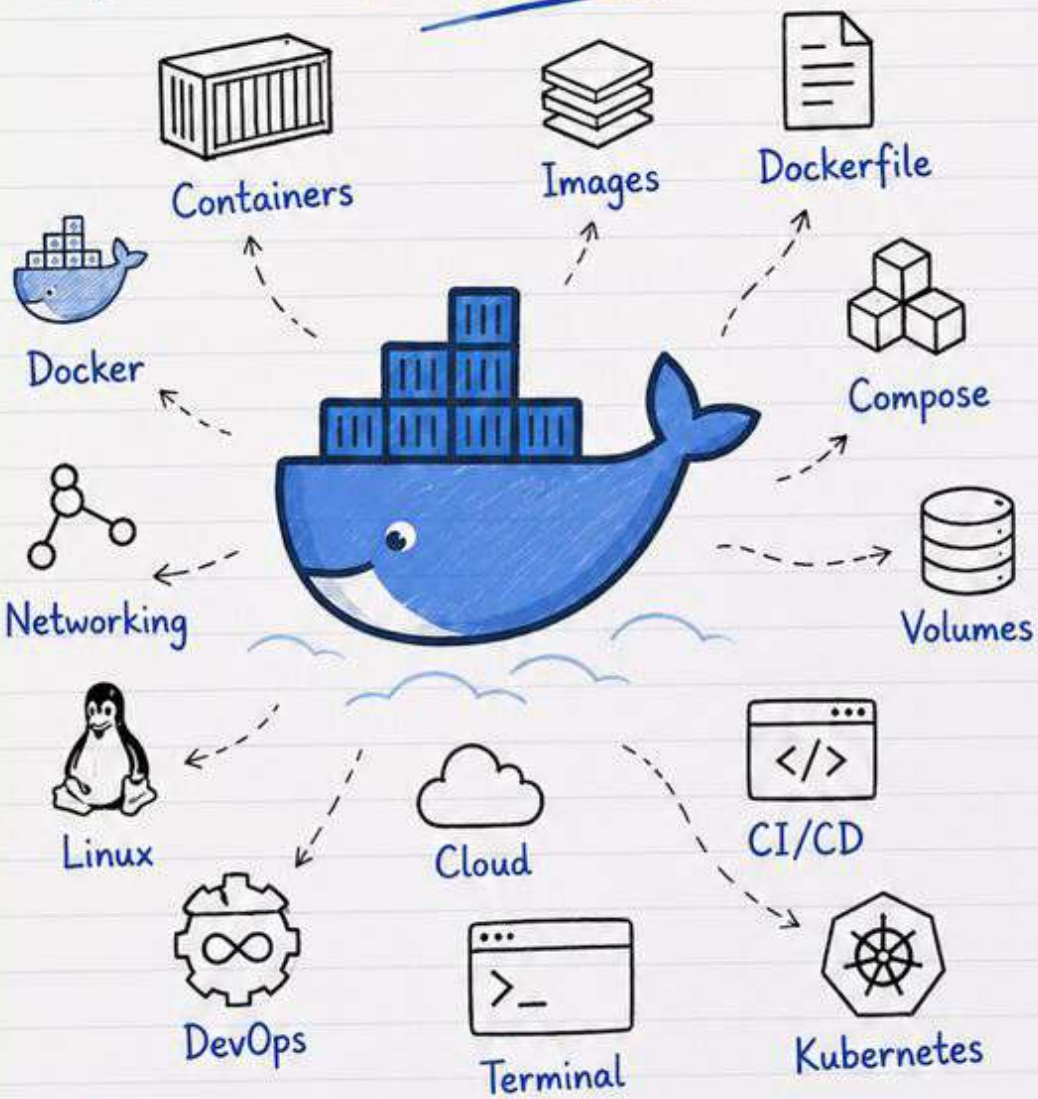
DOCKER COMPLETE NOTES

Beginner to Advanced



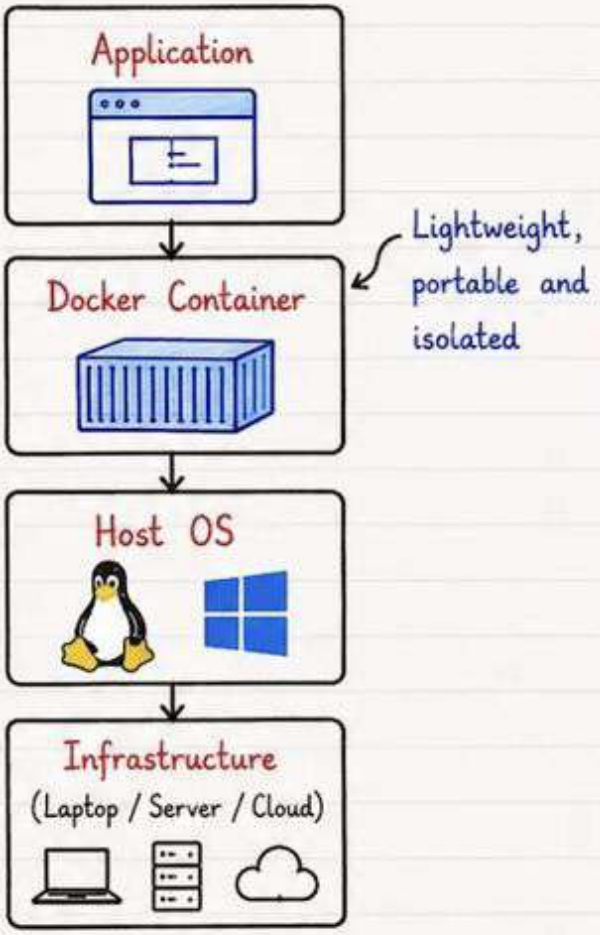
- ☑ 8 Detailed Notes
- ☑ Examples
- ☑ Interview Ready
- ☑ Architecture
- ☑ Command Cheat Sheet

