
Modul 169

Modul	IET-169
Eingereicht von	Davide
Eingereicht bei	Reto Brüttsch
Datum	23. Februar 2025

1.0 Versionskontrolle

Versionskontrolle

Version	Datum	Verantwortlich	Bemerkung
0.0.1	31.01.2025	Davide	<p>Titelseite (Namen, Klasse, Modul, Datum, Version, grafisches Wiedererkennungsmerkmal) sowie Seitenzahlen, Kurze Beschreibung was Container sind und wo diese Nützlich sind.</p> <p>Kurze Beschreibung zu: Was ist Devops?</p> <p>Eintrag zu: Unterschied Virtualisierung und Containerisierung.</p> <p>Eintrag zu: Unterschied Image und Conatiner.</p> <p>Zusammenfassung der Wichtigsten Befehle und ihre funktion (Diese wird laufend ergänzt)</p> <p>PrintScreen OnlyOffice mit eigenem Namen und Datum im Dokument</p>
0.0.2	07.02.2025	Davide	<p>App version 1</p> <p>PrintScreen ihrer Version 1 mit Ihrem Namen als Todo Task</p> <p>PrintScreen der Images bei (gibb) GitLab</p> <p>Aufgabe 1 bei "Ein Image Pushen", Halten Sie alle befehle im Portfolio fest.</p> <p>App version 2 Frontend</p> <p>PrintScreen ihrer Version 2 mit Ihrem Namen als Todo Task</p> <p>PrintScreen der Images bei GitLab</p>
0.0.3	14.02.2025	Davide	<p>Kurze Beschreibung was Docker Compose ist</p> <p>Version 2 der App mit Docker Compose zum Laufen gebracht.</p> <p>Vorgehen, Befehle und Compose Datei im Portfolio festgehalten</p> <p>Portainer installiert und mit PrintScreen festgehalten</p> <p>App via Portainer installiert und im Portfolio vorgehen dokumentiert.</p> <p>Das Learning Beispiel Shop via Docker Compose installiert. Printsreen Wie es läuft und auch von Prometheus und Grafana festgehalten</p> <p>Vorgehen im Portfolio festgehalten.</p>
0.0.4	21.02.2025	Davide	<p>erstellen ein eigenes Projekt und erzeugen ein Image davon.</p> <p>Das Image pushen Sie in das Repository (Printscreen)</p> <p>Anleitung, wie das Image nun von jemand anderem (Lehrperson) installiert werden kann</p>

2.0 Inhaltsverzeichnis

1.0	Versionskontrolle	2
2.0	Inhaltsverzeichnis	3
3.0	Abbildungsverzeichnis	4
4.0	Was sind Container?	5
5.0	Wo sind Container nützlich?	5
6.0	Was ist DevOps?	5
7.0	Was ist der Unterschied von Virtualisierung und Containerisierung?	5
8.0	Was ist der Unterschied von Image und Conatiner?	6
9.0	Zusammenfassung wichtiger Docker Kommandos bei Containern:	6
10.0	Zusammenfassung wichtiger Docker Kommandos bei Images:	7
11.0	Zusammenfassung wichtiger genereller Docker Kommandos:	7
12.0	PrintScreen OnlyOffice mit eigenem Namen und Datum:	8
13.0	PrintScreen der Version 1 mit eigenem Namen als Todo Task	8
14.0	PrintScreen der Images bei (gibb) GitLab	9
15.0	Wichtigste Befehle, um ein Image zu pushen:	9
16.0	PrintScreen der Version 2 mit eigenem Namen als Todo Task	10
17.0	PrintScreen der Images bei (gibb) GitLab	10
18.0	Was ist Docker Compose	11
19.0	Version 2 der App mit Docker Compose zum laufen bringen	11
20.0	Vorgehen und Befehle und Compose Datei	12
	Compose Datei	12
	Vorgehen	12
	Wichtige Befehle:	13
21.0	Portainer PrintScreen	14
	Vorgehen	15
22.0	Learning Beispiel Shop	17
	Vorgehen	17
	Zusätzliche Services	21
	Prometheus	22
	Für was benötigen wir Prometheus?	22
	Grafana	23
23.0	Eigenes Projekt: Webserver	25
	Dockerfile	25
	Docker Compose Datei	25
	PrintScreen des Images bei (gibb) GitLab	26
	Anleitung	26
	Resultat	26

3.0 Abbildungsverzeichnis

Abbildung 1: OnlyOffice Screenshot	8
Abbildung 2: PrintScreen der 1. Version	8
Abbildung 3: PrintScreen der Images bei GitLab	9
Abbildung 4: PrintScreen der 2. Version	10
Abbildung 5: PrintScreen der Images bei GitLab mit 2 Tags bei Todo-App	10
Abbildung 6: Docker Compose	11
Abbildung 7: Compose Datei für Todo_appv2	12
Abbildung 8: Portainer erstelltes Environment	14
Abbildung 9: Stack Details	14
Abbildung 10: Einloggen bei Portainer	15
Abbildung 11: Neues Stack auf Portainer erstellen	15
Abbildung 12: Skript von Thomas Staub verwenden	16
Abbildung 13: Docker Compose von Portainer	16
Abbildung 14: Thomas Staub's Repository	17
Abbildung 15: Downloads Ordner	17
Abbildung 16: In den microservices Order wechseln	18
Abbildung 17: yaml Datei ausführen	18
Abbildung 18: Microservices via Docker Compose aufgesetzt	18
Abbildung 19: Beispiel Shop	18
Abbildung 20: Anmelden	19
Abbildung 21: Katalog von Beispielshop	19
Abbildung 22: virtuelles Geld hinzufügen	20
Abbildung 23: Store von Shop	20
Abbildung 24: Mongo Express	21
Abbildung 25: Mongo Express Catalog	21
Abbildung 26: Events Webseite	22
Abbildung 27: Prometheus Targets	22
Abbildung 28: Prometheus PurchaseStarted Graph	23
Abbildung 29: Startseite von Grafana	23
Abbildung 30: Grafana Dashboard vom Shop	24
Abbildung 31: GitLab von Webserver	25
Abbildung 32: Docker Compose Datei für Webserver	25
Abbildung 33: Webserver Webseite	26

4.0 Was sind Container?

Container stellen eine Virtualisierungstechnik im Computerumfeld dar, die Anwendungen inklusive ihrer Laufzeitumgebungen voneinander trennt. Anders als virtuelle Maschinen haben Container kein eigenes Betriebssystem, sondern nutzen das Betriebssystem des Hostsystems, auf dem sie installiert sind. Alle für die Ausführung erforderlichen Dateien, Konfigurationen, Abhängigkeiten und Bibliotheken sind im Container enthalten.

Eine der bekanntesten Container Lösungen ist Docker. Docker ist eine Open Source Software, welche die benötigten Funktionen zur Virtualisierung der Anwendungen und Isolierung der Container auf einem Hostsystem bereitstellt.

5.0 Wo sind Container nützlich?

Container sind besonders nützlich, um Anwendungen in Cloud Computing Umgebungen bereitzustellen und Anwendungen von Entwicklungs in Produktionsumgebungen zu verschieben. Container ermöglichen es, Anwendungen schnell und konsistent auf verschiedenen Systemen auszuführen, ohne sich um Abhängigkeiten oder unterschiedliche Umgebungen kümmern zu müssen. Sie sind besonders hilfreich für die Skalierbarkeit und Isolation, da mehrere Container auf demselben Host laufen können, ohne sich gegenseitig zu beeinflussen. Ausserdem unterstützen sie eine effiziente Nutzung von Ressourcen, da sie im Vergleich zu virtuellen Maschinen weniger Overhead verursachen.

6.0 Was ist DevOps?

Der Begriff DevOps ist eine Kombination aus Development und Operations. Es ist ein Ansatz zur Automatisierung und Integrierung von Prozessen zwischen Softwareentwicklung und IT Betriebsteams, damit Software schneller und zuverlässiger erstellt, getestet und veröffentlicht werden kann. DevOps zielt darauf ab, die sogenannte „Wall of confusion“ (Barriere) zwischen traditionell isolierten Entwicklungs und IT Betriebsteams abzubauen.

7.0 Was ist der Unterschied von Virtualisierung und Containerisierung?

Bei der Virtualisierung ist das Paket, welches erstellt wird, eine virtuelle Maschine und beinhaltet sowohl ein vollständiges Betriebssystem als auch die Applikation. Ein physischer Server, auf dem drei virtuelle Maschinen ausgeführt werden, verfügt über einen Hypervisor und drei separate Systeme. Im Gegensatz dazu läuft auf einem Server mit drei containerisierten Applikationen nur ein Betriebssystem. Die Container teilen sich einen Betriebssystemkernel. Dank dieser Konstellation benötigen Container weniger Ressourcen als virtuelle Maschinen.

8.0 Was ist der Unterschied von Image und Conatiner?

Ein Docker Image ist eine unveränderliche Datei, welche Quellcode, Abhängigkeiten, Tools und andere Dateien enthält, welche für die Ausführung einer Anwendung (Applikation) erforderlich sind. Anders gesagt, Docker Images sind im Wesentlichen für die Steuerung und Gestaltung von Containern verantwortlich.

Sobald man einen Container erstellt, wird dem unveränderlichen Image eine beschreibbare Schicht hinzugefügt, was bedeutet, dass man es jetzt ändern kann.

Images können ohne Container existieren, während ein Container ein Image ausführen muss, um zu existieren. Ein Container ist somit ein laufendes Image. Containers sind daher von Images abhängig und verwenden diese, um eine Laufzeitumgebung zu schaffen und eine Anwendung auszuführen.

