



---

# Modul 169

Modul	IET-169
Eingereicht von	Davide
Eingereicht bei	Reto Brüttsch
Datum	30. März 2025

# 1.0 Versionskontrolle

## Versionskontrolle

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

			Das Image pushen Sie in das Repository (Printscreen) Anleitung, wie das Image nun von jemand anderem (Lehrperson) installiert werden kann
0.0.5	28.02.2025	Davide	Kurze Beschreibung was Kubernetes ist. Kurze Beschreibung was Microservices sind. Vergleich der lightweight Kubernetes Anwendungen. (4 Stück) Sie haben Kubernetes auf ihrem Rechner oder Lernumgebung installiert. Und eine Anleitung erstellt. PrintScreen, Sie können sich mit Lens auf den Server verbinden.
0.0.6	07.03.2025	Davide	Was ist der Raft-Konsens-Algorithmus, Festhalten wie die App in Kubernetes läuft, Self Healing, Scale Down, Scale up, Printscreen von Rolling Updates Version 2, Eintrag zu Blue Green Deployment
0.0.7	14.03.2025	Davide	Eintrag zu Cluster IP und Node IP, Eintrag zu Loadbalancer, Printscreen wie auf App zugreifen (Ingress), Erklärung warum bei Ingress beim Zugriff auf 127.0.0.1 ein Error 404 erhalten, Printscreen wie Portainer auf Kubernetes installiert ist
0.0.8	21.03.2025	Davide	Eigenes Projekt auf Kubernetes installiert und Dokumentiert (Webserver)
0.0.9	28.03.2025	Davide	Todoapp auf podman zum laufen gekriegt und Dokumentiert.

## 2.0 Inhaltsverzeichnis

1.0	Versionskontrolle .....	2
2.0	Inhaltsverzeichnis .....	4
3.0	Abbildungsverzeichnis.....	5
4.0	Was sind Container? .....	7
5.0	Wo sind Container nützlich? .....	7
6.0	Was ist DevOps? .....	7
7.0	Was ist der Unterschied von Virtualisierung und Containerisierung? .....	7
8.0	Was ist der Unterschied von Image und Conatiner? .....	8
9.0	Zusammenfassung wichtiger Docker Kommandos bei Containern: .....	8
10.0	Zusammenfassung wichtiger Docker Kommandos bei Images:.....	9
11.0	Zusammenfassung wichtiger genereller Docker Kommandos: .....	9
12.0	PrintScreen OnlyOffice mit eigenem Namen und Datum: .....	10
13.0	PrintScreen der Version 1 mit eigenem Namen als Todo Task.....	10
14.0	PrintScreen der Images bei (gibb) GitLab .....	11
15.0	Wichtigste Befehle, um ein Image zu pushen: .....	11
16.0	PrintScreen der Version 2 mit eigenem Namen als Todo Task.....	12
17.0	PrintScreen der Images bei (gibb) GitLab .....	13
18.0	Was ist Docker Compose .....	13
19.0	Version 2 der App mit Docker Compose zum laufen bringen .....	14
20.0	Vorgehen und Befehle und Compose Datei .....	14
	Compose Datei .....	14
	Vorgehen .....	15
	Wichtige Befehle:.....	15
21.0	Portainer PrintScreen .....	17
	Vorgehen .....	17
22.0	Learning Beispiel Shop .....	19
	Vorgehen .....	19
	Zusätzliche Services .....	23
	Prometheus .....	25
	Für was benötigen wir Prometheus? .....	25
	Grafana.....	26
23.0	Eigenes Projekt: Webserver .....	28
	Dockerfile .....	28
	Docker Compose Datei .....	28

PrintScreen des Images bei (gibb) GitLab .....	29
Anleitung .....	29
Resultat.....	29
24.0 Was ist Kubernetes.....	30
25.0 Was sind Microservices .....	30
26.0 Vergleich der lightweight Kubernetes Anwendungen (4 Stück) .....	30
Mks .....	30
Micro8s.....	30
Kind.....	30
MiniKube.....	30
Vergleich.....	31
27.0 Kubernetes Installation Anleitung .....	31
28.0 Printscreen von Verbindung von Lens zu Server.....	32
29.0 Was ist der Raft-Konsens-Algorithmus .....	32
30.0 Festhalten wie die App in Kubernetes läuft .....	33
Self Healing.....	33
Scale Up .....	33
Scale Down.....	33
31.0 PrintScreen das das Rolling Update funktioniert. Version 2 ist im Dashboard ersichtlich ....	34
Blue Green Deployment .....	34
Cluster IP.....	35
Node IP.....	35
LoadBalancer.....	35
32.0 PrintScreen Wie Sie auf die App zugreifen .....	36
33.0 Warum bei Ingress beim Zugriff auf 127.0.0.1 ein Error 404 erhalten.....	36
34.0 PrintScreen wie Portainer auf Kubernetes installiert ist .....	37
35.0 Kubernetes Webserver.....	37
36.0 todo App auf Podman zum laufen gekriegt und Dokumentiert. ....	38

### 3.0 Abbildungsverzeichnis

Abbildung 1: OnlyOffice Screenshot.....	10
Abbildung 2: PrintScreen der 1. Version.....	10
Abbildung 3: PrintScreen der Images bei GitLab .....	11
Abbildung 4: PrintScreen der 2. Version.....	12

Abbildung 5: PrintScreen der Images bei GitLab mit 2 Tags bei Todo-App .....	13
Abbildung 6: Docker Compose .....	14
Abbildung 7: Compose Datei für Todo_appv2 .....	14
Abbildung 8: Portainer erstelltes Environment.....	17
Abbildung 9: Stack Details .....	17
Abbildung 10: Einloggen bei Portainer .....	17
Abbildung 11: Neues Stack auf Portainer erstellen.....	18
Abbildung 12: Skript von Thomas Staub verwenden .....	18
Abbildung 13: Docker Compose von Portainer .....	19
Abbildung 14: Thomas Staub's Repository .....	19
Abbildung 15: Downloads Ordner .....	20
Abbildung 16: In den microservices Order wechseln.....	20
Abbildung 17:yaml Datei ausführen.....	20
Abbildung 18:Microservices via Docker Compose aufgesetzt .....	20
Abbildung 19: Beispiel Shop .....	21
Abbildung 20:Anmelden.....	21
Abbildung 21: Katalog von Beispielshop .....	22
Abbildung 22: virtuelles Geld hinzufügen .....	22
Abbildung 23: Store von Shop .....	23
Abbildung 24: Mongo Express.....	23
Abbildung 25: Mongo Express Catalog.....	24
Abbildung 26: Events Webseite.....	24
Abbildung 27: Prometheus Targets.....	25
Abbildung 28: Prometheus PurchaseStarted Graph .....	25
Abbildung 29: Prometheus Service Down .....	25
Abbildung 30: Startseite von Grafana .....	26
Abbildung 31: Grafana Dashboard vom Shop .....	27
Abbildung 32: GitLab von Webserver.....	28
Abbildung 33: Docker Compose Datei für Webserver .....	28
Abbildung 34: Webserver Webseite.....	29
Abbildung 35: Einstellungen Docker.....	31
Abbildung 36: kubectl installieren .....	31
Abbildung 37:Docker Desktop info bekommen.....	32
Abbildung 38: Mit Docker Desktop verknüpfen .....	32
Abbildung 39:Workloads.....	32
Abbildung 40: Todo App via Kubernetes .....	33
Abbildung 41: Rolling Updates.....	34
Abbildung 42: Todo App Ingress .....	36
Abbildung 43: Todo App Ingress .....	36
Abbildung 44: Portainer auf Kubernetes.....	37
Abbildung 45: Solo-Leveling Webseite.....	38
Abbildung 46: Dockerfiles bearbeiten.....	39
Abbildung 47: podman ps Befehl .....	39
Abbildung 48: Podman Desktop Container Ansicht.....	40
Abbildung 49: Podman Desktop Images Ansicht.....	40
Abbildung 50: Podman Todo App starten .....	40

## 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 Container?

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 push [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:

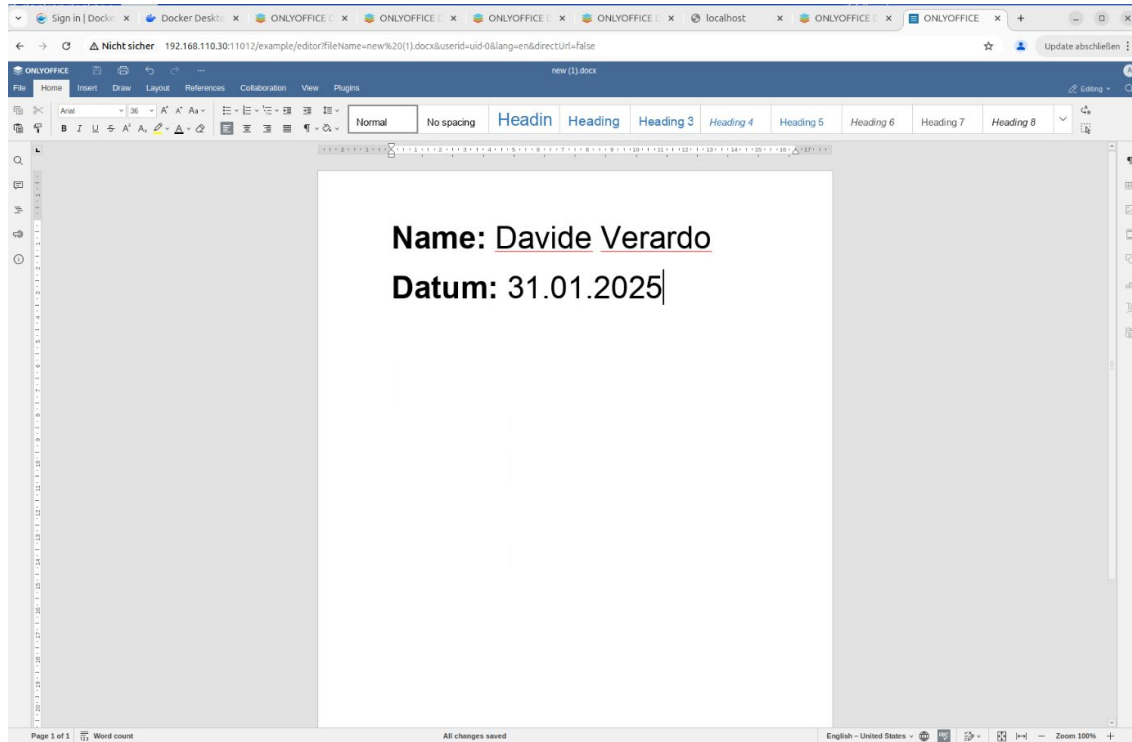


Abbildung 1: OnlyOffice Screenshot

## 13.0 PrintScreen der Version 1 mit eigenem Namen als Todo Task

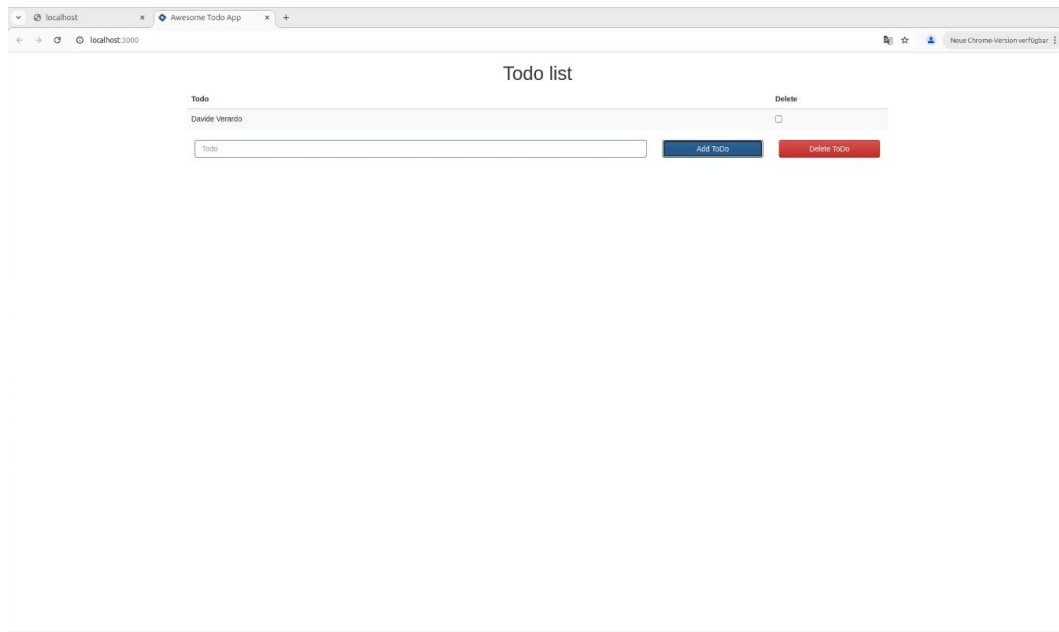


Abbildung 2: PrintScreen der 1. Version

## 14.0 PrintScreen der Images bei (gibb) GitLab

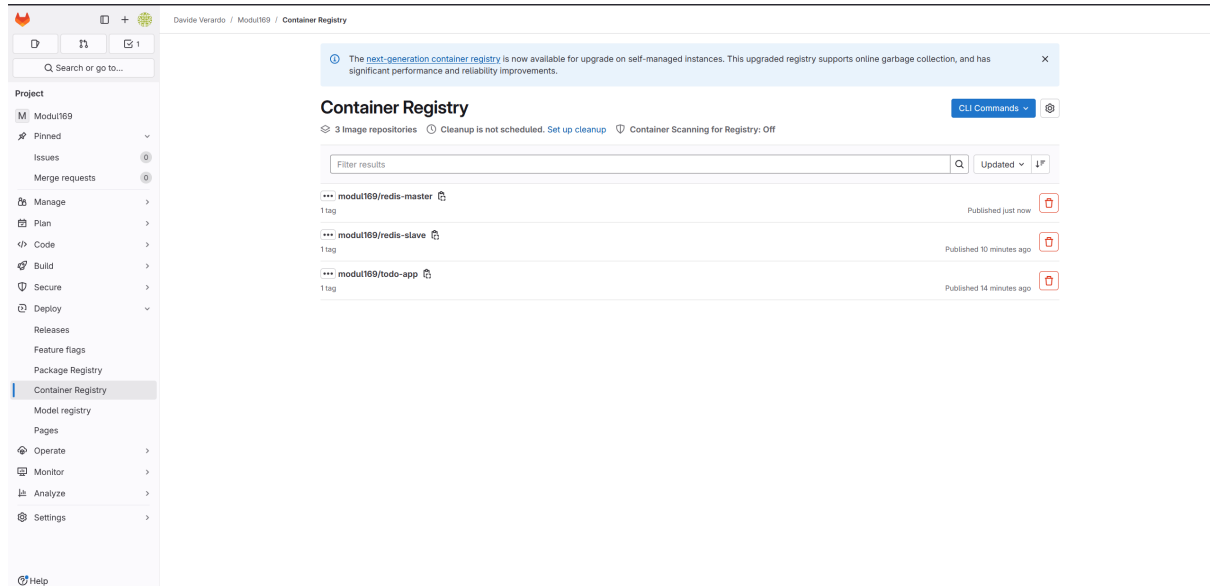


Abbildung 3: PrintScreen der Images bei GitLab

## 15.0 Wichtigste Befehle, um ein Image zu pushen:

- `docker login git-registry.gibb.ch`
- `docker 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`
- `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 push git-registry.gibb.ch/dve146329/modul169/redis-master:v2`
- `docker push git-registry.gibb.ch/dve146329/modul169/redis-slave:v2`
- `docker push git-registry.gibb.ch/dve146329/modul169/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: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`
- `docker build -t webserver_onepiece_app:latest ./app/ --provenance false`
- `docker image tag webserver_onepiece_app:latest git-registry.gibb.ch/dve146329/modul169/webserver_onepiece_app:latest`
- `docker push git-registry.gibb.ch/dve146329/modul169/webserver_onepiece_app:latest`