Docker Notes

– Page 1 –

1. What is Docker ?

Docker is an open-source platform that helps us to build, ship and run applications inside containers. It allows an application to run consistently on any environment.



2. Why Docker was created ?

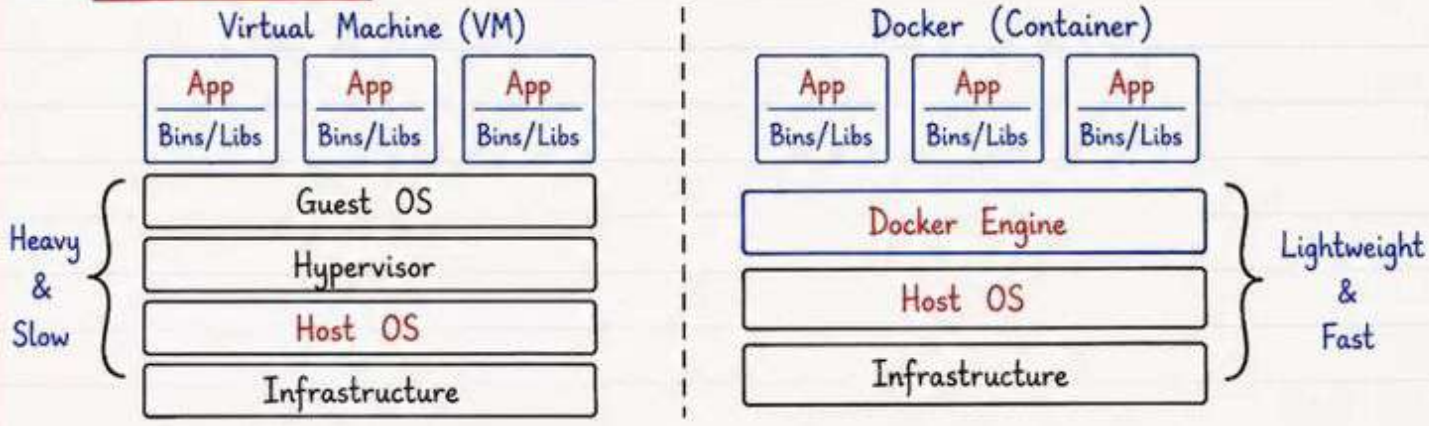
Before Docker, developers faced many issues while deploying applications.

- It works on my machine problem
- Different environments
- Heavy virtual machines
- Complex configuration
- Slow deployment

3. Benefits of Docker

- Lightweight : Containers are smaller and use fewer resources.
- Portable : Build once and run anywhere.
- Consistent : Same environment everywhere.
- Fast Deployment : Start containers in seconds.
- Isolation : Each container runs isolated from others.
- Scalability : Easy to scale applications.

4. VM vs Docker



Docker Architecture

Docker follows a client - server architecture.

1. Docker Client

The user interacts with Docker using Docker CLI (Command Line Interface) or REST API.

2. Docker Daemon

The Docker Daemon (Dockerd) listens to Docker API requests and manages Docker objects like images, containers, networks and volumes.

3. Docker Images

Images are read-only templates used to create containers. They are built using Dockerfile and stored in Docker Hub or local system.

4. Containers

Containers are runnable instances of images. They are lightweight, isolated and portable.