9.0 Zusammenfassung wichtiger Docker Kommandos bei Containern:

Container löschen

- `docker rm [CONTAINER]`

Die Konfiguration eines Containers aktualisieren

- `docker update [CONTAINER]`

Container starten

- `docker start [CONTAINER]`

Container stoppen

- `docker stop [CONTAINER]`

Container neu starten

- `docker restart [CONTAINER]`

Laufende Prozesse in einem Container anhalten

- `docker pause [CONTAINER]`

10.0 Zusammenfassung wichtiger Docker Kommandos bei Images:

Alle Images auflisten, die lokal in der Docker Engine gespeichert sind

- `docker image ls`

Image aus einer Dockerdatei erstellen

- `docker build [URL]`

Image aus einem Repository herunterladen

- `docker pull [IMAGE]`

Image aus einem Container erstellen

- `docker commit [CONTAINER] [NEW_IMAGE_NAME]`

Image entfernen

- `docker rmi [IMAGE]`

Den Verlauf eines Images anzeigen

- `docker history [IMAGE]`

11.0 Zusammenfassung wichtiger genereller Docker Kommandos:

Container aus einem bestehenden Image starten

- `docker run`

Alle laufenden Container anzeigen

- `docker ps`

Dieser Befehl ermöglicht es, einen Befehl innerhalb eines laufenden Containers auszuführen

- `docker exec`

Detaillierte Informationen zu einem Container oder Image abrufen

- `docker inspect`

12.0 PrintScreen OnlyOffice mit eigenem Namen und Datum:

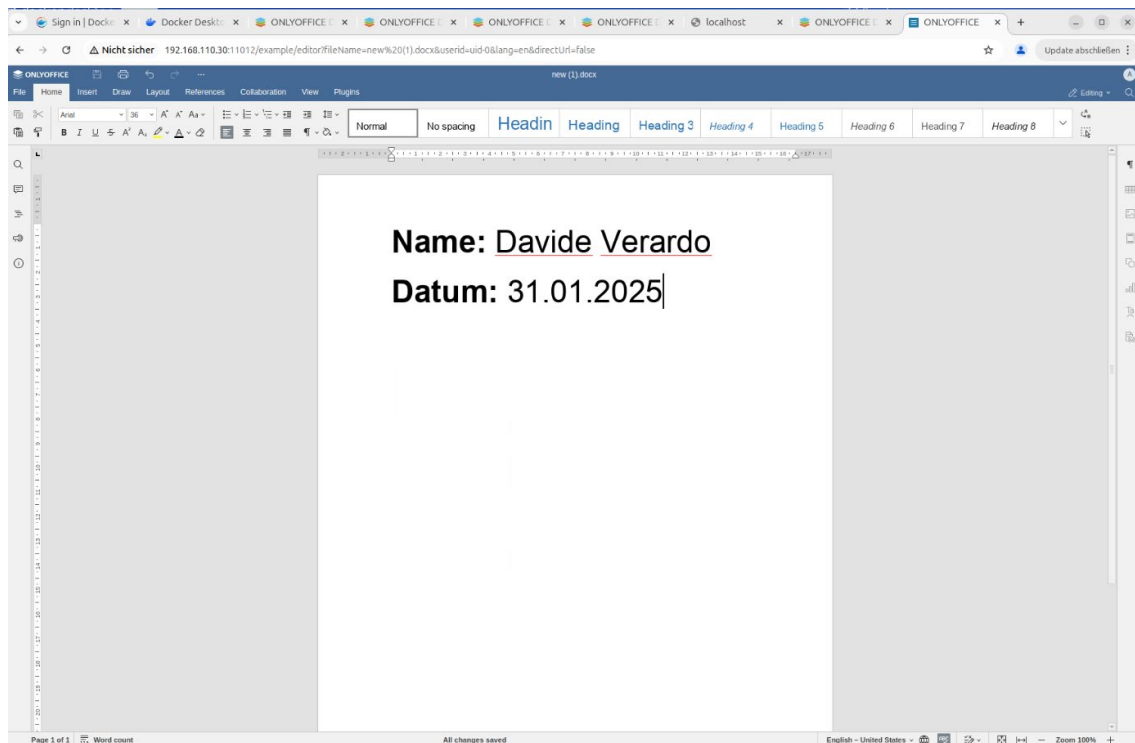


Abbildung 1: OnlyOffice Screenshot

13.0 PrintScreen der Version 1 mit eigenem Namen als Todo Task

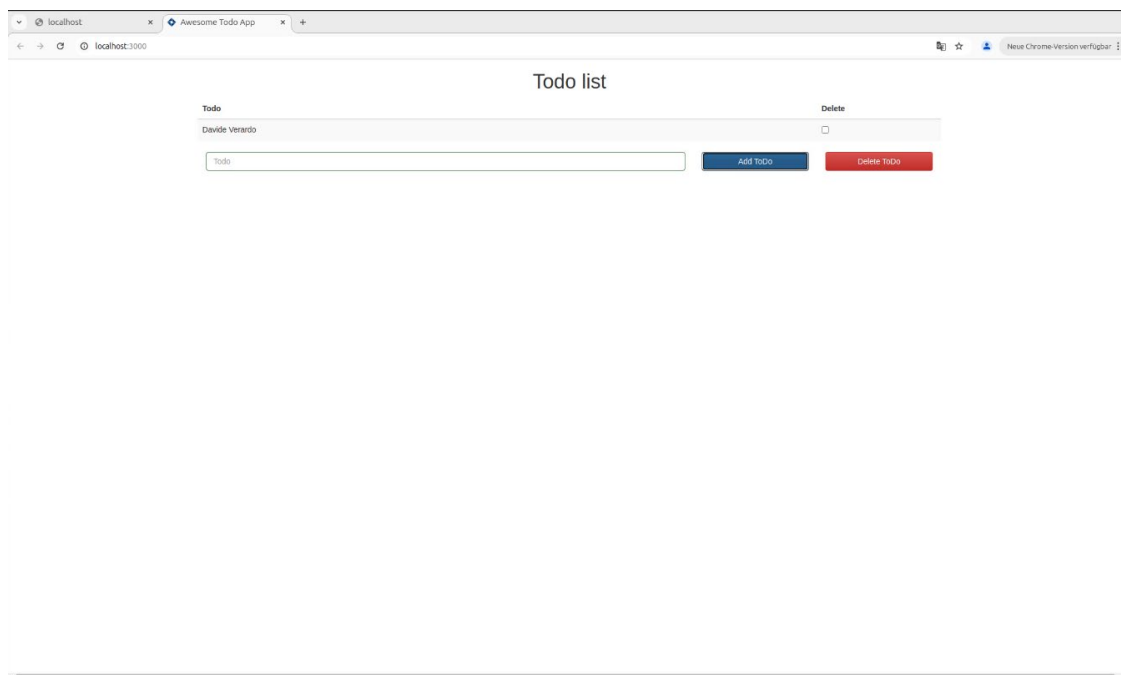


Abbildung 2: PrintScreen der 1. Version

14.0 PrintScreen der Images bei (gibb) GitLab

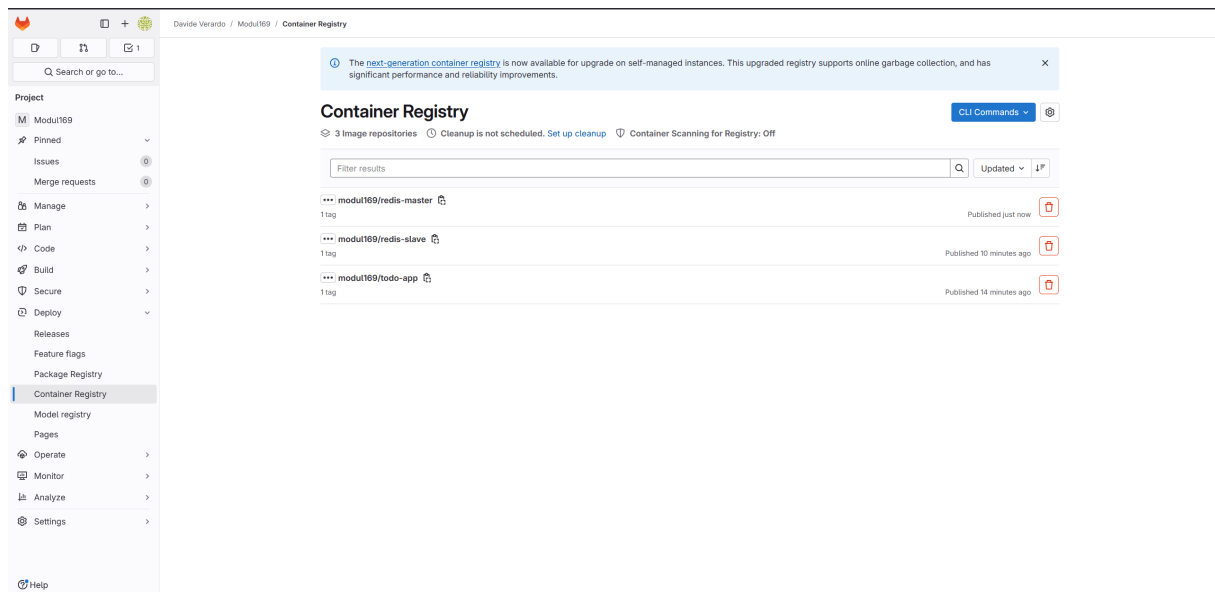


Abbildung 3: PrintScreen der Images bei GitLab

15.0 Wichtigste Befehle, um ein Image zu pushen:

- `docker login git-registry.gibb.ch`
- `docker image tag todo-app:v1 git-registry.gibb.ch/dve146329/modul169/todo-app:v1`
- `docker image tag redis-slave:v1 git-registry.gibb.ch/dve146329/modul169/redis-slave:v1`
- `docker image tag redis-master:v1 git-registry.gibb.ch/dve146329/modul169/redis-master:v1`
- `docker image tag todo-app:v2 git-registry.gibb.ch/dve146329/modul169/todo-app:v2`
- `docker push git-registry.gibb.ch/dve146329/modul169/todo-app:v1`
- `docker push git-registry.gibb.ch/dve146329/modul169/redis-slave:v1`
- `docker push git-registry.gibb.ch/dve146329/modul169/redis-master:v1`
- `docker run --net=todoapp_network --name=redis-master -d redis-master:v2`
- `docker run --net=todoapp_network --name=redis-slave -d redis-slave:v2`
- `docker run --net=todoapp_network --name=frontend -d -p 3000:3000 todo-app:v2`
- `docker run --net=todoapp_network --name=redis-master -d redis-master:v1`
- `docker run --net=todoapp_network --name=redis-slave -d redis-slave:v1`
- `docker run --net=todoapp_network --name=frontend -d -p 3001:3000 todo-app:v1`

