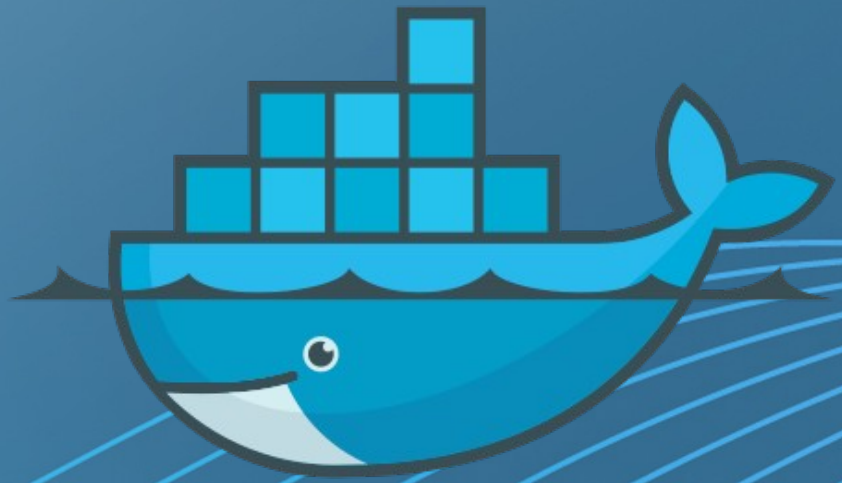


Docker course

Who am I?

TECH

- DevOps engineer @ CINQ ICT,
- DCA / CKA / Azure Solutions Architect
- Azure DevOps, MKE, MSR
- AI Specialist and enthusiast

Private

Married, father of two sons, and owner of two dogs.

- Sports (padel, soccer)

Topics of today

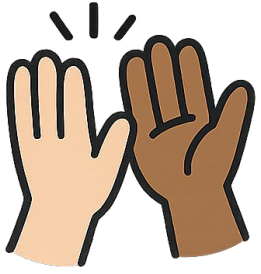
- Introduction & context
- Virtual Machines vs Containers
- Docker - Desktop
- Creating and managing containers
- Creating your own images
- Docker CLI essentials
- Understanding Docker Compose
- Networking & storage basics
- Best practices & troubleshooting
- Hands-on lab
- Q&A and wrap-up

git clone https://github.com/DennisVermeulen/docker-course.git

Training Style

Ask me anything

We do this course together



When I go too fast, tell me

What is Docker

Open-source project to run containers

github.com/moby/moby

Docker Engine / daemon

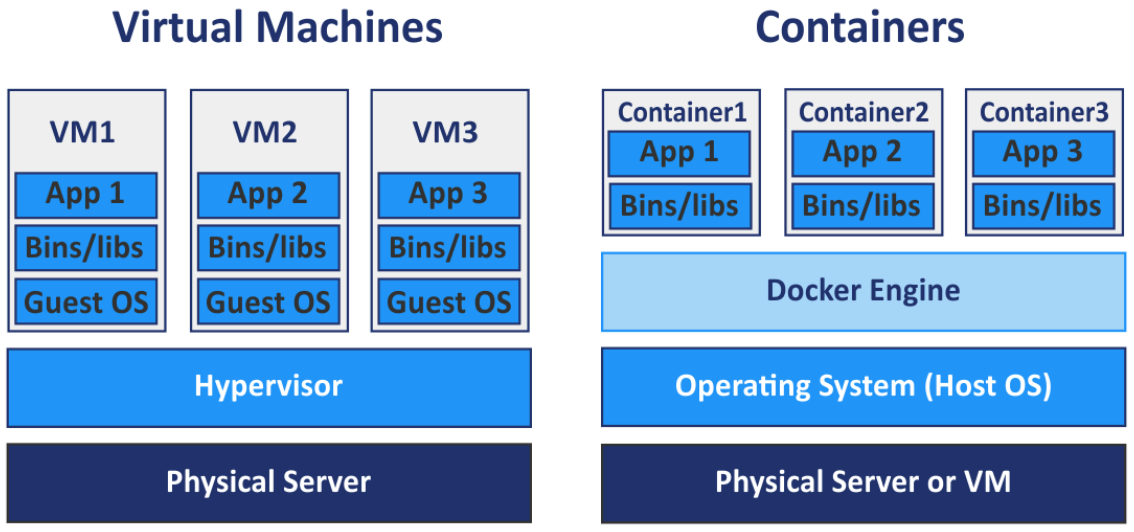
- Manage: images, containers, networks, volumes
- API on HTTP

