your own value, e.g. `<PUBLIC_IP>` becomes `13.233.10.25`.
- `sudo` runs a command as the administrator (root). Most server set-up commands need it.

✔ Tip boxes

Green boxes give shortcuts and good habits that save time.

i Note boxes

Blue boxes give extra background information or differences between operating systems.

⚠ Warning boxes

Orange boxes warn you about mistakes that can break your server, lose data or cost money.

The three operating systems we will use together

In our labs we will practise on three popular Linux distributions that are available as free AMIs (machine images) on AWS:

Item	Amazon Linux 2023	Ubuntu 22.04 / 24.04 LTS	CentOS Stream 9
Family	Red Hat–like (Fedora based)	Debian	Red Hat (RHEL upstream)
Package manager	`yum`	`apt`	`yum`
Default SSH user	`ec2-user`	`ubuntu`	`ec2-user` (older CentOS AMIs: `centos`)
Apache package / service	`httpd`	`apache2`	`httpd`
Nginx default web root	`/usr/share/nginx/html`	`/var/www/html`	`/usr/share/nginx/html`
Apache default web root	`/var/www/html`	`/var/www/html`	`/var/www/html`
Host firewall	none active by default	`ufw` (inactive by default)	`firewalld` (may be installed)
SELinux	permissive/disabled by default on AMI	not used (AppArmor)	**enforcing**
Default root filesystem	XFS	ext4	XFS

> ⚠ **Cost awareness**
>
> AWS bills by the second/hour for running resources. Always **stop or terminate** lab instances, delete unused EBS volumes and snapshots, and release unused Elastic IPs when you finish. Set a **billing alarm / budget** in the AWS Billing console on day one. AWS free-tier offers change from time to time — always check the official AWS Free Tier page for the current rules on your account.

Chapter 1: Networking Fundamentals

Namaskar mitranno! Before we launch any server on the cloud, let's first understand how computers find and talk to each other. Every cloud concept — security groups, VPCs, subnets, Elastic IPs, load balancers — is built on basic networking. In my sessions, students often want to jump straight to EC2, but I always say: networking first, cloud later. If you understand this chapter well, AWS networking becomes ekdum simple.

1.1 Why networking matters in the cloud

When you type `http://13.233.10.25` in a browser, your laptop must know **which machine** to contact (the IP address) and **which program** on that machine should answer (the port). On AWS you will constantly:

- open or close **ports** in security groups (e.g. allow 22 for SSH, 80 for HTTP),
- choose **CIDR ranges** for VPCs and subnets (e.g. `10.0.0.0/16`),
- decide between **public** and **private** IP addresses,
- attach an **Elastic IP** so your server's address does not change.

1.2 IP address — what and why

An **IP (Internet Protocol) address** is a logical number that uniquely identifies a device (host) on a network — like the postal address of a house. Routers use it to deliver packets to the correct destination.

- **IPv4** example: `192.168.1.10` (32 bits)
- **IPv6** example: `2406:da1a:8e8:e801::10` (128 bits)

> **Analogy**
>
> Think of an apartment complex. The **IP address** is the building address (e.g. Shivaji Nagar, Pune). The **port** is the flat number inside the building. The postman (router) delivers to the building; the watchman (operating system) sends the letter to the right flat (application). And just like the **pin code** (e.g. 411005) tells India Post which area a letter belongs to, the **network part** of an IP address tells routers which network the packet belongs to, while the **host part** identifies the exact machine.
>
> Lakshat theva — this is only an analogy. The literal mechanism is: every packet carries a destination IP in its IP header; each router compares that IP with its routing table (longest matching prefix wins) and forwards the packet to the next hop. When the packet reaches the destination host, the OS reads the destination **port** in the TCP/UDP header and hands the data to the process listening on that port.

1.3 Ports

Now, once the packet reaches the right machine, how does it reach the right program? Bagha, that is the job of the port.

A **port** is a 16-bit number (0–65535) that identifies a specific **process/service** on a host. One server with one IP can run a web server (port 80), SSH (port 22) and MySQL (port 3306) at the same time because each listens on a different port. The combination `IP:port` (e.g. `13.233.10.25:443`) is called a **socket**.

Range	Name	Meaning
0 – 1023	Well-known ports	Reserved for standard services; need root to bind on Linux
1024 – 49151	Registered ports	Used by applications (MySQL, MongoDB, dev servers)
49152 – 65535	Dynamic / ephemeral	Temporary client-side ports chosen by the OS

Common ports you must remember

Port	Protocol	Service	Where you will see it
20, 21	TCP	FTP (data, control)	Legacy file transfer (avoid; insecure)
22	TCP	SSH / SCP / SFTP	Logging in to EC2, copying files
23	TCP	Telnet	Legacy, insecure — never open
25	TCP	SMTP	Sending e-mail (AWS throttles port 25 by default)
53	UDP/TCP	DNS	Domain name lookups
67, 68	UDP	DHCP	Automatic IP assignment
80	TCP	HTTP	Websites (unencrypted)
110 / 143	TCP	POP3 / IMAP	Receiving e-mail
443	TCP	HTTPS	Websites with TLS/SSL
3000	TCP	Node.js / React dev server	Express apps, `npm start`
3306	TCP	MySQL / MariaDB	Database
3389	TCP	RDP	Windows Remote Desktop
4200	TCP	Angular dev server	`ng serve`
5000	TCP	Flask dev server	`flask run`
5432	TCP	PostgreSQL	Database
6379	TCP	Redis	Cache
8000	TCP	Gunicorn / Django dev server	Python apps
8080	TCP	Alternate HTTP / Tomcat / Jenkins	Java apps, proxies
27017	TCP	MongoDB	NoSQL database

> ⚠ **Security rule of thumb**
>
> Only ports **22, 80 and 443** normally need to be open to the internet. Application ports (3000, 5000, 8080) should sit **behind a reverse proxy** (Nginx/Apache), and database ports (3306, 27017) should **never** be open to `0.0.0.0/0`.

1.4 IPv4 in detail

This is the most important section of the chapter, so dhyan do. Once IPv4 is clear, CIDR and subnets in AWS will feel very natural.

Structure

An IPv4 address is **32 bits**, written as four **octets** (8 bits each) in dotted-decimal form. Each octet ranges from 0 to 255.

```
   192    .    168    .     1     .    10
11000000   10101000   00000001   00001010
|<-8 bits->|
|<---------------- 32 bits ---------------->|
```

Total possible IPv4 addresses = 2^32 ≈ **4.3 billion** — not enough for today's world of phones, laptops and IoT devices. That is the main reason for NAT, private IPs and IPv6.

Every IPv4 address has two parts:

- **Network portion** — identifies the network (like the street name).
- **Host portion** — identifies the device on that network (like the house number).

The **subnet mask** tells which bits are network and which are host. Example: `255.255.255.0` means the first 24 bits are network bits.

Classful addressing (historical but asked in exams)

Class	First octet range	Leading bits	Default mask	Networks / Hosts per network	Use
A	1 – 126	0	255.0.0.0 (/8)	126 / ~16.7 million	Very large networks
B	128 – 191	10	255.255.0.0 (/16)	16,384 / 65,534	Medium networks
C	192 – 223	110	255.255.255.0 (/24)	~2 million / 254	Small networks
D	224 – 239	1110	—	—	Multicast
E	240 – 255	1111	—	—	Experimental / reserved

> **i What about 127?**
>
> `127.0.0.0/8` is reserved for **loopback**. `127.0.0.1` (also called `localhost`) always means "this same machine". When an app listens on `127.0.0.1:3000`, it is reachable only from the server itself — that is exactly what we want behind a reverse proxy.

CIDR (Classless Inter-Domain Routing)

Classes waste addresses (a company needing 500 hosts would get a whole Class B of 65,534). **CIDR** replaced classes. We write the address followed by `/n`, where **n = number of network bits**.

CIDR	Subnet mask	Total addresses	Usable hosts (traditional)	Usable in AWS subnet
/16	255.255.0.0	65,536	65,534	65,531
/20	255.255.240.0	4,096	4,094	4,091
/24	255.255.255.0	256	254	251
/25	255.255.255.128	128	126	123
/26	255.255.255.192	64	62	59
/27	255.255.255.224	32	30	27
/28	255.255.255.240	16	14	11
/32	255.255.255.255	1	single host	—

Formula: **total addresses = 2^(32 − n)**, usable hosts = 2^(32 − n) − 2 (network address and broadcast address are reserved).

> **i AWS reserves 5 addresses**
>
> In every AWS subnet the first four and the last IP are reserved: network address, VPC router (.1), DNS (.2), future use (.3) and the last (broadcast) address. So a `/24` subnet gives **251** usable IPs. The smallest subnet AWS allows is `/28` and the largest VPC is `/16`.

`0.0.0.0/0` means **"every IPv4 address"** — you will see it in security groups ("anywhere") and route tables (default route).

Worked subnetting example

Chala, aata ek example karuya together — take a pen and paper, don't just read.

Problem: Your college lab is given the network `192.168.10.0/24`. Divide it into **4 equal subnets** for four labs. Find each subnet's network address, usable range and broadcast address.

Step 1 — borrow bits. 4 subnets need 2 extra bits ($2^2 = 4$). New prefix = 24 + 2 = **/26**.

Step 2 — block size. Addresses per subnet = 2^(32−26) = **64**. Mask = `255.255.255.192`.

Step 3 — list subnets (increase the last octet in steps of 64):

Subnet	Network address	First usable	Last usable	Broadcast	Usable hosts
Lab 1	192.168.10.0/26	192.168.10.1	192.168.10.62	192.168.10.63	62
Lab 2	192.168.10.64/26	192.168.10.65	192.168.10.126	192.168.10.127	62
Lab 3	192.168.10.128/26	192.168.10.129	192.168.10.190	192.168.10.191	62
Lab 4	192.168.10.192/26	192.168.10.193	192.168.10.254	192.168.10.255	62

Step 4 — check with binary. Host `192.168.10.100`: last octet 100 = `01100100`. The first two bits (`01`) are the subnet bits → subnet 1 (counting from 0) → it belongs to **Lab 2 (192.168.10.64/26)**.

> **✔ Quick trick for block size**
>
> Block size = 256 − (last non-255 octet of the mask). For `/26` the mask is `255.255.255.192`, so block = 256 − 192 = **64**. Subnets start at 0, 64, 128, 192.

AWS example: A VPC `10.0.0.0/16` is commonly split into subnets `10.0.1.0/24` (public, AZ-a), `10.0.2.0/24` (public, AZ-b), `10.0.11.0/24` (private, AZ-a) and `10.0.12.0/24` (private, AZ-b).

You can check subnet maths on Linux with the `ipcalc` tool (package `ipcalc`):

```
ipcalc 192.168.10.64/26
```

> **RB Ravindra's Tip**
>
> Subnetting becomes easy only with practice, not by reading. Do five problems daily for one week using the block-size trick (256 − mask octet), and verify each answer with `ipcalc`. Interviewers love asking you to split a /24 into four subnets on a whiteboard — tumhi ready asal (you will be ready).

1.5 IPv6

Why IPv6 is needed

- IPv4 has only ~4.3 billion addresses, and the regional registries have run out of fresh IPv4 blocks.
- IPv6 has **2^128 ≈ 3.4 × 10^38** addresses — enough to give every device its own public address, removing the need for NAT.
- Built-in features: simpler header, auto-configuration (SLAAC), better multicast, and no broadcast.
- On AWS, **public IPv4 addresses are charged per hour**, whereas IPv6 addresses are free — another practical reason to learn IPv6.

Format

IPv6 is **128 bits**, written as **8 groups of 4 hexadecimal digits** (16 bits each), separated by colons:

```
2001:0db8:0000:0000:0000:ff00:0042:8329
|16b|                                  8 groups x 16 bits = 128 bits
```

Shortening rules

1. **Leading zeros** in any group may be removed: `0db8` → `db8`, `0042` → `42`, `0000` → `0`.

2. **One** continuous run of all-zero groups may be replaced by `::` — only **once** per address (otherwise the length would be ambiguous).

Full form	Short form
`2001:0db8:0000:0000:0000:ff00:0042:8329`	`2001:db8::ff00:42:8329`
`fe80:0000:0000:0000:0202:b3ff:fe1e:8329`	`fe80::202:b3ff:fe1e:8329`
`0000:0000:0000:0000:0000:0000:0000:0001`	`::1` (loopback)
`2001:0db8:0000:0001:0000:0000:0000:0001`	`2001:db8:0:1::1`

Important IPv6 address types

Prefix	Type	IPv4 equivalent
`::1/128`	Loopback	127.0.0.1
`fe80::/10`	Link-local (auto on every interface)	169.254.x.x
`fc00::/7` (`fd00::/8` in practice)	Unique local (private)	RFC1918 ranges
`2000::/3`	Global unicast (public)	Public IPv4
`ff00::/8`	Multicast	224.0.0.0/4
`::/0`	All IPv6 addresses	0.0.0.0/0

In AWS, a VPC can get an Amazon-provided `/56` IPv6 block and each subnet a `/64`. Remember to add `::/0` rules in security groups if you want IPv6 clients to reach your server.

1.6 Static vs dynamic IP

Feature	Static IP	Dynamic IP
Assigned by	Manually by an admin (or reserved)	Automatically by a **DHCP** server
Changes?	Never, unless changed manually	May change on reboot / lease expiry
Best for	Servers, DNS records, printers, VPN endpoints	Laptops, phones, home users
Cost	Often extra (ISPs charge)	Usually included
AWS equivalent	**Elastic IP**	Auto-assigned public IPv4

DHCP (Dynamic Host Configuration Protocol) hands out IP address, subnet mask, default gateway and DNS server in four steps: **D**iscover → **O**ffer → **R**equest → **A**cknowledge (DORA).

1.7 Public vs private IP

A **public IP** is globally unique and routable on the internet. A **private IP** is used only inside a local network (home, college, VPC) and is **not routable** on the internet. RFC 1918 reserves three private ranges:

Class	Private range (RFC 1918)	CIDR	Typical use
A	10.0.0.0 – 10.255.255.255	`10.0.0.0/8`	Large companies, AWS VPCs
B	172.16.0.0 – 172.31.255.255	`172.16.0.0/12`	AWS **default VPC** uses `172.31.0.0/16`
C	192.168.0.0 – 192.168.255.255	`192.168.0.0/16`	Home Wi-Fi routers

Other special ranges: `100.64.0.0/10` (carrier-grade NAT used by mobile ISPs) and `169.254.0.0/16` (link-local; on EC2, `169.254.169.254` is the **instance metadata service**).

Check your own IPs:

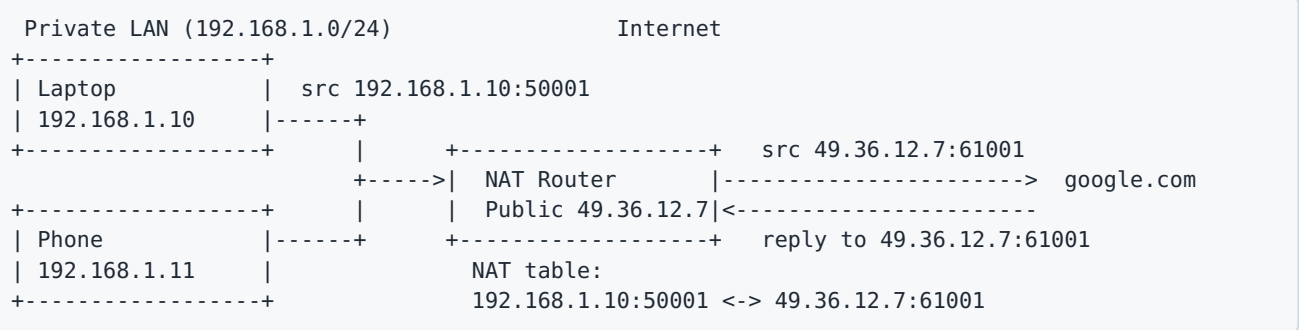
```
# Private IP(s) of this machine
ip -4 addr show
hostname -I

# Public IP as seen by the internet
curl -s https://checkip.amazonaws.com
```

1.8 NAT (Network Address Translation)

In my sessions, students often ask: "Sir, my laptop shows 192.168.1.5 but Google sees a different IP. Why?" The answer is NAT. Samjla ka? Let's see how it works.

Why: Hundreds of devices on a college network use private IPs, but the college may have only a few public IPs. **NAT** lets many private devices share one public IP.

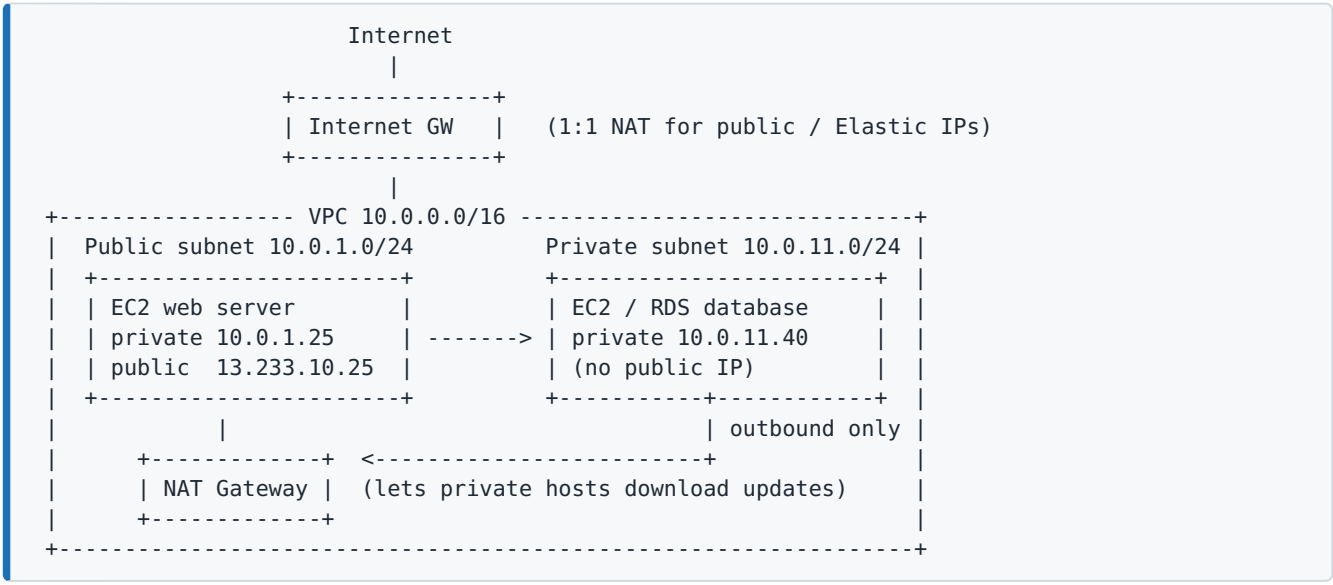
How: The router rewrites the source address of outgoing packets from the private IP to its public IP and keeps a translation table so replies go back to the correct device. The variant that also changes ports (many-to-one) is called **PAT** or NAT overload.



NAT type	Mapping	Example
Static NAT	1 private ↔ 1 public (fixed)	EC2 public IP / Elastic IP mapping
Dynamic NAT	private ↔ one from a pool of public IPs	Enterprise routers
PAT / NAT overload	many private → 1 public using ports	Home Wi-Fi, AWS **NAT Gateway**

> **RB Ravindra's Tip**
>
> When you design a VPC, never choose `192.168.0.0/16` — it clashes with most home and office Wi-Fi networks, and later VPN connections will break. I prefer something like `10.20.0.0/16` for each project. Plan your CIDR once, carefully; changing it later is painful.

1.9 How these concepts map to AWS



Networking concept	AWS feature	Key points
Private IP	Primary private IPv4 of an EC2 ENI	From the subnet CIDR; stays the same for the life of the instance (even across stop/start)
Dynamic public IP	**Auto-assigned public IPv4**	Released on **stop**; a **new one** is given on start. Lost for ever on terminate
Static public IP	**Elastic IP (EIP)**	Stays with your account until you release it; can be re-mapped to another instance. Charged hourly like all public IPv4 addresses
Private network	**VPC** with CIDR (e.g. `10.0.0.0/16`)	Choose from RFC 1918 ranges; `/16` to `/28`
Subnet	**Subnet** (lives in one AZ)	Public subnet = route `0.0.0.0/0` → Internet Gateway
NAT	**NAT Gateway** / Internet Gateway	IGW does 1:1 NAT for public IPs; NAT GW does many-to-one for private subnets
Firewall on ports	**Security Group** (stateful, instance level) and **Network ACL** (stateless, subnet level)	Allow 22 only from My IP; 80/443 from anywhere
DNS	**Route 53** (Chapter 9), and public DNS names like `ec2-13-233-10-25.ap-south-1.compute.amazonaws.com`	

✔ Mumbai and Hyderabad regions

For users in Maharashtra, the **Asia Pacific (Mumbai)** `ap-south-1` region gives the lowest latency. `ap-south-2` (Hyderabad) is the other Indian region.

> ⚠ **Public IP changes after stop/start**
>
> If your website stops working after you stop and start an instance, check the public IP — it has probably changed. Use an Elastic IP (or a load balancer / DNS name) for anything that must keep a fixed address — we practise this in Chapter 14.

Common Mistakes I See Students Make

- **Confusing IP and port.** Opening port 80 in the security group does not give your server a public IP, and having a public IP does not open any port. You need both.
- **Opening every port to 0.0.0.0/0 "just to test".** Then forgetting to close it. Databases like MySQL (3306) and MongoDB (27017) get attacked within hours.
- **Thinking 192.168.x.x or 10.x.x.x is reachable from the internet.** Private IPs never route on the internet; that is the whole point of RFC 1918.
- **Forgetting that the auto-assigned public IP changes after stop/start.** Students update their DNS or WordPress URL, stop the instance for the night, and the next day nothing works.
- **Using `::` twice in an IPv6 address** (e.g. `2001::1::5`) — invalid, because the length becomes ambiguous.
- **Counting all addresses as usable.** A /24 in AWS gives 251 usable IPs, not 256 or 254.

Interview Questions

Q1. What is the difference between a public and a private IP address?

A public IP is globally unique and routable on the internet. A private IP (RFC 1918: 10/8, 172.16/12, 192.168/16) is used inside local networks/VPCs and is not routable on the internet; it reaches the internet through NAT.

Q2. How many usable hosts are there in a /26 subnet? And in an AWS /26 subnet?

2^(32−26) = 64 addresses; 62 usable traditionally (minus network and broadcast). AWS reserves 5, so 59 usable.

Q3. What is CIDR and why was it introduced?

Classless Inter-Domain Routing writes a prefix length (/n) instead of fixed classes, allowing networks of any size and reducing address wastage and routing-table size.

Q4. Why do we need IPv6?

IPv4 has only ~4.3 billion addresses and is exhausted. IPv6 has 2^128 addresses, removes the need for NAT, and supports auto-configuration. On AWS, public IPv4 is also charged, while IPv6 is not.

Q5. What is the difference between an Elastic IP and an auto-assigned public IP on EC2?

The auto-assigned public IP is released when the instance stops and a new one is given on start. An Elastic IP is a static public IPv4 owned by your account until you release it and can be re-mapped between instances.

Q6. What is NAT? Where is it used in AWS?

NAT translates private source addresses to a public address (and back for replies). AWS uses 1:1 NAT at the Internet Gateway for public/Elastic IPs and many-to-one NAT in a NAT Gateway for private subnets.

Q7. Which ports are used by SSH, HTTP, HTTPS, MySQL and MongoDB?

22, 80, 443, 3306 and 27017 respectively.

Practice Questions and Lab Exercises

1. Differentiate between an IP address and a port with a real-life analogy.
2. List the default ports for SSH, HTTP, HTTPS, MySQL, MongoDB and a Flask development server.
3. Identify the class and default mask of: `10.4.5.6`, `172.20.1.1`, `200.1.2.3`, `230.0.0.5`.
4. A network `172.16.0.0/22` must be split into 8 equal subnets. Find the new prefix, block size and the range of the 3rd subnet.
5. Shorten `2406:da1a:0000:0000:0e00:0000:0000:0010` correctly. Why can `::` be used only once?
6. Which of these are private: `10.10.10.10`, `172.32.1.1`, `192.168.100.1`, `100.64.1.1`, `8.8.8.8`?
7. **Lab:** On any Linux machine run `ip addr`, `hostname -I` and `curl -s https://checkip.amazonaws.com`. Explain why the two IPs differ (hint: NAT).
8. Explain the difference between an auto-assigned public IP and an Elastic IP on AWS. What happens to each when you stop the instance?

★ Quick Revision

- **IP** = which host; **port** = which service. Socket = IP:port.
- Must-know ports: 22 SSH, 80 HTTP, 443 HTTPS, 3306 MySQL, 27017 MongoDB, 3000 Node, 5000 Flask, 8080 alt-HTTP.
- IPv4 = 32 bits, 4 octets; classes A/B/C/D/E; CIDR `/n` → 2^(32−n) addresses; AWS reserves 5 per subnet.
- IPv6 = 128 bits, 8 hex groups; drop leading zeros; `::` only once.
- Private (RFC 1918): `10/8`, `172.16/12`, `192.168/16`. Default VPC = `172.31.0.0/16`.
- NAT lets private hosts share a public IP. AWS: IGW (1:1), NAT Gateway (many:1).
- Auto public IP changes on stop/start; **Elastic IP** is static.

Chapter 2: The OSI Model and TCP/IP Model

Mitranno, let's first understand why we even need a layered model. When something breaks in production, a good engineer does not panic — he or she checks layer by layer. This chapter gives you that thinking pattern. Ekdum simple aahe, just stay with me.

2.1 Why do we need a reference model?

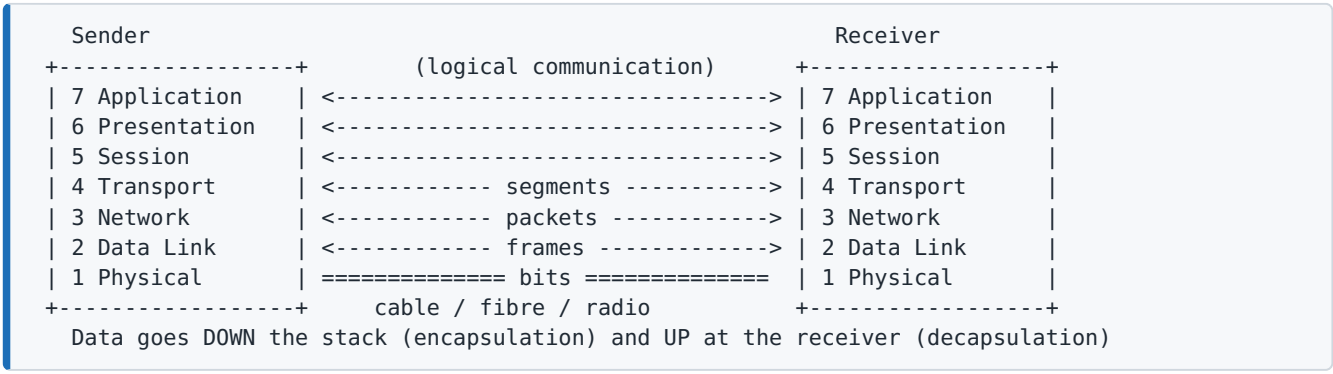
Networking involves cables, Wi-Fi signals, switches, routers, IP addresses, TCP connections, TLS encryption, HTTP requests and finally the web page. Different companies build each piece. A **layered model** divides this big problem into small, independent layers so that:

- each layer does one job and provides a service to the layer above it,
- vendors can build compatible products (a Cisco router works with a Dell laptop),
- troubleshooting becomes systematic ("Is it a cable problem (Layer 1) or a DNS problem (Layer 7)?").

The **OSI (Open Systems Interconnection)** model was published by ISO. It has **7 layers**. The real internet uses the simpler **TCP/IP model**, but OSI is the standard vocabulary — engineers say "Layer 4 load balancer" or "Layer 7 firewall" every day.

> **✔ Mnemonic**
>
> Top to bottom (7 → 1): **A**ll **P**eople **S**eem **T**o **N**eed **D**ata **P**rocessing — Application, Presentation, Session, Transport, Network, Data Link, Physical. Bottom to top (1 → 7): **P**lease **D**o **N**ot **T**hrow **S**ausage **P**izza **A**way.

2.2 The seven layers



#	Layer	Main function	Protocols / standards	Devices	Data unit (PDU)
7	Application	Interface for user applications; network services	HTTP, HTTPS, FTP, SMTP, DNS, SSH, DHCP, SNMP	Gateways, L7 load balancers (AWS ALB), WAF	Data / message

#	Layer	Main function	Protocols / standards	Devices	Data unit (PDU)
6	Presentation	Translation, encryption, compression, data format	TLS/SSL, JPEG, PNG, ASCII, UTF-8, MPEG	—	Data
5	Session	Establish, maintain, terminate sessions; checkpoints	NetBIOS, RPC, SQL session, PPTP	—	Data
4	Transport	End-to-end delivery, ports, segmentation, flow & error control	**TCP**, **UDP**	L4 load balancers (AWS NLB), firewalls	**Segment** (TCP) / Datagram (UDP)
3	Network	Logical addressing (IP), routing between networks	**IPv4, IPv6**, ICMP, ARP*, OSPF, BGP	**Router**, L3 switch	**Packet**
2	Data Link	Physical (MAC) addressing, framing, error detection on a link	Ethernet (802.3), Wi-Fi (802.11), PPP, VLAN (802.1Q)	**Switch**, bridge, NIC	**Frame**
1	Physical	Transmission of raw bits over the medium; voltages, connectors	RJ-45, fibre, DSL, Bluetooth radio, 802.11 PHY	**Hub**, repeater, cables, modem	**Bit**

* ARP works between Layer 2 and 3 (it maps IP → MAC), so books place it in either.

Layer-by-layer in simple words

- **Layer 7 – Application:** What the user's program speaks. The browser speaks HTTP; `ssh` speaks the SSH protocol. Not the application itself (Chrome), but the protocol it uses.
- **Layer 6 – Presentation:** Makes data understandable to both sides — character encoding (UTF-8), compression (gzip) and encryption (TLS). HTTPS = HTTP + TLS.
- **Layer 5 – Session:** Opens, manages and closes a conversation (session). E.g. keeping you logged in, resuming a large download.
- **Layer 4 – Transport:** Adds **source and destination port numbers**. **TCP** is reliable (3-way handshake, acknowledgements, retransmission, ordering) — used by HTTP, SSH, MySQL. **UDP** is fast and connectionless — used by DNS, video calls, online games.
- **Layer 3 – Network:** Adds **source and destination IP addresses** and chooses the path (routing). Routers work here. `ping` (ICMP) tests this layer.
- **Layer 2 – Data Link:** Adds **MAC addresses** (48-bit hardware address like `0a:1b:2c:3d:4e:5f`) to move frames within one local network. Switches work here.
- **Layer 1 – Physical:** Converts frames into electrical signals, light pulses or radio waves.

TCP vs UDP

In my sessions, students often ask: "If TCP is reliable, why not use it everywhere?" Bagha — reliability has a cost (handshakes, acknowledgements, retransmissions). For a quick DNS query or a live video call, speed matters more than perfection.

Feature	TCP	UDP
Connection	Connection-oriented (3-way handshake SYN, SYN-ACK, ACK)	Connectionless

Feature	TCP	UDP
Reliability	Guaranteed delivery, ordering, retransmission	Best effort; may lose packets
Speed / overhead	Slower, 20-byte header	Faster, 8-byte header
Uses	Web (HTTP/1.1, HTTP/2), SSH, e-mail, databases	DNS, DHCP, VoIP, streaming, gaming, HTTP/3 (QUIC)

Encapsulation

As data moves down, each layer adds its own **header** (Layer 2 also adds a trailer):

```
L7-5   [                HTTP data                  ]
L4     [TCP hdr: ports ][        HTTP data          ]            = Segment
L3     [IP hdr: IPs][TCP hdr][      HTTP data        ]            = Packet
L2     [Eth hdr: MACs][IP][TCP][    HTTP data   ][FCS]            = Frame
L1     0101100101110100101010010101001010101010101...           = Bits
```

2.3 The TCP/IP model and comparison

OSI is the language we speak; TCP/IP is what actually runs the internet. Samjla ka? Let's compare them.

The **TCP/IP (Internet) model** is the practical model on which the internet runs. It has **4 layers** (some books show 5 by splitting the bottom layer).

```
       OSI Model                    TCP/IP Model
  +---------------------+      +----------------------+
  | 7 Application       |      |                      |
  | 6 Presentation      | ---> |  Application         |  HTTP, DNS, SSH, TLS
  | 5 Session           |      |                      |
  +---------------------+      +----------------------+
  | 4 Transport         | ---> |  Transport           |  TCP, UDP
  +---------------------+      +----------------------+
  | 3 Network           | ---> |  Internet            |  IP, ICMP, ARP
  +---------------------+      +----------------------+
  | 2 Data Link         | ---> |  Network Access      |  Ethernet, Wi-Fi
  | 1 Physical          |      |  (Link)              |
  +---------------------+      +----------------------+
```

Point	OSI model	TCP/IP model
Layers	7	4 (or 5)
Developed by	ISO (reference/theoretical)	US DoD / DARPA (practical)
Approach	Model first, protocols later	Protocols first, model later
Session & presentation	Separate layers	Merged into Application
Usage	Teaching, troubleshooting vocabulary	Actual internet implementation
Transport layer	Connection-oriented and connectionless	TCP (connection) and UDP (connectionless)

Where AWS services sit

Layer	AWS example
L7	Application Load Balancer (routes by URL/host), CloudFront, AWS WAF, Route 53 (DNS)
L4	Network Load Balancer, **Security Groups** (filter by protocol + port + IP)
L3	VPC, subnets, route tables, Internet/NAT gateways, Network ACLs
L1–L2	Managed by AWS (physical data centres, Nitro hardware, ENIs)

RB Ravindra's Tip

Think of the OSI model like sending a parcel through a courier: you pack the item (application data), write the flat number (port), write the full address with pin code (IP), the local delivery van uses the street route (MAC/data link), and the road itself is the physical layer. But in the exam and the interview, always explain the real mechanism — headers added at each layer (encapsulation) and removed at the receiver (decapsulation).

2.4 Walk-through: what happens when you open a website?

This is my favourite interview question, so dhyan do. Suppose you type `https://www.example.com` in your browser and press Enter. What happens in the next few hundred milliseconds? Let's walk through it together.

```
You ──► Browser ──► DNS lookup ──► TCP handshake ──► TLS handshake ──► HTTP GET
                                                                          │
Page shown ◄── Browser renders ◄── HTTP 200 + HTML/CSS/JS ◄── Web server ◄┘
```

1. **URL parsing (L7):** The browser finds the scheme (`https` → port 443), host (`www.example.com`) and path (`/`).
2. **DNS resolution (L7, over UDP 53):** The browser checks its cache, then the OS cache and `/etc/hosts`, then asks the configured DNS resolver (your ISP, `8.8.8.8`, or on EC2 the VPC resolver at `169.254.169.253`). The resolver queries root → `.com` TLD → authoritative name server and returns the IP, e.g. `93.184.215.14`.
3. **ARP (L2/L3):** Your laptop needs the MAC address of the default gateway (Wi-Fi router) to send the first frame; ARP finds it.
4. **TCP 3-way handshake (L4):** Your laptop picks an ephemeral port (e.g. 51544) and sends **SYN** to `93.184.215.14:443`; the server replies **SYN-ACK**; laptop sends **ACK**. Connection established.
5. **Routing (L3):** Packets travel through your router (NAT changes your private IP to the public IP), your ISP and many internet routers, each forwarding by destination IP.
6. **TLS handshake (L6):** Browser and server agree on encryption keys; the server presents its certificate, which the browser verifies.
7. **HTTP request (L7):** Browser sends `GET / HTTP/1.1` with headers `Host: www.example.com`, cookies, etc.

8. **Server processing:** On AWS this could be: Security group allows 443 → Nginx receives request → serves a static file or proxies to an app (Node/Flask/PHP) → app queries the database → response generated.
9. **HTTP response (L7):** `HTTP/1.1 200 OK` plus HTML. The browser then requests CSS, JS and images (often over the same connection).
10. **Rendering:** Browser builds the DOM, applies CSS, runs JavaScript and shows the page.
11. **Connection close / keep-alive:** Connection reused for more requests, then closed with FIN/ACK.

Now open your terminal and try this with me — you can see several of these steps yourself:

```
# Step 2: DNS lookup
dig +short www.example.com
nslookup www.example.com

# Step 5: path taken by packets (install traceroute if missing)
traceroute www.example.com

# Steps 4, 6, 7, 9 in detail (verbose curl)
curl -v https://www.example.com -o /dev/null
```

i Troubleshooting using layers

Website not opening from your laptop? Work bottom-up: Is the instance running (L1)? Does it have a public IP and route to an IGW (L3)? Does the **security group** allow port 80/443 (L4)? Is Nginx listening (`ss -tlnp`)? Does DNS point to the right IP (L7)? Does the app return errors (L7)?

RB Ravindra's Tip

When a website does not open, say your troubleshooting in layers, loudly, like a checklist: instance running? → route and public IP? → security group port? → service listening? → DNS correct? → app errors? Interviewers are impressed by this structured approach much more than by random guessing.

Common Mistakes I See Students Make

- **Mixing up the order of layers** or placing TCP at the network layer. Use the mnemonic and practise drawing it.
- **Saying "a switch works at Layer 3".** A normal switch works at Layer 2 (MAC addresses); a router works at Layer 3 (IP).
- **Thinking HTTPS is a different layer from HTTP.** HTTPS = HTTP (Layer 7) carried over TLS (Layer 6 in OSI terms).
- **Forgetting DNS in the website walk-through.** Without DNS, the browser does not even know which IP to connect to.
- **Assuming ping failure means the server is down.** ICMP may simply be blocked by the security group.

Interview Questions

Q1. Name the seven OSI layers and the PDU at layers 2, 3 and 4.

Physical, Data Link, Network, Transport, Session, Presentation, Application. PDUs: frame (L2), packet (L3), segment/datagram (L4).

Q2. Explain the TCP three-way handshake.

Client sends SYN, server replies SYN-ACK, client sends ACK. After this the connection is established and data can flow with sequence numbers and acknowledgements.

Q3. What is the difference between TCP and UDP? Give an example of each.

TCP is connection-oriented and reliable (HTTP, SSH, MySQL). UDP is connectionless and faster with no delivery guarantee (DNS, VoIP, video streaming).

Q4. At which layer do an AWS Application Load Balancer and a Network Load Balancer operate?

ALB works at Layer 7 (routes by host/path/headers). NLB works at Layer 4 (TCP/UDP, ports).

Q5. What happens when you type a URL in the browser?

URL parsing → DNS resolution → TCP handshake → TLS handshake → HTTP request → server/app/database processing → HTTP response → browser renders HTML/CSS/JS.

Q6. What is encapsulation?

Each layer adds its own header (and Layer 2 also a trailer) around the data from the layer above before passing it down; the receiver removes them in reverse order (decapsulation).

Practice Questions and Lab Exercises

1. Name the 7 OSI layers with one protocol and one device for each.
2. What is the PDU at the transport, network and data link layers?
3. Compare TCP and UDP. Why does DNS normally use UDP but SSH uses TCP?
4. Draw the OSI vs TCP/IP mapping diagram and write four differences.
5. At which layer does a switch operate? A router? An AWS Application Load Balancer? A Security Group?
6. Explain step by step what happens when you enter `https://www.amazon.in` in a browser.
7. **Lab:** Run `curl -v https://www.example.com -o /dev/null` and identify in the output: the resolved IP, the TCP connect, the TLS handshake and the HTTP status code.
8. **Lab:** Run `dig www.google.com` and `traceroute 8.8.8.8`. Note the number of hops.

★ Quick Revision

- OSI 7 layers: Physical (bits), Data Link (frames, MAC, switch), Network (packets, IP, router), Transport (segments, TCP/UDP, ports), Session, Presentation (TLS, encoding), Application (HTTP, DNS, SSH).
- TCP/IP 4 layers: Network Access, Internet, Transport, Application.
- Encapsulation adds headers going down; decapsulation removes them going up.
- TCP = reliable, 3-way handshake; UDP = fast, connectionless.
- Opening a website: DNS → TCP handshake → TLS → HTTP request → server/app/DB → response → render.

Chapter 3: Linux Basic Commands

Mitranno, from this chapter onwards your terminal is your best friend. Kalji karu naka if the black screen looks scary at first — within a week of daily practice you will be faster on the command line than with a mouse. Let's first understand why Linux matters so much in the cloud, and then we will type commands together.

3.1 Why Linux?

More than 90% of cloud servers run Linux. It is free, stable, secure and fully controllable from the command line, which makes automation easy. On AWS, Amazon Linux, Ubuntu, RHEL/CentOS, Debian and SUSE are all available. There is usually **no graphical desktop** on a server — you manage it through a **shell** (usually `bash`) over SSH.