16.0 PrintScreen der Version 2 mit eigenem Namen als Todo Task

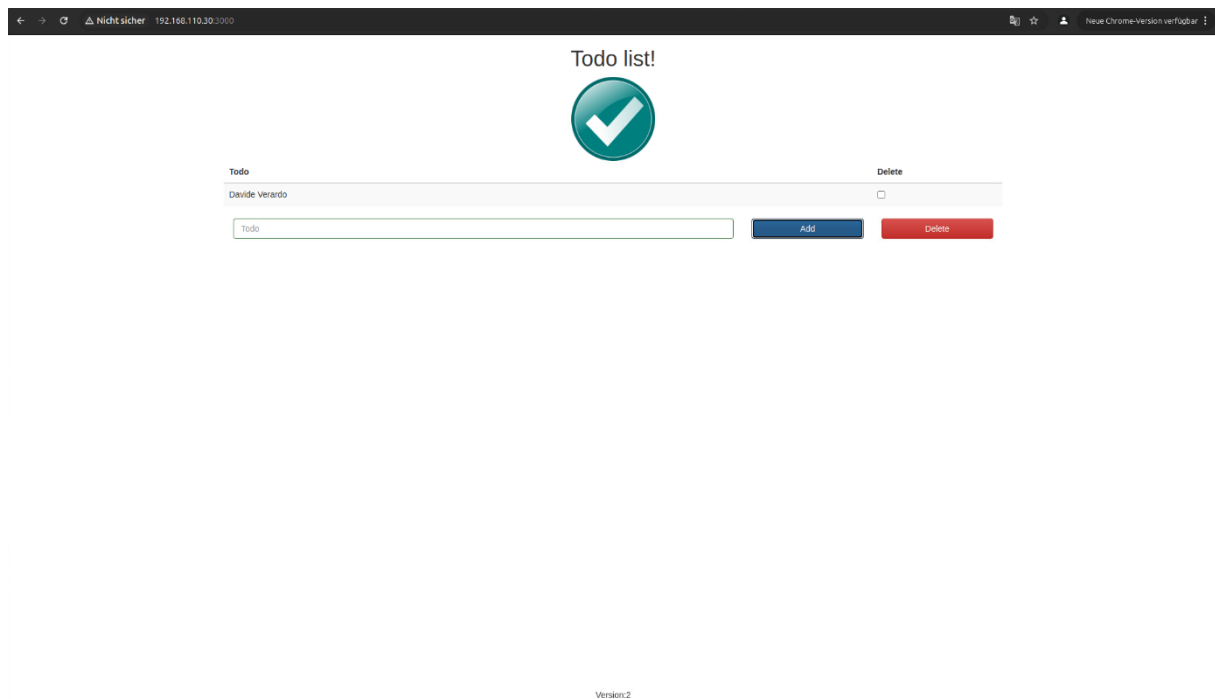


Abbildung 4: PrintScreen der 2. Version

17.0 PrintScreen der Images bei (gibb) GitLab

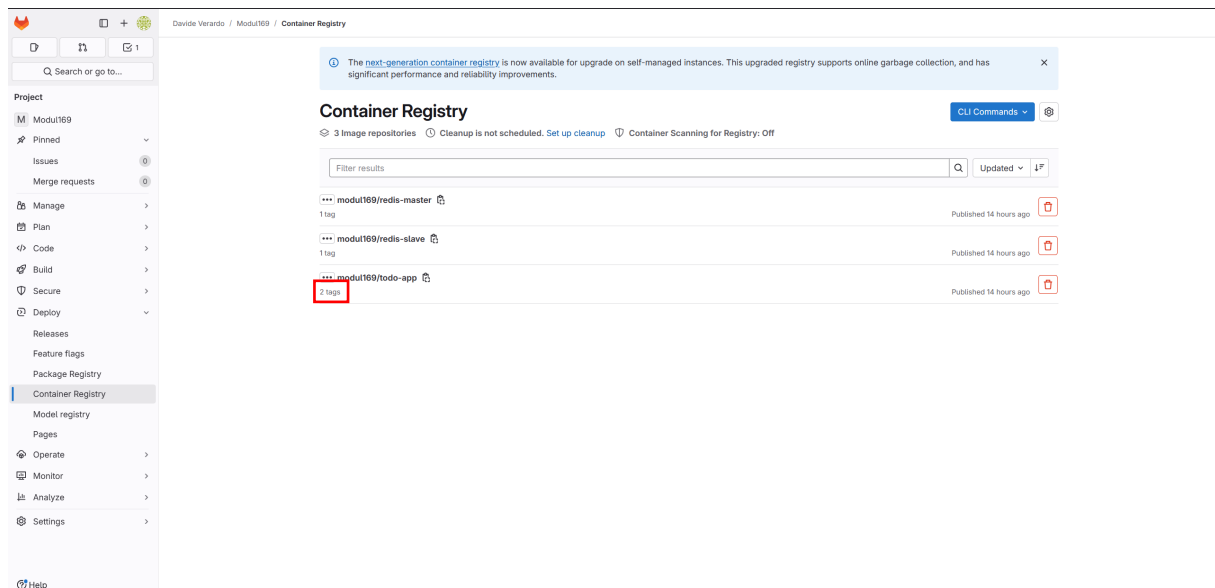


Abbildung 5: PrintScreen der Images bei GitLab mit 2 Tags bei Todo-App

18.0 Was ist Docker Compose

Docker Compose ist ein optionales Tool, welches verwendet wird, um mehrere Container aufzusetzen. Dabei werden die Konfigurationen der verwendeten Dienste in einer YAML Datei festgelegt. Schliesslich kann man mit einem Befehl alle Dienste aus der Konfigurationsdatei erstellen sowie starten. Docker Compose erleichtert das Management komplexer Anwendungen, indem es die Verwaltung mehrerer Container vereinfacht und deren Konfiguration zentralisiert. Mit Docker Compose können Entwickler zudem die Umgebung konsistent zwischen verschiedenen Systemen und Teammitgliedern reproduzieren, was die Entwicklungs und Bereitstellungsprozesse erheblich beschleunigt

19.0 Version 2 der App mit Docker Compose zum laufen bringen

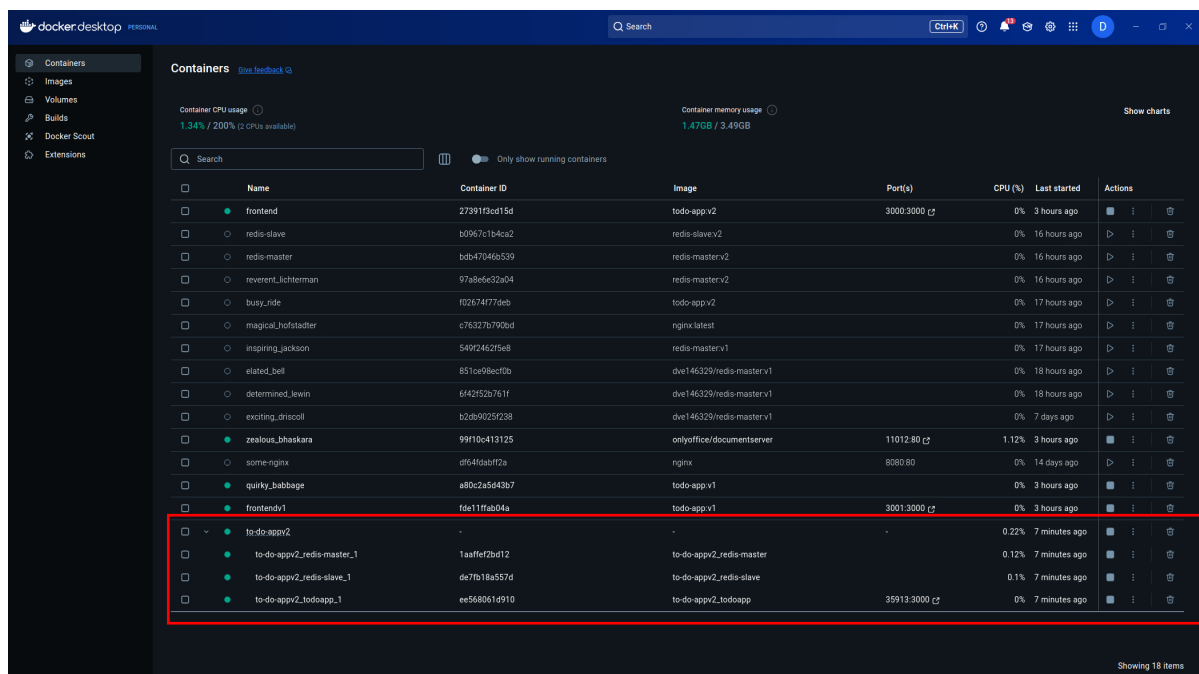


Abbildung 6: Docker Compose

20.0 Vorgehen und Befehle und Compose Datei

Compose Datei

```

version: "3"

services:
  todoapp:
    build: /home/vmadmin/Downloads/cloudmodule-main-to-do-appv2/to-do-appv2/web-frontendv2
    ports:
      - "3000"

    depends_on:
      - redis-master
      - redis-slave

    networks:
      - todoapp_network

  redis-slave:
    build: ./redis-slave
    depends_on:
      - redis-master
    networks:
      - todoapp_network

  redis-master:
    build: ./redis-master
    networks:
      - todoapp_network

networks:
  todoapp_network:
    name: todoapp_network
    driver: bridge

```

Abbildung 7: Compose Datei für Todo_appv2

Vorgehen

- **Docker Compose installieren**

sudo apt install docker-compose

- **"docker-compose.yml" Datei erstellen und denn nötigen Inhalt eintragen**

sudo vi /home/vmadmin/Downloads/cloudmodule-main-to-do-appv2/to-do-appv2/docker-compose.yml

"docker-compose.yml" ausführen

- sudo docker-compose -f /home/vmadmin/Downloads/cloudmodule-main-to-do-appv2/to-do-appv2/docker-compose.yml up -d

Wichtige Befehle:

"docker-compose.yml" Datei erstellen und dann nötigen Inhalt eintragen

- `sudo vi /home/vmadmin/to-do-appv2/docker-compose.yml`

"docker-compose.yml" ausführen

- `sudo docker-compose -f /home/vmadmin/Downloads/cloudmodule-main-to-do-appv2/to-do-appv2/docker-compose.yml up -d`

"docker-compose.yml" stoppen und Container entfernen

- `sudo docker-compose -f /home/vmadmin/Downloads/cloudmodule-main-to-do-appv2/to-do-appv2/docker-compose.yml down`

Container Logs anzeigen

- `docker logs fe...`

Laufende Container listen

- `docker ps`

Live Logs verfolgen

- `docker logs fe -f`

create docker network

- `docker network create todoapp_network`

show docker network

- `docker network ls`

build a docker image

- `docker build -t todo-app:v1 .`

App starten

- `docker run --net=todoapp_network --name=frontend -d -p 3000:3000 todo-app:v1`

Log anzeigen

- `docker logs <container id>`

stream the log

- `docker logs -f <container id>`

Container entfernen

- `docker rm todo-app:v1`

Image entfernen

- `docker rmi todo-app:v1`

System bereinigen

- `docker system prune -a --volumes` #Löscht alle Conatiner, Images, Volumes vom System.