Docker Client

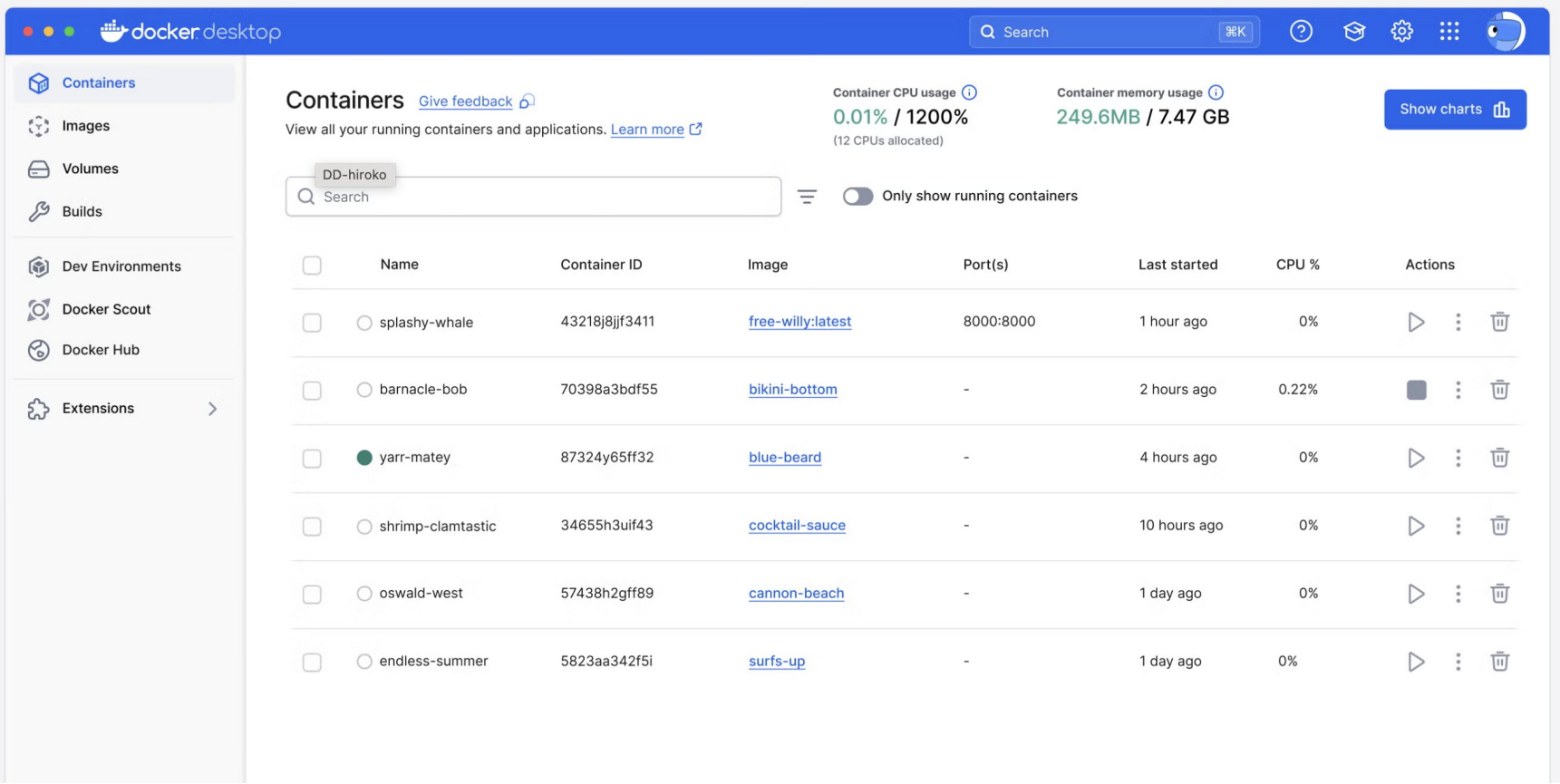
- CLI

Containers

- Package of app + dependecies in a standaridzed unit
- Share kernel of host OS
- Isolation
  - proces, network, filesystem
  - memory, cpu