## 16.0 PrintScreen der Version 2 mit eigenem Namen als Todo Task

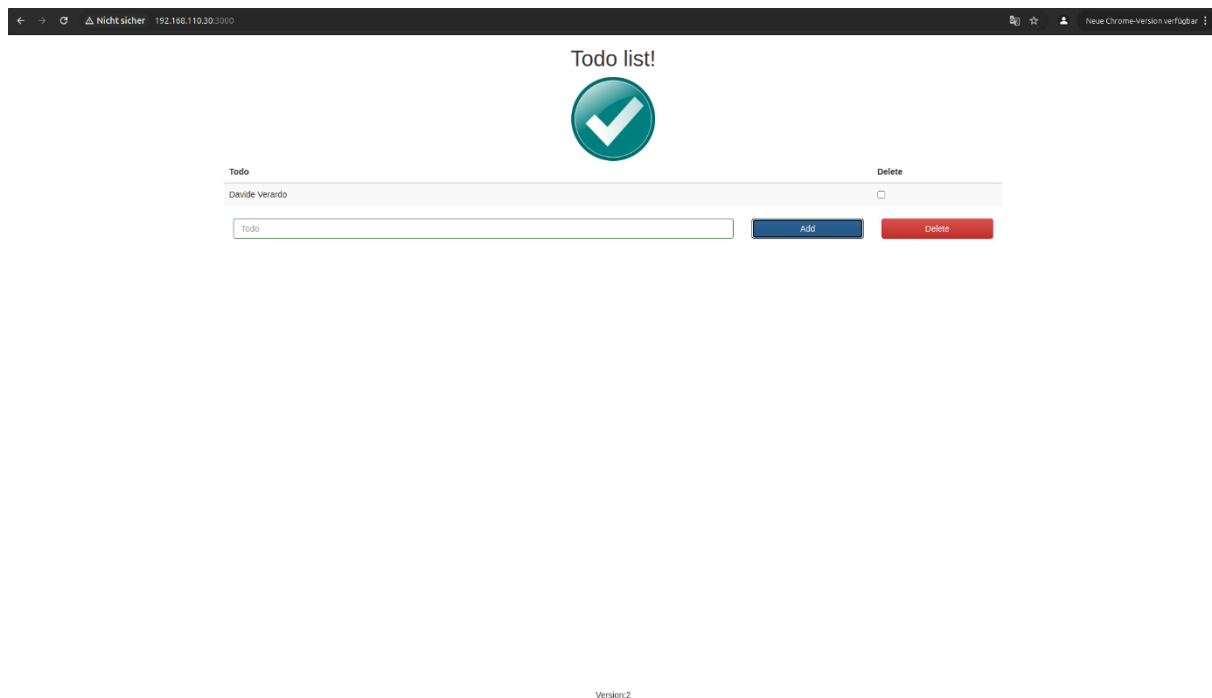


Abbildung 4: PrintScreen der 2. Version

## 17.0 PrintScreen der Images bei (gibb) GitLab

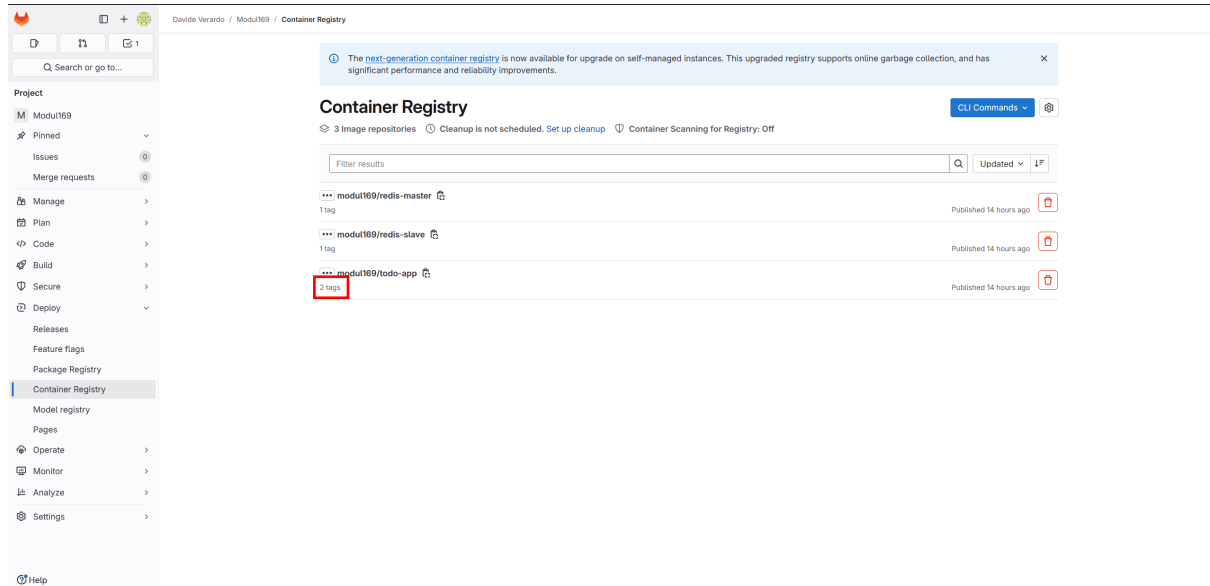


Abbildung 5: PrintScreen der Images bei GitLab mit 2 Tags bei Todo-App

## 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

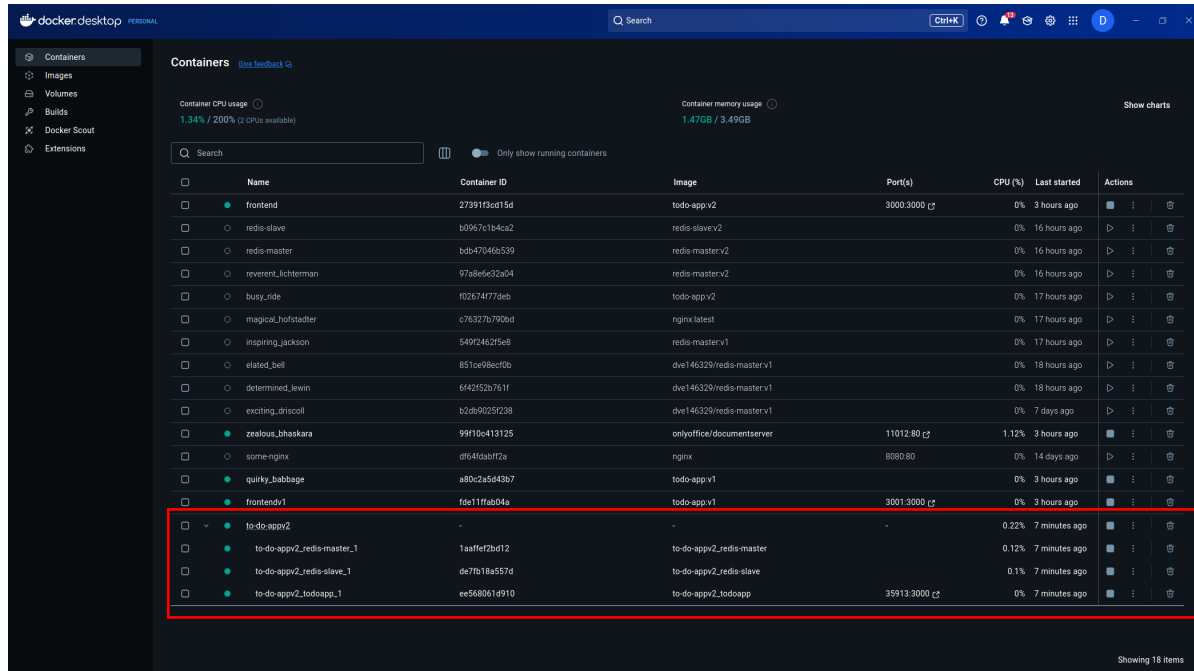


Abbildung 6: Docker Compose

## 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-frontendl
    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.yaml
```

### **"docker-compose.yml" ausführen**

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

## Wichtige Befehle:

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

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

### **"docker-compose.yml" ausführen**

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

### **"docker-compose.yml" stoppen und Container entfernen**

- ```
sudo docker-compose - /home/vmadmin/Downloads/cloudmodule-main-to-do-appv2/to-do-  
appv2/docker-compose.yaml 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

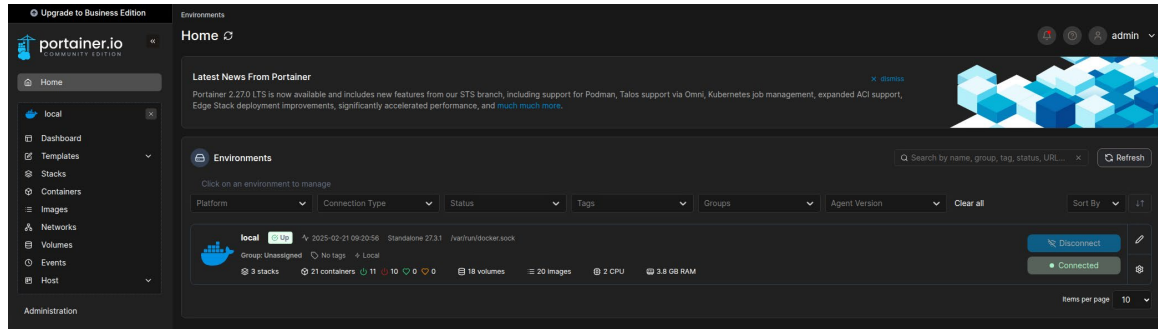


Abbildung 8: Portainer erstelltes Environment

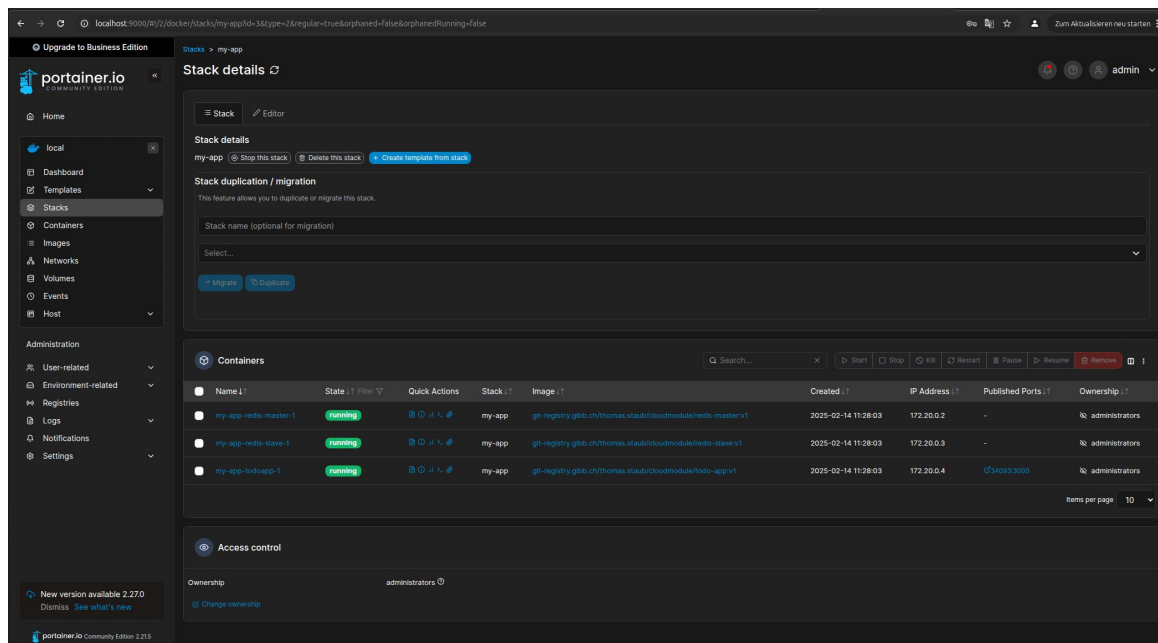


Abbildung 9: Stack Details

## Vorgehen

Zuerst einloggen:

