

Docker Compose

GlobeNotes



Inhaltsverzeichnis

- ▶ Rezepteblog
- ▶ Grundstruktur
- ▶ Wordpress
- ▶ Infrastruktur
- ▶ CI/CD-Pipeline
- ▶ Wechsel des Projekt
- ▶ Global Notes
- ▶ Infrastuktur
- ▶ Konfiguration
- ▶ Testplan
- ▶ Installation

Rezepteblog

- ▶ Ursprüngliche Idee: WordPress-basierter Rezepteblog mit MariaDB in Docker-Containern
- ▶ CI/CD geplant für automatisiertes Deployment
- ▶ Konzept später verworfen zugunsten eines neuen Projektansatzes

Grundstruktur

```
└─ Projekt/
  │
  ├─ .gitlab-ci.yml
  │
  │
  ├─ infra/
  │   │
  │   ├─ docker-compose.yml
  │   │
  │   ├─ .env.example
  │   │
  │   └─ .gitkeep
  │
  │
  ├─ app/
  │   │
  │   └─ .gitkeep
  │
  │
  ├─ pipelines/
  │   │
  │   ├─ DEPLOYMENT-NOTIZEN.md (optional)
  │   │
  │   └─ .gitkeep
  │
  └─ README.md (optional: erklärt Zweck der Ordner)
```

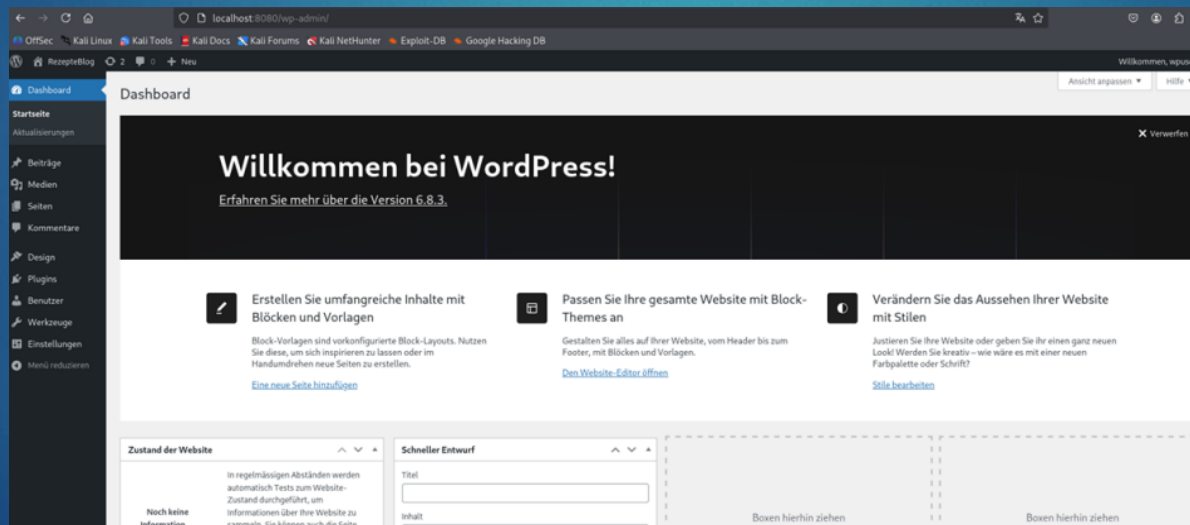
Git add.

Git commit -m "Projektstruktur angelegt"

