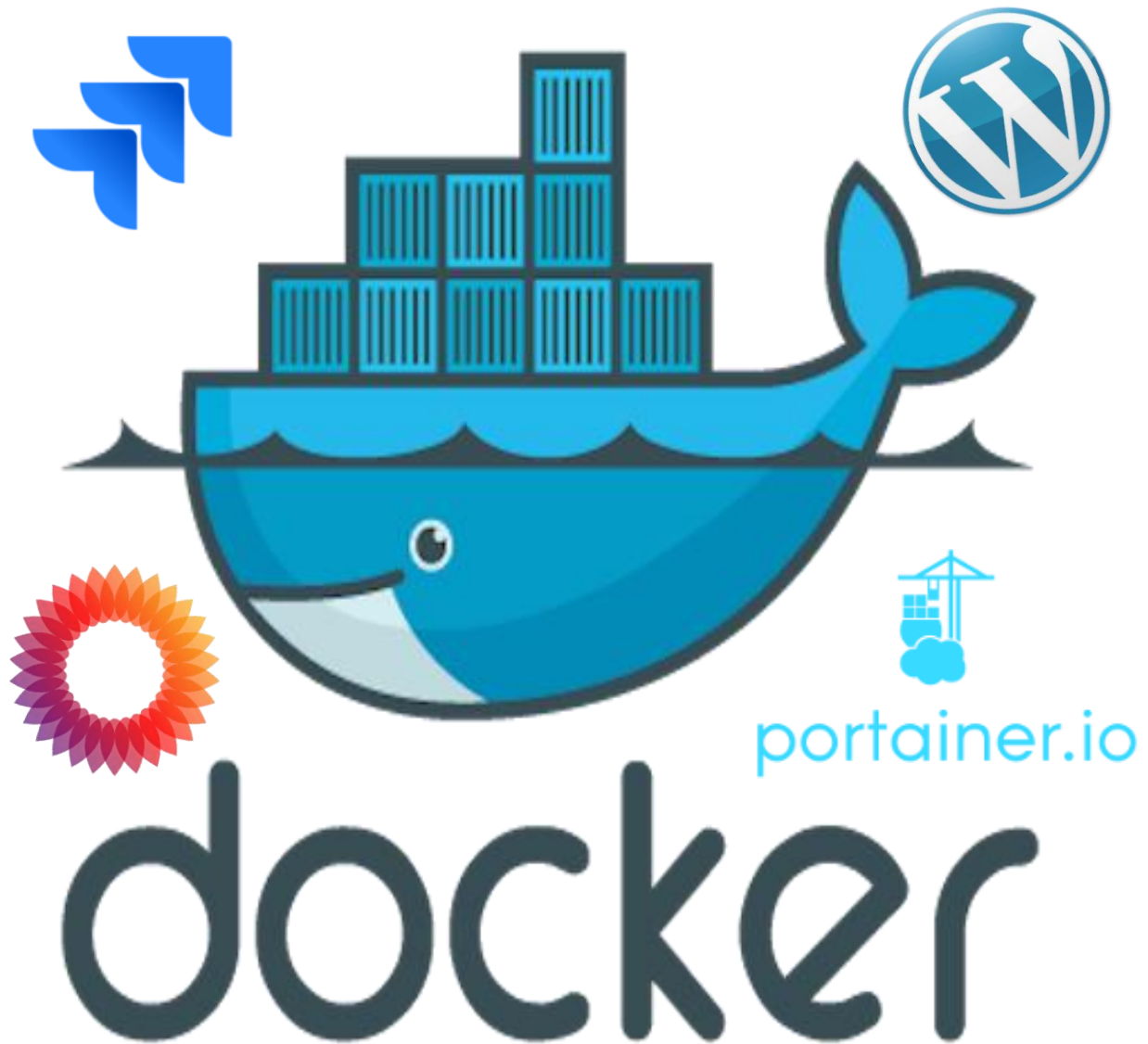


Projektdokumentation

Modul 347



Projekt: Containerisierte Application Stacks mit Docker

Team: Silas & Remo

Datum: 02.07.2025

Inhalt

Einführung.....	3
Zielsetzung.....	3
Infrastruktur	3
Architekturübersicht.....	3
Projektstruktur	3
Infrastrukturdiagramm.....	4
Deployment	5
Systemvorbereitung.....	5
Einrichtungsschritte	5
Zugriff auf die Services	5
Torubleshoot.....	5
Konfiguration.....	6
WordPress	6
MediaWiki	7
Jira	9
Portainer (Monitoring)	9
Löschen der Container.....	12
Testing	13
WordPress	13
Testkonzept.....	13
Testprotokoll	13
MediaWiki	13
Testkonzept.....	13
Testprotokoll	13
Jira	14
Testkonzept.....	14
Testprotokoll	14
Portainer	14
Testkonzept.....	14
Testrpotokoll	14
Testergebnisse	15
Hilfestellungen	15
Persönliche Arbeitsjournale und Fazits	16
Arbeitsjournal.....	16
Projekt Docker	2

