

docker

Modul 169

Modul	IET-169 – Services mit Containern bereitstellen
Klasse	INF2023c
Eingereicht von	Tharaneeka Ravitharan
Version	1.0.0
Eingereicht bei	Reto Brüttsch
Datum	28.03.2025

Inhaltsverzeichnis

1	Versionierung	3
1.1	Versionskontrolle	3
2	Tag 1, KW 5	4
2.1	Container und ihr Nutzen	4
2.2	DevOps	4
2.3	Virtualisierung vs. Containerisierung.....	4
2.4	Unterschied Image und Container	4
2.5	OnlyOffice.....	5
3	Tag 2, KW 6	6
3.1	Todo App Version 1	6
3.2	Images auf GitLab	6
3.2.1	Befehle für das Pushen.....	7
3.3	Todo App Version 2	8
4	Tag 3, KW 7	9
4.1	Docker Compose	9
4.2	To-do-App Version 2 mit Docker Compose.....	9
4.3	Portainer.....	10
4.4	App via Portainer installiert	11
4.5	Shop mit Docker Compose	13
5	Übungsprojekt Webserver	15
5.1	Anleitung Webserver	16
5.1.1	Schritt 1: GitLab Repository erstellen	16
5.1.2	Schritt 2: Projekt lokal einrichten	16
5.1.3	Schritt 3: Docker-Setup einrichten.....	16
5.1.4	Schritt 4: HTML-Webseite erstellen.....	16
5.1.5	Schritt 5: Nginx-Konfiguration erstellen	16
5.1.6	Schritt 6: Dockerfile erstellen	17
5.1.7	Schritt 7: Docker Compose Datei erstellen.....	17
5.1.8	Schritt 8: Docker-Container starten	17
5.1.9	Schritt 9: Projekt in GitLab hochladen	17
5.1.10	Schritt 10: Änderungen an der Webseite veröffentlichen.....	18
5.2	Anleitung für Lehrer	19
6	Befehle und ihre Funktionen	22
7	Tag 5, KW 9	24
7.1	Was ist Kubernetes?	24
7.2	Was sind Microservices?	24
7.3	Vergleich der lightweight Kubernetes Anwendungen	24

7.4	Installation Kubernetes	25
7.5	Lens Verbindung mit dem Server	26
8	Tag 6, KW 10	27
8.1	Raft-Konsens-Algorithmus	27
8.1.1	Was ist der Raft-Konsens-Algorithmus?	27
8.1.2	Grundprinzipien des Raft-Algorithmus:	27
8.1.3	Warum sollte ein Cluster eine ungerade Anzahl an Servern haben?	27
8.2	To-Do-App in Kubernetes	28
8.3	Self-Healing	30
8.4	Scale up und Scale down	30
8.5	Rolling Updates bei der Todo-App Version 2	30
8.6	Blue-Green Deployment	31
9	Tag 7, KW 11	33
9.1	Cluster IP	33
9.2	Node IP	33
9.3	LoadBalancer	33
9.4	Ingress	34
9.4.1	Warum funktioniert der Zugriff über 127.0.0.1 oder localhost nicht?	34
9.5	Portainer auf Kubernetes	35
10	NAS auf Kubernetes	36
10.1	Voraussetzungen	36
10.2	Projekt Aufbau	36
11	To-do-app auf Podman zum Laufen bringen	40

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 Layout Anpassungen 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
0.0.7	07.03.2025	Tharaneeka Ravitharan	- Tag 5, KW 9 - Tag 6, KW 10
0.0.8	14.03.2025	Tharaneeka Ravitharan	- Tag 7, KW 11
0.0.9	21.03.2025	Tharaneeka Ravitharan	- NAS auf Kubernetes
0.1.0	24.03.2025	Tharaneeka Ravitharan	- To-do-App auf Podman - Korrektur
1.0.0	28.03.2025	Tharaneeka Ravitharan	- Abgabe

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

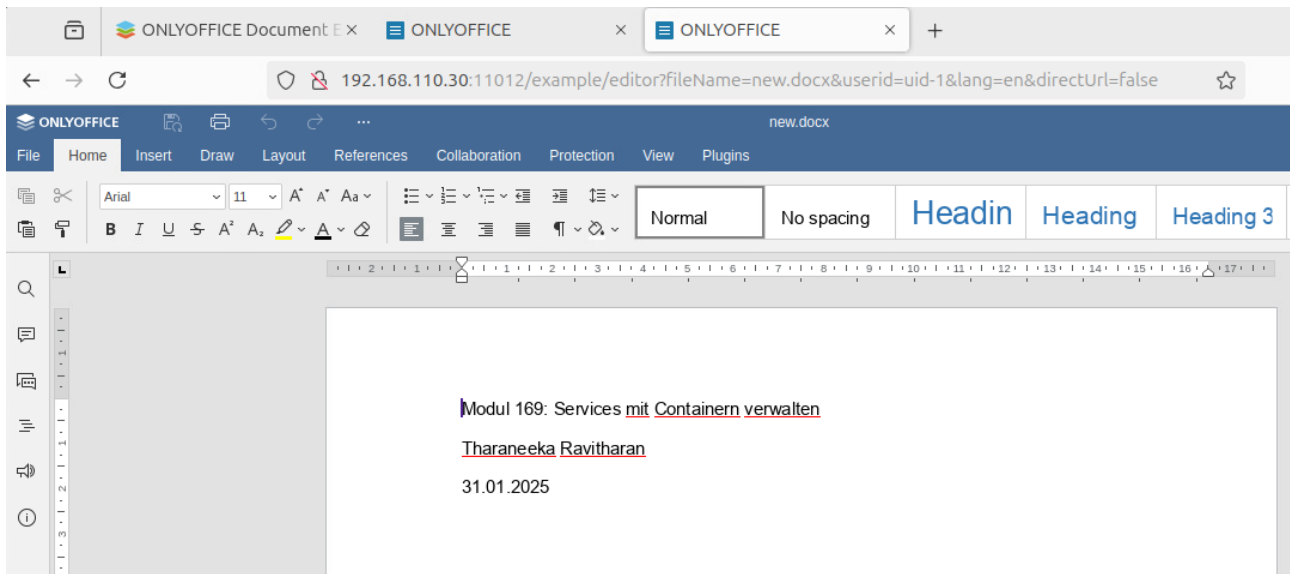
Virtualisierung und Containerisierung sind beide Technologien zur Bereitstellung isolierter Softwareumgebungen, unterscheiden sich jedoch grundlegend. Bei der Virtualisierung wird eine komplette virtuelle Maschine mit eigenem Betriebssystem auf einem Hypervisor ausgeführt, wodurch eine hohe Isolation, 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

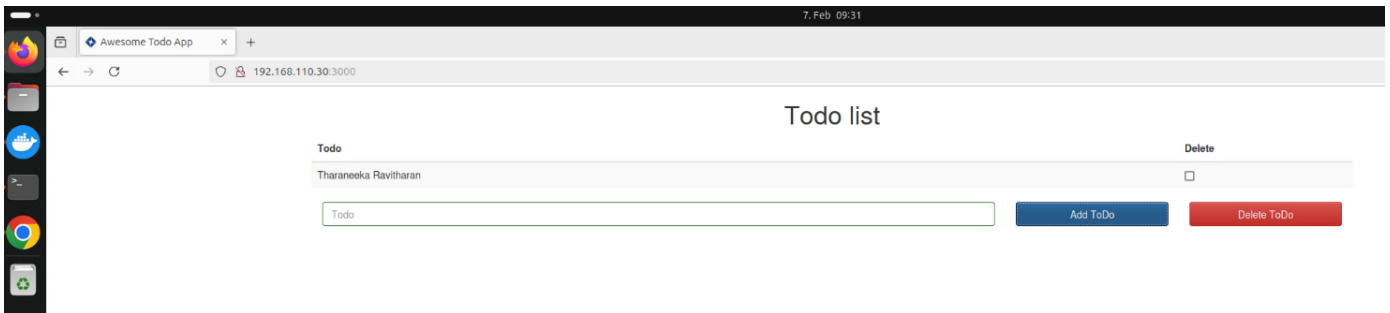
konsistente Bereitstellung, während Container die Flexibilität bieten, Anwendungen in unterschiedlichen Umgebungen auszuführen.

