
Modul 169

Modul	IET-169 – Services mit Containern bereitstellen
Klasse	INF2023c
Eingereicht von	Tharaneeka Ravitharan
Version	0.0.6
Eingereicht bei	Reto Brütsch
Datum	28.02.2025

Inhaltsverzeichnis

1	Versionierung	2
1.1	Versionskontrolle.....	2
2	Tag 1, KW 5	3
2.1	Container und ihr Nutzen	3
2.2	DevOps.....	3
2.3	Virtualisierung vs. Containerisierung.....	3
2.4	Unterschied Image und Container.....	3
2.5	OnlyOffice	4
3	Tag 2, KW 6	5
3.1	Todo App Version 1	5
3.2	Image pushen auf GitLab	5
3.2.1	Befehle für das Pushen.....	6
3.3	Todo App Version 2	7
4	Tag 3, KW 7	8
4.1	Docker Compose.....	8
4.2	Todo App Version 2 mit Docker Compose.....	8
4.3	Portainer	9
4.4	App via Portainer installiert.....	10
4.5	Shop mit Docker Compose	12
5	Übungsprojekt Webserver	14
5.1	Anleitung Webserver	15
5.1.1	Schritt 1: GitLab Repository erstellen	15
5.1.2	Schritt 2: Projekt lokal einrichten.....	15
5.1.3	Schritt 3: Docker-Setup einrichten	15
5.1.4	Schritt 4: HTML-Webseite erstellen	15
5.1.5	Schritt 5: Nginx-Konfiguration erstellen.....	15
5.1.6	Schritt 6: Dockerfile erstellen.....	16
5.1.7	Schritt 7: Docker Compose Datei erstellen	16
5.1.8	Schritt 8: Docker-Container starten	16
5.1.9	Schritt 9: Projekt in GitLab hochladen.....	17
5.1.10	Schritt 10: Änderungen an der Webseite veröffentlichen	17
5.2	Anleitung für Lehrer.....	18
6	Befehle und ihre Funktionen.....	21

1 Versionierung

Diese Dokumentation unterliegt der Versionierung bzw. Versionskontrolle. Bei jeder grösseren Änderung wird die Version angepasst. Ganzzahlige Werte entsprechen Haupt- bzw. Ausgabeversionen. Kleine Änderungen (Rechtschreibkorrekturen, kleinere Layoutanpassungen und dergleichen) werden nicht in der Versionskontrolle festgehalten.

1.1 Versionskontrolle

Versionskontrolle

Version	Datum	Verantwortlich	Bemerkung
0.0.1	31.01.2025	Tharaneeka Ravitharan	- Tag 1, KW 5
0.0.2	07.02.2025	Tharaneeka Ravitharan	- Tag 2, KW 6
0.0.3	14.02.2025	Tharaneeka Ravitharan	- Tag 3, KW 7
0.0.4	21.02.2025	Tharaneeka Ravitharan	- Tag 3, KW 7
0.0.5	23.02.2025	Tharaneeka Ravitharan	- Übungsprojekt Webserver
0.0.6	28.02.2025	Tharaneeka Ravitharan	- Image auf GitLab pushen

2 Tag 1, KW 5

2.1 Container und ihr Nutzen

Container sind leichtgewichtige, eigenständige Softwareeinheiten, die Anwendungen und deren Abhängigkeiten enthalten. Sie stellen sicher, dass eine Anwendung in jeder Umgebung gleichläuft, da sie mit allen benötigten Bibliotheken, Abhängigkeiten und Konfigurationen gebündelt sind. Im Gegensatz zu virtuellen Maschinen teilen sich Container den Kernel des Host-Systems, wodurch sie effizienter und ressourcenschonender sind. Sie bieten eine hohe Portabilität, Skalierbarkeit und ermöglichen eine schnelle Bereitstellung von Software. Besonders in der Cloud, in Microservices-Architekturen und im Continuous Deployment sind Container von großem Nutzen, da sie Entwicklung, Test und Betrieb nahtlos verbinden.

2.2 DevOps

DevOps ist eine Methodologie, die die Entwicklung (Development) und den IT-Betrieb zusammenführt, um Software schneller, effizienter und zuverlässiger bereitzustellen. Ziel ist es, Silos zwischen Teams aufzulösen und durch Automatisierung, kontinuierliche Integration und kontinuierliche Bereitstellung effizientere Prozesse zu schaffen. DevOps setzt auf enge Zusammenarbeit, agile Methoden und Tools wie Container, CI/CD-Pipelines und Monitoring-Systeme. Dadurch können Unternehmen schneller auf Änderungen reagieren, die Qualität ihrer Software verbessern und Ausfallzeiten minimieren. DevOps wird besonders in modernen Softwareentwicklungsprozessen eingesetzt, um Innovation und Wettbewerbsfähigkeit zu steigern.

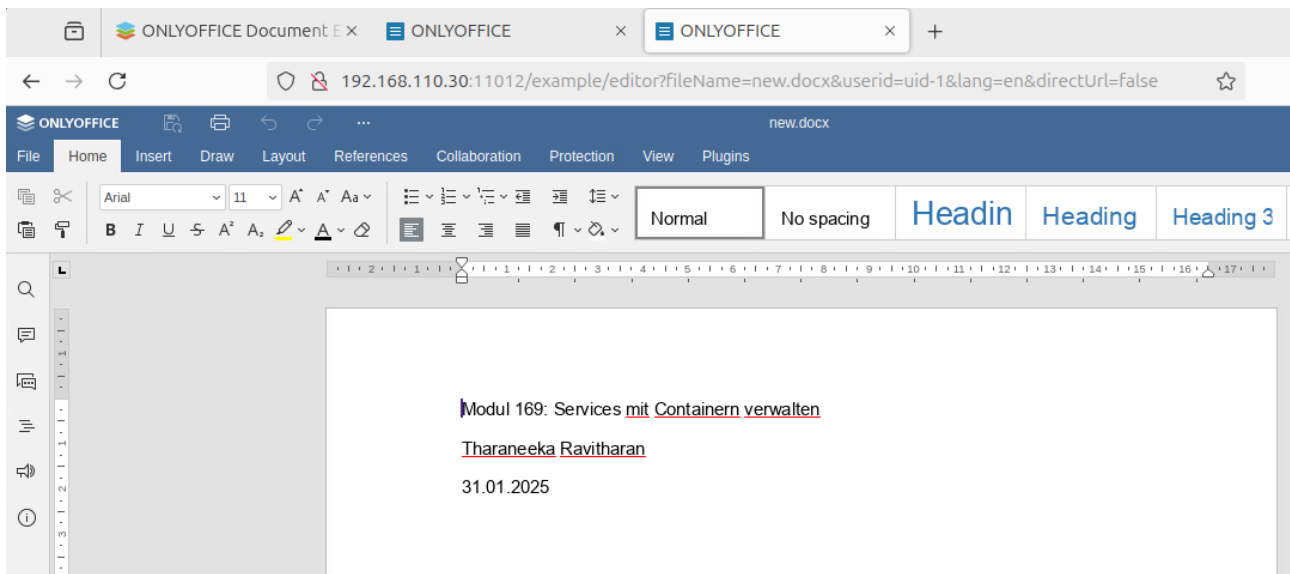
2.3 Virtualisierung vs. Containerisierung

Virtualisierung und Containerisierung sind beide Technologien zur Bereitstellung isolierter Softwareumgebungen, unterscheiden sich jedoch grundlegend. Bei der Virtualisierung wird eine komplette virtuelle Maschine mit eigenem Betriebssystem auf einem Hypervisor ausgeführt, wodurch eine hohe Isolation, aber auch ein hoher Ressourcenverbrauch entsteht. Container hingegen teilen sich den Kernel des Host-Systems und sind dadurch deutlich schlanker und schneller. Während Virtualisierung oft in klassischen Rechenzentren zur Bereitstellung mehrerer Betriebssysteme auf einer Hardware genutzt wird, kommt Containerisierung in modernen Cloud- und Microservices-Architekturen zum Einsatz. Container sind ideal für agile Entwicklungsprozesse, da sie schnelle Bereitstellung, Skalierbarkeit und Portabilität ermöglichen.