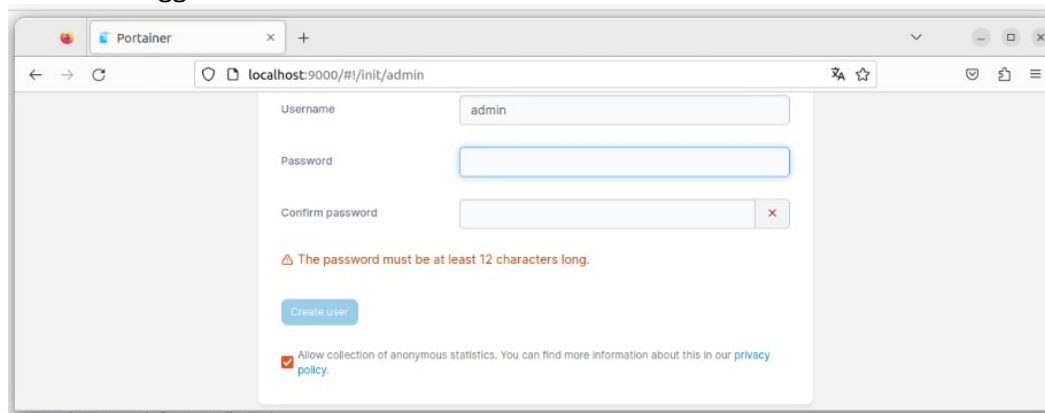


Abbildung 10: Einloggen bei Portainer

## Unter «Stacks» auf Add Stack

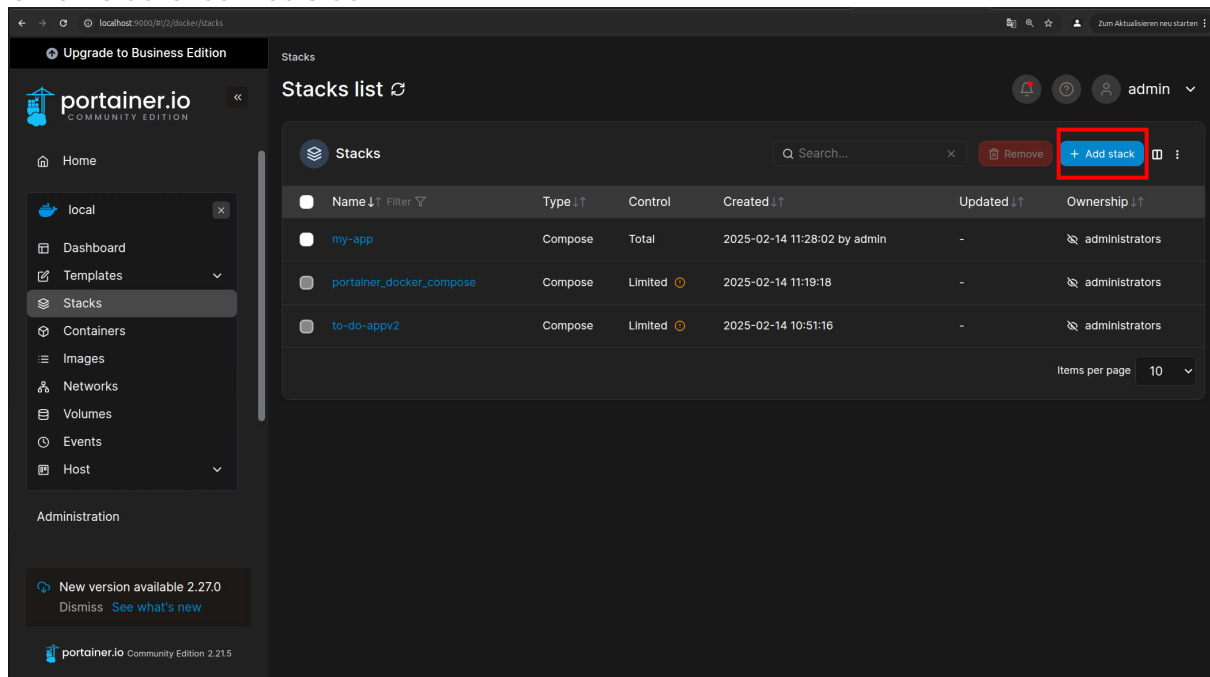


Abbildung 11: Neues Stack auf Portainer erstellen

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

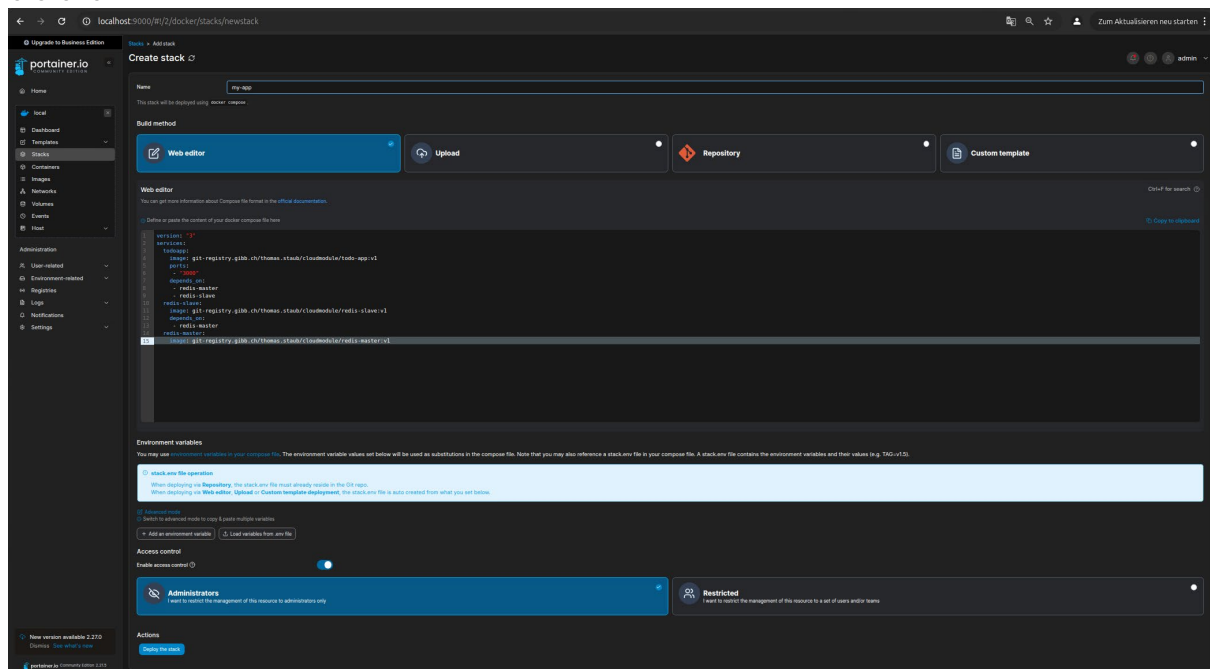


Abbildung 12: Skript von Thomas Staub verwenden

Danach sieht man es auch im Docker Desktop

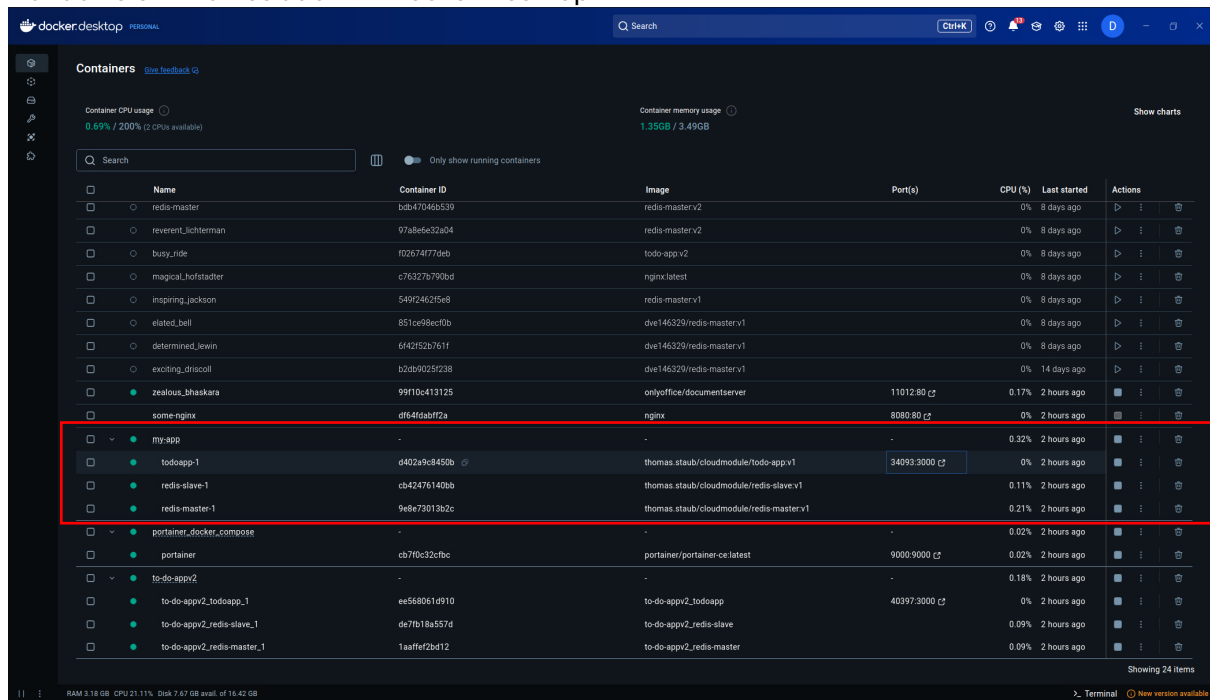


Abbildung 13: Docker Compose von Portainer

## 22.0 Learning Beispiel Shop

### Vorgehen

Zuerst das Repo heruntergeladen.

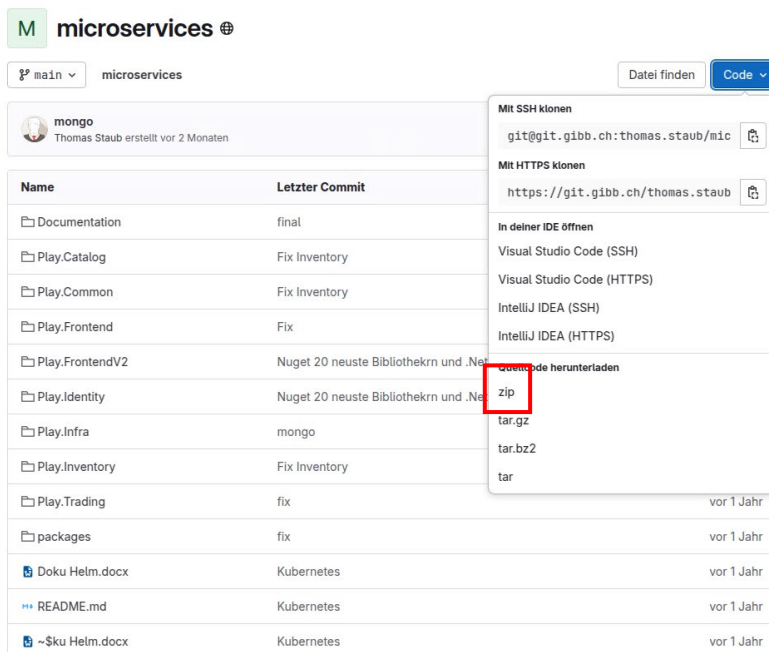


Abbildung 14: Thomas Staub's Repository

Danach entpackt:

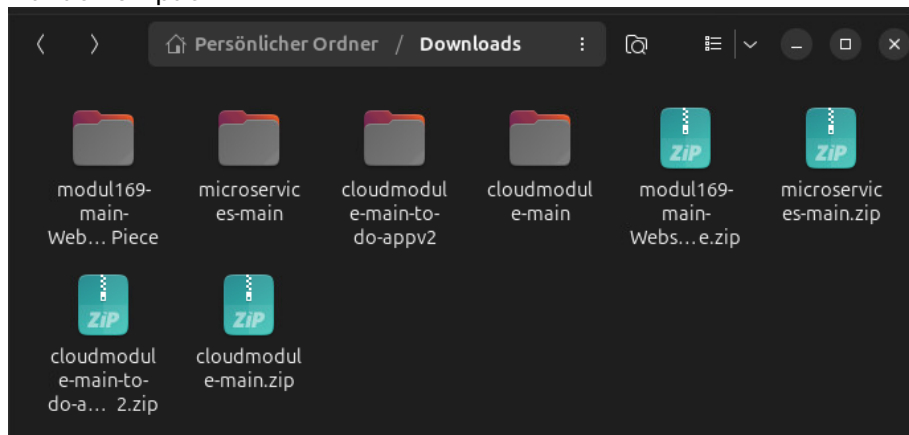


Abbildung 15: Downloads Ordner

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:~/dockermodule/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	f1ed9eab2eb	docker_grafana	4000:3000	0.03%	3 minutes ago			
rabbitmq	9f39f833bfca	rabbitmq.management	15672:15672	1.05%	3 minutes ago	Show all ports (2)		
prometheus	ae8480cb61f8	prom/prometheus	9090:9090	0.15%	3 minutes ago			
mongo	100a2c188df0	mongo	27017:27017	0.37%	3 minutes ago			
jaeger	2a97d6c4300f	jaegertracing/all-in-one	14250:14250	0.01%	3 minutes ago	Show all ports (6)		
seq	e9c1578bc44b	datastus/seq	5341:5341	0.28%	3 minutes ago	Show all ports (2)		
trading	26b83c526daa	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	cce0bc8cfbb3	thomas.staub/microservices/play.frontendv2:1.0	5008:80	0%	3 minutes ago			
inventory	8c35a129fbc1	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.