21.0 Portainer PrintScreen

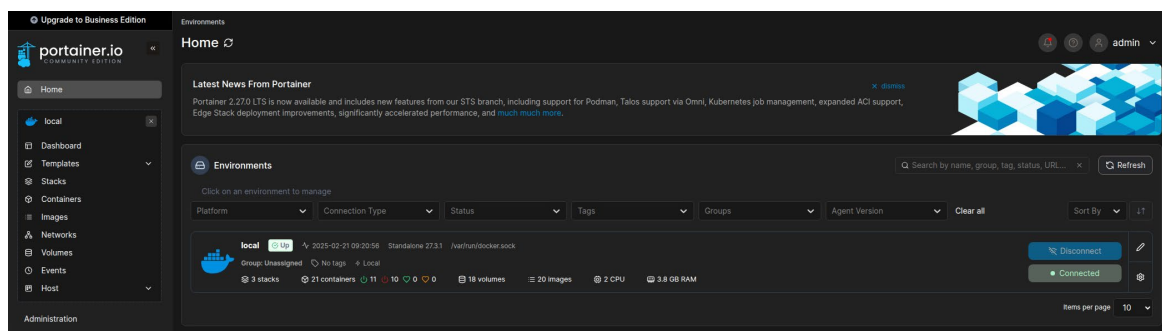


Abbildung 8: Portainer erstelltes Environment

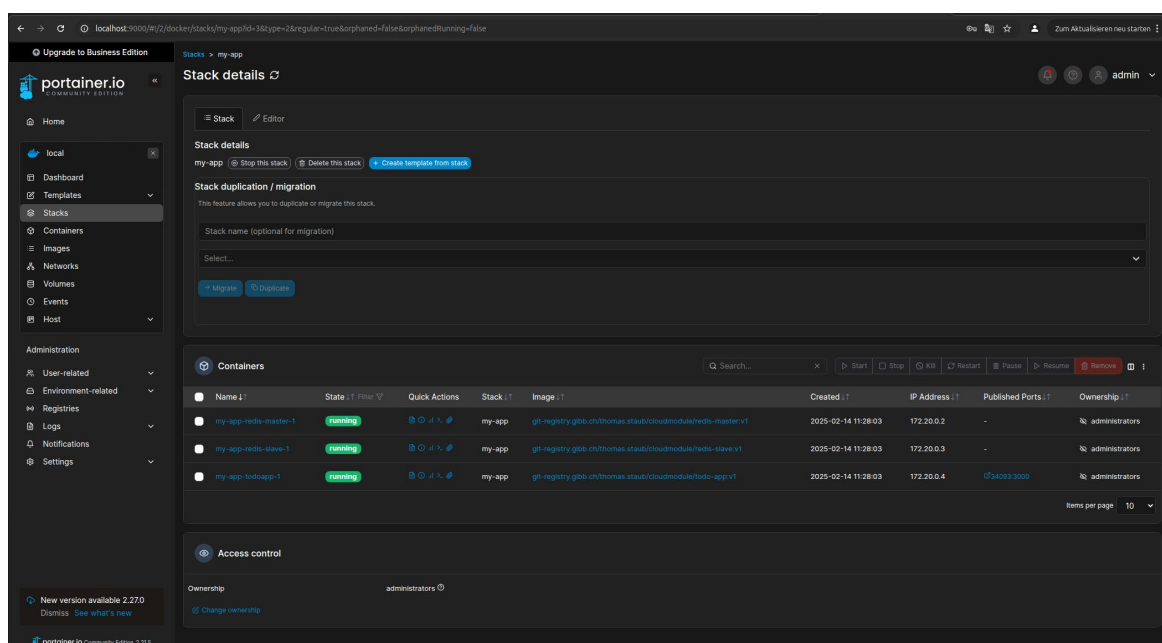


Abbildung 9: Stack Details

Vorgehen

Zuerst einloggen:

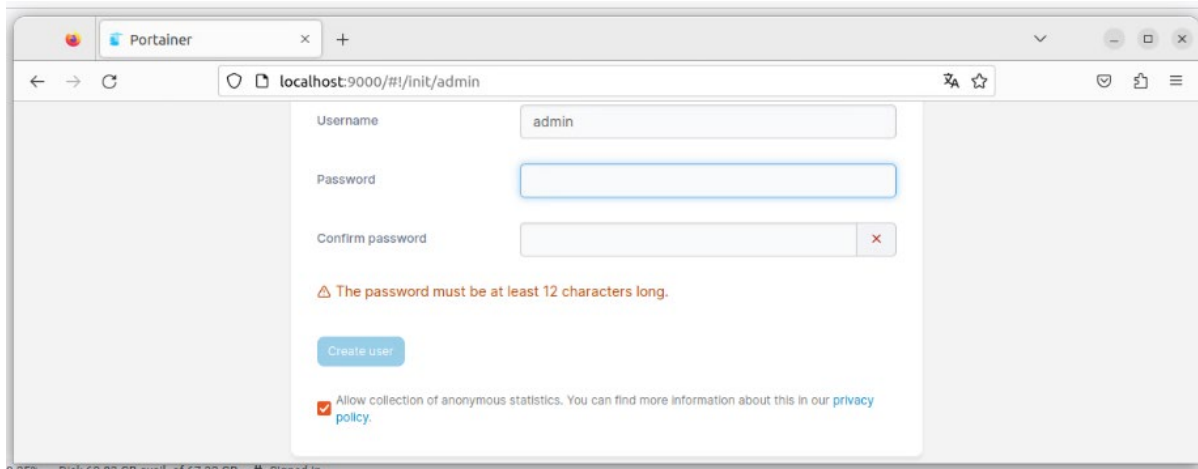


Abbildung 10: Einloggen bei Portainer

Unter «Stacks» auf Add Stack

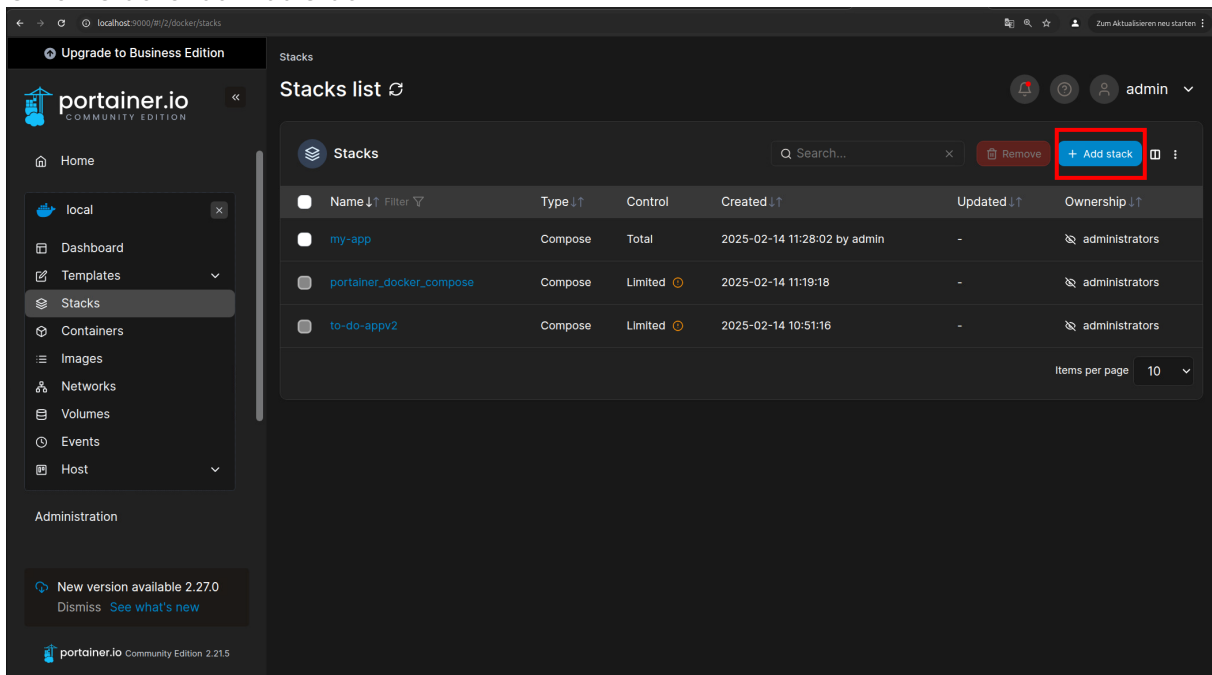


Abbildung 11: Neues Stack auf Portainer erstellen

Danach nehmen wir die Daten aus dem File von Thomas Staub, danach kann man das Stack erstellen.