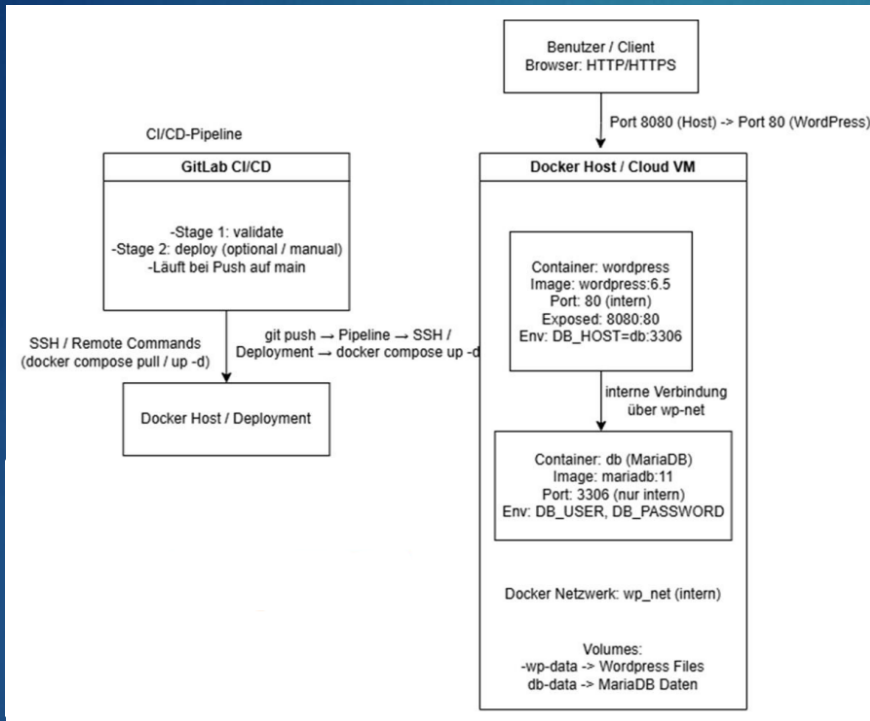
Git push

WordPress

- ▶ docker-compose.yml im Ordner Projekt/infra/ anlegen.
- ▶ Starten: `cd Projekt/infra`
`docker compose up -d`
- ▶ Weboberfläche via `http://localhost:8080` prüfen