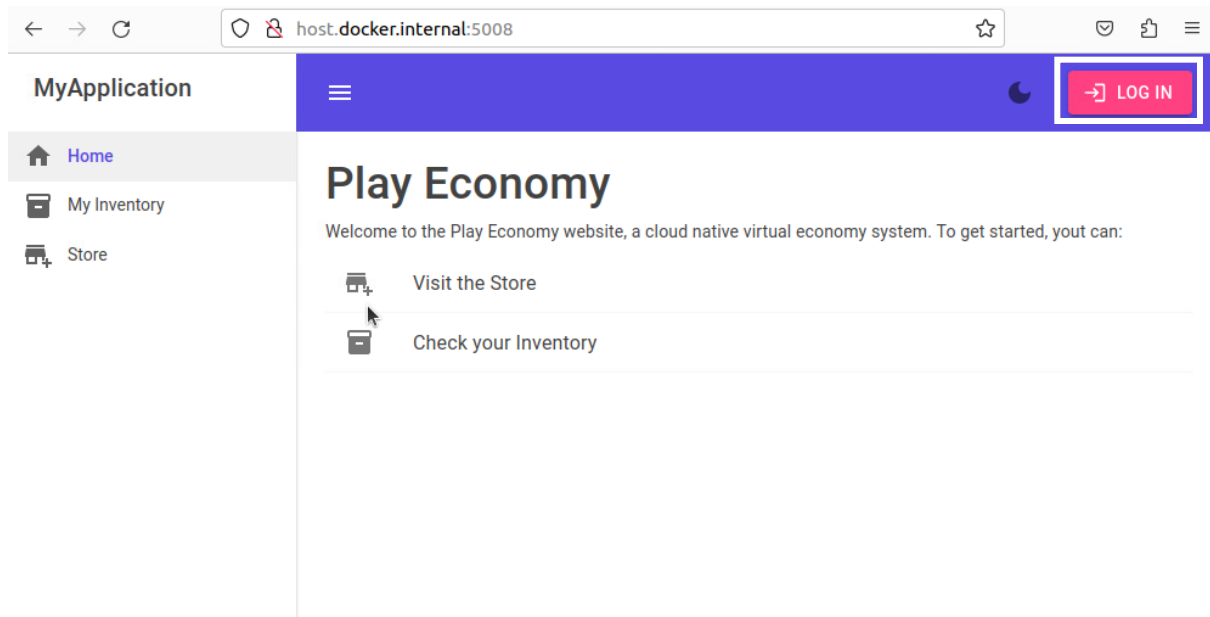


Abbildung 19: Beispiel Shop

Mit [admin@play.com](mailto:admin@play.com) und Pass@word1 anmelden

A screenshot of a web application's login page. The header shows 'Play.Identity.Service' on the left and 'Register Login' on the right. The main heading is 'Log in'. Below it, there are two columns. The left column is for local account login, featuring a form with 'Email' (admin@play.com) and 'Password' (masked with dots) fields, a 'Remember me?' checkbox, and a blue 'Log in' button. Below the button are three links: 'Forgot your password?', 'Register as a new user', and 'Resend email confirmation'. The right column is titled 'Use another service to log in.' and contains a message: '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