2.4 Unterschied Image und Container

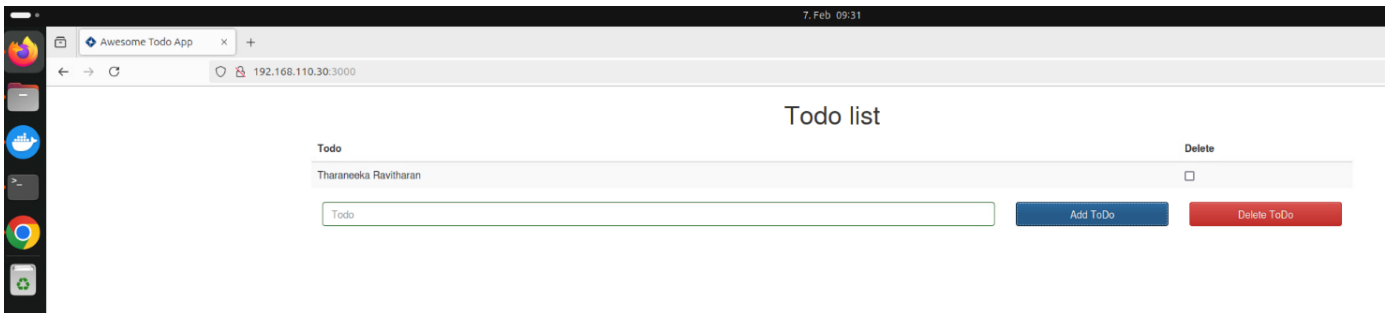
Ein Image ist eine unveränderliche Vorlage, die alle notwendigen Dateien, Abhängigkeiten und Konfigurationen enthält, um eine Anwendung in einer containerisierten Umgebung auszuführen. Es dient als Bauplan für Container. Ein Container hingegen ist eine laufende Instanz eines Images, die zur Ausführung einer Anwendung verwendet wird. Während Images statisch sind und nicht verändert werden, sind Container dynamisch und können zur Laufzeit modifiziert werden. Ein Container basiert immer auf einem Image, kann aber individuelle Laufzeitdaten und Konfigurationsänderungen enthalten. Dadurch ermöglicht die Nutzung von Images eine konsistente Bereitstellung, während Container die Flexibilität bieten, Anwendungen in unterschiedlichen Umgebungen auszuführen.

2.5 OnlyOffice

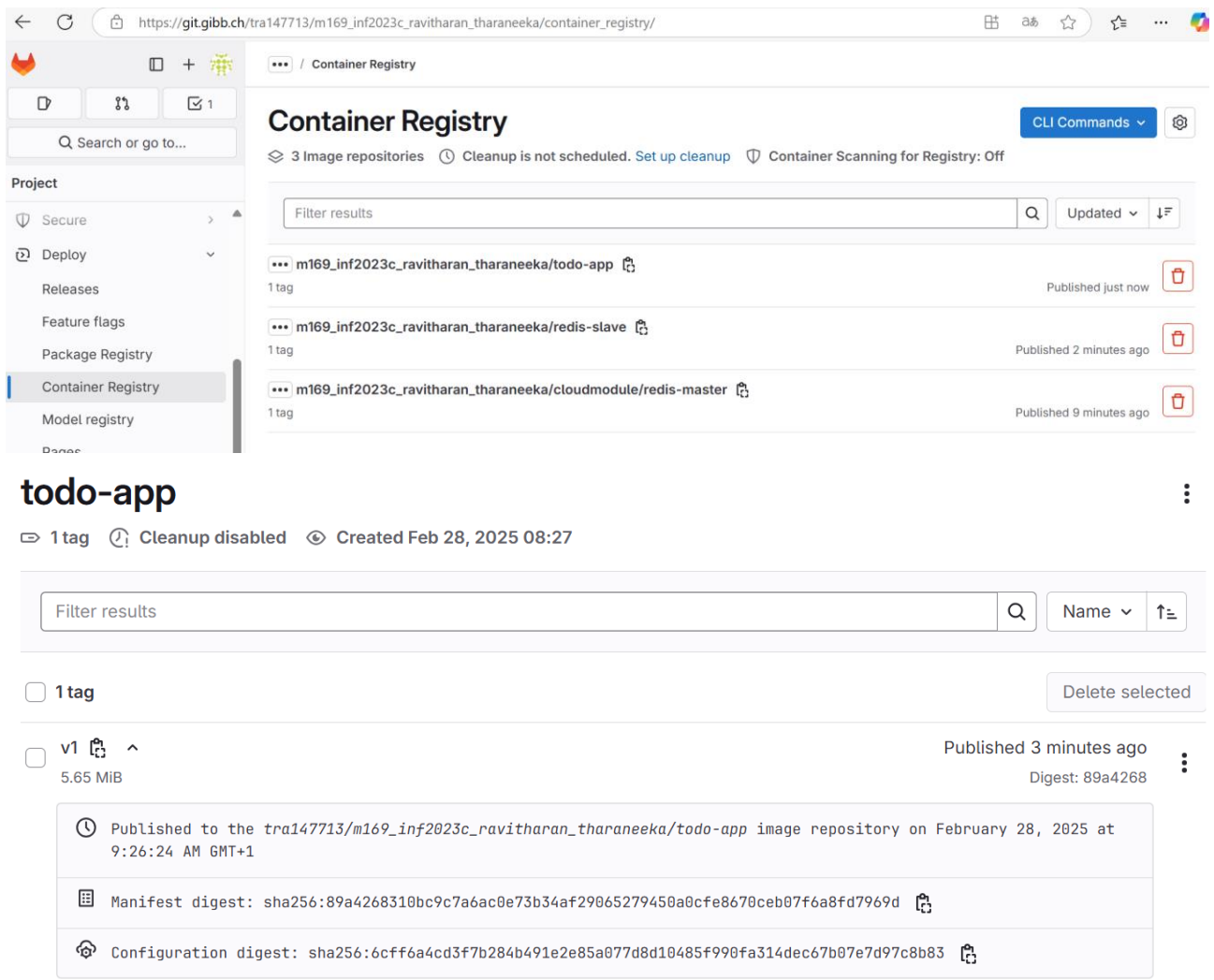


3 Tag 2, KW 6

3.1 Todo App Version 1



3.2 Image pushen auf GitLab



redis-slave

1 tag Cleanup disabled Created Feb 28, 2025 08:25

Name ▾
↑ ▾

☐ 1 tag

☐ v1 ^
11.29 MiB

Published 49 minutes ago
Digest: 160a394

Published to the tra147713/m169_inf2023c_ravitharan_tharaneeka/redis-slave image repository on February 28, 2025 at 8:41:23 AM GMT+1

Manifest digest: sha256:160a394b175b6712a48a56bd4377ccb5f9490be88726329d1778ffa4d93556d