Play with docker-desktop



Docker - Why Containers

Small images

- Resources
- Ship
- Secure
- Realtime scaling

Standardized build/deployment

- Independent of programming language
- Run locally, on VM's or in the cloud
- Fast

Support Agile / DevOps

- Build from code
- Small increments
- Micro services

What are dockerfiles, images & containers

Dockerfile

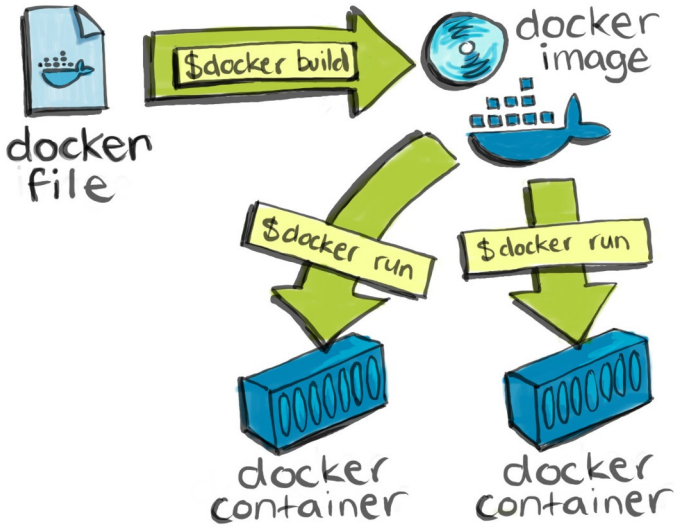
- Create image from code
- Extends base image
- Single file
- Default name: Dockerfile

Image

- Package app + dependencies
- Read-only
- Consist of layers
- Used to create container

Container

- Runnable instance of an image
- Add writeable layer
- Writeable layer in memory



What are dockerfiles, images & containers

Dockerfile

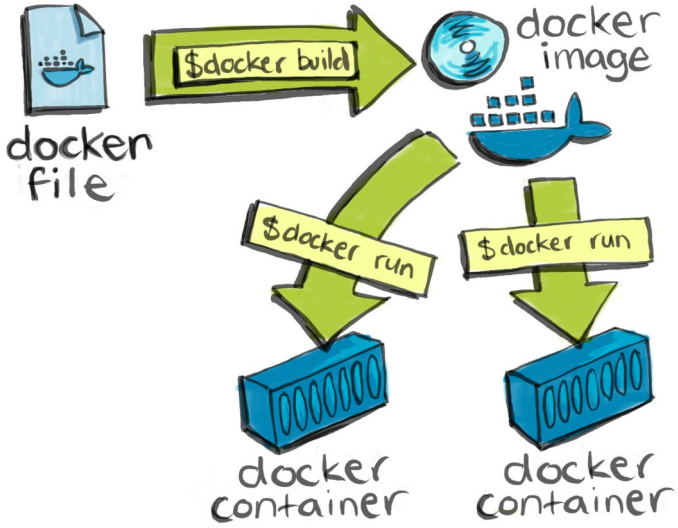
- Create image from code
- Extends base image
- Single file
- Default name: Dockerfile

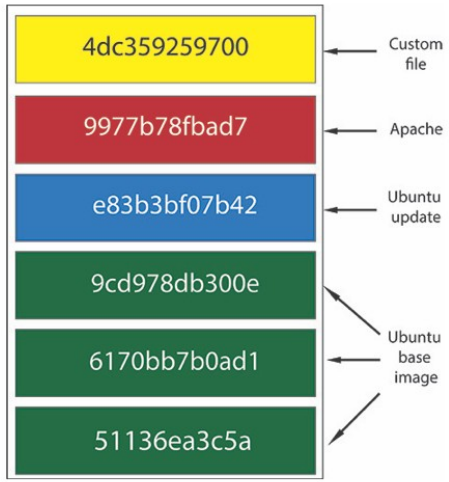
Image

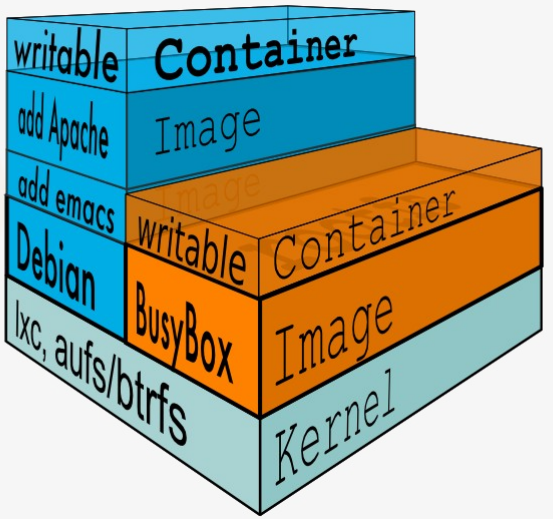
- Package app + dependencies
- Read-only
- Consist of layers
- Used to create container