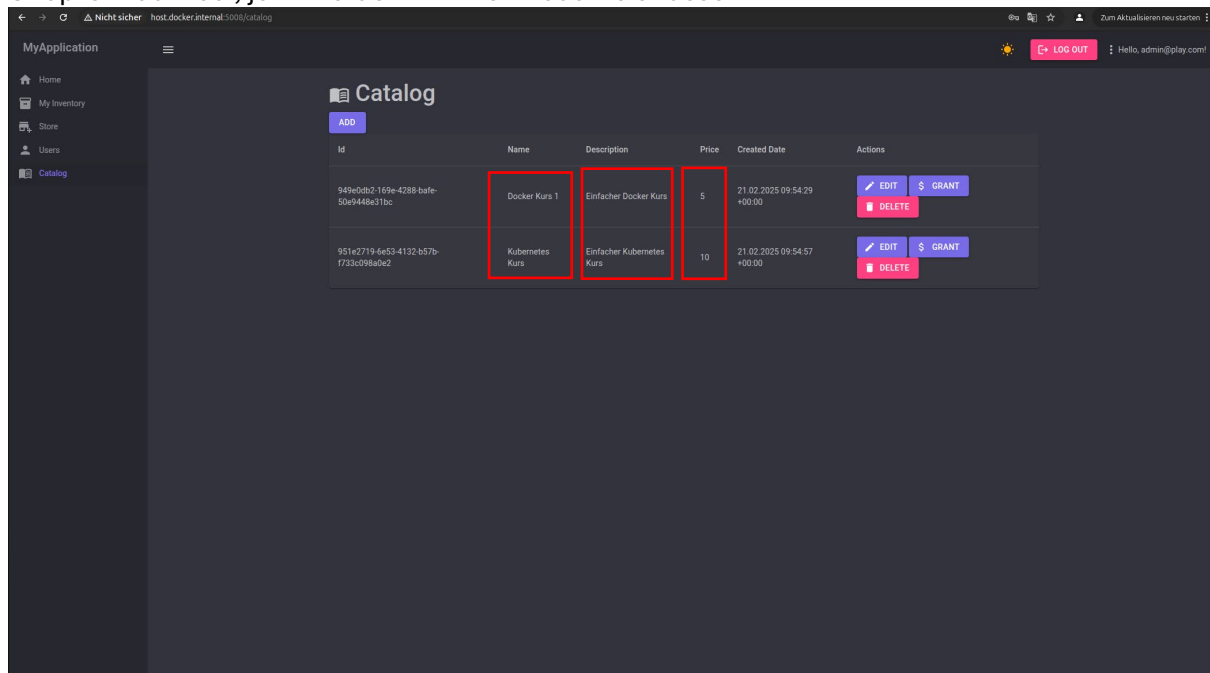


Abbildung 21: Katalog von Beispielshop

Unter Users können wir uns virtuelles Geld «Gil» geben

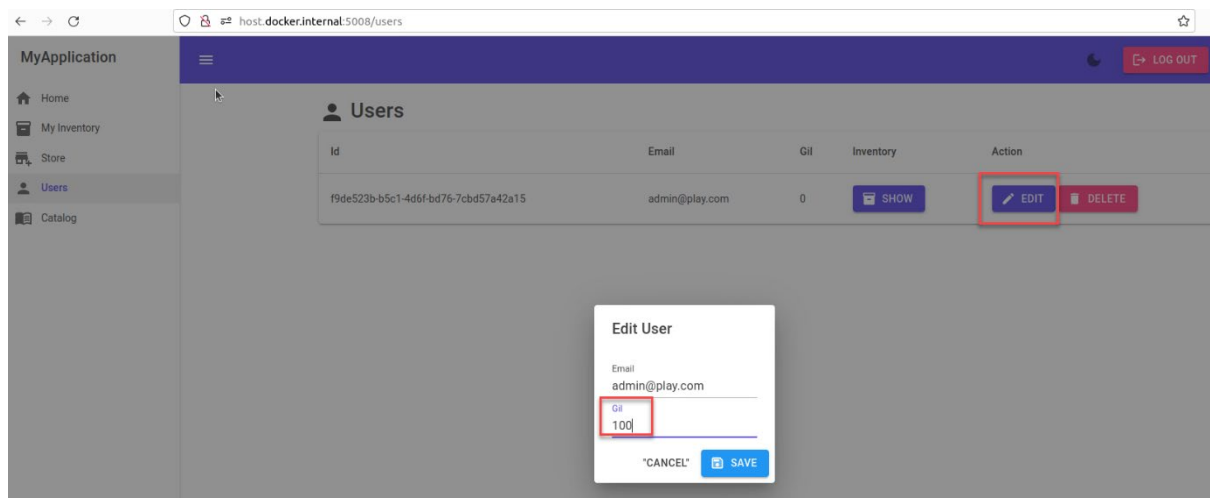


Abbildung 22: virtuelles Geld hinzufügen

Dann kann man mal eines kaufen

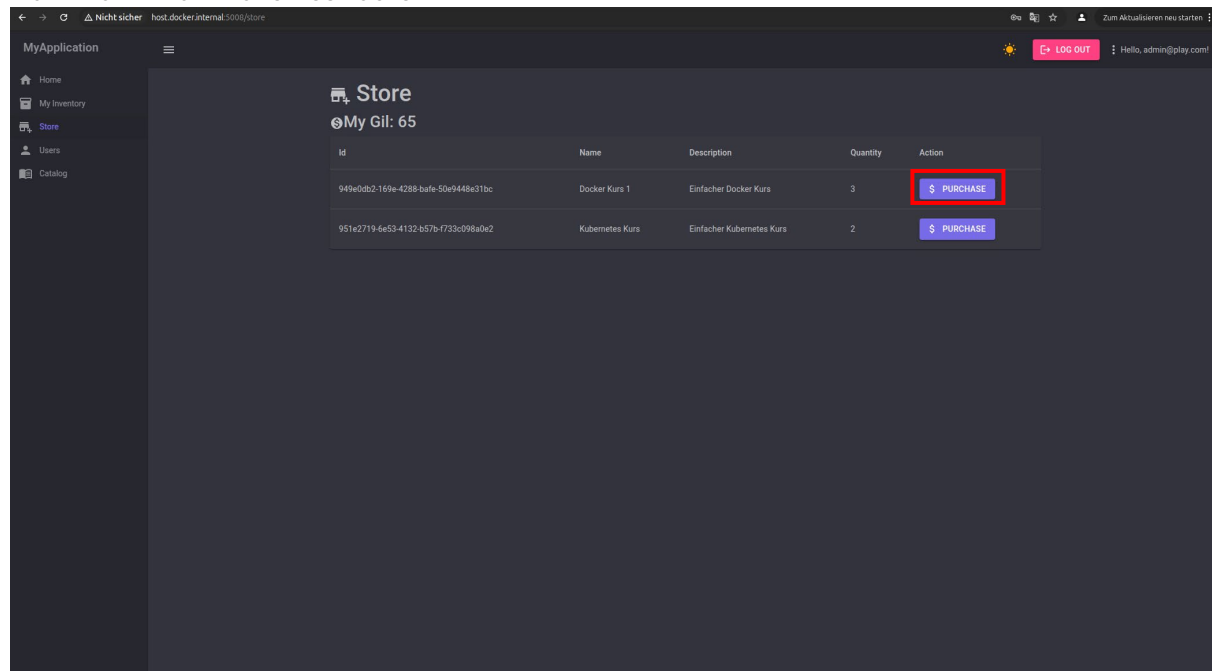


Abbildung 23: Store von Shop

## Zusätzliche Services

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

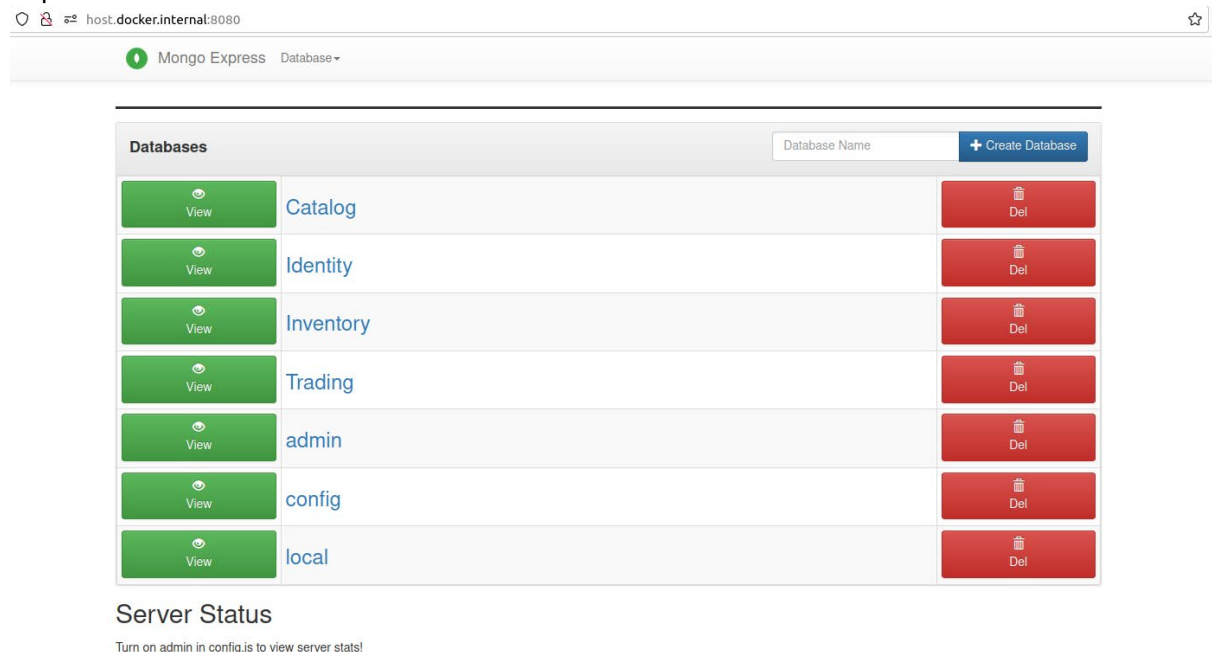


Abbildung 24: Mongo Express

Unter Catalog finden wir auch unsere gespeicherten Kurse

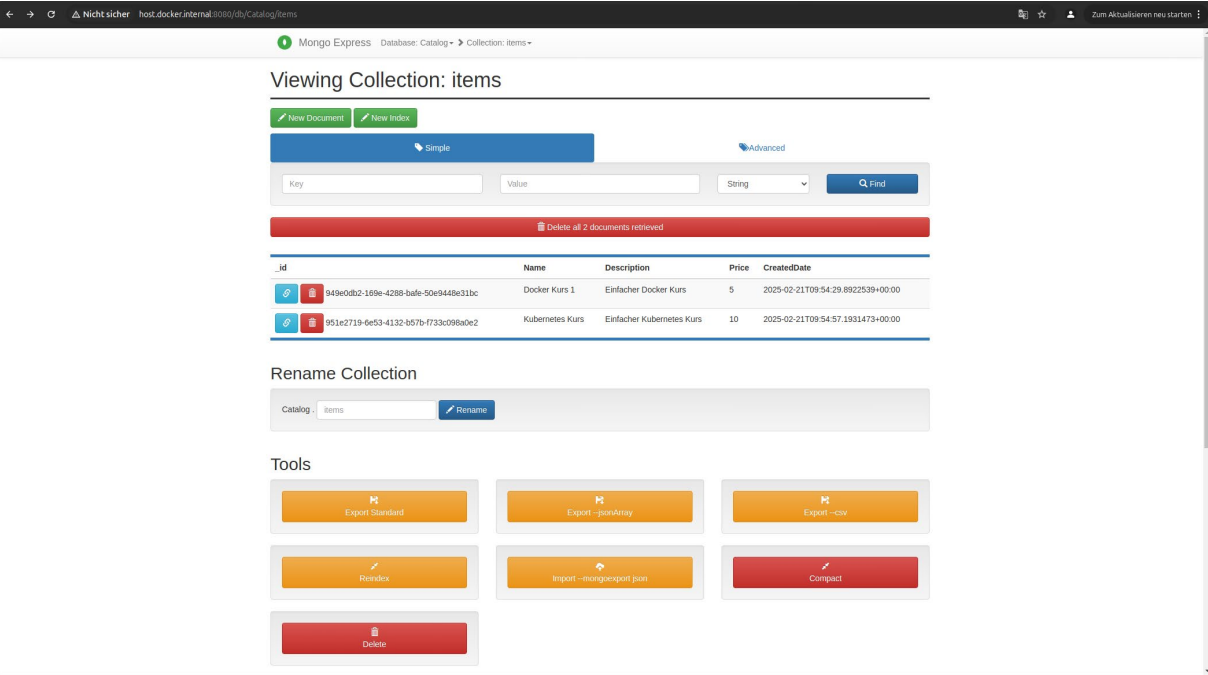


Abbildung 25: Mongo Express Catalog

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

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

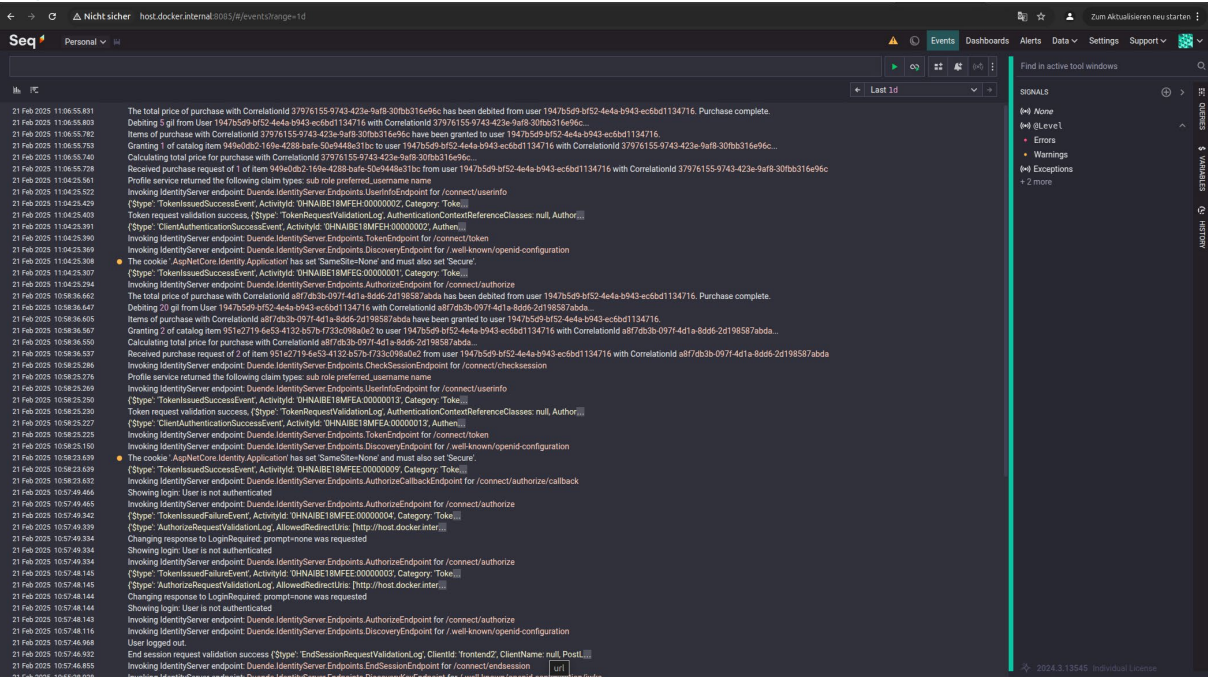


Abbildung 26: Events Webseite



# 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

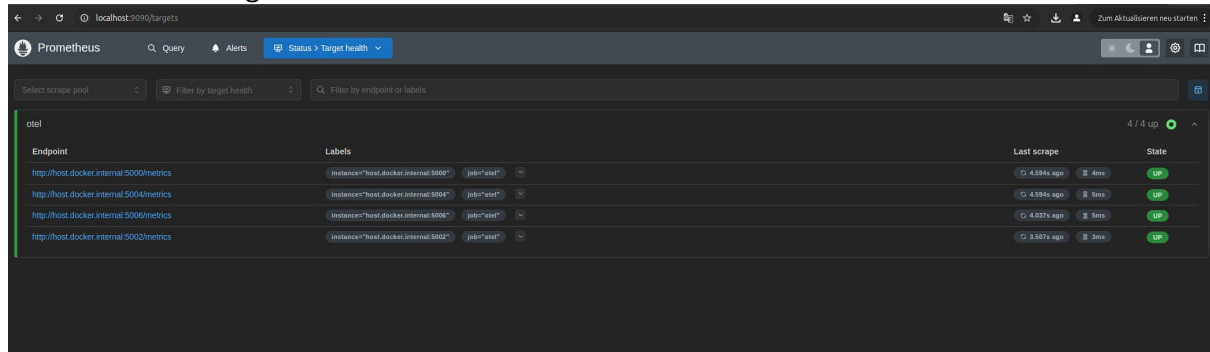


Abbildung 27: Prometheus Targets

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

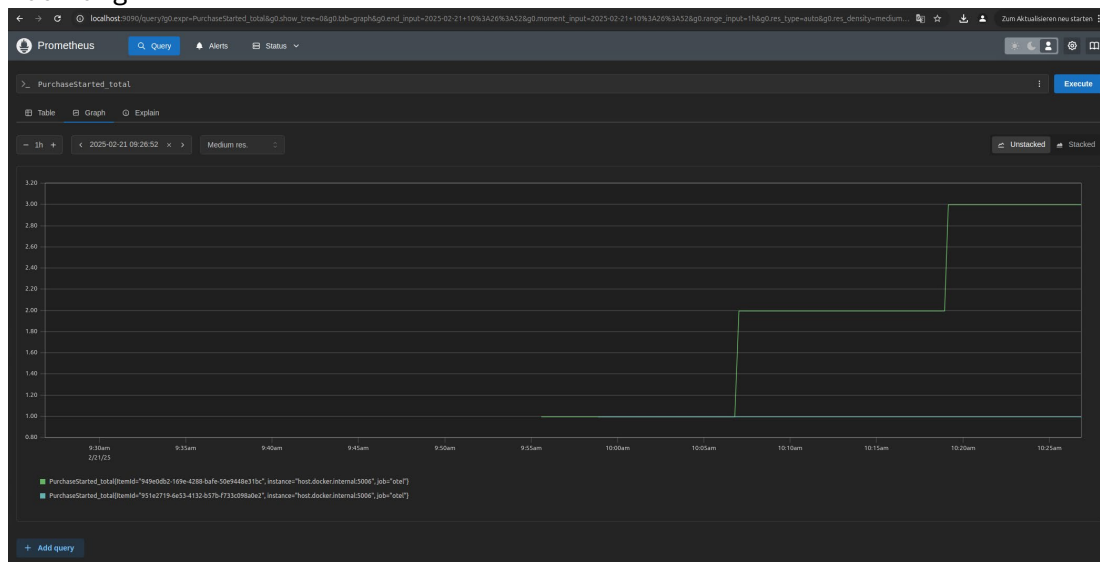


Abbildung 28: Prometheus PurchaseStarted Graph

Falls ein Service down geht sieht man das auch auf Prometheus

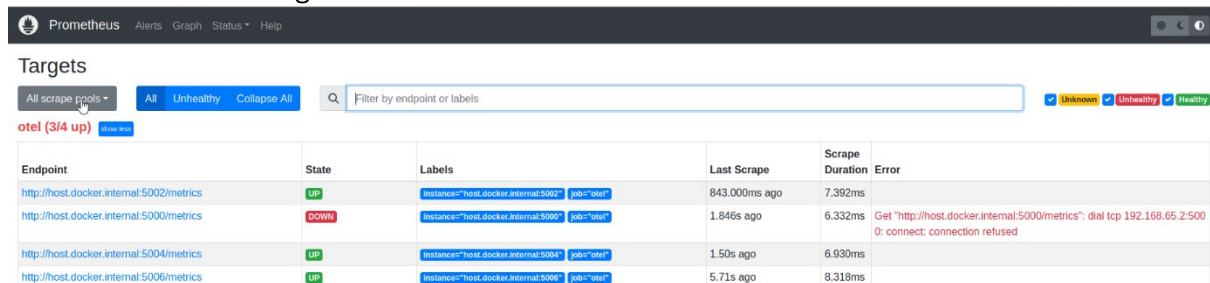


Abbildung 29: Prometheus Service Down

## Grafana

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

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

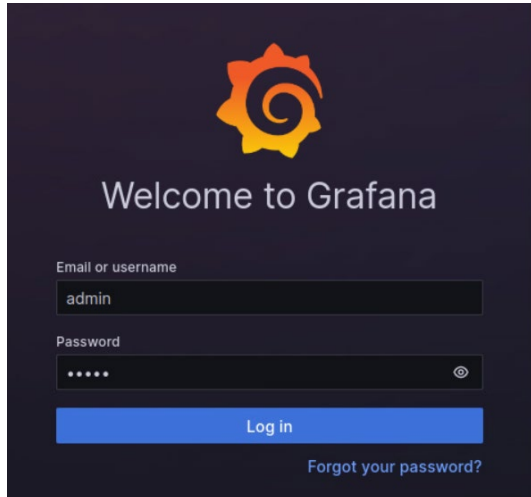
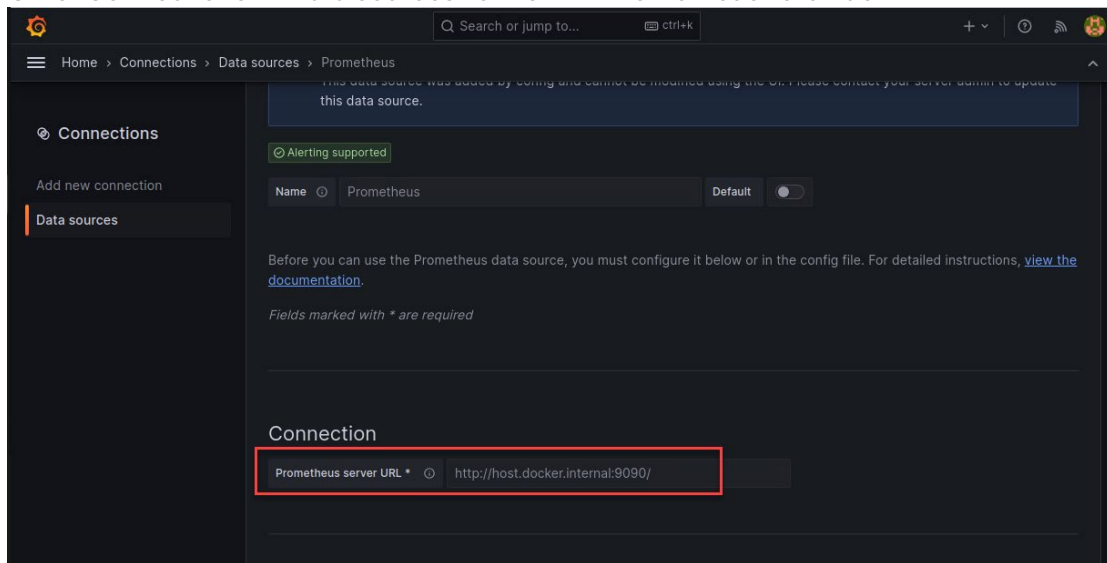
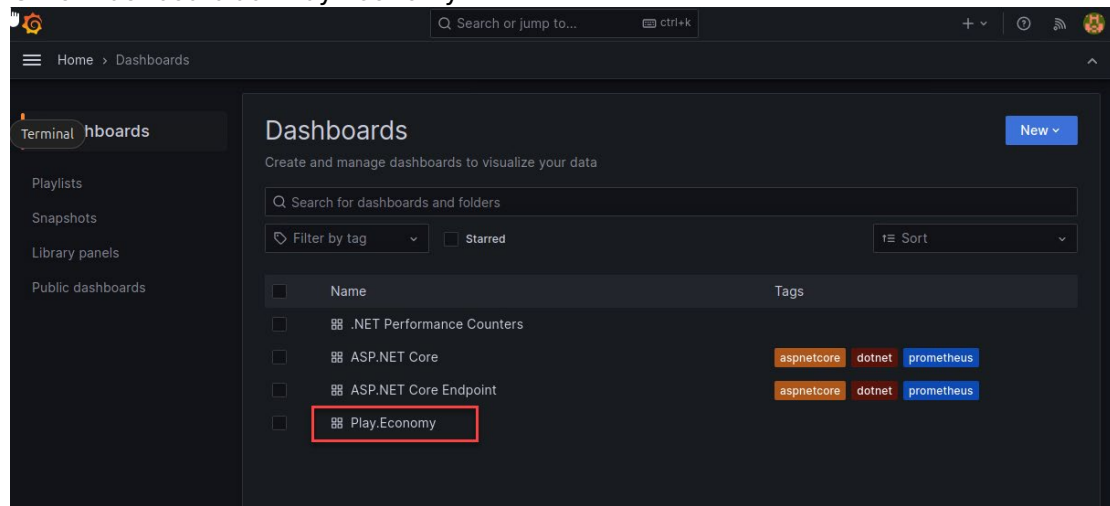


Abbildung 30: Startseite von Grafana

Unter Connections → Data sources können wir Prometheus verbinden



## Unter Dashboard auf Play.Economy



Und schon können wir sehen was in unserem Shop passiert.

