

324-08A - Build von Container-Image in Pipeline

Inhalt

Einleitung	1
Voraussetzungen.....	1
Frontend-Container mit Multi-Stage-Build.....	1
Nginx-Konfiguration für SPA & Caching.....	3
Backend-Container mit Multi-Stage-Build	3
Lokale Entwicklung mit Docker Compose	4
GitLab CI/CD-Pipeline einrichten.....	5
Pipeline laufen lassen & Ergebnisse prüfen	6
Was wir erreicht haben	7

Einleitung

In diesem Tutorial zeigen wir **Schritt für Schritt**, wie wir unser ToDo-Projekt mit React-Frontend und Spring-Boot-Backend vollständig in Docker-Container gepackt und eine automatische GitLab CI/CD-Pipeline aufgesetzt haben.

Hier ist der Link zu unserem Software-Projekt:

https://gitlab.com/me7554020/m324_projekt_todolist-gemeinsam2

Voraussetzungen

- React-App im Ordner frontend/
- Spring-Boot-App im Ordner backend/
- Installierte Tools: Docker, Docker Compose
- GitLab-Projekt mit aktivierter Container Registry
- GitLab Runner mit Docker-in-Docker (dind)

Frontend-Container mit Multi-Stage-Build

Was wir gemacht haben

1. **Dockerfile** im frontend/ angelegt
2. **Stage 1:** Node.js zum Bauen

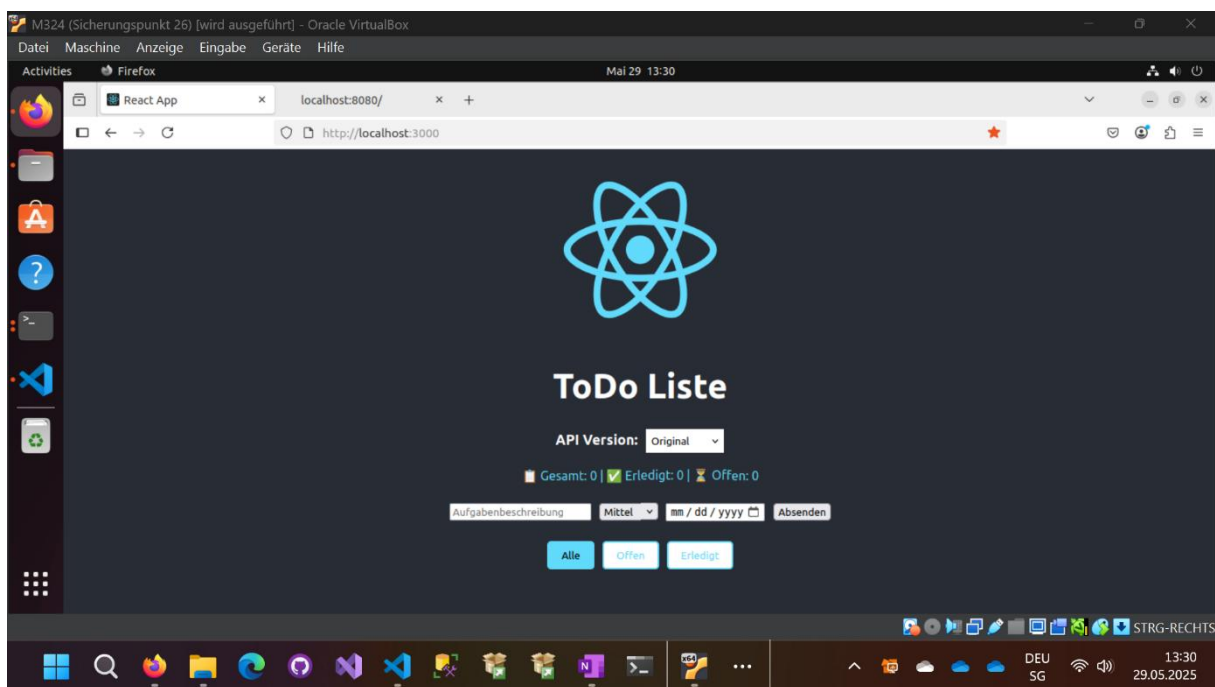
3. **Stage 2:** Nginx zum Ausliefern
4. Multi-Stage-Build reduziert Image-Größe drastisch

https://gitlab.com/me7554020/m324_projekt_todolist-gemeinsam2/-/blob/main/frontend/Dockerfile?ref_type=heads

```
# Stage 1: Build
FROM node:18.12.1 AS build
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN CI=false npm run build

# Stage 2: Production
FROM nginx:alpine
COPY --from=build /app/build /usr/share/nginx/html
COPY nginx.conf /etc/nginx/nginx.conf
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

React-Frontend läuft auf localhost:3000



Nginx-Konfiguration für SPA & Caching

Was wir ergänzt haben (https://gitlab.com/me7554020/m324_projekt_todolist-gemeinsam2/-/blob/main/frontend/nginx.conf?ref_type=heads)