5. Networks

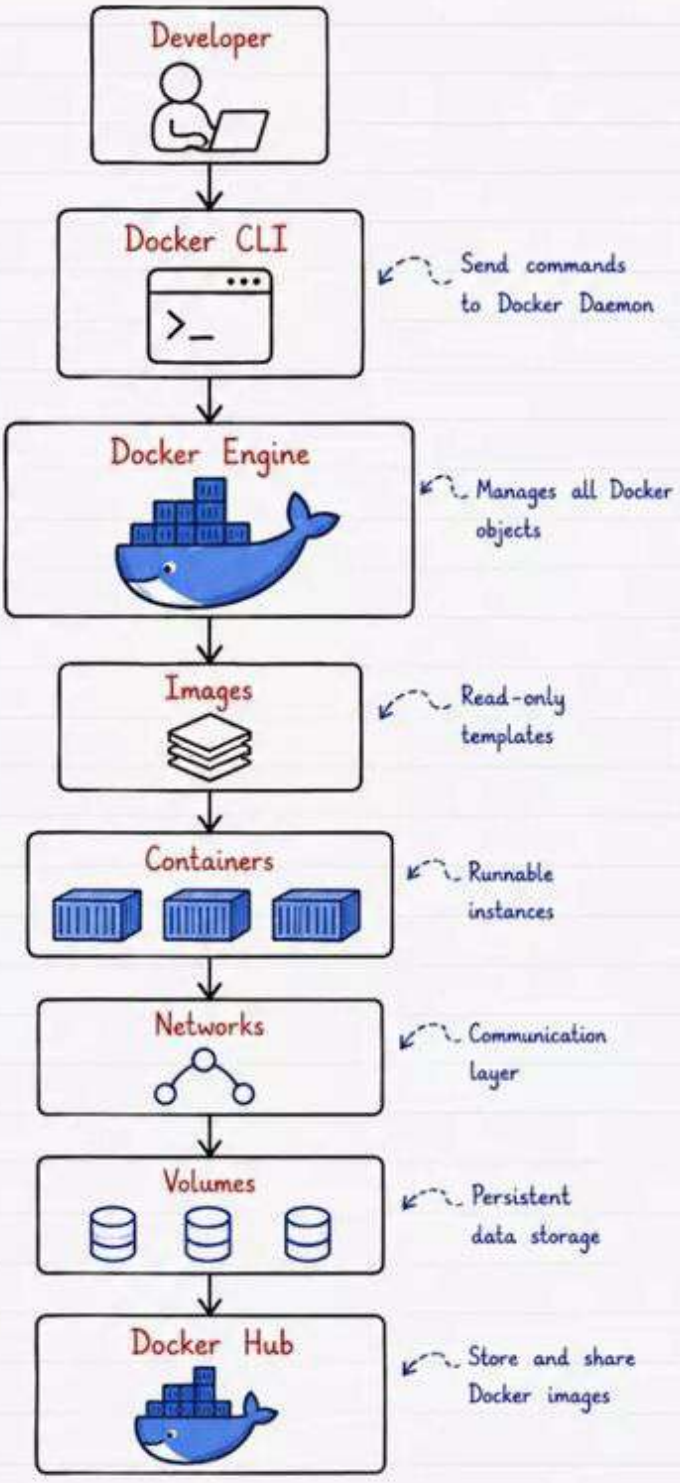
Networks allow containers to communicate with each other and with the outside world.

6. Volumes

Volumes are used to persist data generated by containers even if the container is removed.

7. Docker Hub

Docker Hub is a cloud registry service where images are stored, shared and pulled.



Key Points

- ☑ Docker uses a client-server architecture.
- ☑ Docker Daemon is the core of Docker.
- ☑ Images are used to create containers.
- ☑ Containers are isolated and portable.
- ☑ Networks enable communication.
- ☑ Volumes provide data persistence.
- ☑ Docker Hub stores and distributes images.

Remember :

- Build once
- Run anywhere
- Share easily
- Scale infinitely

IMAGE vs CONTAINER vs DOCKERFILE

1. IMAGE

- An image is a read-only template that contains the application, dependencies, libraries, tools and other files required to run the app.
- Images are used to create containers.

2. CONTAINER

- A container is a running instance of an image.
- Containers are lightweight, isolated and portable.

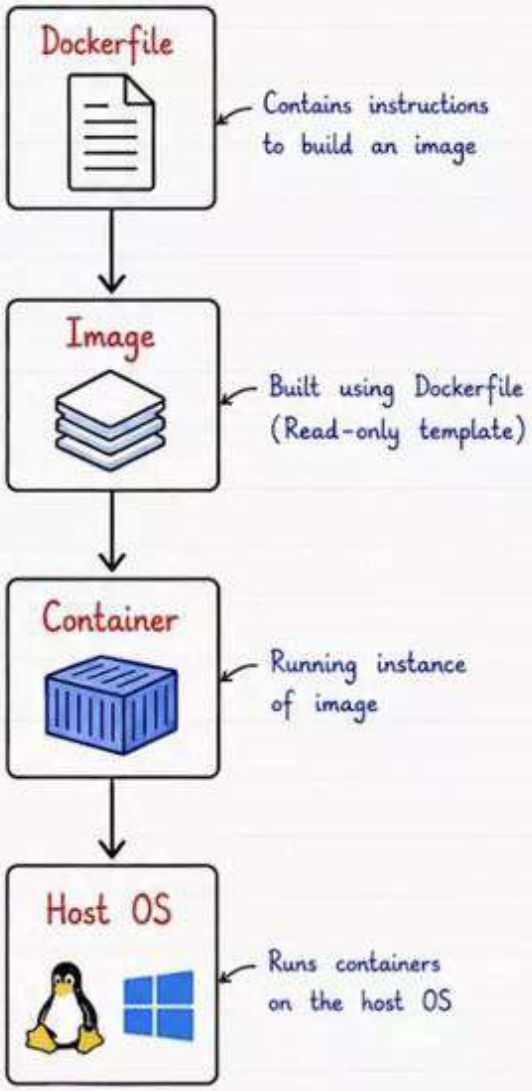
3. DOCKERFILE

- A Dockerfile is a text file that contains a set of instructions to build a Docker image.
- It is used by Docker to automate the image creation process.