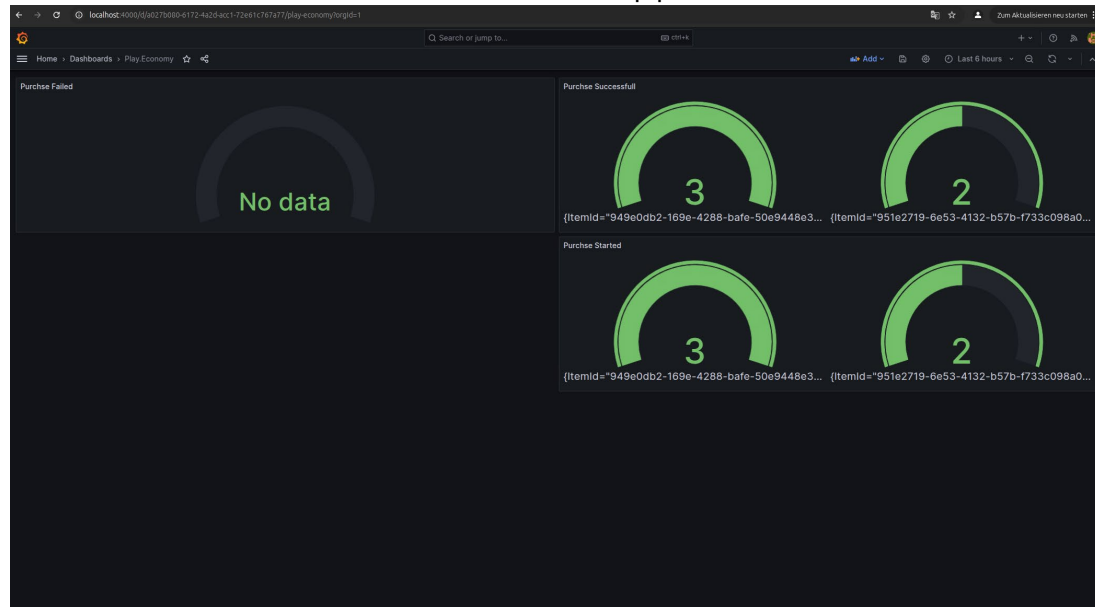
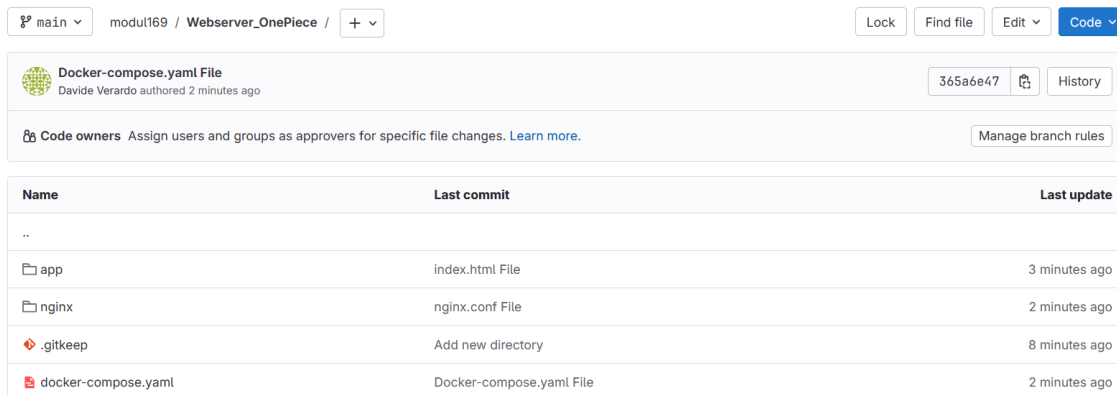


Abbildung 31: Grafana Dashboard vom Shop

## 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 32: 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:

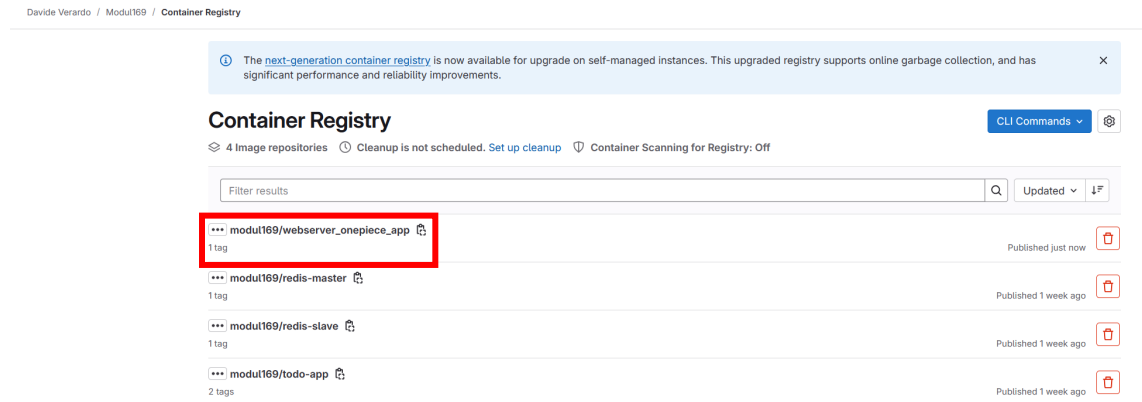
```
version: '3'
services:
  nginx:
    image: jwilder/nginx-proxy:latest
    ports:
      - "3080:80"
    volumes:
      - /var/run/docker.sock:/tmp/docker.sock:ro
    environment:
      - DEFAULT_HOST=davide.example

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

  restart: always
```

Abbildung 33: Docker Compose Datei für Webserver

# PrintScreen des Images bei (gibb) GitLab



## Anleitung

Zuerst den Projekt Ordner herunterladen

[https://git.gibb.ch/dve146329/modul169/-/tree/main/Webserver\\_OnePiece?ref\\_type=heads](https://git.gibb.ch/dve146329/modul169/-/tree/main/Webserver_OnePiece?ref_type=heads)

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 34: Webserver Webseite

## 24.0 Was ist Kubernetes

Kubernetes, auch als K8s bekannt, ist ein Open Source System, das in Google entwickelt wurde. Kubernetes wird verwendet, um Bereitstellungen, Skalierungen sowie Verwaltungen containerisierter Anwendungen zu automatisieren. In anderen Worten, Kubernetes ist ein Orchestrator von Cloud native Microservice Anwendungen.

## 25.0 Was sind Microservices

Eine Microservice Anwendung wird nach ihrer Funktionalität wie Web Frontend, Authentifizierung, Protokollierung, Datenspeicherung, Berichterstellung unterteilt, wodurch Mini Anwendungen oder Mini Services für jede Funktion erstellt wird. Daher der Begriff "Microservice".

## 26.0 Vergleich der lightweight Kubernetes Anwendungen (4 Stück)

### Mks

K3s ist ein leichtgewichtiges Tool, welches entwickelt wurde, um Kubernetes Workloads auf Produktionsebene mit geringen Ressourcen und entfernten Standorten auszuführen. K3s hilft uns, eine einfache, sichere und optimierte Kubernetes Umgebung auf einem lokalen Computer mit virtuellen Maschinen wie VMWare oder VirtualBox auszuführen.

### Micro8s

Microk8s, entwickelt von Canonical, ist eine Kubernetes Distribution, die für schnelle, selbstheilende und hochverfügbare Kubernetes Cluster entwickelt wurde. Es wurde auch für die schnelle und einfache Installation von Single und Multi Node Clustern auf mehreren Betriebssystemen optimiert. Es ist ideal für die Ausführung von Kubernetes in der Cloud, in lokalen Entwicklungsumgebungen sowie auf Edge und IoT Geräten. Es funktioniert auch effektiv auf Standalone Systemen mit ARM oder Intel, z.B. Raspberry Pi.

### Kind

Kind, auch Kubernetes in Docker genannt, wurde in erster Linie zum Testen von Kubernetes entwickelt und hilft einem, Kubernetes Cluster lokal und in CI Pipelines mit Docker Containern als "Knoten" auszuführen. Es unterstützt Multi Node Cluster und kann aus dem Quellcode Kubernetes Release Builds erstellen.

### MiniKube

MiniKube ist das am häufigsten verwendete lokale Kubernetes Installationsprogramm. Es bietet eine benutzerfreundliche Anleitung für die Installation und den Betrieb einzelner Kubernetes Umgebungen auf mehreren Betriebssystemen. Es verfügt über erweiterte Funktionen wie

Lastenausgleich, Dateisystem Mounting und Feature Ports, was es zu einem Favoriten für die lokale Ausführung von Kubernetes macht.

## Vergleich

Jedes dieser Tools bietet eine einfach zu bedienende und leichtgewichtige lokale Kubernetes Umgebung für mehrere Plattformen, aber es gibt ein paar Dinge, die sie von anderen unterscheiden.

K3s, zum Beispiel, bietet eine auf virtuellen Maschinen basierte Kubernetes Umgebung. Um mehrere Server von Kubernetes zu konfigurieren, müssen Sie manuell andere virtuelle Maschinen oder Knoten konfigurieren, was recht schwierig sein kann. Es ist aber für die Verwendung in der Produktion gemacht, was es zu einer der besten Möglichkeiten für die lokale Simulation eines echten Produktionsumgebungs macht.

Während Minikube im Allgemeinen eine gute Wahl für die lokale Ausführung von Kubernetes ist, besteht der Hauptnachteil darin, dass es nur einen Knoten im lokalen Kubernetes Cluster ausführen kann, was es etwas weit von einer effizienten Kubernetes Umgebung mit mehreren Knoten macht.

Im Gegensatz zu miniKube, kann microK8S mehrere Knoten in einem lokalen Kubernetes Cluster ausführen.

## 27.0 Kubernetes Installation Anleitung

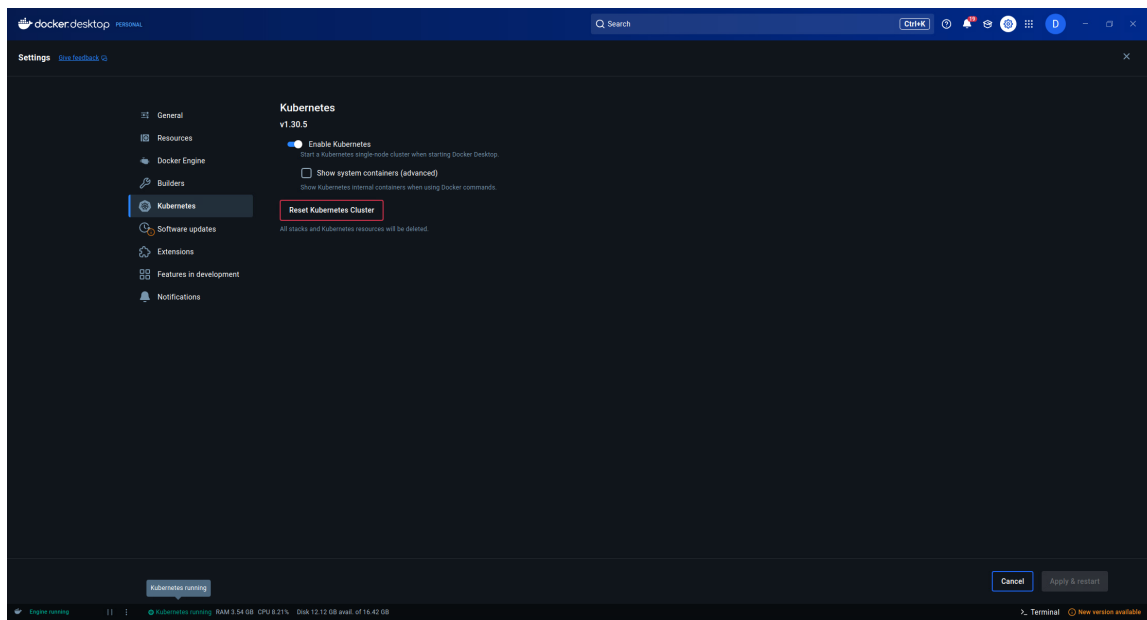


Abbildung 35: Einstellungen Docker

```
vmadmin@li244-vmLP1:~$ sudo snap install kubectl --classic
kubectl 1.32.2 from Canonical✓ installed
```

Abbildung 36: kubectl installieren

```
vmadmin@li244-vmLP1:~$ kubectl config get-contexts
```

CURRENT	NAME	CLUSTER	AUTHINFO	NAMESPACE
*	docker-desktop	docker-desktop	docker-desktop	

Abbildung 37: Docker Desktop info bekommen

```
vmadmin@li244-vmLP1:~$ kubectl config use-context docker-desktop
Switched to context "docker-desktop".
```

Abbildung 38: Mit Docker Desktop verknüpfen

## 28.0 Printscreen von Verbindung von Lens zu Server

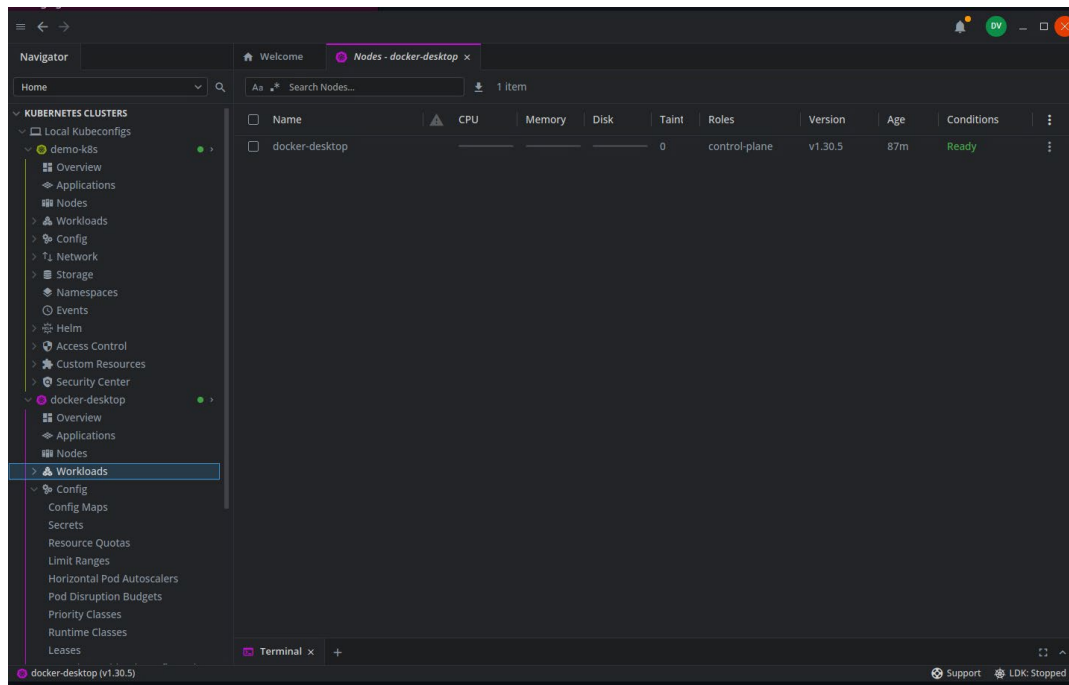


Abbildung 39: Workloads

## 29.0 Was ist der Raft-Konsens-Algorithmus

Der Raft Algorithmus sorgt dafür, dass mehrere Server in einem Cluster sich auf den gleichen Zustand einigen. Er wählt einen Leader, der Änderungen verwaltet und an die anderen Server weitergibt. Eine ungerade Anzahl von Servern ist wichtig, damit immer eine Mehrheit entscheiden kann. Bei einer geraden Anzahl könnte es zu einem Stillstand kommen, wenn die Stimmen gleich verteilt sind. Mit einer ungeraden Anzahl wird das vermieden und das System bleibt funktionsfähig.