Linux file-system hierarchy

```
/                 root of everything
├── bin, sbin     essential commands (now links to /usr/bin, /usr/sbin)
├── boot          kernel and boot loader
├── dev           device files (/dev/nvme0n1, /dev/xvda)
├── etc           configuration files (nginx, ssh, fstab, passwd)
├── home          users' home folders (/home/ec2-user, /home/ubuntu)
├── root          home of the root user
├── opt           optional / third-party software
├── tmp           temporary files (cleared on reboot)
├── usr           programs, libraries, docs (/usr/share/nginx/html)
├── var           variable data: logs (/var/log), web (/var/www), databases (/var/lib/mysql)
├── proc, sys     virtual files with kernel & hardware info
└── mnt, media    mount points for extra disks
```

> **i Everything is a file**
>
> In Linux, disks (`/dev/nvme1n1`), processes (`/proc/1234`) and even hardware info are represented as files. Paths are **case-sensitive**: `Index.html` and `index.html` are different files.

3.2 Navigation

Chala, aata try karu. Log in to any Linux machine (or your EC2 instance) and type these commands with me.

Command	Meaning	Example
`pwd`	Print working (current) directory	`pwd` → `/home/ubuntu`
`ls`	List files	`ls` , `ls -l` , `ls -la` , `ls -lh /var/log`
`cd`	Change directory	`cd /etc` , `cd ..` , `cd ~` , `cd -`
`tree`	Show folder tree (install package `tree`)	`tree -L 2 /etc/nginx`

Command	Meaning	Example
`clear`	Clear the screen (or press Ctrl+L)	
`history`	Previous commands	`history | tail -20`, then `!45` to re-run no. 45

```
pwd                 # where am I?
ls -la              # long listing including hidden files (starting with .)
cd /var/log         # absolute path (starts with /)
cd ../www           # relative path (.. = parent, . = current)
cd ~                # go to home directory (same as just 'cd')
cd -                # go back to the previous directory
```

Understanding `ls -l` output:

```
-rw-r--r--  1 ec2-user ec2-user  1024 Mar  3 10:15 notes.txt
│└─┬─┘└─┬┘  │ └──┬──┘ └──┬──┘  └─┬─┘ └─────┬─────┘ └──┬──┘
│ permissions links owner  group   size  modified time   name
└─ type: - file, d directory, l symbolic link
```

✔ Keyboard shortcuts that save hours

Tab auto-completes file and command names (press twice to see options). **↑ / ↓** scroll through history. **Ctrl+C** stops a running command. **Ctrl+R** searches history. **Ctrl+A / Ctrl+E** jump to start / end of line.

3.3 Working with files and directories

Command	Purpose	Example
`mkdir`	Make directory	`mkdir project`, `mkdir -p a/b/c` (create parents)
`touch`	Create empty file / update timestamp	`touch index.html`
`cp`	Copy	`cp a.txt b.txt`, `cp -r site/ backup/`
`mv`	Move or rename	`mv old.txt new.txt`, `mv file.txt /tmp/`
`rm`	Remove file	`rm file.txt`, `rm -i *.log` (ask first)
`rm -r`	Remove directory recursively	`rm -r olddir`
`rmdir`	Remove empty directory	`rmdir emptydir`
`ln -s`	Create symbolic link (shortcut)	`ln -s /etc/nginx/sites-available/app /etc/nginx/sites-enabled/`
`file`	Show file type	`file /bin/ls`
`stat`	Detailed file info	`stat index.html`

```
mkdir -p ~/labs/site/css
cd ~/labs/site
touch index.html css/style.css
cp index.html index.bak
mv index.bak old-index.html
```

```
ls -R ~/labs
rm old-index.html
```

✖ rm is permanent

There is **no Recycle Bin** on a Linux server. `rm -rf /some/path` deletes instantly and forever. Never run `rm -rf` with a variable that might be empty, and double-check the path before pressing Enter. Never run `sudo rm -rf /`.

3.4 Viewing files

Command	Use
`cat file`	Print entire file
`less file`	Scroll page by page (`Space` next, `b` back, `/word` search, `q` quit)
`more file`	Older, simpler pager
`head -n 20 file`	First 20 lines
`tail -n 20 file`	Last 20 lines
`tail -f /var/log/nginx/access.log`	**Follow** a log live (Ctrl+C to stop)
`wc -l file`	Count lines (`-w` words, `-c` bytes)
`diff a b`	Show differences between two files

```
cat /etc/os-release          # which Linux distribution and version?
head -5 /etc/passwd
sudo tail -f /var/log/messages      # Amazon Linux / CentOS system log
sudo tail -f /var/log/syslog        # Ubuntu system log
```

3.5 Permissions

This is the topic where most "403 Forbidden" and "Permission denied" problems come from, so dhyan do. Once you understand r, w and x properly, half of your troubleshooting is already done.

Every file has an **owner (u)**, a **group (g)** and **others (o)**. Each can have **read (r)**, **write (w)** and **execute (x)** permission.

Permission	On a file	On a directory	Value
r (read)	View contents	List files (`ls`)	4
w (write)	Modify contents	Create/delete files inside	2
x (execute)	Run as program/script	Enter the directory (`cd`)	1

Numeric (octal) notation — add the values for each of user, group, others:

Octal	Symbolic	Meaning	Typical use
777	`rwxrwxrwx`	Everyone everything	**Avoid** — insecure

Octal	Symbolic	Meaning	Typical use
755	`rwxr-xr-x`	Owner full, others read+enter	Directories, scripts, web folders
750	`rwxr-x---`	Owner full, group read, others nothing	Private app directories
700	`rwx------`	Only owner	`~/.ssh` directory
644	`rw-r--r--`	Owner write, others read	Web files (HTML, CSS), configs
640	`rw-r-----`	Owner write, group read	Config with passwords (`wp-config.php`)
600	`rw-------`	Only owner read/write	`~/.ssh/authorized_keys`
400	`r--------`	Only owner read	`.pem` **private key**

```
chmod 755 deploy.sh           # numeric
chmod u+x deploy.sh           # symbolic: add execute for user
chmod go-w file.txt           # remove write from group and others
chmod -R 755 /var/www/html    # recursive
chmod 400 mykey.pem           # required before using an SSH key

sudo chown nginx:nginx /usr/share/nginx/html/index.html        # owner:group
sudo chown -R www-data:www-data /var/www/html                   # Ubuntu web user
sudo chown -R apache:apache /var/www/html                       # AL2023/CentOS Apache user
```

✔ Safe web permissions

A common, safe pattern: directories `755`, files `644`, owned by your login user or the web-server user. Set them in one go: `sudo find /var/www/html -type d -exec chmod 755 {} \;` and `sudo find /var/www/html -type f -exec chmod 644 {} \;`

Ravindra's Tip

Before changing permissions on a live server, always run `ls -l` first and note the original values. And please, never use `chmod -R 777` to "fix" an error — it only hides the real problem and opens a security hole. Find which user needs access, then give exactly that access. Bas itna hi.

3.6 Users, groups and sudo

Command	Purpose
`whoami`	Current user name
`id`	UID, GID and groups of the current user
`sudo useradd -m -s /bin/bash ravi`	Create user with home dir and bash shell (`adduser ravi` on Ubuntu is interactive)
`sudo passwd ravi`	Set password
`sudo usermod -aG wheel ravi`	Add to admin group (**wheel** on AL2023/CentOS, **sudo** on Ubuntu)
`sudo groupadd devops`	Create group
`sudo usermod -aG devops ravi`	Add user to extra group (`-a` = append — never forget it!)
`groups ravi`	Show user's groups

Command	Purpose
`sudo userdel -r ravi`	Delete user and home folder
`su - ravi`	Switch to another user
`sudo -i`	Become root (use carefully, type `exit` to return)

User information lives in `/etc/passwd` (accounts), `/etc/shadow` (hashed passwords) and `/etc/group` (groups). Sudo rules live in `/etc/sudoers` — always edit with `sudo visudo`.

> **i Why sudo instead of logging in as root?**
>
> `sudo` gives admin power only for one command, logs who did what (`/var/log/secure` or `/var/log/auth.log`), and reduces accidents. On EC2, the default user (`ec2-user`/`ubuntu`) has password-less sudo, and direct root login over SSH is disabled.

3.7 Package managers

Think of a package manager like the Play Store on your phone — you ask for an app, it downloads it from a trusted store along with everything it needs. But technically, it is fetching signed packages from configured repositories and resolving dependencies. Bagha, the commands differ slightly between distributions:

A **package manager** downloads software from trusted **repositories**, installs dependencies automatically, and handles updates.

Task	Ubuntu / Debian (`apt`)	Amazon Linux 2023, Amazon Linux 2, CentOS Stream 9 (`yum`)
Refresh package list	`sudo apt update`	(automatic) `sudo yum makecache`
Upgrade all	`sudo apt upgrade -y`	`sudo yum update -y`
Install	`sudo apt install -y nginx`	`sudo yum install -y nginx`*
Remove	`sudo apt remove nginx` (`purge` removes config too)	`sudo yum remove nginx`
Search	`apt search php`	`yum search php`
Info	`apt show nginx`	`yum info nginx`
List installed	`apt list --installed`	`yum list installed`
Which package gives a file	`dpkg -S /usr/sbin/nginx`	`yum provides /usr/sbin/nginx`
Clean cache	`sudo apt autoremove && sudo apt clean`	`sudo yum clean all`
Package file format	`.deb` (`dpkg -i`)	`.rpm` (`rpm -i`)

* On Amazon Linux 2, some packages come from `amazon-linux-extras` (e.g. `sudo amazon-linux-extras install nginx1`). Amazon Linux 2023 has **no** `amazon-linux-extras`; everything is in the regular repositories.

i yum or dnf? Kalji karu naka

In this book I use `yum` on Amazon Linux and CentOS because it works everywhere in the Red Hat family. On newer systems (Amazon Linux 2023, CentOS Stream 9, RHEL 9) `yum` is simply an alias that actually runs `dnf`, the newer package manager — so when you see `dnf install ...` in online guides, it is the same thing. Same options, same repositories.

✔ CentOS Stream: enable EPEL for extra packages

Some tools (e.g. `htop`, `certbot`) come from the **EPEL** repository: `sudo yum install -y epel-release` on CentOS Stream 9.

3.8 Processes

A **process** is a running program. Each has a **PID** (process ID).

Command	Purpose
`ps aux`	All processes with user, CPU, memory
`ps aux | grep nginx`	Find a specific process
`pgrep -a node`	PIDs and command lines matching a name
`top`	Live view (press `q` to quit, `M` sort by memory, `P` by CPU)
`kill PID`	Politely ask process to stop (SIGTERM, 15)
`kill -9 PID`	Force kill (SIGKILL) — last resort
`pkill -f app.py`	Kill by name/pattern
`command &`	Run in background
`jobs`, `fg`, `bg`	Manage background jobs of the current shell
`nohup command &`	Keep running after you log out
`uptime`	Load average and how long the system has run
`free -h`	RAM and swap usage

3.9 Getting help

```
man ls              # full manual page (q to quit, /word to search)
ls --help           # short usage summary
whatis chmod        # one-line description
apropos network     # search manual pages by keyword
type cd             # is it a built-in, alias or program?
which nginx         # full path of a command
```

RB Ravindra's Tip

Before editing any important config file, make a backup: `sudo cp /etc/nginx/nginx.conf /etc/nginx/nginx.conf.bak`. If something breaks, you can restore in one command. This small habit can save your lab server again and again.

3.10 Text editors: nano and vim

nano — easiest for beginners

```
sudo nano /etc/nginx/nginx.conf
```

Shortcuts are shown at the bottom (`^` means Ctrl): **Ctrl+O** then Enter = save, **Ctrl+X** = exit, **Ctrl+W** = search, **Ctrl+K** = cut line, **Ctrl+U** = paste.

vim — available everywhere

Vim has **modes**: Normal (for commands, the default), Insert (for typing) and Command-line (after `:`).

Keys	Action
`i`	Enter insert mode (start typing)
`Esc`	Back to normal mode
`:w`	Save
`:q`	Quit (`:q!` quit without saving)
`:wq` or `ZZ`	Save and quit
`dd` / `yy` / `p`	Delete (cut) line / copy line / paste
`u` / `Ctrl+R`	Undo / redo
`/text` then `n`	Search forward, next match
`:set number`	Show line numbers
`gg` / `G`	Go to first / last line
`:%s/old/new/g`	Replace all

✔ Stuck in vim?

Press `Esc` two times, then type `:q!` and press Enter. You are out without saving.

Common Mistakes I See Students Make

- **Typing `cd /Home` or `Index.html`.** Linux paths are case-sensitive.
- **Using `chmod 777` to fix every error.** It is insecure and usually hides a wrong owner or SELinux label.

- `usermod -G` **without** `-a`**.** This replaces all the user's groups instead of adding one — the user may lose sudo access.
- **Running everything as root with** `sudo -i` and then creating files that the normal user or web server cannot edit.
- **Using** `apt` **on Amazon Linux or** `yum` **on Ubuntu.** First check `cat /etc/os-release`.
- **Getting stuck in vim.** Remember: `Esc` then `:q!`.
- `rm -rf` **with a wrong path.** There is no recycle bin — read the command twice before pressing Enter.

Interview Questions

Q1. What do the permissions 755 and 644 mean?

755 = owner rwx, group r-x, others r-x (typical for directories/scripts). 644 = owner rw-, group r--, others r-- (typical for files).

Q2. What is the difference between `apt` and `yum`?

Both are package managers. `apt` is used on Debian/Ubuntu with .deb packages; `yum` is used on RHEL-family systems like Amazon Linux and CentOS with .rpm packages (on newer releases `yum` runs its successor, `dnf`).

Q3. What is the difference between `su` and `sudo`?

`su` switches to another user (often root) and needs that user's password. `sudo` runs a single command with elevated privileges based on rules in `/etc/sudoers`, and logs it.

Q4. How do you find which process is using the most memory?

Use `top` and press `M`, or `ps aux --sort=-%mem | head`.

Q5. What is the difference between `kill` and `kill -9`?

`kill` sends SIGTERM (15), allowing the process to clean up. `kill -9` sends SIGKILL, which ends it immediately without cleanup — use it only as a last resort.

Q6. Where are user accounts and passwords stored?

Accounts in `/etc/passwd`, hashed passwords in `/etc/shadow`, groups in `/etc/group`.

Practice Questions and Lab Exercises

1. Explain the purpose of `/etc`, `/var/log`, `/home` and `/usr/share`.
2. What is the difference between absolute and relative paths? Give two examples.
3. Convert to numeric: `rwxr-x---`, `rw-r--r--`, `r--------`. Convert `750` and `640` to symbolic.
4. Why must a `.pem` key have permission `400` or `600`?
5. Write the commands to install `git` and `tree` on Ubuntu and on Amazon Linux 2023.
6. **Lab:** Create `~/labs/demo/{html,css,js}`, create `index.html` inside `html`, copy it to `/tmp`, rename it, then delete it.
7. **Lab:** Create user `student1`, add it to the admin group of your distribution, switch to it and run `sudo whoami`.
8. **Lab:** Run `sleep 300 &`, find its PID with `pgrep`, and kill it.

9. **Lab:** Open a file in vim, add three lines, save and quit. Then do the same in nano.

★ Quick Revision

- Navigate: `pwd`, `ls -la`, `cd`. Files: `mkdir -p`, `touch`, `cp -r`, `mv`, `rm -r` (permanent!).
- View: `cat`, `less`, `head`, `tail -f`.
- Permissions r=4, w=2, x=1 → `chmod 755` dirs, `644` files, `400` pem. Ownership `chown user:group`.
- Users: `useradd -m`, `passwd`, `usermod -aG`. Admin group: `wheel` (RHEL/AL) vs `sudo` (Ubuntu).
- Packages: `apt` (Ubuntu), `yum` (Amazon Linux / CentOS; runs `dnf` on newer releases).
- Processes: `ps aux`, `top`, `kill`. Help: `man`, `--help`. Editors: nano (easy), vim (`i`, `Esc`, `:wq`).

Chapter 4: Linux Advanced Commands

Mitranno, basic commands let you move around. Advanced commands let you **search logs, automate tasks, manage services, inspect networks and disks** — the daily work of a cloud engineer. In real jobs, nobody opens a 2 GB log file in nano; we use `grep`, `awk` and pipes. Chala, let's learn the tools that make you look like a pro.

4.1 Searching text: grep

`grep` prints lines that match a pattern.

Option	Meaning
`-i`	Ignore case
`-r` / `-R`	Recursive search in directories
`-n`	Show line numbers
`-v`	Invert (lines that do NOT match)
`-c`	Count matching lines
`-l`	Only file names
`-w`	Match whole word
`-E`	Extended regex (alternation with a pipe symbol, `+`, `?`)
`-A 3 / -B 3 / -C 3`	Show 3 lines after / before / around

```
grep -i "error" /var/log/nginx/error.log
grep -rn "listen" /etc/nginx/
grep -v "^#" /etc/ssh/sshd_config | grep -v "^$"   # hide comments & blank lines
grep -c " 404 " /var/log/nginx/access.log          # how many 404s?
grep -E "Failed|Invalid" /var/log/auth.log         # Ubuntu SSH failures
sudo grep "Failed password" /var/log/secure         # AL2/CentOS (AL2023 uses journalctl)
```

4.2 Finding files: find

```
find /var/www -name "*.html"                 # by name
find / -iname "nginx.conf" 2>/dev/null       # case-insensitive, hide permission errors
find /var/log -type f -size +100M            # files bigger than 100 MB
find /tmp -type f -mtime +7                  # modified more than 7 days ago
find /tmp -type f -mtime +7 -delete          # ...and delete them
find /var/www/html -type d -exec chmod 755 {} \;   # run a command on each result
find . -type f -name "*.log" | xargs ls -lh
```

4.3 Text processing: awk and sed

awk — column-based processing

`awk` splits each line into fields `$1`, `$2`, ... (space-separated by default; `-F` sets the separator).

```
awk '{print $1}' /var/log/nginx/access.log | sort | uniq -c | sort -rn | head
#  ^ top 10 client IPs hitting your web server

awk -F: '{print $1, $7}' /etc/passwd          # user name and shell
awk -F: '$3 >= 1000 {print $1}' /etc/passwd   # normal (non-system) users
df -h | awk 'NR>1 {print $5, $6}'             # usage % and mount point
awk '{sum += $10} END {print sum/1024/1024 " MB"}' /var/log/nginx/access.log  # bytes served
```

sed — stream editor (find & replace)

```
sed 's/http/https/' file.txt              # replace first match per line (prints result)
sed 's/http/https/g' file.txt             # replace all matches
sed -i 's/Listen 80/Listen 8080/' /etc/httpd/conf/httpd.conf   # edit file in place
sed -i.bak 's/old/new/g' config.ini       # in place, keep backup config.ini.bak
sed -n '10,20p' file.txt                  # print only lines 10-20
sed '/^#/d' file.txt                      # delete comment lines
```

> ⚠ **Test sed before using -i**
>
> Run the `sed` command **without** `-i` first and check the output. Once you are happy, add `-i` (or `-i.bak` to keep a backup).

4.4 Pipes and redirection

Pipes are like a relay race — the output of one runner (command) is handed to the next. Technically, the shell connects the standard output (stdout) of the left command to the standard input (stdin) of the right command. Samjla ka? Once you master this, you can combine small commands into powerful one-liners.

Symbol	Meaning	Example
`|` (pipe)	Output of left command becomes input of right command	`ps aux | grep nginx`
`>`	Redirect output to file (overwrite)	`echo "hi" > a.txt`
`>>`	Append output to file	`date >> log.txt`
`<`	Take input from file	`mysql -u root -p mydb < backup.sql`
`2>`	Redirect errors	`find / -name x 2> errors.txt`
`2>&1`	Send errors to same place as output	`./script.sh > out.log 2>&1`
`&>`	Output and errors together (bash)	`cmd &> all.log`
`/dev/null`	"Black hole" — discard	`cmd > /dev/null 2>&1`
`tee`	Write to file **and** screen	`echo "x" | sudo tee /etc/file`

Symbol	Meaning	Example
`&&` / `||`	Run next only if success / failure	`sudo nginx -t && sudo service nginx reload`

i Why `sudo tee` instead of `sudo echo > file`?

In `sudo echo "text" > /etc/file`, the redirection `>` is done by **your** shell (not root), so it fails with Permission denied. Pipe the text into `sudo tee /etc/file` instead (or `sudo tee -a` to append).

Here-documents (used a lot in this book)

A **here-doc** writes multiple lines into a file in one command — perfect for creating config files by copy-paste. Everything between `<<'EOF'` and the line containing only `EOF` is written to the file:

```
sudo tee /tmp/hello.txt > /dev/null <<'EOF'
Line one
Line two with $HOME not expanded because 'EOF' is quoted
EOF
cat /tmp/hello.txt
```

✔ Copy-pasting here-docs

Paste the whole block at once, including the final `EOF` line. The closing `EOF` must be at the very start of the line with nothing after it.

RB Ravindra's Tip

When a website is down, my first three commands are always: `sudo service <name> status`, `sudo tail -n 50 <error log>` and `sudo ss -tlnp`. Nine times out of ten the answer is right there. Make this your reflex too.

4.5 Archiving and compression: tar, gzip, zip

```
tar -czvf site-backup.tar.gz /var/www/html      # create gzip archive (c=create z=gzip v=verbose
f=file)
tar -tzvf site-backup.tar.gz                    # list contents
mkdir -p /tmp/restore
tar -xzvf site-backup.tar.gz -C /tmp/restore    # extract into a folder
tar -cJvf logs.tar.xz /var/log/nginx            # xz compression (smaller, slower)
gzip big.log            # -> big.log.gz        gunzip big.log.gz
zip -r site.zip mysite/                         # install package 'zip' if missing
unzip site.zip -d /var/www/html/                # install package 'unzip' if missing
```

4.6 Services: service, systemctl and journalctl

In my sessions, students often ask: "Sir, I installed Nginx, why is the website not opening after reboot?" Usually they started the service but never enabled it. Bagha the difference below.

Modern Linux uses **systemd** to start and manage background services (daemons) such as `nginx`, `sshd`, `mariadb`. The real tool that talks to systemd is **`systemctl`**. In class I teach the simpler, older **`service`** command first, because it reads like plain English — service nginx start — and it works the same on Ubuntu, Amazon Linux and CentOS. Behind the scenes, on these systems `service` simply calls `systemctl` for you.

Day-to-day control with service

Command	Purpose	What runs underneath
`sudo service nginx start`	Start now	`systemctl start nginx`
`sudo service nginx stop`	Stop now	`systemctl stop nginx`
`sudo service nginx restart`	Stop + start	`systemctl restart nginx`
`sudo service nginx reload`	Re-read config without dropping connections	`systemctl reload nginx`
`sudo service nginx status`	Is it running? Last log lines (press `q` to exit)	`systemctl status nginx`
`service --status-all`	List services and their state (most useful on Ubuntu)	—

Things only systemctl can do

Command	Purpose
`sudo systemctl enable nginx`	Start automatically at every boot
`sudo systemctl disable nginx`	Do not start at boot
`systemctl is-active nginx` / `systemctl is-enabled nginx`	Quick yes/no checks (handy in scripts)
`systemctl list-units --type=service --state=running`	All running services
`sudo systemctl daemon-reload`	Reload systemd after creating or editing a `.service` unit file

Ravindra's Tip: why I still write systemctl enable

Mitranno, dhyan do — `service` can start, stop, restart, reload and show status, but it **cannot** make a service start automatically after a reboot. That job belongs to systemd itself. So throughout this book you will see this pair: `sudo service nginx start` (start it now) and `sudo systemctl enable nginx` (start it at every boot). Similarly, `systemctl daemon-reload` (after writing your own unit files) and `journalctl` (for logs) have no `service` equivalent, so we use them directly. Samjla ka?

✔ service: command not found?

`service` is present by default on Ubuntu, Amazon Linux 2023 and CentOS Stream 9 AMIs. If a very minimal image lacks it, install it: `sudo yum install -y initscripts` on Amazon Linux 2023, or `sudo yum install -y initscripts-service` on CentOS Stream 9.

Logs with journalctl

systemd stores service logs in its journal; read them with `journalctl` (there is no `service` equivalent):

```
journalctl -u nginx                # all logs of the nginx service
journalctl -u nginx -f             # follow live (Ctrl+C to stop)
journalctl -u nginx --since "1 hour ago"
journalctl -p err -b               # only errors since last boot
journalctl -u sshd -n 30 --no-pager # last 30 lines (service is 'ssh' on Ubuntu)
sudo journalctl --disk-usage
```

> **i Logs on Amazon Linux 2023**
>
> Amazon Linux 2023 does **not** install `rsyslog` by default, so `/var/log/messages` and `/var/log/secure` may not exist. Use `journalctl` instead (or install `rsyslog`).

4.7 Networking commands

Command	Purpose	Example
`ip addr` (`ip a`)	Interfaces and IP addresses	`ip -4 a`
`ip route`	Routing table / default gateway	`ip r`
`ss -tulnp`	Listening TCP/UDP ports with process	`sudo ss -tlnp`
`netstat -tulnp`	Older equivalent of `ss` (package `net-tools`)	`sudo netstat -tlnp`
`ping`	Reachability (ICMP)	`ping -c 4 google.com`
`curl`	HTTP client — test websites/APIs	`curl -I http://localhost`
`wget`	Download files	`wget https://wordpress.org/latest.tar.gz`
`dig` / `nslookup`	DNS lookup (package `bind-utils` or `dnsutils`)	`dig +short example.com`
`traceroute` / `tracepath`	Route to a host	`tracepath google.com`
`nc` (netcat)	Test if a TCP port is open	`nc -zv 10.0.1.25 3306`
`hostnamectl`	Show / set hostname	`sudo hostnamectl set-hostname web01`

```
sudo ss -tlnp                      # which programs listen on which ports?
sudo ss -tlnp | grep ':80 '        # is anything on port 80?
curl -I http://localhost           # only response headers (HTTP status)
curl -s http://localhost:3000/api/health
curl -o page.html https://example.com
wget -q -O - https://checkip.amazonaws.com    # print public IP
ping -c 3 8.8.8.8
```

✔ Reading `ss -tlnp` output

`0.0.0.0:80` or `*:80` → listening on all interfaces (reachable from outside if the security group allows). `127.0.0.1:3000` → only reachable from the server itself (good for apps behind a reverse proxy).

i ping to EC2 fails?

Security groups block ICMP by default. A failed `ping` does not mean the server is down — test with `curl` or `nc` on the actual port, or allow All ICMP – IPv4 from your IP for testing.

Installing missing network tools

```
# Ubuntu
sudo apt install -y net-tools dnsutils traceroute netcat-openbsd
# Amazon Linux 2023 / CentOS Stream 9
sudo yum install -y net-tools bind-utils traceroute nmap-ncat
```

4.8 Disk and storage commands

```
df -h            # disk space per mounted filesystem (human readable)
df -Th           # include filesystem type (xfs, ext4)
sudo du -sh /var/log                    # total size of a folder
sudo du -h --max-depth=1 /var | sort -h # which sub-folder is big?
lsblk            # block devices (disks & partitions) as a tree
lsblk -f         # with filesystem type, UUID and mount point
sudo blkid       # UUIDs of all filesystems
findmnt /        # what is mounted at /
sudo mount /dev/nvme1n1 /data
sudo umount /data
cat /etc/fstab   # permanent mounts (read at boot)
free -h          # memory
```

`/etc/fstab` **format** (covered in depth in Chapter 13):

```
# <device>                                <mount point> <type> <options>        <dump> <fsck>
UUID=3f1c2b9a-6d1e-4a5f-9c1e-2b7d8a9e0f11   /data         xfs    defaults,nofail  0      2
```

4.9 Scheduling tasks: cron

`cron` runs commands automatically on a schedule. Edit your table with `crontab -e`, list with `crontab -l`.

```
┌──────── minute (0-59)
│ ┌────── hour (0-23)
│ │ ┌──── day of month (1-31)
│ │ │ ┌── month (1-12)
│ │ │ │ ┌ day of week (0-7, 0 and 7 = Sunday)
│ │ │ │ │
* * * * *  command
```

Schedule	Meaning
`*/5 * * * *`	Every 5 minutes
`0 2 * * *`	Every day at 02:00
`30 9 * * 1-5`	09:30 Monday to Friday
`0 0 1 * *`	Midnight on the 1st of every month
`@reboot`	Once at every boot

Now open your terminal and try this with me. Example — back up the website every night at 1:30 AM. Run `crontab -e` and add this line:

```
30 1 * * * tar -czf $HOME/backups/site-$(date +\%F).tar.gz /var/www/html
```

(`%` has a special meaning in crontab, so it is written as `\%`. Create the folder first with `mkdir -p ~/backups`.)

i cron on Amazon Linux 2023

AL2023 does not ship cron by default. Install it: `sudo yum install -y cronie`, then `sudo service crond start` and `sudo systemctl enable crond`. (systemd timers are the modern alternative.) The server clock is UTC by default — set India time with `sudo timedatectl set-timezone Asia/Kolkata`.

RB Ravindra's Tip

Before scheduling any cron job, run the exact command manually first and check it works. Cron runs with a very small `PATH` and no interactive shell, so always use full paths (`/usr/bin/tar`, `/home/ubuntu/backup.sh`) and redirect output to a log file: `>> /home/ubuntu/backup.log 2>&1`. Lakshat theva!

4.10 Environment variables

```
echo $HOME $USER $PATH $SHELL
env | sort                      # all environment variables
export APP_ENV=production       # set for this session and child processes
echo $APP_ENV
unset APP_ENV
echo 'export PATH=$PATH:$HOME/bin' >> ~/.bashrc   # make permanent for your user
source ~/.bashrc                # reload without logging out
```

✓ Keep secrets out of code

Store database passwords in environment variables or a `.env` file with permission `600`, never hard-coded in files that go to GitHub.

4.11 Remote access: ssh and scp

```
ssh -i ~/keys/mykey.pem ec2-user@13.233.10.25              # log in
ssh -i ~/keys/mykey.pem ubuntu@13.233.10.25 'uptime'        # run one command remotely
scp -i ~/keys/mykey.pem index.html ec2-user@13.233.10.25:/home/ec2-user/        # upload file
scp -i ~/keys/mykey.pem -r ./site ubuntu@13.233.10.25:~/                        # upload folder
scp -i ~/keys/mykey.pem ec2-user@13.233.10.25:/var/log/nginx/error.log .        # download
rsync -avz -e "ssh -i ~/keys/mykey.pem" ./site/ ubuntu@13.233.10.25:~/site/     # smart sync
```

Save typing with an SSH config file `~/.ssh/config`:

```
Host web1
    HostName 13.233.10.25
    User ec2-user
    IdentityFile ~/keys/mykey.pem
```

Now just type `ssh web1` or `scp file.txt web1:~/`.

4.12 Monitoring: top and htop

`top` is always available. `htop` is friendlier (colour, mouse, F-keys): `sudo apt install -y htop` (Ubuntu), `sudo yum install -y htop` (AL2023; on CentOS Stream enable EPEL first).

In top	Action
`P` / `M`	Sort by CPU / memory
`k`	Kill a process (enter PID)
`1`	Show each CPU core
`q`	Quit

Key numbers: **load average** (should be below the number of vCPUs), **%CPU**, **%MEM**, **RES** memory, and **wa** (I/O wait — high means disk is slow).

4.13 Basic shell scripting

Ekdum simple aahe — a script is just the commands you already know, saved in a file. Let's write one together.

A shell script is a text file containing commands, run top to bottom. It is the first step towards automation (and towards EC2 **user data**, which runs a script at first boot).

```
#!/bin/bash
# file: sysinfo.sh  - print a quick health report
set -uo pipefail         # stop on undefined variables and pipe failures

NAME=$(hostname)
echo "Health report for $NAME at $(date)"
echo "---------------------------------"
echo "Uptime : $(uptime -p)"
echo "Disk / : $(df -h / | awk 'NR==2 {print $5 " used"}')"
echo "Memory : $(free -m | awk '/Mem/ {printf "%d/%d MB used", $3, $2}')"
```

```
# if-else
if systemctl is-active --quiet nginx; then
  echo "Nginx  : running"
else
  echo "Nginx  : NOT running"
fi

# loop
for svc in sshd crond; do
  echo "$svc -> $(systemctl is-active $svc 2>/dev/null)"
done

# function with argument
check_port() {
  if ss -tln | grep -q ":$1 "; then echo "Port $1 open"; else echo "Port $1 closed"; fi
}
check_port 22
check_port 80
```

```
nano sysinfo.sh        # paste the script, save with Ctrl+O, exit with Ctrl+X
chmod +x sysinfo.sh
./sysinfo.sh
```

Concept	Syntax
Variable	`NAME="Asha"`, use `$NAME` or `${NAME}` (no spaces around `=`)
Command output	`TODAY=$(date +%F)`
Arguments	`$1`, `$2`, all: `$@`, count: `$#`
Exit status of last command	`$?` (0 = success)
Test file exists	`if [ -f /etc/nginx/nginx.conf ]; then ... fi`
Test directory	`[ -d /var/www ]`
Compare numbers	`[ "$a" -gt 10 ]` (`-eq -ne -lt -le -ge`)
Compare strings	`[ "$a" = "yes" ]`
Read input	`read -p "Enter name: " NAME`

Common Mistakes I See Students Make

- **Using `sed -i` without testing first** and corrupting a config file. Test without `-i`, or use `-i.bak`.
- **`sudo echo "x" > /etc/file`** — fails with Permission denied. Use `echo "x" | sudo tee /etc/file`.
- **Starting a service but not enabling it**, so it is gone after reboot. Use `sudo service <name> start` **and** `sudo systemctl enable <name>`.
- **Forgetting that the app is listening on 127.0.0.1** and wondering why it is not reachable from the browser — that is actually correct behind a reverse proxy.
- **Editing `/etc/fstab` and rebooting without `sudo mount -a`.** One typo can stop the instance from booting.
- **Cron jobs using relative paths or `%` without escaping.**

- **Scripts without** `chmod +x` or with Windows line endings (CRLF) — fix with `sed -i 's/\r$//' script.sh`.

Interview Questions

Q1. How do you find the top 10 IP addresses in an Nginx access log?

`awk '{print $1}' access.log | sort | uniq -c | sort -rn | head`

Q2. What is the difference between `>` and `>>`? What does `2>&1` do?

`>` overwrites a file, `>>` appends. `2>&1` redirects standard error to wherever standard output is going.

Q3. How do you check which ports a Linux server is listening on?

`sudo ss -tlnp` (or the older `netstat -tlnp`).

Q4. What is the difference between `service <name> restart` and `service <name> reload`?

`restart` stops and starts the service (brief downtime). `reload` asks the service to re-read its configuration without dropping existing connections (if supported).

Q5. What is the difference between `service` and `systemctl`?

`systemctl` is the native systemd tool. `service` is an older wrapper that, on systemd systems, forwards start/stop/restart/reload/status to `systemctl`. Enabling at boot (`systemctl enable`) and `daemon-reload` require `systemctl`.

Q6. Explain the five fields of a cron expression.

Minute, hour, day of month, month, day of week — e.g. `30 2 * * 1` runs at 02:30 every Monday.

Q7. How do you see the logs of a systemd service?

`journalctl -u <service>`; add `-f` to follow live, `-n 50` for the last 50 lines, `--since "1 hour ago"` to filter by time.

Q8. What is the difference between `df` and `du`?

`df` shows free/used space per mounted filesystem; `du` shows how much space specific files or directories use.

Practice Questions and Lab Exercises

1. Write a command to count how many times `404` appears in an Nginx access log.
2. Find all `.conf` files under `/etc` that were modified in the last 2 days.
3. Using `awk`, print the 1st and 9th columns (IP and status code) of an access log.
4. Explain `>`, `>>`, `2>&1` and the pipe symbol with examples.
5. What is the difference between `service nginx restart` and `service nginx reload`? Between `service nginx start` and `systemctl enable nginx`?
6. Which command shows the ports on which a server is listening? How do you know whether an app is exposed to the internet?
7. Write a cron entry that runs `/home/ubuntu/backup.sh` every Sunday at 11:00 PM.
8. **Lab:** Write a script `backup.sh` that creates `~/backups/web-<date>.tar.gz` of `/var/www/html` and deletes backups older than 7 days.

9. **Lab:** Use `scp` to upload a folder to your EC2 instance and `rsync` to sync only changed files.

★ Quick Revision

- `grep` (search text), `find` (search files), `awk` (columns), `sed` (replace).
- Pipes, redirection `> >> 2> 2>&1`, `tee` for writing root-owned files, here-docs for multi-line files.
- `tar -czvf` create, `tar -xzvf` extract.
- `sudo service <name> start|stop|restart|reload|status`; boot-time with `sudo systemctl enable <name>`; logs with `journalctl -u <name> -f`.
- Network: `ip a`, `ss -tlnp`, `curl -I`, `dig`, `nc -zv`. Disk: `df -h`, `du -sh`, `lsblk -f`, `blkid`, `/etc/fstab`.
- `crontab -e` (5 fields). `export VAR=value`, `~/.bashrc`. `ssh -i`, `scp -r`, `rsync -avz`.
- Scripts: `#!/bin/bash`, `chmod +x`, variables, `if`, `for`, functions.

Chapter 5: Amazon EC2 (Elastic Compute Cloud)

5.1 Why EC2? Cloud vs on-premises

Mitranno, let's first understand why EC2 exists at all. Imagine your college wants to host a result portal. The traditional **on-premises** way: buy servers (₹2–5 lakh each), a UPS, air-conditioning, network switches, a server room, hire an admin, wait weeks for delivery, and still the site crashes on result day when lakhs of students log in together.

With **cloud computing**, you rent compute power over the internet and pay only for what you use.

Point	On-premises	AWS EC2 (cloud)
Upfront cost	High capital expense (CapEx)	Zero upfront; pay-as-you-go (OpEx)
Time to get a server	Weeks	Minutes
Scaling	Buy more hardware; slow	Launch more / bigger instances in minutes; Auto Scaling
Maintenance	You handle hardware, power, cooling	AWS handles physical infrastructure
Availability	One building; single point of failure	Multiple Availability Zones and Regions
Global reach	Hard	Deploy in any AWS Region worldwide
Security	Fully your responsibility	**Shared responsibility**: AWS secures the cloud, you secure what is in the cloud

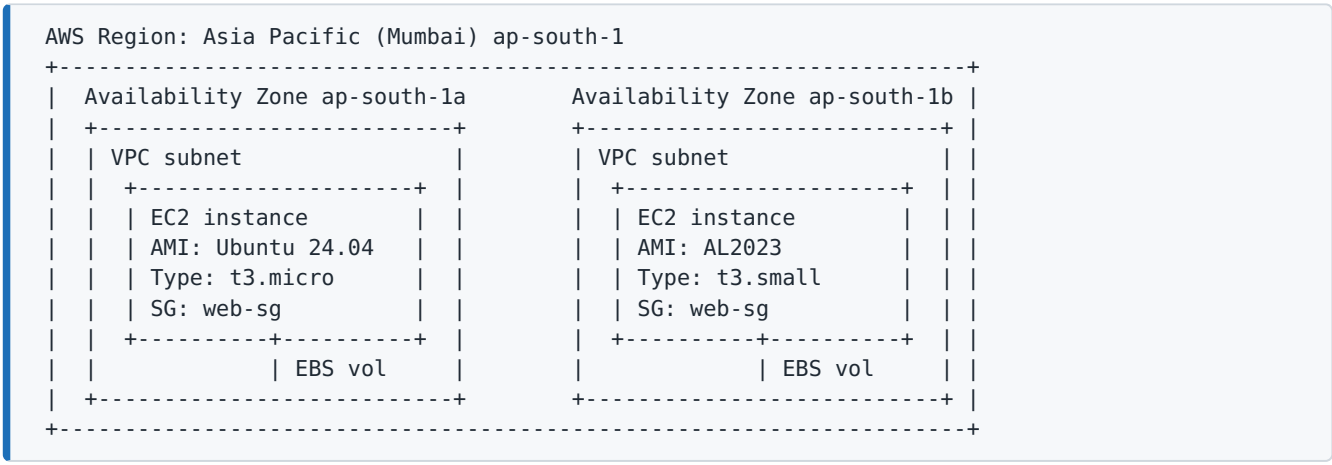
Cloud service models: IaaS (you manage OS and above — EC2), **PaaS** (you manage only code — Elastic Beanstalk), **SaaS** (you just use the software — Gmail). EC2 is IaaS.

> **i Shared responsibility model**
>
> **AWS** is responsible for the data centres, hardware, hypervisor and global network. **You** are responsible for the guest OS (updates, patches), security group rules, users and keys, applications and data. A hacked EC2 because of an open port 22 with a weak password is your responsibility.