Fazit..... 17

Einführung

Dieses Projekt verfolgt das Ziel, mehrere Applikationen (WordPress, MediaWiki, Jira) sowie eine Monitoring-Lösung (Portainer) in einer containerisierten Umgebung bereitzustellen. Die Containerisierung erfolgt mittels Docker, um eine einfache, portable und wiederholbare Bereitstellung zu ermöglichen. Die Applikationen nutzen jeweils eigene Datenbanken mit persistenter Speicherung, wodurch Daten auch bei Neustarts erhalten bleiben.

Zielsetzung

- Vollständig lauffähige Docker-Container für WordPress, MediaWiki, Jira, Portainer
- Persistente Speicherung der Anwendungsdaten über Docker-Volumes
- Automatisierte Startreihenfolge und Netzwerk-Kommunikation über Docker-Netzwerke
- Dokumentation inkl. Infrastruktur, Konfiguration, Testplan und Installationsanleitung
- Präsentation der Ergebnisse in 10 Minuten

Infrastruktur

Architekturübersicht

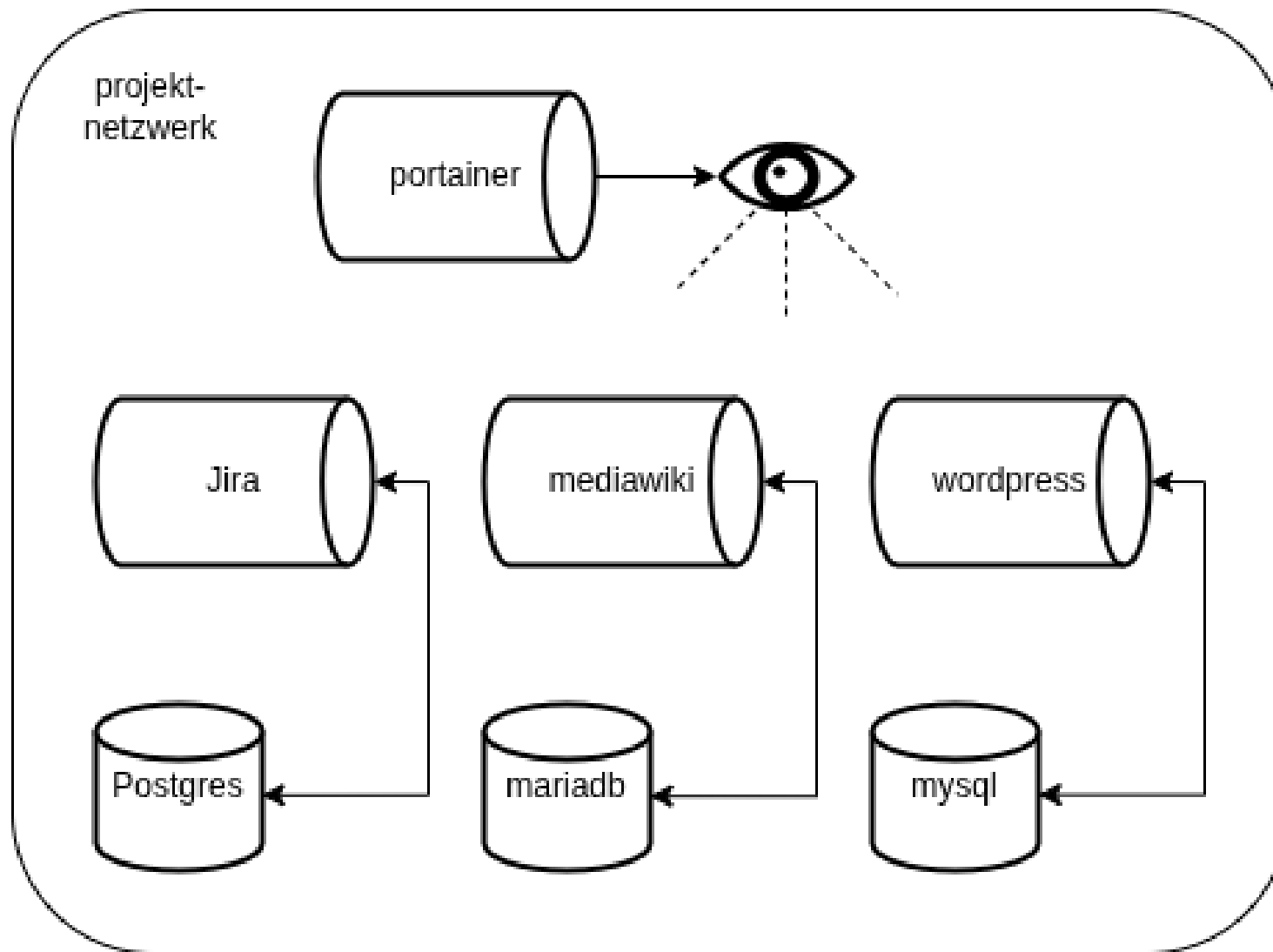
Die Architektur basiert auf mehreren Docker-Containern, die in Netzwerken organisiert sind. Es gibt ein Hauptnetzwerk: (gleiche Farben kommunizieren direkt miteinander)