Configuration digest: sha256:9fecf9aa080a7a125e3a1569fed0595c2180e56e060211b80b11eae034770d63

cloudmodule/redis-master

1 tag Cleanup disabled Created Feb 28, 2025 08:17

Name ▾
↑ ▾

☐ 1 tag

☐ v1 ^
11.29 MiB

Published 50 minutes ago
Digest: f8799c6

Published to the tra147713/m169_inf2023c_ravitharan_tharaneeka/cloudmodule/redis-master image repository on February 28, 2025 at 8:41:04 AM GMT+1

Manifest digest: sha256:f8799c6f5b3a5d15c9fb88496491a33153d0e560bd6a25e297d0c1043d1aca37

Configuration digest: sha256:0d8739e7b359377b83cbd3f5ff153cfe16265ab0ba54a20ec6625b4b85ba76c9

3.2.1 Befehle für das Pushen

1. Diese Befehle bauen Docker-Images aus den angegebenen Verzeichnissen und taggen sie mit den entsprechenden Namen und Tags. Die `-t`-Option wird verwendet, um dem Image einen Tag zu geben. Der Tag besteht aus dem Image-Namen und einer Versionsnummer:

```
docker build -t redis-master:v1 ./redis-master --provenance false
```

```
docker build -t redis-slave:v1 ./redis-slave --provenance false
```

```
docker build -t todo-app:v1 ./web-frontend --provenance false
```

2. Diese Befehle **taggen** (vergeben einen neuen Tag) für die zuvor erstellten Docker-Images, sodass sie in eine spezifische **Docker-Registry** hochgeladen werden können. Ein Tag ist eine Art "Alias" für das Image, das angibt, wie das Image in der Registry gespeichert werden soll:

```
docker image tag redis-master:v1 git
registry.gibb.ch/xxxxx/m169_inf2023c_xxxxxxx/redis-master:v1
```

```
docker image tag redis-slave:v1 git-registry.gibb.ch/xxxxx/m169_inf2023c_xxxxxx/redis-slave:v1
```

```
docker image tag todo-app:v1 git-registry.gibb.ch/xxxxx/m169_inf2023c_xxxxxx/todo-app:v1
```

3. Diese Befehle laden die zuvor lokal erstellten und mit neuen Tags versehenen Docker-Images in eine **Docker-Registry** hoch:

```
docker push git-registry.gibb.ch/xxxxx/m169_inf2023c_xxxxxx/redis-master:v1
```

```
docker push git-registry.gibb.ch/xxxxx/m169_inf2023c_xxxxxx/redis-slave:v1
```

```
docker push git-registry.gibb.ch/xxxxx/m169_inf2023c_xxxxxx/todo-app:v1
```

4. Diese Befehle laden die Docker-Images von der angegebenen Docker-Registry herunter. Wenn die Images in der Registry bereits vorhanden sind, werden sie auf dein System heruntergeladen, sodass man sie lokal verwenden kann:

```
docker pull git-registry.gibb.ch/xxxxx/xxxxx/redis-master:v1
```

```
docker pull git-registry.gibb.ch/xxxxx/xxxxx/redis-slave:v1
```

```
docker pull git-registry.gibb.ch/xxxxx/xxxxx/todo-app:v1
```

3.3 Todo App Version 2

The top screenshot shows the 'Awesome Todo App' running in a web browser at localhost:3000. The page displays a 'Todo list!' with a green checkmark icon. Below the icon, there is a table with two rows of todo items: 'Tharaneeka Ravitharan' and 'Neue Version Tharaneeka Ravitharan'. Each row has a 'Delete' checkbox. At the bottom, there is an input field labeled 'Todo' and two buttons: 'Add' (blue) and 'Delete' (red).

The bottom screenshot shows the Docker Registry interface for the 'todo-app' repository. The interface displays two tags: 'v1' and 'v2'. The 'v1' tag is selected and shows details: Published 44 minutes ago, Digest: 89a4268, and a list of digests for the manifest, configuration, and other components. The 'v2' tag is also shown with its details: Published 2 minutes ago, Digest: 23df5da, and a list of digests for the manifest, configuration, and other components.

4 Tag 3, KW 7

4.1 Docker Compose

Docker Compose ist ein Tool, mit dem man Multi-Container-Anwendungen in Docker definieren und verwalten kann. Es nutzt eine YAML-Datei (`docker-compose.yml`), um die Konfiguration von Containern, Netzwerken und Volumes zu beschreiben. Anschließend kann man mit einem einzigen Befehl (`docker-compose up`) die gesamte Anwendung starten. Das macht es besonders nützlich für die Entwicklung und das Testen komplexer Anwendungen mit mehreren abhängigen Diensten wie Datenbanken, Caching-Systemen oder Message Brokern.

4.2 Todo App Version 2 mit Docker Compose

1. Man muss die zweite Version (to-do-appv2) bauen und den alten Container ersetzen.
2. Neuen Container mit `to-do-appv2` bauen:
 - a. In das Verzeichnis der neuen Version navigieren:
`cd path/to/to-do-appv2`
 - b. Dann das neue Frontend-Image mit einer neuen Version (v2) bauen:
`docker build -t todo-app:v2`
3. Alten Frontend-Container stoppen und entfernen
 - a. Stoppen und entfernen des alten Containers:
`docker stop frontend`
`docker rm frontend`
 - b. Falls man das alte Image nicht mehr braucht, kann man es löschen:
`docker rmi todo-app:v1`
4. Die neue Version mit dem gleichen Netzwerk und Port starten:
`docker run --net=todoapp_network --name=frontend -d -p 3000:3000 todo-app:v2`
5. Überprüfen, ob alles funktioniert
Öffne `http://localhost:3000` und prüfe, ob die neue Version geladen wird.



```
1 version: "3"
2 services:
3   todoapp:
4     build: ./web-frontendv2
5     ports:
6       - "3000"
7     networks:
8       - todoapp_network
9   networks:
10    - todoapp_network
11 networks:
12   todoapp_network:
13     name: todoapp_network
14     driver: bridge
```

```

vmadmin@li1244-vmLP1:~/cloudmodule-main$ cd to-do-appv2
vmadmin@li1244-vmLP1:~/cloudmodule-main/to-do-appv2$ ls
docker-compose.yaml  README.md  web-frontendv2
vmadmin@li1244-vmLP1:~/cloudmodule-main/to-do-appv2$ docker-compose -f docker-compose.yaml up -d
Building todoapp
[+] Building 7.2s (11/11) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 258B
=> WARN: MaintainerDeprecated: Maintainer instruction is deprecated in favor of using label (line 2)
=> [internal] load metadata for docker.io/library/alpine:3.16.2
=> [auth] library/alpine:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/5] FROM docker.io/library/alpine:3.16.2@sha256:65a2763f593ae85fab3b5486dc9e80f744ec5b449f269b699b5efd37a07ad32e
=> => resolve docker.io/library/alpine:3.16.2@sha256:65a2763f593ae85fab3b5486dc9e80f744ec5b449f269b699b5efd37a07ad32e
=> [internal] load build context
=> => transferring context: 237B
=> CACHED [2/5] COPY ./bin/todo-app /app/todo-app
=> CACHED [3/5] COPY ./public /app/public
=> CACHED [4/5] RUN chmod +x /app/todo-app
=> CACHED [5/5] WORKDIR /app
=> exporting to image
=> => exporting layers
=> => exporting manifest sha256:2368fa2f955712ead8d31e5a3f8d67503779afda2c76b4da23961044747161c
=> => exporting config sha256:10a001759eedeFe8830cf88618ce34ea4cee2ea740516e7596df04e8d51afe1
=> => exporting attestation manifest sha256:b5b1e765f607face7b201c5de4d89bb0818d4603b1c83b1072ab31c0ad5f7085
=> => exporting manifest list sha256:4ecc0fbaee86017eb1b402d5124f6e55bece0a30ff03c2b4b5720f08ff1bf7d
=> => naming to docker.io/library/to-do-appv2_todoapp:latest
=> => unpacking to docker.io/library/to-do-appv2_todoapp:latest

1 warning found (use docker --debug to expand):
- MaintainerDeprecated: Maintainer instruction is deprecated in favor of using label (line 2)

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/tg5o2z5dmpdoi48e62cd30s60
WARNING: Image for service todoapp was built because it did not already exist. To rebuild this image you must use `docker-compose build` or `docker-compose up --build`.
Creating to-do-appv2_todoapp_1 ... done
vmadmin@li1244-vmLP1:~/cloudmodule-main/to-do-appv2$ docker ps