5.2 What is EC2? Key concepts

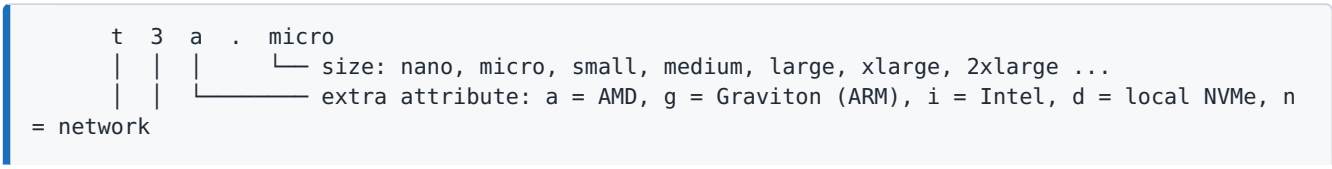
Think of EC2 like renting a fully furnished flat instead of building a house: you choose the size (instance type), the interior setup (AMI), the lock and key (key pair) and the society gate rules (security group). Technically, EC2 runs your virtual machine on AWS's Nitro hypervisor in a data centre of the region you choose. Bagha the key terms:

Amazon EC2 provides resizable virtual servers called **instances** in the AWS cloud.



Term	What it is	Example / notes
Instance	A virtual server	`i-0abc123def4567890`
AMI (Amazon Machine Image)	Template with OS + software used to launch instances	Amazon Linux 2023, Ubuntu 24.04, CentOS Stream 9, your own custom AMI
Instance type	Hardware size: vCPU, RAM, network	`t3.micro` (2 vCPU, 1 GiB), `t3.medium` (2 vCPU, 4 GiB)
Key pair	Public/private key used for SSH login instead of a password	AWS keeps public key; you download `mykey.pem` **once**
Security group (SG)	Stateful virtual firewall at instance level	Allow TCP 22 from My IP, TCP 80/443 from `0.0.0.0/0`
EBS volume	Network-attached block storage (virtual hard disk)	Root volume 8 GiB gp3
Instance store	Temporary disk physically attached to host	Data lost on stop/terminate (only on some types)
Elastic IP	Static public IPv4 address	Survives stop/start; re-mappable
Region	Geographic area with multiple data centres	`ap-south-1` Mumbai, `us-east-1` N. Virginia
Availability Zone (AZ)	One or more isolated data centres inside a region	`ap-south-1a`, `ap-south-1b`, `ap-south-1c`
VPC / subnet	Your private network / a slice of it in one AZ	Default VPC `172.31.0.0/16`
User data	Script that runs automatically at first boot	Install Nginx on launch
IAM role (instance profile)	Gives the instance AWS permissions without storing access keys	Allow instance to read from S3

Instance type naming



```
        │  └──────────── generation (higher = newer, better price/performance)
        └─────────────── family: t = burstable general purpose
```

Family	Category	Use case	Examples
T	General purpose, **burstable** (CPU credits)	Dev/test, small websites, labs	`t3.micro`, `t3.small`, `t4g.small`
M	General purpose, balanced	App servers, small databases	`m7i.large`, `m7g.large`
C	Compute optimised	Batch processing, gaming, high-traffic web	`c7i.xlarge`
R / X	Memory optimised	Large databases, in-memory caches	`r7i.large`
I / D	Storage optimised	NoSQL, data warehouses	`i4i.large`
P / G / Inf / Trn	Accelerated (GPU/ML chips)	Machine learning, video	`g5.xlarge`

✔ Graviton (ARM) instances

Types with **g** (e.g. `t4g`, `m7g`) use AWS Graviton ARM processors — usually cheaper for the same performance. Choose an **arm64** AMI for them. All software in this book works on both x86_64 and arm64.

Pricing models

Model	How it works	Discount vs On-Demand	Best for
On-Demand	Pay per second (Linux, min 60 s); no commitment	—	Labs, unpredictable workloads
Savings Plans	Commit to a $/hour spend for 1 or 3 years	Up to ~72%	Steady long-term usage
Reserved Instances	Commit to a specific instance type/region for 1 or 3 years	Up to ~72%	Always-on production servers
Spot Instances	Use spare capacity; AWS can reclaim with 2-minute notice	Up to ~90%	Fault-tolerant batch jobs, CI, rendering
Dedicated Hosts / Instances	Physical server dedicated to you	Premium	Licensing, compliance

What you pay for: instance running time, EBS storage (even when the instance is stopped), snapshots, public IPv4 addresses (including Elastic IPs), and data transfer **out** to the internet.

⚠ Free tier

AWS offers a free tier / free plan for new accounts (typically covering small instance types such as `t2.micro` / `t3.micro`, some EBS storage and limited data transfer, or credits for new accounts). The exact offer depends on when your account was created and **changes over time** — always check aws.amazon.com/free and look for the **"Free tier eligible"** label in the console. Set up an **AWS Budget** with e-mail alerts before starting labs.

> **Ravindra's Tip**
>
> The very first thing to do in a new AWS account is not launching an instance — it is creating an **AWS Budget** with an email alert (even a small amount like ₹500 equivalent) and enabling MFA on the root user. Then create an IAM user for daily work. Security and cost control first, experiments later.

5.3 How to launch an EC2 instance (console, step by step)

Chala, aata actual launch karuya. Open the console and do every step along with me. **Before you start:** sign in to the AWS Management Console, and select the **Mumbai (ap-south-1)** region from the top-right region menu.

Step 1 — Open EC2. Search "EC2" in the top search bar → **EC2 Dashboard** → **Launch instance**.

Step 2 — Name and tags. Name: `web-server-1`. (This creates a tag `Name=web-server-1`.)

Step 3 — Choose an AMI. Under Application and OS Images choose one of:

- **Amazon Linux 2023 AMI** (default, free-tier eligible) — user `ec2-user`
- **Ubuntu Server 24.04 LTS** (or 22.04 LTS) — user `ubuntu`
- **CentOS Stream 9** — search Browse more AMIs → AWS Marketplace / Community AMIs for the official "CentOS Stream 9" image — user `ec2-user` (older CentOS images used `centos`)

Keep architecture **64-bit (x86)** unless you chose a Graviton type.

Step 4 — Instance type. Choose a free-tier eligible type such as `t3.micro` or `t2.micro` (label shown in the list).

Step 5 — Key pair (login). Click **Create new key pair** → name `mykey` → type **RSA** or **ED25519** → format **.pem** (for OpenSSH on Linux/Mac/Windows 10+) or **.ppk** (for PuTTY) → **Create**. The file downloads **only once** — keep it safe.

Step 6 — Network settings. Click **Edit**:

- VPC: default VPC; Subnet: No preference (or pick an AZ)
- **Auto-assign public IP: Enable**
- Firewall (security groups): **Create security group**, name `web-sg`, with inbound rules:

Type	Protocol	Port	Source	Why
SSH	TCP	22	**My IP** (e.g. `49.36.12.7/32`)	Only you can log in
HTTP	TCP	80	Anywhere-IPv4 `0.0.0.0/0`	Website
HTTPS	TCP	443	Anywhere-IPv4 `0.0.0.0/0`	Secure website

Step 7 — Configure storage. Default root volume 8 GiB **gp3** (Ubuntu defaults to 8 GiB; you can set up to 30 GiB within typical free-tier storage limits).

Step 8 — Advanced details (optional). Paste a **User data** script to auto-install a web server at first boot, for example on Amazon Linux 2023:

```
#!/bin/bash
yum install -y nginx
echo "<h1>Hello from $(hostname -f)</h1>" > /usr/share/nginx/html/index.html
service nginx start
systemctl enable nginx
```

Step 9 — Launch. Review the Summary panel → **Launch instance** → **View all instances**. Wait until Instance state = Running and Status checks = 2/2 checks passed. Copy the **Public IPv4 address**.

Step 10 — Test. If you used the user-data script, open `http://<PUBLIC_IP>` in a browser (note: **http**, not https).

> ⚠ **The browser shows a timeout?**
>
> Almost always one of: (1) security group does not allow port 80 from `0.0.0.0/0`, (2) you typed `https://` but no TLS is configured, (3) the web server is not running, (4) instance is in a subnet without a route to an Internet Gateway, or (5) no public IP assigned.

5.4 Connecting to your instance

In my sessions, this is where most students get stuck for the first time — wrong username, wrong key permissions, or port 22 blocked. Kalji karu naka, follow the steps exactly and it will work.

Default usernames

AMI	SSH user
Amazon Linux 2023 / Amazon Linux 2	`ec2-user`
Ubuntu	`ubuntu`
CentOS Stream 9 (official)	`ec2-user` (older CentOS 7/8 AMIs: `centos`)
RHEL	`ec2-user`
Debian	`admin`
SUSE	`ec2-user`

From Linux or macOS (OpenSSH)

```
cd ~/Downloads
chmod 400 mykey.pem                          # private key must not be readable by others
ssh -i mykey.pem ec2-user@<PUBLIC_IP>        # Amazon Linux / CentOS Stream
ssh -i mykey.pem ubuntu@<PUBLIC_IP>          # Ubuntu
```

Type `yes` when asked "Are you sure you want to continue connecting?" (first time only — it stores the server fingerprint in `~/.ssh/known_hosts`).

> **✖ UNPROTECTED PRIVATE KEY FILE!**
>
> If you see `Permissions 0644 for 'mykey.pem' are too open`, SSH refuses to use the key. Fix with `chmod 400 mykey.pem`. Never share your `.pem`, never upload it to GitHub, never e-mail it.

From Windows — option 1: built-in OpenSSH (Windows 10/11)

Open **PowerShell** and fix the key permissions (Windows equivalent of `chmod 400`):

```
cd $env:USERPROFILE\Downloads
icacls.exe mykey.pem /reset
icacls.exe mykey.pem /grant:r "$($env:USERNAME):(R)"
icacls.exe mykey.pem /inheritance:r
ssh -i .\mykey.pem ec2-user@<PUBLIC_IP>
```

From Windows — option 2: PuTTY

1. Install PuTTY (includes **PuTTYgen**).
2. If you downloaded a `.pem`: open **PuTTYgen → Load** → choose All files → select `mykey.pem` → **Save private key** as `mykey.ppk`.
3. Open **PuTTY** → Session → Host Name: `ec2-user@<PUBLIC_IP>` (or `ubuntu@...`), Port 22.
4. Connection → SSH → Auth → Credentials → Private key file for authentication → browse to `mykey.ppk`.
5. Back to Session, type a name under Saved Sessions → **Save** → **Open** → **Accept** the host key.

From Windows — option 3: MobaXterm

1. Install MobaXterm (Home edition is free).
2. **Session → SSH** → Remote host: `<PUBLIC_IP>`, tick Specify username: `ec2-user` / `ubuntu`.
3. Advanced SSH settings → tick **Use private key** → select `mykey.pem` (MobaXterm accepts .pem directly).
4. **OK**. The left panel gives a graphical SFTP browser — drag-and-drop files to upload.

Other connection options

- **EC2 Instance Connect** (browser-based): select instance → **Connect** → EC2 Instance Connect tab → **Connect**. Works out-of-the-box on Amazon Linux and Ubuntu AMIs when port 22 is open to the EC2 Instance Connect IP range for your region (or to `0.0.0.0/0`).
- **Session Manager** (AWS Systems Manager): no port 22 needed at all; requires the SSM agent and an IAM role (`AmazonSSMManagedInstanceCore`). Recommended in companies.

First commands after logging in

```
cat /etc/os-release                # confirm distribution
sudo yum update -y                 # AL2023 / CentOS Stream
sudo apt update && sudo apt upgrade -y   # Ubuntu
curl -s http://169.254.169.254/latest/meta-data/ -H "X-aws-ec2-metadata-token: $(curl -s -X PUT h
```

```
ttp://169.254.169.254/latest/api/token -H 'X-aws-ec2-metadata-token-ttl-seconds: 60')"
sudo timedatectl set-timezone Asia/Kolkata
```

i Instance metadata (IMDSv2)

`169.254.169.254` is a special address inside every instance that returns its metadata — instance ID, public IP, AZ, IAM role credentials. New instances require **IMDSv2** (token-based), which is why the command above first requests a token. Example: append `public-ipv4` or `placement/availability-zone` to the URL.

RB Ravindra's Tip

Keep one folder like `~/aws-keys/` for all your `.pem` files, run `chmod 400` on each, and name keys clearly (`mumbai-lab-key.pem`). If you lose a private key, AWS cannot give it back. And never, ever push a key to GitHub — bots scan public repositories for secrets within minutes.

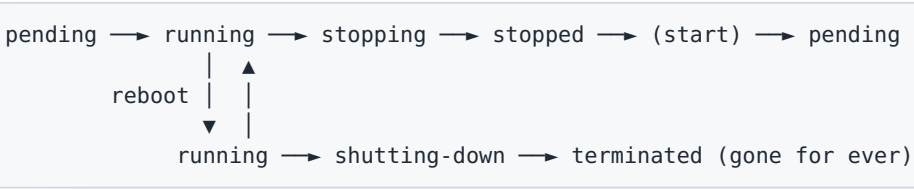
5.5 Elastic IP (static public IP)

```
EC2 console → Network & Security → Elastic IPs → Allocate Elastic IP address → Allocate
Select the EIP → Actions → Associate Elastic IP address → choose instance → Associate
```

To remove: **Disassociate**, then **Release** (otherwise you keep paying for it). For the full story — charges, the 5-per-region limit, and replacing a server without changing DNS — see **Chapter 14, section 14.1**. To give the Elastic IP a domain name, see **Chapter 9**.

5.6 Instance lifecycle: stop, reboot, hibernate, terminate

Dhyan do — this table decides whether you lose data and whether you keep paying. Read it twice.



Action	What happens	Public IP	EBS root volume	Billing
Reboot	OS restarts on same host	Kept	Kept	Continues
Stop	Instance shut down; may move to new host on start	**Released** (new one on start); EIP kept	Kept	Instance charges stop; **EBS & EIP still charged**
Hibernate	RAM saved to EBS, then stop	Same as stop	Kept	Same as stop
Terminate	Instance deleted permanently	Released	**Deleted** if Delete on termination = yes (default for root)	All instance charges stop

> **✖ Terminate is permanent**
>
> Terminated instances cannot be recovered. Enable **Termination protection** (Actions → Instance settings → Change termination protection) for important servers, and take an EBS snapshot or AMI before experimenting.

5.7 EC2 best practices

- **Least privilege security groups:** SSH (22) only from My IP; never open databases (3306, 27017) to `0.0.0.0/0`.
- **Protect keys:** `chmod 400`, one key per person/team, rotate when someone leaves. Never commit keys to Git.
- **Use IAM roles** for instances instead of storing AWS access keys on the server.
- **Keep the OS patched:** `sudo yum update -y` / `sudo apt upgrade -y` regularly.
- **Right-size:** start small (`t3.micro`), monitor with CloudWatch, then scale up.
- **Tag everything:** `Name`, `Owner`, `Project`, `Environment` — helps billing and clean-up.
- **Backups:** EBS snapshots (automate with Amazon Data Lifecycle Manager) or AMIs.
- **Use Elastic IP or a DNS name** for servers that need a fixed address.
- **High availability:** production apps run in ≥ 2 AZs behind a load balancer.
- **Cost hygiene:** stop idle instances; delete unattached volumes, old snapshots and unused EIPs; set AWS Budgets alerts.
- **Monitor:** CloudWatch metrics and alarms (CPU, status check failures).

Common Mistakes I See Students Make

- **Selecting the wrong region** (e.g. N. Virginia) and then "losing" the instance. The console shows resources only for the selected region.
- **SSH source set to My IP, then changing Wi-Fi/mobile hotspot.** Your public IP changes and SSH times out — update the rule.
- **Wrong username**: `ubuntu@` for Amazon Linux, or `ec2-user@` for Ubuntu → Permission denied (publickey).
- **Forgetting `chmod 400` on the `.pem` file.**
- **Typing `https://` for a server that only has HTTP** and thinking the server is down.
- **Terminating when they meant to stop**, and losing the root volume.
- **Leaving instances, Elastic IPs and volumes running after the lab** and getting a bill.

Interview Questions

Q1. What is an AMI?

An Amazon Machine Image is a template containing the OS, software and configuration (backed by EBS snapshots) used to launch EC2 instances.

Q2. What is a security group? Is it stateful?

A virtual firewall at the instance/ENI level with allow rules for inbound and outbound traffic. It is stateful — return traffic for an allowed connection is automatically allowed.

Q3. What is the difference between stopping and terminating an instance?

Stop shuts it down and keeps the EBS root volume (auto public IP is released, EBS still billed). Terminate deletes the instance permanently and, by default, its root volume.

Q4. Explain EC2 pricing models.

On-Demand (pay per second, no commitment), Savings Plans and Reserved Instances (commitment for big discounts), Spot (spare capacity, up to ~90% off, can be interrupted), Dedicated Hosts/Instances (physical isolation).

Q5. How do you connect to a Linux EC2 instance from Windows?

Using the built-in OpenSSH client (`ssh -i key.pem user@IP` after fixing key permissions with icacls), PuTTY with a .ppk key, MobaXterm, EC2 Instance Connect or Session Manager.

Q6. What is user data?

A script passed at launch that runs automatically (as root, via cloud-init) on the first boot — used to install packages or configure the server.

Q7. What is the instance metadata service?

An endpoint at 169.254.169.254 inside each instance that returns metadata like instance ID, IPs, AZ and IAM role credentials. IMDSv2 requires a session token.

Practice Questions and Lab Exercises

1. List five advantages of EC2 over on-premises servers. Explain the shared responsibility model.
2. Define AMI, instance type, key pair, security group, EBS, Elastic IP, Region and AZ.
3. Decode the instance type `m7g.2xlarge`.
4. Compare On-Demand, Reserved, Savings Plans and Spot pricing with a use case for each.
5. What are the default SSH users for Amazon Linux, Ubuntu and CentOS Stream AMIs?
6. What happens to the public IP, the EBS volume and billing when you stop vs terminate an instance?
7. **Lab:** Launch an Amazon Linux 2023 `t3.micro` in Mumbai with a security group allowing SSH (My IP) and HTTP. Use the user-data script from Step 8 and open the page in a browser.
8. **Lab:** Connect to the same instance from Windows using both PuTTY and OpenSSH.
9. **Lab:** Stop and start the instance and note the public IP change. Then allocate and associate an Elastic IP, repeat, and observe. Finally release the EIP.

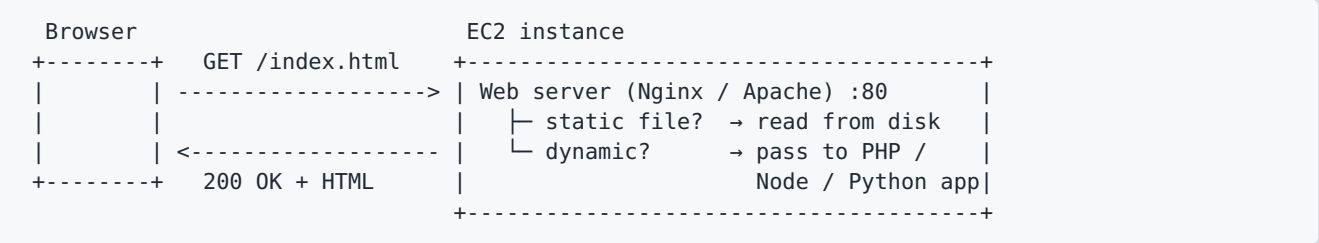
★ Quick Revision

- EC2 = IaaS virtual servers; pay-as-you-go; launch in minutes.
- Launch = AMI + instance type + key pair + security group + storage (+ user data).
- Instance types: family/generation/attribute.size, e.g. `t3.micro`.
- Pricing: On-Demand, Savings Plans, Reserved, Spot, Dedicated.
- SSH: `chmod 400 key.pem; ssh -i key.pem ec2-user@IP` (Ubuntu → `ubuntu`). Windows: OpenSSH (icacls), PuTTY (.ppk), MobaXterm.
- Stop keeps EBS but releases auto public IP; terminate deletes root EBS. Use Elastic IP for fixed IP.

Chapter 6: Web Servers — Nginx and Apache

6.1 What is a web server?

Mitranno, till now we launched a server. But a server without a web server is like a shop with the shutter down — customers come but nobody serves them. Let's understand what a web server really does. A **web server** is software that listens on a port (80 for HTTP, 443 for HTTPS), accepts HTTP requests from browsers and returns responses — HTML pages, images, CSS, JavaScript files, or data produced by an application.



A web server does three main jobs:

1. **Serve static content** — files that are the same for everyone (HTML, CSS, JS, images).
2. **Pass dynamic requests** to an application (PHP-FPM, Node.js, Gunicorn) — called **reverse proxying** — and return its output.
3. **Handle cross-cutting concerns** — TLS/HTTPS, compression, caching, logging, virtual hosts (many websites on one server), load balancing, rate limiting.

Common HTTP status codes

Code	Meaning	Typical cause
200	OK	Success
301 / 302	Moved permanently / temporarily	Redirect (e.g. HTTP → HTTPS)
304	Not modified	Browser cache is fresh
400	Bad request	Malformed request
401 / 403	Unauthorised / **Forbidden**	Missing login / file permissions, no index file, SELinux
404	**Not Found**	Wrong path or wrong document root
500	Internal server error	Bug in app / PHP error
502	**Bad Gateway**	Reverse-proxy target (app) is down or wrong port/socket
503	Service unavailable	Overloaded or maintenance
504	Gateway timeout	App took too long to respond

6.2 Nginx vs Apache

In my sessions, students often ask: "Sir, which one is better?" Honest answer — both are excellent; the right choice depends on the use case. Bagha the comparison:

Apache HTTP Server is the classic, highly flexible web server with a module for almost everything. **Nginx** (pronounced "engine-x") was designed to solve the C10K problem (10,000 simultaneous connections) using an event-driven architecture. Today both are widely used, often together.

Feature	Nginx	Apache (httpd / apache2)
Architecture	Event-driven, asynchronous; few worker processes handle thousands of connections	Process/thread per connection (MPMs: prefork, worker, **event**)
Static content	Extremely fast, low memory	Fast, but uses more memory under heavy load
Dynamic content (PHP)	Via external **PHP-FPM** (FastCGI)	Via `mod_php` (embedded) or PHP-FPM
Configuration	Central config files; `server {}` blocks	Central config + per-directory `.htaccess` files
`.htaccess` support	No (faster, but less flexible for shared hosting)	Yes (WordPress pretty links, shared hosting)
Reverse proxy / load balancer	Excellent, built in	Good (`mod_proxy`, `mod_proxy_balancer`)
Modules	Compiled in or dynamic modules	Huge set of dynamically loadable modules
Virtual hosts	`server { server_name ... }`	`<VirtualHost *:80> ServerName ... </VirtualHost>`
Typical use	Reverse proxy for Node/Python, static sites, high traffic	LAMP apps, WordPress, legacy apps, shared hosting

✔ Which should I choose?

For static sites and as a reverse proxy in front of Node.js/Python apps, **Nginx** is the default modern choice. For PHP applications that rely on `.htaccess` (many WordPress plugins), **Apache** is the easiest. Only **one** of them can listen on port 80 at a time — stop one before starting the other on the same server.

RB Ravindra's Tip

Keep this quick-reference table open whenever you work on a new distribution. Most beginner errors are simply using the Ubuntu path on Amazon Linux (or vice versa) — `apache2` vs `httpd`, `/var/www/html` vs `/usr/share/nginx/html`. Check the OS first with `cat /etc/os-release`, then choose the path.

6.3 Quick reference: package, service, paths per OS

Item	Amazon Linux 2023	Ubuntu 22.04 / 24.04	CentOS Stream 9
Nginx package / service	`nginx` / `nginx`	`nginx` / `nginx`	`nginx` / `nginx`
Nginx main config	`/etc/nginx/nginx.conf`	`/etc/nginx/nginx.conf`	`/etc/nginx/nginx.conf`
Nginx site configs	`/etc/nginx/conf.d/*.conf`	`/etc/nginx/sites-available/` + symlink in `sites-enabled/` (also `conf.d/`)	`/etc/nginx/conf.d/*.conf`
Nginx default web root	`/usr/share/nginx/html`	`/var/www/html`	`/usr/share/nginx/html`
Nginx worker user	`nginx`	`www-data`	`nginx`
Nginx logs	`/var/log/nginx/`	`/var/log/nginx/`	`/var/log/nginx/`
Apache package / service	`httpd` / `httpd`	`apache2` / `apache2`	`httpd` / `httpd`
Apache main config	`/etc/httpd/conf/httpd.conf`	`/etc/apache2/apache2.conf`	`/etc/httpd/conf/httpd.conf`
Apache site configs	`/etc/httpd/conf.d/*.conf`	`/etc/apache2/sites-available/` (enable with `a2ensite`)	`/etc/httpd/conf.d/*.conf`
Apache default web root	`/var/www/html`	`/var/www/html`	`/var/www/html`
Apache user	`apache`	`www-data`	`apache`
Apache logs	`/var/log/httpd/`	`/var/log/apache2/`	`/var/log/httpd/`
Config test	`sudo nginx -t` / `sudo apachectl configtest`	`sudo nginx -t` / `sudo apache2ctl configtest`	`sudo nginx -t` / `sudo apachectl configtest`
Auto-start after install?	No — `service ... start` + `systemctl enable`	**Yes**	No — `service ... start` + `systemctl enable`
Extra step	—	`ufw` if enabled	**firewalld** + **SELinux**

6.4 Amazon Linux 2023

Chala, aata install karuya. Pick the OS of your instance and type these commands with me.

Nginx on Amazon Linux 2023

```
sudo yum install -y nginx
sudo service nginx start
sudo systemctl enable nginx
sudo service nginx status
sudo nginx -t                              # test configuration syntax
curl -I http://localhost                   # expect HTTP/1.1 200 OK
ls /usr/share/nginx/html                   # default web root
```

Apache on Amazon Linux 2023

```
sudo yum install -y httpd
sudo service httpd start
sudo systemctl enable httpd
sudo service httpd status
sudo apachectl configtest                    # expect "Syntax OK"
echo "<h1>Apache on Amazon Linux 2023</h1>" | sudo tee /var/www/html/index.html
curl http://localhost
```

> **i Amazon Linux 2 (older)**
>
> On Amazon Linux 2 use `sudo yum install -y httpd`, and for Nginx `sudo amazon-linux-extras install -y nginx1`. Everything else (paths, service names) is the same.

6.5 Ubuntu 22.04 / 24.04

Nginx on Ubuntu

```
sudo apt update
sudo apt install -y nginx
sudo service nginx start
sudo systemctl enable nginx          # already started by apt, this makes sure
sudo service nginx status
sudo nginx -t
curl -I http://localhost
ls /var/www/html                             # contains index.nginx-debian.html
ls -l /etc/nginx/sites-enabled/              # 'default' -> ../sites-available/default
```

Apache on Ubuntu

```
sudo apt update
sudo apt install -y apache2
sudo service apache2 start
sudo systemctl enable apache2
sudo service apache2 status
sudo apache2ctl configtest
curl -I http://localhost
```

Useful Ubuntu-only Apache helpers:

Command	Purpose
`sudo a2ensite mysite.conf` / `a2dissite`	Enable / disable a virtual host
`sudo a2enmod rewrite proxy proxy_http` / `a2dismod`	Enable / disable modules
`sudo a2enconf` / `a2disconf`	Enable / disable extra config snippets

> **i Ubuntu firewall (ufw)**
>
> `ufw` is **inactive** by default on Ubuntu EC2 AMIs — the AWS security group is your firewall. If you enable ufw, allow SSH first or you will lock yourself out: `sudo ufw allow OpenSSH && sudo ufw allow 'Nginx Full' && sudo ufw enable` (use `'Apache Full'` for Apache).

6.6 CentOS Stream 9

Nginx on CentOS Stream 9

```
sudo yum install -y nginx
sudo service nginx start
sudo systemctl enable nginx
sudo service nginx status
sudo nginx -t
curl -I http://localhost
```

Apache on CentOS Stream 9

```
sudo yum install -y httpd
sudo service httpd start
sudo systemctl enable httpd
sudo service httpd status
sudo apachectl configtest
curl -I http://localhost          # shows the CentOS test page until you add index.html
```

firewalld (only if it is running)

CentOS cloud images may or may not have `firewalld` running. Check, and if active, open HTTP/HTTPS:

```
sudo systemctl is-active firewalld
sudo firewall-cmd --permanent --add-service=http
sudo firewall-cmd --permanent --add-service=https
sudo firewall-cmd --reload
sudo firewall-cmd --list-all
```

If `firewalld` is inactive or not installed, the AWS security group alone controls access — that is fine.

SELinux basics

Dhyan do — SELinux is the number one reason why something that works on Ubuntu gives "403 Forbidden" or "502 Bad Gateway" on CentOS. It is not your enemy; it is an extra security guard. Let's learn how to talk to it.

CentOS Stream runs **SELinux in enforcing mode**. SELinux labels every file and process; a web server may only read files labelled `httpd_sys_content_t` (both Nginx and Apache run in the `httpd_t` domain).

```
getenforce                                  # Enforcing / Permissive / Disabled
ls -Z /usr/share/nginx/html /var/www/html   # view SELinux labels
sudo restorecon -Rv /var/www/html           # fix labels after copying/moving files
sudo setsebool -P httpd_can_network_connect 1      # allow reverse proxy to apps (Node/Flask)
sudo setsebool -P httpd_can_network_connect_db 1   # allow web server/PHP to reach a remote DB
sudo ausearch -m avc -ts recent             # see recent SELinux denials (package audit)
```

> ⚠ **Do not simply disable SELinux**
>
> Turning SELinux off (`setenforce 0`) "fixes" 403/502 errors but removes an important security layer. Fix the label or boolean instead. Temporarily using `sudo setenforce 0` **only to confirm** that SELinux is the cause is fine — then set it back with `sudo setenforce 1`.

6.7 Understanding the configuration files

Nginx structure

```
# /etc/nginx/nginx.conf (simplified)
user nginx;                         # www-data on Ubuntu
worker_processes auto;              # one worker per CPU core
events { worker_connections 1024; }
http {
    include       /etc/nginx/mime.types;
    access_log    /var/log/nginx/access.log;
    sendfile      on;
    include /etc/nginx/conf.d/*.conf;        # your site files
    # Ubuntu also has: include /etc/nginx/sites-enabled/*;

    server {                                 # default server block
        listen       80;
        server_name  _;
        root         /usr/share/nginx/html;
        location / { try_files $uri $uri/ =404; }
    }
}
```

Apache structure

```
# /etc/httpd/conf/httpd.conf (AL2023/CentOS, simplified)
Listen 80
User apache
Group apache
DocumentRoot "/var/www/html"
<Directory "/var/www/html">
    Options Indexes FollowSymLinks
    # change None to All to allow .htaccess files
    AllowOverride None
    Require all granted
</Directory>
ErrorLog "logs/error_log"
IncludeOptional conf.d/*.conf
```

> **Ravindra's Tip**
>
> Make `sudo nginx -t && sudo service nginx reload` a single habit — never restart blindly. In production, a restart with a broken config means downtime; a failed `-t` test costs you nothing. Same for Apache with `apachectl configtest`.

> **i Going deeper**
>
> This section gives only the big picture. In **Chapter 8** we read these files line by line and change the document root, ports and virtual hosts.

6.8 Everyday management commands

Task	Nginx	Apache (AL2023/CentOS)	Apache (Ubuntu)
Test config	`sudo nginx -t`	`sudo apachectl configtest`	`sudo apache2ctl configtest`
Reload (no downtime)	`sudo service nginx reload`	`sudo service httpd reload`	`sudo service apache2 reload`
Restart	`sudo service nginx restart`	`sudo service httpd restart`	`sudo service apache2 restart`
Error log	`sudo tail -f /var/log/nginx/error.log`	`sudo tail -f /var/log/httpd/error_log`	`sudo tail -f /var/log/apache2/error.log`
Access log	`/var/log/nginx/access.log`	`/var/log/httpd/access_log`	`/var/log/apache2/access.log`
Version	`nginx -v`	`httpd -v`	`apache2 -v`
Loaded modules	`nginx -V`	`httpd -M`	`apache2ctl -M`

> **✓ Always test before reload**
>
> Use `sudo nginx -t && sudo service nginx reload`. The reload runs only if the test passes, so a typo never takes your site down.

Common Mistakes I See Students Make

- **Installing both Nginx and Apache and starting both** → Address already in use on port 80.
- **Editing the config but not reloading** the service, then wondering why nothing changed.
- **Putting files in `/var/www/html` while Nginx on Amazon Linux serves `/usr/share/nginx/html`.**
- **Using `apache2` commands on Amazon Linux/CentOS** (it is `httpd` there).
- **Disabling SELinux permanently** instead of fixing labels or booleans.
- **Adding comments at the end of an Apache directive line** (Apache does not allow inline comments).
- **Forgetting the semicolon `;` at the end of an Nginx directive** — `nginx -t` will tell you the exact line.

Interview Questions

Q1. What is the difference between Nginx and Apache?

Nginx is event-driven and asynchronous, very efficient for static content and reverse proxying. Apache uses process/thread-based MPMs, has many modules and supports per-directory `.htaccess` files.

Q2. What is a reverse proxy?

A server that receives client requests and forwards them to backend servers/apps (e.g. Node on 127.0.0.1:3000), then returns the response. It provides TLS termination, caching, load balancing and security.

Q3. How do you test Nginx configuration before applying it?

`sudo nginx -t`, then `sudo service nginx reload` if the test passes.

Q4. What do HTTP 403, 404 and 502 mean?

403 Forbidden (permission/SELinux/no index), 404 Not Found (wrong path/root), 502 Bad Gateway (the upstream app behind the proxy is down or unreachable).

Q5. What is SELinux and how do you fix a web-content label issue?

Security-Enhanced Linux, a mandatory access control system on RHEL/CentOS. Fix labels with `sudo restorecon -Rv /var/www/site` (or `semanage fcontext` for custom paths).

Q6. What is `.htaccess` and why doesn't Nginx support it?

A per-directory Apache config file read on every request. Nginx avoids it for performance; all configuration lives in central files.

Practice Questions and Lab Exercises

1. What are the three main jobs of a web server?
2. Give six differences between Nginx and Apache. When would you choose each?
3. Write the package name, service name, config path and default web root of Apache on Ubuntu and on CentOS Stream 9.
4. What is `.htaccess`? Why doesn't Nginx support it?
5. What do HTTP status codes 403, 404 and 502 mean? Give one cause of each.
6. What is SELinux? Which command fixes file labels after copying website files?
7. **Lab:** Install Nginx on Amazon Linux 2023, verify with `curl -I`, then stop Nginx, install Apache and verify it serves a custom page.
8. **Lab:** On Ubuntu, list the files in `sites-available` and `sites-enabled` and explain the symlink.
9. **Lab:** On CentOS Stream 9, check `getenforce` and the firewalld status and open HTTP if needed.

★ Quick Revision

- Web server = serves static files + reverse-proxies dynamic requests + TLS/logging/vhosts.
- Nginx = event-driven, great reverse proxy; Apache = modules + `.htaccess`.
- Apache is `httpd` on AL2023/CentOS, `apache2` on Ubuntu.
- Nginx root: `/usr/share/nginx/html` (AL2023/CentOS), `/var/www/html` (Ubuntu). Apache root: `/var/www/html`.
- `sudo service <name> start` + `sudo systemctl enable <name>`; test with `nginx -t` / `apachectl configtest`.
- CentOS: firewalld `--add-service=http`, SELinux `restorecon`, `setsebool -P httpd_can_network_connect 1`.

Chapter 7: Static Website Hosting (3 OSes × 2 Web Servers)

7.1 Why and what

Mitranno, this is the chapter where you will see your own web page live on the internet for the first time — that moment is always special in my sessions. Let's first understand what a static website is. A **static website** is made only of files — HTML, CSS, JavaScript, images, fonts — that are sent to the browser exactly as stored on disk. There is no server-side code and no database. Examples: portfolio/resume sites, college fest pages, documentation, landing pages and the built output of React/Angular/Vue apps.

Why host it on EC2? It is the best way to learn web-server configuration, permissions and troubleshooting. (In production, very simple static sites are often hosted on Amazon S3 + CloudFront, but the EC2 skills here apply to every dynamic app you will host later.)

How it works: Browser → port 80 on the EC2 public IP → security group allows it → Nginx/Apache maps the URL path to a file under the **document root** → returns the file.

```
URL:   http://13.233.10.25/css/style.css
root:  /var/www/mysite
file:  /var/www/mysite/css/style.css     (root + URL path)
```

7.2 The hosting matrix

Six combinations may sound like a lot, but bagha — the idea is the same everywhere: files in a folder, a config that points to that folder, correct permissions, reload. Only paths and names change. Samjla ka?

We will host the same site in all **six** combinations. All six use a dedicated folder `/var/www/mysite` and a virtual host (server block), which is the professional way. A quick "default root" method is also shown.

OS	Web server	Install	Site config file	Document root used	Web-server user	Extra step
Amazon Linux 2023	Nginx	`yum install nginx`	`/etc/nginx/conf.d/mysite.conf`	`/var/www/mysite`	`nginx`	—
Amazon Linux 2023	Apache	`yum install httpd`	`/etc/httpd/conf.d/mysite.conf`	`/var/www/mysite`	`apache`	—
Ubuntu 22.04/24.04	Nginx	`apt install nginx`	`/etc/nginx/sites-available/mysite` + symlink	`/var/www/mysite`	`www-data`	disable `default` site

OS	Web server	Install	Site config file	Document root used	Web-server user	Extra step
Ubuntu 22.04/24.04	Apache	`apt install apache2`	`/etc/apache2/sites-available/mysite.conf`	`/var/www/mysite`	`www-data`	`a2ensite`, `a2dissite 000-default`
CentOS Stream 9	Nginx	`yum install nginx`	`/etc/nginx/conf.d/mysite.conf`	`/var/www/mysite`	`nginx`	firewalld, `restorecon`
CentOS Stream 9	Apache	`yum install httpd`	`/etc/httpd/conf.d/mysite.conf`	`/var/www/mysite`	`apache`	firewalld, `restorecon`

Default document roots (quick method — just copy files here): Nginx on AL2023/CentOS → `/usr/share/nginx/html`; Nginx on Ubuntu → `/var/www/html`; Apache on all three → `/var/www/html`.

> **i Pre-requisites for every combination**
>
> 1. EC2 instance running, security group allows **TCP 80** (and 443 later) from `0.0.0.0/0` and **TCP 22** from your IP.
> 2. You are logged in via SSH as `ec2-user` (AL2023/CentOS) or `ubuntu` (Ubuntu).
> 3. Only **one** web server runs on port 80. If you installed both, stop and disable the other one.

7.3 Step 1 (common): create the sample website

Now open your terminal and try this with me. Run this on the server (it creates `~/mysite` with an `index.html`, a CSS file and a custom 404 page):

```
mkdir -p ~/mysite/css
cat > ~/mysite/index.html <<'EOF'
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>My First Cloud Website</title>
  <link rel="stylesheet" href="css/style.css">
</head>
<body>
  <header><h1>Namaste from AWS EC2!</h1></header>
  <main>
    <p>This static website is served by a web server running on an EC2 instance.</p>
    <ul>
      <li>Cloud: Amazon Web Services</li>
      <li>Server: EC2 in the Mumbai region</li>
      <li>Content: HTML + CSS</li>
    </ul>
  </main>
  <footer>Hosted as part of the AWS Hands-On Handbook lab</footer>
</body>
</html>
EOF
cat > ~/mysite/css/style.css <<'EOF'
```

```
body   { font-family: Arial, sans-serif; margin: 0; background: #f4f7fb; color: #222; }
header { background: #0b3d6e; color: #fff; padding: 20px; text-align: center; }
main   { max-width: 700px; margin: 30px auto; background: #fff; padding: 20px; border-radius:
8px; }
footer { text-align: center; color: #777; padding: 20px; }
EOF
cat > ~/mysite/404.html <<'EOF'
<!DOCTYPE html><html><body><h1>404 - Page not found</h1><a href="/">Go home</a></body></html>
EOF
ls -R ~/mysite
```

7.4 Step 2 (common): ways to upload your own files

Option A — scp from your laptop (run on your laptop, not on the server):

```
# Linux / macOS / Windows PowerShell - upload whole folder to the server's home directory
scp -i mykey.pem -r ./mysite ec2-user@<PUBLIC_IP>:~/
# Ubuntu server:
scp -i mykey.pem -r ./mysite ubuntu@<PUBLIC_IP>:~/
```

Option B — git clone (run on the server). Push your site to a GitHub repository first:

```
sudo yum install -y git          # AL2023 / CentOS  (Ubuntu: sudo apt install -y git)
git clone https://github.com/<your-username>/<your-repo>.git ~/mysite
# later, to update:
cd ~/mysite && git pull
```

Option C — WinSCP / MobaXterm / FileZilla (SFTP): connect with host `<PUBLIC_IP>`, port 22, your user and private key, then drag and drop.

> **✔ Deploying directly into the web root**
>
> You cannot `scp` directly into `/var/www/...` because it is owned by root. Either upload to your home folder and `sudo cp` (as shown below), or make your login user the owner of the site folder once: `sudo chown -R $USER:$USER /var/www/mysite`. Files stay world-readable (`644`), so the web server can still read them.

