সহজ কথায় বলতে গেলে, Docker (**ডকার**) **হলো এমন একটি প্রযুক্তি যা আপনার অ্যাপ্লিকেশনকে একটি ছোট প্যাকেজ বা বাক্সের মধ্যে ভরে ফেলে, যাতে এটি যেকোনো কম্পিউটারে কোনো ঝামেলা ছাড়াই চলতে পারে।**

১. ডকার আসলে কী? (What is Docker?)

আগেকার দিনে যখন ডেভেলপাররা কোড লিখতেন, তখন একটি কমন সমস্যা হতো: *"আমার কম্পিউটারে কোড চলে, কিন্তু সার্ভারে চলছে না!"* এর কারণ হতে পারে আপনার পিসিতে পাইথনের ভার্সন ৩.১০, কিন্তু সার্ভারে ভার্সন ৩.৮।

ডকার এই সমস্যার সমাধান দেয়। এটি আপনার কোড, লাইব্রেরি এবং যাবতীয় সেটিংসকে একটি Container (কন্টেইনার)-এর মধ্যে এমনভাবে প্যাক করে দেয় যে, সেটি আপনার পিসি, আপনার বন্ধুর পিসি বা ক্লাউড সার্ভার—সবখানেই একইভাবে কাজ করে।

২. কেন ডকার ব্যবহার করবেন?

- Consistency: **যেকোনো পরিবেশে একইভাবে কাজ করে।**
- Isolation: **একটি কন্টেইনারের সমস্যা অন্য কন্টেইনারে প্রভাব ফেলে না।**
- Lightweight: **এটি ভার্চুয়াল মেশিন** (VM) **এর চেয়ে অনেক কম মেমরি খরচ করে।**
- Scalability: **খুব দ্রুত অনেকগুলো কন্টেইনার তৈরি বা বন্ধ করা যায়।**

৩. ডকার কীভাবে কাজ করে? (Key Components)

ডকার বুঝতে হলে আপনাকে ৪টি গুরুত্বপূর্ণ বিষয় বুঝতে হবে:

ক. Dockerfile

এটি একটি সাধারণ টেক্সট ফাইল যেখানে লেখা থাকে ডকার কীভাবে আপনার ইমেজটি তৈরি করবে। এটি অনেকটা রান্নার রেসিপির মতো।

উদাহরণ: "প্রথমে পাইথন ইনস্টল করো, তারপর কোডগুলো কপি করো, শেষে অ্যাপটি রান করো।"

খ. Docker Image

ডকারফাইল বিল্ড করলে একটি Image তৈরি হয়। এটি একটি স্ট্যাটিক ফাইল যা রান করার জন্য তৈরি। এটি অনেকটা সফটওয়্যারের ইনস্টলার ফাইলের মতো।

গ. Docker Container

যখন আপনি একটি ইমেজকে রান (Run) করেন, তখন সেটি একটি Container হয়ে যায়। ইমেজ যদি হয় কোনো গেমের সিডি, তবে কন্টেইনার হলো সেই গেমটি যখন স্ক্রিনে চলছে।

ঘ. Docker Hub

এটি হলো ডকার ইমেজগুলোর অনলাইন স্টোর (যেমন GitHub)। এখান থেকে আপনি অন্যের তৈরি ইমেজ (যেমন: MySQL, Python, Node.js) নামিয়ে ব্যবহার করতে পারেন।

৪. Docker **বনাম** Virtual Machine (VM)

অনেকে ডকারকে ভার্চুয়াল মেশিনের সাথে গুলিয়ে ফেলেন। এদের মূল পার্থক্য হলো:

- VM: পুরো একটি অপারেটিং সিস্টেম (OS) লোড করে, যা অনেক ভারী।
- Docker: কম্পিউটারের আসল অপারেটিং সিস্টেম ব্যবহার করে শুধু প্রয়োজনীয় ফাইলগুলো আলাদা করে রাখে। তাই এটি খুব দ্রুত স্টার্ট হয়।

৫. প্রয়োজনীয় ডকার কমান্ড (Basic Commands)

ডকার চালানোর জন্য নিচের কমান্ডগুলো সবচেয়ে বেশি ব্যবহৃত হয়:

কমান্ড	কাজ
docker build	ডকারফাইল থেকে ইমেজ তৈরি করা।
docker images	আপনার পিসিতে কী কী ইমেজ আছে তা দেখা।
docker run	ইমেজ থেকে কন্টেইনার চালানো।
docker ps	বর্তমানে কোন কোন কন্টেইনার চলছে তা দেখা।
docker stop	একটি কন্টেইনার বন্ধ করা।
docker pull	অনলাইন (Docker Hub) থেকে ইমেজ ডাউনলোড করা।

