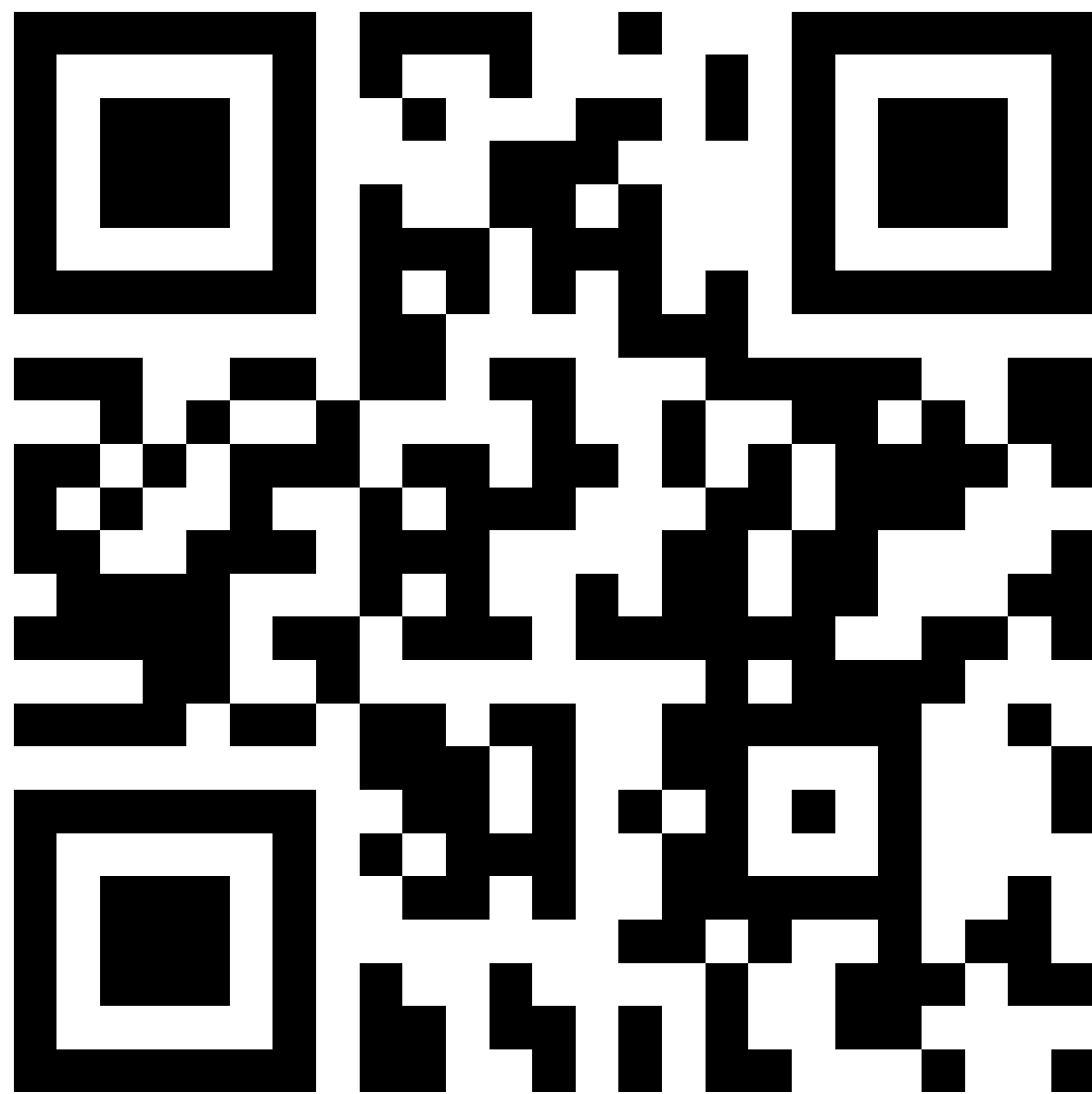


Web Engineering

DHBW Stuttgart · Informatik / Wirtschaftsinformatik

Sommersemester 2026 · Sitzung 9 — Docker, Proxies and DBs

<https://librete.ch/dhbw/>



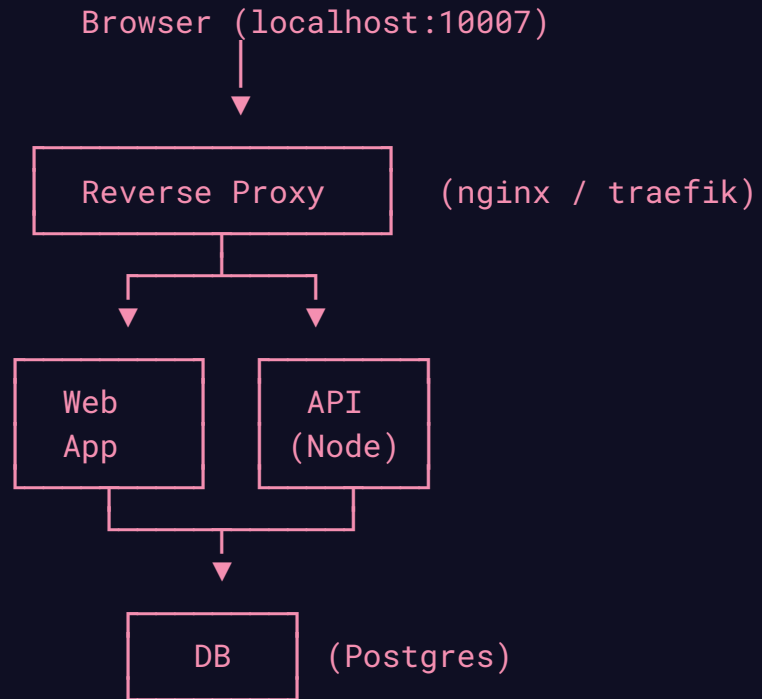
Docker

Service Orchestration

Inhalt

1. Architektur: Reverse Proxy + API + DB
2. Container vs VM
3. `Dockerfile`
4. `compose.yml`
5. Docker CLI / Compose Befehle
6. Live Demo: `dhbw-docker`