Container

- Runnable instance of an image
- Add writeable layer
- Writeable layer in memory







What Are Docker Image Layers?

Each Docker image consists of layers stacked on top of each other. This concept originates from the Union File System.

- How does it work?
- Each instruction in a Dockerfile that results in a change (such as RUN, COPY, ADD) creates a new layer.
- Layers are cached, so if nothing changes in a given layer, Docker can reuse the previously built layer.
- Docker uses this caching mechanism to speed up builds.

⚡ Best Practices for Fast Builds

1. Put Frequently Changing Instructions at the Bottom

Docker builds from top to bottom. If a layer near the top changes, all layers beneath it will be rebuilt.

✅ Good example:

```
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
```

❌ Bad example:

```
COPY . .
RUN pip install -r requirements.txt
```

Why?

In the bad example, any change in your code triggers a full rebuild, including reinstalling dependencies.

LAB 1.0

⚡ Best Practices for Fast Builds

2. Combine RUN Instructions Where Possible

✅ Efficient:

```
RUN apt update && apt install -y curl && rm -rf /var/lib/apt/lists/*
```

❌ Inefficient:

```
RUN apt update
RUN apt install curl
```

Why?

Each RUN instruction creates a new layer.

LAB 1.1

⚡ Best Practices for Fast Builds

3. Use `.dockerignore`

Avoid including unnecessary files in the build context (`e.g., .git/, node_modules/, __pycache__/`).

This speeds up builds and improves caching

Example `.dockerignore:`.

```
.git
__pycache__/
node_modules/
*.log
```

⚡ Best Practices for Fast Builds

4. Use Lightweight Base Images

Choose smaller base images to reduce build time and image size.

✅ Examples:

`python:3.12-slim` instead of `python:3.12`

Docker Build Best Practices

Practice	Why
Use .dockerignore	Smaller build context, faster builds
Order COPY and RUN	Better caching and reuse
Choose minimal base image	Smaller images, faster downloads
Combine RUN statements	Fewer layers, less overhead

Why Dockerfiles

Image as code

- Version control
- One source of truth
- Easy collaboration and sharing

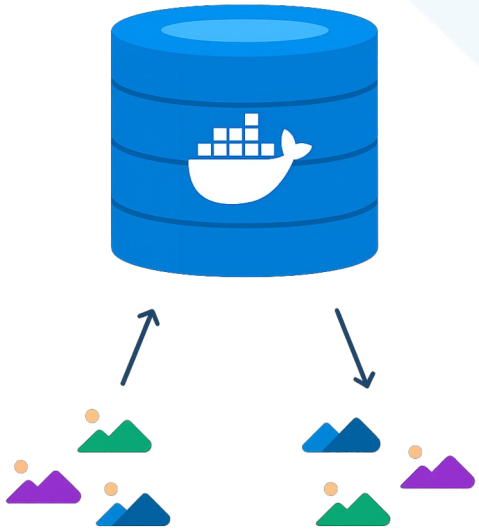
Easy

- Just the easiest way to build images

Why Registry

What

- Store images
- Central place



Why

- Distribution
- Traceability
- Optional:

  CI/CD,

  Vulnerability scanning,

  Notary (image trust),

Examples

- Docker Hub (public)
- On-premise registries
- PaaS:

  Azure Container Registry (ACR),

  JFrog

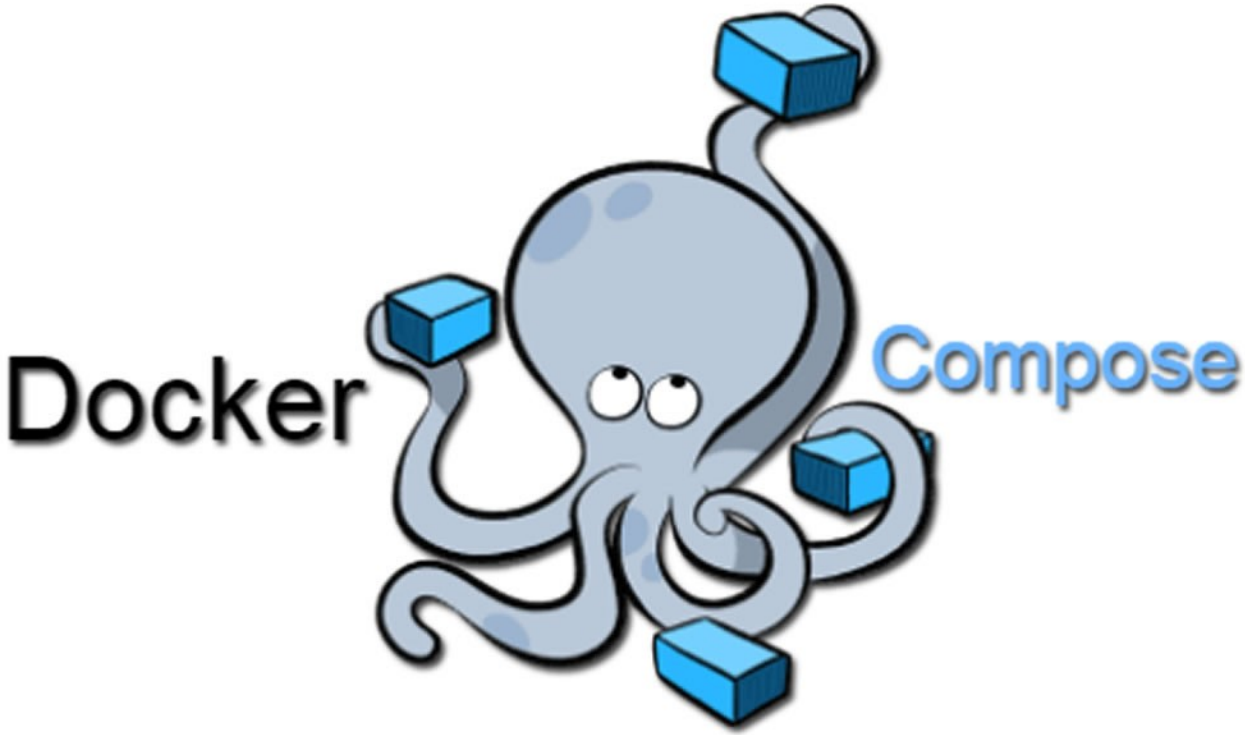
docker-cli

```
-d (daemonize) = run the container in the background
-p = portforwarding into the host

$ docker ps (list running containers)
$ docker ps -a (list all containers, including stopped)
$ docker images (list images)
$ docker pull ubuntu (download ubuntu image)
```

LAB APP

Docker RECAP

- What is Docker
- Why containers
- What are images, Dockerfiles & containers
- Why Dockerfiles
  Dockfile INSTRUCTIONS
- Docker CLI
- Labs: Install LAB1.0, LAB1.1, LAB APP
- Dockerfile – Best practices
  Container Registry


Docker
Compose

What is Docker-Compose

Docker Compose is a tool for defining and running multi-container Docker applications. With Docker Compose, you use a Compose file to configure the services of your application. Then, with a single command, you create and start all the services defined in the configuration.

By using Docker Compose, we can run each of these technologies in separate containers on the same host and enable them to communicate with one another. Each container will expose a port for communication with other containers.

The communication and uptime of these containers is managed by Docker Compose.

Three-Step Process

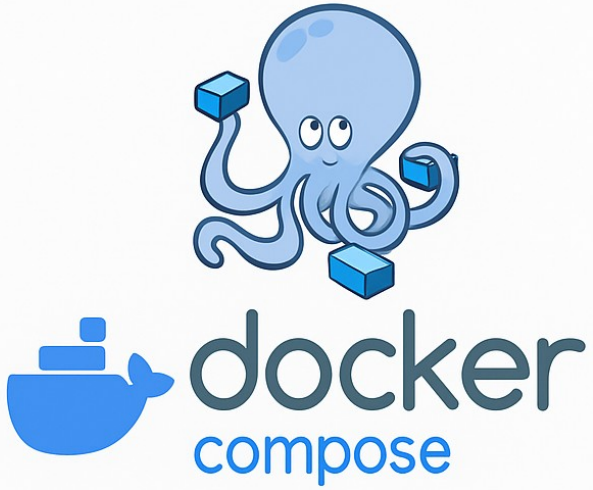
1. Define your app's environment with a Dockerfile so it can be reproduced anywhere.