2.5 OnlyOffice

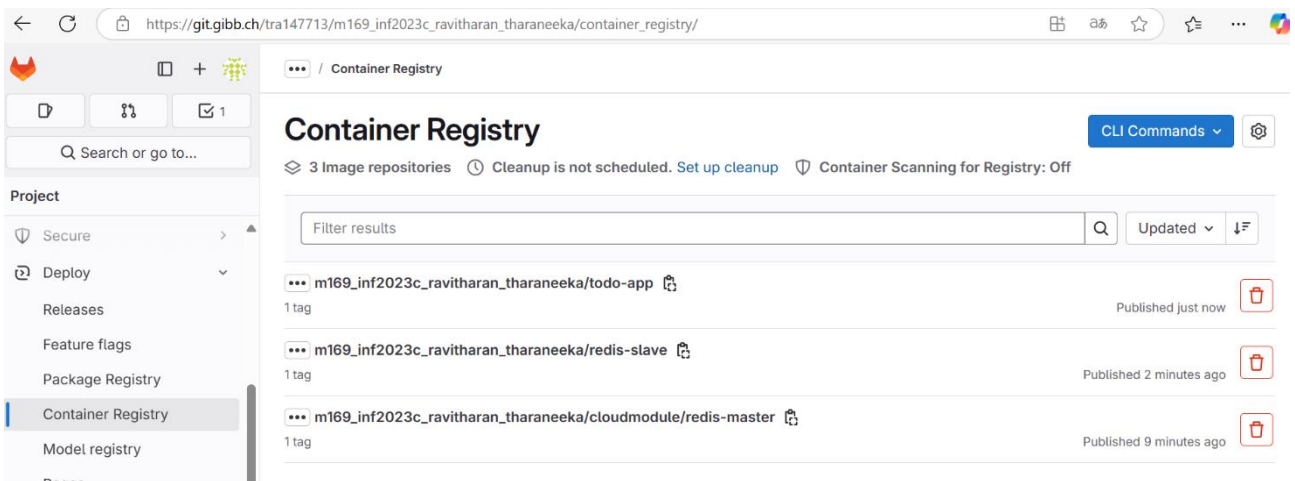


3 Tag 2, KW 6

3.1 Todo App Version 1

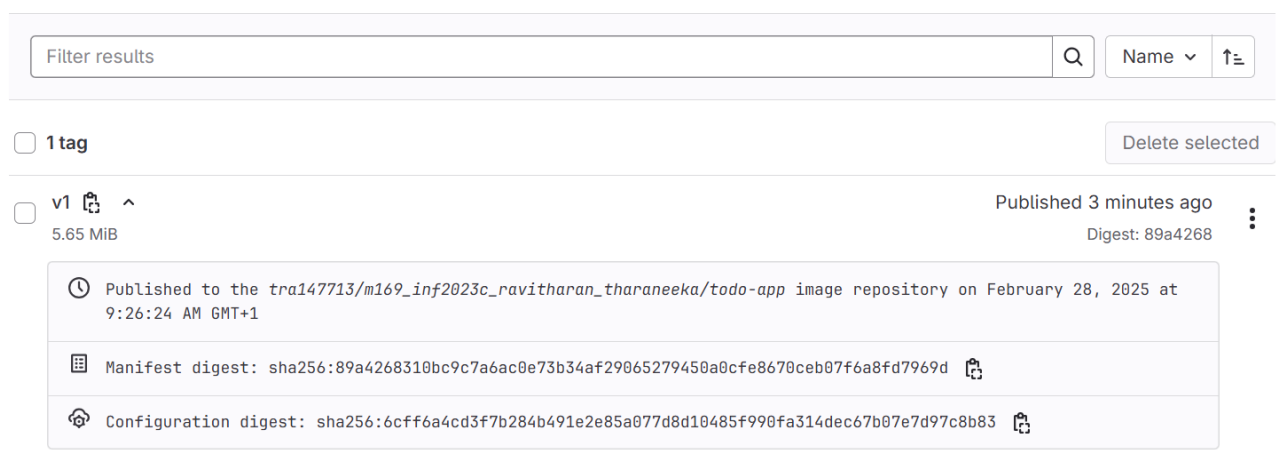


3.2 Images auf GitLab



todo-app

1 tag Cleanup disabled Created Feb 28, 2025 08:27



redis-slave

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

☐ 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: sha256:160a394b175bf6712a48a56bd4377ccb5f9490be88726329d1778ffa4d93556d

Configuration digest: sha256:9fecf9aa080a7a125e3a1569fed0595c2180e56e060211b80b11eae034770d63

cloudmodule/redis-master

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

☐ 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: sha256:0d8739e7b359377b83cbd3f5ff153cfe1b265ab0ba54a20ec6625b4b85ba76c9

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_xxxxxxx/redis-slave:v1
```



```
docker image tag todo-app:v1 git-registry.gibb.ch/xxxxx/ml69_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/ml69_inf2023c_xxxxxx/redis-master:v1
```

```
docker push git-registry.gibb.ch/xxxxx/ml69_inf2023c_xxxxxx/redis-slave:v1
```