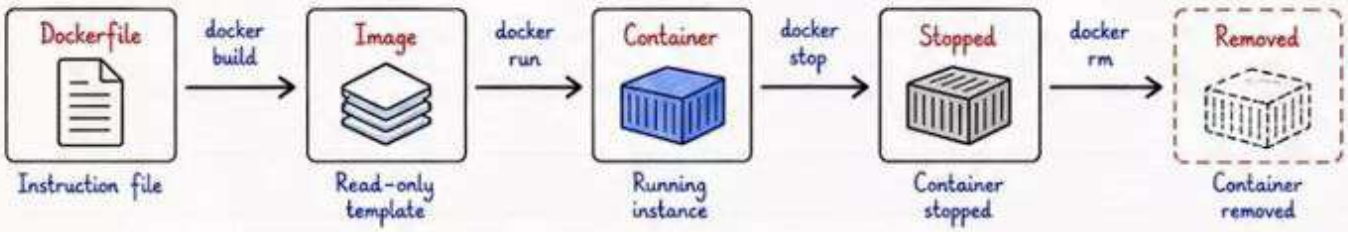
4. COMPARISON TABLE

Feature	Image	Container	Dockerfile
Definition	Read-only template	Running instance	Instruction file
State	Static	Dynamic	N/A
Purpose	To create containers	To run applications	To build images
Change	Cannot be changed	Can be changed	Can be edited
Life	Permanent until deleted	Temporary, can be stopped/removed	Persistent
Size	Larger	Smaller	Very small
Examples	mysql:8, nginx:latest	nginx container	Dockerfile

5. OVERVIEW DIAGRAM



6. DOCKER LIFECYCLE

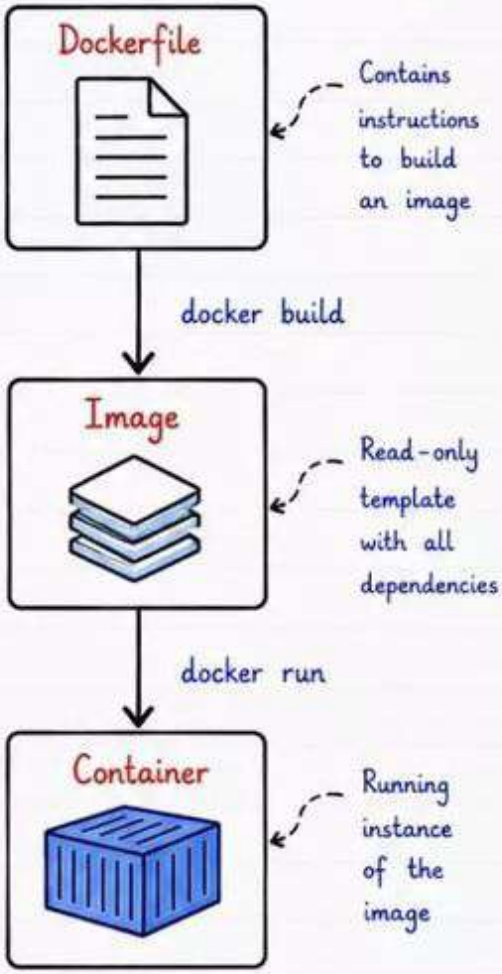


Build once → Run anywhere | Lightweight | Portable | Consistent

Dockerfile Explained

1. What is Dockerfile ?

A Dockerfile is a text file that contains a set of instructions to build a Docker image automatically.



2. Sample Dockerfile

```
1  FROM ubuntu:22.04                          # Base image
2  WORKDIR /app                               # Set working directory
3  COPY . /app                                # Copy files to image
4  RUN apt update && apt install -y python3   # Install python
5  RUN pip3 install flask                     # Install flask
6  ENV FLASK_ENV=production                   # Set environment
7  EXPOSE 5000                                # Expose port
8  CMD ["python3", "app.py"]                  # Default command
9  # Or using ENTRYPOINT
10 # ENTRYPOINT ["python3", "app.py"]
11 # CMD ["--host", "0.0.0.0"]
```

3. Important Instructions

- **FROM** : Specifies the base image.
- **WORKDIR** : Sets the working directory inside the container.
- **COPY** : Copies files or directories from host to container.
- **ADD** : Similar to COPY, also supports URLs and auto-extracts .tar files.
- **ENV** : Sets environment variables inside the container.
- **EXPOSE** : Informs Docker that the container listens on the given port.
- **CMD** : Specifies the default command to run when container starts.
- **ENTRYPOINT** : Configures a container that will run as an executable.

4. CMD vs ENTRYPOINT

CMD	ENTRYPOINT
• Provides default command and arguments. • Can be overridden when running container. • Only one CMD instruction is respected. • Used mostly for defining default behavior. • Example: CMD ["python3", "app.py"]	• Configures the container to run as executable. • Cannot be overridden easily. • Only one ENTRYPOINT instruction is respected. • Used when the container must always run a specific program. • Example: ENTRYPOINT ["python3", "app.py"]

Key Takeaway

- Dockerfile builds the image.
- Image is the blueprint.
- Container is the running instance.
- Build once, run anywhere!

Docker Networking

Docker containers can communicate with each other and with outside world using different types of networks.

1. Bridge (Default)

- Containers on the same bridge network can communicate with each other.

2. Host

- Container shares the host network. No isolation at network level.

3. None

- Container has no network access. Useful for complete isolation.

4. Overlay

- Used to connect containers across multiple Docker hosts (Swarm mode).