2. Define the services that make up your app in docker-compose.yml, so they can run together in an isolated environment.

3. Finally, run docker-compose up or docker-compose start to launch the entire application.

Docker Compose Options

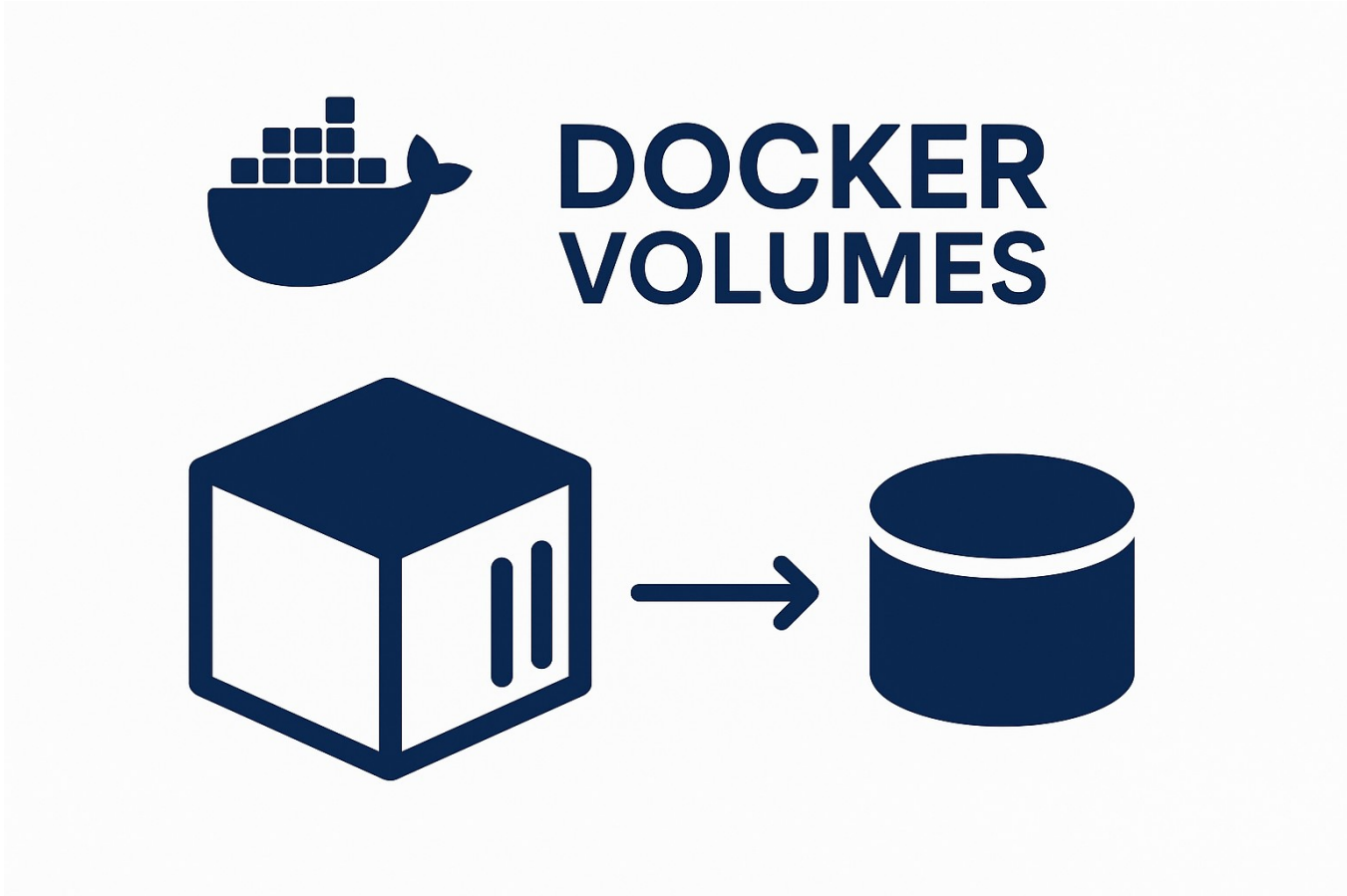
- Environment variables
- Volumes
- Networks
- Ports
- Driver



```yaml
services:
  ▷Run Service
  db:
    image: postgres:16-alpine
    restart: unless-stopped
    environment:
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
      POSTGRES_DB: ${POSTGRES_DB}
    ports:
      - "5432:5432"
    volumes:
      - ./db-data:/var/lib/postgresql/data
      - ./db-init:/docker-entrypoint-initdb.d
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U ${POSTGRES_USER}"]
      interval: 5s
      timeout: 5s
      retries: 5

  ▷Run Service
  app:
    build: ./app
    command: ["node", "index.js"]
    ports:
      - "3000:3000"
    environment:
      DB_HOST: db
      DB_PORT: 5432
      DB_USER: ${POSTGRES_USER}
      DB_PASSWORD: ${POSTGRES_PASSWORD}
      DB_NAME: ${POSTGRES_DB}
    depends_on:
      db:
        condition: service_healthy

volumes:
  db-data:
    driver: local

networks:
  backend-net:
    driver: bridge
```