```
server {
    listen 80;
    root /usr/share/nginx/html;
    index index.html;

    # React-Router SPA Support
    location / {
        try_files $uri $uri/ /index.html;
    }

    # Static Assets Caching
    location ~* \.(css|js|png|jpg|woff2?)$ {
        expires 1y;
        add_header Cache-Control "public, immutable";
    }

    gzip on;
}
```

Ergebnis

- SPA-Routing funktioniert
- Assets werden gecached
- Gzip für bessere Performance

Backend-Container mit Multi-Stage-Build

Was wir gemacht haben

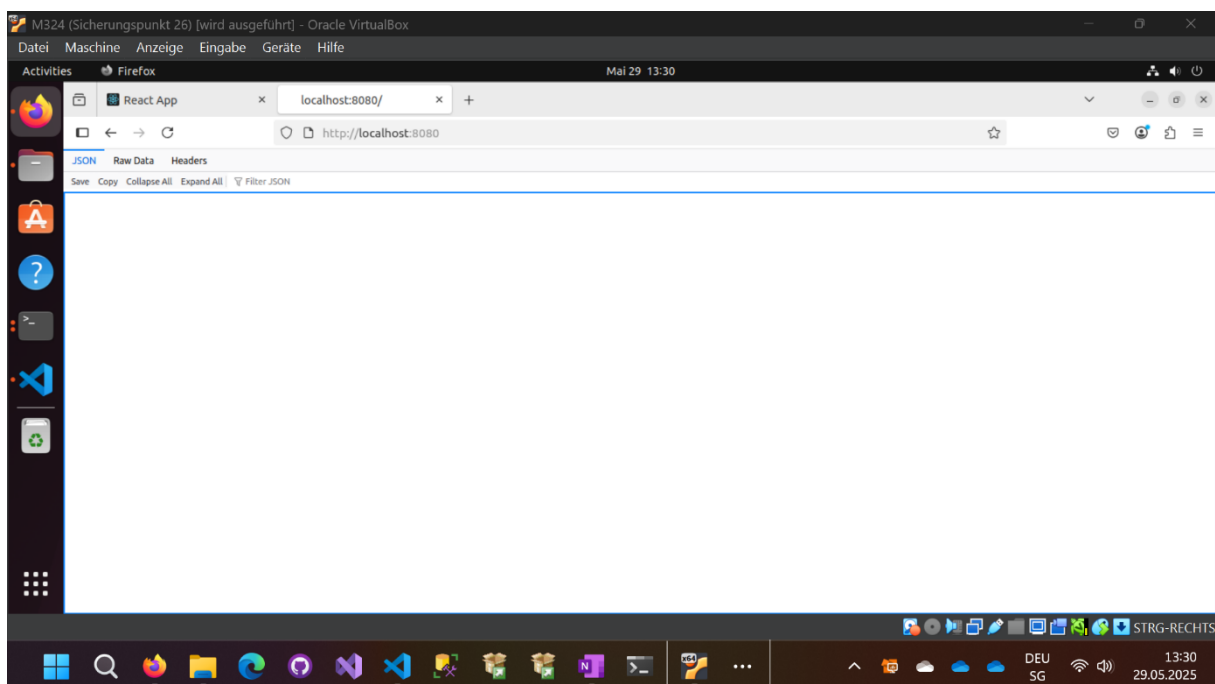
1. **Dockerfile** im backend/ angelegt
2. **Stage 1:** Maven zum Compilieren
3. **Stage 2:** Alpine-JRE zum Ausführen
4. Healthcheck für automatisches Restarting

https://gitlab.com/me7554020/m324_projekt_todolist-gemeinsam2/-/blob/main/backend/Dockerfile?ref_type=heads

```
# Stage 1: Maven Build
FROM maven:3.8.3-openjdk-17 AS build
WORKDIR /app
COPY pom.xml .
```

```
RUN mvn dependency:go-offline
COPY src ./src
RUN mvn clean package -DskipTests
```

```
# Stage 2: Runtime
FROM eclipse-temurin:17-jre-alpine
WORKDIR /app
COPY --from=build /app/target/*.jar app.jar
EXPOSE 8080
HEALTHCHECK --interval=30s --timeout=3s CMD curl -f http://localhost:8080/ || exit 1
ENTRYPOINT ["java","-jar","app.jar"]
```



Lokale Entwicklung mit Docker Compose

Unsere docker-compose.dev.yml

(https://gitlab.com/me7554020/m324_projekt_todolist-gemeinsam2/-/blob/main/docker-compose.dev.yml?ref_type=heads)