## 30.0 Festhalten wie die App in Kubernetes läuft

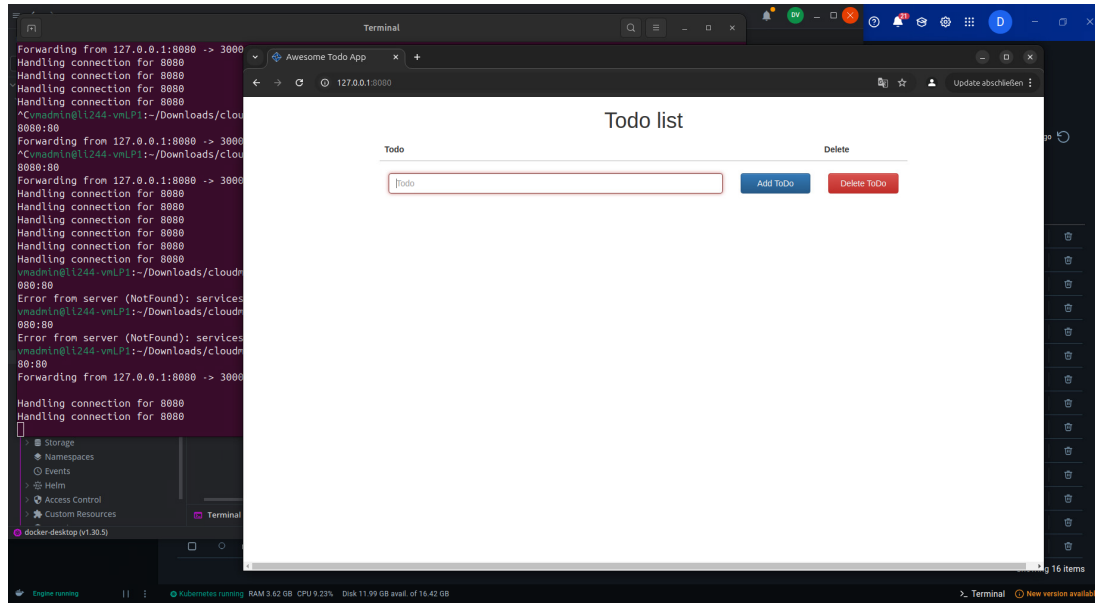


Abbildung 40: Todo App via Kubernetes

## Self Healing

Kubernetes überwacht in einem Cluster ständig die Container und Anwendungen. Im Falle eines Containerfehlers oder Containerproblems versucht Kubernetes automatisch, den Container neu zu starten oder eine alternative Instanz bereitzustellen. Dieser Ablauf wird als Self Healing bezeichnet und trägt dazu bei, Ausfallzeiten und Serviceunterbrechungen zu verringern.

Dadurch stellt Kubernetes sicher, dass Anwendungen stets verfügbar und funktionsfähig bleiben, selbst wenn einzelne Container ausfallen.

## Scale Up

Kubernetes bietet auch eine automatische Skalierungsfunktion namens Skalierung. Wenn die Nachfrage an Anwendungen oder Dienste steigt, kann Kubernetes die Anzahl der ausgeführten Container oder Instanzen automatisch erhöhen. Dieser Prozess trägt dazu bei, dass die bereitgestellte Anwendung reibungslos läuft und die Anforderungen eines Benutzers erfüllt und sorgt gleichzeitig für eine optimale Ressourcennutzung, indem nicht benötigte Instanzen bei geringerer Nachfrage reduziert werden.

## Scale Down

Im Gegenzug zur Scale Up gibt es auch die Scale Down Funktion. Wenn die Nachfrage an Anwendungen oder Dienste sinken, kann Kubernetes die Anzahl der ausgeführten Container oder Instanzen automatisch reduzieren, um Ressourcen zu sparen sowie Kosten zu senken.

Dieser Prozess hilft, die Effizienz der Infrastruktur zu maximieren und eine optimale Nutzung der verfügbaren Ressourcen sicherzustellen.

## 31.0 PrintScreen das das Rolling Update funktioniert. Version 2 ist im Dashboard ersichtlich

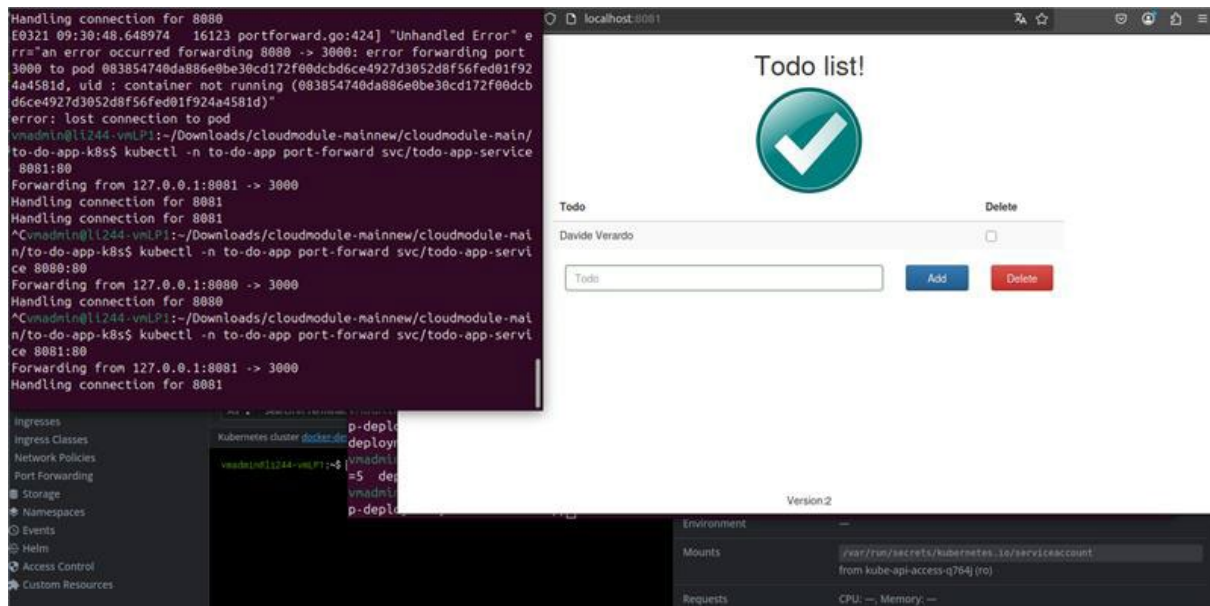


Abbildung 41: Rolling Updates

## Blue Green Deployment

Blue-Green Deployment ist eine Technologiemethode zur Minimierung der Ausfallzeiten und der Risiken, indem zwei ähnliche Produktivumgebungen – Blau und Grün – bereitgehalten werden.

Zu jedem Zeitpunkt ist nur eine der beiden Umgebungen (Blau oder Grün) aktiv, wobei der gesamte Produktivbetrieb über die jeweils aktive Umgebung läuft. In unserem Beispiel ist die blaue Umgebung im Betrieb, während die grüne Umgebung inaktiv ist.

Wenn eine neue Version einer Anwendung bereitsteht, erfolgt die Bereitstellung sowie der abschließende Test in der aktuell nicht produktiven Umgebung, also der grünen Umgebung. Nach der Bereitstellung und dem erfolgreichen Testen in der inaktiven Umgebung wird der Router so konfiguriert, dass alle eingehenden Anfragen nun an die grüne Umgebung weitergeleitet werden statt an die blaue. Die blaue Umgebung ist nun nicht mehr aktiv, die grüne Umgebung hingegen schon.

Mit dieser Technik werden Betriebsunterbrechungen bei der Anwendungsbereitstellung vermieden. Die blau-grüne Bereitstellung verringert außerdem das Risiko: Sollte mit der neuen Version in der grünen Umgebung etwas Unerwartetes geschehen, kann man sofort zur vorherigen Version zurückkehren, indem man einfach wieder auf die blaue Umgebung umschaltet

## Cluster IP

Cluster IP ist eine virtuelle IP, die eigentlich ein gefälschtes IP Netzwerk ist. Der Dienst kann eine einheitliche Eingangsadresse für eine Gruppe von Containeranwendungen mit derselben Funktion bereitstellen und die Anfragelast auf jede Containeranwendung im Backend verteilen.

Mehrere Container im Backend können, als Cluster betrachtet werden, und der Service ist der Einstiegspunkt des Clusters. Daher wird die Service IP auch Cluster IP genannt.

- Cluster IP wirkt nur auf das k8s Service Objekt und wird von K8s verwaltet.
- Cluster IP kann nicht angefunkelt werden, da es kein "Entity Network Object" gibt, das antworten kann.
- Cluster IP kann nur mit Service Port kombiniert werden, um einen bestimmten Kommunikationsport zu bilden. Separate Cluster IPs haben keine Basis für TCP/IP Kommunikation und gehören zu einem geschlossenen Bereich wie einem K8s Cluster.

## Node IP

Kubernetes verwendet verschiedene IP Bereiche, um den Nodes, Pods, und Services IP Adressen zuzuweisen. Jedem Node wird eine IP Adresse aus dem Virtual Private Cloud (VPC) Netzwerk des Clusters zugewiesen. Diese Node IP sorgt für die Konnektivität von Kontrollkomponenten wie kube proxy und kubelet zum Kubernetes API Server.

## LoadBalancer

Wenn Sie ein Service als LoadBalancer deklarieren, wird dieser Dienst extern über den Load Balancer des Cloud Anbieters verfügbar gemacht. Bei Microk8s ist es zum Beispiel der MetalB.

Ein LoadBalancer ist meist die beste Option für eine Produktionsumgebung, mit zwei Einschränkungen:

- Jeder Dienst, den Sie als LoadBalancer bereitstellen, hat seine eigene IP Adresse.
- Die LoadBalancer Abrechnung basiert normalerweise auf der Anzahl der geöffneten Dienste, was teuer sein kann.

## 32.0 PrintScreen Wie Sie auf die App zugreifen

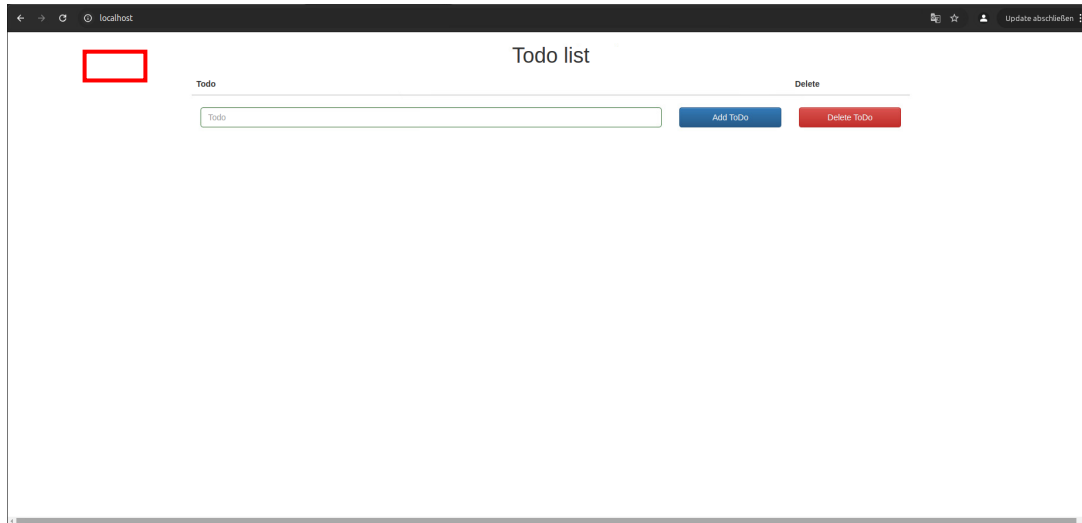


Abbildung 42: Todo App Ingress

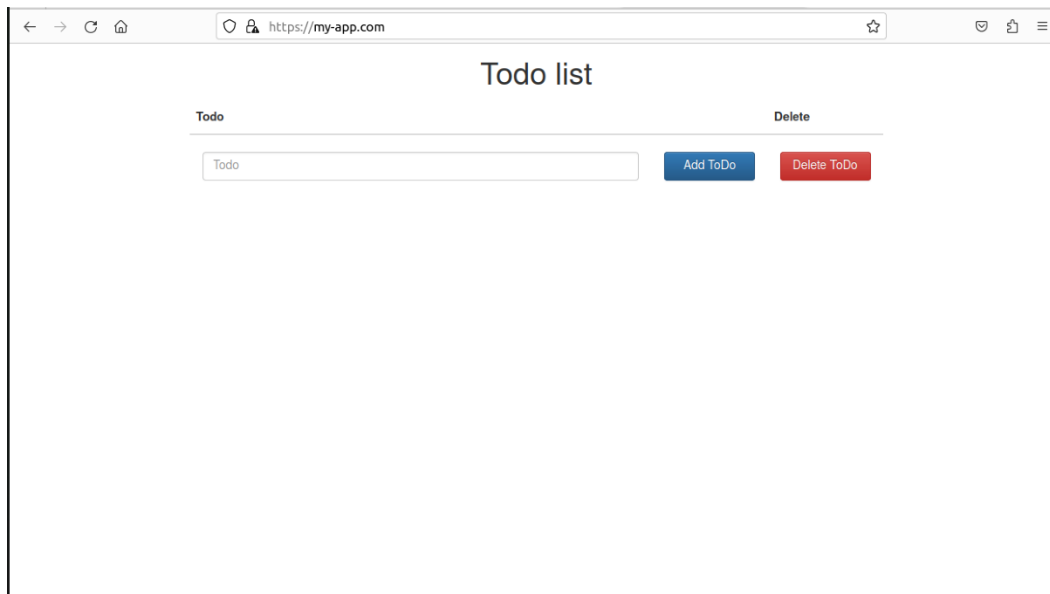


Abbildung 43: Todo App Ingress

## 33.0 Warum bei Ingress beim Zugriff auf 127.0.0.1 ein Error 404 erhalten

Beim Zugriff auf einen Ingress über localhost funktioniert alles, da der Ingress auf diesen Hostnamen konfiguriert ist. Nutzt man stattdessen 127.0.0.1, wird im HTTP-Header ein anderer Host übermittelt. Der Ingress kann dadurch keine passende Regel finden und gibt einen 404-Fehler zurück. Obwohl localhost und 127.0.0.1 technisch dieselbe Maschine meinen, unterscheidet sich der Hostname, auf den der Ingress reagiert. Um beide Varianten zu unterstützen, muss man entweder beide Hostnamen im Ingress definieren oder beim Zugriff den Host-Header anpassen