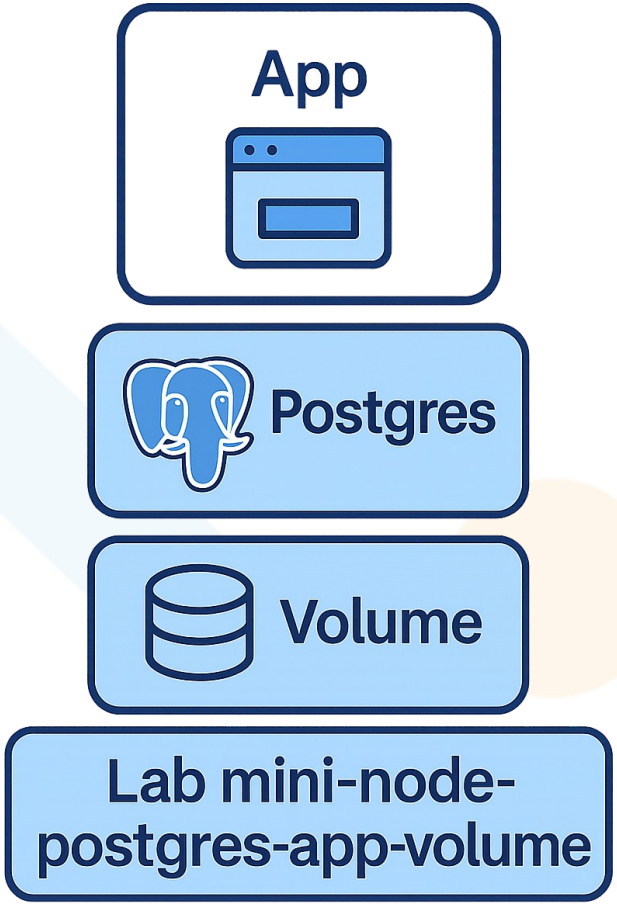
Make build with docker compose

LAB: mini-node-postgres-app


DOCKER
VOLUMES

Difference Between Named and Unnamed (Anonymous) Volumes

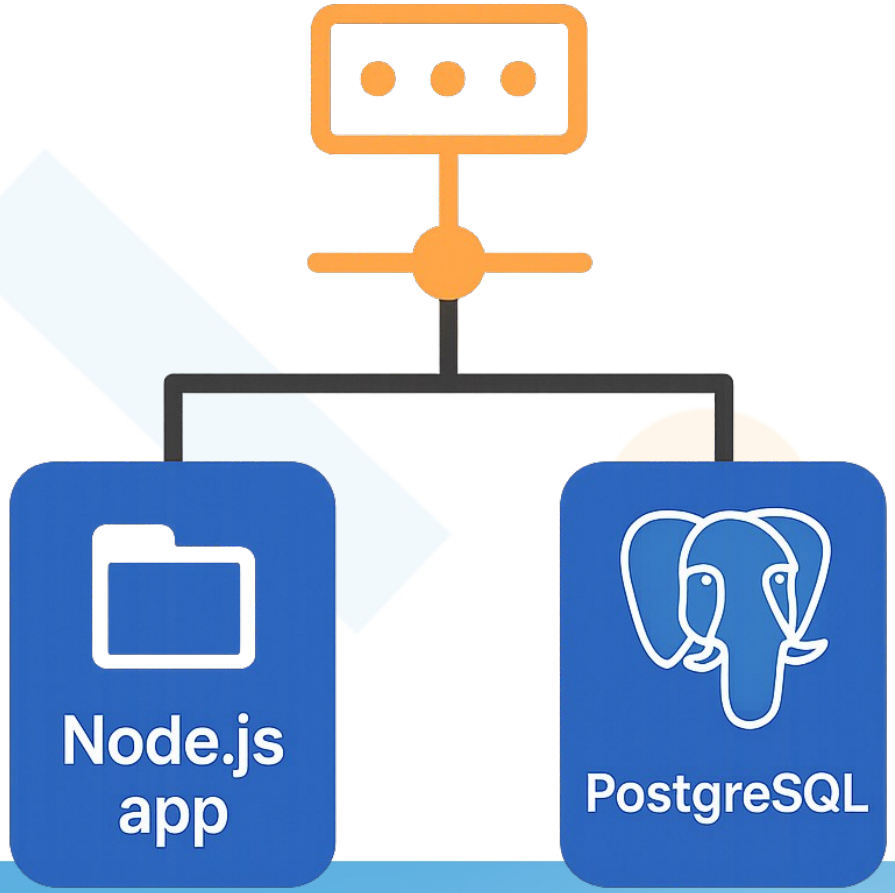
Name specified	Named Volume	Unnamed (Anonymous) Volume
Reusable	✅ Yes (you define the name)	❌ No (Docker generates a random name)
Manageable	✅ Yes (can be reused across containers or runs)	❌ No (not easily reusable)
Declared in Compose	Usually declared under *volumes:*section	Typically used inline, without declaration
Persistence	Survives *docker compose down*	Removed with *docker compose down -v*
Volume path	/var/lib/docker/volumes/db-data/_data	/var/lib/docker/volumes/random-id/_data


App
Postgres
Volume
Lab mini-node-postgres-app-volume

What is a Docker network?

In Docker, a network is a virtual channel that allows containers to communicate with each other, the host, and external systems. Every container you start is part of a network.