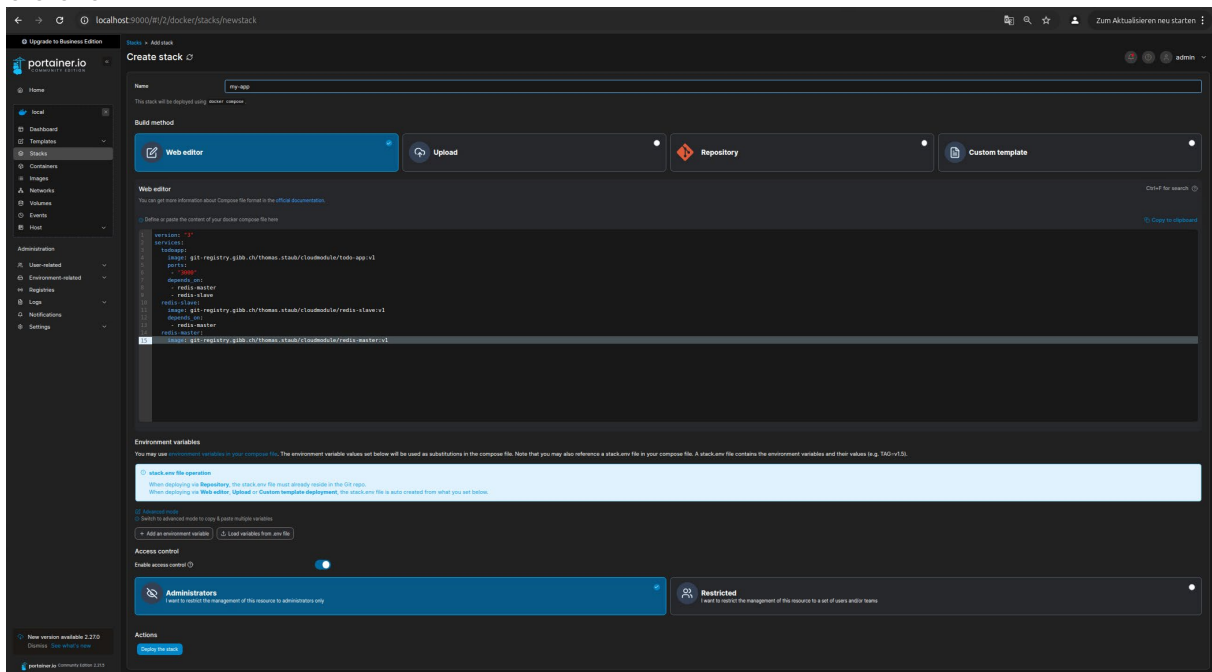


Abbildung 12: Skript von Thomas Staub verwenden

Danach sieht man es auch im Docker Desktop

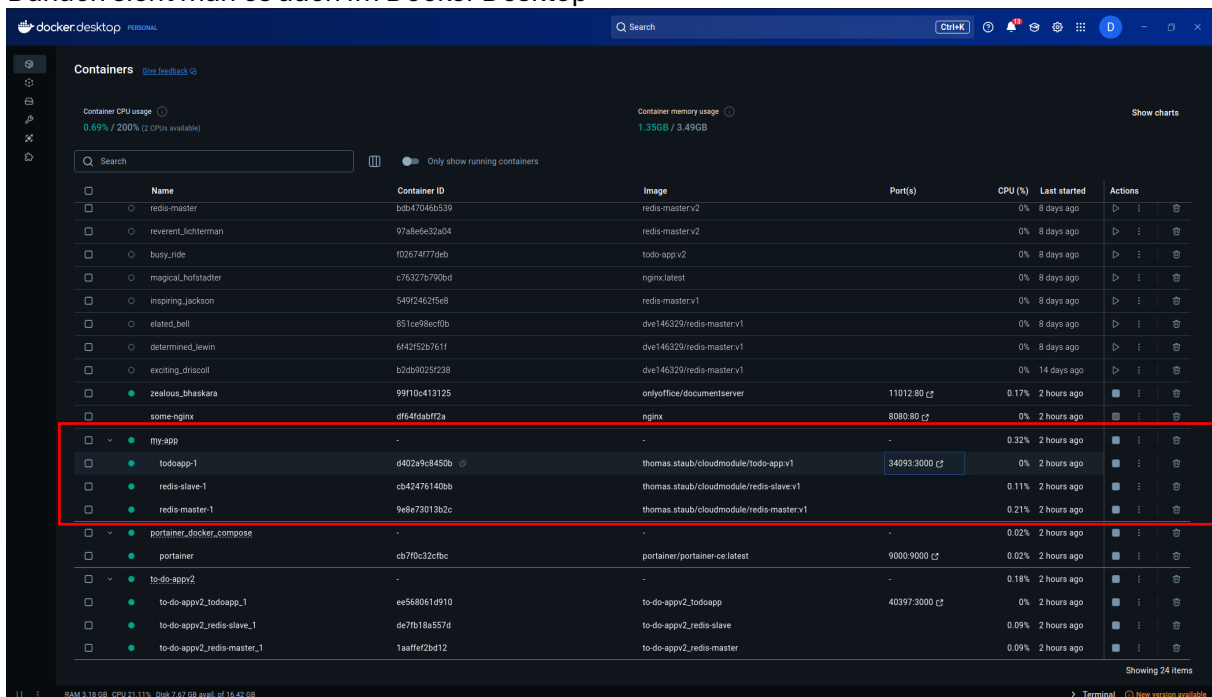
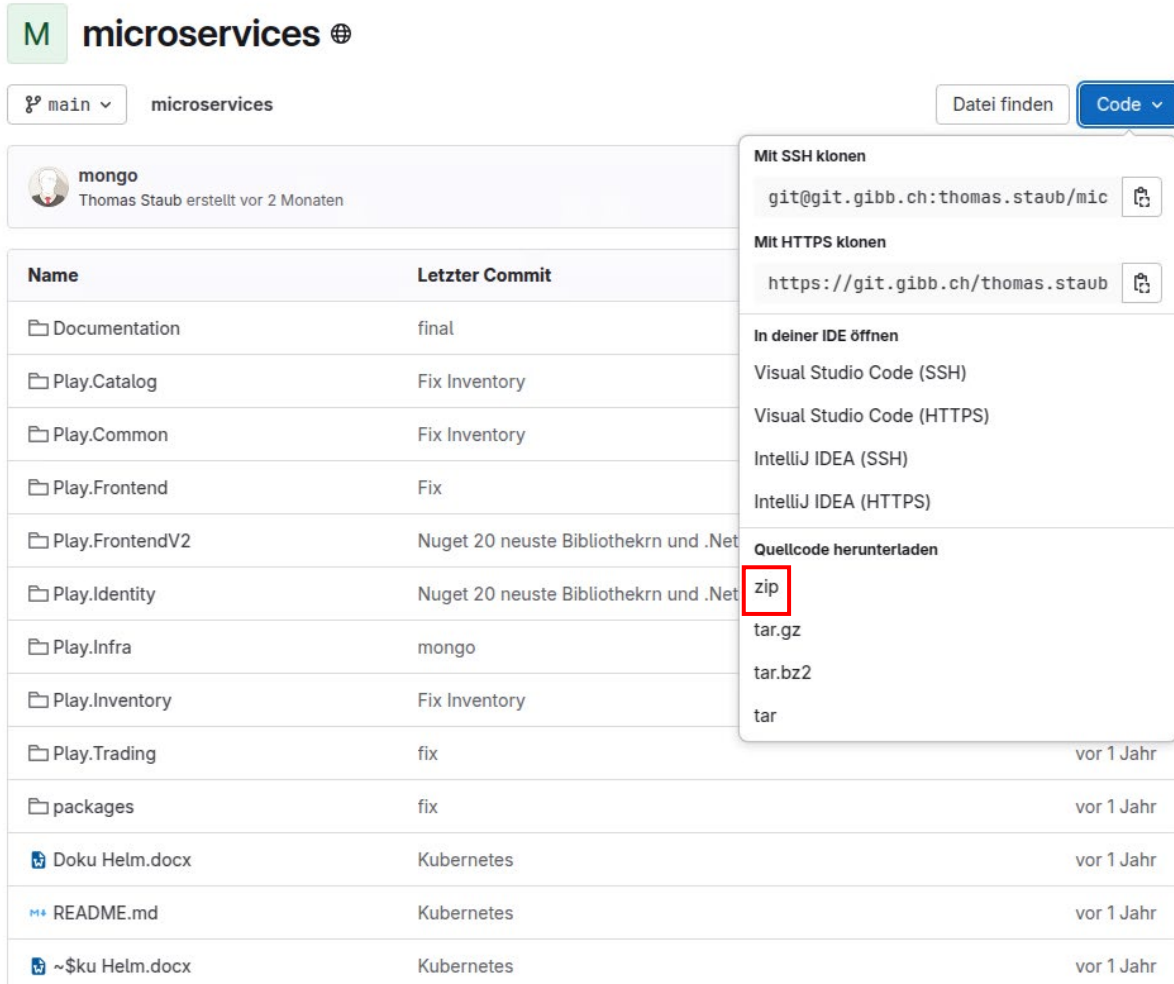


Abbildung 13: Docker Compose von Portainer

22.0 Learning Beispiel Shop

Vorgehen

Zuerst das Repo heruntergeladen.



microservices 🌐

main microservices

mongo
Thomas Staub erstellt vor 2 Monaten

Name	Letzter Commit	
Documentation	final	
Play.Catalog	Fix Inventory	
Play.Common	Fix Inventory	
Play.Frontend	Fix	
Play.FrontendV2	Nuget 20 neuste Bibliothekrn und .Net	
Play.Identity	Nuget 20 neuste Bibliothekrn und .Net	
Play.Infra	mongo	
Play.Inventory	Fix Inventory	
Play.Trading	fix	vor 1 Jahr
packages	fix	vor 1 Jahr
Doku Helm.docx	Kubernetes	vor 1 Jahr
README.md	Kubernetes	vor 1 Jahr
~\$ku Helm.docx	Kubernetes	vor 1 Jahr

Mit SSH klonen
git@git.gibb.ch:thomas.staub/mic

Mit HTTPS klonen
https://git.gibb.ch/thomas.staub

In deiner IDE öffnen
Visual Studio Code (SSH)
Visual Studio Code (HTTPS)
IntelliJ IDEA (SSH)
IntelliJ IDEA (HTTPS)

Quellcode herunterladen
zip
tar.gz
tar.bz2
tar

Abbildung 14: Thomas Staub's Repository

Danach entpackt:

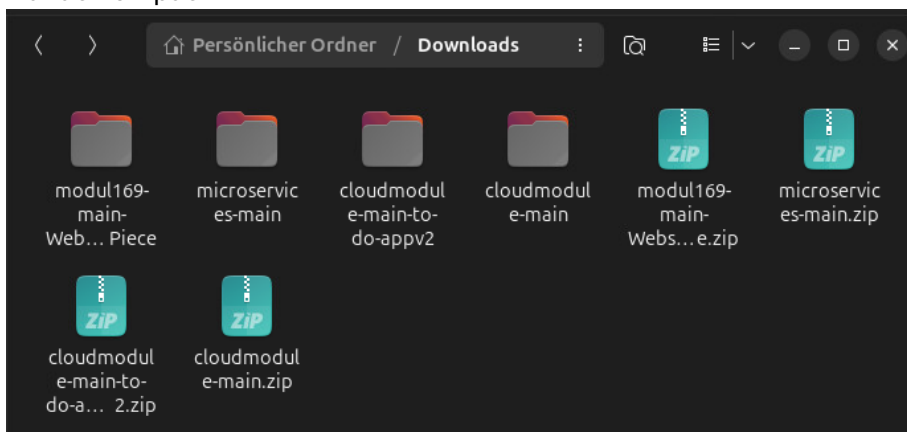


Abbildung 15: Downloads Ordner

In den Ordner wechseln:

```
vmadmin@li244-vmLP1:~$ cd Downloads/microservices-main/Play.Infra/docker/
vmadmin@li244-vmLP1:~/Downloads/microservices-main/Play.Infra/docker$
```

Abbildung 16: In den microservices Order wechseln

Und die .yaml Datei ausführen

```
vmadmin@li244-vmLP1:~/dockermodul/microservices-main/Play.Infra/docker$ docker-compose -f docker-compose.yaml up
```

Abbildung 17:yaml Datei ausführen

Danach sollten relativ viele Container gestartet werden

docker					4.6%	3 minutes ago			
frontend3	bb812e5443e8	thomas.staub/microservices/play.frontend.1.0	3002:3000 ↗	0.01%	3 minutes ago				
mongo-express	e6a0328fad41	mongo-express	8080:8081	0%	2 days ago				
grafana	f1ed9eae2eb	docker_grafana	4000:3000 ↗	0.03%	3 minutes ago				
rabbitmq	9f39f833bfca	rabbitmq.management	15672:15672 ↗ Show all ports (2)	1.05%	3 minutes ago				
prometheus	ae8480cb61f8	prom/prometheus	9090:9090 ↗	0.15%	3 minutes ago				
mongo	100a2c188d0	mongo	27017:27017 ↗	0.37%	3 minutes ago				
jaeger	2a97d6c4300f	jaegertracing/all-in-one	14250:14250 ↗ Show all ports (6)	0.01%	3 minutes ago				
seq	e9c1578bc44b	datalust/seq	5341:5341 ↗ Show all ports (2)	0.28%	3 minutes ago				
trading	26b83c626daa	thomas.staub/microservices/play.trading.1.0	5006:5006 ↗	0.06%	3 minutes ago				
catalog	4412d8c2f525	thomas.staub/microservices/play.catalog.1.0	5000:5000 ↗	0.62%	3 minutes ago				
frontendv2	cc0b0c8cfbb3	thomas.staub/microservices/play.frontendv2.1.0	5008:80 ↗	0%	3 minutes ago				
inventory	8c35a129fb1	thomas.staub/microservices/play.inventory.1.0	5004:5004 ↗	2.02%	3 minutes ago				
identity	0426e6d1bc95	thomas.staub/microservices/play.identity.1.0	5002:5002 ↗	0%	3 minutes ago				

Abbildung 18:Microservices via Docker Compose aufgesetzt

Sobald alle Container laufen, können Sie sich im Browser auf <http://host.docker.internal:5008/> verbinden.