```
docker push git-registry.gibb.ch/xxxxx/ml69_inf2023c_xxxxxx/todo-app:v1
```

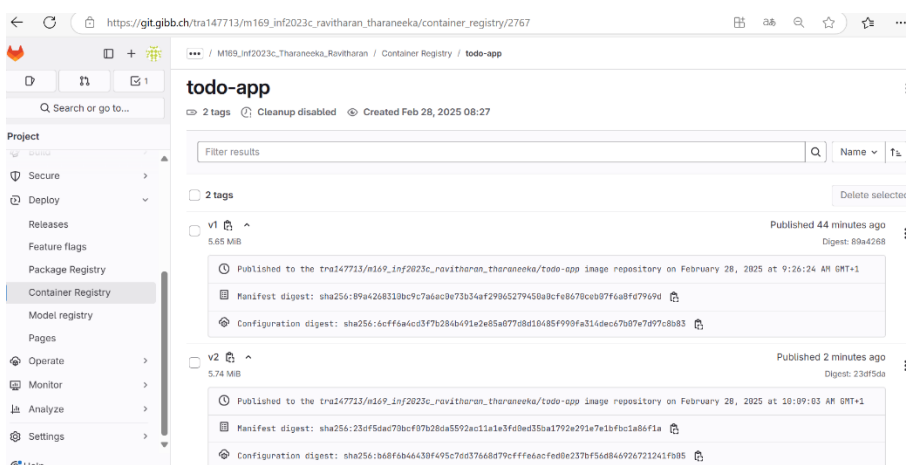
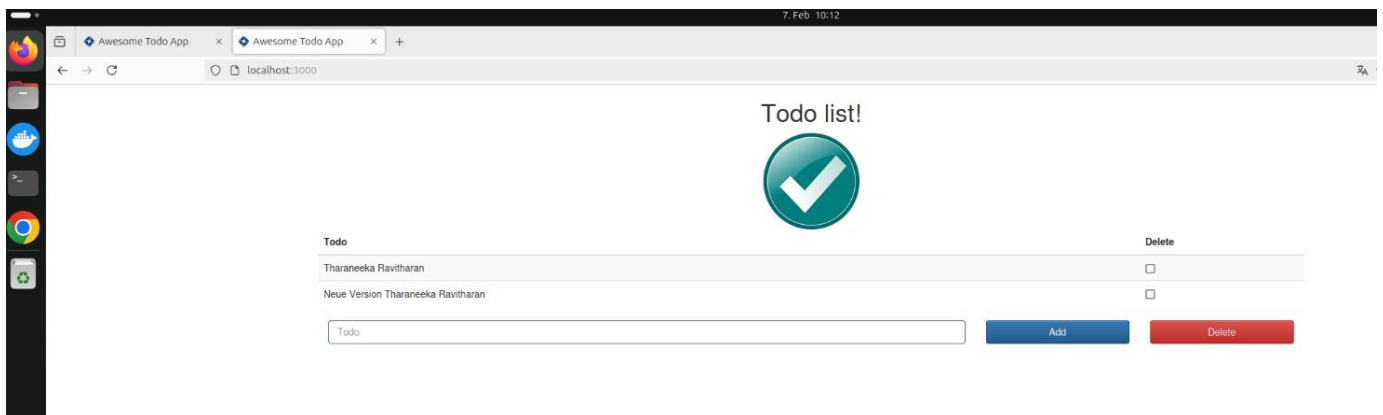
4. Diese Befehle laden die Docker-Images von der angegebenen Docker-Registry herunter. Wenn die Images in der Registry bereits vorhanden sind, werden sie auf dein System heruntergeladen, sodass man sie lokal verwenden kann:

```
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



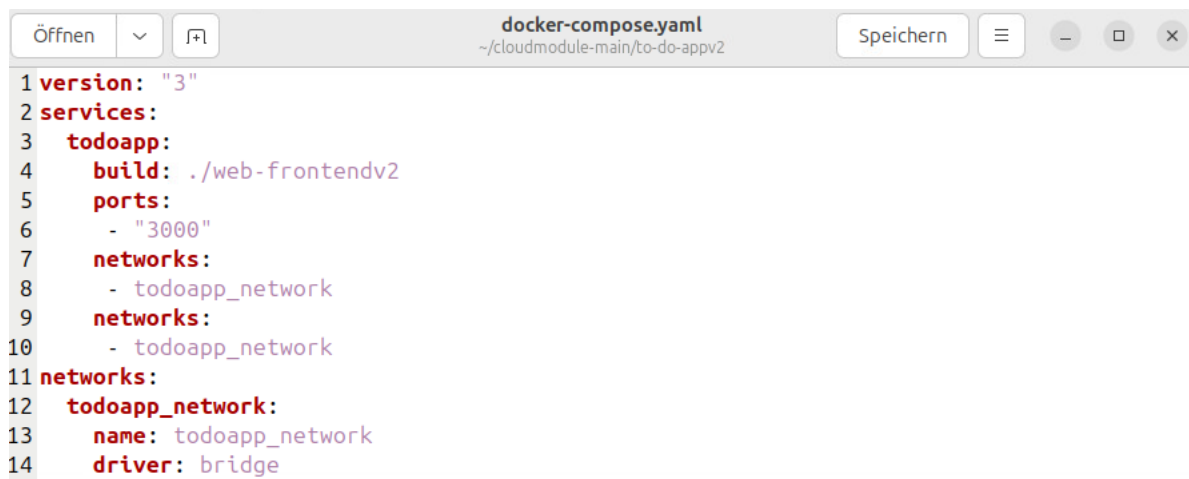
4 Tag 3, KW 7

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 To-do-App Version 2 mit Docker Compose

1. Man muss die zweite Version (to-do-appv2) bauen und den alten Container ersetzen.
2. Docker-Compose File muss so geändert werden, dass alles mit dem Docker Netzwerk verbunden 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
```

3. Neuen Container mit to-do-appv2 bauen:
 - a. In das Verzeichnis der neuen Version navigieren:
`cd path/to/to-do-appv2/web-frontendv2`
 - b. Dann das neue Frontend-Image mit einer neuen Version (v2) bauen:
`docker build -t todo-app:v2`
4. 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`
5. Die neue Version mit dem gleichen Netzwerk und Port starten:
`docker run --net=todoapp_network --name=frontend -d -p 3000:3000 todo-app:v2`
6. Überprüfen, ob alles funktioniert
<http://localhost:3000> öffnen und prüfen, ob die neue Version geladen wird.