5. Storage in Docker

- Docker provides storage options to persist data generated by containers.

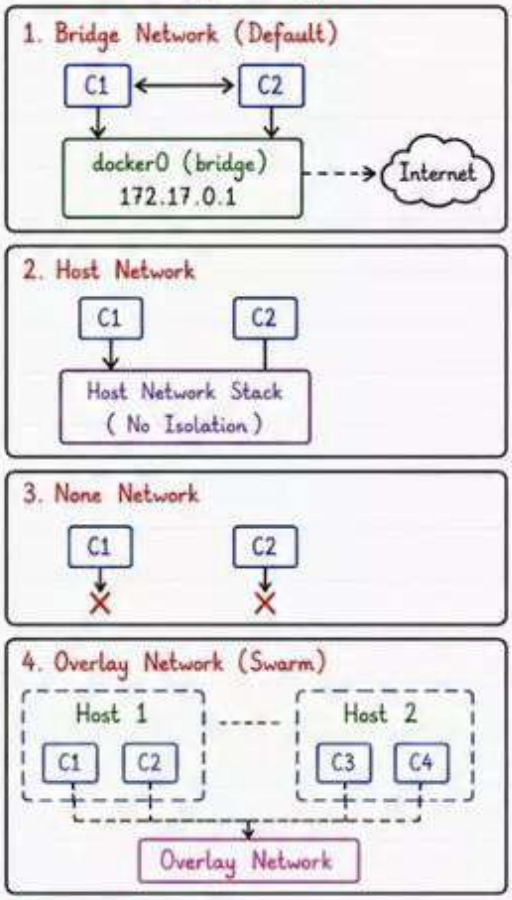
6. Volumes

- Managed by Docker. Stored in Docker host (/var/lib/docker/volumes). Recommended for production.

7. Bind Mounts

- Uses a directory from the host.
- Platform dependent.
- Good for development. (code changes reflect immediately).

Network Types Diagrams



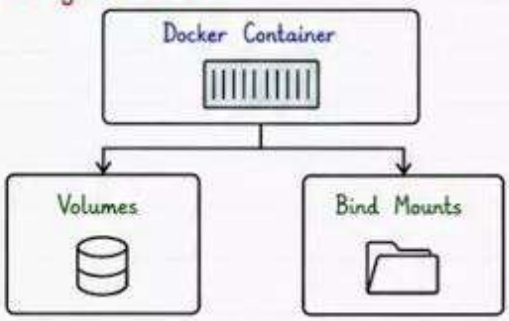
- Containers communicate with each other.
- Access to external network.

- Containers share the host network.
- Best performance.

- No network access
- Complete isolation.

- Connect containers across multiple hosts.
- Requires Docker Swarm.

Storage in Docker



- Data persists even if container is removed.
- Volumes are managed by Docker.
- Bind Mounts use host directories.

Volumes vs Bind Mounts

Feature	Volumes	Bind Mounts
Managed by	Docker	Host
Location	Inside Docker host	Anywhere on host system
Portability	High	Lower (depends on host path)
Backup	Easy (docker volume backup)	Manual
Best for	Production	Development
Performance	Better (Docker optimized)	Good

Key Takeaway : Use Volumes for production and Bind Mounts for development. Choose the right network based on isolation and communication needs.

Docker Compose

Docker Compose is a tool for defining and running multi-container Docker applications.

1. What is Docker Compose ?

- Docker Compose allows us to define and run multi-container Docker applications.
- It uses a YAML file to configure services, networks and volumes.
- With a single command, we can start, stop and manage entire application.

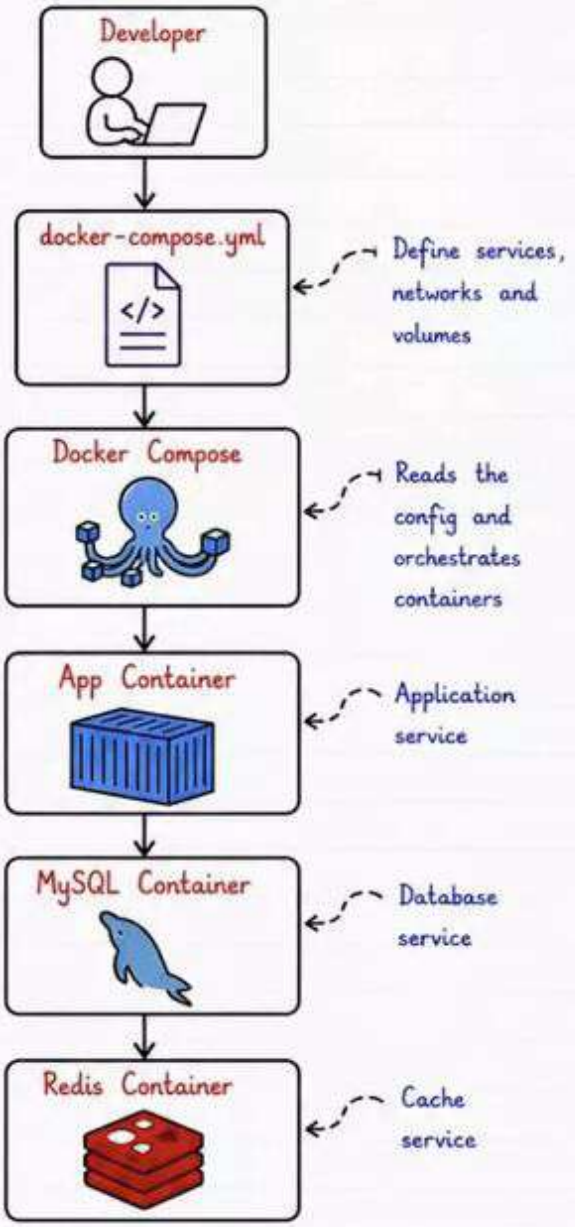
2. docker-compose.yml (Example)

```yaml
version: '3.8'                    # Compose file version
services:                         # All services
  app:                            # Application service
    build: ./app                  # Build from ./app
    container_name: myapp         # Name of the container
    ports:
      - "8080:8080"               # Port mapping host:container
    depends_on:
      - db                        # Depends on db service
    environment:
      - DB_HOST=db                # Environment variables
  db:
    image: mysql:8                # Database service
    container_name: mysql         # Use MySQL image
    environment:                  # Name of database container
      - MYSQL_ROOT_PASSWORD=root  # Set root password
```