projekt-netzwerk
WordPress Container
MySQL Datenbank für WordPress
Jira Container
PostgreSQL Datenbank für Jira
Portainer Container
MediaWiki Container
MariaDB Datenbank für MediaWiki

Projektstruktur

```
-- modul347
|-- README.md
|-- dokumentation.txt
|-- praesentation.txt
|-- projekt
|   |-- INSTALLATION.md
|   |-- jira
|   |   |-- docker-compose.yml
|   |-- mediawiki
|   |   |-- docker-compose.yml
|   |-- monitoring_portainer
|   |   |-- docker-compose.yml
|   |-- password.txt
|   |-- run_all.sh
|   |-- websites.txt
|   |-- wordpress
|   |   |-- docker-compose.yml
|
7 directories, 11 files
```

Infrastrukturdiagramm



Deployment

Sie finden im Github unter dem Link: <https://github.com/wettsteinremodev/modul347> in welchem sich eine Installationsanleitung befindet.

die Datei **INSTALLATION.md** befindet sich im «projekt» Ordner des Github Repositories. Diese Datei beinhaltet den detaillierteren Ablauf zur Einrichtung der Container.

Systemvorbereitung

Folgende Voraussetzungen müssen gegeben sein:

- **Ubuntu 24.04** LTS wird verwendet. Und ist up to date!
- **Docker / Docker Compose** installiert
- **Git** installiert



Einrichtungsschritte

1. Öffne das Terminal im Ordner "projekt".
2. Im Terminal: **bash run_all.sh** (automatische Installation aller Container)
3. Im Terminal: **docker ps -a** (überprüfen, ob das script funktioniert hat)

Zugriff auf die Services

Service	URL
MediaWiki	http://localhost:8081
WordPress	http://localhost:8080
Jira	http://localhost:8082
Portainer	http://localhost:9000

Troubleshoot

Falls sie Probleme mit den Containern haben können sie diese leicht **mit docker restart «CONTAINERNAME»** neustarten.

Konfiguration

WordPress

Image: wordpress:latest

Ports: Host 8080 → Container 80

Datenbank: MySQL 8.x (mysql:latest)

Datenbank-Konfiguration:
Datenbankname: wp_db
Benutzer: remo
Passwort: 123



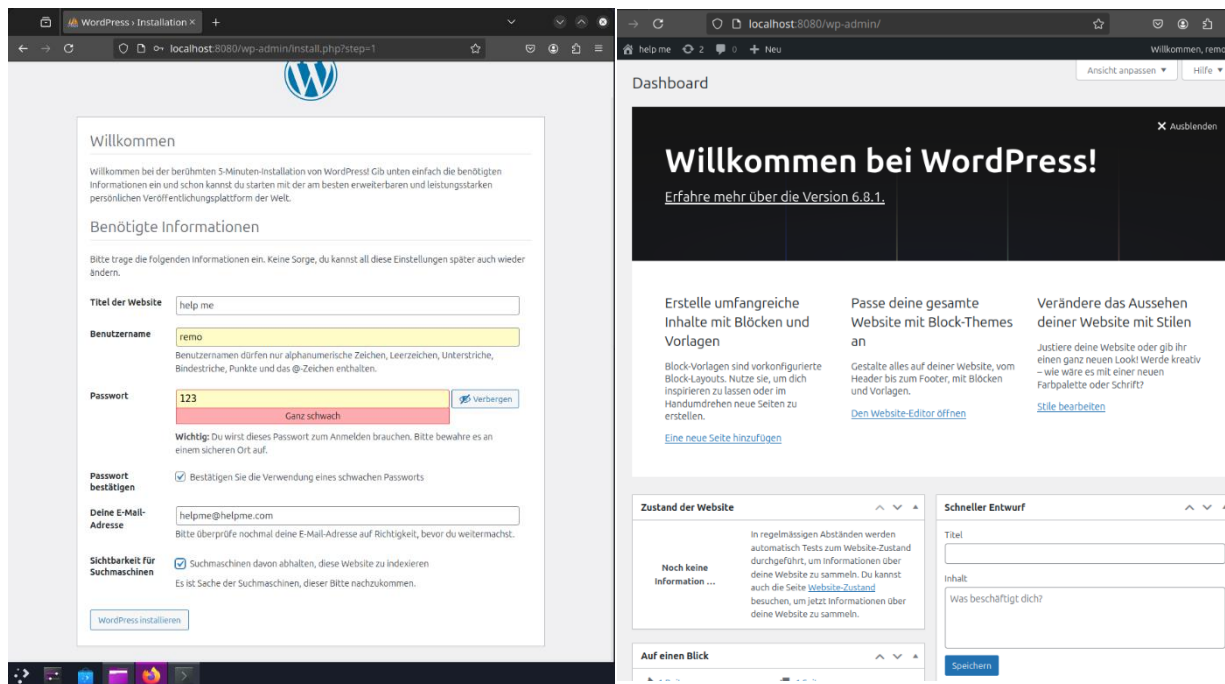
Persistenz: Docker Volume db_data sichert MySQL-Daten auf dem Host.