## 34.0 PrintScreen wie Portainer auf Kubernetes installiert ist

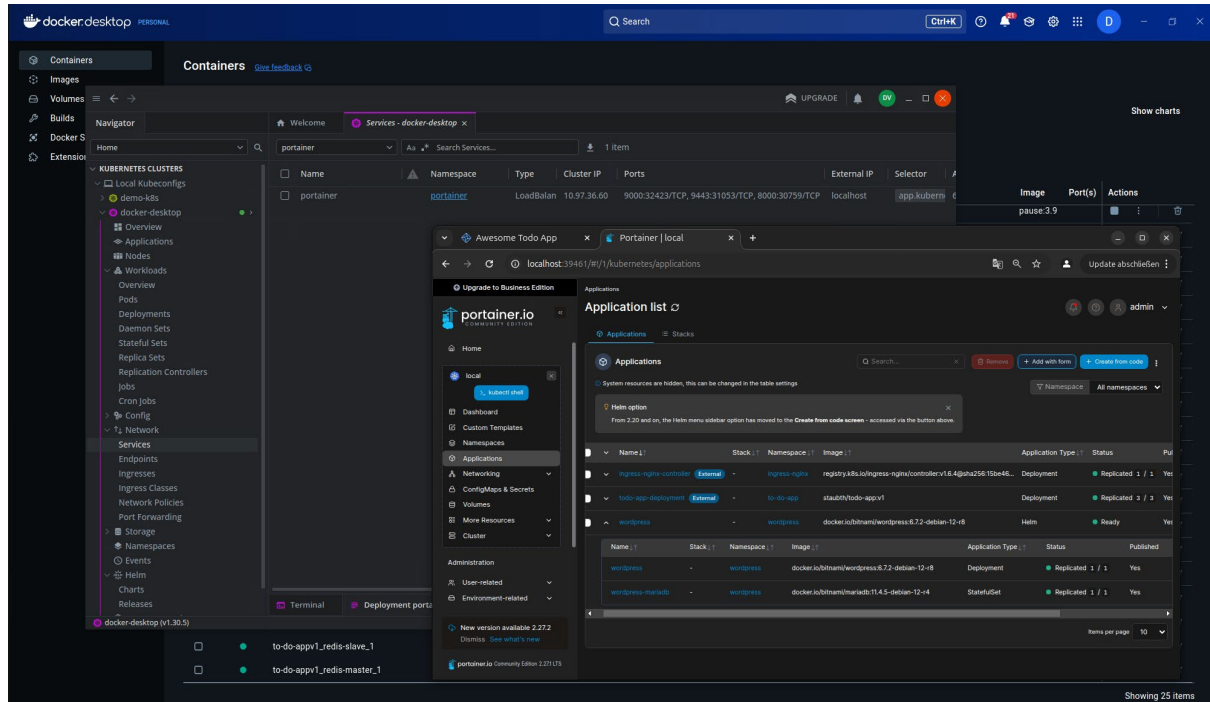


Abbildung 44: Portainer auf Kubernetes

## 35.0 Kubernetes Webserver

### Zuerst den Projekt Ordner herunterladen

[https://git.gibb.ch/dve146329/modul169/-/tree/main/Solo-Leveling\\_Kubernetes\\_Webapp](https://git.gibb.ch/dve146329/modul169/-/tree/main/Solo-Leveling_Kubernetes_Webapp)

### Als nächstes ins Ordner wechseln

```
cd Downloads/modul169-main/Solo-Leveling_Kubernetes_Webapp
```

### Minikube herunterladen

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube
minikube start
minikube dashboard
```

### kubectll herunterladen

```
sudo snap install kubectll --classic
```

### Namespace erstellen

```
kubectll apply -f namespace.yaml
```

### Alle YAML Dateien anwenden

```
kubectll apply -f . -n my-project
```

### Webserver aufrufen:

Via Nodeport:

```
kubectl get svc webserver-service
```

### Minikube-IP abrufen

```
minikube ip
```

### Im Browser eingeben

`http://<Minikube-IP>:<NodePort>`

z.B. <http://192.168.49.2:32393>

### Via Port-Forwarding

```
kubectl port-forward svc/webserver-service 8080:80
```

**Achtung:** Immer wieder eingeben sonst geht es zurück auf den Nodeport.

Heisst sobald man Ctrl + C macht kann man via Minikube Ip und Nodeport wieder auf den Webserver zugreifen

### Im Browser eingeben

<http://localhost:8080>

### Folgendes Ergebnis sollte erscheinen:

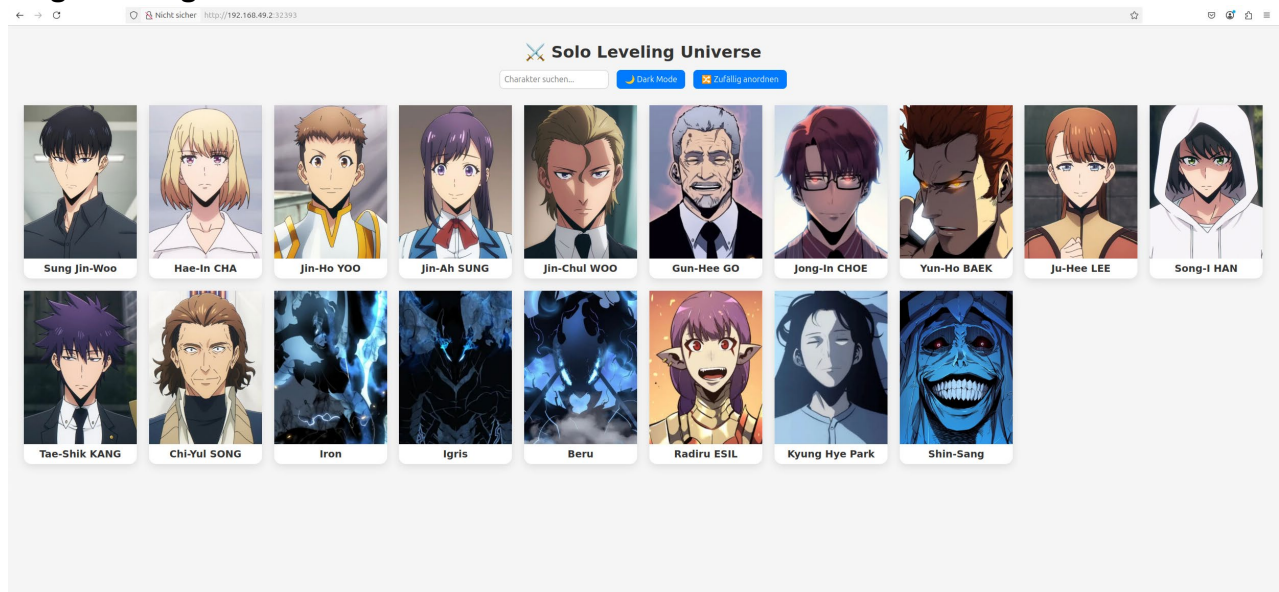


Abbildung 45: Solo-Leveling Webseite

## 36.0 todo App auf Podman zum laufen gekriegt und Dokumentiert.

Podman herunterladen:

- `sudo apt install podman`

Podman Desktop installieren:

- flatpack install flathub io.podman\_dekstop.PodmanDesktop

Pyhton3 installieren:

- sudo apt install python-venv

Virtuelle Umgebung mit Namen “podman-env” im aktuellen Verzeichnis erstellen:

- python -m venv podman-env

Aktivierung der virtuellen Pyhton Umgebung:

- source podman-env/bin/activate

Podman Compose installieren

- pip install podman-compose

FROM Zeile in den Dockerfiles “Master, Slave und Todoapp ändern”:

```
FROM docker.io/redis:alpine3.16
MAINTAINER Johannes M. Scheuermann <johannes.scheuermann@inovex.de>

COPY start-redis-master.sh /start-redis-master.sh
RUN chmod +x /start-redis-master.sh
CMD /start-redis-master.sh
```

Abbildung 46: Dockerfiles bearbeiten

Ins Verzeichnis wechseln:

- cd /Downloads/cloudmodule-main/to-do-appv1/web-frontend

Podman Images bauen:

- podman build -t redis-master:v1 -f Dockefile .
- podman build -t redis-slave:v1 -f Dockefile .
- podman build -t todo-app:v1 -f Dockerfile .

Netzwerk erstellen: podman

- network create todoapp\_network

Redis Master, Redis Slave und Frontend Container starten:

- podman run --network=todoapp\_network --name=redis-master -d redis-master:v1
- podman run --network=todoapp\_network --name=redis-slave -d redis-slave:v1
- podman run --network=todoapp\_network --name=frontend -d -p 3000:3000 todo-app:v1

Mit podman ps sieht man ob alles geklappt hat:

```
(podman-env) vmadmin@li244-vmLP1:~/Downloads/cloudmodule-main/to-do-appv1/web-frontend$ podman ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
6fd6fe788b27   localhost/redis-master:v1          ./todo-app              24 minutes ago Up 24 minutes                               redis-master
ac76d373c0ad   localhost/redis-slave:v1          ./todo-app              23 minutes ago Up 23 minutes                               redis-slave
0d9040429c20   localhost/todo-app:v1             ./todo-app              23 minutes ago Up 23 minutes   0.0.0.0:3000->3000/tcp frontend
(podman-env) vmadmin@li244-vmLP1:~/Downloads/cloudmodule-main/to-do-appv1/web-frontend$
```

Abbildung 47: podman ps Befehl

Um Podman Dekstop zu starten:

- flatpack run io.podman\_desktop.PodmanDesktop

Podman Dekstop Container:



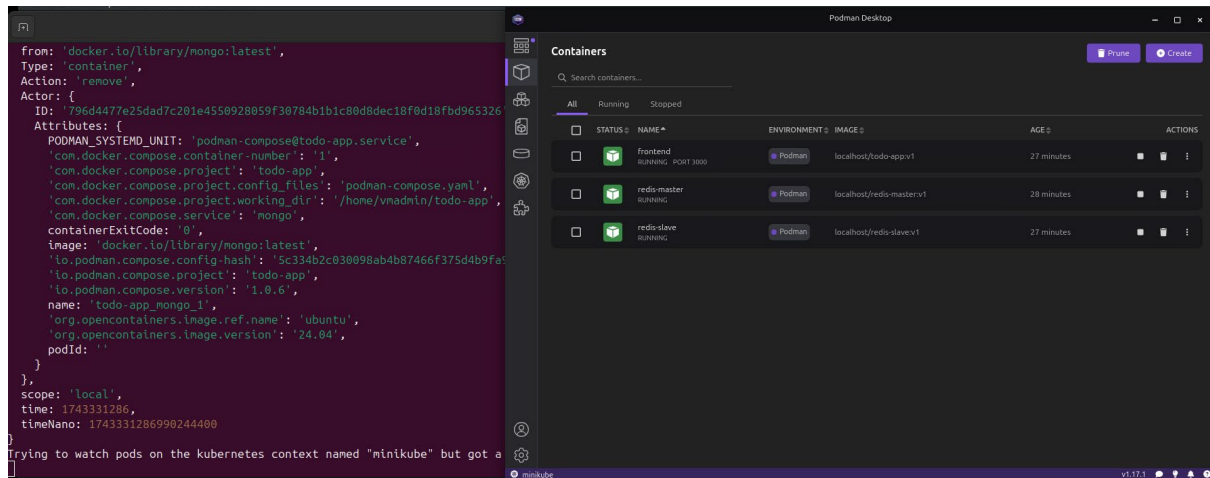


Abbildung 48: Podman Desktop Container Ansicht

## Podman Desktop Images:

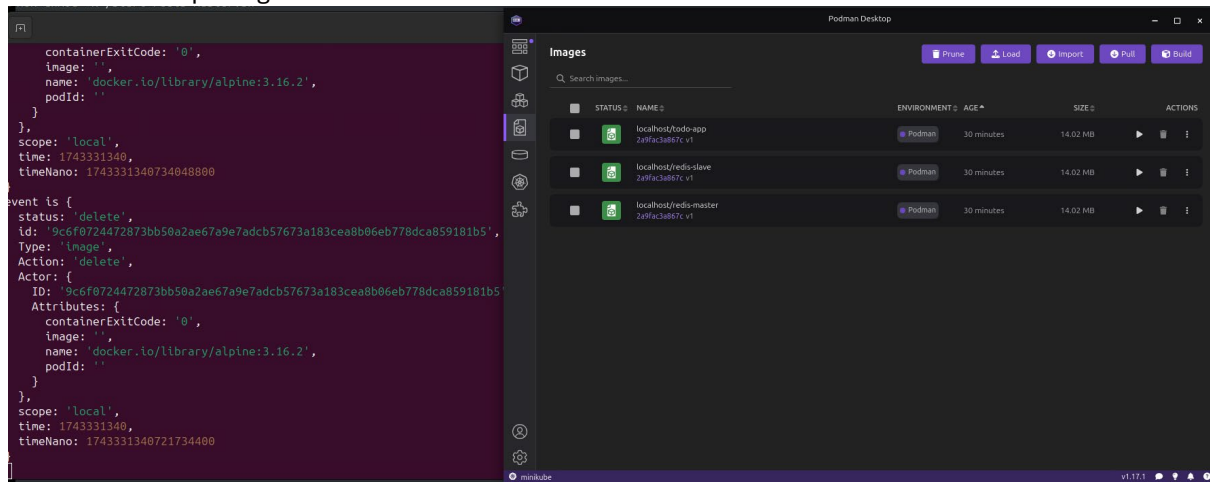


Abbildung 49: Podman Desktop Images Ansicht

## Todo App starten:

- localhost:3000

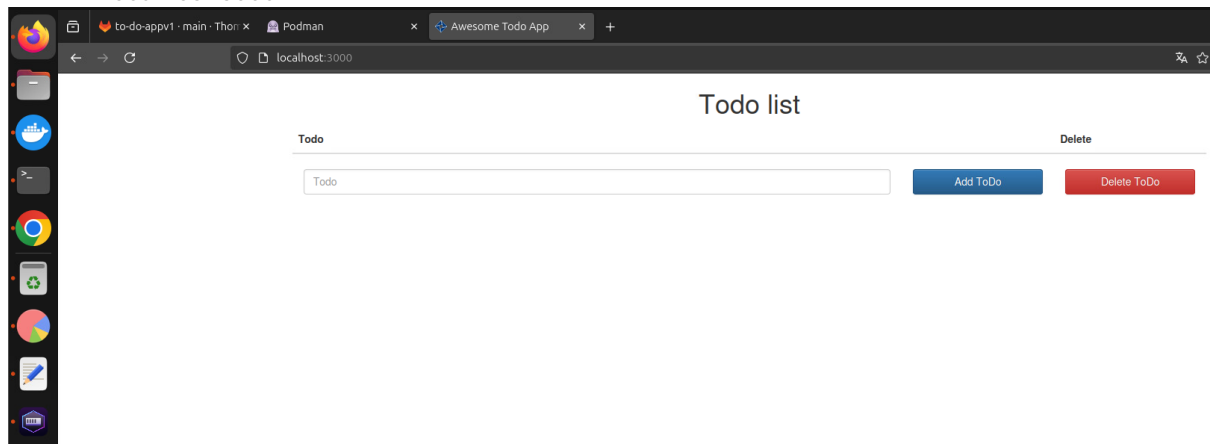


Abbildung 50: Podman Todo App starten