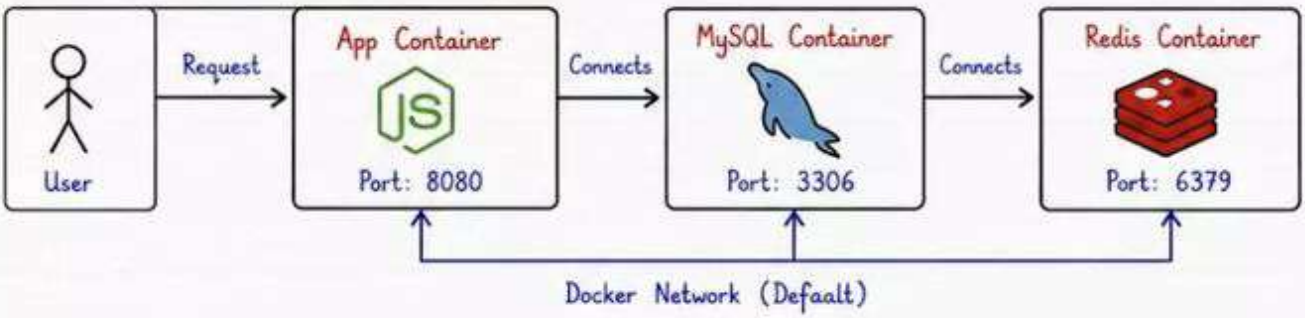
3. Important Commands

- `docker compose up` → Build (if needed) and start all services.
- `docker compose down` → Stop and remove containers, networks, volumes.
- `docker compose logs` → View logs of all services.
- `docker compose ps` → List all running services.
- `docker compose build` → Build or rebuild services.

4. How it Works ?



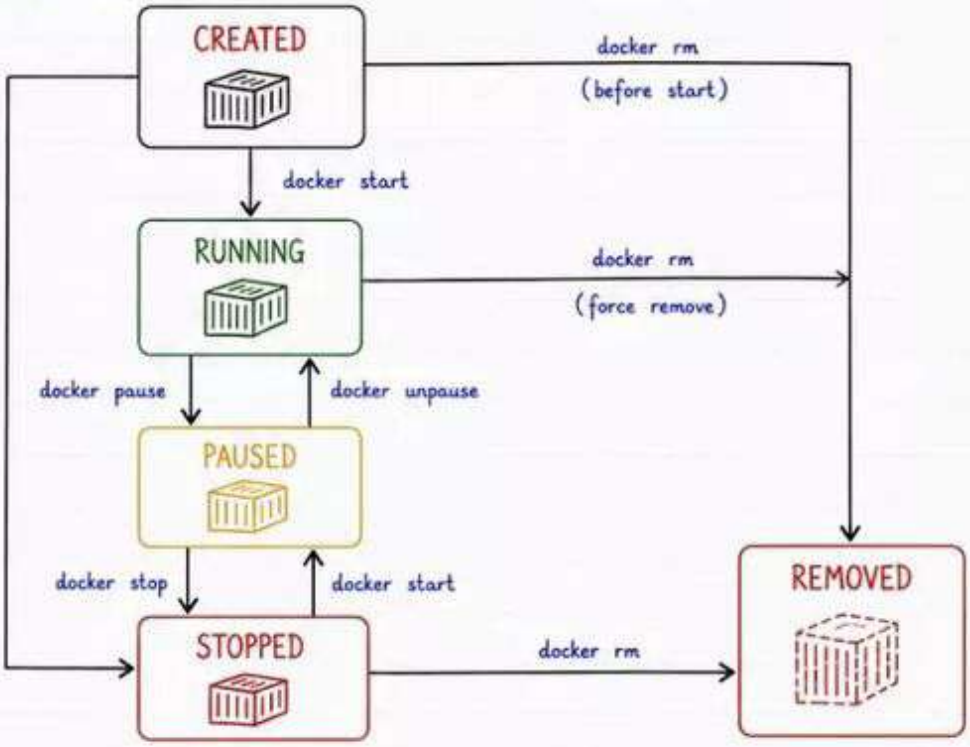
5. Multi Container Architecture (Example)



Docker Container Lifecycle

A Docker container goes through different states during its lifetime.