7.5 Step 3 (common): copy files and set permissions

```
sudo mkdir -p /var/www/mysite
sudo cp -r ~/mysite/* /var/www/mysite/
sudo find /var/www/mysite -type d -exec chmod 755 {} \;
sudo find /var/www/mysite -type f -exec chmod 644 {} \;
ls -la /var/www/mysite
```

Now follow the section for your combination.

Ravindra's Tip

Before you blame the web server, test from inside the server with `curl -I http://localhost`. If that works but the browser doesn't, the problem is outside (security group, firewall, IP, `https`). If it fails inside too, check the service and error log. This one test cuts your troubleshooting time in half.

7.6 Combination 1 — Amazon Linux 2023 + Nginx

```
sudo yum install -y nginx
sudo service nginx start
sudo systemctl enable nginx

sudo tee /etc/nginx/conf.d/mysite.conf > /dev/null <<'EOF'
server {
    listen 80;
    listen [::]:80;
    server_name YOUR_SERVER_NAME;
    root /var/www/mysite;
    index index.html;

    location / {
        try_files $uri $uri/ =404;
    }
    error_page 404 /404.html;

    access_log /var/log/nginx/mysite_access.log;
    error_log  /var/log/nginx/mysite_error.log;
}
EOF

# put this server's public IP (or your domain) as server_name
sudo sed -i "s/YOUR_SERVER_NAME/$(curl -s https://checkip.amazonaws.com)/" /etc/nginx/conf.d/
mysite.conf
sudo nginx -t && sudo service nginx reload
curl -s http://localhost -H "Host: $(curl -s https://checkip.amazonaws.com)" | head -5
```

Open `http://<PUBLIC_IP>` in your browser.

i Why server_name = public IP here?

Amazon Linux and CentOS already have a default `server { server_name _; }` block inside `/etc/nginx/nginx.conf` that serves `/usr/share/nginx/html`. Giving our block the exact public IP (or a domain like `mysite.example.com`) makes Nginx pick our block when the browser asks for that host. If you later attach an Elastic IP or a domain, update `server_name` and reload. **Quick method instead:** simply copy your files into `/usr/share/nginx/html/` — no config needed.

7.7 Combination 2 — Amazon Linux 2023 + Apache

```
sudo service nginx stop 2>/dev/null     # free port 80 if Nginx was installed
sudo systemctl disable nginx 2>/dev/null
sudo yum install -y httpd
sudo service httpd start
sudo systemctl enable httpd
```

```
sudo tee /etc/httpd/conf.d/mysite.conf > /dev/null <<'EOF'
<VirtualHost *:80>
    ServerName mysite.local
    DocumentRoot /var/www/mysite
    <Directory /var/www/mysite>
        Options -Indexes +FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>
    ErrorDocument 404 /404.html
    ErrorLog  /var/log/httpd/mysite_error.log
    CustomLog /var/log/httpd/mysite_access.log combined
</VirtualHost>
EOF

sudo apachectl configtest && sudo service httpd reload
curl -s http://localhost | head -5
```

i How Apache picks the site

With name-based virtual hosts, the **first** `<VirtualHost *:80>` loaded is used for any request whose Host does not match another vhost. Since ours is the only vhost, it answers requests to the public IP too. Add `ServerAlias www.example.com` when you have a domain.

7.8 Combination 3 — Ubuntu + Nginx

```
sudo service apache2 stop 2>/dev/null
sudo systemctl disable apache2 2>/dev/null
sudo apt update && sudo apt install -y nginx

sudo tee /etc/nginx/sites-available/mysite > /dev/null <<'EOF'
server {
    listen 80 default_server;
    listen [::]:80 default_server;
    server_name _;
    root /var/www/mysite;
    index index.html;

    location / {
        try_files $uri $uri/ =404;
    }
    error_page 404 /404.html;

    access_log /var/log/nginx/mysite_access.log;
    error_log  /var/log/nginx/mysite_error.log;
}
EOF

sudo ln -s /etc/nginx/sites-available/mysite /etc/nginx/sites-enabled/mysite
sudo rm -f /etc/nginx/sites-enabled/default          # disable the Ubuntu welcome site
sudo chown -R www-data:www-data /var/www/mysite       # optional; 644/755 is enough to read
sudo nginx -t && sudo service nginx reload
curl -s http://localhost | head -5
```

7.9 Combination 4 — Ubuntu + Apache

```
sudo service nginx stop 2>/dev/null
sudo systemctl disable nginx 2>/dev/null
sudo apt update && sudo apt install -y apache2

sudo tee /etc/apache2/sites-available/mysite.conf > /dev/null <<'EOF'
<VirtualHost *:80>
    ServerName mysite.local
    DocumentRoot /var/www/mysite
    <Directory /var/www/mysite>
        Options -Indexes +FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>
    ErrorDocument 404 /404.html
    ErrorLog  ${APACHE_LOG_DIR}/mysite_error.log
    CustomLog ${APACHE_LOG_DIR}/mysite_access.log combined
</VirtualHost>
EOF

sudo a2ensite mysite.conf
sudo a2dissite 000-default.conf
sudo a2enmod rewrite                # useful for .htaccess rules later
sudo apache2ctl configtest && sudo service apache2 reload
curl -s http://localhost | head -5
```

7.10 Combination 5 — CentOS Stream 9 + Nginx

```
sudo yum install -y nginx
sudo service nginx start
sudo systemctl enable nginx

sudo tee /etc/nginx/conf.d/mysite.conf > /dev/null <<'EOF'
server {
    listen 80;
    listen [::]:80;
    server_name YOUR_SERVER_NAME;
    root /var/www/mysite;
    index index.html;
    location / {
        try_files $uri $uri/ =404;
    }
    error_page 404 /404.html;
}
EOF
sudo sed -i "s/YOUR_SERVER_NAME/$(curl -s https://checkip.amazonaws.com)/" /etc/nginx/conf.d/
mysite.conf

# SELinux: give the files the web-content label
sudo restorecon -Rv /var/www/mysite
ls -Z /var/www/mysite            # should show httpd_sys_content_t

# firewalld: only if it is active
if systemctl is-active --quiet firewalld; then
  sudo firewall-cmd --permanent --add-service=http --add-service=https
  sudo firewall-cmd --reload
fi
```

```
sudo nginx -t && sudo service nginx reload
curl -s http://localhost -H "Host: $(curl -s https://checkip.amazonaws.com)" | head -5
```

> ⚠ **SELinux and custom folders**
>
> `/var/www/...` automatically gets the correct label, but a folder elsewhere (e.g. `/data/site` or `/home/ec2-user/site`) does not. For such folders add a rule, then relabel: `sudo semanage fcontext -a -t httpd_sys_content_t "/data/site(/.*)?"` and `sudo restorecon -Rv /data/site` (the `semanage` command comes from package `policycoreutils-python-utils`). Files **moved** with `mv` keep their old label — always run `restorecon` after moving.

7.11 Combination 6 — CentOS Stream 9 + Apache

```
sudo service nginx stop 2>/dev/null
sudo systemctl disable nginx 2>/dev/null
sudo yum install -y httpd
sudo service httpd start
sudo systemctl enable httpd

sudo tee /etc/httpd/conf.d/mysite.conf > /dev/null <<'EOF'
<VirtualHost *:80>
    ServerName mysite.local
    DocumentRoot /var/www/mysite
    <Directory /var/www/mysite>
        Options -Indexes +FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>
    ErrorDocument 404 /404.html
    ErrorLog  /var/log/httpd/mysite_error.log
    CustomLog /var/log/httpd/mysite_access.log combined
</VirtualHost>
EOF

sudo restorecon -Rv /var/www/mysite
if systemctl is-active --quiet firewalld; then
  sudo firewall-cmd --permanent --add-service=http --add-service=https
  sudo firewall-cmd --reload
fi
sudo apachectl configtest && sudo service httpd reload
curl -s http://localhost | head -5
```

7.12 Hosting multiple sites on one server (virtual hosts)

Think of one building with many shops: the building has one address (IP), but each shop has its own name board (domain). Technically, the browser sends the domain in the HTTP `Host` header, and the web server picks the matching `server_name` / `ServerName` block.

One server can host many websites; the web server chooses the site by the **Host** header (domain name) sent by the browser.

```
site1.example.com ─┐                          ┌─► /var/www/site1
                   ├─► 13.233.10.25:80 ──────┤
site2.example.com ─┘   (one IP, one port)     └─► /var/www/site2
```

```
# Nginx: one server block per site
server { listen 80; server_name site1.example.com; root /var/www/site1; }
server { listen 80; server_name site2.example.com; root /var/www/site2; }
```

```
# Apache: one VirtualHost per site
<VirtualHost *:80>
    ServerName site1.example.com
    DocumentRoot /var/www/site1
</VirtualHost>
<VirtualHost *:80>
    ServerName site2.example.com
    DocumentRoot /var/www/site2
</VirtualHost>
```

Point both domains' DNS **A records** to the server's Elastic IP (allocate and attach it as in section 5.5; details in section 14.1). Buying a domain and creating these records step by step is covered in **Chapter 9**. Without real domains you can test from your laptop by adding a line `13.233.10.25  site1.example.com` to your `hosts` file (`/etc/hosts` on Linux/Mac, `C:\Windows\System32\drivers\etc\hosts` on Windows), or with `curl -H "Host: site1.example.com" http://13.233.10.25`.

> **Ravindra's Tip**
>
> For projects you show in interviews or put on your resume, spend a little on a domain name and add HTTPS with certbot. A link like `https://yourname.dev` looks far more professional than `http://13.233.x.x`. Just remember to attach an Elastic IP first so the DNS record never breaks.

7.13 Adding HTTPS (optional, needs a domain)

Browsers mark plain HTTP as "Not secure". Free TLS certificates are available from **Let's Encrypt** via `certbot`. You need a domain name whose A record points to your Elastic IP (see Chapter 9) and port 443 open.

```
# Ubuntu
sudo apt install -y certbot python3-certbot-nginx      # or python3-certbot-apache
sudo certbot --nginx -d mysite.example.com             # or --apache

# Amazon Linux 2023
sudo yum install -y certbot python3-certbot-nginx      # or python3-certbot-apache
sudo certbot --nginx -d mysite.example.com

# CentOS Stream 9 (certbot comes from EPEL)
sudo yum install -y epel-release
sudo yum install -y certbot python3-certbot-nginx
sudo certbot --nginx -d mysite.example.com
```

> **i Amazon Linux 2023: if the certbot package is not found**
>
> Install certbot in a Python virtual environment instead: `sudo python3 -m venv /opt/certbot && sudo /opt/certbot/bin/pip install certbot certbot-nginx && sudo ln -s /opt/certbot/bin/certbot /usr/bin/certbot` (use `certbot-apache` for Apache).

Certbot edits the server block, installs the certificate and sets up automatic renewal (check with `sudo certbot renew --dry-run`).

7.14 Troubleshooting static sites

Kalji karu naka if you see errors here — every student sees them. What matters is fixing them calmly, one check at a time.

Symptom	Likely cause	Fix
Browser spins then **timeout**	Security group missing port 80; firewalld blocking; no public IP	Add inbound HTTP 80 from `0.0.0.0/0`; `firewall-cmd --add-service=http`; check public IP
Connection refused	Web server not running / listening on another port	`sudo service nginx status`; `sudo ss -tlnp`
Still shows **default welcome page**	Your vhost not loaded or another block wins	Ubuntu: remove `sites-enabled/default`; AL/CentOS: check `server_name`; `sudo nginx -T` shows full config
403 Forbidden	No `index.html`; wrong permissions; SELinux label; `Require` missing	`ls -la`; `chmod 755` dirs / `644` files; `restorecon -Rv`; `namei -l /var/www/mysite/index.html`
404 Not Found	Wrong `root` / `DocumentRoot`; filename case (`Index.html`)	Check config path; `ls` exact names; check error log
CSS/images not loading	Wrong relative path; file not uploaded	Open browser DevTools (F12) → Network tab
502 Bad Gateway	(Reverse proxy only) backend app down	See Chapter 10
Config change has no effect	Forgot to reload; syntax error	`sudo nginx -t && sudo service nginx reload`
Site broke after stop/start	Public IP changed	Use Elastic IP; update `server_name`
`nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address in use)`	Apache (or another Nginx) already on port 80	`sudo ss -tlnp | grep :80`; stop the other server

Golden troubleshooting commands:

```
curl -I http://localhost                    # does it work from inside the server?
sudo ss -tlnp | grep -E ':80|:443'          # who is listening?
sudo tail -n 30 /var/log/nginx/error.log    # Nginx errors
sudo tail -n 30 /var/log/httpd/error_log    # Apache errors (AL2023 / CentOS)
sudo tail -n 30 /var/log/apache2/error.log  # Apache errors (Ubuntu)
namei -l /var/www/mysite/index.html         # permissions of every folder in the path
```

✓ Inside works, outside doesn't?

If `curl http://localhost` works on the server but the browser cannot open `http://<PUBLIC_IP>`, the problem is **outside** the web server: security group, firewalld, NACL, public IP, or you typed `https`.

Common Mistakes I See Students Make

- **Uploading files to the home folder and forgetting to copy them** into the document root.
- **Using `mv` from the home folder on CentOS** → files keep the wrong SELinux label → 403. Run `restorecon -Rv`.
- **Naming the file `Index.html` or `index.htm`** when the config expects `index.html`.
- **Leaving the Ubuntu `default` site enabled**, so the welcome page keeps appearing.
- **Wrong `root` path** — a missing or extra folder level (e.g. `/var/www/mysite/mysite`).
- **Not opening port 80 in the security group**, or typing `https://` without a certificate.
- **Uploading with Windows tools into a root-owned folder** and getting Permission denied — upload to home, then `sudo cp`.

Interview Questions

Q1. What is a document root?

The directory from which the web server serves files; the URL path is appended to it to find the file.

Q2. How does one server host multiple websites on the same IP and port?

Name-based virtual hosting: the web server reads the HTTP `Host` header and chooses the matching `server_name` (Nginx) or `ServerName` (Apache) block.

Q3. A static site returns 403 Forbidden. How do you troubleshoot?

Check an index file exists, permissions along the whole path (`namei -l`), ownership, SELinux labels (`ls -Z`, `restorecon`), and the `Require all granted` / `root` directives; read the error log.

Q4. What is the difference between `sites-available` and `sites-enabled` on Ubuntu?

`sites-available` holds all site configs; only those symlinked into `sites-enabled` (via `ln -s` or `a2ensite`) are loaded.

Q5. How do you upload a local folder to an EC2 instance?

`scp -i key.pem -r ./site user@IP:~/`, `rsync -avz`, `git clone` on the server, or SFTP clients like WinSCP/MobaXterm.

Q6. How do you add HTTPS to a site on EC2?

Point a domain to the Elastic IP, open port 443, install certbot and run `certbot --nginx -d domain` (or `--apache`); it installs a Let's Encrypt certificate and auto-renewal.

Practice Questions and Lab Exercises

1. What is a static website? Give three examples.
2. Write the document root, site config location and web-server user for all six combinations.
3. Why do we run `restorecon` on CentOS? What happens to SELinux labels when files are moved with `mv`?

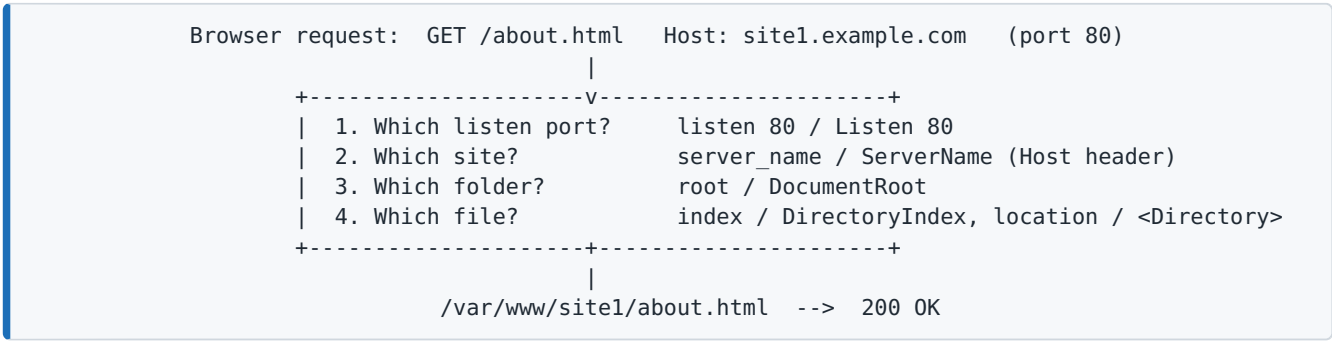
4. Write an Nginx server block and an Apache virtual host for `college.example.com` with root `/var/www/college`.
5. A site gives 403 Forbidden. List four possible causes and how you would check each.
6. **Lab:** Host your personal portfolio on Ubuntu + Nginx using `git clone`.
7. **Lab:** Host the same site on CentOS Stream 9 + Apache. Deliberately `mv` a file from your home folder into the site and observe the 403; fix it with `restorecon`.
8. **Lab:** Host two sites on one server using name-based virtual hosts and test with `curl -H "Host: ..."`.

★ Quick Revision

- Static site = files only. URL path maps to files under the document root.
- Steps: create/upload files → copy to `/var/www/mysite` → `755` dirs / `644` files → vhost/server block → test config → reload → browse.
- Nginx: `conf.d/*.conf` (AL/CentOS) or `sites-available` + symlink (Ubuntu). Apache: `conf.d/*.conf` (AL/CentOS) or `a2ensite` (Ubuntu).
- CentOS extras: `restorecon -Rv`, firewalld `--add-service=http`.
- Troubleshoot: `curl -I localhost`, `ss -tlnp`, error logs, security group.
- Next: Chapter 8 explains every config line (root, ports, vhosts); Chapter 9 gives the site a real domain.

Chapter 8: Configuring Nginx and Apache

Mitranno, in Chapter 6 we installed Nginx and Apache, and in Chapter 7 we hosted our first sites by copy-pasting a server block. That was enough to get started, but in real projects you will be asked to change things. Move the website to another folder, run it on another port, host three sites on one server, redirect `www` or hide the version number. In this chapter we open the configuration files and understand them line by line. Once you can read a config file comfortably, you'll handle web servers with confidence. Chala, suru karuya!



Every web server answers these four questions for every request. Once you know **where** each answer is written, you can configure any site.

8.1 Where the configuration lives

Item	Nginx (AL2023 / CentOS)	Nginx (Ubuntu)	Apache (AL2023 / CentOS)	Apache (Ubuntu)
Main file	`/etc/nginx/nginx.conf`	`/etc/nginx/nginx.conf`	`/etc/httpd/conf/httpd.conf`	`/etc/apache2/apache2.conf`
Your site files	`/etc/nginx/conf.d/*.conf`	`/etc/nginx/sites-available/`	`/etc/httpd/conf.d/*.conf`	`/etc/apache2/sites-available/*.conf`
How a site is enabled	Any `.conf` in `conf.d` is loaded	Symlink into `sites-enabled/`	Any `.conf` in `conf.d` is loaded	`sudo a2ensite name.conf`
Default site	`server` block inside `nginx.conf`	`sites-enabled/default`	`welcome.conf` + `/var/www/html`	`000-default.conf`
Default web root	`/usr/share/nginx/html`	`/var/www/html`	`/var/www/html`	`/var/www/html`
Ports file	inside each `server`	inside each `server`	`Listen` in `httpd.conf`	`/etc/apache2/ports.conf`
Worker user	`nginx`	`www-data`	`apache`	`www-data`
Logs	`/var/log/nginx/`	`/var/log/nginx/`	`/var/log/httpd/`	`/var/log/apache2/`
Test config	`sudo nginx -t`	`sudo nginx -t`	`sudo apachectl configtest`	`sudo apache2ctl configtest`

Item	Nginx (AL2023 / CentOS)	Nginx (Ubuntu)	Apache (AL2023 / CentOS)	Apache (Ubuntu)
Show loaded sites	`sudo nginx -T`	`sudo nginx -T`	`sudo httpd -S`	`sudo apache2ctl -S`

Ravindra's Tip

Never edit the main file (`nginx.conf`, `httpd.conf`, `apache2.conf`) for a website. Put each site in its **own small file** in `conf.d` or `sites-available`. Then you can copy one site to another server, disable a site by renaming one file, and an OS update won't overwrite your work. Ekdum clean setup.

✓ Always back up before editing

`sudo cp /etc/nginx/conf.d/mysite.conf /etc/nginx/conf.d/mysite.conf.bak` takes one second. On AL2023/CentOS, a backup must **not** end in `.conf`, or it will be loaded as a second site. That's why we name it `.bak`.

8.2 Reading an Nginx configuration

Nginx configuration is made of **directives** (`name value;`) and **blocks** (`name { ... }`), also called contexts:

```
main context  (user, worker_processes, error_log)
 ├── events { worker_connections ... }
 └── http {  (mime types, logging, gzip, include conf.d/*.conf)
        └── server {            ← one per website (a "server block")
              listen ...; server_name ...; root ...;
              └── location /path { ... }   ← rules for part of the URL
           }
     }
```

A fully commented server block:

```
# /etc/nginx/conf.d/mysite.conf   (Ubuntu: /etc/nginx/sites-available/mysite)
server {
    listen 80;                          # IPv4 port
    listen [::]:80;                     # IPv6 port
    server_name example.com www.example.com;   # which Host names this block answers

    root  /var/www/mysite;              # document root: URL "/" maps here
    index index.html index.htm;         # file served for a folder request

    location / {
        try_files $uri $uri/ =404;      # file? folder? otherwise 404
    }

    location /images/ {
        expires 7d;                     # let browsers cache images for 7 days
    }

    error_page 404 /404.html;           # custom error page (file inside root)
    autoindex off;                      # never list folder contents

    access_log /var/log/nginx/mysite_access.log;
```

```
    error_log  /var/log/nginx/mysite_error.log;
}
```

Directive	What it does	Example
`listen`	Port (and optional IP) to accept connections on; `default_server` makes this the fallback	`listen 80 default_server;`
`server_name`	Names matched against the `Host` header; `_` is a "match nothing" placeholder	`server_name example.com www.example.com;`
`root`	Folder that URL paths are appended to	`root /var/www/mysite;`
`index`	File(s) to try for a folder URL	`index index.html index.php;`
`location`	Rules for matching URL paths	`location /api/ { ... }`
`try_files`	Try files in order, then fall back	`try_files $uri $uri/ /index.html;` (SPA)
`return`	Send a redirect or status immediately	`return 301 https://$host$request_uri;`
`error_page`	Custom error page	`error_page 404 /404.html;`
`client_max_body_size`	Maximum upload size (default 1 MB)	`client_max_body_size 20M;`
`server_tokens`	Show/hide Nginx version in headers and error pages	`server_tokens off;`

> ⚠ **The two classic Nginx syntax errors**
>
> Every directive ends with a **semicolon** `;`, and every `{` needs a matching `}`. `sudo nginx -t` prints the exact file and line number, e.g. `unexpected "}" in /etc/nginx/conf.d/mysite.conf:9`. Read it slowly. It is almost always the line before the one reported.

How Nginx chooses the server block

1. It looks at blocks whose `listen` matches the port of the request.
2. Among those, it picks the block whose `server_name` matches the `Host` header exactly (then wildcard names like `*.example.com`).
3. If nothing matches, it uses the block marked `default_server` for that port, or the **first** block loaded if none is marked.

That rule explains the most common complaint: "I configured my site but I still see the welcome page." Some other block won, because it is the default or because your `server_name` doesn't match what the browser sent.

8.3 Task: change the Nginx document root

Suppose the website must be served from `/var/www/portfolio` instead of the default folder.

```
# 1. create the folder and a test page
sudo mkdir -p /var/www/portfolio
echo '<h1>Portfolio served from /var/www/portfolio</h1>' | sudo tee /var/www/portfolio/index.html
```

```
sudo chmod 755 /var/www/portfolio
sudo chmod 644 /var/www/portfolio/index.html
```

Amazon Linux 2023 / CentOS Stream 9: create a site file (the built-in block inside `nginx.conf` still exists, so we check which block is the default):

```
sudo tee /etc/nginx/conf.d/portfolio.conf > /dev/null <<'__EOCONF__'
server {
    listen 80;
    listen [::]:80;
    server_name _;
    root /var/www/portfolio;
    index index.html;
    location / { try_files $uri $uri/ =404; }
}
__EOCONF__
grep -n "default_server" /etc/nginx/nginx.conf      # if found, remove "default_server" there
sudo nginx -t && sudo service nginx reload
```

i Why grep for default_server?

In `/etc/nginx/nginx.conf` the line `include /etc/nginx/conf.d/*.conf;` comes **before** the built-in `server { ... }` block. So if no block says `default_server`, your `conf.d` block loads first and wins for unknown names. But some package versions mark the built-in block `listen 80 default_server;`, and then it wins instead. The fix is either to put your real domain or public IP in `server_name` (as in Chapter 7), or to move `default_server` to your own block.

Ubuntu: edit the site file and change one line:

```
sudo sed -i 's#root /var/www/html;#root /var/www/portfolio;#' /etc/nginx/sites-available/default
grep -n "root" /etc/nginx/sites-available/default
sudo nginx -t && sudo service nginx reload
```

CentOS Stream 9 only: SELinux. Folders under `/var/www` automatically get the correct `httpd_sys_content_t` label. For any **other** path, such as `/srv/portfolio` or `/data/site`, you must teach SELinux about it, or you'll get **403 Forbidden**:

```
sudo yum install -y policycoreutils-python-utils          # provides semanage
sudo semanage fcontext -a -t httpd_sys_content_t "/srv/portfolio(/.*)?"
sudo restorecon -Rv /srv/portfolio
ls -Z /srv/portfolio                                       # should show httpd_sys_content_t
```

Verify with `curl -s http://localhost | head -3` and then from your browser.

✖ Never point the root at /root or /home/ec2-user

The web server runs as `nginx` / `www-data` / `apache`, which can't read inside home folders (permissions `700`/`750`). Changing folder permissions on your home to "fix" a 403 exposes your SSH keys and files. Keep websites under `/var/www` (or `/srv`) and copy files there.

8.4 Reading an Apache configuration

Apache uses **directives** (one per line, no semicolon) and **sections** in angle brackets:

```
# /etc/httpd/conf.d/mysite.conf   (Ubuntu: /etc/apache2/sites-available/mysite.conf)
<VirtualHost *:80>
    ServerName  example.com
    ServerAlias www.example.com
    ServerAdmin webmaster@example.com

    DocumentRoot /var/www/mysite
    DirectoryIndex index.html index.php

    <Directory /var/www/mysite>
        Options -Indexes +FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>

    ErrorDocument 404 /404.html
    ErrorLog  /var/log/httpd/mysite_error.log
    CustomLog /var/log/httpd/mysite_access.log combined
</VirtualHost>
```

(On Ubuntu use `${APACHE_LOG_DIR}/mysite_error.log`, which points to `/var/log/apache2/`.)

Directive	What it does
`<VirtualHost *:80>`	A site listening on port 80 of all IPs
`ServerName` / `ServerAlias`	Main host name / extra names for this site
`DocumentRoot`	Folder that URL paths map to
`DirectoryIndex`	File served for a folder request
`<Directory path>`	Rules for a folder on disk
`Options -Indexes`	Disable automatic folder listing
`AllowOverride All`	Allow `.htaccess` files (needed by WordPress permalinks)
`Require all granted`	Allow everyone to access this folder (Apache 2.4 syntax)
`ErrorLog` / `CustomLog`	Per-site log files
`Listen`	Ports Apache binds to (in `httpd.conf` or `ports.conf`)
`ServerTokens Prod` + `ServerSignature Off`	Hide version details

> ⚠ **No inline comments in Apache**
>
> `Require all granted # allow all` is an error in Apache. Comments must be on their own line starting with `#`.

How Apache chooses the virtual host

For a given IP:port, Apache compares the `Host` header with every `ServerName`/`ServerAlias`. If none matches, the **first** virtual host loaded for that port answers, and files load in alphabetical order. That's why Ubuntu's default site is called **`000-default.conf`**: it sorts first. `sudo apachectl -S` (Ubuntu: `sudo apache2ctl -S`) prints this list, showing which vhost is the default:

```
*:80   is a NameVirtualHost
       default server example.com (/etc/httpd/conf.d/mysite.conf:1)
```

```
        port 80 namevhost example.com (/etc/httpd/conf.d/mysite.conf:1)
                alias www.example.com
```

8.5 Task: change the Apache DocumentRoot

Option A: edit the virtual host (recommended). Change `DocumentRoot` **and** the matching `<Directory>` path together:

```
# Ubuntu: default site
sudo sed -i 's#/var/www/html#/var/www/portfolio#' /etc/apache2/sites-available/000-default.conf
sudo tee /etc/apache2/conf-available/portfolio-dir.conf > /dev/null <<'__EOCONF__'
<Directory /var/www/portfolio>
    Options -Indexes +FollowSymLinks
    AllowOverride All
    Require all granted
</Directory>
__EOCONF__
sudo a2enconf portfolio-dir
sudo apache2ctl configtest && sudo service apache2 reload
```

```
# Amazon Linux 2023 / CentOS Stream 9: own virtual host
sudo tee /etc/httpd/conf.d/portfolio.conf > /dev/null <<'__EOCONF__'
<VirtualHost *:80>
    ServerName portfolio.local
    DocumentRoot /var/www/portfolio
    <Directory /var/www/portfolio>
        Options -Indexes +FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>
</VirtualHost>
__EOCONF__
sudo apachectl configtest && sudo service httpd reload
```

Option B: change the global default in `/etc/httpd/conf/httpd.conf`, which has both `DocumentRoot "/var/www/html"` and `<Directory "/var/www/html">`. Change both, then test and reload. It works, but Option A is cleaner.

> **i Ubuntu security default**
>
> `/etc/apache2/apache2.conf` denies access to `/` and allows only `/var/www` and `/usr/share`. A DocumentRoot outside these (e.g. `/srv/site`) needs its own `<Directory>` block with `Require all granted`. Otherwise you get **403 Forbidden** even with perfect file permissions.

> **RB Ravindra's Tip**
>
> When a student shows me a 403, I ask three questions in this order: Does the file exist and have 644/755 along the whole path? (`namei -l`), Does a `<Directory>` / *root* point to exactly this path?, and on CentOS, Is the SELinux label right? (`ls -Z`). Almost every 403 is answered by one of these three. Samjla ka?

8.6 Task: run a site on a different port

Sometimes a second site or an admin panel must run on another port, e.g. **8081**.

```
# Nginx: /etc/nginx/conf.d/admin.conf   (Ubuntu: sites-available + symlink)
server {
    listen 8081;
    server_name _;
    root /var/www/admin;
    index index.html;
}
```

```
# Apache (AL2023/CentOS): add to /etc/httpd/conf.d/admin.conf
Listen 8081
<VirtualHost *:8081>
    DocumentRoot /var/www/admin
    <Directory /var/www/admin>
        Require all granted
    </Directory>
</VirtualHost>
```

On Ubuntu Apache, put `Listen 8081` in `/etc/apache2/ports.conf` (not in the vhost file). Then:

1. Test and reload the web server.
2. Check that it listens: `sudo ss -tlnp | grep 8081`.
3. **Security group:** add an inbound rule Custom TCP 8081 (from My IP for admin pages).
4. **CentOS Stream 9:** open firewalld if active (`sudo firewall-cmd --permanent --add-port=8081/tcp && sudo firewall-cmd --reload`) and allow the port in SELinux:

```
sudo semanage port -l | grep -w http_port_t              # ports web servers may bind to
sudo semanage port -a -t http_port_t -p tcp 8081          # add 8081 if not listed
```

Without the SELinux step, the service fails to start with `bind() to 0.0.0.0:8081 failed (13: Permission denied)`.

8.7 Multiple sites with name-based virtual hosts

In section 7.12 we saw the idea. Now let's build it properly: `site1.example.com` and `site2.example.com` on one server, plus a **catch-all default** that answers requests using the raw IP.

```
for s in site1 site2; do
  sudo mkdir -p /var/www/$s
  echo "<h1>Welcome to $s</h1>" | sudo tee /var/www/$s/index.html
done
```

Nginx (AL2023/CentOS: files in `conf.d/`; Ubuntu: files in `sites-available/` plus a symlink for each):

```
# site1.conf
server {
    listen 80;
    server_name site1.example.com;
```

```
    root /var/www/site1;
    access_log /var/log/nginx/site1_access.log;
}

# site2.conf
server {
    listen 80;
    server_name site2.example.com;
    root /var/www/site2;
    access_log /var/log/nginx/site2_access.log;
}

# 00-default.conf: requests by IP or unknown names
server {
    listen 80 default_server;
    server_name _;
    return 444;          # close connection (or: return 301 http://site1.example.com;)
}
```

```
# Ubuntu only: enable and disable
sudo ln -s /etc/nginx/sites-available/site1.conf /etc/nginx/sites-enabled/
sudo ln -s /etc/nginx/sites-available/site2.conf /etc/nginx/sites-enabled/
sudo rm -f /etc/nginx/sites-enabled/default
```

Apache:

```
# 00-default.conf  (loads first, so it is the fallback)
<VirtualHost *:80>
    ServerName default.invalid
    DocumentRoot /var/www/html
</VirtualHost>

# site1.conf
<VirtualHost *:80>
    ServerName site1.example.com
    DocumentRoot /var/www/site1
    ErrorLog  /var/log/httpd/site1_error.log
</VirtualHost>

# site2.conf
<VirtualHost *:80>
    ServerName site2.example.com
    DocumentRoot /var/www/site2
    ErrorLog  /var/log/httpd/site2_error.log
</VirtualHost>
```

```
# Ubuntu Apache: enable sites (log path ${APACHE_LOG_DIR})
sudo a2ensite site1.conf site2.conf
sudo a2dissite 000-default.conf          # optional
sudo apache2ctl configtest && sudo service apache2 reload
```

Test without buying domains:

```
curl -H "Host: site1.example.com" http://localhost        # Welcome to site1
curl -H "Host: site2.example.com" http://localhost        # Welcome to site2
curl -I http://localhost                                  # default block answers
```

From your laptop, add both names to your `hosts` file pointing to the server's Elastic IP, then open them in the browser. In Chapter 9 we'll replace this trick with real DNS records.

8.8 Useful everyday configurations

Redirect www to the main domain (or the reverse)

```
server {
    listen 80;
    server_name www.example.com;
    return 301 http://example.com$request_uri;
}
```

```
<VirtualHost *:80>
    ServerName www.example.com
    Redirect permanent / http://example.com/
</VirtualHost>
```

Redirect HTTP to HTTPS

Certbot adds this automatically. By hand, it's `return 301 https://$host$request_uri;` (Nginx) or `Redirect permanent / https://example.com/` (Apache) in the port-80 block.

Hide version numbers

```
# Nginx: inside http { } of nginx.conf
server_tokens off;
# Apache (AL2023/CentOS): new file /etc/httpd/conf.d/security.conf
ServerTokens Prod
ServerSignature Off
# Apache (Ubuntu): edit /etc/apache2/conf-available/security.conf (already enabled)
```

Check with `curl -I http://localhost`. The `Server:` header should show only `nginx` or `Apache`.

Increase the upload limit

`client_max_body_size 20M;` (Nginx, in `http` or `server`). For PHP apps, also raise `upload_max_filesize` and `post_max_size` in `php.ini`.

Password-protect a folder (basic auth)

```
sudo yum install -y httpd-tools            # Ubuntu: sudo apt install -y apache2-utils
sudo htpasswd -c /etc/nginx/.htpasswd admin
```

```
location /admin/ {
    auth_basic "Restricted";
    auth_basic_user_file /etc/nginx/.htpasswd;
}
```

(Apache equivalent inside `<Directory>`: `AuthType Basic`, `AuthName "Restricted"`, `AuthUserFile /etc/httpd/.htpasswd`, `Require valid-user`.) Use this only together with HTTPS, because basic auth passwords are only base64-encoded.

Enable compression

```
# Nginx, inside http { }
gzip on;
gzip_types text/css application/javascript application/json image/svg+xml;
```

Apache: `sudo a2enmod deflate` on Ubuntu. On AL2023/CentOS, `mod_deflate` is loaded by default, so just add `AddOutputFilterByType DEFLATE text/html text/css application/javascript`.

8.9 The safe change workflow

Dhyan do, this is the habit that separates a professional from a beginner:

```
sudo cp site.conf site.conf.bak                  # 1. backup (name must not end in .conf)
sudo nano /etc/nginx/conf.d/site.conf            # 2. edit
sudo nginx -t                                    # 3. test syntax
sudo service nginx reload                        # 4. reload (no downtime)
curl -I -H "Host: example.com" http://localhost  # 5. verify from inside
sudo tail -n 20 /var/log/nginx/error.log         # 6. check the log
```

For Apache, replace step 3 with `sudo apachectl configtest` (Ubuntu: `sudo apache2ctl configtest`) and step 4 with `sudo service httpd reload` (Ubuntu: `sudo service apache2 reload`). If something breaks, copy the `.bak` back, test and reload. You'll be online again in seconds.

> **✔ reload vs restart**
>
> `reload` re-reads the configuration while keeping existing connections, and refuses to apply a broken config. `restart` stops everything first. Use `restart` only when a module or port change needs it, or when reload doesn't pick up a change.

8.10 Troubleshooting configuration problems

Symptom	Likely cause	Fix
403 Forbidden after changing root	New folder outside `/var/www` without `<Directory>`/permissions; SELinux label wrong	`namei -l /path/index.html`; add `<Directory>` with `Require all granted`; CentOS: `semanage fcontext` + `restorecon -Rv`
Default welcome page instead of your site	Another block is `default_server` / loads first; `server_name` doesn't match	`sudo nginx -T | grep -E "server_name|listen"`; `sudo apachectl -S`; remove Ubuntu `default` / `000-default`
`nginx -t`: `unexpected "}"` or `directive is not terminated`	Missing `;` or `{`	Fix the line before the one reported
`Invalid command 'Require'` or `'#'`	Typo or inline comment in Apache	Put comments on their own line; check spelling
Service fails on new port: `Permission denied`	SELinux port not allowed (CentOS)	`semanage port -a -t http_port_t -p tcp PORT`

Symptom	Likely cause	Fix
New port not reachable from browser	Security group / firewalld not opened	Add inbound rule; `firewall-cmd --add-port`
`Address already in use`	Both Nginx and Apache on port 80	`sudo ss -tlnp | grep :80`; stop one
Changes have no effect	Not reloaded; edited a `.bak` or wrong file	`sudo nginx -T` shows what is really loaded
Downloads PHP file instead of running it	PHP handler not configured	See Chapter 10 (PHP-FPM)

Common Mistakes I See Students Make

- **Changing `DocumentRoot` but not the matching `<Directory>`**, then getting 403.
- **Saving backups as `site.conf.bak.conf`** or `site-old.conf` in `conf.d`, which loads the same site twice (conflicting server name warning).
- **Forgetting the symlink on Ubuntu Nginx**, or linking with a relative path that points nowhere (`ls -l sites-enabled` shows broken links in red).
- **Writing `server_name` with `http://`**, like `server_name http://example.com;`. Only the bare host name belongs there.
- **Pointing the web root at `/home/ec2-user/site`** and changing home permissions to 777.
- **Opening a new port in the config but not in the security group.**
- **Disabling SELinux** instead of using `semanage fcontext` / `semanage port`.
- **Restarting without `nginx -t`** and taking the site down with a typo.

Interview Questions

Q1. What is a server block in Nginx and a virtual host in Apache?

A configuration section that defines one website: the port it listens on, the host names it answers (`server_name` / `ServerName`), its document root and its rules and logs. Many can exist on one server.

Q2. How does Nginx decide which server block handles a request?

It matches the port in `listen`, then the `Host` header against `server_name` (exact, then wildcard, then regex). If nothing matches, it uses the `default_server` for that port, or the first block defined.

Q3. You changed the DocumentRoot and now get 403 Forbidden. What do you check?

File/folder permissions along the path (`namei -l`), a matching `<Directory>` with `Require all granted`, the index file, and on SELinux systems the context (`ls -Z`, fix with `semanage fcontext` + `restorecon`).

Q4. How do you run Apache on port 8081?

Add `Listen 8081` (Ubuntu: `ports.conf`), create `<VirtualHost *:8081>`, test and reload, open the port in the security group/firewall, and on CentOS allow it with `semanage port -a -t http_port_t -p tcp 8081`.

Q5. What is the difference between `reload` and `restart`?

Reload applies new configuration gracefully without dropping connections and keeps the old config if the new one is invalid. Restart stops and starts the service, briefly interrupting traffic.

Q6. How do you see which virtual hosts Apache has loaded?

`sudo apachectl -S` (Ubuntu: `sudo apache2ctl -S`). For Nginx, `sudo nginx -T` dumps the full loaded configuration.

Q7. How do you redirect `www.example.com` to `example.com`?

A separate block for `www.example.com` with `return 301 http://example.com$request_uri;` (Nginx) or `Redirect permanent / http://example.com/` (Apache).