```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
15d7312bf07b	to-do-appv2_todoapp	"/todo-app"	22 seconds ago	Up 21 seconds	0.0.0.0:44893->3000/tcp	to-do-appv2_todoapp_1
8d63db507e54	to-do-appv1_todoapp	"/todo-app"	13 minutes ago	Up 13 minutes	0.0.0.0:41773->3000/tcp	to-do-appv1_todoapp_1
9b627a211a01	to-do-appv1_redis-slave	"docker-entrypoint.s..."	13 minutes ago	Up 13 minutes	6379/tcp	to-do-appv1_redis-slave_1
3badd9d0e521	to-do-appv1_redis-master	"docker-entrypoint.s..."	13 minutes ago	Up 13 minutes	6379/tcp	to-do-appv1_redis-master_1
cb656cdc36dc	todo-app:v2	"/todo-app"	6 days ago	Up 43 minutes	0.0.0.0:3000->3000/tcp	frontend
1ab16cb00e06	redis-slave:v1	"docker-entrypoint.s..."	6 days ago	Up 43 minutes	6379/tcp	redis-slave
16e98dbdde9e	redis-master:v1	"docker-entrypoint.s..."	6 days ago	Up 43 minutes	6379/tcp	redis-master
58187e695959	onlyoffice/documentserver	"/app/ds/run-documen..."	13 days ago	Up 45 minutes	443/tcp, 0.0.0.0:11012->80/tcp	xenodochial_bell

```

vmadmin@li1244-vmLP1:~/cloudmodule-main/to-do-appv2$ S

```

4.3 Portainer

portainer.io

New Portainer Installation

Please create the initial administrator user:

Username:

Password:

Confirm password:

⚠ The password must be at least 12 characters long.

☒ Allow collection of anonymous statistics. You can find more information about this in our [privacy policy](#).

[Restore Portainer from backup](#)

portainer.io COMMUNITY EDITION

Environment: None selected

Administration

- User-related
- Environment-related
- Registries
- Logs
- Notifications
- Settings

Quick Setup

Environment Wizard

Welcome to Portainer

We have connected your local environment of Docker to Portainer.

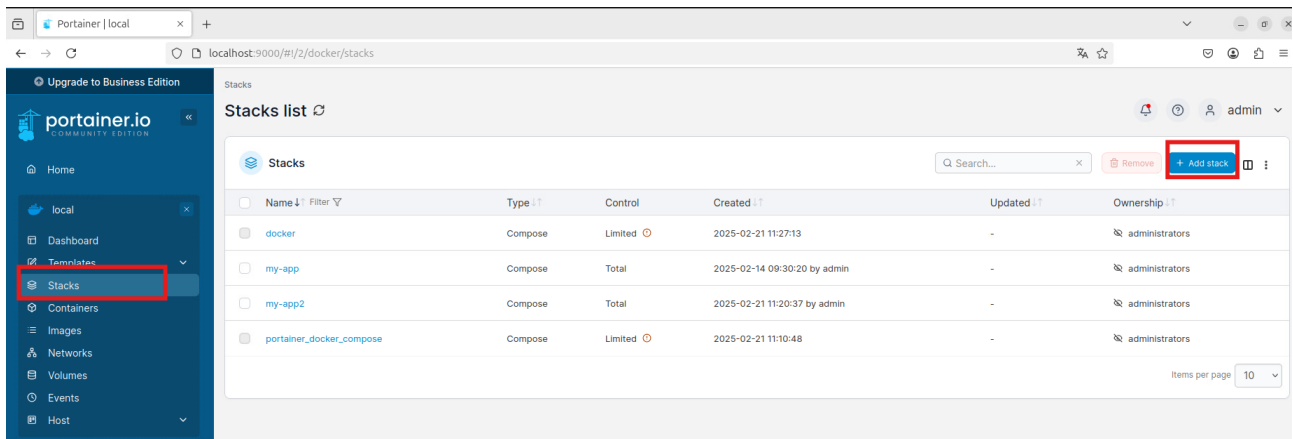
Get started below with your local portainer or connect more container environments.

Get Started
Proceed using the local environment which Portainer is running in

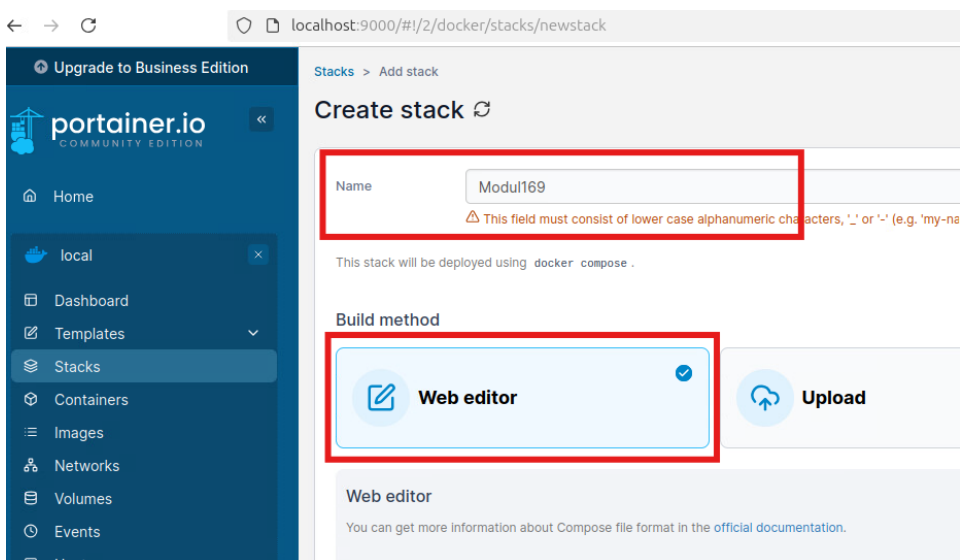
Add Environments
Connect to other environments