1. **Created**
   - Container is created but not started.
2. **Running**
   - Container is up and running.
3. **Paused**
   - Container is running but all processes are paused.
4. **Stopped**
   - Container is stopped but still exists.
5. **Removed**
   - Container is deleted and cannot be recovered.

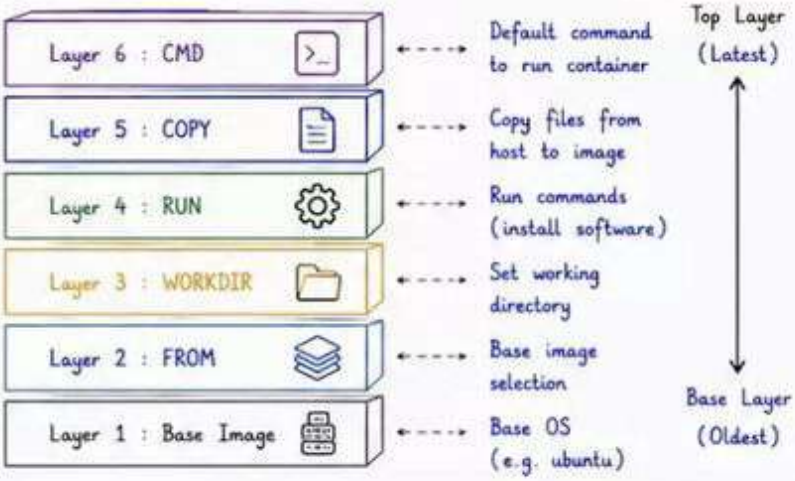


Remember:

- docker stop stops the container.
- docker start starts the stopped container.
- docker rm removes the container.
- docker rm -f force removes the container.

Docker Image Layers

Docker images are built using a layered architecture. Each instruction in the Dockerfile adds a new layer on top of the previous one.



Layer Caching

Layer 6	✗	Rebuilt (changes here)
Layer 5	✓	Cached (reused)
Layer 4	✓	Cached (reused)
Layer 3	✓	Cached (reused)
Layer 2	✓	Cached (reused)
Layer 1	✓	Cached (reused)

If a layer changes, all layers above it are rebuilt. Layers below it are reused from cache.

Key Takeaway: Build once → Run anywhere | Cache makes builds fast | Small layers = Better performance

COMMAND	DESCRIPTION	EXAMPLE
docker version	Show Docker version information	docker version
docker info	Display system-wide information	docker info
docker login	Log in to Docker Hub	docker login
docker logout	Log out from Docker Hub	docker logout
docker search	Search images on Docker Hub	docker search nginx
docker pull	Pull an image from Docker Hub	docker pull nginx
docker images	List all local Docker images	docker images
docker build	Build an image from a Dockerfile	docker build -t myapp .
docker run	Run a container from an image	docker run -d --name myctr nginx
docker ps	List running containers	docker ps
docker stop	Stop a running container	docker stop myctr
docker start	Start a stopped container	docker start myctr
docker restart	Restart a container	docker restart myctr
docker rm	Remove a container	docker rm myctr
docker logs	View logs of a container	docker logs myctr
docker exec	Execute a command in a container	docker exec -it myctr bash
docker inspect	Inspect a container or image	docker inspect myctr
docker cp	Copy files/folders between container and host	docker cp myctr:/app ./app
docker network ls	List all Docker networks	docker network ls
docker volume ls	List all Docker volumes	docker volume ls
docker compose up	Start services defined in docker-compose.yml	docker compose up -d
docker compose down	Stop and remove services, containers, networks	docker compose down
docker compose logs	View logs from services	docker compose logs -f
docker compose ps	List services defined in compose file	docker compose ps

Quick Tips :

- Use -d to run containers in detached mode.
- Use docker --help for more options.

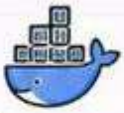
Answer Like a Pro!

INTERVIEW QUESTIONS

1. What is Docker?

Ans:

Docker is a platform that allows developers to build, package, and run applications in containers. It ensures consistency across different environments.



2. Difference between Image and Container?

Ans:

- Image is a read-only template with instructions to create a container.
- Container is a running instance of an image. It is writable and has a lifecycle.

3. CMD vs ENTRYPOINT?

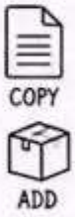
Ans:

- CMD provides default command and arguments.
- ENTRYPOINT configures a container that will run as an executable.
- CMD can be overridden, ENTRYPOINT cannot be easily overridden.

4. COPY vs ADD?

Ans:

- COPY copies files/directories from host to container.
- ADD does the same but also supports URLs and auto-extracts tar files.
- Use COPY for simple file copy.



5. Volume vs Bind Mount?

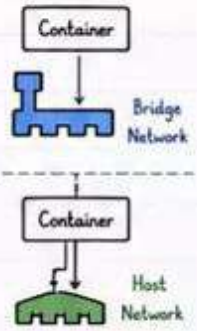
Ans:

- Volume: Managed by Docker, stored in Docker host (/var/lib/docker/volumes).
- Bind Mount: Maps a file or directory from the host to the container.
- Volumes are better for portability and backups. Bind mounts are good for dev.