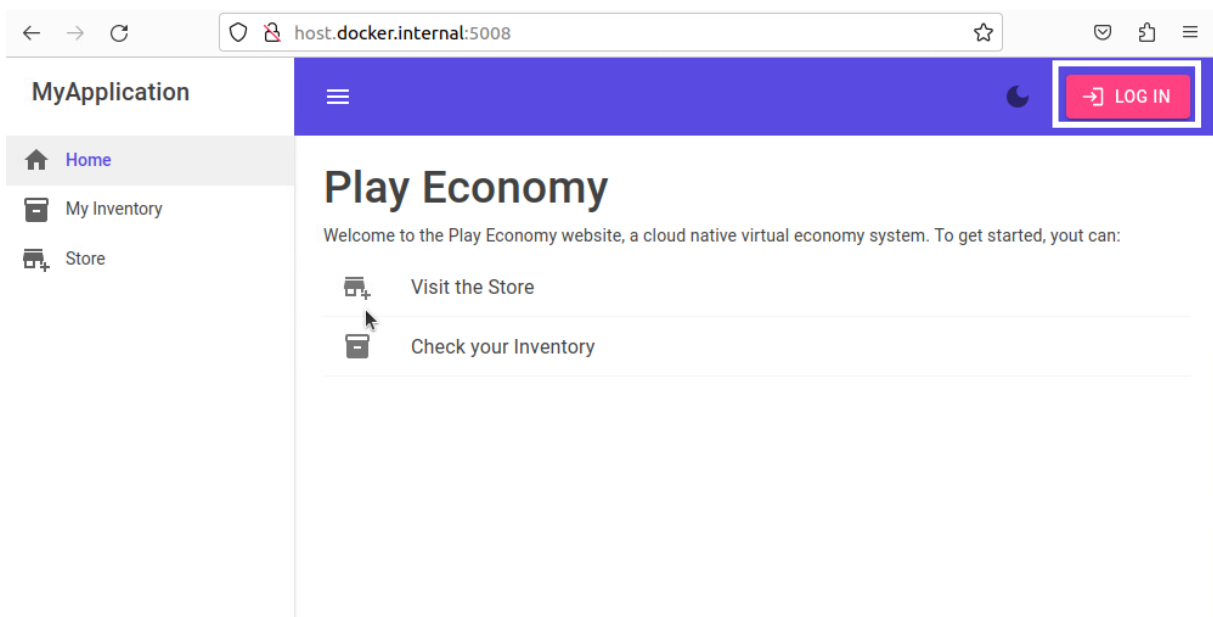


Abbildung 19: Beispiel Shop

Mit admin@play.com und Pass@word1 anmelden

Play.Identity.Service

Register Login

Log in

Use a local account to log in.

☐ Remember me?

Log in

[Forgot your password?](#)
[Register as a new user](#)
[Resend email confirmation](#)

Use another service to log in.

There are no external authentication services configured. See [this article about setting up this ASP.NET application to support logging in via external services.](#)

Abbildung 20: Anmelden

Shop ist noch leer, jetzt werden wir zwei Produkte erfassen

MyApplication

Home

My Inventory

Store

Users

Catalog

Catalog

ADD

Id	Name	Description	Price	Created Date	Actions
949e0db2-169e-4288-bafe-50e9448a31bc	Docker Kurs 1	Einfacher Docker Kurs	5	21.02.2025 09:54:29 +00:00	<div>EDIT</div> <div>GRANT</div> <div>DELETE</div>
951e2719-6e53-4132-b57b-1723c098a0e2	Kubernetes Kurs	Einfacher Kubernetes Kurs	10	21.02.2025 09:54:57 +00:00	<div>EDIT</div> <div>GRANT</div> <div>DELETE</div>

Abbildung 21: Katalog von Beispielshop

Unter Users können wir uns virtuelles Geld «Gil» geben

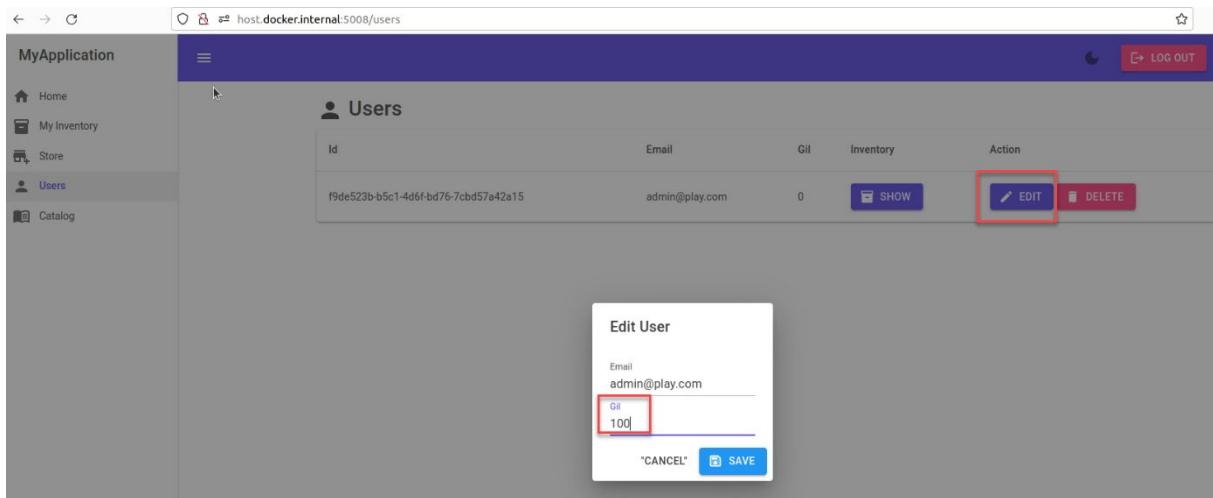


Abbildung 22: virtuelles Geld hinzufügen

Dann kann man mal eines kaufen

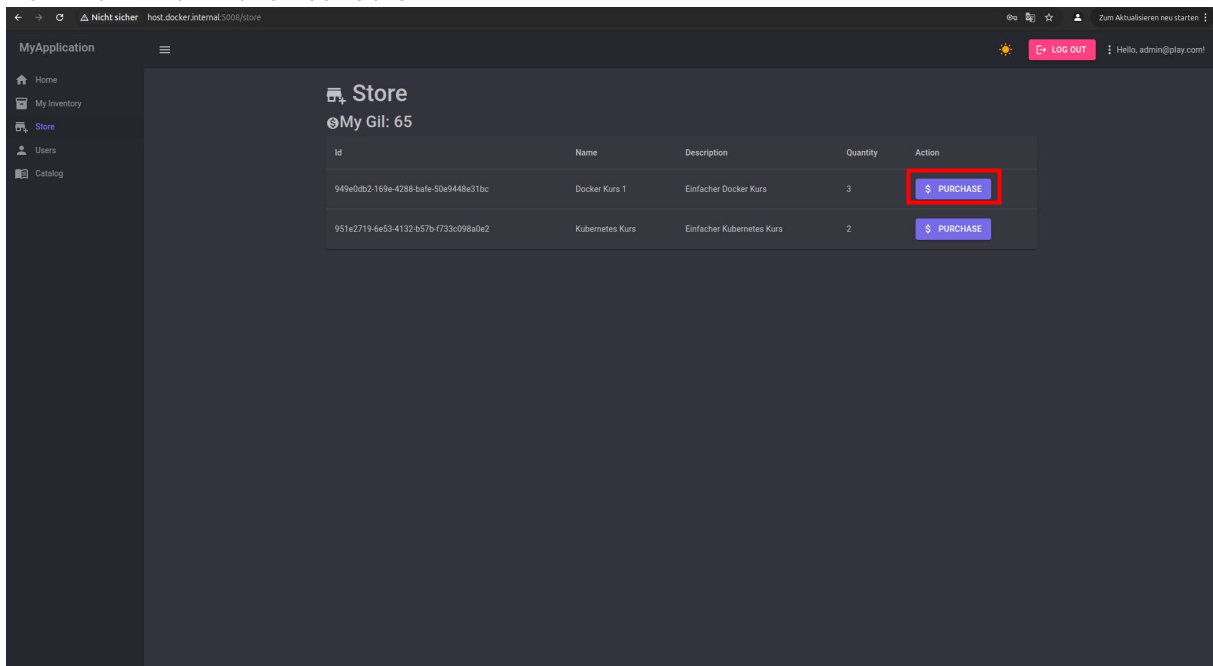
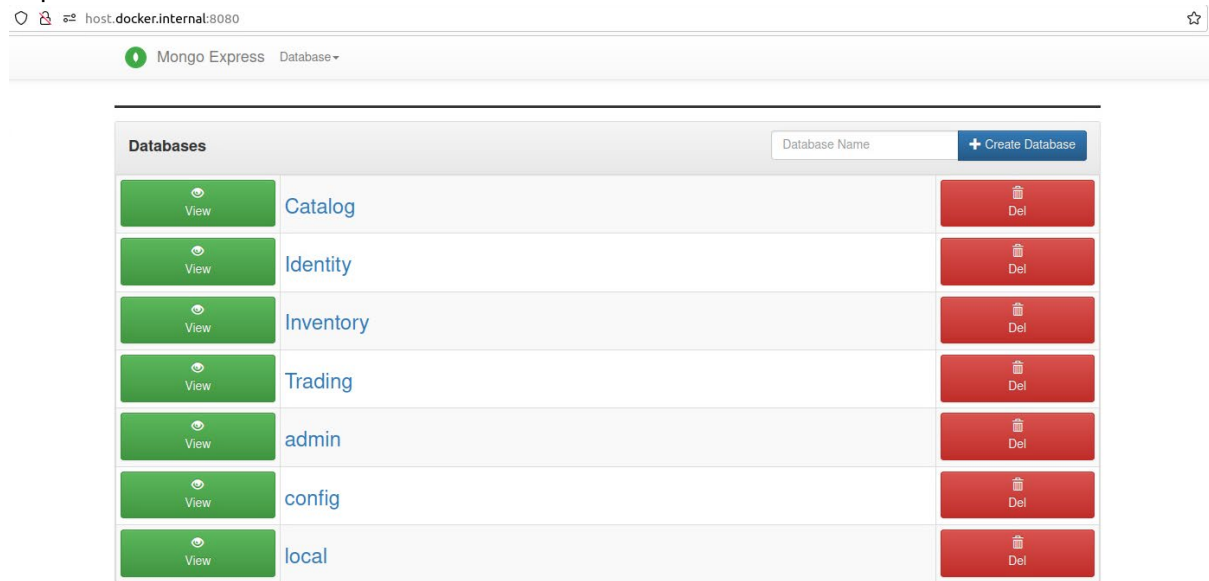


Abbildung 23: Store von Shop

Zusätzliche Services

Im Browser können wir nun zu <http://host.docker.internal:8080/> und finden dort unseren Mongo Express vor.



Server Status

Turn on admin in config.js to view server stats!