Netzwerk: projekt-netzwerk

Besonderheiten:

depends_on sorgt für Startreihenfolge (MySQL vor WordPress).

WordPress Umgebungsvariablen für DB-Zugang werden gesetzt.



MediaWiki

Image: mediawiki:1.39

Ports: Host 8081 → Container 80

Datenbank: MariaDB 10.5 (mariadb:10.5)

Datenbank-Konfiguration:
Datenbankserver: db
Datenbankname: wiki
Benutzer: remo
Passwort: 1234asdf



Persistenz:

Datenbank-Volume: db_data (empfehlenswert: separates Volume)

Uploads/Image-Verzeichnis: mediawiki_data

Netzwerk: wikinet

Besonderheiten:

depends_on startet MariaDB vor MediaWiki.

Medienuploads bleiben durch Volume erhalten.

Wikki-Login-Data

Wikki-user: **remo**

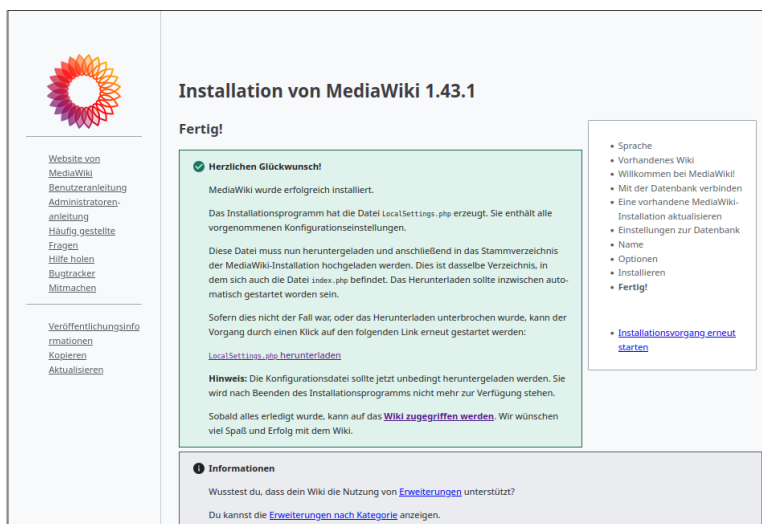
Wikki-user-pw: **123456asdfgh**

Wikki PHP setup

Adding the LocalSettings.php to the docker container

1. docker cp LocalSettings.php mediawiki:/var/www/html/LocalSettings.php
2. docker restart mediawiki

```
test@347UBU:~/ubu347/LB-Projekt-347_Remo_Silas/Github/modul347/projekt/mediawiki$ docker cp LocalSettings.php mediawiki:/var/www/html/LocalSettings.php
Successfully copied 6.14kB to mediawiki:/var/www/html/LocalSettings.php
test@347UBU:~/ubu347/LB-Projekt-347_Remo_Silas/Github/modul347/projekt/mediawiki$ docker restart mediawiki
mediawiki
test@347UBU:~/ubu347/LB-Projekt-347_Remo_Silas/Github/modul347/projekt/mediawiki$
```



Installation von MediaWiki 1.43.1

Fertig!

✓ **Herzlichen Glückwunsch!**

MediaWiki wurde erfolgreich installiert.

Das Installationsprogramm hat die Datei LocalSettings.php erzeugt. Sie enthält alle vorgenommenen Konfigurationseinstellungen.