6. Bridge vs Host Network?

Ans:

- Bridge: Default network. Containers communicate via Docker bridge (NAT).
- Host: Container shares the host's network namespace. No isolation.



7. What is Docker Compose?

Ans:

Docker Compose is a tool for defining and running multi-container Docker applications using a docker-compose.yml file.

8. What are Dockerfile Layers?

Ans:

Each instruction in a Dockerfile creates a new layer. Layers are cached and reused to make builds faster and smaller.

9. What is Docker Hub?

Ans:

Docker Hub is a cloud-based registry service provided by Docker to store and share container images.

10. What is Docker Registry?

Ans:

A Docker Registry stores Docker images. Docker Hub is a public registry. You can also run a private registry.

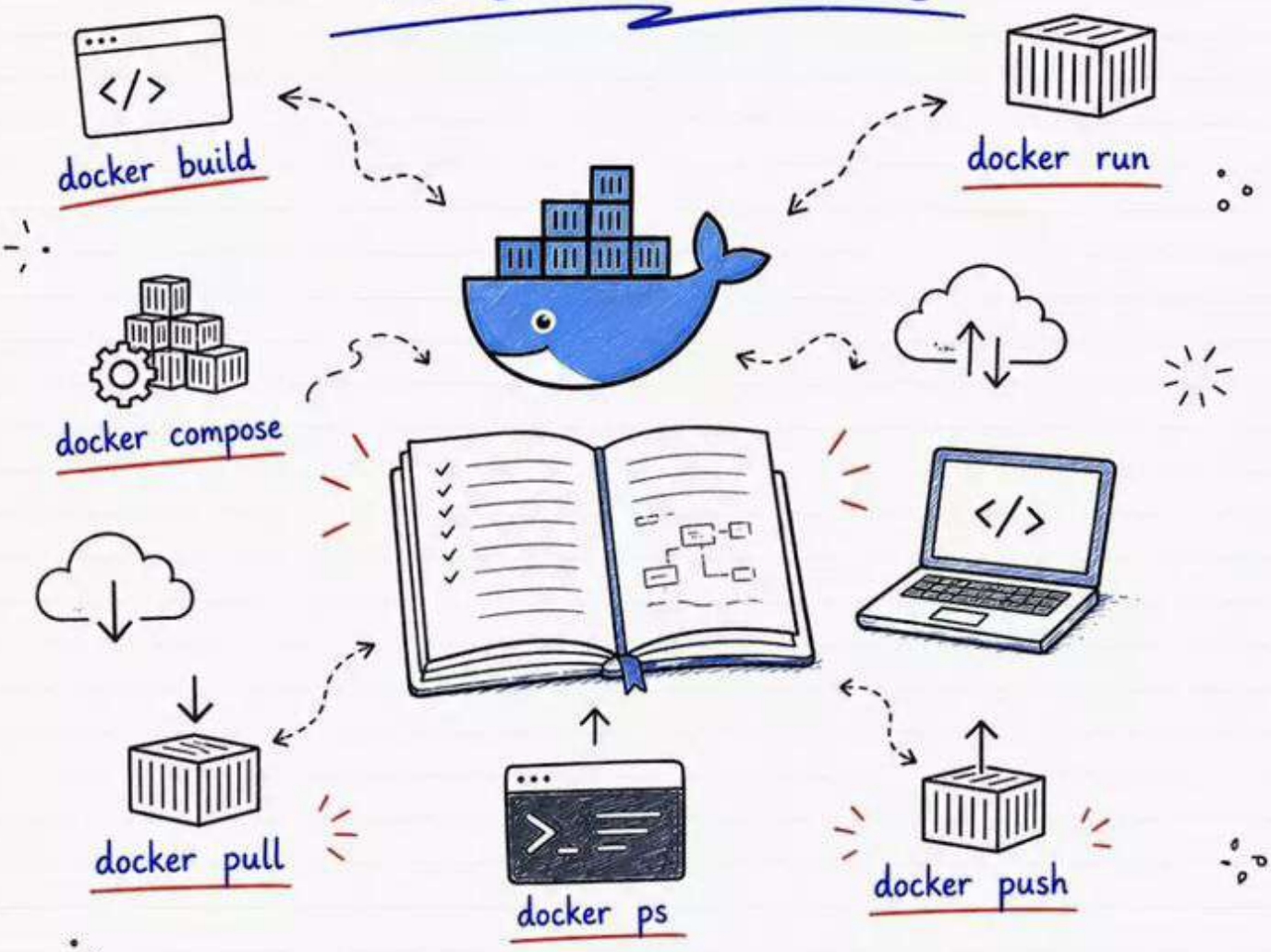
11. Docker Best Practices?

Ans:

- ✓ Use official base images.
- ✓ Keep images small (use .dockerignore).
- ✓ Don't run containers as root.
- ✓ Use specific image tags, avoid :latest.
- ✓ Clean up unused images, containers, volumes regularly.
- ✓ Store secrets using Docker secrets (not in Dockerfile).
- ✓ Use healthcheck to monitor containers

THANK YOU

Happy Learning!



> Practice Daily • Build Once • Run Anywhere

See You In The Next Notes