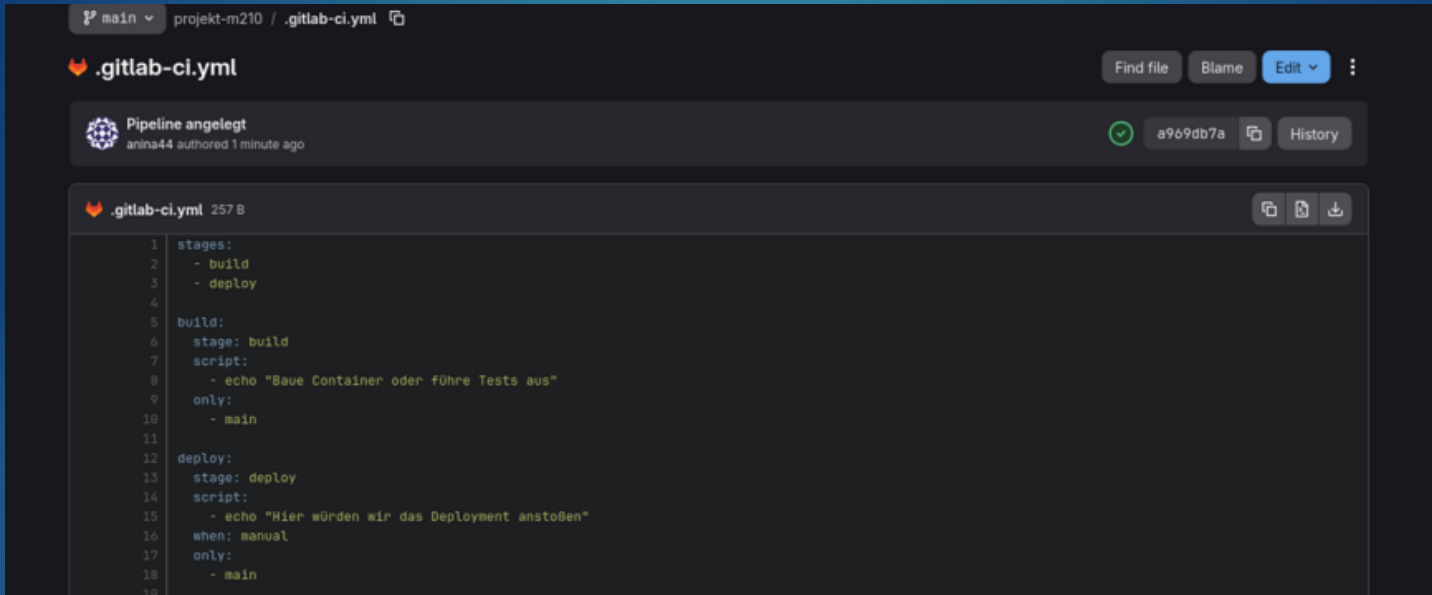


Infrastruktur



- ▶ Containerisierte Infrastruktur mit WordPress-Webserver & MariaDB
- ▶ Verwaltung über Docker Compose, internes Docker-Netzwerk
- ▶ WordPress extern über Port 8080 erreichbar, DB nur intern
- ▶ Persistenz durch Docker-Volumes für Dateien & Datenbank

CI/CD-Pipeline



```
1 stages:
2   - build
3   - deploy
4
5 build:
6   stage: build
7   script:
8     - echo "Baue Container oder führe Tests aus"
9   only:
10    - main
11
12 deploy:
13   stage: deploy
14   script:
15     - echo "Hier würden wir das Deployment anstoßen"
16   when: manual
17   only:
18     - main
19
```

- ▶ Anzeige der Datei **.gitlab-ci.yml** im GitLab-Repository
- ▶ Pipeline mit zwei Stages: **build** und **deploy**
- ▶ *Build*-Job führt einfachen Echo-Test aus
- ▶ *Deploy*-Job ist manuell auslösbar („when: manual“)
- ▶ Pipeline wurde erfolgreich erstellt

Wechsel des Projekts

Wir entschieden uns statt einer WordPress-Umgebung unser bestehendes Projekt **GlobeNotes** weiterzuentwickeln und darin die CI/CD-Anforderungen zu integrieren.

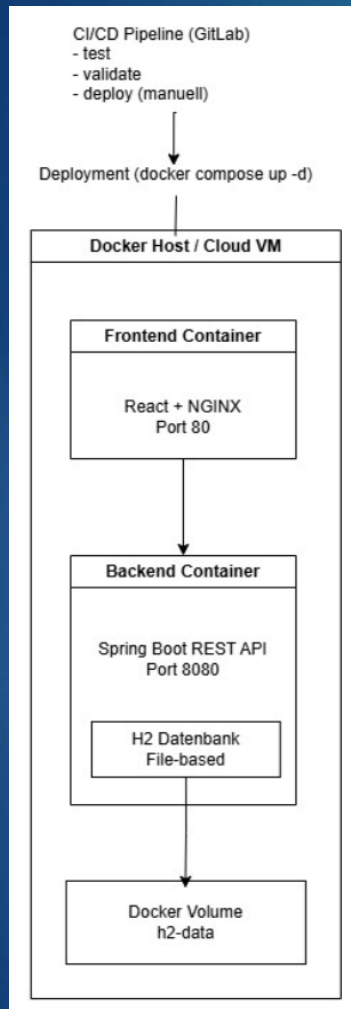
So konnten wir die Modulziele praxisnah erfüllen und ein realistischeres Anwendungsszenario nutzen.

GLOBAL NOTES

Global Notes

- ▶ Webapp *GlobeNotes* + persistente H2-Datenbank
- ▶ Betrieb vollständig in **Docker-Containern**, verwaltet über **docker-compose**
- ▶ **CI/CD-Pipeline (GitLab CI)** für Validierung & automatisiertes Deployment
- ▶ **Funktionen:** Reiseziele erfassen/anzeigen/löschen, filtern nach Kategorien/Reisearten, Bilder speichern, persistente Daten
- ▶ **Fokus des Projekts:** Frontend–Backend-Kommunikation, Docker-Deployment, GitLab CI/CD, persistente Datenhaltung

Infrastruktur



- Infrastruktur umfasst **CI/CD-Pipeline**, **Docker-Container-Architektur** und **persistente Datenhaltung**.
- GitLab CI/CD wird bei jedem Push auf **main** automatisch gestartet: Build von Frontend & Backend, Validierung der docker-compose.yml, Deployment via `docker compose up -d`.
- Die Anwendung läuft auf einem **Docker Host** mit getrennten Containern:
- **Frontend**: NGINX-Container auf Port 80
- **Backend**: Spring-Boot-Container mit REST-API auf Port 8080
- **H2-Datenbank**: file-basiert, persistiert in einem **Docker Volume**
- Alle Container kommunizieren über ein **internes Docker-Netzwerk**.
- Benutzer greift per Browser auf das Frontend zu; dieses sendet API-Anfragen an das Backend, welches auf die persistente H2-Datenbank zugreift.