Practice Questions and Lab Exercises

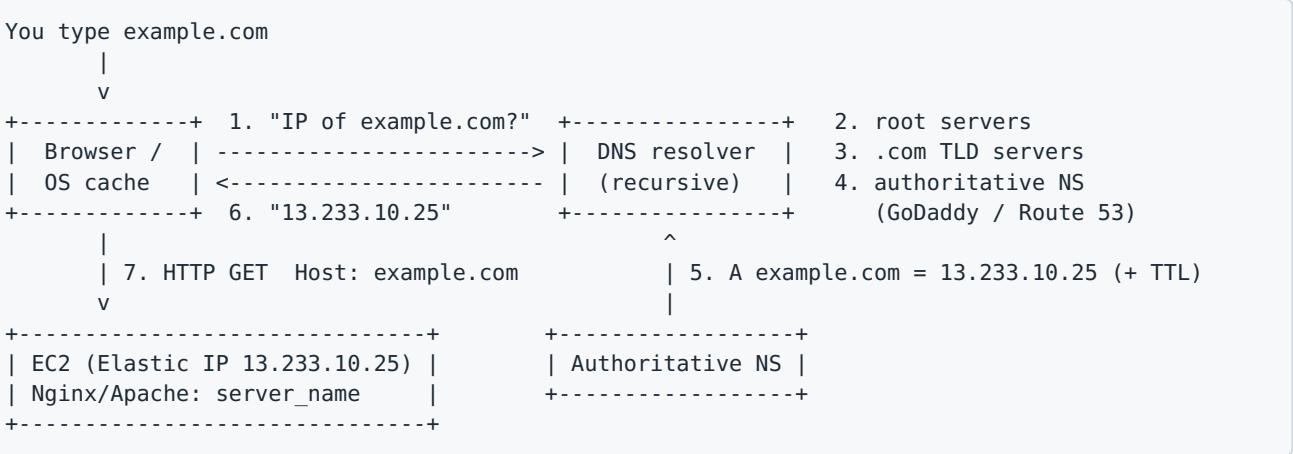
1. Draw the four questions a web server answers for every request and name the directive for each in Nginx and Apache.
2. Write the site-file location and enable method for Nginx and Apache on Ubuntu and on Amazon Linux 2023.
3. Why is Ubuntu's default Apache site named `000-default.conf`?
4. What does `Options -Indexes` do? What is its Nginx equivalent?
5. **Lab:** Move your Nginx site to `/var/www/portfolio`, then (on CentOS Stream 9) to `/srv/portfolio` using `semanage fcontext`.
6. **Lab:** Run a second site on port 8081 with Apache. Open it in the security group from My IP only.
7. **Lab:** Host `site1.example.com` and `site2.example.com` plus a catch-all default on one server. Verify all three with `curl -H "Host: ..."`.
8. **Lab:** Hide the server version and check the `Server:` header with `curl -I` before and after.
9. **Lab:** Deliberately remove a `;` from your server block and read the `nginx -t` error message.

★ Quick Revision

- Four questions per request: port (`listen` / `Listen`) → site (`server_name` / `ServerName`, Host header) → folder (`root` / `DocumentRoot`) → file (`index` / `DirectoryIndex`).
- Site files: `conf.d/*.conf` (AL2023/CentOS), `sites-available` + symlink or `a2ensite` (Ubuntu). Never edit main files for a site.
- Fallback site: Nginx `default_server` or first block; Apache first vhost loaded (`000-default.conf`). Inspect with `nginx -T` / `apachectl -S`.
- Change root: new folder 755/644 → update `root` / `DocumentRoot` **and** `<Directory>` → CentOS: `semanage fcontext` + `restorecon` for paths outside `/var/www`.
- New port: config + security group + firewalld + `semanage port` (CentOS).
- Workflow: backup (`.bak`) → edit → `nginx -t` / `apachectl configtest` → `sudo service ... reload` → `curl -I` → logs.

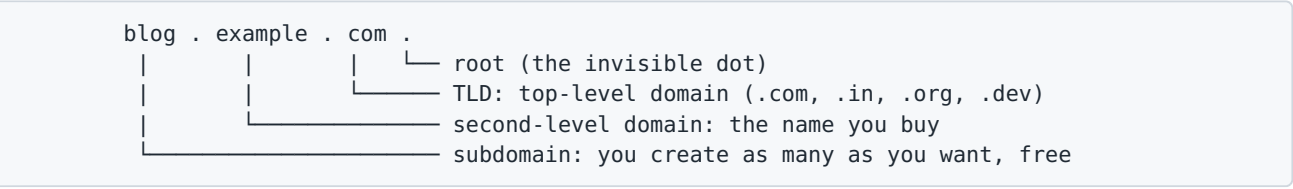
Chapter 9: Domain Names, DNS and Subdomains

Mitranno, so far we have opened our websites with `http://13.233.10.25`. It works, but nobody will remember that number, and you can't put it on a visiting card. In this chapter we give our server a real name like `example.com`, add `www`, create subdomains like `blog.example.com` and `api.example.com`, and understand what happens behind the scenes when someone types that name. We'll use **GoDaddy** as the registrar example because many of you already have an account there, but the steps are almost the same on Namecheap, Hostinger, BigRock or Route 53. Chala, apla server la nav deuya (let's give our server a name)!



9.1 What is a domain name?

A domain name is a human-friendly label for an IP address. Read it **from right to left**:



Term	Meaning	Example
TLD	Top-level domain, managed by a registry	`.com`, `.in`, `.org`, `.dev`, `.co.in`
Domain / apex / root domain	The name you register	`example.com` (in DNS consoles often written `@`)
Subdomain	Any name to the left of your domain	`www.example.com`, `blog.example.com`
FQDN	Fully qualified domain name, the complete name	`api.example.com.`
Registrar	Company that sells/renews domains for you	GoDaddy, Namecheap, Hostinger, Route 53 Domains

Term	Meaning	Example
Registry	Organisation that runs a TLD	Verisign (`.com`), NIXI (`.in`)
DNS hosting / name servers	Servers that store your records and answer queries	`ns51.domaincontrol.com` (GoDaddy), `ns-123.awsdns-15.com` (Route 53)
Web hosting	The machine that serves the website	Your EC2 instance

> **Ravindra's Tip**
>
> Keep three things separate in your head: **who you bought the name from** (registrar), **who answers DNS questions** (name servers), and **where the website runs** (EC2). They can be three different companies. Students mix them up and then search for "EC2 settings" inside GoDaddy. Once this picture is clear, DNS becomes very simple.

9.2 How DNS resolution works

1. The browser checks its own cache, then the OS cache and the `hosts` file.
2. If not found, it asks the **recursive resolver** (your ISP, a public resolver like `8.8.8.8` / `1.1.1.1`, or on EC2 the VPC resolver).
3. The resolver asks a **root server** → "ask the `.com` servers".
4. The `.com` **TLD server** → "the name servers for `example.com` are `ns51.domaincontrol.com`...". These NS entries come from what you set at the registrar.
5. The **authoritative name server** replies with the record, e.g. `A 13.233.10.25`, along with a **TTL**.
6. The resolver caches the answer for TTL seconds and gives it to the browser, which then connects to the IP and sends `Host: example.com`.

DNS normally uses **UDP port 53** (TCP 53 for large answers and zone transfers). See also section 2.4 on opening a website step by step.

9.3 DNS record types you must know

Record	Purpose	Example (name → value)
A	Name → IPv4 address	`@` → `13.233.10.25`
AAAA	Name → IPv6 address	`@` → `2406:da1a:...`
CNAME	Name → another name (alias)	`www` → `example.com`
MX	Mail servers for the domain (with priority)	`@` → `10 mail.example.com`
TXT	Free text, used for verification, SPF, DKIM	`@` → `"v=spf1 include:_spf.google.com ~all"`
NS	Which name servers are authoritative	`@` → `ns51.domaincontrol.com`
SOA	Start of authority: zone admin info and serials	created automatically
CAA	Which certificate authorities may issue certificates	`@` → `0 issue "letsencrypt.org"`

Record	Purpose	Example (name → value)
Alias (Route 53)	Like a CNAME but allowed at the apex, for AWS resources	`@` → load balancer / CloudFront

> ⚠ **CNAME rules**
>
> A CNAME **cannot** be used at the apex (`@`), and a name with a CNAME can't have any other record. For the root domain always use an **A** record (or a Route 53 Alias). Use CNAME for names like `www`.

TTL and "propagation"

TTL (Time To Live) is how many seconds resolvers may cache a record. With TTL `3600`, a resolver that looked up your old IP may keep using it for up to one hour after you change it. That delay is what people call **DNS propagation**. Nothing is actually pushed around the world; old cached answers simply expire.

- Before a planned change (e.g. moving to a new server), lower the TTL to `300` (5 minutes) a day earlier.
- After the change is stable, raise it again (e.g. `3600`) to reduce lookups.
- A **new** domain's name server change at the registrar can take longer, sometimes a few hours, because TLD servers and resolvers cache NS records with long TTLs.

9.4 Before you point a domain: prepare the server

1. **A working website on EC2**, tested with `curl -I http://localhost` and `http://<PUBLIC_IP>` (Chapters 7 and 8).
2. **An Elastic IP**, so the address never changes after stop/start. Quick steps:
   - EC2 console → **Network & Security → Elastic IPs → Allocate Elastic IP address → Allocate**.
   - Select it → **Actions → Associate Elastic IP address** → choose your instance → **Associate**.
   - Note the address, e.g. `13.233.10.25`. SSH now uses this IP too.

   (Charges, the default limit of 5 per region, moving an EIP to a replacement server and releasing it safely are covered in detail in **section 14.1**.) 3. **Security group:** inbound **HTTP 80** and **HTTPS 443** from `0.0.0.0/0` (and `::/0` for IPv6).

> ✖ **Never point DNS to the auto-assigned public IP**
>
> It changes on every stop/start. Your domain would then point to an IP that may soon belong to someone else's instance. Always use an Elastic IP (or a load balancer).

9.5 Buying a domain on GoDaddy

The GoDaddy website changes its look from time to time, so the button names below may vary slightly. The flow stays the same.

1. Go to godaddy.com and sign in (or create an account and verify your email/mobile).
2. Search for the name you want, e.g. `yourname-cloud`. Compare TLDs such as `.com`, `.in`, `.dev` and `.online`.
3. Add the domain to the cart. **Carefully review the add-ons:** email plans, website builders, SSL certificates and hosting are not needed for EC2 hosting (we'll use free Let's Encrypt certificates). Remove what you don't need.
4. Check the registration period and the **renewal** price, which is often higher than the first-term offer. Enable **domain privacy** if offered, to hide your personal contact details from public WHOIS.
5. Pay and complete the purchase. GoDaddy may send an email to **verify the registrant contact**. Click it, or the domain can be suspended.
6. Open **My Products** (or Domain Portfolio). Your new domain is listed there with its default GoDaddy name servers.

> **✔ Practising without spending?**
>
> You can learn most of this chapter with the `hosts` file trick from section 8.7. For real practice, cheap TLDs (such as `.online`, `.site` or `.xyz`) are often available at low first-term prices. Just turn off auto-renew if you don't want to keep them.

9.6 Pointing a GoDaddy domain to EC2 (GoDaddy DNS)

This is the simplest method: GoDaddy stays both registrar and DNS host.

1. **My Products → Domains →** click your domain → **DNS** (or Manage DNS).
2. In the **DNS Records** list, find the **A** record with name `@`. A new domain usually points it to GoDaddy's "Parked" page. Click **Edit** (pencil icon).
3. Set **Value / Points to** = your Elastic IP `13.233.10.25`, **TTL** = 600 seconds (or Custom → 600) → **Save**.
4. Find the **CNAME** record `www`. Make sure its value is `@` (meaning `example.com`). If it is missing, **Add New Record → CNAME, Name** `www`**, Value** `@`.
5. Delete any extra **A** records for `@` that still point to parking/forwarding IPs, and turn off **Forwarding** if it was enabled. Two different A records for the same name make visitors randomly land on the wrong server.
6. Leave **NS**, **SOA** and any **MX/TXT** records (email, verification) as they are.

The DNS table should look like this:

Type	Name	Value	TTL
A	`@`	`13.233.10.25`	600 seconds
CNAME	`www`	`@`	1 hour

Type	Name	Value	TTL
A	`blog`	`13.233.10.25`	600 seconds
NS	`@`	`ns51.domaincontrol.com` (do not edit)	1 hour

Usually the new A record works within minutes. Verify with `dig` (section 9.9) before blaming the web server.

> **i Using another registrar?**
>
> Namecheap: Domain List → Manage → Advanced DNS. Hostinger: Domains → DNS / Nameservers. The same records apply: **A** `@` → Elastic IP, **CNAME** `www` → root domain.

9.7 Alternative: GoDaddy registrar + Route 53 DNS

Companies running on AWS usually keep DNS in **Amazon Route 53**. It integrates with load balancers (Alias records), health checks and infrastructure as code. You can keep the domain registered at GoDaddy and only move DNS:

1. AWS console → **Route 53 → Hosted zones → Create hosted zone** → Domain name `example.com`, Type **Public hosted zone** → **Create**.
2. Route 53 creates **NS** and **SOA** records. Copy the **four name servers** (e.g. `ns-123.awsdns-15.com`, `ns-456.awsdns-57.net`, …).
3. **Create record →** Record name (empty) for the apex, type **A**, value `13.233.10.25`, TTL 300 → **Create records**. Add another with record name `www` (type A to the EIP, or CNAME to `example.com`).
4. At **GoDaddy: My Products → your domain → DNS → Nameservers → Change nameservers → "I'll use my own nameservers"** → enter the four Route 53 name servers → **Save** and confirm.
5. Wait for the NS change (minutes to a few hours). Check with `dig NS example.com +short`. The Route 53 names should appear.

> **⚠ Records now live only in Route 53**
>
> After changing name servers, the records in GoDaddy's DNS page are **ignored**. Recreate any MX/TXT (email, verification) records in Route 53 **before** switching, or email will stop. A Route 53 hosted zone has a small monthly charge plus per-query charges. Delete unused hosted zones.

You can also register domains directly in **Route 53 → Registered domains → Register domains**. Route 53 then creates the hosted zone for you automatically.

9.8 Configure the web server for the domain

DNS brings the visitor to your IP. The web server must now recognise the name.

```
# Nginx: /etc/nginx/conf.d/example.conf  (Ubuntu: sites-available/example + symlink)
server {
    listen 80;
```

```
    listen [::]:80;
    server_name example.com www.example.com;
    root /var/www/example;
    index index.html;
    location / { try_files $uri $uri/ =404; }
}
```

```
# Apache: /etc/httpd/conf.d/example.conf  (Ubuntu: sites-available/example.conf + a2ensite)
<VirtualHost *:80>
    ServerName  example.com
    ServerAlias www.example.com
    DocumentRoot /var/www/example
    <Directory /var/www/example>
        Require all granted
    </Directory>
</VirtualHost>
```

```
sudo mkdir -p /var/www/example && echo '<h1>example.com is live!</h1>' | sudo tee /var/www/
example/index.html
sudo nginx -t && sudo service nginx reload                  # Nginx
sudo apachectl configtest && sudo service httpd reload      # Apache AL2023/CentOS
sudo apache2ctl configtest && sudo service apache2 reload   # Apache Ubuntu
curl -I http://example.com
```

If you had put the public IP in `server_name` in Chapter 7, replace it with the domain names now.

9.9 Verify DNS: dig, nslookup and friends

```
dig example.com +short                  # A record → 13.233.10.25
dig www.example.com +short              # CNAME chain → example.com. → 13.233.10.25
dig example.com                         # full answer incl. TTL countdown
dig @8.8.8.8 example.com +short         # ask Google's public resolver
dig @1.1.1.1 example.com +short         # ask Cloudflare's resolver
dig NS example.com +short               # which name servers are authoritative?
dig MX example.com +short               # mail records
dig +trace example.com                  # follow root → TLD → authoritative
nslookup example.com                    # works on Windows, macOS and Linux
host example.com                        # short output (Linux)
curl -I http://example.com              # DNS + web server together
```

`dig` comes from **bind-utils** on Amazon Linux/CentOS (`sudo yum install -y bind-utils`) and **dnsutils** on Ubuntu (`sudo apt install -y dnsutils`).

Reading `dig` output:

```
;; ANSWER SECTION:
example.com.        587     IN      A       13.233.10.25
     name          TTL(s) class   type     value
```

Flush stale caches on your laptop if the server already shows the new IP but your browser still doesn't:

OS	Command
Windows	`ipconfig /flushdns`
macOS	`sudo dscacheutil -flushcache; sudo killall -HUP mDNSResponder`

OS	Command
Linux (systemd-resolved)	`resolvectl flush-caches`
Chrome browser	open `chrome://net-internals/#dns` → Clear host cache

> **Ravindra's Tip**
>
> When a domain "doesn't work", split the problem in two: `dig example.com +short` tells you if **DNS** is right, and `curl -I http://13.233.10.25 -H "Host: example.com"` tells you if the **web server** is right. If both pass, it's only caching on your laptop. Kalji karu naka, just wait out the TTL or flush your cache. This one habit saves hours.

9.10 Subdomains

Subdomains are free and unlimited. Each one is just another DNS record plus (usually) another server block or virtual host.

```
example.com      ┐
www.example.com  ┼─► A 13.233.10.25 (web-1) ─► server_name decides the folder
blog.example.com ┘                             /var/www/example, /var/www/blog
api.example.com  ──► A 13.234.55.60 (app server, Node on :3000 behind Nginx)
shop.example.com ──► CNAME my-alb-123.ap-south-1.elb.amazonaws.com
```

Step 1: DNS record (GoDaddy → DNS → Add New Record)

Type	Name	Value	Use
A	`blog`	`13.233.10.25`	Same server, different site
A	`api`	`13.234.55.60`	Different EC2 instance (its own Elastic IP)
CNAME	`shop`	`my-alb-123.ap-south-1.elb.amazonaws.com`	AWS load balancer / other hostname
A	`*`	`13.233.10.25`	Wildcard: any undefined subdomain

In the **Name** field type only the subdomain part (`blog`), not `blog.example.com`. Otherwise you may create `blog.example.com.example.com`.

Step 2: a site for the subdomain on the server

```
sudo mkdir -p /var/www/blog
echo '<h1>Welcome to blog.example.com</h1>' | sudo tee /var/www/blog/index.html
```

```
# Nginx: blog.conf
server {
    listen 80;
    server_name blog.example.com;
    root /var/www/blog;
    index index.html;
}
```

```
# Apache: blog.conf
<VirtualHost *:80>
```

```
    ServerName blog.example.com
    DocumentRoot /var/www/blog
    <Directory /var/www/blog>
        Require all granted
    </Directory>
</VirtualHost>
```

Test, reload, then verify with `dig blog.example.com +short` and `curl http://blog.example.com`.

Subdomain for a backend app (reverse proxy)

```
server {
    listen 80;
    server_name api.example.com;
    location / {
        proxy_pass http://127.0.0.1:3000;        # Express / Flask app (Chapter 10)
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

(On CentOS remember `sudo setsebool -P httpd_can_network_connect 1`.)

9.11 HTTPS for your domain and subdomains

Once DNS resolves and port 443 is open, one certbot command secures all names (installation per OS is in section 7.13):

```
sudo certbot --nginx -d example.com -d www.example.com -d blog.example.com     # or --apache
sudo certbot certificates                # list certificates and expiry dates
sudo certbot renew --dry-run             # test automatic renewal
```

Certbot verifies that you control each name by making an HTTP request to it, so **every `-d` name must already resolve to this server**. If one name fails, check `dig name +short` first. Wildcard certificates (`*.example.com`) need a DNS challenge (a TXT record) instead. Use them later, once you're comfortable.

9.12 Troubleshooting domains

Symptom	Likely cause	Fix
`dig` returns nothing / **NXDOMAIN**	Record missing, typo in name, domain not verified or expired	Check the DNS page; verify registrant email; check expiry
`dig` shows **old/parking IP**	Old A record still present, TTL not expired, or forwarding on	Delete the parked A record; turn off forwarding; wait for TTL; `dig @8.8.8.8`
Records correct in GoDaddy but not used	Name servers point elsewhere (e.g. Route 53)	`dig NS example.com +short`; edit records where NS points
Domain resolves, browser **times out**	Security group / firewalld; wrong Elastic IP	`curl -I http://EIP`; open 80/443

Symptom	Likely cause	Fix
Domain shows **default welcome page**	`server_name` / `ServerName` doesn't include this name; default site wins	Add the name; remove Ubuntu default; `nginx -T`, `apachectl -S`
`www` works but apex doesn't (or reverse)	Only one of the records exists, or only one name is in `server_name`	Add A `@` and CNAME `www`; list both names in config
Works on mobile data, not on laptop	Local DNS cache	Flush cache (section 9.9)
certbot: Timeout during connect / NXDOMAIN	DNS not yet pointing here, or port 80 closed	Fix DNS first; open 80; retry
Email stopped after moving DNS	MX/TXT records not recreated	Copy MX/TXT to the new DNS host

Common Mistakes I See Students Make

- **Pointing the domain to the auto-assigned public IP**, then stopping the instance overnight.
- **Leaving GoDaddy's "Parked" A record** next to the new one, so the site randomly shows a parking page.
- **Typing the full name `blog.example.com` in the Name field** instead of just `blog`.
- **Creating a CNAME for the apex `@`**.
- **Editing records in GoDaddy after switching name servers to Route 53**, where they are ignored.
- **Forgetting `www`**, either in DNS or in `server_name`.
- **Buying all the add-ons** (hosting, email, SSL) that are not needed for EC2.
- **Not verifying the registrant email**, and the domain gets suspended.
- **Running certbot before DNS resolves**, and hitting the rate limit after repeated failures.

Interview Questions

Q1. Explain how DNS resolution works when you type a URL.

Browser/OS cache and `hosts` file → recursive resolver → root server → TLD server (returns NS of the domain) → authoritative name server (returns the record) → resolver caches it for the TTL and returns the IP → browser connects.

Q2. What is the difference between an A record and a CNAME?

An A record maps a name to an IPv4 address. A CNAME maps a name to another name, which is then resolved. A CNAME can't be used at the zone apex or alongside other records for the same name.

Q3. What is TTL? How would you plan a server migration with minimum DNS delay?

Time To Live, the number of seconds a resolver may cache a record. Lower the TTL (e.g. 300) well before the change, switch the record (or better, move the Elastic IP), verify, then raise the TTL again.

Q4. What is the difference between a registrar and a DNS hosting provider?

The registrar registers and renews the domain and stores which name servers are authoritative. The DNS host runs those name servers and answers queries with your records. They can be the same company or different ones (GoDaddy registrar + Route 53 DNS).

Q5. How do you host `blog.example.com` on the same EC2 instance as `example.com`?

Add a DNS record `blog` (A to the same Elastic IP), create `/var/www/blog`, add a server block/virtual host with `server_name blog.example.com`, test, reload and optionally extend the certificate with certbot.

Q6. Why use an Elastic IP or load balancer for DNS records?

The auto-assigned public IP changes on stop/start. A DNS record pointing to it would break, or point to someone else's instance.

Q7. What is a Route 53 Alias record?

A Route 53 extension that maps a name, including the zone apex, to AWS resources like load balancers, CloudFront or S3 websites. It follows their IP changes automatically, and alias queries to AWS resources are not charged.

Q8. Which commands do you use to check DNS?

`dig name +short`, `dig @8.8.8.8 name`, `dig NS domain`, `dig +trace name`, `nslookup name`, `host name`.

Practice Questions and Lab Exercises

1. Label each part of `api.shop.example.co.in.` (subdomain, second-level, TLD, root).
2. Explain registrar, registry, name server and web host with one example each.
3. List eight DNS record types and one use of each.
4. What is DNS propagation really? How does TTL affect it?
5. Why can't you use a CNAME at the apex? What do you use instead on Route 53?
6. **Lab:** Attach an Elastic IP to your web server and map `example.com` and `www.example.com` using GoDaddy DNS (or your registrar). Verify with `dig` and `nslookup`.
7. **Lab:** Create `blog.` and `api.` subdomains: `blog` as a static site on the same server, `api` as a reverse proxy to a Node app on port 3000.
8. **Lab:** Create a Route 53 public hosted zone, recreate your records, switch the GoDaddy name servers and confirm with `dig NS`. (Delete the hosted zone afterwards if you don't need it.)
9. **Lab:** Secure all names with one certbot command and test renewal with `--dry-run`.

★ Quick Revision

- Domain parts, right to left: root `.` → TLD → your domain → subdomains (free, unlimited).
- Registrar (buy/renew) ≠ name servers (answer DNS) ≠ web host (EC2).
- Resolution: cache → resolver → root → TLD → authoritative → cached for **TTL**. "Propagation" = caches expiring.
- Records: **A** (IPv4), AAAA, **CNAME** (alias, not at apex), MX, TXT, NS, SOA, CAA, Route 53 Alias.
- Setup: website works → **Elastic IP** (details in 14.1) → GoDaddy DNS: A `@` → EIP, CNAME `www` → `@`, remove parked records → `server_name` / `ServerName` + `ServerAlias` → reload → `dig` → certbot.
- Route 53 option: hosted zone → copy 4 NS → set custom name servers at GoDaddy → recreate MX/TXT first.
- Subdomain = DNS record + its own server block/vhost (or proxy to an app).
- Verify: `dig +short`, `dig @8.8.8.8`, `dig NS`, `dig +trace`, `nslookup`; flush local cache if needed.

Chapter 10: Dynamic Website Hosting (PHP, Python, Node.js with MySQL)

10.1 Why and what

Mitranno, static sites are nice, but real applications need logins, forms and data. Let's first understand what changes when a site becomes dynamic. A **dynamic website** generates pages at request time using server-side code and usually a database — login systems, e-commerce, result portals, blogs. The page for user A can differ from the page for user B.

```
                    3-tier architecture on a single EC2 instance
Browser ──HTTP:80──► [ Web tier ]  Nginx / Apache (reverse proxy, static files)
                          │
                          ▼  FastCGI socket / HTTP 127.0.0.1:3000 or :5000
                      [ App tier ]  PHP-FPM  |  Gunicorn + Flask  |  Node + Express
                          │
                          ▼  TCP 127.0.0.1:3306 or unix socket
                      [ Data tier ] MySQL / MariaDB  (never exposed to the internet)
```

Language	App server / runtime	How the web server talks to it	Process manager
PHP	**PHP-FPM** (FastCGI Process Manager) or Apache `mod_php`	FastCGI over a unix socket	systemd (`php-fpm` / `php8.x-fpm`)
Python (Flask/ Django)	**Gunicorn** (WSGI server)	HTTP reverse proxy to `127.0.0.1:5000`	systemd unit
Node.js (Express)	Node itself	HTTP reverse proxy to `127.0.0.1:3000`	**pm2** or systemd unit

✔ Why a reverse proxy?

App servers are good at running code but not at handling slow clients, TLS, compression, static files and security headers. Nginx/Apache in front does that job, lets you use port 80/443 without running your app as root, and lets many apps share one server.

10.2 Installing MySQL / MariaDB

Think of the database like the college office register where every student's record is kept — the web app only asks the office for data; it does not keep the register itself. Technically, the app opens a TCP (or unix socket) connection to the database server on port 3306 and sends SQL queries.

MariaDB is a community fork of MySQL and is fully compatible for our purposes (same `mysql` client, same SQL, same port 3306).

OS	Package	Service	Admin login
Amazon Linux 2023	`mariadb105-server`	`mariadb`	`sudo mysql`
Ubuntu 22.04/24.04	`mysql-server` (MySQL 8.x) or `mariadb-server`	`mysql` (or `mariadb`)	`sudo mysql` (root uses auth_socket)
CentOS Stream 9	`mariadb-server` (or `mysql-server` for MySQL 8)	`mariadb` (or `mysqld`)	`sudo mysql`
Amazon Linux 2 (older)	`mariadb-server`	`mariadb`	`sudo mysql`

Amazon Linux 2023

```
sudo yum install -y mariadb105-server
sudo service mariadb start
sudo systemctl enable mariadb
sudo service mariadb status
```

Ubuntu

```
sudo apt update
sudo apt install -y mysql-server
sudo service mysql start
sudo systemctl enable mysql
sudo service mysql status
```

CentOS Stream 9

```
sudo yum install -y mariadb-server
sudo service mariadb start
sudo systemctl enable mariadb
sudo service mariadb status
```

Securing the installation

```
sudo mysql_secure_installation
```

Recommended answers: switch to unix_socket authentication → **n** (keep current) or accept default; set/change root password → optional (root can already log in with `sudo mysql`); **remove anonymous users → Y**; **disallow root login remotely → Y**; **remove test database → Y**; **reload privilege tables → Y**. On Ubuntu MySQL you will first be asked about the VALIDATE PASSWORD component — choose **Y** and level **1 (MEDIUM)** for practice.

> **i Ubuntu MySQL root login**
>
> On Ubuntu, MySQL's `root@localhost` uses the **auth_socket** plugin: it logs in only via `sudo mysql` (no password). If `mysql_secure_installation` complains while setting the root password, simply skip that step. Applications should **never** use root — create a dedicated user as shown below.

Create the application database and user

Chala, aata database tayar karuya (let's set up the database). This creates database `appdb`, user `appuser` with password `App@Pass123` (change it!), and a sample `students` table used by all three apps in this chapter:

```
sudo mysql <<'EOF'
CREATE DATABASE IF NOT EXISTS appdb;
CREATE USER IF NOT EXISTS 'appuser'@'localhost' IDENTIFIED BY 'App@Pass123';
CREATE USER IF NOT EXISTS 'appuser'@'127.0.0.1' IDENTIFIED BY 'App@Pass123';
GRANT ALL PRIVILEGES ON appdb.* TO 'appuser'@'localhost';
GRANT ALL PRIVILEGES ON appdb.* TO 'appuser'@'127.0.0.1';
FLUSH PRIVILEGES;
USE appdb;
CREATE TABLE IF NOT EXISTS students (
  id    INT AUTO_INCREMENT PRIMARY KEY,
  name  VARCHAR(100) NOT NULL,
  city  VARCHAR(50)  NOT NULL
);
INSERT INTO students (name, city) VALUES ('Aarav Patil','Pune'), ('Sneha Deshmukh','Nagpur'),
('Rohan Kulkarni','Nashik');
EOF

# verify as the new user (enter App@Pass123 when prompted)
mysql -u appuser -p -h 127.0.0.1 -e "SELECT * FROM appdb.students;"
```

Useful SQL/admin commands:

```
SHOW DATABASES;
SELECT user, host FROM mysql.user;
SHOW GRANTS FOR 'appuser'@'localhost';
ALTER USER 'appuser'@'localhost' IDENTIFIED BY 'NewPass@456';
DROP USER 'appuser'@'127.0.0.1';
```

```
# backup and restore
mysqldump -u appuser -p appdb > appdb-backup.sql
mysql -u appuser -p appdb < appdb-backup.sql
```

> ⚠ **Keep the database private**
>
> MySQL/MariaDB should listen only on `127.0.0.1` (check with `sudo ss -tlnp | grep 3306`) and port **3306 must not be open** in the security group. If the database is on a separate EC2/RDS, allow 3306 **only from the web server's security group**, not from `0.0.0.0/0`.

> **Ravindra's Tip**
>
> Never let your application log in as the database `root` user. Create one user per application with access to only its own database. If the app is ever hacked, the damage stays limited to that one database. Lakshat theva — least privilege everywhere.

10.3 PHP with MySQL

Install PHP

OS + web server	Install command	PHP handler
AL2023 + Nginx or Apache	`sudo yum install -y php php-fpm php-mysqlnd php-cli`	PHP-FPM socket `/run/php-fpm/www.sock`
CentOS Stream 9 + Nginx or Apache	`sudo yum install -y php php-fpm php-mysqlnd php-cli`	PHP-FPM socket `/run/php-fpm/www.sock`
Ubuntu + Nginx	`sudo apt install -y php-fpm php-mysql`	PHP-FPM socket `/run/php/php8.x-fpm.sock`
Ubuntu + Apache	`sudo apt install -y php libapache2-mod-php php-mysql`	Apache `mod_php` (no FPM needed)

On AL2023 / CentOS, start PHP-FPM and restart the web server:

```
sudo service php-fpm start
sudo systemctl enable php-fpm
sudo service nginx restart        # or: sudo service httpd restart
php -v
```

On Amazon Linux 2023 and CentOS, **Apache** is automatically wired to PHP-FPM by `/etc/httpd/conf.d/php.conf`, and the default **Nginx** server is wired by `/etc/nginx/default.d/php.conf`. For your own Nginx server block, add the PHP location shown below.

Nginx server block for PHP

AL2023 / CentOS Stream 9 — `/etc/nginx/conf.d/phpapp.conf`:

```
sudo mkdir -p /var/www/phpapp
sudo tee /etc/nginx/conf.d/phpapp.conf > /dev/null <<'EOF'
server {
    listen 80;
    server_name YOUR_SERVER_NAME;
    root /var/www/phpapp;
    index index.php index.html;

    location / {
        try_files $uri $uri/ /index.php?$args;
    }
    location ~ \.php$ {
        try_files $uri =404;
        fastcgi_pass unix:/run/php-fpm/www.sock;
        fastcgi_index index.php;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    }
}
EOF
sudo sed -i "s/YOUR_SERVER_NAME/$(curl -s https://checkip.amazonaws.com)/" /etc/nginx/conf.d/
phpapp.conf
sudo nginx -t && sudo service nginx reload
```

Ubuntu — `/etc/nginx/sites-available/phpapp`:

```
ls /run/php/                    # note the socket name, e.g. php8.3-fpm.sock (24.04) or php8.1-
fpm.sock (22.04)
sudo mkdir -p /var/www/phpapp
sudo tee /etc/nginx/sites-available/phpapp > /dev/null <<'EOF'
server {
    listen 80 default_server;
    listen [::]:80 default_server;
    server_name _;
    root /var/www/phpapp;
    index index.php index.html;

    location / {
        try_files $uri $uri/ /index.php?$args;
    }
    location ~ \.php$ {
        include snippets/fastcgi-php.conf;
        fastcgi_pass unix:/run/php/php-fpm.sock;
    }
}
EOF
sudo ln -sf /etc/nginx/sites-available/phpapp /etc/nginx/sites-enabled/phpapp
sudo rm -f /etc/nginx/sites-enabled/default /etc/nginx/sites-enabled/mysite
sudo nginx -t && sudo service nginx reload
```

`/run/php/php-fpm.sock` is a symlink to the versioned socket; if it does not exist on your system, replace it with the exact name shown by `ls /run/php/`.

Apache virtual host for PHP

With Apache, PHP works automatically once PHP is installed (mod_php on Ubuntu, php-fpm via `php.conf` on AL2023/CentOS). Just point a virtual host at the folder (as in Chapter 7) with `DirectoryIndex index.php index.html`, or use the default `/var/www/html`.

The PHP application

```
sudo tee /var/www/phpapp/db.php > /dev/null <<'EOF'
<?php
$dsn  = "mysql:host=127.0.0.1;dbname=appdb;charset=utf8mb4";
$user = "appuser";
$pass = "App@Pass123";
try {
    $pdo = new PDO($dsn, $user, $pass, [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION]);
} catch (PDOException $e) {
    http_response_code(500);
    die("Database connection failed: " . htmlspecialchars($e->getMessage()));
}
EOF

sudo tee /var/www/phpapp/index.php > /dev/null <<'EOF'
<?php
require __DIR__ . "/db.php";
if ($_SERVER["REQUEST_METHOD"] === "POST" && !empty($_POST["name"]) && !empty($_POST["city"])) {
    $stmt = $pdo->prepare("INSERT INTO students (name, city) VALUES (?, ?)");   // prepared =
safe from SQL injection
    $stmt->execute([$_POST["name"], $_POST["city"]]);
    header("Location: /");
    exit;
}
$rows = $pdo->query("SELECT id, name, city FROM students ORDER BY id")-
```

```
>fetchAll(PDO::FETCH_ASSOC);
?>
<!DOCTYPE html>
<html><head><meta charset="utf-8"><title>PHP + MySQL on EC2</title></head>
<body style="font-family:Arial;max-width:700px;margin:30px auto">
<h1>Students (PHP + MySQL)</h1>
<table border="1" cellpadding="6">
<tr><th>ID</th><th>Name</th><th>City</th></tr>
<?php foreach ($rows as $r): ?>
<tr><td><?= $r["id"] ?></td><td><?= htmlspecialchars($r["name"]) ?></td><td><?=
htmlspecialchars($r["city"]) ?></td></tr>
<?php endforeach; ?>
</table>
<h3>Add student</h3>
<form method="post">
  <input name="name" placeholder="Name" required>
  <input name="city" placeholder="City" required>
  <button type="submit">Add</button>
</form>
<p>Served by PHP <?= phpversion() ?> on <?= gethostname() ?></p>
</body></html>
EOF

sudo chmod 640 /var/www/phpapp/db.php
# let the PHP process user read db.php:  apache (AL2023/CentOS php-fpm & httpd) or www-data
(Ubuntu)
sudo chown root:apache /var/www/phpapp/db.php      # Ubuntu: sudo chown root:www-data /var/www/
phpapp/db.php
sudo restorecon -Rv /var/www/phpapp 2>/dev/null   # CentOS only (harmless elsewhere)
curl -s http://localhost/ -H "Host: $(curl -s https://checkip.amazonaws.com)" | head -20
```

i Quick PHP test page

`echo "<?php phpinfo();" | sudo tee /var/www/phpapp/info.php` shows full PHP configuration. **Delete it after testing** — it reveals sensitive details.

⚠ PHP file downloads instead of running?

If the browser downloads `index.php` or shows raw PHP code, the web server is not passing `.php` files to PHP: PHP-FPM is not running, the `location ~ \.php$` block is missing, or the socket path is wrong. Check `sudo service php-fpm status` (Ubuntu: `php8.3-fpm`) and the Nginx error log.

10.4 Python (Flask) with MySQL

Ekdum simple aahe — the pattern is the same as PHP, only the app server changes. Type these commands with me.

Install Python tools

```
# Amazon Linux 2023 / CentOS Stream 9
sudo yum install -y python3 python3-pip git
# Ubuntu
sudo apt install -y python3 python3-venv python3-pip git
```

Create the app in /opt/flaskapp

We keep apps in `/opt/<app>` owned by the login user. (On CentOS, SELinux may block systemd from running programs stored inside home directories, so `/opt` avoids that problem on every OS.)