Abbildung 24: Mongo Express

Unter Catalog finden wir auch unsere gespeicherten Kurse

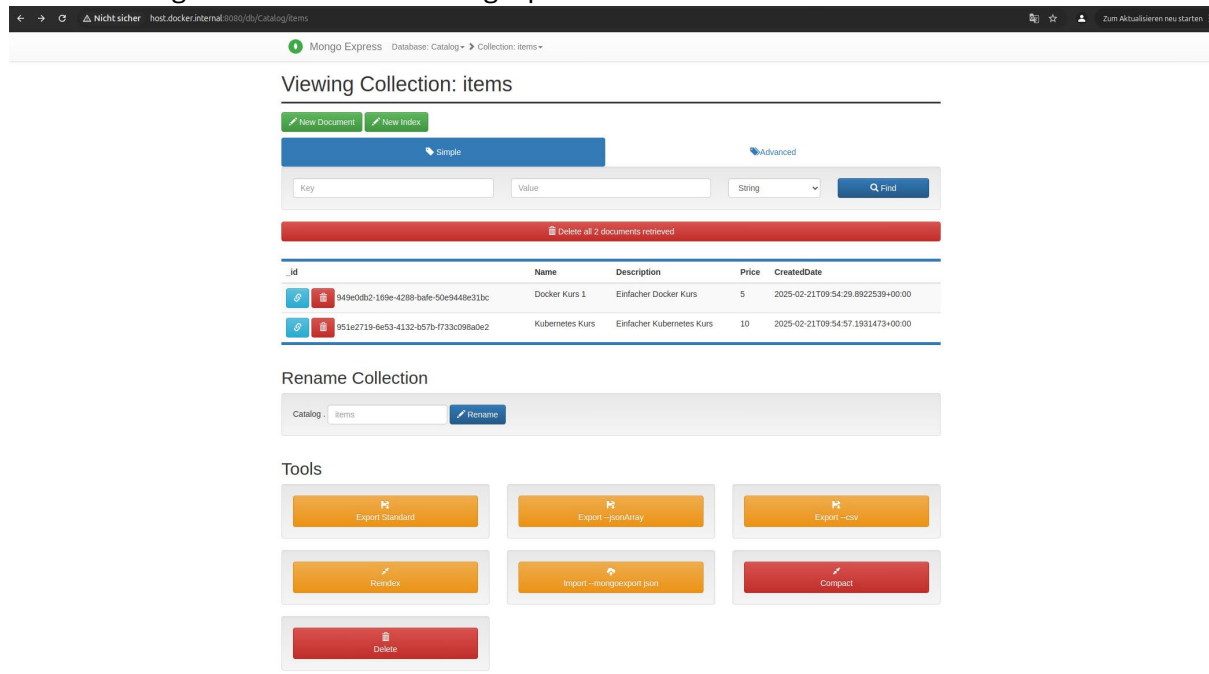


Abbildung 25: Mongo Express Catalog

Seq hilft uns, die Logs der verschiedenen Services zentral an einem Ort zu sehen

<http://host.docker.internal:8085>

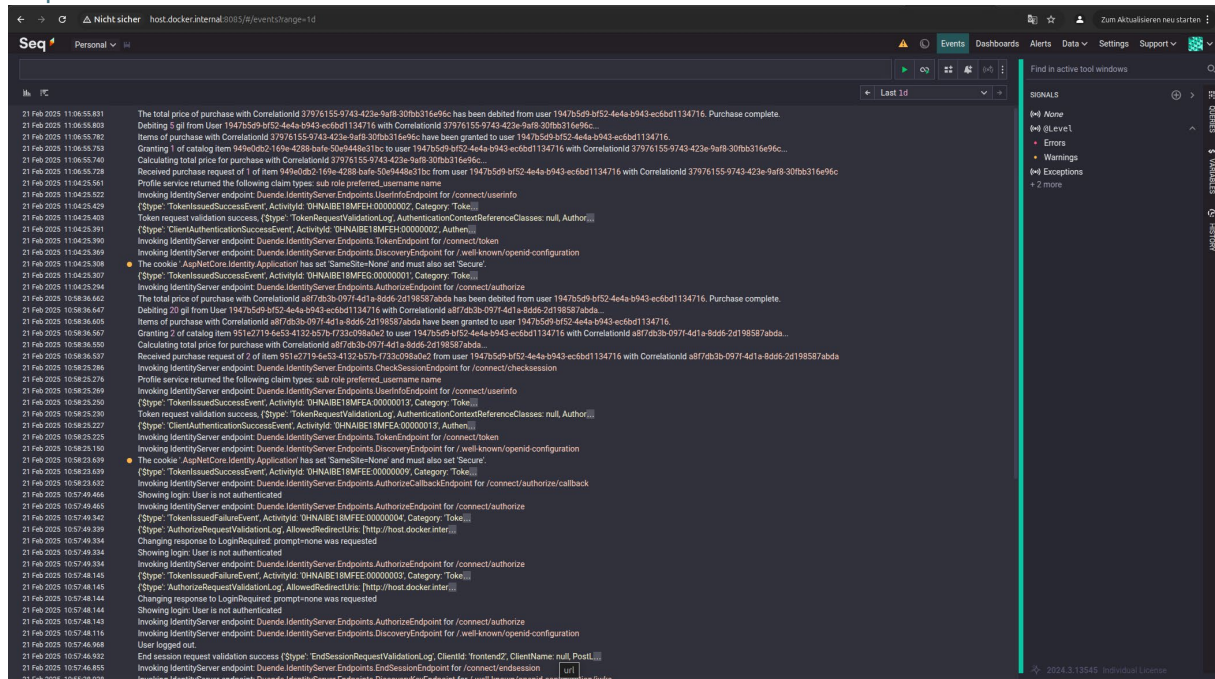


Abbildung 26: Events Webseite

Prometheus

Für was benötigen wir Prometheus?

Wir verwenden es, um numerische Metriken von Diensten zu sammeln, die rund um die Uhr laufen und den Zugriff auf Metrik Daten über http Endpunkte ermöglichen

Unter Status → Target Health finden wir unseres Services

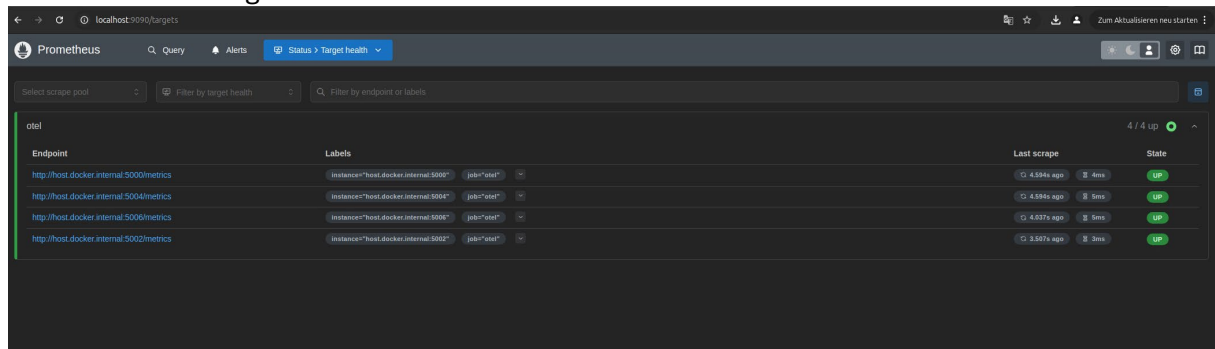


Abbildung 27: Prometheus Targets

Unter Query → PurchaseStarted_total, sehen wir wie sich der Graph verändert wenn jemand ein Kauf tätigt.

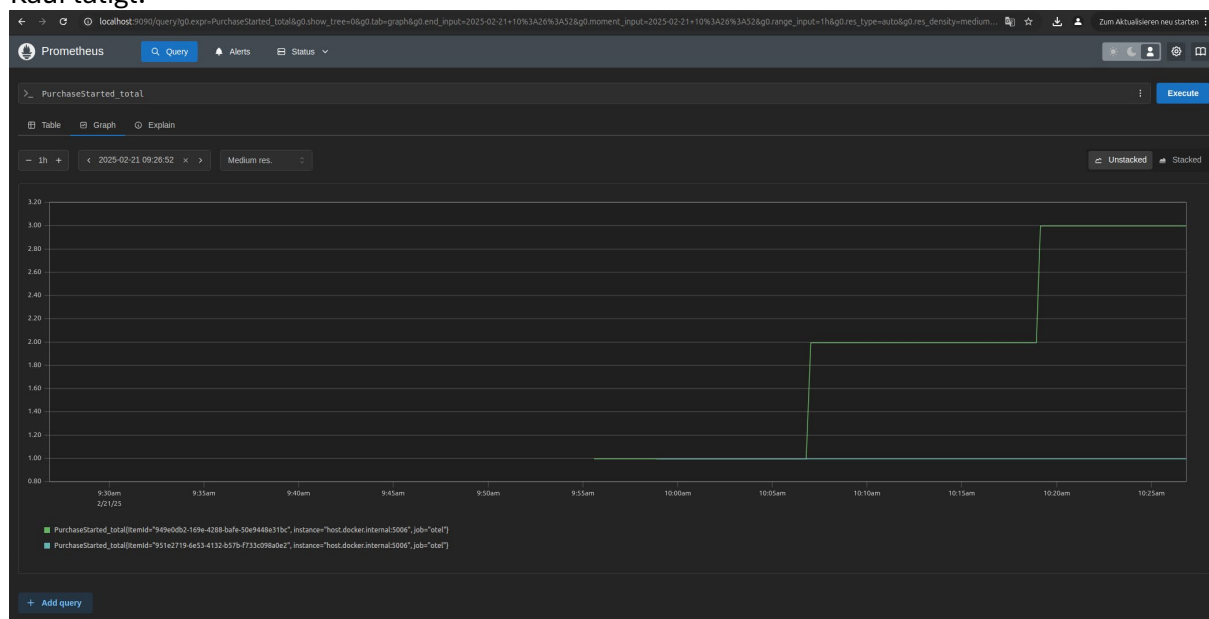


Abbildung 28: Prometheus PurchaseStarted Graph

Falls ein Service down geht sieht man das auch auf Prometheus

The figure shows the Prometheus Targets page. At the top, there is a search bar labeled 'Filter by endpoint or labels'. Below the search bar, there is a table with columns: Endpoint, State, Labels, Last Scrape, Scrape Duration, and Error. The table lists four targets. The first three targets are in the 'UP' state. The fourth target, `http://host.docker.internal:5000/metrics`, is in the 'DOWN' state with an error message: `Get "http://host.docker.internal:5000/metrics": dial tcp 192.168.65.2:5000: connect: connection refused`.

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
<code>http://host.docker.internal:5002/metrics</code>	UP	<code>instance="host.docker.internal:5002" job="otel"</code>	843.000ms ago	7.392ms	
<code>http://host.docker.internal:5000/metrics</code>	DOWN	<code>instance="host.docker.internal:5000" job="otel"</code>	1.846s ago	6.332ms	Get "http://host.docker.internal:5000/metrics": dial tcp 192.168.65.2:5000: connect: connection refused
<code>http://host.docker.internal:5004/metrics</code>	UP	<code>instance="host.docker.internal:5004" job="otel"</code>	1.50s ago	6.930ms	
<code>http://host.docker.internal:5006/metrics</code>	UP	<code>instance="host.docker.internal:5006" job="otel"</code>	5.71s ago	8.318ms	

Grafana

Ein grosses Dashboard auf dem angezeigt wird, was in unserem Shop passiert.