Konfiguration

- ▶ Betrieb der App über Docker Compose mit Backend, Frontend und persistenter Datenhaltung.
- ▶ **Docker Compose:** Definiert Services (Backend, Frontend), Volumes (Datenbank, Bilder) und Ports (Backend 8080, Frontend 80).
- ▶ **Backend:** Spring Boot mit H2-Datenbank, konfiguriert für automatische Datenpersistenz und API auf Port 8080.
- ▶ **Frontend:** React-App (Vite), ausgeliefert über NGINX auf Port 80, kommuniziert per HTTP mit Backend.
- ▶ **CI/CD (GitLab):** Automatischer Build und Validierung bei Push auf main, optional manuelles Deployment mit `docker compose up -d`.
- ▶ Alle Konfigurationen liegen im Ordner **infra/** und ermöglichen ein direktes Deployment ohne manuelle Schritte.

Testplan

Nr.	Testfall	Schritte	Erwartetes Ergebnis	Resultat
1	Deployment via Docker Compose	1. Terminal öffnen 2. In Projekt/infra/ navigieren 3. docker compose up –build ausführen	Frontend- und Backend-Container starten ohne Fehler, Volumes werden erzeugt	Gut
2	Frontend erreichbar	1. Browser öffnen 2. http://localhost aufrufen	GlobeNotes-UI wird angezeigt	Gut
3	Backend erreichbar	1. Browser öffnen 2. http://localhost:8080 aufrufen	Spring Boot liefert Whitelabel-Fehlerseite → zeigt Backend läuft	Gut
4	API-Funktion erreichbar	1. Browser öffnen 2. http://localhost:8080/reiseziele/ aufrufen	JSON-Antwort oder leeres Array wird angezeigt	Gut
5	CRUD-Funktion – Löschen	1. Frontend öffnen 2. Ein Reiseziel löschen 3. Seite aktualisieren	Eintrag ist nicht mehr sichtbar, kein Fehler angezeigt	Gut
6	Datenpersistenz (H2 Volume)	1. Ein oder mehrere Einträge erstellen 2. docker compose down + docker compose up -d ausführen	Einträge sind nach Neustart weiterhin sichtbar (Persistenz funktioniert)	Gut
7	H2-Konsole erreichbar	1. Browser öffnen 2. http://localhost:8080/h2-console aufrufen 3. Datenbankverbindung testen	Tabellen sichtbar, Verbindung erfolgreich	Gut
8	Pipeline-Build & Tests	1. Merge Request erstellen 2. CI ausführen lassen	Frontend-Build und Backend-Build laufen erfolgreich	Gut

Installation

- ▶ Voraussetzungen:
 - Docker Desktop installiert
 - GitLab Repo geklont
- ▶ Deployment (Docker):
 - Docker desktop starten
 - cd infra
 - docker compose up --build

Danach erreichbar via:

<http://localhost> → Frontend

<http://localhost:8080> → Backend

<http://localhost:8080/h2-console> → JDBC URL : jdbc:h2:file:./data/reiseziele

The screenshot shows a web browser window with the address bar displaying 'localhost:8080/h2-console/login.jsp'. The browser's address bar also shows a 'VPN' icon. The page has a navigation bar with 'English', 'Preferences', 'Tools', and 'Help'. The main content area is titled 'Login' and contains a form with the following fields and buttons:

- Saved Settings:** A dropdown menu showing 'Generic H2 (Embedded)'.
- Setting Name:** A text input field containing 'Generic H2 (Embedded)', with 'Save' and 'Remove' buttons to its right.
- Driver Class:** A text input field containing 'org.h2.Driver'.
- JDBC URL:** A text input field containing 'jdbc:h2:file:./data/reiseziele'.
- User Name:** A text input field containing 'sa'.
- Password:** A text input field.
- Buttons:** 'Connect' and 'Test Connection' buttons at the bottom.

Dankeschön!