4.4 App via Portainer installiert

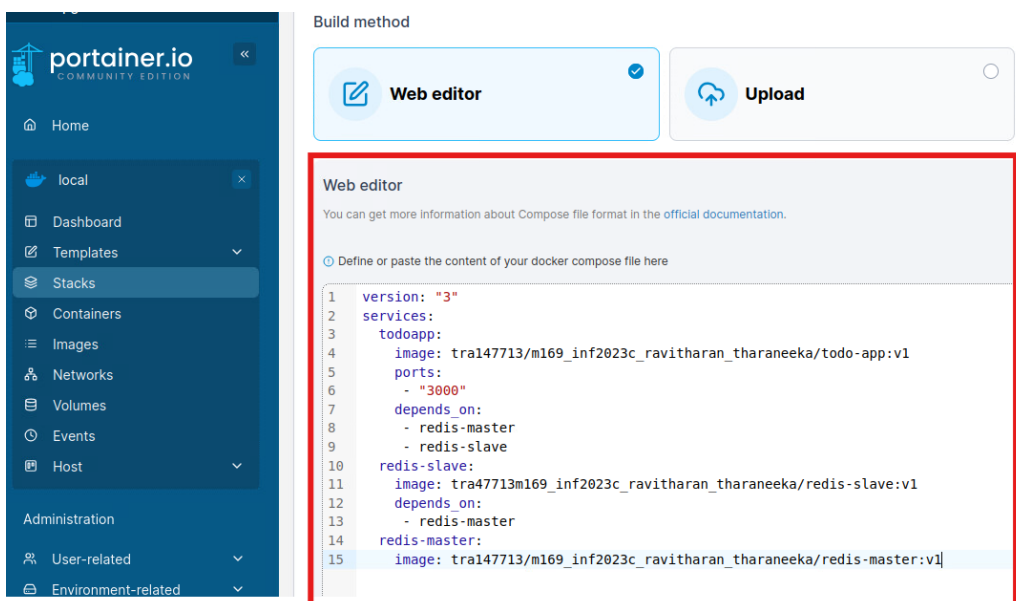
1. Portainer öffnet und einen neuen Stack erstellen:



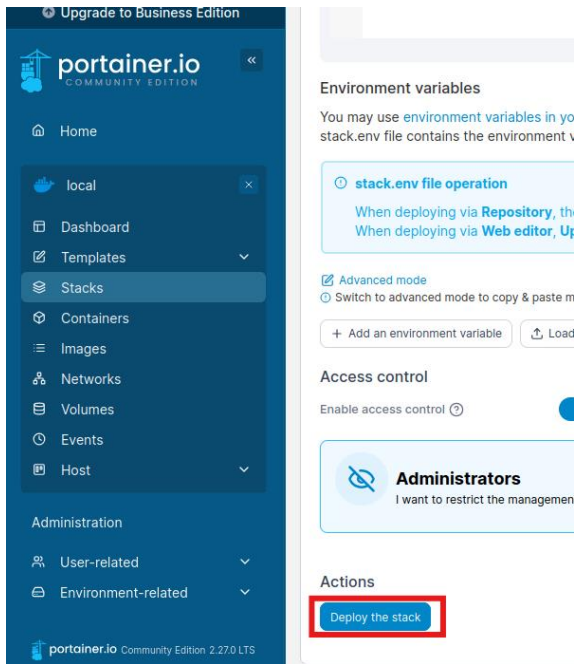
2. Dem Stack einen Namen geben und auf **Web Editor** klicken:



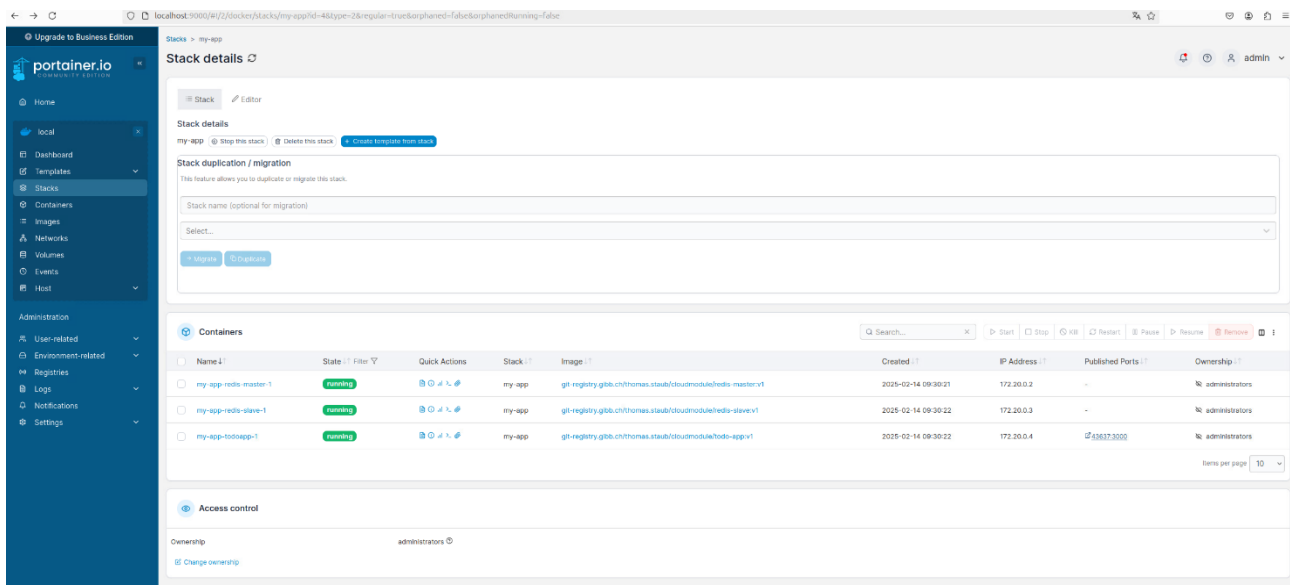
3. Dann den Inhalt vom **docker-compose.yaml** kopieren und unten einfügen:



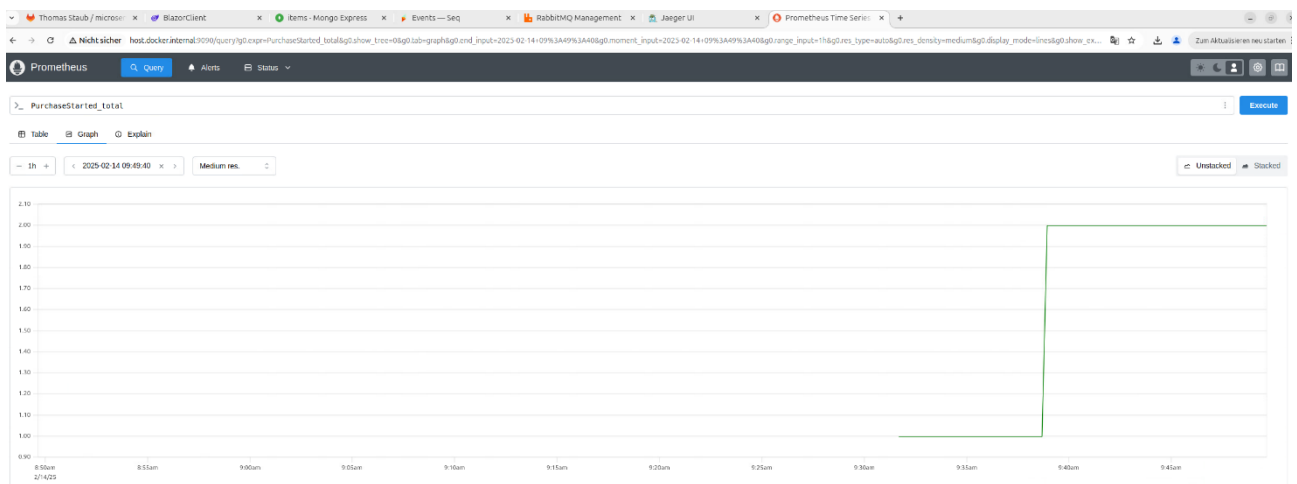
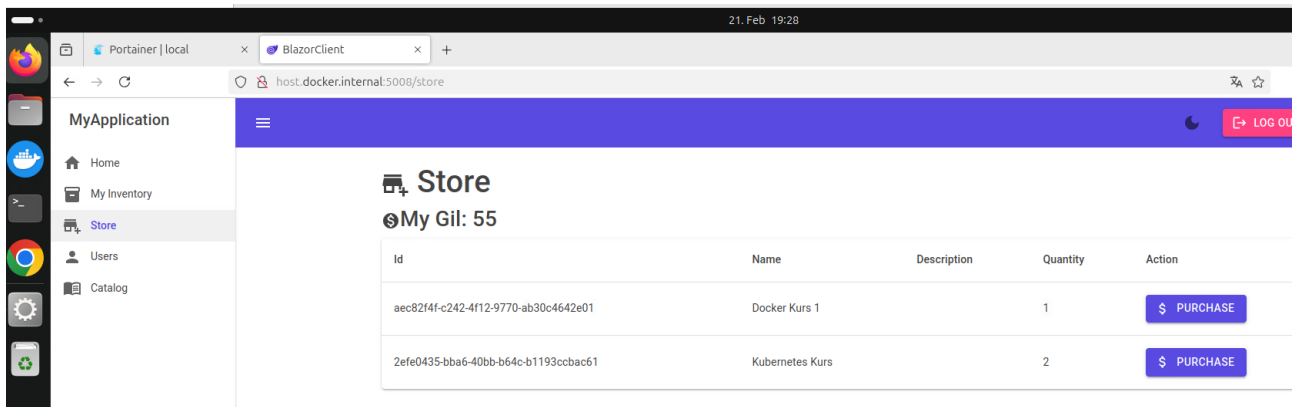
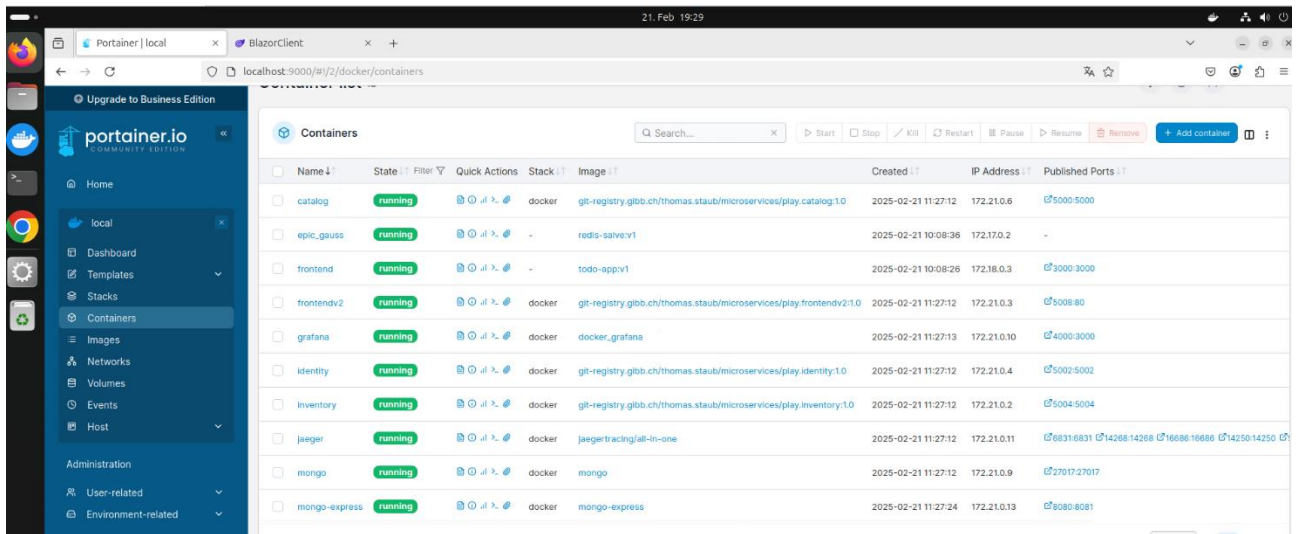
4. Und dann weiter unten auf **Deploy the stack** klicken:

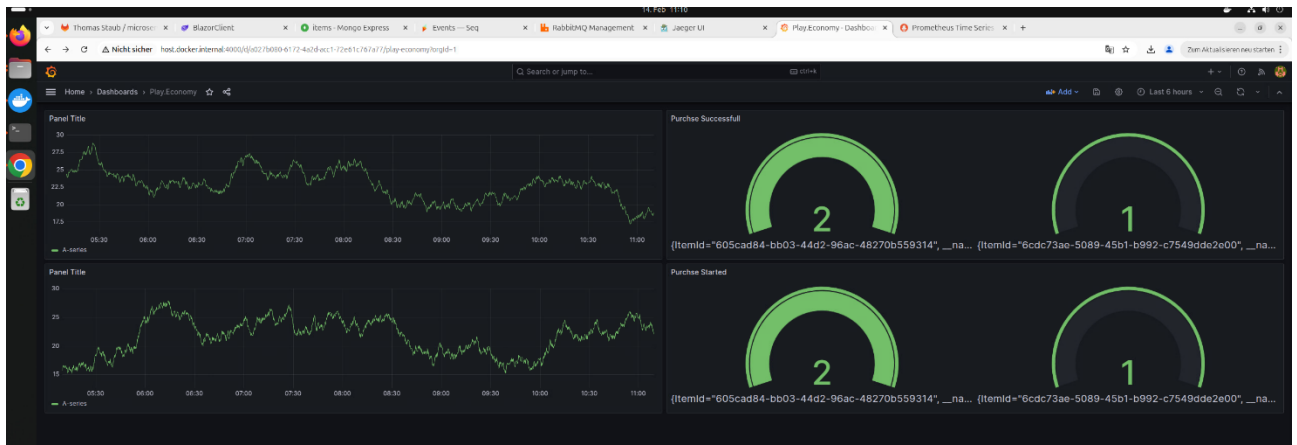


5. Die Stacks sollten dann mit ihrem Inhalt so aussehen sein:



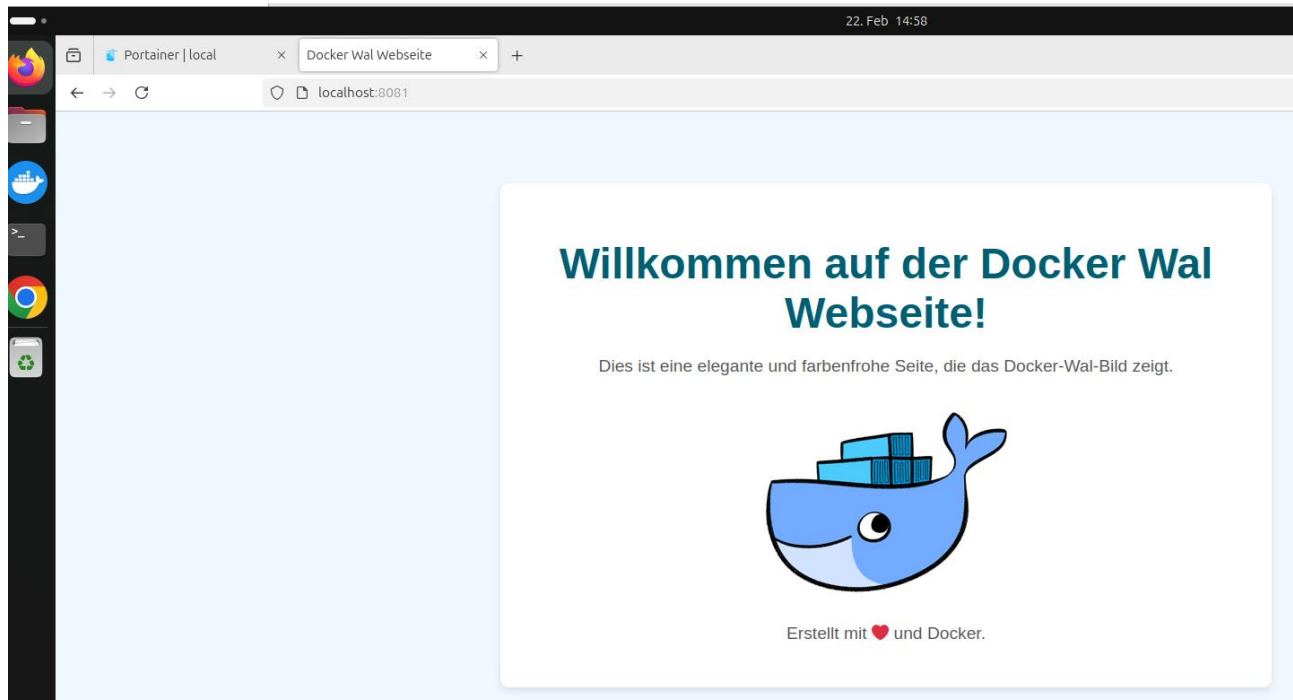
4.5 Shop mit Docker Compose





5 Übungsprojekt Webserver

Für mein Übungsprojekt habe ich einen Webserver in einem Docker-Container eingerichtet und das erstellte Image anschließend auf GitLab hochgeladen. Der Server läuft auf Port 8081. So sieht die Webseite derzeit aus:



Das gepushte Image auf dem Repository:

Tharaneeka Ravitharan / M169_INF2023c_Ravitharan_Tharaneeka

M169_INF2023c_Ravitharan_Tharaneeka

main

m169_inf2023c_ravitharan_tharaneeka

Initialer Commit
vmadmin authored 17 hours ago

9d9f68a5 History

Name	Last commit	Last update
nginx	Initialer Commit	17 hours ago
Dockerfile	Initialer Commit	17 hours ago
README.md	Initial commit	2 weeks ago
docker-compose.yml	Initialer Commit	17 hours ago

5.1 Anleitung Webserver

5.1.1 Schritt 1: GitLab Repository erstellen

1. Bei **GitLab anmelden** oder ein neues Konto erstellen.
2. Auf **"Neues Projekt"** → **"Create blank project"** klicken.
3. Einen passenden Namen vergeben, z. B. webserver-projekt.
4. Die Sichtbarkeit auf **"Privat"** oder **"Öffentlich"** setzen.
5. **"Create repository"** klicken.

5.1.2 Schritt 2: Projekt lokal einrichten

1. Das Projekt wird nun auf dem lokalen Rechner eingerichtet:

```
git clone https://gitlab.com/<NUTZERNAME>/webserver-projekt.git
cd webserver-projekt
```

5.1.3 Schritt 3: Docker-Setup einrichten

1. Folgende Dateien und Verzeichnisse im Projektordner erstellen:

```
mkdir nginx
touch nginx/index.html
touch nginx/default.conf
touch docker-compose.yml
touch Dockerfile
```

5.1.4 Schritt 4: HTML-Webseite erstellen

1. Die Datei `nginx/index.html` mit einem gewünschten Inhalt erstellen.

5.1.5 Schritt 5: Nginx-Konfiguration erstellen

1. Die Datei `nginx/default.conf` mit folgendem Inhalt erstellen:

```
nginx

server {
    listen 80;
    server_name localhost;

    root /usr/share/nginx/html;
    index index.html;

    location / {
        try_files $uri $uri/ =404;
    }

    location /bilder/ {
        alias /usr/share/nginx/html/bilder/;
    }
}
```


5.1.6 Schritt 6: Dockerfile erstellen

1. Die Datei `Dockerfile` mit folgendem Inhalt erstellen:

dockerfile

```
FROM nginx:latest
COPY nginx/index.html /usr/share/nginx/html/index.html
COPY nginx/default.conf /etc/nginx/conf.d/default.conf
COPY nginx/bilder /usr/share/nginx/html/bilder
```

5.1.7 Schritt 7: Docker Compose Datei erstellen

1. Die Datei `docker-compose.yml` mit folgendem Inhalt erstellen:

yaml

```
version: '3.8'
services:
  webserver:
    build: .
    ports:
      - "8081:80"
    volumes:
      - ./nginx/index.html:/usr/share/nginx/html/index.html
      - ./nginx/bilder:/usr/share/nginx/html/bilder
    restart: always
```

5.1.8 Schritt 8: Docker-Container starten

1. Das Projekt kann nun mit folgenden Befehlen ausgeführt werden:

```
docker-compose up -d
```

2. Die Webseite sollte nun unter <http://localhost:8081> im Browser aufrufbar sein.
3. Um den Container zu stoppen:

```
docker-compose down
```

5.1.9 Schritt 9: Projekt in GitLab hochladen

1. Das Projekt wird nun auf GitLab hochgeladen:

```
git add .  
git commit -m "Initialer Commit"  
git branch -M main  
git push -u origin main
```

5.1.10 Schritt 10: Änderungen an der Webseite veröffentlichen

Falls Änderungen an der Webseite vorgenommen werden sollen:

1. Die Datei `nginx/index.html` bearbeiten und speichern.
2. Änderungen hochladen:

```
git add nginx/index.html  
git commit -m "Webseite aktualisiert"  
git push origin main
```

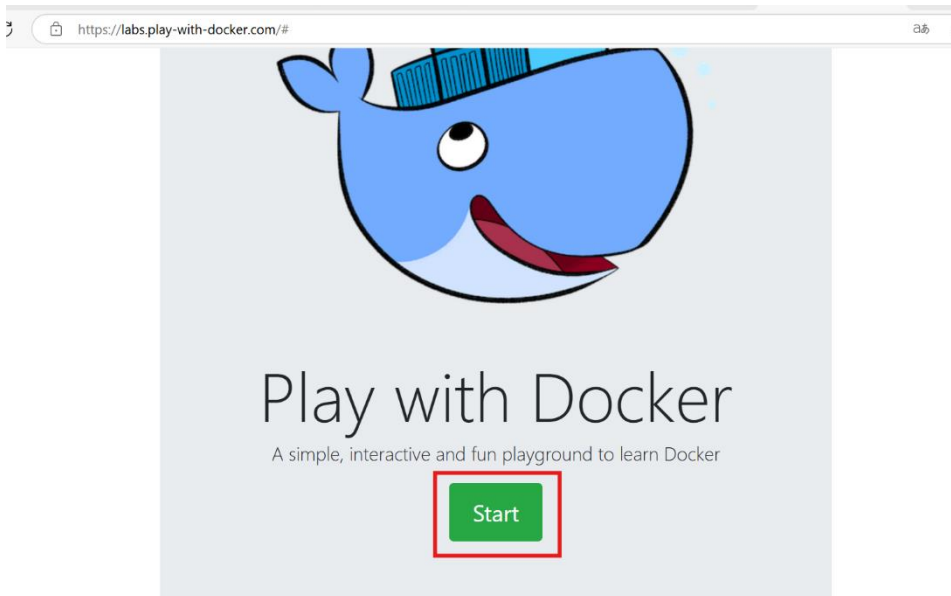
3. Den Container neu starten, damit die Änderungen aktiv werden:

```
docker-compose down  
docker-compose up -d
```