```
sudo mkdir -p /opt/flaskapp && sudo chown $USER:$USER /opt/flaskapp
cd /opt/flaskapp
python3 -m venv venv
source venv/bin/activate
pip install --upgrade pip
pip install flask pymysql cryptography gunicorn
pip freeze > requirements.txt
```

```
cat > /opt/flaskapp/.env <<'EOF'
DB_HOST=127.0.0.1
DB_USER=appuser
DB_PASS=App@Pass123
DB_NAME=appdb
EOF
chmod 600 /opt/flaskapp/.env
```

```
cat > /opt/flaskapp/app.py <<'EOF'
import os
import pymysql
from flask import Flask, jsonify, redirect, render_template_string, request

app = Flask(__name__)

def get_db():
    return pymysql.connect(
        host=os.getenv("DB_HOST", "127.0.0.1"),
        user=os.getenv("DB_USER", "appuser"),
        password=os.getenv("DB_PASS", ""),
        database=os.getenv("DB_NAME", "appdb"),
        cursorclass=pymysql.cursors.DictCursor,
    )

PAGE = """
<!DOCTYPE html><html><head><meta charset="utf-8"><title>Flask + MySQL</title></head>
<body style="font-family:Arial;max-width:700px;margin:30px auto">
<h1>Students (Flask + MySQL)</h1>
<table border="1" cellpadding="6"><tr><th>ID</th><th>Name</th><th>City</th></tr>
{% for s in rows %}<tr><td>{{ s.id }}</td><td>{{ s.name }}</td><td>{{ s.city }}</td></tr>{%
endfor %}
</table>
<h3>Add student</h3>
<form method="post" action="/add">
  <input name="name" placeholder="Name" required>
  <input name="city" placeholder="City" required>
  <button>Add</button>
</form>
</body></html>
"""

@app.route("/")
def index():
    conn = get_db()
    with conn, conn.cursor() as cur:
        cur.execute("SELECT id, name, city FROM students ORDER BY id")
        rows = cur.fetchall()
```

```
    return render_template_string(PAGE, rows=rows)

@app.route("/add", methods=["POST"])
def add():
    conn = get_db()
    with conn, conn.cursor() as cur:
        cur.execute("INSERT INTO students (name, city) VALUES (%s, %s)",
                    (request.form["name"], request.form["city"]))
        conn.commit()
    return redirect("/")

@app.route("/api/students")
def api_students():
    conn = get_db()
    with conn, conn.cursor() as cur:
        cur.execute("SELECT id, name, city FROM students")
        return jsonify(cur.fetchall())

@app.route("/health")
def health():
    return {"status": "ok"}

if __name__ == "__main__":
    app.run(host="127.0.0.1", port=5000, debug=False)
EOF
```

Test it manually

```
cd /opt/flaskapp && source venv/bin/activate
set -a && source .env && set +a            # load variables from .env into this shell
gunicorn --bind 127.0.0.1:5000 app:app &   # start in background for a quick test
sleep 2
curl -s http://127.0.0.1:5000/api/students
kill %1                                     # stop the test server
deactivate
```

Run with Gunicorn as a systemd service

```
sudo tee /etc/systemd/system/flaskapp.service > /dev/null <<EOF
[Unit]
Description=Flask app served by Gunicorn
After=network.target mariadb.service mysql.service

[Service]
User=$USER
Group=$USER
WorkingDirectory=/opt/flaskapp
EnvironmentFile=/opt/flaskapp/.env
ExecStart=/opt/flaskapp/venv/bin/gunicorn --workers 3 --bind 127.0.0.1:5000 app:app
Restart=always
RestartSec=3

[Install]
WantedBy=multi-user.target
EOF

sudo systemctl daemon-reload
sudo service flaskapp start
sudo systemctl enable flaskapp
```

```
sudo service flaskapp status
curl -s http://127.0.0.1:5000/health
```

(Here the here-doc delimiter `EOF` is **not** quoted, so `$USER` is replaced by your user name — `ec2-user` or `ubuntu`.)

10.5 Node.js (Express) with MySQL

Install Node.js (LTS) from NodeSource

```
# Amazon Linux 2023 / CentOS Stream 9
curl -fsSL https://rpm.nodesource.com/setup_22.x | sudo bash -
sudo yum install -y nodejs git

# Ubuntu
curl -fsSL https://deb.nodesource.com/setup_22.x | sudo -E bash -
sudo apt install -y nodejs git

node -v && npm -v
```

✔ Alternative: nvm

`nvm` (Node Version Manager) installs Node per user without sudo and lets you switch versions: install it from the official nvm GitHub page, then `nvm install --lts`. On Amazon Linux 2023 you can also use the distribution packages (`yum search nodejs`).

Create the Express app

```
sudo mkdir -p /opt/nodeapp && sudo chown $USER:$USER /opt/nodeapp
cd /opt/nodeapp
npm init -y
npm install express mysql2 dotenv

cat > .env <<'EOF'
DB_HOST=127.0.0.1
DB_USER=appuser
DB_PASS=App@Pass123
DB_NAME=appdb
PORT=3000
EOF
chmod 600 .env

cat > app.js <<'EOF'
require("dotenv").config();
const express = require("express");
const mysql = require("mysql2/promise");

const app = express();
app.use(express.urlencoded({ extended: true }));
app.use(express.json());

const pool = mysql.createPool({
  host: process.env.DB_HOST,          // use 127.0.0.1, not "localhost" (avoids IPv6 ::1 issue)
  user: process.env.DB_USER,
  password: process.env.DB_PASS,
  database: process.env.DB_NAME,
```

```
  connectionLimit: 10,
});

const esc = (s) => String(s).replace(/[&<>"']/g, (c) => `&#${c.charCodeAt(0)};`);

app.get("/", async (req, res) => {
  try {
    const [rows] = await pool.query("SELECT id, name, city FROM students ORDER BY id");
    const trs = rows.map(r => `<tr><td>${r.id}</td><td>${esc(r.name)}</td><td>${esc(r.city)}</
td></tr>`).join("");
    res.send(`<!DOCTYPE html><html><body style="font-family:Arial;max-width:700px;margin:30px
auto">
      <h1>Students (Node.js + Express + MySQL)</h1>
      <table border="1" cellpadding="6"><tr><th>ID</th><th>Name</th><th>City</th></tr>${trs}</
table>
      <h3>Add student</h3>
      <form method="post" action="/add"><input name="name" required placeholder="Name">
      <input name="city" required placeholder="City"><button>Add</button></form></body></html>`);
  } catch (err) {
    res.status(500).send("Database error: " + esc(err.message));
  }
});

app.post("/add", async (req, res) => {
  await pool.execute("INSERT INTO students (name, city) VALUES (?, ?)", [req.body.name,
req.body.city]);
  res.redirect("/");
});

app.get("/api/students", async (req, res) => {
  const [rows] = await pool.query("SELECT id, name, city FROM students");
  res.json(rows);
});

app.get("/health", (req, res) => res.json({ status: "ok" }));

const port = process.env.PORT || 3000;
app.listen(port, "127.0.0.1", () => console.log(`API listening on 127.0.0.1:${port}`));
EOF

node app.js &          # quick test
sleep 2 && curl -s http://127.0.0.1:3000/api/students
kill %1
```

Option A — keep it running with pm2

```
sudo npm install -g pm2
cd /opt/nodeapp
pm2 start app.js --name nodeapp
pm2 status
pm2 logs nodeapp --lines 20        # Ctrl+C to exit logs
pm2 startup systemd                # prints a 'sudo env PATH=...' command - copy & run it
pm2 save                           # remember the current process list for reboot
```

Option B — run it as a systemd service

```
sudo tee /etc/systemd/system/nodeapp.service > /dev/null <<EOF
[Unit]
Description=Node.js Express app
After=network.target mariadb.service mysql.service
```

```
[Service]
User=$USER
WorkingDirectory=/opt/nodeapp
ExecStart=/usr/bin/node /opt/nodeapp/app.js
Restart=always
Environment=NODE_ENV=production

[Install]
WantedBy=multi-user.target
EOF
sudo systemctl daemon-reload
sudo service nodeapp start
sudo systemctl enable nodeapp
```

Use **either** pm2 **or** systemd for one app, not both (they would fight for port 3000).

10.6 Reverse proxy to the Python/Node app

In my sessions, students often ask: "Sir, my app runs on port 3000, why not just open 3000 in the security group?" Bagha — the reverse proxy gives you port 80/443, TLS, logging and protection, and keeps the app private on 127.0.0.1. That is how it is done in real companies.

Nginx reverse proxy (all OSes)

Save as `/etc/nginx/conf.d/app.conf` on AL2023/CentOS, or `/etc/nginx/sites-available/app` (plus symlink into `sites-enabled` and remove `default`) on Ubuntu. Use port **5000** for Flask or **3000** for Node.

```
server {
    listen 80;
    server_name YOUR_SERVER_NAME;          # public IP / domain  (Ubuntu: use "listen 80
default_server;" and "server_name _;")

    location / {
        proxy_pass http://127.0.0.1:3000;  # 5000 for Flask/Gunicorn
        proxy_http_version 1.1;
        proxy_set_header Host              $host;
        proxy_set_header X-Real-IP         $remote_addr;
        proxy_set_header X-Forwarded-For   $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header Upgrade           $http_upgrade;     # WebSocket support
        proxy_set_header Connection        "upgrade";
    }
}
```

```
sudo nginx -t && sudo service nginx reload
sudo setsebool -P httpd_can_network_connect 1     # CentOS Stream 9 only - otherwise 502 Bad
Gateway
```

Apache reverse proxy

```
<VirtualHost *:80>
    ServerName mysite.local
    ProxyPreserveHost On
    ProxyPass        / http://127.0.0.1:3000/
```

```
    ProxyPassReverse / http://127.0.0.1:3000/
    ErrorLog  logs/app_error.log
    CustomLog logs/app_access.log combined
</VirtualHost>
```

- **AL2023 / CentOS:** save as `/etc/httpd/conf.d/app.conf` (`mod_proxy` and `mod_proxy_http` are loaded by default), then `sudo apachectl configtest && sudo service httpd reload`. On CentOS also run `sudo setsebool -P httpd_can_network_connect 1`.
- **Ubuntu:** save as `/etc/apache2/sites-available/app.conf`, change the log lines to `${APACHE_LOG_DIR}/app_error.log` and `${APACHE_LOG_DIR}/app_access.log`, then `sudo a2enmod proxy proxy_http && sudo a2ensite app.conf && sudo a2dissite 000-default.conf && sudo service apache2 reload`.

> **RB Ravindra's Tip**
>
> When you see **502 Bad Gateway**, don't touch Nginx first. Run `curl -I http://127.0.0.1:3000` (or 5000) on the server. If the app itself doesn't answer, the problem is the app — check `pm2 logs` or `journalctl -u <service>`. Fix the backend, and the 502 disappears automatically.

10.7 Troubleshooting dynamic sites

Symptom	Cause	Fix
502 Bad Gateway	App not running; wrong port/ socket; SELinux blocks proxy	`sudo service flaskapp status` / `pm2 status`; `curl 127.0.0.1:3000`; `setsebool -P httpd_ can_network_connect 1`
504 Gateway Timeout	App too slow / stuck on DB	Check app logs; DB running?
`Access denied for user 'appuser'`	Wrong password or host (`localhost` vs `127.0.0.1`)	`SELECT user,host FROM mysql.user;` recreate user
`ECONNREFUSED ::1:3306` (Node)	`localhost` resolved to IPv6	Use `DB_HOST=127.0.0.1`
`Can't connect to MySQL server`	MariaDB/MySQL stopped	`sudo service mariadb start` (or `mysql`)
`cryptography package is required` (Python)	MySQL 8 auth needs it	`pip install cryptography`
PHP shown as text / downloaded	PHP handler missing	Install/start php-fpm; check `location ~ \.php$`
Blank page / 500 in PHP	PHP error	`sudo tail /var/log/php-fpm/www-error.log` (AL/CentOS) or Nginx/Apache error log
systemd service `status=203/EXEC`	Wrong path in `ExecStart`, or SELinux blocking a binary in a home folder	Check path; keep apps in `/opt`

Common Mistakes I See Students Make

- **Hard-coding passwords in code and pushing to GitHub.** Use `.env` files with permission 600 and add them to `.gitignore`.
- **Using `localhost` in Node.js** and getting `ECONNREFUSED ::1:3306`. Use `127.0.0.1`.
- **Running the app with `node app.js` in the SSH session** and closing the terminal — the app dies. Use pm2 or systemd.
- **Running both pm2 and a systemd unit for the same app** → port 3000 already in use.
- **Opening port 3306 or 3000 to the internet** instead of using the reverse proxy.
- **Forgetting `setsebool -P httpd_can_network_connect 1` on CentOS** → 502 Bad Gateway.
- **Not restarting PHP-FPM/Apache after installing PHP modules.**

Interview Questions

Q1. What is PHP-FPM?

FastCGI Process Manager — a separate service that runs PHP worker processes; Nginx (or Apache) passes .php requests to it over a unix socket or TCP.

Q2. Why do we run Gunicorn in front of a Flask app?

Flask's built-in server is for development only. Gunicorn is a production WSGI server that runs multiple worker processes reliably; Nginx sits in front for TLS and static files.

Q3. What is pm2 used for?

A process manager for Node.js that keeps apps running, restarts them on crash, starts them at boot (`pm2 startup` + `pm2 save`), manages logs and supports cluster mode.

Q4. Why should the database never be exposed to the internet?

Open database ports are scanned and attacked constantly (brute force, ransomware). The DB should listen on localhost or a private subnet and allow access only from the app's security group.

Q5. How do you create a MySQL user with access to only one database?

`CREATE USER 'app'@'localhost' IDENTIFIED BY '...'; GRANT ALL PRIVILEGES ON appdb.* TO 'app'@'localhost'; FLUSH PRIVILEGES;`

Q6. What causes a 502 Bad Gateway error behind Nginx?

The upstream is unreachable: app not running, wrong port/socket in `proxy_pass` / `fastcgi_pass`, app crashed, or SELinux blocking the connection.

Q7. How do you run an application as a service on Linux?

Create a systemd unit file in `/etc/systemd/system/`, then `sudo systemctl daemon-reload`, `sudo service <name> start` and `sudo systemctl enable <name>`.

Practice Questions and Lab Exercises

1. Draw the 3-tier architecture of a dynamic website on a single EC2 instance.
2. What is PHP-FPM? How does Nginx communicate with it?

3. Why should the app listen on `127.0.0.1` and not `0.0.0.0`?
4. Write SQL to create a database `college`, a user `web` with password, and grant all privileges on that database only.
5. What is the purpose of Gunicorn? Of pm2?
6. Explain each line of the Flask systemd unit file.
7. What causes 502 Bad Gateway? Give three checks.
8. **Lab:** Deploy the PHP app on Ubuntu + Nginx, add three students via the form and verify the rows in MySQL.
9. **Lab:** Deploy the Flask app on Amazon Linux 2023 behind Nginx with a systemd service; reboot the instance and confirm the app starts automatically.
10. **Lab:** Deploy the Node app on CentOS Stream 9 with pm2 behind Nginx. Observe the 502 error before running `setsebool` and after.

★ Quick Revision

- Dynamic = code + DB. Web server → app server → database.
- DB: AL2023 `mariadb105-server`, Ubuntu `mysql-server`, CentOS `mariadb-server`; `mysql_secure_installation`; create dedicated user; never open 3306.
- PHP: PHP-FPM socket (`/run/php-fpm/www.sock` or `/run/php/php8.x-fpm.sock`); Apache on Ubuntu uses `libapache2-mod-php`.
- Python: venv + Flask + PyMySQL + Gunicorn under systemd on `127.0.0.1:5000`.
- Node: NodeSource LTS + Express + mysql2; pm2 (`start`, `startup`, `save`) or systemd on `127.0.0.1:3000`.
- Reverse proxy: Nginx `proxy_pass`, Apache `ProxyPass`; on CentOS `setsebool -P httpd_can_network_connect 1`.

Chapter 11: Application Hosting — WordPress, Flask and Express

Mitranno, in Chapter 10 you learnt the building blocks. Now it is time to put them together. In this chapter we deploy three **complete applications** the way you would in a real job: WordPress (the most popular CMS in the world), a Python Flask application with Gunicorn + Nginx, and a Node.js Express application with pm2 + Nginx.

11.1 WordPress

Why and what

WordPress is an open-source content management system (CMS) written in PHP that uses MySQL/MariaDB. It powers a very large share of all websites — blogs, college department sites, small business and news portals. Hosting it yourself teaches the complete LAMP/LEMP workflow.

```
 Browser → Apache/Nginx :80 → PHP (mod_php / PHP-FPM) → WordPress PHP files (/var/www/
wordpress)
                                                                    |
                                                                    ▼
                                                  MariaDB/MySQL database "wordpress"
```

Requirements: PHP 7.4+ (8.x recommended), MySQL 8.0+ or MariaDB 10.5+, a web server with URL rewriting, ~1 GB RAM (a `t3.micro` works for learning; add swap if needed).

Step 1 — Install the stack

Chala, aata WordPress install karuya. Choose your OS and follow along — don't skip any step.

Ubuntu (Apache):

```
sudo apt update
sudo apt install -y apache2 mysql-server php libapache2-mod-php php-mysql \
  php-curl php-gd php-mbstring php-xml php-intl php-zip php-soap wget unzip
sudo service apache2 start
sudo service mysql start
sudo systemctl enable apache2 mysql
```

Amazon Linux 2023 (Apache):

```
sudo yum install -y httpd mariadb105-server php php-fpm php-mysqlnd \
  php-gd php-mbstring php-xml php-intl php-opcache wget
sudo service httpd start
sudo service mariadb start
sudo service php-fpm start
sudo systemctl enable httpd mariadb php-fpm
```

CentOS Stream 9 (Apache):

```
sudo yum install -y httpd mariadb-server php php-fpm php-mysqlnd \
  php-gd php-mbstring php-xml php-intl php-opcache wget tar
sudo service httpd start
sudo service mariadb start
sudo service php-fpm start
sudo systemctl enable httpd mariadb php-fpm
```

Then secure the database: `sudo mysql_secure_installation` (see Chapter 10).

Step 2 — Create the WordPress database

```
sudo mysql <<'EOF'
CREATE DATABASE wordpress DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
CREATE USER 'wpuser'@'localhost' IDENTIFIED BY 'Wp@Pass123';
GRANT ALL PRIVILEGES ON wordpress.* TO 'wpuser'@'localhost';
FLUSH PRIVILEGES;
EOF
```

Step 3 — Download and extract WordPress

```
cd /tmp
wget https://wordpress.org/latest.tar.gz
tar -xzf latest.tar.gz
sudo mv wordpress /var/www/wordpress
ls /var/www/wordpress
```

Step 4 — Configure wp-config.php

```
cd /var/www/wordpress
sudo cp wp-config-sample.php wp-config.php
sudo sed -i "s/database_name_here/wordpress/" wp-config.php
sudo sed -i "s/username_here/wpuser/" wp-config.php
sudo sed -i "s/password_here/Wp@Pass123/" wp-config.php
grep DB_ wp-config.php
```

Add unique security keys (salts). Get a fresh set from the official generator:

```
curl -s https://api.wordpress.org/secret-key/1.1/salt/
sudo nano /var/www/wordpress/wp-config.php
```

In nano, delete the eight lines that contain `put your unique phrase here` and paste the eight lines printed by `curl`. Save (Ctrl+O, Enter) and exit (Ctrl+X).

Step 5 — Ownership and permissions

```
# Ubuntu:
sudo chown -R www-data:www-data /var/www/wordpress
# Amazon Linux 2023 / CentOS Stream 9:
sudo chown -R apache:apache /var/www/wordpress

sudo find /var/www/wordpress -type d -exec chmod 755 {} \;
sudo find /var/www/wordpress -type f -exec chmod 644 {} \;
sudo chmod 640 /var/www/wordpress/wp-config.php
```

CentOS Stream 9 only — SELinux: allow WordPress to write uploads/plugins and to make outgoing connections (updates, plugin downloads):

```
sudo yum install -y policycoreutils-python-utils
sudo semanage fcontext -a -t httpd_sys_rw_content_t "/var/www/wordpress/wp-content(/.*)?"
sudo restorecon -Rv /var/www/wordpress
sudo setsebool -P httpd_can_network_connect 1
```

Step 6a — Apache virtual host

Ubuntu (`/etc/apache2/sites-available/wordpress.conf`):

```
sudo tee /etc/apache2/sites-available/wordpress.conf > /dev/null <<'EOF'
<VirtualHost *:80>
    ServerName blog.local
    DocumentRoot /var/www/wordpress
    <Directory /var/www/wordpress>
        AllowOverride All
        Require all granted
    </Directory>
    ErrorLog  ${APACHE_LOG_DIR}/wordpress_error.log
    CustomLog ${APACHE_LOG_DIR}/wordpress_access.log combined
</VirtualHost>
EOF
sudo a2ensite wordpress.conf
sudo a2dissite 000-default.conf
sudo a2enmod rewrite
sudo apache2ctl configtest && sudo service apache2 reload
```

Amazon Linux 2023 / CentOS Stream 9 (`/etc/httpd/conf.d/wordpress.conf`):

```
sudo tee /etc/httpd/conf.d/wordpress.conf > /dev/null <<'EOF'
<VirtualHost *:80>
    ServerName blog.local
    DocumentRoot /var/www/wordpress
    DirectoryIndex index.php index.html
    <Directory /var/www/wordpress>
        AllowOverride All
        Require all granted
    </Directory>
    ErrorLog  /var/log/httpd/wordpress_error.log
    CustomLog /var/log/httpd/wordpress_access.log combined
</VirtualHost>
EOF
sudo apachectl configtest && sudo service httpd reload
```

(`mod_rewrite` is enabled by default on AL2023/CentOS. `AllowOverride All` lets WordPress's `.htaccess` create pretty permalinks.)

Step 6b — Or use Nginx instead of Apache (LEMP)

Install `nginx` and PHP-FPM instead of Apache (`sudo apt install -y nginx php-fpm ...` on Ubuntu; stop/disable Apache first). Server block:

```
server {
    listen 80;
    server_name YOUR_SERVER_NAME;                # public IP or domain
    root /var/www/wordpress;
```

```
    index index.php index.html;
    client_max_body_size 64M;                    # allow bigger media uploads

    location / {
        try_files $uri $uri/ /index.php?$args;    # pretty permalinks
    }
    location ~ \.php$ {
        try_files $uri =404;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        fastcgi_pass unix:/run/php-fpm/www.sock;  # Ubuntu: unix:/run/php/php-fpm.sock
    }
    location ~* \.(css|js|png|jpg|jpeg|gif|svg|webp|ico)$ {
        expires 30d;
        access_log off;
    }
    location ~ /\.ht { deny all; }
}
```

i PHP-FPM user with Nginx

On AL2023/CentOS PHP-FPM runs as user `apache` by default (even with Nginx), so `chown -R apache:apache` is still correct. On Ubuntu both Nginx and PHP-FPM run as `www-data`.

RB Ravindra's Tip

Attach an **Elastic IP before** you open the WordPress installer. WordPress saves the site URL in its database, and if the public IP changes later, every link and CSS file points to the old IP. Two minutes of planning saves an hour of fixing.

Step 7 — Finish installation in the browser

Open `http://<PUBLIC_IP>/` → choose language → enter Site Title, admin Username, strong Password, Email → **Install WordPress** → log in at `http://<PUBLIC_IP>/wp-admin`.

⚠ WordPress and changing IPs

WordPress stores the site URL in the database. If the public IP changes (stop/start without an Elastic IP), the site redirects to the old IP and breaks. **Attach an Elastic IP before installing** (see section 14.1), or use a domain (Chapter 9). To fix after the fact, add `define('WP_HOME','http://NEW_IP'); define('WP_SITEURL','http://NEW_IP');` to `wp-config.php`.

WordPress hardening checklist

- Strong admin password; don't use the username `admin`.
- Keep WordPress core, themes and plugins updated; delete unused ones.
- `wp-config.php` permission `640`; disable file editing in the dashboard: add `define('DISALLOW_FILE_EDIT', true);`.
- Point a domain to the server (Chapter 9) and enable HTTPS with certbot (sections 7.13 and 9.11).
- Back up the database (`mysqldump wordpress`) and `wp-content` regularly; take EBS snapshots.

- Increase PHP upload limits if needed: `upload_max_filesize` and `post_max_size` in `/etc/php.ini` (AL2023/CentOS) or `/etc/php/8.x/apache2/php.ini` / `/etc/php/8.x/fpm/php.ini` (Ubuntu), then restart PHP-FPM/Apache.

11.2 Flask application with Gunicorn + Nginx

In my sessions, students often run `python app.py` on the server and think the deployment is done. Bagha — that is only the development server. Let's do it the production way.

Why Gunicorn?

`flask run` / `app.run()` starts a **development server**: single-threaded, not secure, not meant for production. **Gunicorn** ("Green Unicorn") is a production **WSGI** server that runs several worker processes of your app. **Nginx** sits in front to handle clients, static files and TLS.

```
Client ──► Nginx :80 ─┬─ /static/*  → served directly from /opt/flaskdemo/static
                      └─ everything else → proxy_pass http://127.0.0.1:8000
                                                  │
                                    Gunicorn master (systemd service)
                                    ├── worker 1 (Flask app)
                                    ├── worker 2
                                    └── worker 3
```

Step 1 — Project structure

```
/opt/flaskdemo
├── app.py            # Flask application
├── wsgi.py           # entry point for Gunicorn
├── requirements.txt
├── templates/
│   └── index.html
├── static/
│   └── style.css
└── venv/             # virtual environment (not committed to Git)
```

Step 2 — Install and create the app

```
# AL2023 / CentOS: sudo yum install -y python3 python3-pip git nginx
# Ubuntu:          sudo apt install -y python3 python3-venv python3-pip git nginx
sudo mkdir -p /opt/flaskdemo/{templates,static} && sudo chown -R $USER:$USER /opt/flaskdemo
cd /opt/flaskdemo
python3 -m venv venv
./venv/bin/pip install flask gunicorn
./venv/bin/pip freeze > requirements.txt

cat > app.py <<'EOF'
import socket
from datetime import datetime
from flask import Flask, render_template, jsonify

app = Flask(__name__)

@app.route("/")
def home():
    return render_template("index.html", host=socket.gethostname(), now=datetime.now())
```

```
@app.route("/api/info")
def info():
    return jsonify(app="flaskdemo", host=socket.gethostname())
EOF

cat > wsgi.py <<'EOF'
from app import app

if __name__ == "__main__":
    app.run()
EOF

cat > templates/index.html <<'EOF'
<!DOCTYPE html>
<html><head><meta charset="utf-8"><title>Flask on EC2</title>
<link rel="stylesheet" href="/static/style.css"></head>
<body><div class="card"><h1>Flask + Gunicorn + Nginx</h1>
<p>Served by host <b>{{ host }}</b> at {{ now.strftime("%d-%m %H:%M:%S") }}</p></div></body></
html>
EOF

cat > static/style.css <<'EOF'
body { font-family: Arial, sans-serif; background: #eef3f9; }
.card { max-width: 600px; margin: 60px auto; background: #fff; padding: 30px; border-radius:
10px; }
EOF
```

If your code is on GitHub, replace the file-creation steps with `git clone https://github.com/<you>/<repo>.git /opt/flaskdemo` and `./venv/bin/pip install -r requirements.txt`.

Step 3 — Test Gunicorn manually

```
cd /opt/flaskdemo
./venv/bin/gunicorn --workers 3 --bind 127.0.0.1:8000 wsgi:app &
sleep 2 && curl -s http://127.0.0.1:8000/api/info
kill %1
```

`wsgi:app` means "module `wsgi.py`, object `app`". A common rule for workers is **(2 × number of CPUs) + 1**.

Step 4 — systemd service

```
sudo tee /etc/systemd/system/flaskdemo.service > /dev/null <<EOF
[Unit]
Description=Gunicorn instance serving flaskdemo
After=network.target

[Service]
User=$USER
Group=$USER
WorkingDirectory=/opt/flaskdemo
Environment="PATH=/opt/flaskdemo/venv/bin"
ExecStart=/opt/flaskdemo/venv/bin/gunicorn --workers 3 --bind 127.0.0.1:8000 --access-logfile -
wsgi:app
Restart=always

[Install]
WantedBy=multi-user.target
EOF
```

```
sudo systemctl daemon-reload
sudo service flaskdemo start
sudo systemctl enable flaskdemo
sudo service flaskdemo status
```

Step 5 — Nginx configuration

```
# AL2023 / CentOS path shown; on Ubuntu use /etc/nginx/sites-available/flaskdemo + symlink +
remove default
sudo tee /etc/nginx/conf.d/flaskdemo.conf > /dev/null <<'EOF'
server {
    listen 80;
    server_name YOUR_SERVER_NAME;

    location /static/ {
        alias /opt/flaskdemo/static/;
        expires 7d;
    }
    location / {
        proxy_pass http://127.0.0.1:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
EOF
sudo sed -i "s/YOUR_SERVER_NAME/$(curl -s https://checkip.amazonaws.com)/" /etc/nginx/conf.d/
flaskdemo.conf
sudo service nginx start
sudo systemctl enable nginx
sudo nginx -t && sudo service nginx reload
```

CentOS Stream 9 SELinux extras:

```
sudo setsebool -P httpd_can_network_connect 1
sudo semanage fcontext -a -t httpd_sys_content_t "/opt/flaskdemo/static(/.*)?"
sudo restorecon -Rv /opt/flaskdemo/static
```

Open `http://<PUBLIC_IP>/` — you should see the styled page.

> **Ravindra's Tip**
>
> Keep your application code in Git from day one, even for small college projects. Deployment then becomes `git pull` + restart, and if something breaks you can roll back instantly with `git checkout <previous-commit>`. Interviewers also like seeing a clean GitHub history.

Step 6 — Updating the application

```
cd /opt/flaskdemo
git pull                                    # if deployed from Git
./venv/bin/pip install -r requirements.txt
sudo service flaskdemo restart
journalctl -u flaskdemo -n 50 --no-pager    # check logs
```

11.3 Express application with pm2 + Nginx

Ekdum simple aahe — once you have done Flask, Express follows exactly the same idea: app on localhost, a process manager to keep it alive, Nginx in front.

Why pm2?

If you start Node with `node app.js` and close the SSH session, the app dies. If it crashes, nobody restarts it. **pm2** is a production process manager for Node.js that keeps apps alive, restarts on crash, starts them at boot, manages logs, and can run one process per CPU core (**cluster mode**) with **zero-downtime reloads**.

Step 1 — Install Node.js, pm2 and Nginx

```
# AL2023 / CentOS Stream 9
curl -fsSL https://rpm.nodesource.com/setup_22.x | sudo bash -
sudo yum install -y nodejs nginx git
# Ubuntu
curl -fsSL https://deb.nodesource.com/setup_22.x | sudo -E bash -
sudo apt install -y nodejs nginx git

sudo npm install -g pm2
```

Step 2 — Create the Express app

```
sudo mkdir -p /opt/expressdemo/public && sudo chown -R $USER:$USER /opt/expressdemo
cd /opt/expressdemo
npm init -y
npm install express

cat > server.js <<'EOF'
const express = require("express");
const os = require("os");
const app = express();
const PORT = process.env.PORT || 3000;

app.use(express.json());
app.use(express.static("public"));

const todos = [{ id: 1, task: "Learn EC2" }, { id: 2, task: "Deploy with pm2" }];

app.get("/api/todos", (req, res) => res.json(todos));
app.post("/api/todos", (req, res) => {
  const todo = { id: todos.length + 1, task: req.body.task };
  todos.push(todo);
  res.status(201).json(todo);
});
app.get("/api/health", (req, res) => res.json({ status: "ok", host: os.hostname(), pid:
process.pid }));

app.listen(PORT, "127.0.0.1", () => console.log(`Express running on 127.0.0.1:${PORT}`));
EOF

cat > public/index.html <<'EOF'
<!DOCTYPE html><html><head><meta charset="utf-8"><title>Express on EC2</title></head>
<body style="font-family:Arial;max-width:600px;margin:40px auto">
<h1>Express + pm2 + Nginx</h1><ul id="list"></ul>
```

```
<script>
fetch("/api/todos").then(r => r.json()).then(items => {
  document.getElementById("list").innerHTML = items.map(t => "<li>" + t.task + "</li>").join("");
});
</script></body></html>
EOF
```

Step 3 — pm2 ecosystem file (cluster mode)

```
cat > ecosystem.config.js <<'EOF'
module.exports = {
  apps: [{
    name: "expressdemo",
    script: "server.js",
    cwd: "/opt/expressdemo",
    instances: "max",          // one process per vCPU
    exec_mode: "cluster",
    env: { NODE_ENV: "production", PORT: 3000 },
    max_memory_restart: "300M"
  }]
};
EOF

pm2 start ecosystem.config.js
pm2 status
curl -s http://127.0.0.1:3000/api/health
pm2 startup systemd        # copy and run the 'sudo env PATH=...' line it prints
pm2 save
```

Useful pm2 commands

Command	Purpose
`pm2 list` / `pm2 status`	Show processes
`pm2 logs expressdemo`	Stream logs (`--lines 100`)
`pm2 monit`	Live CPU/memory dashboard
`pm2 restart expressdemo`	Restart (brief downtime)
`pm2 reload expressdemo`	Zero-downtime reload (cluster mode)
`pm2 stop` / `pm2 delete expressdemo`	Stop / remove from list
`pm2 save`	Save process list for resurrection at boot
`pm2 install pm2-logrotate`	Rotate logs so the disk doesn't fill up

Step 4 — Nginx reverse proxy

```
# AL2023 / CentOS path; Ubuntu: /etc/nginx/sites-available/expressdemo + symlink + remove default
sudo tee /etc/nginx/conf.d/expressdemo.conf > /dev/null <<'EOF'
upstream express_backend {
    server 127.0.0.1:3000;
    keepalive 16;
}
server {
    listen 80;
    server_name YOUR_SERVER_NAME;
```

```
    location / {
        proxy_pass http://express_backend;
        proxy_http_version 1.1;
        proxy_set_header Connection "";
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
EOF
sudo sed -i "s/YOUR_SERVER_NAME/$(curl -s https://checkip.amazonaws.com)/" /etc/nginx/conf.d/
expressdemo.conf
sudo setsebool -P httpd_can_network_connect 1 2>/dev/null   # CentOS only
sudo service nginx start
sudo systemctl enable nginx
sudo nginx -t && sudo service nginx reload
```

Browse to `http://<PUBLIC_IP>/` and `http://<PUBLIC_IP>/api/health` (refresh a few times — the `pid` changes because requests are shared across cluster workers).

Step 5 — Deploying updates

```
cd /opt/expressdemo
git pull
npm ci --omit=dev           # clean install exactly from package-lock.json
pm2 reload expressdemo      # zero downtime
```

Common Mistakes I See Students Make

- **Installing WordPress before attaching an Elastic IP** — the site breaks after stop/start.
- **Wrong ownership** of WordPress files (not `www-data`/`apache`) → plugins and uploads fail.
- **Forgetting** `AllowOverride All` **/** `a2enmod rewrite` → permalinks give 404.
- **Skipping the security salts** in `wp-config.php`.
- **Running Flask with** `app.run(debug=True)` **in production** — debug mode can allow remote code execution.
- **Wrong module path in Gunicorn** (`app:app` vs `wsgi:app`) → ModuleNotFoundError in `journalctl`.
- **Not running the command printed by** `pm2 startup`, so apps don't come back after reboot.

Interview Questions

Q1. What are the steps to host WordPress on an EC2 instance?

Install web server + PHP + MySQL/MariaDB, create DB and user, download and extract WordPress, configure `wp-config.php` with DB details and salts, set ownership/permissions, configure a vhost with URL rewriting, then complete the web installer.

Q2. Why use Nginx in front of Gunicorn?

Nginx handles slow clients, TLS, compression and static files efficiently and buffers requests, while Gunicorn focuses on running Python code.

Q3. What does `wsgi:app` mean in a Gunicorn command?

Import the module `wsgi` (wsgi.py) and use the object named `app` as the WSGI application.

Q4. How many Gunicorn workers should you run?

A common starting point is (2 × CPU cores) + 1, then tune based on load and memory.

Q5. What is the difference between `pm2 restart` and `pm2 reload`?

`restart` kills and restarts processes (brief downtime). `reload` restarts cluster workers one by one for zero downtime.

Q6. How do you make pm2 apps start after a reboot?

Run `pm2 startup` (and execute the printed command), then `pm2 save` to store the current process list.

Practice Questions and Lab Exercises

1. List the steps to install WordPress on a LAMP server.
2. Why does WordPress need `AllowOverride All` on Apache? What is the Nginx equivalent?
3. What SELinux changes are needed for WordPress on CentOS Stream 9 and why?
4. Why shouldn't you use the Flask development server in production? What does `wsgi:app` mean?
5. Explain the Nginx `location /static/` block with `alias`.
6. What is pm2 cluster mode? What is the difference between `pm2 restart` and `pm2 reload`?
7. **Lab:** Install WordPress on Ubuntu with Apache, attach an Elastic IP first, and publish a post.
8. **Lab:** Install WordPress on Amazon Linux 2023 with Nginx + PHP-FPM.
9. **Lab:** Deploy the Flask demo, then change the template, push/pull, and restart the service.
10. **Lab:** Deploy the Express demo with pm2 cluster mode; reboot the instance and prove it came back automatically.

★ Quick Revision

- WordPress = PHP + MySQL: install stack → create DB/user → download → `wp-config.php` + salts → ownership (`www-data` / `apache`) → vhost with `AllowOverride All` (or Nginx `try_files ... /index.php?$args`) → browser install. Use an Elastic IP.
- Flask: venv → Gunicorn (`wsgi:app`, 2×CPU+1 workers) → systemd → Nginx `proxy_pass` + `/static/` alias.
- Express: NodeSource → pm2 (ecosystem file, cluster mode, `startup`, `save`) → Nginx upstream + proxy.
- CentOS: `httpd_can_network_connect`, `httpd_sys_rw_content_t` for writable folders.

Chapter 12: Technology Stacks — LAMP, LEMP, MEAN, MERN

12.1 What is a "stack" and why does it matter?

Mitranno, you will see words like LAMP and MERN in almost every job posting. Let's first understand what a stack really means. A **stack** is a standard combination of technologies that work well together to build and run a web application: an operating system, a web server, a database and a programming language/framework. Companies advertise jobs as "MERN developer" or "LAMP administrator", so knowing these stacks end-to-end is valuable.

Stack	Components	Language	Database type	Typical use
LAMP	**L**inux, **A**pache, **M**ySQL/MariaDB, **P**HP	PHP	Relational (SQL)	WordPress, Moodle, classic web apps
LEMP	**L**inux, Nginx (**E**ngine-x), **M**ySQL/MariaDB, **P**HP	PHP	Relational (SQL)	High-traffic PHP sites, Laravel
MEAN	**M**ongoDB, **E**xpress, **A**ngular, **N**ode.js	JavaScript / TypeScript	Document (NoSQL)	Enterprise SPAs, dashboards
MERN	**M**ongoDB, **E**xpress, **R**eact, **N**ode.js	JavaScript	Document (NoSQL)	Startups, SPAs, social apps

12.2 LAMP stack

Full LAMP installation

Amazon Linux 2023:

```
sudo yum install -y httpd php php-fpm php-mysqlnd php-gd php-mbstring php-xml mariadb105-server
sudo service httpd start
sudo service php-fpm start
sudo service mariadb start
sudo systemctl enable httpd php-fpm mariadb
sudo mysql_secure_installation
```

Ubuntu 22.04 / 24.04:

```
sudo apt update
sudo apt install -y apache2 mysql-server php libapache2-mod-php php-mysql php-gd php-mbstring php
-xml
sudo service apache2 start
```

```
sudo service mysql start
sudo systemctl enable apache2 mysql
sudo mysql_secure_installation
```

CentOS Stream 9:

```
sudo yum install -y httpd php php-fpm php-mysqlnd php-gd php-mbstring php-xml mariadb-server
sudo service httpd start
sudo service php-fpm start
sudo service mariadb start
sudo systemctl enable httpd php-fpm mariadb
sudo mysql_secure_installation
if systemctl is-active --quiet firewalld; then sudo firewall-cmd --permanent --add-service=http &
& sudo firewall-cmd --reload; fi
```

Test the stack

```
sudo tee /var/www/html/dbtest.php > /dev/null <<'EOF'
<?php
$conn = new mysqli("127.0.0.1", "appuser", "App@Pass123", "appdb");
if ($conn->connect_error) { die("DB connection failed: " . $conn->connect_error); }
echo "<h2>LAMP/LEMP works! PHP " . phpversion() . " connected to MySQL " . $conn->server_info .
"</h2>";
EOF
curl -s http://localhost/dbtest.php
```

(Create `appuser`/`appdb` first as in Chapter 10. Delete `dbtest.php` after testing.) Deploy any PHP app by copying it to `/var/www/html` or a virtual-host folder (Chapter 7), exactly like WordPress in Chapter 11.

> **Ravindra's Tip**
>
> In interviews, don't just expand the acronym. Draw the request flow: browser → web server → language runtime → database, and mention the ports and process managers. That shows you have actually deployed the stack, not just memorised it.

12.3 LEMP stack

```
                      +------------------ EC2 (Linux) -------------------+
 Browser ──HTTP:80──► |  Nginx ──FastCGI (unix socket)──► PHP-FPM        |
                      |    static files served directly      |          |
                      |                                      ▼          |
                      |                              MySQL/MariaDB :3306  |
                      +-------------------------------------------------+
```

Full LEMP installation

Amazon Linux 2023 (default Nginx server already handles `.php` via `/etc/nginx/default.d/php.conf`):

```
sudo yum install -y nginx php php-fpm php-mysqlnd php-gd php-mbstring php-xml mariadb105-server
sudo service nginx start
sudo service php-fpm start
```

```
sudo service mariadb start
sudo systemctl enable nginx php-fpm mariadb
sudo mysql_secure_installation
echo "<?php phpinfo();" | sudo tee /usr/share/nginx/html/info.php
curl -s http://localhost/info.php | grep -m1 "PHP Version"
```

Ubuntu:

```
sudo apt update
sudo apt install -y nginx mysql-server php-fpm php-mysql php-gd php-mbstring php-xml
sudo service nginx start
sudo service mysql start
sudo systemctl enable nginx mysql
sudo mysql_secure_installation
```

Then enable PHP in the Ubuntu default site — edit `/etc/nginx/sites-available/default`: add `index.php` to the `index` line and un-comment the PHP block so it reads:

```
location ~ \.php$ {
    include snippets/fastcgi-php.conf;
    fastcgi_pass unix:/run/php/php-fpm.sock;
}
```

```
sudo nginx -t && sudo service nginx reload
echo "<?php phpinfo();" | sudo tee /var/www/html/info.php
curl -s http://localhost/info.php | grep -m1 "PHP Version"
```

CentOS Stream 9:

```
sudo yum install -y nginx php php-fpm php-mysqlnd php-gd php-mbstring php-xml mariadb-server
sudo service nginx start
sudo service php-fpm start
sudo service mariadb start
sudo systemctl enable nginx php-fpm mariadb
sudo mysql_secure_installation
echo "<?php phpinfo();" | sudo tee /usr/share/nginx/html/info.php
sudo restorecon -Rv /usr/share/nginx/html
curl -s http://localhost/info.php | grep -m1 "PHP Version"
```

Remove `info.php` after testing: `sudo rm /usr/share/nginx/html/info.php` (or `/var/www/html/info.php` on Ubuntu).

12.4 MEAN and MERN architecture

Think of a MERN app like a restaurant: the menu card in your hand is the React frontend (runs in the browser), the waiter is Nginx, the kitchen is the Express API, and the store room is MongoDB. Technically, the browser downloads static JS files, then makes HTTP calls to `/api/...`, which Nginx proxies to Node.js, which queries MongoDB.

Both stacks use **JavaScript everywhere**: the frontend (Angular or React) runs in the browser, the backend API (Express on Node.js) runs on the server, and data is stored as JSON-like documents in **MongoDB**.