```

vmadmin@11244-vmLP1:~/cloudmodule-main$ cd to-do-appv2
vmadmin@11244-vmLP1:~/cloudmodule-main/to-do-appv2$ ls
docker-compose.yaml  README.md  web-frontendlv2
vmadmin@11244-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@sha256:65a2763f593ae85fab3b5406dc9e80f744ec5b449f269b699b5efd37a07ad32e
=> => resolve docker.io/library/alpine:3.16.2@sha256:65a2763f593ae85fab3b5406dc9e80f744ec5b449f269b699b5efd37a07ad32e
=> [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 sha256:2368fa2f9557120ead0d31e5a3f8d67503779afda2c76b4da23961044747161c
=> => exporting config sha256:10a001759eedefeb830cf88618ce3dea4cee2ea740516e7596df04e8d51afe1
=> => exporting attestation manifest sha256:b5b1e765f607face7b201c5de4d89bb0818d4603b1c83b1072ab31c0ad5f7085
=> => exporting manifest list sha256:4ecc0fbaee86017eb1b402d5124f6e55bece0a30ff83c2b4b5720f0f8ff1bf7d
=> => 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/tg5o2z5dmpdoi40e62cd30s60
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@11244-vmLP1:~/cloudmodule-main/to-do-appv2$ docker ps

```

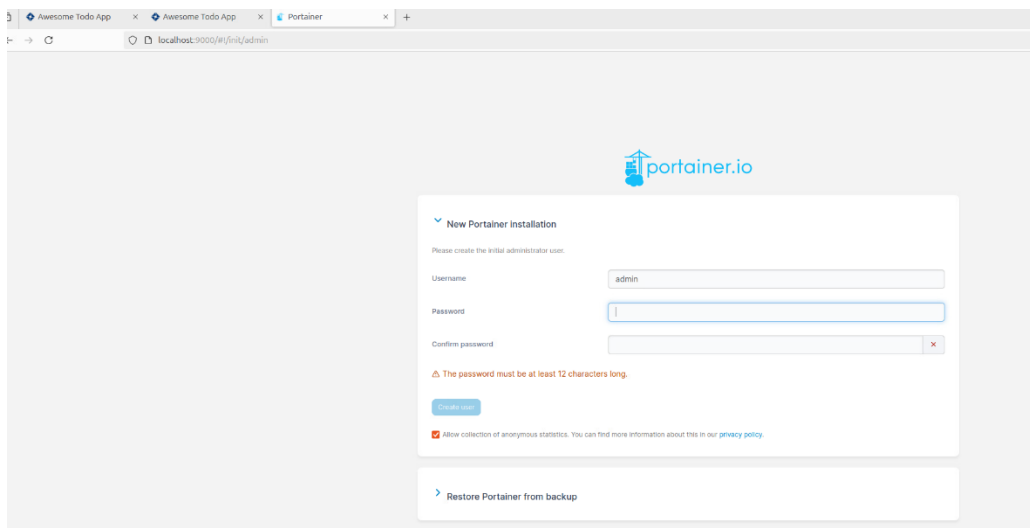
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
15d7312bf07b	to-do-appv2_todoapp	"./todo-app"	22 seconds ago	Up 21 seconds	0.0.0.0:44893->3000/tcp	to-do-appv2_todoapp_1
8d63db507e54	to-do-appv1_todoapp	"./todo-app"	13 minutes ago	Up 13 minutes	0.0.0.0:41773->3000/tcp	to-do-appv1_todoapp_1
9b627a211a01	to-do-appv1_redis-slave	"docker-entrypoint.s..."	13 minutes ago	Up 13 minutes	6379/tcp	to-do-appv1_redis-slave_1
3bad090e521	to-do-appv1_redis-master	"docker-entrypoint.s..."	13 minutes ago	Up 13 minutes	6379/tcp	to-do-appv1_redis-master_1
cb656dc36dc	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

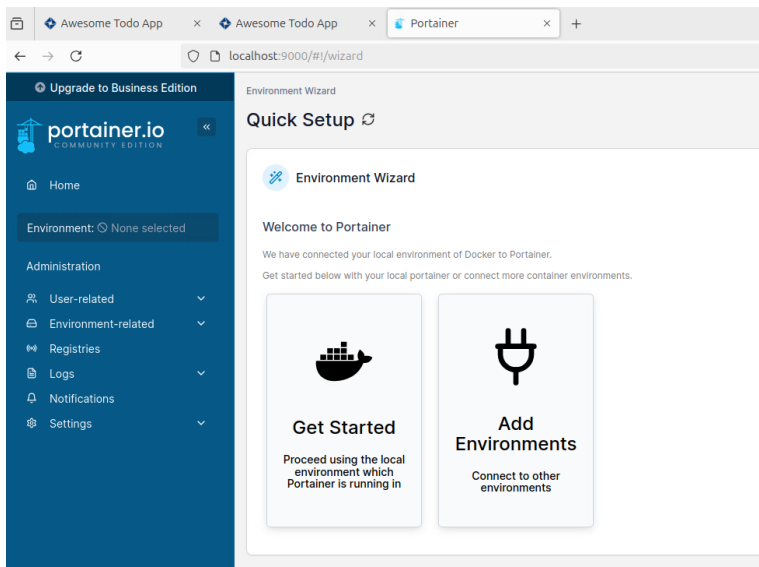
```

vmadmin@11244-vmLP1:~/cloudmodule-main/to-do-appv2$

```

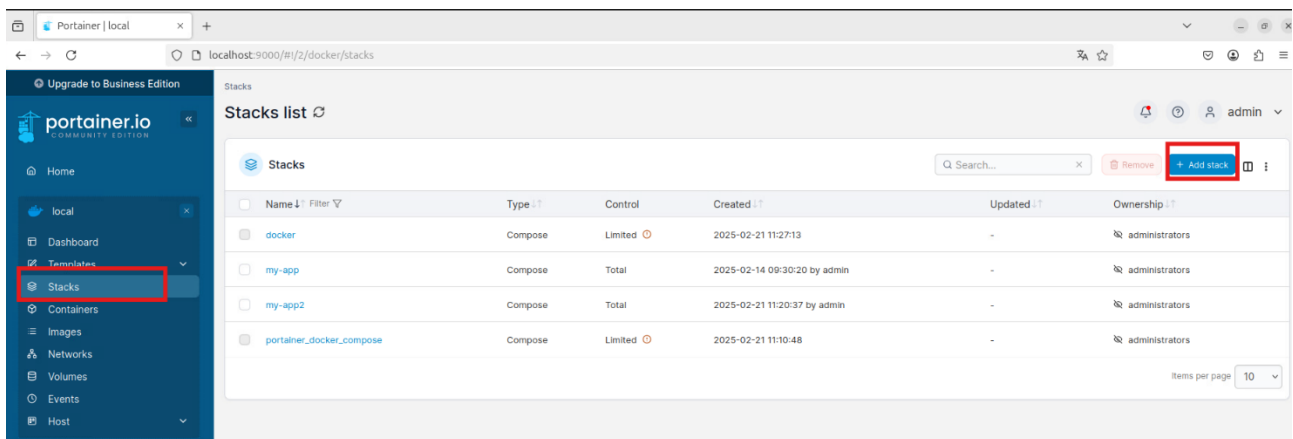
4.3 Portainer



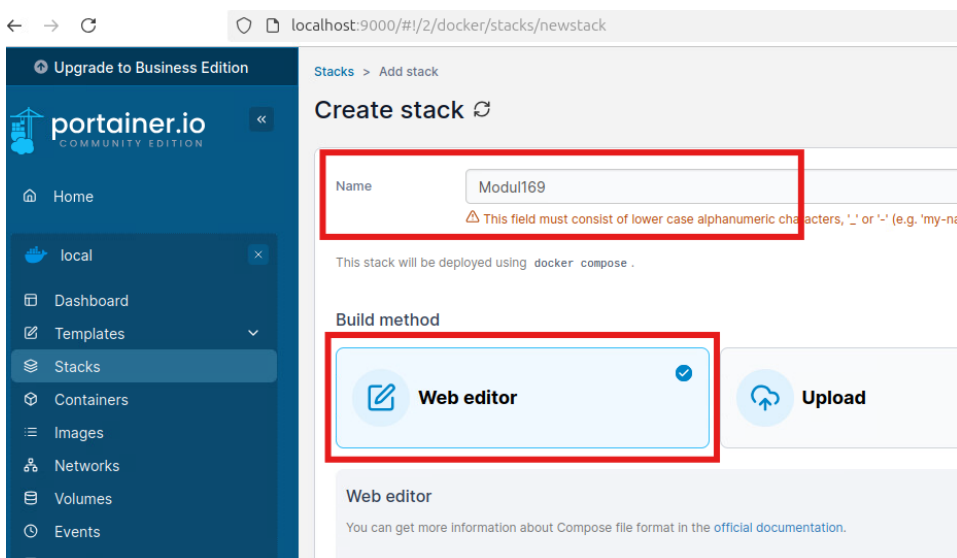


4.4 App via Portainer installiert

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:

The screenshot shows the Portainer.io interface. On the left is a sidebar with navigation options: Home, local, Dashboard, Templates, Stacks, Containers, Images, Networks, Volumes, Events, Host, Administration, User-related, and Environment-related. The main area is titled 'Build method' and has two tabs: 'Web editor' (selected) and 'Upload'. Below the tabs, the 'Web editor' section contains a text area with a Docker Compose file. The file content is as follows:

```
1 version: "3"
2 services:
3   todoapp:
4     image: tra147713/m169_inf2023c_ravitharan_tharaneeka/todo-app:v1
5     ports:
6       - "3000"
7     depends_on:
8       - redis-master
9       - redis-slave
10  redis-slave:
11    image: tra47713m169_inf2023c_ravitharan_tharaneeka/redis-slave:v1
12    depends_on:
13      - redis-master
14  redis-master:
15    image: tra147713/m169_inf2023c_ravitharan_tharaneeka/redis-master:v1
```

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

The screenshot shows the Portainer.io interface after the Docker Compose file has been pasted. The sidebar is the same. The main area shows the 'Environment variables' section, which includes a 'stack.env file operation' box, an 'Advanced mode' checkbox, and an 'Access control' section. At the bottom, the 'Actions' section contains a button labeled 'Deploy the stack', which is highlighted with a red rectangle.

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

The screenshot shows the Portainer.io interface with the 'Stack details' page for a stack named 'my-app'. The left sidebar shows the navigation menu with 'Stacks' selected. The main content area shows the 'Stack details' section with a 'Stack duplication / migration' form. Below this, there is a table of containers for the 'my-app' stack.

Name	State	Quick Actions	Stack	Image	Created	IP Address	Published Ports	Ownership
my-app-redis-master-1	running	[Stop] [Restart] [Kill] [Pause] [Resume] [Remove]	my-app	git-registry.glb.ch/thomas.staub/cloudmodules/redis-master:v1	2025-02-14 09:30:21	172.20.0.2	-	administrators
my-app-redis-slave-1	running	[Stop] [Restart] [Kill] [Pause] [Resume] [Remove]	my-app	git-registry.glb.ch/thomas.staub/cloudmodules/redis-slave:v1	2025-02-14 09:30:22	172.20.0.3	-	administrators
my-app-todoapp-1	running	[Stop] [Restart] [Kill] [Pause] [Resume] [Remove]	my-app	git-registry.glb.ch/thomas.staub/cloudmodules/todo-app:v1	2025-02-14 09:30:22	172.20.0.4	43637:3000	administrators

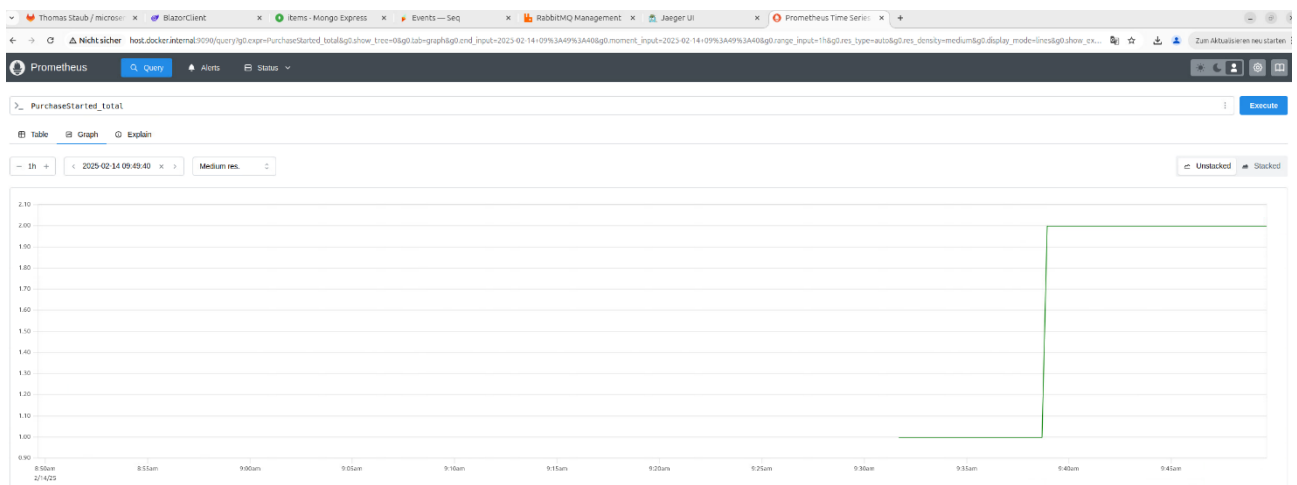
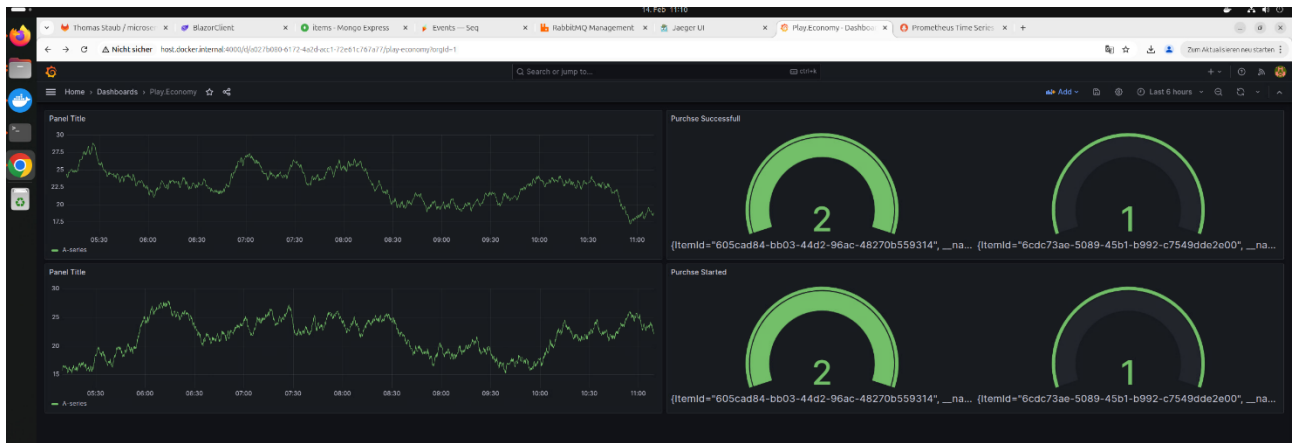
4.5 Shop mit Docker Compose

The screenshot shows the Portainer.io interface with the 'Containers' page. The left sidebar shows the navigation menu with 'Containers' selected. The main content area shows a table of containers.

Name	State	Quick Actions	Stack	Image	Created	IP Address	Published Ports
catalog	running	[Stop] [Restart] [Kill] [Pause] [Resume] [Remove]	docker	git-registry.glb.ch/thomas.staub/microservices/play.catalog:1.0	2025-02-21 11:27:12	172.21.0.6	5000:5000
epic_gauss	running	[Stop] [Restart] [Kill] [Pause] [Resume] [Remove]	-	redis-slave:v1	2025-02-21 10:08:36	172.17.0.2	-
frontend	running	[Stop] [Restart] [Kill] [Pause] [Resume] [Remove]	-	todo-app:v1	2025-02-21 10:08:26	172.18.0.3	3000:3000
frontendv2	running	[Stop] [Restart] [Kill] [Pause] [Resume] [Remove]	docker	git-registry.glb.ch/thomas.staub/microservices/play.frontendv2:1.0	2025-02-21 11:27:12	172.21.0.3	5008:80
grafana	running	[Stop] [Restart] [Kill] [Pause] [Resume] [Remove]	docker	docker.grafana	2025-02-21 11:27:13	172.21.0.10	4000:3000
identity	running	[Stop] [Restart] [Kill] [Pause] [Resume] [Remove]	docker	git-registry.glb.ch/thomas.staub/microservices/play.identity:1.0	2025-02-21 11:27:12	172.21.0.4	5002:5002
inventory	running	[Stop] [Restart] [Kill] [Pause] [Resume] [Remove]	docker	git-registry.glb.ch/thomas.staub/microservices/play.inventory:1.0	2025-02-21 11:27:12	172.21.0.2	5004:5004
jaeger	running	[Stop] [Restart] [Kill] [Pause] [Resume] [Remove]	docker	jaegertracing/all-in-one	2025-02-21 11:27:12	172.21.0.11	6831:6831 14268:14268 16886:16886 14250:14250
mongo	running	[Stop] [Restart] [Kill] [Pause] [Resume] [Remove]	docker	mongo	2025-02-21 11:27:12	172.21.0.9	27017:27017
mongo-express	running	[Stop] [Restart] [Kill] [Pause] [Resume] [Remove]	docker	mongo-express	2025-02-21 11:27:24	172.21.0.13	8080:8081

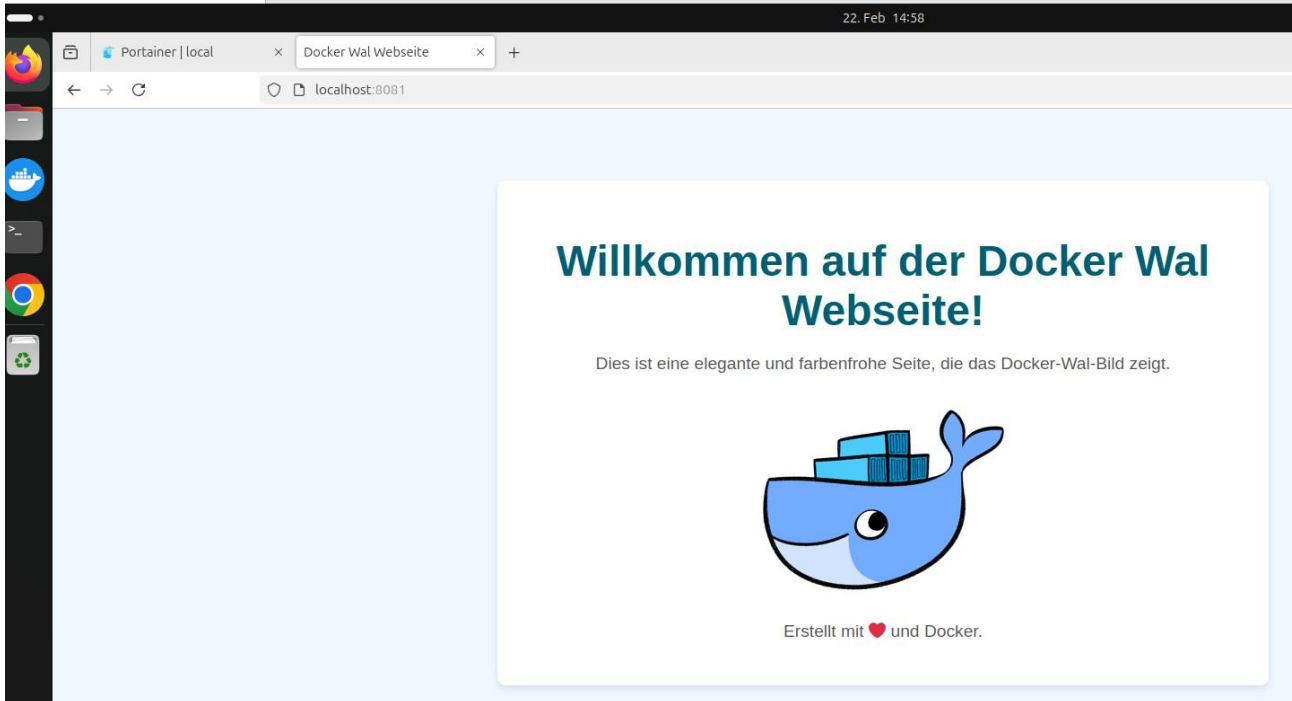
The screenshot shows the Portainer.io interface with the 'Store' page. The left sidebar shows the navigation menu with 'Store' selected. The main content area shows a table of items for sale.

Id	Name	Description	Quantity	Action
aec82f4f-c242-4f12-9770-ab30c4642e01	Docker Kurs 1		1	PURCHASE
2efe0435-bba6-40bb-b64c-b1193ccbac61	Kubernetes Kurs		2	PURCHASE

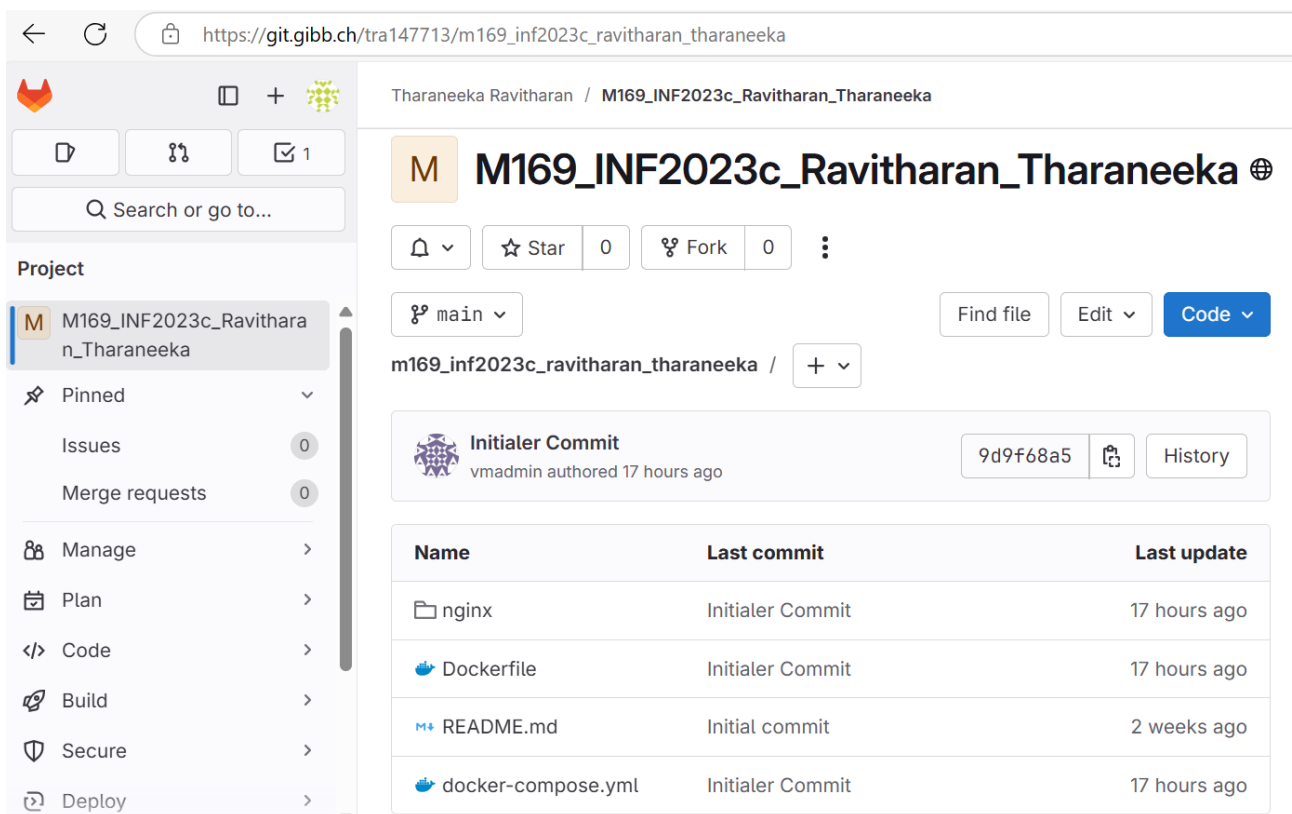


5 Übungsprojekt Webserver

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:



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
```