<http://host.docker.internal:4000> einloggen mit: admin, admin

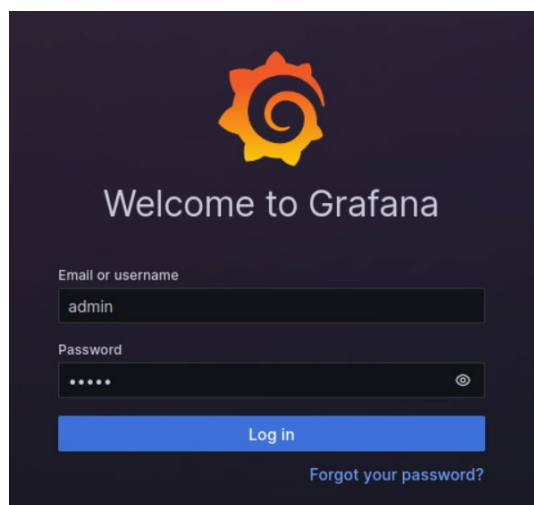
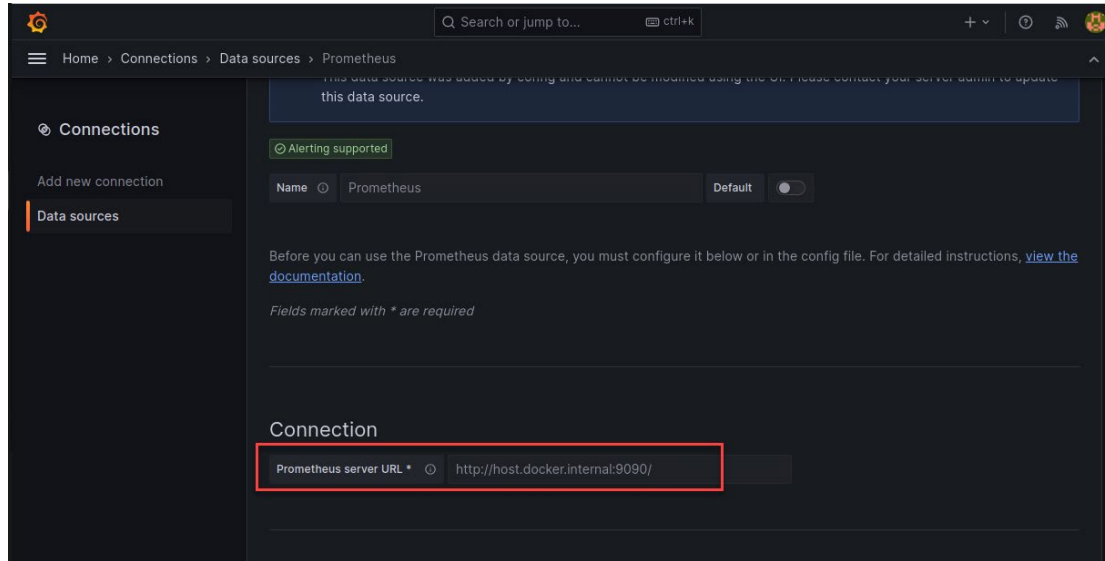
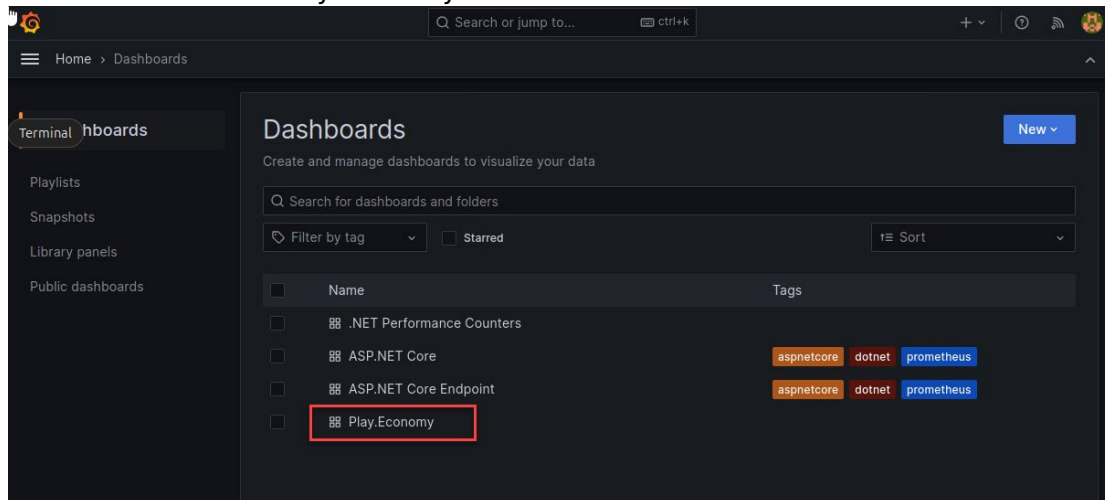


Abbildung 29: Startseite von Grafana

Unter Connections → Data sources können wir Prometheus verbinden



Unter Dashboard auf Play.Economy



Und schon können wir sehen was in unserem Shop passiert.

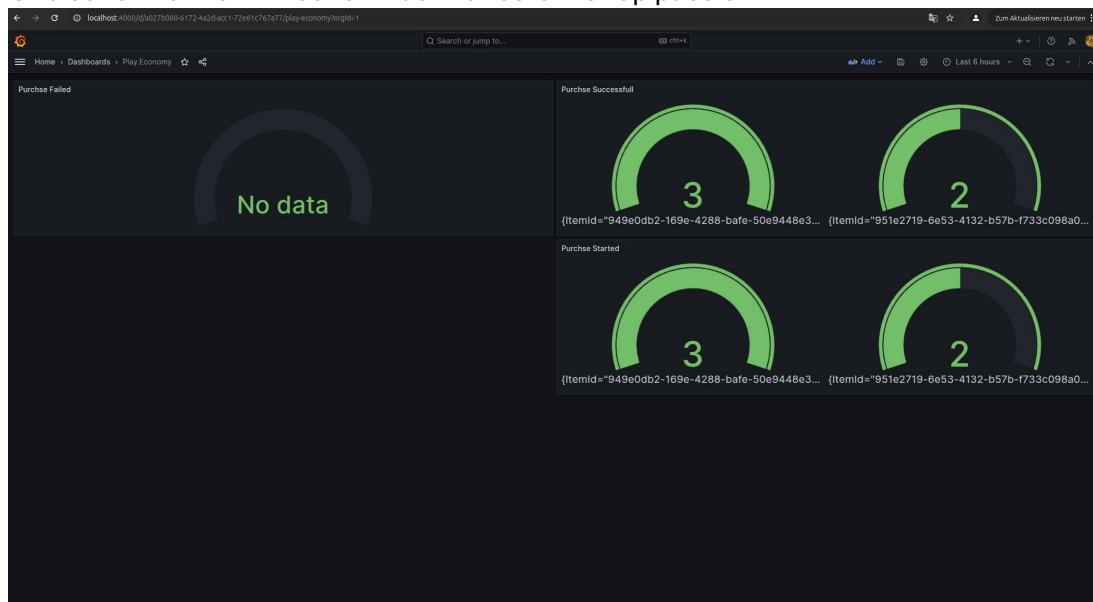
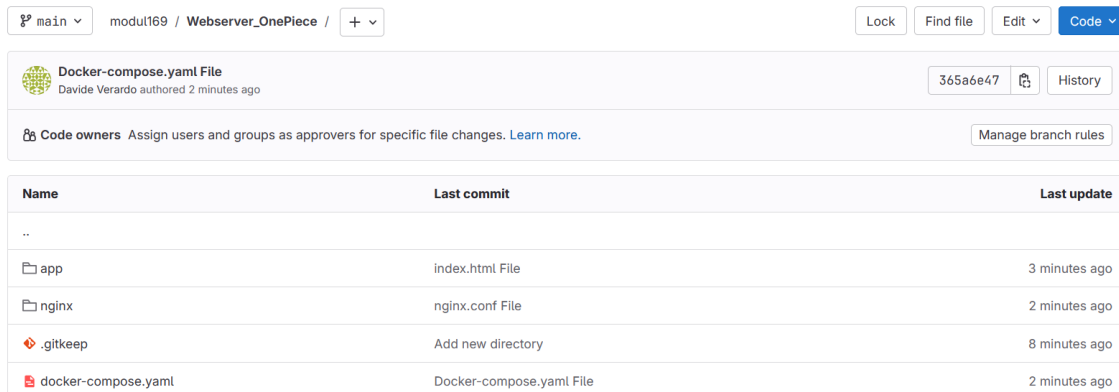


Abbildung 30: Grafana Dashboard vom Shop

23.0 Eigenes Projekt: Webserver

Das ganze Projekt findet man unter folgendem Link: <https://git.gibb.ch/dve146329/modul169.git>



Name	Last commit	Last update
..		
app	index.html File	3 minutes ago
nginx	nginx.conf File	2 minutes ago
.gitkeep	Add new directory	8 minutes ago
docker-compose.yaml	Docker-compose.yaml File	2 minutes ago

Abbildung 31: GitLab von Webserver

Dockerfile

Zuerst habe ich die "index.html" Datei erstellt und anschliessend habe ich den dazugehörigen "Dockerfile" erstellt:

```
FROM nginx:latest
COPY ./index.html /usr/share/nginx/html/index.html
```

Docker Compose Datei

Mein "docker-compose.yaml" Datei sieht so aus:

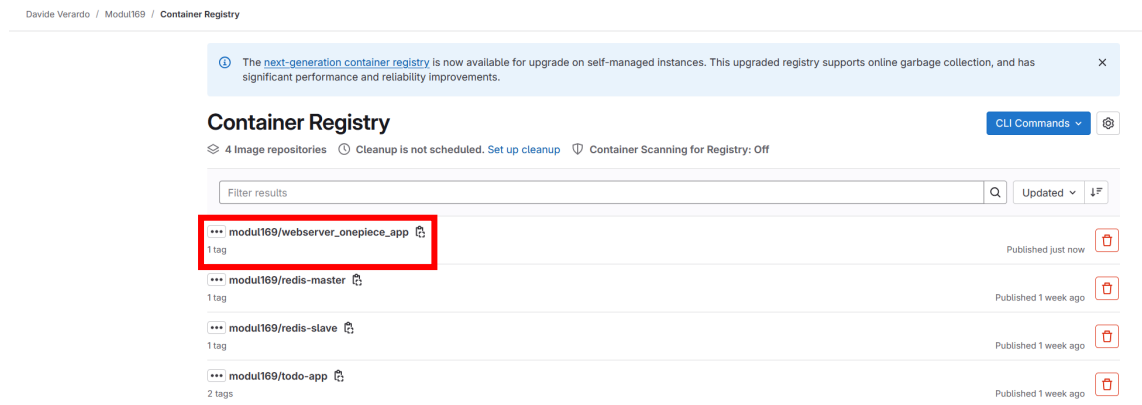
```
version: '3'
services:
  nginx:
    image: jwilder/nginx-proxy:latest
    ports:
      - "3080:80"
    volumes:
      - /var/run/docker.sock:/tmp/docker.sock:ro
    environment:
      - DEFAULT_HOST=davide.example

  app:
    build: ./app
    ports:
      - "8080:80"
    environment:
      - VIRTUAL_HOST=davide.example

  restart: always
```

Abbildung 32: Docker Compose Datei für Webserver

PrintScreen des Images bei (gibb) GitLab



Anleitung

Zuerst den Projekt Ordner klonen

```
git clone https://git.gibb.ch/dve146329/modul169.git
```

Als nächstes ins Ordner wechseln

```
cd modul169/Webserver_OnePiece
```

Danach die Container mit der "docker-compose.yaml" Datei starten

```
docker-compose -f docker-compose.yaml up -d
```

Und schon kann man im Webbrowser den folgenden Link eingeben

<http://localhost:3080/>

Resultat

Wenn alles funktioniert hat, sollte folgende Webseite erscheinen.



Abbildung 33: Webserver Webseite