Architektur



Container vs VM

	VM	Container
Kernel	eigener	Host-Kernel
Boot	Minuten	Sekunden
Größe	GB	MB
Isolation	stark	namespaces, cgroups
Image	komplett	Layer (diff)

→ **Docker** = Container-Engine + Image-Format + Registry.

Dockerfile – Express API

```
FROM node:lts-slim AS builder

WORKDIR /app

COPY ./package*.json ./
RUN npm ci --no-audit --no-fund --no-progress --loglevel=error

COPY ./ ./

ENV NODE_ENV="production"

EXPOSE 8080

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

npm ci statt **npm i** für reproduzierbare Builds.

Multi-Stage Build

```
FROM node:lts-slim AS builder
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build

FROM node:lts-slim AS runtime
WORKDIR /app
COPY --from=builder /app/dist ./dist
COPY --from=builder /app/node_modules ./node_modules
COPY package*.json ./
EXPOSE 8080
CMD ["node", "dist/index.js"]
```

→ kleines Final-Image, keine Build-Tools im Runtime.

.dockerignore

```
node_modules  
dist  
.git  
.env  
*.log  
coverage
```

Spart Build-Kontext und verhindert Geheimnis-Leaks.

compose.yml – Services

```
name: dhbw-docker-app

services:
  api:
    build: ./api
    container_name: dhbw-api
    environment:
      DATABASE_USERNAME: ${DATABASE_USERNAME}
      DATABASE_PASSWORD: ${DATABASE_PASSWORD}
      DATABASE_NAME: ${DATABASE_NAME}
      DATABASE_HOST: db
      DATABASE_PORT: 5432
    ports:
      - "10000:8080"
    networks: [net, data]
    depends_on:
      db:
        condition: service_healthy
```

compose.yml – Datenbank + Proxy

```
db:
  image: postgres:16-alpine
  environment:
    POSTGRES_USER: ${DATABASE_USERNAME}
    POSTGRES_PASSWORD: ${DATABASE_PASSWORD}
    POSTGRES_DB: ${DATABASE_NAME}
  volumes:
    - dbdata:/var/lib/postgresql/data
  networks: [data]
  healthcheck:
    test: ["CMD-SHELL", "pg_isready -U $$DATABASE_USERNAME"]
    interval: 5s
    retries: 5

proxy:
  image: nginx:alpine
  ports: ["10007:80"]
  volumes: [".:/nginx.conf:/etc/nginx/nginx.conf:ro"]
  networks: [net]
  depends_on: [api]

volumes:
  dbdata:

networks:
  net:
  data:
```

Docker Compose – Befehle

```
docker compose -p "dhbw-docker-app" \  
  -f compose.yml \  
  --env-file .env \  
  up \  
  --build \  
  --remove-orphans \  
  --force-recreate
```

Flag	Bedeutung
-p	Projektname
-f	mehrere Compose-Files möglich
--env-file	.env einlesen
--build	Images neu bauen
--remove-orphans	verwaiste Container weg
--force-recreate	Container neu starten

Compose – Alltag

```
docker compose up -d           # detached starten
docker compose ps              # Status
docker compose logs -f api     # Logs streamen
docker compose exec api sh     # Shell im Container
docker compose down -v         # Stop + Volumes löschen
docker compose restart api     # Neustart einzelner Service
```

Health Checks

```
api:  
  healthcheck:  
    test: ["CMD", "curl", "-f", "http://localhost:8080/health"]  
    interval: 10s  
    timeout: 3s  
    retries: 3  
    start_period: 5s
```

`depends_on: { condition: service_healthy }` wartet, bis Health passt.

Networks: Segmentierung

- **net** – öffentlich erreichbar (Proxy – App)
- **data** – nur intern (App – DB)
- DB ist **nicht** vom Internet erreichbar
- Service-DNS: **api**, **db**, **proxy** (Container-Name = Hostname)

→ Defense in Depth.

Volumes – Daten persistent

```
volumes:  
  dbdata:           # benannt  
  - ./code:/app     # bind mount (dev)  
  - /app/node_modules # anonymous (überschreibt bind)
```

`docker compose down` ohne `-v` lässt Volumes leben.

Live Demo

Repo: <https://github.com/nextlevelshit/dhbw-docker>

```
git clone https://github.com/nextlevelshit/dhbw-docker
cd dbw-docker
cp .env.example .env
docker compose up --build
```

→ <http://localhost:10007>

100 Concepts of Docker (Exkurs)

- Image • Container • Volume • Network
- Dockerfile • Layer Cache • BuildKit
- Compose • Stack • Swarm • Kubernetes
- Registry • Tag • Digest
- Dev/Prod-Parity (12-Factor)
- Secrets • Configs
- Logging Drivers
- Health Checks • Restart Policies