```
version: '3.8'
services:
  backend:
    build: ./backend
    container_name: todo-backend-dev
    ports: ["8080:8080"]
    environment:
      - SPRING_PROFILES_ACTIVE=docker
      - SPRING_DATASOURCE_URL=jdbc:h2:mem:tododb
```

```
healthcheck:
  test: ["CMD","curl -f http://localhost:8080/"]
  interval: 30s
  retries: 3

frontend:
  build: ./frontend
  container_name: todo-frontend-dev
  ports: ["3000:80"]
  depends_on:
    backend:
      condition: service_healthy
  environment:
    - REACT_APP_BACKEND_URL=http://localhost:8080

networks:
  default:
    driver: bridge
```

Wir haben es gestartet und es hat funktioniert:

```
kris@kris-VirtualBox:~/M324_PROJEKT_TODOLIST-Gemeinsam2$ docker-compose -f docker-compose.dev.yml up -d
Starting todo-backend-dev ... done
Creating todo-frontend-dev ... done
kris@kris-VirtualBox:~/M324_PROJEKT_TODOLIST-Gemeinsam2$
```

GitLab CI/CD-Pipeline einrichten

Unsere `.gitlab-ci.yml` (https://gitlab.com/me7554020/m324_projekt_todoлист-gemeinsam2/-/blob/main/.gitlab-ci.yml?ref_type=heads)

```
stages:
  - build
  - docker-build
  - docker-push

variables:
  FRONTEND_IMG: $CI_REGISTRY_IMAGE/frontend
  BACKEND_IMG: $CI_REGISTRY_IMAGE/backend

build:api:
  stage: build
  image: maven:3.8.3-openjdk-17
  script:
    - cd backend && mvn clean install -DskipTests
  only: [main, merge_requests]

docker:build:
```

```
stage: docker-build
image: docker:latest
services: [docker:dind]
before_script:
  - echo $CI_JOB_TOKEN | docker login -u $CI_REGISTRY_USER --password-stdin
  $CI_REGISTRY
script:
  - docker build -t $BACKEND_IMG:latest backend
  - docker build -t $FRONTEND_IMG:latest frontend
only: [main]

docker:push:
stage: docker-push
image: docker:latest
services: [docker:dind]
before_script:
  - echo $CI_JOB_TOKEN | docker login -u $CI_REGISTRY_USER --password-stdin
  $CI_REGISTRY
script:
  - docker push $BACKEND_IMG:latest
  - docker push $FRONTEND_IMG:latest
only: [main]
```

Pipeline laufen lassen & Ergebnisse prüfen

- Nach jedem Push auf main läuft unsere Pipeline automatisch.
- Build → Docker-Build → Docker-Push: alles grün!

The screenshot shows the GitLab Pipelines page for the repository 'me / M324_PROJEKT_TODOLIST-Gemeinsam2'. The interface includes a search bar, a filter dropdown, and a table of pipeline runs. The table has columns for Status, Pipeline, Created by, Stages, and Actions. The first pipeline run, 'Fix Dockerfile', is marked as 'Passed' with a green checkmark and a download icon. The second pipeline run, 'Add Docker configuration with Docker...', is marked as 'Failed' with a red X and a refresh/download icon. The third pipeline run, 'Edit DemoApplication.java for API-Vers...', is marked as 'Passed' with a green checkmark and a download icon. The fourth pipeline run, 'remove unnecessary comments', is marked as 'Passed' with a green checkmark and a download icon.

Status	Pipeline	Created by	Stages	Actions
Passed 00:04:20 11 minutes ago	Fix Dockerfile #1842791937 main latest branch		✓ ✓ ✓	Download
Failed 00:02:29 24 minutes ago	Add Docker configuration with Docker... #1842776875 main branch		✓ ✗ >	Refresh Download
Passed 00:01:07 45 minutes ago	Edit DemoApplication.java for API-Vers... #1842746935 14 latest merge request		✓	Download
Passed 00:01:10	remove unnecessary comments #1841645098 13 786b8dd4		✓	Download

Was wir erreicht haben

- **Schlanke Docker-Images** durch Multi-Stage-Builds
- **Robuste Services** dank Healthchecks und Alpine-Basis
- **Ein-Kommando-Setup** für lokale Entwicklung
- **Vollautomatische CI/CD** von Push bis Registry
- **Blueprint für künftige Microservices**

Mit diesem Setup haben wir unsere Deployment-Geschwindigkeit um ein Vielfaches gesteigert und eine solide Grundlage für alle kommenden Projekte gelegt.