৬. Docker Compose

যখন আপনার প্রজেক্টে একাধিক জিনিস থাকে (যেমন: একটি কন্টেইনারে ওয়েবসাইট, আরেকটিতে ডাটাবেস), তখন সেগুলোকে একসাথে ম্যানেজ করার জন্য Docker Compose ব্যবহার করা হয়। এটি একটি .yml ফাইলের মাধ্যমে সব কন্টেইনারকে এক কমান্ডে চালু করতে সাহায্য করে।

ডকার শেখার প্রথম ধাপ হলো আপনার পিসিতে Docker Desktop ইনস্টল করা এবং একটি সিম্পল "Hello World" কন্টেইনার চালানো।

🐳 Docker কী?

Docker = Containerization platform

সহজ ভাষায় 👇

“আমার মেশিনে কাজ করে, কিন্তু তোমার মেশিনে না” — এই সমস্যার সমাধান = **Docker**

Docker অ্যাপ + সব dependency একসাথে **container** বানায়
➡ যেকোনো জায়গায় same ভাবে run করবে

🤯 Docker কেন দরকার?

আগে কী হতো ❌

- OS ভিন্ন
- Library version conflict
- Setup করতে ঘণ্টা লাগতো

Docker এ কী হয় ✅

- Same environment everywhere
- Fast deploy
- Lightweight
- Microservices friendly
- Cloud ready (AWS, GCP, Azure)

🆚 Virtual Machine vs Docker

VM	Docker
Full OS	OS share করে
Heavy	Lightweight
Slow start	Fast
GB size	MB size

👉 Docker = **VM এর lightweight version**

🧱 Docker Core Concepts (Very Important)

Dockerfile → Image → Container

🔹 Dockerfile কী?

👉 একটা **recipe / blueprint**
👉 কীভাবে image বানবে সেটা লেখা থাকে

Example:

```
FROM node:18

WORKDIR /app

COPY . .

RUN npm install

CMD ["npm","start"]
```

🔹 Docker Image কী?

👉 Read-only template

👉 App + dependency + config

docker images

🔹 Docker Container কী?

👉 Image থেকে তৈরি running instance

👉 Live app

docker ps

⚙️ Docker Install

docker --version

docker compose version

🚀 Docker Basic Commands (Must Know)

Image

docker pull nginx

docker build -t myapp .

docker rmi image_id

Container

docker run nginx

docker run -d -p 3000:3000 myapp

docker stop container_id

docker rm container_id

🌐 Port Mapping

docker run -p 8080:80 nginx

📌 Browser → localhost:8080
📌 Container → 80

📁 Volume (Data persist)

docker volume create mydata

docker run -v mydata:/data myapp

👉 Container delete হলেও data safe

🌍 Bind Mount

docker run -v $(pwd):/app myapp

👉 Local file ↔ container sync
👉 Dev time super useful

🔐 Environment Variable

docker run -e DB_HOST=localhost myapp

🧠 Docker Network

docker network create mynet

docker run --network=mynet app1

docker run --network=mynet app2

👉 Containers talk using name

🧩 **Docker Compose (Very Important)**

👉 Multiple container manage করে

docker-compose.yml

```
version: "3"
services:
  web:
    build: .
    ports:
      - "3000:3000"
  db:
    image: mongo
    ports:
      - "27017:27017"
```

Run:

```
docker compose up -d
docker compose down
```

🛠 Real Project Example (Next.js + Mongo)

```
services:
 next:
  build: .
  ports:
   - "3000:3000"
  depends_on:
   - mongo
 mongo:
  image: mongo
```

🧹 Docker Clean Commands

```
docker system prune
docker image prune
docker container prune
```

🐞 Debug Container

```
docker logs container_id
docker exec -it container_id bash
```

🔒 Docker Security (Interview)

✓ Use official images
✓ Small base image (alpine)

✓ Don’t run as root

✓ Secrets via env

🧪 Docker vs Kubernetes

Docker	Kubernetes
Container run	Container manage
Single host	Multi host
Simple	Complex

👉 Docker = **build & run**

👉 K8s = **orchestrate**

🏗️ Docker in System Design

- Microservices
- CI/CD
- Blue-Green deploy
- Scaling
- Cloud infra

🎯 FAANG Interview Questions

Q: Docker image vs container?

👉 Blueprint vs running app

Q: COPY vs ADD?

👉 COPY safer

Q: CMD vs ENTRYPOINT?

👉 CMD overrideable

Q: Why Docker faster than VM?

👉 No full OS

🧠 Daily Docker Workflow

docker build -t app .

docker run -d -p 3000:3000 app

docker push repo/app