Diese Datei muss nun heruntergeladen und anschließend in das Stammverzeichnis der MediaWiki-Installation hochgeladen werden. Dies ist dasselbe Verzeichnis, in dem sich auch die Datei index.php befindet. Das Herunterladen sollte inzwischen automatisch gestartet worden sein.

Sofern dies nicht der Fall war, oder das Herunterladen unterbrochen wurde, kann der Vorgang durch einen Klick auf den folgenden Link erneut gestartet werden:

[LocalSettings.php herunterladen](#)

Hinweis: Die Konfigurationsdatei sollte jetzt unbedingt heruntergeladen werden. Sie wird nach Beenden des Installationsprogramms nicht mehr zur Verfügung stehen.

Sobald alles erledigt wurde, kann auf das [Wiki zugegriffen werden](#). Wir wünschen viel Spaß und Erfolg mit dem Wiki.

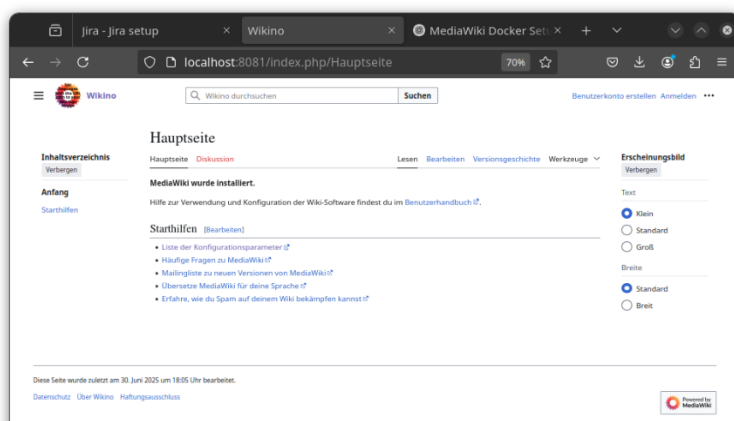
- Sprache
- Vorhandenes Wiki
- Willkommen bei MediaWiki!
- Mit der Datenbank verbinden
- Eine vorhandene MediaWiki-Installation aktualisieren
- Einstellungen zur Datenbank
- Name
- Optionen
- Installieren
- Fertig!

• [Installationsvorgang erneut starten](#)

Informationen

Wusstest du, dass dein Wiki die Nutzung von [Erweiterungen](#) unterstützt?

Du kannst die [Erweiterungen nach Kategorie](#) anzeigen.



Wikino

Hauptseite

MediaWiki wurde installiert.

Hilfe zur Verwendung und Konfiguration der Wiki-Software findest du im [Benutzerhandbuch](#).

Starthilfen (Bearbeiten)

- Liste der Konfigurationsparameter
- Häufige Fragen zu MediaWiki
- Mailingliste zu neuen Versionen von MediaWiki
- Übernimm MediaWiki für deine Sprache
- Erfahre, wie du Spam auf deinem Wiki bekämpfen kannst

Diese Seite wurde zuletzt am 30. Juni 2025 um 18:05 Uhr bearbeitet.

[Datenschutz](#) [Über Wikino](#) [Haftungsausschluss](#)

Jira

Image: atlassian/jira-software

Ports: Host 8082 → Container 8080

Datenbank: PostgreSQL (postgres:latest)



Datenbank-Konfiguration:
Hostname: jira-db-1
Datenbankname: jira_db
Benutzer: jira_user
Passwort: jira_pass

✓ The database connection test was successful.

Database setup [Language](#)

[Learn more about connecting Jira to a database.](#)

Database Type: PostgreSQL ▼

Hostname: jira-db-1
Hostname or IP address of the database server.

Port: 5432
TCP Port Number for the database server.

Database: jira_db
The name of the database to connect to.

Username: jira_user
The username used to access the database.

Password:
The password used to access the database.

Schema: public
Specify the schema name for your database.

[Next](#) [Test Connection](#)

Persistenz: Docker Volume db_data (empfehlenswert eigenes Volume für PostgreSQL)

Netzwerk: projekt-netzwerk

Besonderheiten:

depends_on sorgt für Datenbank-Start vor Jira.

Verbindungs-URL als JDBC String konfiguriert.