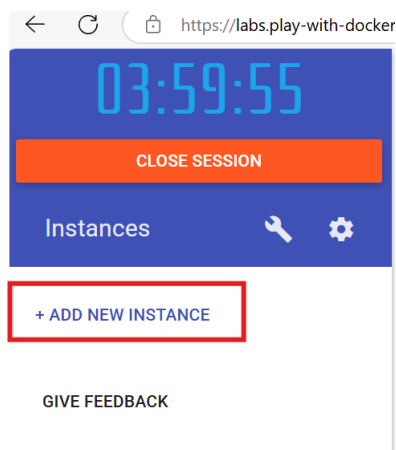
5.2 Anleitung für Lehrer

Jetzt folgt der Prozess, den der Lehrer in **Play with Docker** ausführen muss:

1. Schritt: Play with Docker öffnen: <https://labs.play-with-docker.com/>
2. Schritt: Ein neues Projekt starten
3. Der Lehrer muss auf den Button **"Start"** klicken, um ein neues Projekt zu starten. Das öffnet eine neue Docker-Umgebung.



4. Danach ist man in dieser Umgebung. Da auf **"ADD NEW INSTANCE"** klicken:



5. Im Terminal unten dran diesen Befehl eingeben:

```
git clone https://git.gibb.ch/tra147713/m169_inf2023c_ravitharan_tharaneeka
```

```
[node1] (local) root@192.168.0.28 ~
$ git clone https://git.gibb.ch/tra147713/m169_inf2023c_ravitharan_tharaneeka
Cloning into 'm169_inf2023c_ravitharan_tharaneeka'...
warning: redirecting to https://git.gibb.ch/tra147713/m169_inf2023c_ravitharan_tharaneeka.git/
remote: Enumerating objects: 12, done.
remote: Counting objects: 100% (12/12), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 12 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (12/12), 29.22 KiB | 14.61 MiB/s, done.
```

6. Danach in das Projektverzeichnis wechseln:

```
cd m169_inf2023c_ravitharan_tharaneeka
```

7. Docker-Container starten:

```
docker-compose up -d
```

```
[node1] (local) root@192.168.0.13 ~/m169_inf2023c_ravitharan_tharaneeka
$ docker-compose up -d
WARN[0000] /root/m169_inf2023c_ravitharan_tharaneeka/docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion
[+] Building 9.4s (10/10) FINISHED                                docker:default
=> [webserver internal] load build definition from Dockerfile      0.0s
=> => transferring dockerfile: 215B                                0.0s
=> [webserver internal] load metadata for docker.io/library/nginx:latest 1.5s
=> [webserver internal] load .dockerignore                        0.0s
=> => transferring context: 2B                                       0.0s
=> [webserver 1/4] FROM docker.io/library/nginx:latest@sha256:91734281c0ebfc6flaea979cfeed5079cfe7 7.5s
=> => resolve docker.io/library/nginx:latest@sha256:91734281c0ebfc6flaea979cfeed5079cfe786228a71cc 0.0s
```

8. Der Port sollte dann geöffnet werden und so zu sehen sein:

cuth9a0l_cuth9b0l2o9000f44r20

IP

192.168.0.13

OPEN PORT

8081

Memory

6.89% (275.5MiB / 3.906GiB)

CPU

0.19%

SSH

ssh ip172-18-0-14-cuth9a0l2o9000f44r1g@direct.labs.play



DELETE



EDITOR

9. Wenn man dann auf den klickt, sollte dann diese Webseite zusehen sein:



6 Befehle und ihre Funktionen

Befehle	Funktionen
<code>gpg --generate-key</code>	Schlüssel für Login in Docker-Desktop erstellen
<code>pass init <generated gpg-id public key></code>	Verbindung herstellen
<code>docker pull hello-world</code>	Lädt das hello-world Image herunter
<code>docker images -a</code>	Zeigt die Images im Terminal
<code>docker run hello-world</code>	Das Image wird ausgeführt
<code>docker run -it ubuntu bash</code>	Ausführung vom Image und Wechsel in ein anderes Verzeichnis
<code>docker ps -a</code>	Zeigt die Container im Terminal
<code>docker run --name some-nginx -d -p 8080:80 nginx</code>	Erzeugt einen neuen Container im Hintergrund und verbindet die Ports
<code>docker run --name some-nginx -d --rm -p 8080:80 nginx</code>	Erzeugt einen neuen Container und löscht diesen nach beenden gleich wieder
<code>docker kill some-nginx</code>	Stoppt den Container schnell aber auf die harte Tour
<code>docker stop some-nginx</code>	Stoppt den Container indem versch wird die einzelnen Prozesse sanft herunterzufahren
<code>docker rm some-nginx</code>	Container wird gelöscht
<code>docker start some-nginx</code>	Container wird gestartet
<code>docker system prune -a --volumes</code>	Löscht alle Conatiner, Images, Volumes vom System.
<code>docker logs some-nginx</code>	Zeigt die letzten Logs an
<code>docker build [URL]</code>	Erstellt ein Image aus einer Dockerfile-Quelle
<code>docker build -t</code>	Erstellt ein Image aus einer Dockerfile im aktuellen Verzeichnis und taggt es
<code>docker images</code>	Docker Images werden angezeigt
<code>docker ps -a</code>	Docker Container werden angezeigt
<code>docker rm</code>	Delete a container (if it is not running)
<code>docker update</code>	Update the configuration of one or more containers
<code>docker start</code>	Start a container
<code>docker stop</code>	Stop a running container
<code>docker restart</code>	Stop a running container and start it up again
<code>docker pause</code>	Pause processes in a running container
<code>docker unpause</code>	Unpause processes in a running container
<code>docker wait</code>	Block a container until others stop (after which it prints their exit codes)
<code>docker kill</code>	Kill a container by sending a SIGKILL to a running container
<code>docker attach</code>	Attach local standard input, output, and error streams to a running container
<code>docker pull [IMAGE]</code>	Pull an image from a registry
<code>docker push [IMAGE]</code>	Push an image to a registry
<code>docker import [URL/FILE]</code>	Create an image from a tarball
<code>docker commit [CONTAINER] [NEW_IMAGE_NAME]</code>	Create an image from a container
<code>docker rmi [IMAGE]</code>	Remove an image
<code>docker load [TAR_FILE/STDIN_FILE]</code>	Load an image from a tar archive or stdin
<code>docker image ls</code>	List all images that are locally stored with the docker engine
<code>docker history [IMAGE]</code>	Show the history of an image
<code>docker network ls</code>	List networks
<code>docker network rm [NETWORK]</code>	Remove one or more networks

<code>docker network inspect [NETWORK]</code>	Show information on one or more networks
<code>docker network connect [NETWORK] [CONTAINER]</code>	Connects a container to a network
<code>docker network disconnect [NETWORK] [CONTAINER]</code>	Disconnect a container from a network
<code>docker login git-registry.gibb.ch</code>	Git Login
<code>docker image tag redis-master:v1 staubth/redis-master:v1</code>	Image taggen
<code>docker image push staubth/redis- master:v1</code>	Image pushen
<code>docker logs fe -f</code>	Loganzeige Streamen so sehen wir jeweils, wenn ein Eintrag geändert wird.
<code>docker-compose up -d</code>	Container starten, die sich in der yml-Datei befindet