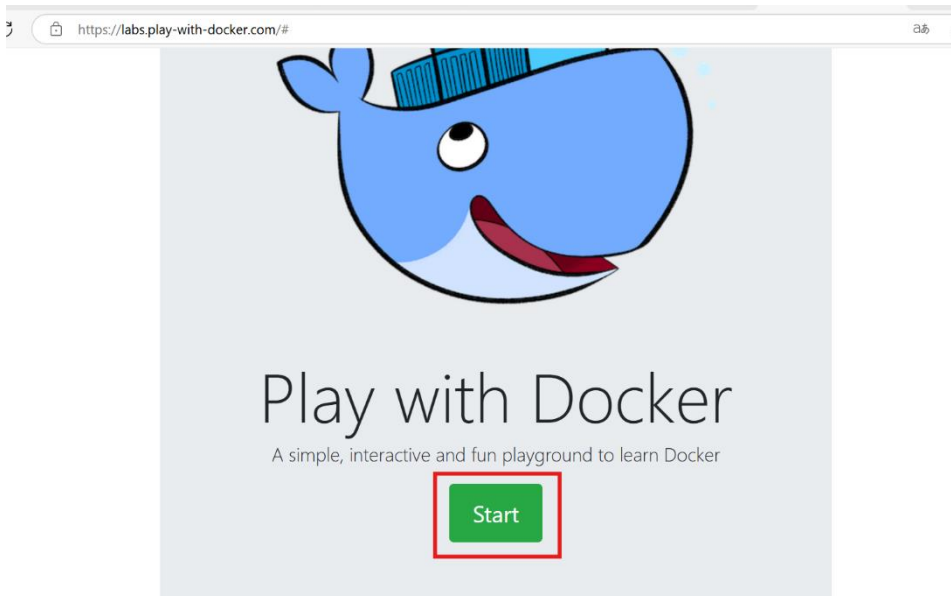
3. Den Container neu starten, damit die Änderungen aktiv werden:

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

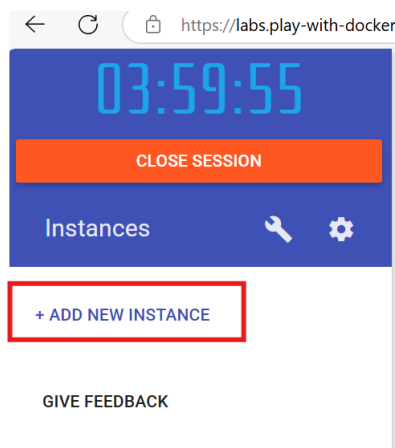
5.2 Anleitung für Lehrer

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
=> [webserver internal] load build definition from Dockerfile                                docker:default
=> => 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

7 Tag 5, KW 9

7.1 Was ist Kubernetes?

Kubernetes ist eine Open-Source-Plattform zur Automatisierung der Bereitstellung, Skalierung und Verwaltung von containerisierten Anwendungen. Es ermöglicht die Orchestrierung von Containern über mehrere Server hinweg und sorgt für eine effiziente Nutzung von Ressourcen, hohe Verfügbarkeit und einfache Skalierung

7.2 Was sind Microservices?

Microservices sind ein Architekturmuster, bei dem eine Anwendung in kleine, eigenständige Dienste aufgeteilt wird. Jeder Microservice ist unabhängig, erfüllt eine spezifische Funktion und kommuniziert über APIs mit anderen Microservices. Diese Architektur erleichtert Skalierbarkeit, Wartung und Weiterentwicklung von Anwendungen.

7.3 Vergleich der lightweight Kubernetes Anwendungen

Merkmal	K3s	MicroK8s	Minikube	Kubeadm
Entwickler	Rancher (SUSE)	Canonical (Ubuntu)	Kubernetes Community	Kubernetes Community
Grösse	< 100 MB	~200 MB	Variabel (je nach Hypervisor)	Variabel
Optimierung	Ressourcenarm, ideal für Edge & IoT	Einfach & modular für Dev/CI/CD	Lokale Tests & Entwicklung	Produktionscluster
Installation	Sehr einfach (Single Binary)	Einfach (Snap)	Einfach (CLI)	Komplex (manuelle Konfiguration erforderlich)
Cluster-Modus	Unterstützt Multi-Node	Unterstützt Multi-Node	Single-Node (Hauptsächlich für Tests)	Ja, für produktionsreife Cluster
Datenbank	SQLite oder etcd	etcd	etcd	etcd
Container-Runtime	Containerd (Standard)	Containerd, CRI-O, Docker	Docker, CRI-O, Containerd	Flexibel (Docker, Containerd, CRI-O)
Add-ons	Eingebaute Helm & Traefik-Integration	Modulares Add-on-System (DNS, Istio, etc.)	Unterstützt verschiedene Plugins	Manuell konfigurierbar
Ressourcenverbrauch	Sehr niedrig	Mittel	Mittel bis hoch	Hoch (vollwertiger Kubernetes-Cluster)
Sicherheit	Gut (automatische TLS-Zertifikate)	Gut (Ubuntu-Sicherheitsupdates)	Niedrig (für Entwicklung)	Sehr hoch (ideal für Produktion)
Upgrade & Wartung	Automatische Upgrades möglich	Automatische Updates über Snap	Manuelles Update erforderlich	Manuelles Update erforderlich
Einsatzgebiete	Edge, IoT, kleine	Lokale Entwicklung, kleine Cluster	Entwicklung, Testing, Experimente	On-Premises, Cloud-Cluster,