Portainer (Monitoring)

Image: portainer/portainer-ce

Ports: 9000:9000 (UI), 8000:8000 (Agent)

Volumes:

Docker-Socket Bind-Mount /var/run/docker.sock für direkte Docker-Kommunikation

portainer_data Volume für Persistenz der Einstellungen

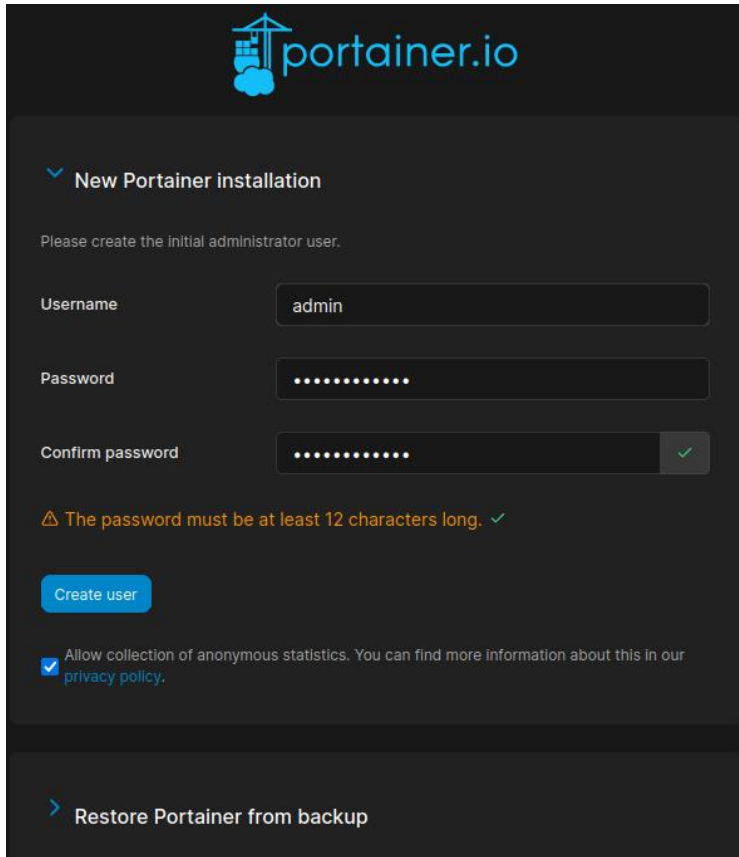
Projekt Docker



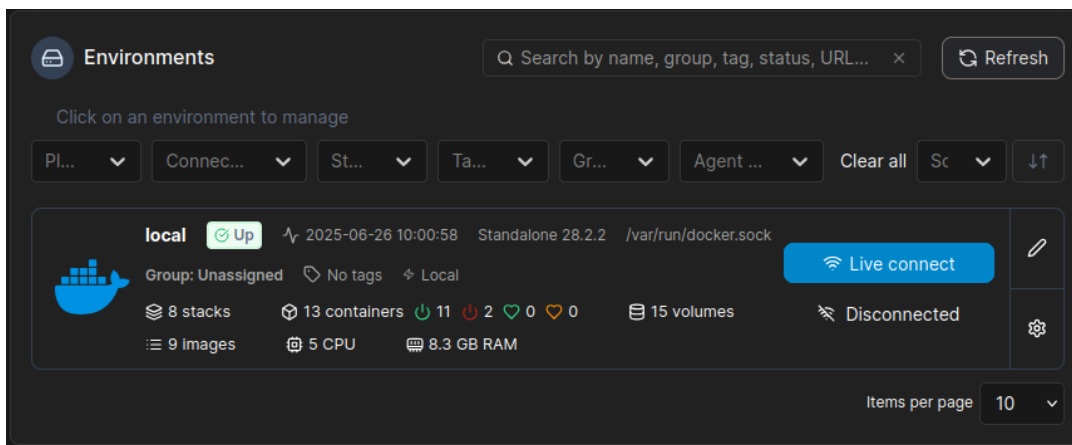
Netzwerk: projekt-netzwerk

Besonderheiten:

Web-UI zur einfachen Verwaltung und Monitoring aller Docker-Container.



The image shows the 'New Portainer installation' screen. At the top is the Portainer.io logo. Below it, a section titled 'New Portainer installation' with a checkmark icon. A message says 'Please create the initial administrator user.' There are three input fields: 'Username' with 'admin', 'Password' with masked dots, and 'Confirm password' with masked dots and a green checkmark icon. A warning message states 'The password must be at least 12 characters long.' with a green checkmark. A blue 'Create user' button is below. At the bottom, there is a checkbox for 'Allow collection of anonymous statistics' which is checked, with a link to the 'privacy policy'. Below this is a link to 'Restore Portainer from backup'.