```
                    +------------------------- EC2 -----------------------------+
                    |                                                           |
```

```
  Browser  ──:80──►  |  Nginx                                                  |
  (runs the SPA)     |   ├── /        → static build of React/Angular          |
                     |   │             (/var/www/mern  or  /var/www/mean)      |
                     |   │             try_files $uri /index.html              |
                     |   └── /api/*   → proxy_pass http://127.0.0.1:5000       |
                     |                    Node.js + Express API (pm2)          |
                     |                          │ Mongoose                     |
                     |                          ▼                              |
                     |                    MongoDB 127.0.0.1:27017 (mongod)     |
                    +-----------------------------------------------------------+
```

Why build the frontend? React/Angular dev servers (`npm start`, `ng serve` on ports 3000/4200) are only for development. For production we run `npm run build` / `ng build`, which produces plain HTML/CSS/JS files that Nginx serves very efficiently as a static site.

12.5 Installing MongoDB from the official repository

Dhyan do — always install MongoDB from the official MongoDB repository, not random tutorials. Type these commands with me for your OS.

The MongoDB package in some distribution repositories is outdated or missing, so we always use the **official MongoDB repository**. We use MongoDB **8.0** (a current production release series); for a different version replace `8.0` in the commands.

Amazon Linux 2023

```
sudo tee /etc/yum.repos.d/mongodb-org-8.0.repo > /dev/null <<'EOF'
[mongodb-org-8.0]
name=MongoDB Repository
baseurl=https://repo.mongodb.org/yum/amazon/2023/mongodb-org/8.0/$basearch/
gpgcheck=1
enabled=1
gpgkey=https://pgp.mongodb.com/server-8.0.asc
EOF
sudo yum install -y mongodb-mongosh-shared-openssl3
sudo yum install -y mongodb-org
sudo service mongod start
sudo systemctl enable mongod
```

Ubuntu 22.04 / 24.04

```
sudo apt install -y gnupg curl
curl -fsSL https://pgp.mongodb.com/server-8.0.asc | \
  sudo gpg -o /usr/share/keyrings/mongodb-server-8.0.gpg --dearmor
echo "deb [ arch=amd64,arm64 signed-by=/usr/share/keyrings/mongodb-server-8.0.gpg ] https://
repo.mongodb.org/apt/ubuntu $(. /etc/os-release && echo $VERSION_CODENAME)/mongodb-org/8.0
multiverse" | \
  sudo tee /etc/apt/sources.list.d/mongodb-org-8.0.list
sudo apt update
sudo apt install -y mongodb-org
sudo service mongod start
sudo systemctl enable mongod
```

(`$VERSION_CODENAME` is `jammy` on 22.04 and `noble` on 24.04.)

CentOS Stream 9 (RHEL 9 family)

```
sudo tee /etc/yum.repos.d/mongodb-org-8.0.repo > /dev/null <<'EOF'
[mongodb-org-8.0]
name=MongoDB Repository
baseurl=https://repo.mongodb.org/yum/redhat/9/mongodb-org/8.0/$basearch/
gpgcheck=1
enabled=1
gpgkey=https://pgp.mongodb.com/server-8.0.asc
EOF
sudo yum install -y mongodb-org
sudo service mongod start
sudo systemctl enable mongod
```

i MongoDB and SELinux on CentOS

With SELinux enforcing, `mongod` may log permission denials (or fail to start) because of its monitoring features. MongoDB publishes an official SELinux policy (the `mongodb-selinux` project on MongoDB's GitHub) — install it as described in the MongoDB installation docs for RHEL. For a quick lab, check `sudo service mongod status` and `sudo ausearch -m avc -ts recent`.

Verify and basic use

```
sudo service mongod status
sudo ss -tlnp | grep 27017          # should show 127.0.0.1:27017 only
mongosh --eval 'db.runCommand({ ping: 1 })'
```

Item	Value
Service	`mongod`
Config file	`/etc/mongod.conf` (YAML)
Data directory	`/var/lib/mongo` (RPM) or `/var/lib/mongodb` (Ubuntu)
Log	`/var/log/mongodb/mongod.log`
Default bind	`127.0.0.1` port `27017`
Shell	`mongosh`

```
// inside mongosh
show dbs
use notesdb
db.notes.insertOne({ text: "Hello MongoDB", createdAt: new Date() })
db.notes.find()
```

⚠ Never expose MongoDB to the internet

Keep `bindIp: 127.0.0.1` in `/etc/mongod.conf` and do **not** open 27017 in the security group. Many databases have been wiped by attackers because they were left open without authentication. For production also enable authentication: create an admin user in `mongosh` (`use admin` then `db.createUser({user:"admin", pwd:passwordPrompt(), roles:["root"]})`), add `security:` / `authorization: enabled` to `mongod.conf`, and restart `mongod`.

12.6 Common backend API for MEAN and MERN (Node + Express + MongoDB)

Install Node.js LTS and pm2 as in Chapter 11 (NodeSource `setup_22.x`, then `sudo npm install -g pm2`), plus Nginx and Git.

```
sudo mkdir -p /opt/notes-api && sudo chown $USER:$USER /opt/notes-api
cd /opt/notes-api
npm init -y
npm install express mongoose dotenv

cat > .env <<'EOF'
PORT=5000
MONGO_URL=mongodb://127.0.0.1:27017/notesdb
EOF

cat > server.js <<'EOF'
require("dotenv").config();
const express = require("express");
const mongoose = require("mongoose");

const app = express();
app.use(express.json());

mongoose.connect(process.env.MONGO_URL)
  .then(() => console.log("MongoDB connected"))
  .catch((err) => { console.error("MongoDB error:", err.message); process.exit(1); });

const Note = mongoose.model("Note",
  new mongoose.Schema({ text: { type: String, required: true } }, { timestamps: true }));

app.get("/api/health", (req, res) => res.json({ status: "ok" }));

app.get("/api/notes", async (req, res) => {
  res.json(await Note.find().sort({ createdAt: -1 }));
});

app.post("/api/notes", async (req, res) => {
  try {
    res.status(201).json(await Note.create({ text: req.body.text }));
  } catch (err) {
    res.status(400).json({ error: err.message });
  }
});

app.delete("/api/notes/:id", async (req, res) => {
  await Note.findByIdAndDelete(req.params.id);
  res.status(204).end();
});

const port = process.env.PORT || 5000;
app.listen(port, "127.0.0.1", () => console.log(`API on 127.0.0.1:${port}`));
EOF

pm2 start server.js --name notes-api
pm2 save
curl -s -X POST http://127.0.0.1:5000/api/notes -H "Content-Type: application/json" -d '{"text":"
First note"}'
curl -s http://127.0.0.1:5000/api/notes
```

(Run `pm2 startup systemd` once and execute the printed command so pm2 starts at boot.)

> **✔ Low memory? Add swap before building frontends**
>
> `npm install` and production builds of React/Angular can need more than the 1 GiB RAM of a `t3.micro`. Add a 2 GiB swap file first: `sudo dd if=/dev/zero of=/swapfile bs=1M count=2048 && sudo chmod 600 /swapfile && sudo mkswap /swapfile && sudo swapon /swapfile` (add `/swapfile none swap sw 0 0` to `/etc/fstab` to keep it after reboot). Or build on your laptop and upload only the build folder with `scp`.

12.7 MERN: React frontend served by Nginx

Chala, aata frontend build karuya. Bagha how the React code becomes simple static files that Nginx can serve.

Build the React app (Vite)

```
cd /opt
sudo mkdir -p /opt/mern-frontend && sudo chown $USER:$USER /opt/mern-frontend
cd /opt/mern-frontend
npm create vite@latest . -- --template react
npm install
```

(If `create vite` asks any questions, accept the defaults and do **not** start the dev server.)

Replace `src/App.jsx` with a small notes UI that talks to `/api`:

```
cat > src/App.jsx <<'EOF'
import { useEffect, useState } from "react";

export default function App() {
  const [notes, setNotes] = useState([]);
  const [text, setText] = useState("");

  const load = () => fetch("/api/notes").then((r) => r.json()).then(setNotes);
  useEffect(() => { load(); }, []);

  const add = async (e) => {
    e.preventDefault();
    if (!text.trim()) return;
    await fetch("/api/notes", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ text }),
    });
    setText("");
    load();
  };

  return (
    <div style={{ fontFamily: "Arial", maxWidth: 600, margin: "40px auto" }}>
      <h1>MERN Notes on AWS EC2</h1>
      <form onSubmit={add}>
        <input value={text} onChange={(e) => setText(e.target.value)} placeholder="Write a
note" />
        <button type="submit">Add</button>
      </form>
      <ul>{notes.map((n) => <li key={n._id}>{n.text}</li>)}</ul>
    </div>
```

```
  );
}
EOF

npm run build            # creates the dist/ folder
sudo mkdir -p /var/www/mern
sudo cp -r dist/* /var/www/mern/
sudo restorecon -Rv /var/www/mern 2>/dev/null     # CentOS only
```

(With Create React App the output folder is `build/` instead of `dist/`.)

Nginx configuration for MERN

```
# AL2023 / CentOS: /etc/nginx/conf.d/mern.conf
# Ubuntu: save as /etc/nginx/sites-available/mern, symlink into sites-enabled, remove 'default',
#         and use "listen 80 default_server;" + "server_name _;"
sudo tee /etc/nginx/conf.d/mern.conf > /dev/null <<'EOF'
server {
    listen 80;
    server_name YOUR_SERVER_NAME;
    root /var/www/mern;
    index index.html;

    # Single Page App: unknown paths go to index.html (client-side routing)
    location / {
        try_files $uri $uri/ /index.html;
    }

    # API requests go to the Node/Express backend
    location /api/ {
        proxy_pass http://127.0.0.1:5000;
        proxy_http_version 1.1;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }

    # Cache hashed static assets
    location /assets/ {
        expires 30d;
        add_header Cache-Control "public, immutable";
    }
}
EOF
sudo sed -i "s/YOUR_SERVER_NAME/$(curl -s https://checkip.amazonaws.com)/" /etc/nginx/conf.d/
mern.conf
sudo setsebool -P httpd_can_network_connect 1 2>/dev/null     # CentOS only
sudo nginx -t && sudo service nginx reload
```

Open `http://<PUBLIC_IP>/`, add notes, refresh — they persist in MongoDB.

> **Ravindra's Tip**
>
> On a small `t3.micro`, build your React/Angular app on your laptop (`npm run build`) and upload only the `dist/` folder with `scp`. Builds need a lot of RAM and can freeze a 1 GiB instance. The server should only serve files, not build them — this is also how CI/CD pipelines work in companies.

12.8 MEAN: Angular frontend served by Nginx

The backend (Express + MongoDB on port 5000) is exactly the same as in 12.6.

Build the Angular app

```
export NG_CLI_ANALYTICS=false
sudo mkdir -p /opt/mean-frontend && sudo chown $USER:$USER /opt/mean-frontend
cd /opt/mean-frontend
npx -y @angular/cli@latest new frontend --defaults --ssr=false --skip-git --interactive=false
cd frontend
ls src/app/
```

Replace the root component. In **Angular 20 and newer** the file is `src/app/app.ts` (class `App`); in **Angular 17–19** it is `src/app/app.component.ts` (class `AppComponent` — change the class name in the code below to match). This version uses **signals**, so it works whether or not the project uses zone.js:

```
cat > src/app/app.ts <<'EOF'
import { Component, OnInit, signal } from '@angular/core';

interface Note { _id: string; text: string; }

@Component({
  selector: 'app-root',
  standalone: true,
  template: `
    <div style="font-family:Arial;max-width:600px;margin:40px auto">
      <h1>MEAN Notes on AWS EC2</h1>
      <input #box placeholder="Write a note">
      <button (click)="add(box.value); box.value = ''">Add</button>
      <ul>
        @for (n of notes(); track n._id) { <li>{{ n.text }}</li> }
      </ul>
    </div>
  `,
})
export class App implements OnInit {
  notes = signal<Note[]>([]);

  async ngOnInit() { await this.load(); }

  async load() {
    const res = await fetch('/api/notes');
    this.notes.set(await res.json());
  }

  async add(text: string) {
    if (!text.trim()) return;
    await fetch('/api/notes', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ text }),
    });
    await this.load();
  }
}
EOF
```

```
npx ng build                                    # output: dist/frontend/browser/
sudo mkdir -p /var/www/mean
sudo cp -r dist/frontend/browser/* /var/www/mean/
sudo restorecon -Rv /var/www/mean 2>/dev/null   # CentOS only
```

> **i Angular output folder**
>
> Angular 17+ puts the production build in `dist/<project-name>/browser/`. Older versions used `dist/<project-name>/`. Check with `ls dist/*`.

Nginx configuration for MEAN

Identical to MERN — only the root folder changes:

```
sudo sed 's#/var/www/mern#/var/www/mean#' /etc/nginx/conf.d/mern.conf | sudo tee /etc/nginx/
conf.d/mean.conf > /dev/null
sudo rm /etc/nginx/conf.d/mern.conf           # keep only one of them for the same server_name
sudo nginx -t && sudo service nginx reload
```

(Angular's built assets are not in `/assets/`, so that caching block simply does nothing — harmless.)

12.9 Stack comparison summary

Point	LAMP	LEMP	MEAN	MERN
Web server	Apache	Nginx	Nginx (reverse proxy)	Nginx (reverse proxy)
Backend	PHP	PHP-FPM	Node.js + Express	Node.js + Express
Frontend	Server-rendered PHP/ HTML	Server-rendered PHP/ HTML	Angular SPA (TypeScript)	React SPA (JSX)
Database	MySQL/MariaDB	MySQL/MariaDB	MongoDB	MongoDB
Process manager	systemd (httpd/ apache2)	systemd (php-fpm)	pm2 / systemd	pm2 / systemd
Ports used internally	80, 3306	80, FPM socket, 3306	80, 5000, 27017	80, 5000, 27017
Security group opens	22, 80, 443	22, 80, 443	22, 80, 443	22, 80, 443

Common Mistakes I See Students Make

- **Running `npm start` / `ng serve` on the server** and opening port 3000/4200 to the world instead of building and serving with Nginx.
- **Missing `try_files $uri $uri/ /index.html;`** → refreshing a React/Angular route gives 404.
- **Copying the whole project folder** (with `node_modules`) instead of only the `dist/` build output.

- **Opening port 27017** in the security group or changing `bindIp` to `0.0.0.0` without authentication.
- **Installing MongoDB from an outdated distro package** or mixing repository versions.
- **Building on a 1 GiB instance without swap** → the build is killed (out of memory).

Interview Questions

Q1. Expand LAMP, LEMP, MEAN and MERN.

LAMP: Linux, Apache, MySQL, PHP. LEMP: Linux, Nginx, MySQL, PHP. MEAN: MongoDB, Express, Angular, Node.js. MERN: MongoDB, Express, React, Node.js.

Q2. What is the difference between LAMP and LEMP?

LEMP replaces Apache with Nginx; PHP then runs in PHP-FPM, connected via FastCGI, instead of Apache's `mod_php`.

Q3. How is a React app deployed to production?

Run `npm run build` to produce static files, copy them to a web root, serve with Nginx using `try_files ... /index.html`, and proxy `/api` to the backend.

Q4. Why is MongoDB bound to 127.0.0.1 by default?

To prevent remote access; exposing it without authentication has led to many data breaches.

Q5. What is the role of Express in MERN?

Express is the Node.js web framework that implements the REST API — routing, request parsing and database access through a driver like Mongoose.

Q6. SQL vs NoSQL — when would you choose MongoDB?

MongoDB (document store) suits flexible or evolving schemas and JSON-like data; relational databases suit structured data with strong relationships and transactions.

Practice Questions and Lab Exercises

1. Expand LAMP, LEMP, MEAN and MERN. Draw the architecture diagram of any two.
2. Differentiate LAMP and LEMP. Why is PHP-FPM needed in LEMP?
3. Why do we build React/Angular apps before deployment instead of running `npm start`?
4. Explain `try_files $uri $uri/ /index.html;` in a single-page app.
5. Write the steps to install MongoDB from the official repository on Ubuntu 24.04.
6. Which ports must be open in the security group for a MERN app? Which must never be open?
7. **Lab:** Build a LAMP server on Amazon Linux 2023 and run `dbtest.php`.
8. **Lab:** Build a LEMP server on Ubuntu and show `phpinfo()`.
9. **Lab:** Deploy the MERN notes app on Ubuntu 24.04. Reboot and verify that MongoDB, the API (pm2) and Nginx all come back.
10. **Lab:** Deploy the MEAN notes app on Amazon Linux 2023.

★ Quick Revision

- LAMP = Linux+Apache+MySQL+PHP; LEMP = Nginx+PHP-FPM instead of Apache.
- MEAN/MERN = MongoDB + Express + Angular/React + Node.js — JavaScript everywhere.
- MongoDB: official repo (`mongodb-org`), service `mongod`, port 27017, keep `bindIp 127.0.0.1`.
- SPA deploy: `npm run build` / `ng build` → copy `dist/` to `/var/www/...` → Nginx `try_files ... /index.html` + `location /api/ { proxy_pass http://127.0.0.1:5000; }`.
- API runs with pm2 (`start`, `save`, `startup`).

Chapter 13: Amazon EBS (Elastic Block Store)

13.1 What and why

Mitranno, sooner or later every server runs out of disk space — usually at the worst possible time. This chapter teaches you how to handle storage calmly. Let's first understand what EBS is. **Amazon EBS** provides **block-level storage volumes** — virtual hard disks — that you attach to EC2 instances. The root disk of almost every EC2 instance is an EBS volume.

Why EBS?

- **Persistent:** data survives instance stop/start (unlike instance store, which is lost).
- **Flexible:** increase size, change type or performance **while the volume is in use**.
- **Durable:** automatically replicated within its Availability Zone.
- **Backups:** point-in-time **snapshots** stored in Amazon S3 (managed by AWS), copyable across regions.
- **Encryption:** one tick box (AWS KMS) encrypts data at rest, in transit to the instance and in snapshots.

i Key rules

- An EBS volume lives in **one Availability Zone** and can only attach to instances **in the same AZ**. To move it to another AZ/region, take a snapshot and create a new volume from it.
- Normally one volume attaches to one instance at a time (io1/io2 support Multi-Attach for special clustered workloads).
- You pay for the **provisioned size** (GiB-month), even if the disk is empty and even while the instance is stopped.

13.2 EBS volume types

Type	Category	Size	Max IOPS / volume	Max throughput	Use case
gp3	General purpose SSD	1 GiB – 16 TiB (newer limits allow larger)	3,000 baseline included; more can be provisioned independently of size	125 MiB/s baseline; more can be provisioned	**Default choice**: boot volumes, web/app servers, dev/test, small DBs
gp2	General purpose SSD (previous generation)	1 GiB – 16 TiB	3 IOPS per GiB (min 100, burst to 3,000, max 16,000)	up to 250 MiB/s	Legacy; migrate to gp3 (usually ~20% cheaper)
io2 Block Express	Provisioned IOPS SSD	4 GiB – 64 TiB	up to 256,000	up to 4,000 MiB/s	Mission-critical databases (Oracle, SAP HANA, large MySQL) needing consistent low latency; 99.999% durability
io1	Provisioned IOPS SSD (older)	4 GiB – 16 TiB	up to 64,000	up to 1,000 MiB/s	Legacy high-IOPS workloads
st1	Throughput optimised HDD	125 GiB – 16 TiB	500	500 MiB/s	Big data, log processing, data warehouses (large sequential reads) — **cannot be boot volume**
sc1	Cold HDD	125 GiB – 16 TiB	250	250 MiB/s	Infrequently accessed archives, lowest cost — **cannot be boot volume**

✔ gp3 vs gp2

With **gp2**, performance depends on size (a 100 GiB gp2 gets only 300 baseline IOPS). With **gp3**, every volume gets 3,000 IOPS and 125 MiB/s regardless of size, and you can buy more IOPS/throughput independently. Converting gp2 → gp3 is an online Modify volume operation with no downtime.

IOPS = input/output operations per second (small random reads/writes — databases). **Throughput** = MiB per second (large sequential transfers — logs, video, backups).

RB Ravindra's Tip

For almost every lab and small project, choose **gp3**. It gives 3,000 IOPS even on a small volume and usually costs less than gp2. Only move to io2 when a real database benchmark proves you need it — don't over-engineer.

13.3 Snapshots

A **snapshot** is a point-in-time backup of an EBS volume.

- **Incremental:** the first snapshot copies all used blocks; later ones store only changed blocks (cheaper), yet each snapshot can restore the full volume.
- **Regional:** a snapshot can create volumes in any AZ of the region; you can **copy** it to another region (disaster recovery) and **share** it with other accounts.
- **AMIs** are built from snapshots of the root volume.
- **Automation:** Amazon Data Lifecycle Manager (DLM) or AWS Backup can create and delete snapshots on a schedule.
- **Consistency:** for databases, stop writes (or stop the instance / flush tables) before taking a snapshot to get an application-consistent backup.

Console: EC2 → Elastic Block Store → Volumes → select volume → **Actions → Create snapshot** → description/tag → **Create snapshot**. Restore: Snapshots → select → **Actions → Create volume from snapshot** → choose AZ (same as target instance), type, size.

AWS CLI equivalents (optional):

```
aws ec2 create-snapshot --volume-id vol-0123456789abcdef0 --description "before-upgrade"
aws ec2 describe-snapshots --owner-ids self --query "Snapshots[].
[SnapshotId,VolumeId,StartTime,State]" --output table
aws ec2 create-volume --snapshot-id snap-0123456789abcdef0 --availability-zone ap-south-1a --
volume-type gp3
```

13.4 NVMe device naming (important!)

On modern **Nitro-based** instances (t3, t4g, m5, m6i, m7i, c5, c7g, r6i ...), EBS volumes appear as **NVMe** devices, not with the name you chose in the console.

Console "device name"	Older Xen instances (t2, m4)	Nitro instances (t3 and newer)
`/dev/xvda` or `/dev/sda1` (root)	`/dev/xvda`	`/dev/nvme0n1`
`/dev/sdf` (extra volume)	`/dev/xvdf`	`/dev/nvme1n1` (number depends on attach order)
Partition 1 of root	`/dev/xvda1`	`/dev/nvme0n1p1`

To map an NVMe device to its EBS volume ID:

```
lsblk -o NAME,SIZE,TYPE,FSTYPE,MOUNTPOINT,SERIAL     # SERIAL shows vol0123456789abcdef0
ls -l /dev/disk/by-id/ | grep -i nvme-Amazon
sudo nvme list                                       # package nvme-cli
```

> ⚠ **NVMe names can change**
>
> `/dev/nvme1n1` today could become `/dev/nvme2n1` after a reboot or re-attach. That is why `/etc/fstab` must use the filesystem **UUID**, never the device name. On Amazon Linux, symlinks like `/dev/sdf` → `nvme1n1` are also created for convenience.

13.5 Increasing the size of a volume (root or data)

In my sessions, students often ask: "Sir, I increased the size in the console, why does *df -h* still show 8G?" Bagha — the disk grew, but the partition and filesystem inside Linux did not. Samjla ka? Let's fix it in two parts.

You can grow a volume **without stopping the instance**. It is a two-part job: **(A)** make the EBS volume bigger in AWS, then **(B)** make the partition and filesystem inside Linux use the new space.

```
  Before: [ EBS 8 GiB  ][ partition 8 GiB ][ filesystem 8 GiB ]
  Step A: [ EBS 20 GiB                  ][ partition 8 GiB ][ filesystem 8 GiB ]   ← lsblk shows
20G disk, df still 8G
  Step B1 growpart:  [ partition 20 GiB ]
  Step B2 xfs_growfs / resize2fs: [ filesystem 20 GiB ]                             ← df now
shows 20G
```

> ✖ **Take a snapshot first**
>
> Always create a snapshot of the volume before resizing. Also note: EBS volumes can only be made **bigger**, never smaller, and after a modification you must wait (usually up to 6 hours) before modifying the same volume again.

Part A — Modify the volume in the console

1. EC2 → Instances → select instance → Storage tab → click the **Volume ID**.
2. Select the volume → **Actions → Modify volume**.
3. Enter the new **Size** (e.g. 20 GiB); optionally change type to gp3 → **Modify** → confirm.
4. Wait until the Volume state shows **in-use – optimizing** or **completed** (the new size is usable as soon as it enters optimizing).

CLI alternative: `aws ec2 modify-volume --volume-id vol-0123456789abcdef0 --size 20`

Part B — Extend partition and filesystem in Linux

Step 1 — check the current layout:

```
lsblk
df -hT
```

Example output on a Nitro instance after Part A (root volume grown from 8 to 20 GiB):

```
NAME          MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
nvme0n1       259:0    0   20G  0 disk
├─nvme0n1p1   259:1    0    8G  0 part /
```

```
├─nvme0n1p127 259:2    0    1M  0 part
└─nvme0n1p128 259:3    0   10M  0 part /boot/efi

Filesystem     Type  Size  Used Avail Use% Mounted on
/dev/nvme0n1p1 xfs   8.0G  1.9G  6.1G  24% /
```

The disk is 20G but partition `nvme0n1p1` and the filesystem are still 8G.

Step 2 — grow the partition (note the **space** between disk and partition number):

```
# growpart is in package cloud-utils-growpart (AL2023/CentOS) or cloud-guest-utils (Ubuntu) - usually pre-installed
sudo growpart /dev/nvme0n1 1          # Nitro instance
# sudo growpart /dev/xvda 1           # older Xen instance (t2)
lsblk                                  # nvme0n1p1 should now be 20G
```

Step 3 — grow the filesystem. First check the type with `df -hT`:

```
# XFS (default on Amazon Linux 2023, Amazon Linux 2 and CentOS Stream 9) - give the MOUNT POINT
sudo xfs_growfs -d /

# ext4 (default on Ubuntu) - give the PARTITION DEVICE
sudo resize2fs /dev/nvme0n1p1         # Ubuntu root is usually /dev/nvme0n1p1 (or /dev/xvda1 on Xen)
```

Step 4 — verify:

```
df -hT /
lsblk
```

> **i Data volume without a partition**
>
> If you formatted a whole disk directly (e.g. `/dev/nvme1n1` with no `p1` partition, as we do in 13.6), **skip `growpart`**. Just run `sudo xfs_growfs -d /data` (XFS) or `sudo resize2fs /dev/nvme1n1` (ext4) after modifying the volume.

> **✓ Ubuntu often auto-grows the root on reboot**
>
> cloud-init on Ubuntu and Amazon Linux grows the root partition automatically at boot. If you increased the root volume and rebooted, `df -h` may already show the new size — the manual steps above are for doing it **live** without a reboot.

13.6 Adding a new EBS volume

Chala, aata ek navin disk add karuya (let's add a new disk). Think of it like plugging a new external hard disk into your laptop — but technically, AWS attaches a network block device that appears as an NVMe disk, and you must format and mount it yourself.

Goal: add a 10 GiB gp3 disk to an instance and mount it permanently at `/data`.

Step 1 — Find the instance's Availability Zone

EC2 → Instances → select instance → Networking tab → **Availability zone** (e.g. `ap-south-1a`). Or on the instance:

```
TOKEN=$(curl -s -X PUT http://169.254.169.254/latest/api/token -H "X-aws-ec2-metadata-token-ttl-seconds: 60")
curl -s -H "X-aws-ec2-metadata-token: $TOKEN" http://169.254.169.254/latest/meta-data/placement/availability-zone; echo
```

Step 2 — Create the volume (same AZ!)

EC2 → Elastic Block Store → Volumes → **Create volume** → Volume type **gp3** → Size **10 GiB** → Availability Zone **same as the instance** → (optional) tick Encrypt this volume → add tag `Name=data-vol` → **Create volume**. Wait until state is **Available**.

Step 3 — Attach it

Select the volume → **Actions → Attach volume** → choose your instance → Device name `/dev/sdf` → **Attach volume**. State becomes **In-use**.

Step 4 — Identify the new disk

```
lsblk
```

```
NAME          MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
nvme0n1       259:0    0   20G  0 disk
└─nvme0n1p1   259:1    0   20G  0 part /
nvme1n1       259:4    0   10G  0 disk                ← new, no filesystem, not mounted
```

Make sure it is empty (a brand-new volume shows `data` or nothing, **not** a filesystem):

```
sudo file -s /dev/nvme1n1        # "/dev/nvme1n1: data" means empty
lsblk -f /dev/nvme1n1
```

> **✖ mkfs erases everything**
>
> Formatting a disk that already contains data (for example a volume restored from a snapshot) **destroys** that data. Only run `mkfs` on a new empty volume, and double-check the device name.

Step 5 — Create a filesystem

```
# XFS (recommended on Amazon Linux / CentOS; install xfsprogs on Ubuntu if you prefer XFS)
sudo mkfs -t xfs /dev/nvme1n1

# OR ext4 (common on Ubuntu)
# sudo mkfs -t ext4 /dev/nvme1n1
```

Step 6 — Create a mount point and mount

```
sudo mkdir -p /data
sudo mount /dev/nvme1n1 /data
df -hT /data
```

Step 7 — Make the mount permanent with /etc/fstab (UUID + nofail)

A plain `mount` is forgotten after reboot. Add an entry to `/etc/fstab` using the **UUID**:

```
sudo blkid /dev/nvme1n1
# /dev/nvme1n1: UUID="8a5c6f1e-2b3d-4c5e-9f10-1a2b3c4d5e6f" BLOCK_SIZE="512" TYPE="xfs"

sudo cp /etc/fstab /etc/fstab.bak           # always back up first
UUID=$(sudo blkid -s UUID -o value /dev/nvme1n1)
FSTYPE=$(sudo blkid -s TYPE -o value /dev/nvme1n1)
echo "UUID=$UUID  /data  $FSTYPE  defaults,nofail  0  2" | sudo tee -a /etc/fstab
cat /etc/fstab
```

Test the fstab entry before rebooting:

```
sudo umount /data
sudo mount -a                 # mounts everything in fstab; any error here means fix fstab NOW
sudo systemctl daemon-reload
df -hT /data
```

fstab field	Value	Meaning
1 device	`UUID=...`	Stable identifier — survives NVMe renaming
2 mount point	`/data`	Folder where the disk appears
3 type	`xfs` / `ext4`	Filesystem type
4 options	`defaults,nofail`	`nofail` = boot continues even if the volume is missing/detached
5 dump	`0`	Legacy backup flag
6 fsck order	`2`	Check after root (root = 1; 0 = never check)

> ⚠ **Why nofail matters**
>
> Without `nofail`, if the volume is detached or fails, the instance can hang in **emergency mode** at boot and you cannot SSH in. A wrong UUID has the same effect — that is why we test with `sudo mount -a` before rebooting.

Step 8 — Use it

```
sudo chown $USER:$USER /data
echo "hello EBS" > /data/test.txt
sudo reboot
# after reconnecting:
df -hT /data && cat /data/test.txt
```

Common uses: move MySQL data (`/var/lib/mysql`), website files, logs or backups onto a separate volume so they survive root-volume problems and can be snapshotted separately.

RB Ravindra's Tip

Treat `/etc/fstab` with respect: back it up, use the UUID, add `nofail`, and always run `sudo mount -a` before rebooting. A single typo in fstab can make an instance unbootable — and then you need a rescue instance to fix it. Lakshat theva!

13.7 Detaching a volume safely

Detaching a mounted volume can corrupt data. Always unmount inside Linux first.

```
# 1. Stop anything using the disk
sudo lsof +f -- /data            # list open files on /data (install lsof if missing)
cd ~                             # make sure your shell is not inside /data

# 2. Unmount
sudo umount /data                # "target is busy" = something still uses it

# 3. Remove or comment the fstab line so boot does not look for it
sudo sed -i.bak '\#[[:space:]]/data[[:space:]]#s/^/#/' /etc/fstab
cat /etc/fstab
```

1. Console: Volumes → select → **Actions → Detach volume** → confirm. State becomes **Available**.
2. If you no longer need it: take a snapshot (optional) → **Actions → Delete volume** (stops billing).

✔ Force detach

Force detach is only for emergencies when a normal detach is stuck; it can cause data loss or filesystem corruption.

i More on snapshots and AMIs

Hands-on snapshot restores, cross-region copies, AMIs and Data Lifecycle Manager automation are covered step by step in **Chapter 14**.

13.8 EBS best practices

- Use **gp3** by default; convert old gp2 volumes to gp3.
- Keep data on **separate volumes** from the OS; use UUID + `nofail` in fstab.
- **Encrypt** volumes (set EBS encryption by default in EC2 settings for the region).
- Automate **snapshots** with Data Lifecycle Manager / AWS Backup, and test restores.
- Monitor free space (`df -h`) and CloudWatch metrics; grow before the disk is full.
- Delete **unattached volumes** and old snapshots to save money.
- Check Delete on termination settings: root volumes default to deleted, extra volumes default to kept.

Common Mistakes I See Students Make

- **Creating the volume in a different AZ** from the instance — it simply won't appear in the attach list.
- **Running `mkfs` on a volume that already has data** (e.g. restored from a snapshot) and wiping it.
- **Using `/dev/sdf` or `/dev/nvme1n1` in fstab** instead of the UUID.
- **Forgetting `nofail`** and the instance hangs at boot when the volume is missing.
- **Running `resize2fs` on an XFS filesystem** (use `xfs_growfs`) or giving `xfs_growfs` a device instead of the mount point.
- **Forgetting the space in `growpart /dev/nvme0n1 1`.**
- **Detaching without unmounting**, risking data corruption.
- **Leaving unattached volumes and old snapshots** that keep costing money.

Interview Questions

Q1. What is the difference between EBS and instance store?

EBS is persistent network-attached storage that survives stop/start and can be snapshotted. Instance store is temporary disk on the physical host; data is lost when the instance stops or terminates.

Q2. Can you attach an EBS volume to an instance in another AZ?

No. Create a snapshot, then create a new volume from it in the target AZ (or copy the snapshot to another region).

Q3. Compare gp3 and gp2.

gp2 performance scales with size (3 IOPS/GiB, burst to 3,000). gp3 gives a 3,000 IOPS / 125 MiB/s baseline regardless of size, lets you provision more independently, and is usually cheaper.

Q4. How do you increase the root volume size without downtime?

Modify the volume in the console, then `sudo growpart /dev/nvme0n1 1`, then `sudo xfs_growfs -d /` (XFS) or `sudo resize2fs /dev/nvme0n1p1` (ext4); verify with `df -hT`.

Q5. Why do we use UUID and `nofail` in /etc/fstab?

NVMe device names can change between boots; the UUID is stable. `nofail` lets the system boot even if the volume is missing.

Q6. Are EBS snapshots full or incremental?

Incremental — only changed blocks after the first snapshot are stored, but each snapshot can restore the full volume.

Q7. What are st1 and sc1 used for?

HDD volumes for large sequential workloads: st1 for throughput-heavy data like logs/big data, sc1 for cold, rarely accessed data. Neither can be a boot volume.

Practice Questions and Lab Exercises

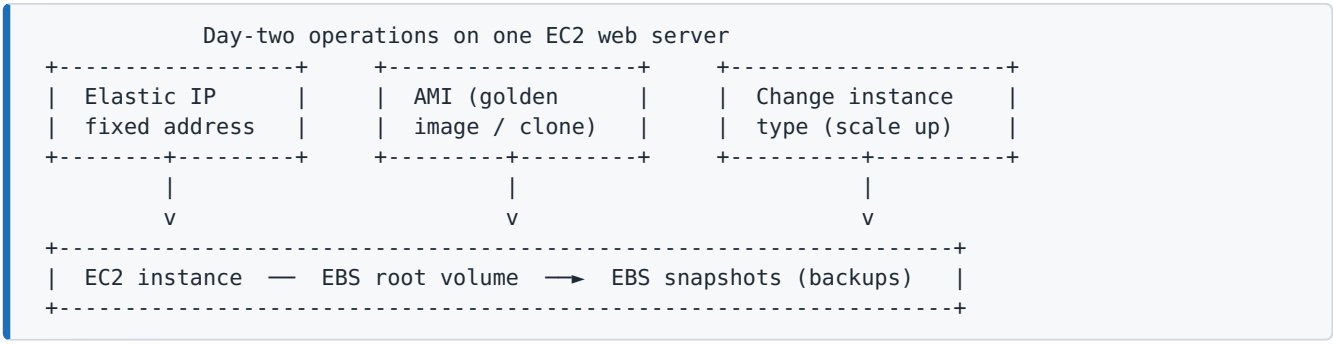
1. What is EBS? How is it different from instance store?
2. Compare gp3, gp2, io2, st1 and sc1 volume types with one use case each. Which cannot be used as boot volumes?
3. What is an incremental snapshot? How do you move an EBS volume to another AZ?
4. Why do device names differ between the console (`/dev/sdf`) and Linux (`/dev/nvme1n1`)?
5. List the commands to grow an XFS root volume and an ext4 root volume after modifying the size.
6. Why do we use UUID and `nofail` in `/etc/fstab`? How do you test fstab before rebooting?
7. **Lab:** Increase your root volume from 8 GiB to 12 GiB on a running Amazon Linux 2023 instance and verify with `df -hT`.
8. **Lab:** Repeat on Ubuntu (ext4) using `resize2fs`.
9. **Lab:** Create a 5 GiB gp3 volume, attach, format as XFS, mount at `/data` with a permanent fstab entry, reboot and verify.
10. **Lab:** Take a snapshot of the data volume, create a new volume from it in another AZ, attach it to another instance and read your file (do **not** run `mkfs`!).
11. **Lab:** Detach the volume safely and delete it.

★ Quick Revision

- EBS = persistent network block storage in **one AZ**; snapshots are regional & incremental.
- Types: **gp3** (default SSD, 3,000 IOPS baseline), gp2 (older, size-based IOPS), io2 (provisioned high IOPS), st1 (throughput HDD), sc1 (cold HDD).
- Nitro: `/dev/sdf` appears as `/dev/nvme1n1`; use UUIDs.
- Grow: console Modify volume → `growpart /dev/nvme0n1 1` → `xfs_growfs -d /` (XFS) or `resize2fs /dev/nvme0n1p1` (ext4) → `df -hT`.
- Add: create in same AZ → attach → `lsblk` → `file -s` → `mkfs` → `mkdir` → `mount` → fstab `UUID=... /data xfs defaults,nofail 0 2` → `mount -a`.
- Detach: stop users → `umount` → remove fstab line → detach in console.

Chapter 14: EC2 Operations — Elastic IP, AMI, Snapshots and Vertical Scaling

Mitranno, launching a server is only day one. In real projects, the day-two work matters more: keeping the same IP when you replace a server, taking a backup before an upgrade, cloning a perfectly configured server in minutes, and giving a slow server more CPU and RAM. Ekdum practical chapter aahe — let's learn these operations one by one, and please do every lab with me.



14.1 Elastic IP in depth

Why do we need it?

Remember Chapter 1 and 5: the **auto-assigned public IPv4 address** is released when you stop the instance, and a **new** one is given when you start it. Your DNS record, your WordPress site URL, your friends' bookmarks — all point to the old IP and break. Bagha, an **Elastic IP (EIP)** solves this: it is a static public IPv4 address that belongs to **your account** (not to an instance) until you release it.

Think of it like a landline number that you keep even when you shift to a new flat — you just "port" it to the new address. Technically, AWS keeps a 1:1 NAT mapping at the Internet Gateway from the EIP to the private IP of the instance (its network interface), and you can change that mapping at any time.

Point	Auto-assigned public IP	Elastic IP
Survives stop/start	No — new IP after start	**Yes**
Owned by	The instance (temporarily)	Your AWS account
Move to another instance	Not possible	Yes — re-associate in seconds
Default limit	—	**5 per region** per account (a service quota; you can request more)
Charges	Public IPv4 is charged hourly	Public IPv4 is charged hourly — **whether attached or idle**

> ⚠ **Public IPv4 addresses cost money**
>
> AWS charges an hourly fee for every public IPv4 address, including Elastic IPs — attached to a running instance, attached to a stopped instance, or just sitting unassociated in your account. Check the current rate on the Amazon VPC pricing page. Lakshat theva: **release** EIPs you no longer need.

Allocate and associate (console)

Step 1 — Allocate: EC2 → Network & Security → Elastic IPs → Allocate Elastic IP address → Network border group: your region (e.g. `ap-south-1`) → Public IPv4 address pool: Amazon's pool of IPv4 addresses → add tag `Name=web-eip` → **Allocate**.

Step 2 — Associate: select the EIP → **Actions → Associate Elastic IP address** → Resource type **Instance** → choose the instance (and its private IP) → **Associate**.

Step 3 — Verify: the instance's Public IPv4 address now shows the EIP. From the server:

```
curl -s https://checkip.amazonaws.com      # should print the Elastic IP
```

> **i The old public IP is gone**
>
> When you associate an EIP, the instance's previous auto-assigned public IP is released immediately. Reconnect SSH using the new Elastic IP. If you later disassociate the EIP, the instance gets a fresh auto-assigned IP (if its subnet setting allows) — not the old one.

Server replacement with zero DNS change

This is the most useful real-world trick with Elastic IPs. Suppose `web-1` is old and you have built a new, upgraded `web-2`:

```
Before:  www.example.com ──► EIP 13.232.50.10 ──► web-1 (old)
After:   www.example.com ──► EIP 13.232.50.10 ──► web-2 (new)      DNS record unchanged!
```

1. Build and test `web-2` using its own temporary public IP (or an AMI of `web-1`, see 14.2).
2. Select the EIP → **Actions → Associate Elastic IP address** → choose `web-2` → tick **Allow this Elastic IP address to be reassociated** → **Associate**.
3. Traffic now goes to `web-2` within seconds. Keep `web-1` stopped for a day as a rollback option, then terminate it.

Disassociate and release

- **Disassociate:** select the EIP → **Actions → Disassociate Elastic IP address**. The EIP stays in your account (still charged).
- **Release:** select the EIP → **Actions → Release Elastic IP addresses → Release**. The address goes back to AWS; you usually cannot get the same IP again.

AWS CLI equivalents (optional)

```
aws ec2 allocate-address --domain vpc
aws ec2 associate-address --instance-id i-0123456789abcdef0 --allocation-id eipalloc-0abc1234def5
67890
aws ec2 associate-address --instance-id i-0fedcba9876543210 --allocation-id eipalloc-0abc1234def5
67890 --allow-reassociation
aws ec2 describe-addresses --output table
aws ec2 disassociate-address --association-id eipassoc-0123abcd4567ef890
aws ec2 release-address --allocation-id eipalloc-0abc1234def567890
```

Ravindra's Tip

Attach the Elastic IP **before** you create DNS records (Chapter 9), install WordPress or configure `server_name` — then nothing ever needs to change. And when you finish a lab, do a quick "EIP audit": EC2 → Elastic IPs — anything without an associated instance should be released. Bas itna hi, and your bill stays clean.

14.2 AMI (Amazon Machine Image)

What is an AMI?

An **AMI** is a template used to launch instances. It is made of:

Part	Meaning
Root volume snapshot(s)	EBS snapshot of the OS disk (plus any extra data volumes included)
Block device mapping	Which volumes to create at launch, their sizes, types and device names
Launch permissions	Who can use it: only your account, specific accounts, or public
Metadata	Architecture (x86_64/arm64), virtualization, boot mode, ENA support, description

In simple words, an AMI is like a "photocopy master" of a fully set-up computer — every instance launched from it starts as an exact copy. Technically, launching from an AMI creates new EBS volumes from its snapshots according to the block device mapping.

AMI type	Provided by	Examples
AWS/Quick Start	AWS and OS vendors	Amazon Linux 2023, Ubuntu 24.04, Windows Server
AWS Marketplace	Software vendors (may include software charges)	Hardened OS images, WordPress by vendors, firewalls
Community AMIs	Anyone who made an AMI public	CentOS Stream images, custom builds (verify the publisher!)
My AMIs	You	Your own configured web server

Create an AMI from a configured instance

Suppose your `web-1` from Chapter 7 or 9 is working perfectly. Let's make a reusable image of it.

Step 1 — Clean up (optional but good): remove temporary files and secrets you don't want copied, e.g. `rm -f ~/.bash_history`.

Step 2 — Create image: EC2 → Instances → select `web-1` → **Actions → Image and templates → Create image**.

- Image name: `web-nginx-v1`, description: `Nginx + mysite, tested`.
- **No reboot** — leave **unticked** (recommended) — see the table below.
- Instance volumes: review sizes/types; add tags → **Create image**.

Step 3 — Wait: EC2 → Images → AMIs → status changes from pending to **available** (a few minutes). Under Snapshots you will see the snapshot(s) created for it.

No-reboot option	Pros	Cons
Off (default — instance reboots)	Filesystem is flushed and consistent; safest image	Short downtime during reboot
On (no reboot)	Zero downtime	Data being written at that moment may be inconsistent (like pulling the power cable); databases may need recovery

Launch a new instance from your AMI and verify

1. EC2 → AMIs → select `web-nginx-v1` → **Launch instance from AMI**.
2. Choose instance type, the same key pair and the `web-sg` security group → **Launch instance**.
3. When it is running, open `http://<NEW_PUBLIC_IP>` — your site should appear without installing anything.