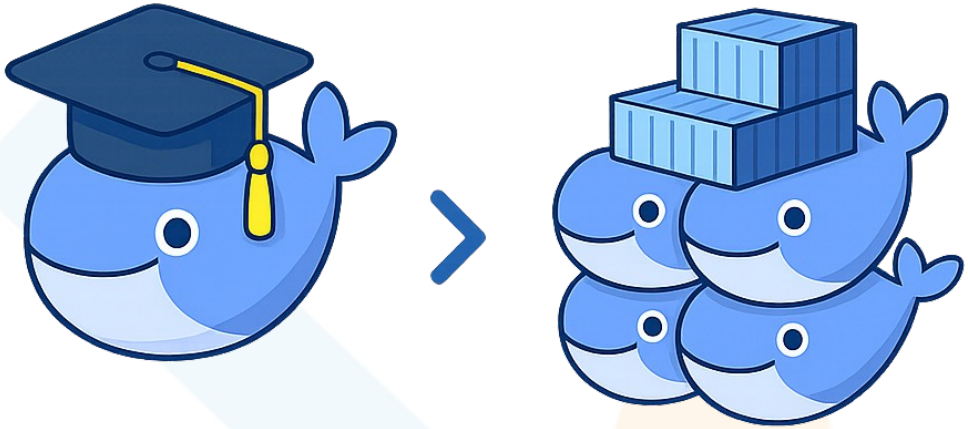
Lab mini-node-postgres-app-network



🔌 Important network types

Docker automatically creates several network types:

Type	Description
`bridge`	**Default network** for standalone containers. Containers can communicate with each other via their container name.
`host`	The container shares the host's network stack. No isolation (Linux only).
custom	User-defined bridge networks with improved DNS resolution and isolation.



docker

docker
swarm

What is Docker Swarm?

Docker Swarm is Docker's native clustering and container orchestration tool. It allows you to group multiple Docker hosts (machines) into a single virtual cluster that you can deploy, manage, and scale containers on – as if they were running on one machine.

Kubernetes and Docker Swarm

Kubernetes	Docker Swarm
Developed by Google	Developed by Docker, Inc.
Container orchestration system	Container orchestration tool


portainer.io