The image shows the 'Environments' screen in Portainer. At the top is a search bar 'Search by name, group, tag, status, URL...' and a 'Refresh' button. Below is a message 'Click on an environment to manage'. There are several filter buttons: 'Pl...', 'Connec...', 'St...', 'Ta...', 'Gr...', 'Agent ...', 'Clear all', 'Sc', and a sort button '↓↑'. The main content area shows a single environment named 'local' with a green 'Up' status. It includes a ship icon, a timestamp '2025-06-26 10:00:58', version 'Standalone 28.2.2', and socket path '/var/run/docker.sock'. Below this, it shows 'Group: Unassigned', 'No tags', and 'Local'. A 'Live connect' button is present. At the bottom, it displays statistics: '8 stacks', '13 containers' (11 green, 2 red, 0 yellow, 0 orange), '15 volumes', '9 Images', '5 CPU', and '8.3 GB RAM'. A 'Disconnected' status is shown with a plug icon. On the right, there are edit and settings icons. At the bottom right, there is a pagination control 'Items per page 10'.

Environment summary

Dashboard

admin

Environment info

Environment	local 5 8.3 GB - Standalone 28.2.2
URL	/var/run/docker.sock
GPU	none
Tags	-

8
Stacks

13
Containers
11 running 2 stopped
0 healthy 0 unhealthy

9
Images
6 GB

15
Volumes

12
Networks

Containers

Container list

admin

Containers

Search...

Start Stop Kill Restart Pause Resume Remove Add

Name	State	Quick Actions	Stack	Image	Created
jira-db-1	running		jira	postgres:latest	2025-06-10 10:10:10
jira-jira-1	running		jira	atlassian/jira-software	2025-06-10 10:10:10
jira_db	running		jira	postgres:latest	2025-06-10 10:10:10
mediawiki	running		mediawiki	mediawiki:1.39	2025-06-10 10:10:10
mongo	running		mongo	mongo	2025-06-10 10:10:10
mongo-express	exited - code 255		mongo-express	mongo-express	2025-06-10 10:10:10
mongo-shell	exited - code 255		mongo-shell	mongo	2025-06-10 10:10:10
mongodb-mongo-1	running		mongodb	mongo	2025-06-10 10:10:10
mongodb-mongo-express-1	running		mongodb	mongo-express	2025-06-10 10:10:10
portainer	running		monitoring_portainer	portainer/portainer-ce	2025-06-10 10:10:10

Items per page 10 1 2

Löschen der Container

Um alle Container zu löschen, um Sie zB. erneut zu installieren kann der folgende Befehl lafengelassen werden:

```
docker kill $(docker ps -q) && docker rm $(docker ps -a -q) && docker volume rm $(docker volume ls -q) && docker rmi $(docker images -q)
```

COPY: docker kill \$(docker ps -q) && docker rm \$(docker ps -a -q) && docker volume rm \$(docker volume ls -q) && docker rmi \$(docker images -q)

Hinweis: löscht ALLE Docker Container, welche in Docker vorhanden sind!

Testing

WordPress

Testkonzept



Testziel	Sicherstellen, dass Wordpress mit Datenbank funktioniert und Webseite erreichbar ist
Testumgebung	Docker-Container auf lokalem Host, Port 8080
Testfälle	<ul style="list-style-type: none"> - Startet Container korrekt? - Verbindung zur MySQL? - Webseite aufrufbar?
Erwartetes Ergebnis	Wordpress-Webseite unter localhost:8080 erreichbar, Inhalte persistent

Testprotokoll

Testfall	Ergebnis	Bemerkung
docker-compose up	Erfolgreich	Keine Fehler
Aufruf: localhost:8080	Erfolgreich	Webseite lädt, Startseite sichtbar
Beitrag erstellen + neu laden	Erfolgreich	Inhalt bleibt erhalten

MediaWiki

Testkonzept



Testziel	MediaWiki-Instanz funktionsfähig und persistente Speicherung möglich
Testumgebung	Docker, Port 8081
Testfälle	<ul style="list-style-type: none"> - Webseite erreichbar? - Verbindung zu DB? - Artikel speichern funktioniert?
Erwartetes Ergebnis	MediaWiki-Webseite lädt, Artikel können erstellt und gespeichert werden

Testprotokoll

Testfall	Ergebnis	Bemerkung
docker-compose up	Erfolgreich	Beide Container starten ohne Fehler
Aufruf über localhost:8081	Erfolgreich	Setup-Seite und Hauptseite sichtbar
Artikel erstellen & danach aufrufen	Erfolgreich	Artikel bleibt bestehen nach Neustart

Jira

Testkonzept

Testziel	Jira-Webinterface aufrufbar, Verbindung zur PostgreSQL
Testumgebung	Docker, Port 8082
Testfälle	<ul style="list-style-type: none"> - Lädt Web-UI? - Verbindung zur DB? - Projekt anlegen möglich?
Erwartetes Ergebnis	Jira GUI erreichbar, Login funktioniert, Projekte lassen sich verwalten



Testprotokoll

Testfall	Ergebnis	Bemerkung
Start des Containers	Erfolgreich	DB wurde automatisch verbunden
Öffnen: localhost:8082	Erfolgreich	Jira Setup-Seite erscheint
Neues Projekt erstellen	Nicht Erfolgreich	Keinen Zugangscode!