```
# on the new instance
sudo service nginx status          # or httpd / apache2
curl -I http://localhost
ls /var/www/mysite
```

> **i Things that differ on the copy**
>
> The new instance gets a new instance ID, private IP and public IP. If your config hard-codes the old IP (for example Nginx `server_name 13.x.x.x` or WordPress site URL), update it — or move the Elastic IP to the new instance.

Copy an AMI to another region

AMIs are **regional**. To launch the same server in another region (for disaster recovery or users abroad): select the AMI → **Actions → Copy AMI** → destination region (e.g. `ap-south-2` Hyderabad) → name → (optional) encryption → **Copy AMI**. Then switch the console to the destination region and wait for it to become available.

```
aws ec2 copy-image --source-region ap-south-1 --source-image-id ami-0123456789abcdef0 \
  --region ap-south-2 --name "web-nginx-v1-hyd"
```

Share an AMI with another AWS account (brief)

Select the AMI → **Actions → Edit AMI permissions** → keep Private → **Add account ID** → enter the 12-digit account ID → **Save changes**. The other account finds it under AMIs → Private images. If the AMI's snapshots are encrypted with a customer managed KMS key, that key must also be shared; AMIs encrypted with the default `aws/ebs` key cannot be shared this way. Never make an AMI **public** if it contains your code, keys or passwords.

Deregister an AMI and delete its snapshots

Deregistering an AMI does **not** delete its snapshots — and snapshots keep costing money.

1. EC2 → AMIs → select the AMI → **Actions → Deregister AMI** (newer consoles offer a Delete associated snapshots option — the UI may vary) → confirm.
2. EC2 → Snapshots → find snapshots whose description says `Created by CreateImage(i-...) for ami-...` → **Actions → Delete snapshot**.

```
aws ec2 deregister-image --image-id ami-0123456789abcdef0
aws ec2 delete-snapshot --snapshot-id snap-0123456789abcdef0
```

The golden AMI idea

A **golden AMI** is a standard, pre-hardened, fully patched image with your common software (web server, agents, monitoring, security settings) already installed. Teams launch every new server from it instead of configuring each one by hand, so all servers are identical and launch fast. Rebuild the golden AMI regularly with the latest patches (tools like EC2 Image Builder automate this) and version it: `golden-web-v1`, `v2`, ...

> **Ravindra's Tip**
>
> Before any risky change — OS upgrade, PHP version change, big config edit — take an AMI (or at least a snapshot). If the change goes wrong, launch from the AMI and you are back in minutes. In my sessions I call it the "undo button" of the cloud. Name AMIs with a clear version label like `web-v3-before-php-upgrade` so you know exactly what's inside.

14.3 EBS snapshots in practice

Quick recap

- A snapshot is a **point-in-time, incremental** backup of an EBS volume: the first copies all used blocks, later ones only changed blocks.
- Snapshots are stored in **Amazon S3, managed by AWS** — you don't see them in your S3 buckets.
- Snapshots are **regional**: a volume created from a snapshot can go into **any AZ** of the region; copy the snapshot to use it in another region.

Create a snapshot of a volume

EC2 → Elastic Block Store → Volumes → select the volume (check the Attached resources column) → **Actions → Create snapshot** → description `web-1-root-before-upgrade` → tag `Name` → **Create snapshot**. Watch it under Snapshots until Completed. (Or from the instance's Storage tab → click the volume ID.)

```
aws ec2 create-snapshot --volume-id vol-0123456789abcdef0 --description "web-1-root-before-upgrade"
```

> **i Consistency**
>
> A snapshot captures what is **on disk** at that moment. For a data volume, stop the application or database writes (or at least `sync`) first. For a **root volume**, the cleanest snapshot is taken with the instance **stopped**. A snapshot of a running root volume usually works, but may be "crash-consistent" only.

Create a volume from a snapshot and mount it (file restore)

Scenario: a student accidentally deleted `/var/www/mysite/index.html`. We restore just that file from last night's snapshot without touching the running server.

Step 1 — Create volume: Snapshots → select the snapshot → **Actions → Create volume from snapshot** → type gp3 → size (same or larger) → **Availability Zone = same as the instance you will attach to** → **Create volume**.

Step 2 — Attach: Volumes → select the new volume → **Actions → Attach volume** → instance → device `/dev/sdg` → **Attach**.

Step 3 — Mount read-only and copy the file (do **not** run `mkfs` — it already has data!):

```
lsblk                                  # new disk, e.g. nvme1n1 with partition nvme1n1p1 (root snapshot)
sudo mkdir -p /restore
# XFS copy of a disk from the SAME instance has a duplicate UUID -> use nouuid
sudo mount -o ro,nouuid /dev/nvme1n1p1 /restore      # XFS (Amazon Linux / CentOS)
# sudo mount -o ro /dev/nvme1n1p1 /restore           # ext4 (Ubuntu)
ls /restore/var/www/mysite/
sudo cp -a /restore/var/www/mysite/index.html /var/www/mysite/
sudo umount /restore
```

Step 4 — Clean up: detach the temporary volume and delete it.

> **✓ Full restore of a broken root volume**
>
> To roll back a whole server: stop the instance → create a volume from the good snapshot in the same AZ → detach the broken root volume → attach the restored volume with the **same root device name** shown in the instance's Storage tab (e.g. `/dev/xvda`) → start. Newer consoles also offer **Replace root volume**, which does this for you. Alternatively, create an AMI from the snapshot and launch a new instance.

Copy a snapshot to another region

Select the snapshot → **Actions → Copy snapshot** → destination region → (optional) encrypt → **Copy snapshot**.

```
aws ec2 copy-snapshot --source-region ap-south-1 --source-snapshot-id snap-0123456789abcdef0 \
  --region ap-south-2 --description "DR copy of web-1 root"
```

Create an AMI from a snapshot

For a snapshot of a **root** volume: select it → **Actions → Create image from snapshot** → name → architecture (must match the original, e.g. x86_64) → root device name (e.g. `/dev/xvda`) → virtualization hvm → boot mode (as the original) → **Create image**. You can then launch instances from it. (Creating the AMI from the instance, as in 14.2, is easier because AWS fills these details automatically.)

Automate snapshots with Amazon Data Lifecycle Manager

Manual snapshots are forgotten; automated ones are not. General steps (UI may vary):

1. Tag the volumes (or instances) to protect, e.g. `Backup = daily`.
2. EC2 → Elastic Block Store → Lifecycle Manager → Create lifecycle policy.
3. Policy type **EBS snapshot policy** (or EBS-backed AMI policy to create AMIs) → target resource type Volume (or Instance) → target tag `Backup = daily`.
4. IAM role: **Default role** → policy status Enabled.
5. Schedule: e.g. every **24 hours** starting at 03:00 **UTC** (08:30 IST) → retention type Count → keep **7** → optionally copy tags, enable cross-region copy.
6. **Create policy**. Next day, check Snapshots for automatic snapshots tagged by DLM.

(AWS Backup is the broader, multi-service alternative used in many companies.)

Delete old snapshots

Snapshots → sort by Started → select old/unneeded snapshots → **Actions → Delete snapshot**. You cannot delete a snapshot that is used by a registered AMI — deregister the AMI first. Because snapshots are incremental, AWS keeps any blocks still needed by newer snapshots, so deleting an old one never breaks the newer ones.

14.4 Vertical scaling: changing the instance type

Scale up vs scale out

	Vertical scaling (scale up/down)	Horizontal scaling (scale out/in)
What	Make one server bigger/smaller (more vCPU/RAM)	Add/remove more servers of the same size
AWS way	Change instance type (e.g. `t3.micro` → `t3.medium`)	Auto Scaling group (ASG) behind an Elastic Load Balancer (ELB)

	Vertical scaling (scale up/down)	Horizontal scaling (scale out/in)
Downtime	Yes — stop/start needed	No — instances added/removed while running
Limit	Largest instance size available	Practically unlimited
Best for	Single servers, databases, quick fixes, labs	Stateless web/app tiers, high availability, variable traffic

Think of a busy dosa stall: vertical scaling is buying a bigger tawa for the same cook; horizontal scaling is opening two more counters with more cooks. Technically, vertical scaling moves your EBS-backed instance to different hardware with more resources; horizontal scaling distributes requests across multiple instances via a load balancer.

Steps to change the instance type

Step 1 — Check the current size (on the instance):

```
nproc          # number of vCPUs
free -h        # RAM
lscpu | grep -E "Model name|Architecture|^CPU\(s\)"
```

Step 2 — Stop the instance: EC2 → Instances → select → **Instance state → Stop instance** → wait for Stopped. (Take a snapshot/AMI first if this is important.)

Step 3 — Change type: Actions → Instance settings → Change instance type (the UI may vary slightly) → choose, e.g., `t3.small` → **Apply** (or **Change**). Incompatible types are usually greyed out or rejected with a message.

Step 4 — Start: Instance state → Start instance → note the new public IP (unless you use an Elastic IP) → SSH in again.

Step 5 — Verify:

```
nproc
free -h
lscpu | grep -E "Model name|Architecture|^CPU\(s\)"
sudo service nginx status        # make sure your apps came back (enabled at boot)
```

CLI alternative (instance must be stopped):

```
aws ec2 stop-instances --instance-ids i-0123456789abcdef0
aws ec2 wait instance-stopped --instance-ids i-0123456789abcdef0
aws ec2 modify-instance-attribute --instance-id i-0123456789abcdef0 --instance-type "{\"Value\":
\"t3.small\"}"
aws ec2 start-instances --instance-ids i-0123456789abcdef0
```

What changes and what stays

Changes	Stays the same
vCPU, RAM, network/EBS bandwidth	All **EBS volumes and their data** (root and data)
Auto-assigned public IP (new one after start) — unless an **Elastic IP** is attached	**Private IP** address(es) and ENIs

Changes	Stays the same
Instance store data is **lost** (it's temporary host disk)	Instance ID, security groups, IAM role, tags, key pair
Hourly price	AMI/OS and installed software

Compatibility cautions

- **Architecture must match:** an x86_64 instance (t3, m7i) cannot simply become an Arm/Graviton type (t4g, m7g). The AMI/OS is compiled for one architecture — for Graviton, launch a new instance from an arm64 AMI and migrate.
- **Drivers for Nitro types:** modern types (t3, m5 and newer) need **ENA** networking and **NVMe** storage drivers. Current Amazon Linux, Ubuntu and CentOS Stream AMIs have them; very old AMIs moving from `t2` to `t3` may fail to boot or lose networking. Also remember disk names change to `/dev/nvme...` — another reason fstab uses UUIDs (Chapter 13).
- **Availability:** not every instance type is offered in every AZ/region; the console tells you if a type is unsupported.
- **Other features:** hibernation, instance store volumes, or placement groups can restrict which types you can switch to.
- **Downtime:** the stop/start takes a few minutes — plan it in a maintenance window and inform users.

> ⚠ **Free tier caution**
>
> Only specific instance types (for example `t2.micro`/`t3.micro`, depending on your account and region) are covered by the free tier/free plan. Scaling up to `t3.small` or larger is billed normally from the moment it starts. Scale back down after your test.

> **Ravindra's Tip**
>
> Before scaling up, find out **what** is actually short — CPU, RAM or disk I/O. Check `top`, `free -h` and CloudWatch metrics. If RAM is the problem, a bigger CPU won't help; if a T-instance keeps running out of **CPU credits**, a larger T-type or an M-type fixes it. And when you find yourself scaling up again and again for a web app, it is time to think horizontal: an Auto Scaling group behind a load balancer. Samjla ka?

Common Mistakes I See Students Make

- **Allocating Elastic IPs and never releasing them** — idle EIPs are still charged.
- **Forgetting to update SSH/DNS after associating an EIP** — the old public IP is gone.
- **Deregistering an AMI but leaving its snapshots**, which keep costing money.
- **Creating an AMI with "No reboot" from a busy database server** and getting a corrupted copy.
- **Creating a volume from a snapshot in a different AZ** from the instance — it cannot be attached.

- **Running `mkfs` on a volume restored from a snapshot**, wiping the very data they wanted to recover.
- **Mounting an XFS copy on the same instance without `nouuid`** and getting a "duplicate UUID" error.
- **Changing an x86 instance to a Graviton type** and expecting it to boot.
- **Forgetting that the public IP changes after a type change** and thinking the server is dead.
- **Scaling up beyond free-tier types** and leaving it running.

Interview Questions

Q1. What is an Elastic IP and why is it needed?

A static public IPv4 address owned by your account. The auto-assigned public IP changes on stop/start; an EIP stays the same and can be moved between instances, so DNS records don't need to change.

Q2. How would you replace a web server without changing DNS?

Build and test the new instance, then re-associate the Elastic IP from the old instance to the new one (allow reassociation). Traffic switches within seconds; keep the old instance briefly for rollback.

Q3. What does an AMI consist of?

One or more EBS snapshots (root and optionally data volumes), a block device mapping, launch permissions and metadata such as architecture and virtualization type.

Q4. What is the risk of the "No reboot" option when creating an AMI?

The filesystem is captured while running, so in-flight writes may not be flushed and the image may be inconsistent (crash-consistent); databases might need recovery.

Q5. Are EBS snapshots full or incremental? Where are they stored?

Incremental — only changed blocks after the first snapshot. They are stored in Amazon S3, managed by AWS (not visible in your buckets), and are regional.

Q6. How do you restore a single deleted file from an EBS snapshot?

Create a volume from the snapshot in the instance's AZ, attach it, mount it read-only (with `nouuid` for XFS on the same instance), copy the file back, then unmount, detach and delete the temporary volume.

Q7. How do you automate EBS snapshots?

Use Amazon Data Lifecycle Manager (tag-based policies with schedules and retention) or AWS Backup.

Q8. What is the difference between vertical and horizontal scaling? What changes when you change an instance type?

Vertical = bigger instance (needs stop/start); horizontal = more instances behind a load balancer (ASG + ELB). After a type change, the auto-assigned public IP changes (unless EIP) and instance store data is lost; EBS data, private IP and instance ID stay.

Practice Questions and Lab Exercises

1. Why does the public IP of an instance change after stop/start? How does an Elastic IP fix it?
2. What is the default Elastic IP limit per region? Are idle Elastic IPs free?
3. List the parts of an AMI. Differentiate AWS, Marketplace, Community and own AMIs.
4. Explain the pros and cons of the "No reboot" option.
5. Why must you delete snapshots after deregistering an AMI?
6. What is a golden AMI? Why do companies use it?
7. Compare scale up and scale out with an example.
8. **Lab:** Allocate an EIP, associate it with `web-1`, stop/start the instance and confirm the IP stays the same. Then build `web-2`, re-associate the EIP to it, and finally release the EIP.
9. **Lab:** Create an AMI of your working web server, launch a new instance from it and confirm the website loads without any installation. Copy the AMI to `ap-south-2`, then deregister both copies and delete their snapshots.
10. **Lab:** Take a snapshot of your root volume, delete a file from the website, restore it by creating and mounting a volume from the snapshot.
11. **Lab:** Create a DLM policy that takes a daily snapshot of volumes tagged `Backup=daily` and keeps 7.
12. **Lab:** Change your instance from `t3.micro` to `t3.small`, verify with `nproc`, `free -h` and `lscpu`, then change it back.

★ Quick Revision

- **Elastic IP:** static public IPv4 owned by your account; allocate → associate → (re-associate for server replacement) → disassociate → release. Default limit 5 per region; all public IPv4 is charged hourly, even idle.
- **AMI:** root snapshot(s) + block device mapping + launch permissions. Create from instance (reboot = consistent), launch copies, copy across regions, share by account ID, deregister **and** delete snapshots. Golden AMI = standard hardened image.
- **Snapshots:** incremental, stored in S3 (managed), regional. Create volume from snapshot in any AZ → attach → mount (`ro,nouuid` for XFS copy) → restore. Copy across regions, make AMIs, automate with DLM, delete old ones.
- **Vertical scaling:** stop → Change instance type → start → verify (`nproc`, `free -h`, `lscpu`). Public IP changes (unless EIP), instance store lost, EBS and private IP stay. Watch architecture, ENA/NVMe drivers, AZ availability, free tier. For web tiers prefer horizontal scaling (ASG + ELB).

Appendix A: Troubleshooting Checklist

Mitranno, when something breaks, ghabru naka (don't panic). Work through the list **from outside to inside** — most problems are found in the first few steps.

A.1 Cannot SSH into the instance

Check	How
Instance is **running** and status checks 2/2	EC2 console
Correct **public IP** (changes after stop/start)	Console → Instance → Public IPv4
Security group allows **TCP 22** from your current IP (mobile hotspot / college Wi-Fi IPs change!)	SG inbound rules → My IP
Correct **username**	`ec2-user` (AL/CentOS Stream), `ubuntu` (Ubuntu), `centos` (old CentOS)
Correct **key** and permissions	`chmod 400 key.pem` / icacls on Windows; key pair name shown in console
Subnet has route `0.0.0.0/0` → **Internet Gateway**	VPC → Route tables
Network ACL allows 22 inbound and ephemeral ports outbound	VPC → Network ACLs
College/office network blocks outbound port 22	Try mobile hotspot, or use EC2 Instance Connect / Session Manager
Verbose output	`ssh -v -i key.pem ec2-user@IP`

Common error messages:

Message	Meaning
`Connection timed out`	Network/security group/NACL/wrong IP
`Connection refused`	Host reachable but sshd not running (rare on EC2)
`Permission denied (publickey)`	Wrong user name or wrong key
`UNPROTECTED PRIVATE KEY FILE`	Key permissions too open → `chmod 400`
`REMOTE HOST IDENTIFICATION HAS CHANGED`	New instance reused the same IP → `ssh-keygen -R <IP>`

A.2 Website not loading

1. `curl -I http://localhost` **on the server** — works? If not, the problem is on the server (step 3 onwards).
2. From your laptop: `curl -I http://<PUBLIC_IP>` — timeout → security group (80/443), firewalld, NACL, public IP, typed `https`.
3. `sudo service nginx status` (or `httpd` / `apache2`) — running? `sudo nginx -t` / `apachectl configtest` — syntax OK?

4. `sudo ss -tlnp | grep -E ':80|:443'` — who is listening? Two web servers fighting for port 80?
5. Read the **error log** (`/var/log/nginx/error.log`, `/var/log/httpd/error_log`, `/var/log/apache2/error.log`).
6. **403** → permissions (`namei -l <file>`), missing index file, SELinux (`restorecon -Rv`, `ls -Z`).
7. **404** → wrong root / DocumentRoot / file name case.
8. **502/504** → backend app down or wrong port; `curl 127.0.0.1:<port>`; `pm2 status`; `sudo service <app> status`; CentOS: `setsebool -P httpd_can_network_connect 1`.
9. **500** → application error; check app logs (`journalctl -u <app>`, `pm2 logs`, PHP-FPM log).

A.3 Database problems

Symptom	Check
Can't connect	`sudo service mariadb status` / `mysql` / `mongod`; `sudo ss -tlnp | grep -E '3306|27017'`
Access denied	User/host/password: `SELECT user,host FROM mysql.user;`
Works locally, not from another server	`bind-address` / `bindIp`, security group from the app server's SG only
mongod won't start	`sudo tail -50 /var/log/mongodb/mongod.log`; disk full? permissions on data dir? SELinux?

A.4 Server problems

Symptom	Check
Disk full	`df -h`, `sudo du -h --max-depth=1 / | sort -h`, clear old logs `sudo journalctl --vacuum-size=200M`, grow EBS
Out of memory / processes killed	`free -h`, `dmesg | grep -i oom`, add swap, bigger instance type
High CPU	`top` / `htop`; T-instance CPU credits exhausted? (CloudWatch CPUCreditBalance)
Instance won't boot after editing fstab	Stop instance, detach root volume, attach to a helper instance, fix fstab, re-attach (or use EC2 Serial Console)
Service not starting at boot	`systemctl is-enabled <svc>`; `sudo systemctl enable <svc>`; `pm2 startup` + `pm2 save`
Website/SSH broken after stop/start or type change	Public IP changed — attach an Elastic IP (Chapter 14)
Time zone wrong in logs	`sudo timedatectl set-timezone Asia/Kolkata`

A.5 Web-server configuration and domains

Symptom	Check / fix
403 after changing root / DocumentRoot	`namei -l <file>` (755 dirs, 644 files); matching `<Directory>` with `Require all granted`; CentOS SELinux: `ls -Z`, `semanage fcontext -a -t httpd_sys_content_t "/path(/.*)?"` + `restorecon -Rv`; never serve from home folders

Symptom	Check / fix
New port not reachable	`sudo ss -tlnp` (listening?); security group inbound rule; `firewall-cmd --add-port=PORT/tcp`; CentOS: `semanage port -a -t http_port_t -p tcp PORT` (service fails with Permission denied otherwise)
Domain not resolving	`dig example.com +short` and `dig @8.8.8.8 example.com`; record name typo (`blog`, not `blog.example.com`); `dig NS example.com` — editing records at the wrong DNS host?; domain verified and not expired?; wait for TTL / flush local cache
Default site shows instead of your vhost	`server_name` / `ServerName` + `ServerAlias` include the exact name?; Ubuntu: remove `sites-enabled/default` or `a2dissite 000-default.conf`; Nginx `default_server` elsewhere; inspect `sudo nginx -T` / `sudo apachectl -S`; forgot reload?
Domain resolves but page times out	Security group 80/443; DNS points to the right **Elastic IP**?; `curl -I http://EIP -H "Host: example.com"`

Appendix B: Command Cheat Sheet

B.1 Package and service management

Task	Ubuntu	Amazon Linux 2023 / CentOS Stream 9
Update package lists / upgrade	`sudo apt update && sudo apt upgrade -y`	`sudo yum update -y`
Install	`sudo apt install -y <pkg>`	`sudo yum install -y <pkg>`
Remove	`sudo apt remove <pkg>`	`sudo yum remove <pkg>`
Search	`apt search <word>`	`yum search <word>`
Start now + enable at boot	`sudo service <svc> start` + `sudo systemctl enable <svc>`	same
Status / logs	`sudo service <svc> status`, `journalctl -u <svc> -f`	same
Apache service	`apache2`	`httpd`
MySQL service	`mysql`	`mariadb`
Web user	`www-data`	`nginx` / `apache`
Firewall	`sudo ufw allow 'Nginx Full'`	`sudo firewall-cmd --permanent --add-service=http && sudo firewall-cmd --reload`

B.2 Files, permissions, users

```
ls -la            cd /path          pwd              mkdir -p a/b
cp -r src dst     mv old new        rm -r dir        touch file
cat f   less f    head -n 20 f      tail -f f        wc -l f
chmod 755 dir     chmod 644 file    chmod 400 key.pem
sudo chown -R user:group /path
sudo useradd -m -s /bin/bash u      sudo passwd u    sudo usermod -aG wheel u  (Ubuntu: sudo)
```

B.3 Search and text

```
grep -rin "text" /path              find / -name "*.conf" 2>/dev/null
awk '{print $1}' file               sed -i 's/old/new/g' file
cmd | sort | uniq -c | sort -rn     cmd > out.txt 2>&1      echo x | sudo tee -a /etc/file
```

B.4 Networking

```
ip a            ip r              hostname -I          curl -s https://checkip.amazonaws.com
sudo ss -tlnp   ping -c 4 host    curl -I http://host  dig +short domain
nc -zv host 3306                  traceroute host      sudo hostnamectl set-hostname web01
```

B.5 Disk and EBS

```
df -hT          du -sh /path       lsblk -f            sudo blkid
sudo file -s /dev/nvme1n1          sudo mkfs -t xfs /dev/nvme1n1
sudo mkdir /data && sudo mount /dev/nvme1n1 /data     sudo umount /data
sudo growpart /dev/nvme0n1 1       sudo xfs_growfs -d /     sudo resize2fs /dev/nvme0n1p1
sudo mount -a   # test /etc/fstab  (UUID=... /data xfs defaults,nofail 0 2)
```

B.6 Web servers

```
sudo nginx -t && sudo service nginx reload
sudo apachectl configtest && sudo service httpd reload          # AL2023 / CentOS
sudo apache2ctl configtest && sudo service apache2 reload       # Ubuntu
sudo a2ensite site.conf ; sudo a2dissite 000-default.conf ; sudo a2enmod rewrite proxy proxy_http
sudo restorecon -Rv /var/www ; sudo setsebool -P httpd_can_network_connect 1   # CentOS SELinux
sudo nginx -T | grep -E "listen|server_name|root"               # what Nginx really loaded
sudo apachectl -S                                               # Apache vhosts + default
(Ubuntu: apache2ctl -S)
sudo ln -s /etc/nginx/sites-available/site /etc/nginx/sites-enabled/   # Ubuntu Nginx enable site
sudo semanage fcontext -a -t httpd_sys_content_t "/srv/site(/.*)?" ; sudo restorecon -Rv /srv/
site
sudo semanage port -a -t http_port_t -p tcp 8081                # CentOS: allow a new web port
curl -H "Host: site1.example.com" http://localhost              # test a vhost without DNS
```

Setting	Nginx (`server { }`)	Apache (`<VirtualHost>`)
Port	`listen 80;`	`<VirtualHost *:80>` + `Listen 80`
Site name	`server_name example.com www.example.com;`	`ServerName example.com` + `ServerAlias www.example.com`
Folder	`root /var/www/site;`	`DocumentRoot /var/www/site` + `<Directory>`
Index file	`index index.html;`	`DirectoryIndex index.html`
No folder listing	`autoindex off;`	`Options -Indexes`
Redirect	`return 301 https://$host$request_uri;`	`Redirect permanent / https://example.com/`
Hide version	`server_tokens off;`	`ServerTokens Prod` + `ServerSignature Off`

B.7 DNS and domains

```
dig example.com +short           dig www.example.com +short       dig NS example.com +short
dig @8.8.8.8 example.com +short  dig MX example.com +short        dig +trace example.com
nslookup example.com             host example.com                 curl -I http://example.com
sudo yum install -y bind-utils   # dig on AL2023/CentOS  (Ubuntu: sudo apt install -y dnsutils)
ipconfig /flushdns               # Windows     resolvectl flush-caches   # Linux laptop
sudo certbot --nginx -d example.com -d www.example.com -d blog.example.com
```

Typical records: **A** `@` → Elastic IP · **CNAME** `www` → `@` · **A** `blog` → Elastic IP · TTL 300–3600.

B.8 Databases

```
sudo mysql                                  # admin shell (socket auth)
sudo mysql_secure_installation
mysql -u appuser -p -h 127.0.0.1 appdb
```

```
mysqldump -u appuser -p appdb > backup.sql   mysql -u appuser -p appdb < backup.sql
mongosh         # MongoDB shell: show dbs ; use db ; db.col.find()
```

B.9 App process managers

```
pm2 start app.js --name api ; pm2 status ; pm2 logs api ; pm2 reload api ; pm2 startup ; pm2 save
sudo systemctl daemon-reload ; sudo service flaskapp start ; sudo systemctl enable flaskapp ; jou
rnalctl -u flaskapp -f
gunicorn --workers 3 --bind 127.0.0.1:8000 wsgi:app
```

B.10 EC2 operations (AWS CLI, optional)

```
aws ec2 allocate-address --domain vpc
aws ec2 associate-address --instance-id i-xxx --allocation-id eipalloc-xxx --allow-reassociation
aws ec2 release-address --allocation-id eipalloc-xxx
aws ec2 create-image --instance-id i-xxx --name web-v1            # add --no-reboot only if you
accept the risk
aws ec2 copy-image --source-region ap-south-1 --source-image-id ami-xxx --region ap-south-2 --
name web-v1-dr
aws ec2 deregister-image --image-id ami-xxx ; aws ec2 delete-snapshot --snapshot-id snap-xxx
aws ec2 create-snapshot --volume-id vol-xxx --description "before-upgrade"
aws ec2 modify-instance-attribute --instance-id i-xxx --instance-type "{\"Value\":
\"t3.small\"}"   # instance stopped
nproc ; free -h ; lscpu                                            # verify after resizing
```

B.11 SSH and file transfer

```
ssh -i key.pem ec2-user@IP                  ssh -i key.pem ubuntu@IP
scp -i key.pem file ec2-user@IP:~/          scp -i key.pem -r dir ubuntu@IP:~/
scp -i key.pem ec2-user@IP:/path/file .     rsync -avz -e "ssh -i key.pem" dir/ ubuntu@IP:~/dir/
```

Appendix C: Glossary

Term	Meaning
A record	DNS record mapping a name to an IPv4 address
AMI	Amazon Machine Image — template (OS + software) to launch EC2 instances
Auto Scaling group (ASG)	Group of EC2 instances that AWS adds/removes automatically based on demand or health
AZ (Availability Zone)	One or more isolated data centres within an AWS Region
Block device mapping	Part of an AMI/launch that lists which volumes (root, data) attach to an instance, their device names, sizes and types
Bastion host	A hardened server used as the only SSH entry point into private subnets
CNAME	DNS alias record pointing one name to another name; not allowed at the apex
CIDR	Classless Inter-Domain Routing notation, e.g. `10.0.0.0/16`
cloud-init	Tool that configures an instance at first boot (runs user data, grows root partition)
Daemon	Background service process (e.g. `sshd`, `nginx`)
DLM (Data Lifecycle Manager)	AWS service that automates creation and deletion of EBS snapshots and EBS-backed AMIs
DHCP	Protocol that assigns IP addresses automatically
DNS	Domain Name System — translates names to IP addresses using a hierarchy of root, TLD and authoritative name servers
Document root / DocumentRoot	Folder from which a web server serves files (`root` in Nginx, `DocumentRoot` in Apache)
EBS	Elastic Block Store — persistent block storage volumes for EC2
EC2	Elastic Compute Cloud — AWS virtual servers
Elastic IP	Static public IPv4 address allocated to your AWS account
ELB (Elastic Load Balancer)	Managed load balancer (ALB/NLB) that spreads traffic across instances
ENI	Elastic Network Interface — virtual network card of an instance
FastCGI / PHP-FPM	Protocol / process manager that lets web servers run PHP efficiently
firewalld	Dynamic host firewall used on RHEL/CentOS
FQDN	Fully qualified domain name, e.g. `blog.example.com.`
fstab	`/etc/fstab` — file listing filesystems to mount at boot
Golden AMI	Standard, patched, pre-configured image from which all servers are launched
Gunicorn	Production WSGI HTTP server for Python web apps
Horizontal scaling	Adding/removing instances (scale out/in), usually with ASG + ELB
IAM	Identity and Access Management — users, groups, roles and policies in AWS

Term	Meaning
IGW	Internet Gateway — connects a VPC to the internet
IMDS	Instance Metadata Service at `169.254.169.254` (use IMDSv2 tokens)
IOPS	Input/output operations per second — storage performance measure
Key pair	Public/private key used for SSH authentication to EC2
LTS	Long Term Support release (e.g. Ubuntu 24.04 LTS, Node.js LTS)
NAT	Network Address Translation — maps private IPs to a public IP
NAT Gateway	AWS managed service giving private subnets outbound internet access
Name server (NS)	Authoritative DNS server that answers queries for a domain; set at the registrar
NACL	Network Access Control List — stateless subnet-level firewall
Nitro	AWS hypervisor/hardware platform; EBS appears as NVMe devices
pm2	Production process manager for Node.js applications
Port	16-bit number identifying a service on a host (0–65535)
Registrar	Company that registers and renews domain names (GoDaddy, Namecheap, Route 53 Domains)
Region	Geographic area containing multiple AZs, e.g. `ap-south-1` (Mumbai)
Reverse proxy	Server (Nginx/Apache) that forwards client requests to backend apps
Route 53	AWS DNS service: hosted zones, records, Alias records, domain registration
RFC 1918	Standard defining private IPv4 ranges 10/8, 172.16/12, 192.168/16
Scale up (vertical scaling)	Moving to a bigger instance type (more vCPU/RAM); needs stop/start
Server block	Nginx `server { }` section defining one website (port, `server_name`, root)
Security group	Stateful virtual firewall attached to instances/ENIs
SELinux	Security-Enhanced Linux — mandatory access control on RHEL/CentOS
Service quota	Per-account, per-region limit on resources (e.g. 5 Elastic IPs by default)
Snapshot	Incremental point-in-time backup of an EBS volume
SPA	Single Page Application (React, Angular)
Subdomain	Name to the left of your domain, e.g. `blog.example.com`; created free with a DNS record
Subnet	A range of IP addresses within a VPC, located in one AZ
systemd	Linux init system and service manager (`systemctl`, `journalctl`)
TLD	Top-level domain such as `.com`, `.in`, `.org`
TLS/SSL	Encryption protocol used by HTTPS
TTL	Time To Live — seconds a DNS answer may be cached by resolvers
User data	Script passed to an instance that runs at first boot
UUID	Universally unique identifier of a filesystem, used in fstab

Term	Meaning
Virtual host	Apache `<VirtualHost>` section defining one website; name-based vhosts share one IP and port
VPC	Virtual Private Cloud — your isolated network in AWS
WSGI	Python standard interface between web servers and Python apps

Quick-Revision Summary (All Chapters)

Chapter	Must-remember points
1 Networking	IP = host, port = service; 22/80/443/3306/27017/3000/5000/8080; IPv4 32-bit, CIDR 2^(32−n), AWS reserves 5/subnet; IPv6 128-bit, `::` once; RFC 1918 private ranges; NAT; auto public IP vs Elastic IP
2 OSI	7 layers (Physical → Application), PDUs bit/frame/packet/segment/data; TCP/IP 4 layers; TCP vs UDP; DNS → TCP → TLS → HTTP
3 Linux basics	`ls cd cp mv rm mkdir cat less tail`; `chmod 755/644/400`, `chown`; `useradd`, `usermod -aG`; `apt` vs `yum`; `ps top kill`; nano/vim
4 Linux advanced	`grep find awk sed`; pipes/redirection/`tee`; `tar`; `service` (+ `systemctl enable`)/`journalctl`; `ip ss curl dig`; `df du lsblk mount fstab`; cron; env vars; `ssh scp rsync`; bash scripts
5 EC2	AMI + type + key pair + SG + EBS; pricing models; SSH users; `chmod 400`; PuTTY/.ppk, MobaXterm, OpenSSH; stop vs terminate; best practices
6 Web servers	Nginx (event-driven, proxy) vs Apache (modules, .htaccess); `httpd` vs `apache2`; web roots; `nginx -t`; firewalld/SELinux on CentOS
7 Static hosting	Files → `/var/www/mysite` → 755/644 → server block / vhost → test → reload; `restorecon`; 403/404 troubleshooting
8 Configuring web servers	Port → site (Host header) → folder → file; `conf.d` vs `sites-available`; `default_server` / first vhost wins; change root + `<Directory>`; `semanage fcontext`/`port`; redirects, `server_tokens off`; backup → test → reload → `curl`
9 Domains & DNS	Registrar ≠ name servers ≠ web host; resolver → root → TLD → authoritative, TTL; A, CNAME (not apex), MX, TXT, NS; Elastic IP first; GoDaddy A `@` → EIP, CNAME `www`; Route 53 hosted zone + NS; subdomain = record + vhost; `dig`/`nslookup`; certbot `-d` for each name
10 Dynamic hosting	MariaDB/MySQL install & secure; app user; PHP-FPM; Flask + Gunicorn + systemd; Express + pm2; reverse proxy; 502 fixes
11 Apps	WordPress full install; Flask with Gunicorn + Nginx; Express with pm2 cluster + Nginx
12 Stacks	LAMP, LEMP, MEAN, MERN; MongoDB official repo; SPA build served by Nginx + `/api` proxy
13 EBS	gp3/gp2/io2/st1/sc1; snapshots; NVMe names; grow: modify → `growpart` → `xfs_growfs`/`resize2fs`; add: attach → `mkfs` → mount → fstab UUID `nofail`; detach safely
14 EC2 operations	Elastic IP: allocate → associate → re-associate → release (5/region, charged even idle); AMI = snapshots + block device mapping + permissions, create/launch/copy/share, deregister + delete snapshots, golden AMI; snapshot restore (`ro,nouuid`), DLM; change instance type: stop → change → start, verify `nproc`/`free -h`/`lscpu`

A Last Word from Ravindra

Shabbas, mitranno — you have reached the end of the handbook! Now the real learning starts: rebuild every lab on your own, without looking at the book, until it feels natural. Break things, fix them, and write down what you learnt. Keep practising, keep asking "why", and all the best for your projects and interviews. Chala, bhetuya pudhchya session madhe (let's meet in the next session)!

— Ravindra Bagale, Cloud Trainer · linkedin.com/in/ravindra-bagale