Portainer

Testkonzept

Testziel	Portainer-WebUI erreichbar, Containerübersicht korrekt dargestellt
Testumgebung	Docker, Ports 9000 (UI) und 8000 (agent)
Testfälle	<ul style="list-style-type: none"> - Startet korrekt? - Zugriff über Web? - Container sichtbar?
Erwartetes Ergebnis	Zugriff auf localhost:9000, Login möglich, alle Container sichtbar



Testprotokoll

Testfall	Ergebnis	Bemerkung
Start des Containers	Erfolgreich	Keine Fehler beim Boot
Zugriff über localhost:9000	Erfolgreich	Web-Oberfläche lädt
Ansicht der laufenden Container	Erfolgreich	Alle Microservices sind sichtbar

Testergebnisse

Alle Services starten und laufen stabil.	
Daten bleiben bei Neustarts erhalten.	
Webinterfaces reagieren erwartungsgemäß.	
Monitoring mit Portainer funktioniert einwandfrei.	
Keinen Aktivierungscode für Jira vorhanden	

Hilfestellungen

Unterstützung durch Dozenten bei Docker-Netzwerkfragen

Offizielle Docker-Dokumentation für Volumes und Compose

Tutorials und Community-Foren (Stack Overflow, Atlassian Docs)

Zusammenarbeit und Wissensaustausch im Team

Persönliche Arbeitsjournale und Fazits

Arbeitsjournal

Datum	Wer	Tätigkeit	Bemerkung
22.05.2025	Beide	Projektstart, Aufgabenverteilung und Grobplanung	Jira (Silas) & Wordpress (Remo) zugewiesen
29.05.2025	Remo	Docker-Setup für Wordpress und MySQL erstellt	Netzwerk- und Volume-Konfiguration getestet
29.05.2025	Silas	Jira + PostgreSQL in docker-compose eingebunden	Verbindungsprobleme gelöst
05.06.2025	Remo	Fehlerbehebung bei Wordpress DB-Verbindung	depends_on und Umgebungsvariablen angepasst
05.06.2025	Silas	Testkonzept für Jira geschrieben	Jira-Weboberfläche auf Erreichbarkeit geprüft
12.05.2025	Remo	Portainer installiert und mit Docker Socket verbunden	UI getestet, Übersicht über Container
12.06.2025	Silas	MediaWiki-Setup begonnen	Unterstützung durch Remo bei DB-Host-Problemen
19.06.2025	Remo	Testfälle für Wordpress dokumentiert und durchgeführt	Alles funktional, inkl. DB-Persistenz
19.06.2025	Silas	Dokumentation ergänzt, Jira-Details eingetragen	Gemeinsame Überarbeitung mit Remo
19.06.2025	Beide	MediaWiki-Tests durchgeführt	Uploads, Bilder, Persistenz geprüft
26.06.2025	Silas	Tests für MediaWiki abgeschlossen	Erfolgreich, keine Fehler
26.06.2025	Beide	Präsentation vorbereitet + Infrastrukturdiagramm finalisiert	Zeit geübt, Struktur abgestimmt
26.06.2025	Beide	Gesamtsystem getestet, Projektpaket erstellt	Finales ZIP vorbereitet, alles läuft stabil
30.06.2025	Beide	Präsentation vorbereitet	Ready to go

Fazit

Silas:

Die Konfiguration von komplexeren Anwendungen wie Jira in einem Container-Umfeld war anfangs anspruchsvoll. Vor allem das Zusammenspiel mit einer externen Datenbank erforderte viel Sorgfalt. Durch strukturierte Teamarbeit und stetiges Testen konnten wir die Hürden aber gut überwinden. Die Projektarbeit hat mir geholfen, technische Zusammenhänge besser zu verstehen und gleichzeitig die Bedeutung einer sauberen Dokumentation zu schätzen.

Remo:

Die Arbeit mit Docker-Containern hat mein Verständnis für Microservice-Architekturen deutlich verbessert. Besonders spannend fand ich den Umgang mit persistenten Volumes und die Herausforderung, mehrere Services in einem gemeinsamen Netzwerk zu betreiben. Auch die automatisierte Bereitstellung mithilfe von docker-compose war eine sehr lehrreiche Erfahrung. Ich konnte mein Wissen im Bereich Linux, Netzwerke und Service-Orchestrierung stark vertiefen.