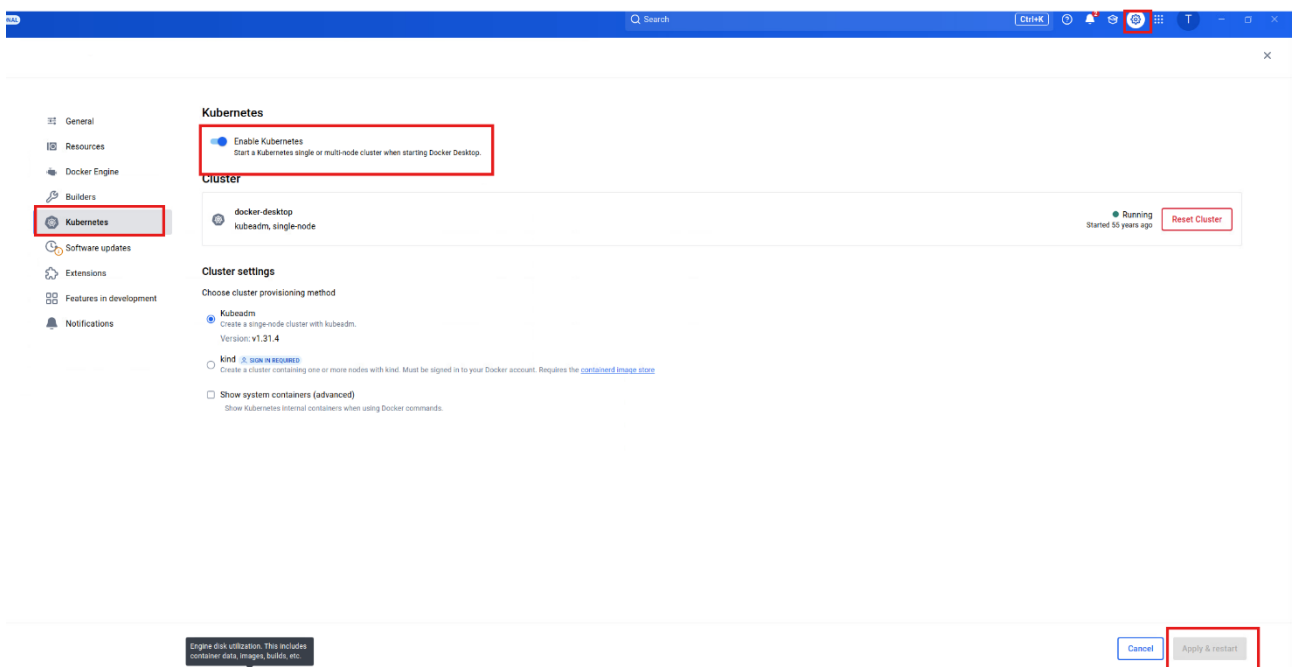
	Kubernetes-Cluste			Produktionsumgebungen
--	-------------------	--	--	-----------------------

Wann sollte man welche Lösung wählen?

- **K3s** → Perfekt für Edge-Computing, IoT, Embedded-Systeme oder kleine Server mit wenig Ressourcen.
- **MicroK8s** → Ideal für lokale Entwicklung oder kleine Cluster, wenn einfache Installation und Verwaltung gewünscht ist.
- **Minikube** → Bester Kandidat für lokale Tests, Entwicklung und das Erlernen von Kubernetes auf einem einzelnen Rechner.
- **kubeadm** → Die richtige Wahl für produktive Cluster in Rechenzentren oder in der Cloud, wenn volle Kontrolle über die Konfiguration benötigt wird.

7.4 Installation Kubernetes

1. Auf Docker Desktop Kubernetes aktivieren:



2. Im Terminal wird das installiert:

```
sudo snap install kubectl --classic
```

3. Verknüpfung mit der Docker Instanz:

```
kubectl config get-contexts
kubectl config use-context docker-desktop
```

4. Die Installation sollte nun funktioniert haben

7.5 Lens Verbindung mit dem Server

The screenshot shows the Kubernetes Lens interface. The left sidebar is titled 'KUBERNETES CLUSTERS' and lists 'Local Kubeconfigs', 'demo-k8s', and 'docker-desktop'. The 'Pods' view is selected for the 'docker-desktop' cluster. The main table displays 11 items in the 'to-do-app' namespace. The table columns are Name, Namespace, Cont..., CPU, Memor, Restart, and Controlle.

Name	Namespace	Cont...	CPU	Memor	Restart	Controlle
redis-master-2qw5d	to-do-app	■	N/A	N/A	0	Replicati
redis-slave-6dzfg	to-do-app	■	N/A	N/A	0	Replicati
redis-slave-9942c	to-do-app	■	N/A	N/A	0	Replicati
redis-slave-n78tl	to-do-app	■	N/A	N/A	0	Replicati
redis-slave-nzhzr	to-do-app	■	N/A	N/A	0	Replicati
redis-slave-zcw59	to-do-app	■	N/A	N/A	0	Replicati
todo-app-deployment-5b88c7454	to-do-app	■	N/A	N/A	0	ReplicaS
todo-app-deployment-5b88c7454	to-do-app	■	N/A	N/A	0	ReplicaS
todo-app-deployment-5b88c7454	to-do-app	■	N/A	N/A	0	ReplicaS
todo-app-deployment-5b88c7454	to-do-app	■	N/A	N/A	0	ReplicaS
todo-app-deployment-5b88c7454	to-do-app	■	N/A	N/A	0	ReplicaS

8 Tag 6, KW 10

8.1 Raft-Konsens-Algorithmus

8.1.1 Was ist der Raft-Konsens-Algorithmus?

Der **Raft-Konsens-Algorithmus** ist ein Algorithmus, der entwickelt wurde, um sicherzustellen, dass verteilte Systeme (wie ein Kubernetes-Cluster) konsistent bleiben, selbst wenn einige Knoten (Server) ausfallen oder Netzwerkprobleme auftreten. Raft wird vor allem in verteilten Systemen verwendet, die eine **hochverfügbare** und **fehlerresistente** Verwaltung von Daten erfordern. Der Algorithmus sorgt dafür, dass alle Knoten im Cluster denselben Zustand haben, was zu einer einheitlichen Sicht auf die Daten führt.

Raft erreicht Konsens durch eine **Leader-Wahl**, bei der ein Knoten im Cluster als **Leader** fungiert, der alle Änderungen (z.B. Datenbankeinträge oder Statusänderungen) koordiniert. Alle anderen Knoten im Cluster sind **Follower**, die dem Leader folgen.

8.1.2 Grundprinzipien des Raft-Algorithmus:

1. **Leader-Wahl:** Ein Knoten wird zufällig als Leader gewählt. Der Leader ist verantwortlich für die Koordination der Transaktionen.
2. **Log-Replikation:** Der Leader empfängt alle Anfragen und führt sie aus, indem er die Änderungen in seinem **Log** protokolliert. Diese Änderungen werden dann an die Follower repliziert.
3. **Sicherstellung der Konsistenz:** Jeder Follower muss die Änderungen vom Leader empfangen und in seinem Log speichern. Erst wenn die Mehrheit der Knoten (also eine **Quorum**) die Änderung bestätigt, wird die Änderung als endgültig betrachtet.
4. **Sicherstellung der Fehlerresistenz:** Wenn der Leader ausfällt, wird ein neuer Leader durch einen Wahlprozess bestimmt, sodass das System weiterhin funktioniert.

8.1.3 Warum sollte ein Cluster eine ungerade Anzahl an Servern haben?

Die Wahl einer ungeraden Anzahl von Knoten in einem Cluster ist eine wichtige Best Practice beim Einsatz des Raft-Konsens-Algorithmus und anderer Konsensverfahren wie Paxos. Der Grund dafür liegt in der Quorum-Regel.

Quorum und Konsens:

- In Raft wird ein **Quorum** als die Mehrheit der Knoten im Cluster definiert. Um eine Änderung als gültig zu betrachten, müssen mindestens **mehr als die Hälfte** der Knoten die Änderung bestätigen.
- Wenn der Cluster eine **gerade Anzahl** an Knoten hat, könnte es im Falle eines Ausfalls von Knoten zu **Gleichstand** kommen, wodurch der Konsens nicht erreicht werden kann. Das bedeutet, dass das System nicht mehr entscheiden kann, ob eine Änderung durchgeführt wird oder nicht.

8.2 To-Do-App in Kubernetes

1. Namespace erstellen (damit alle Ressourcen nicht in den „Default“ Namespace landen):

```
kubectl create namespace to-do-app
```

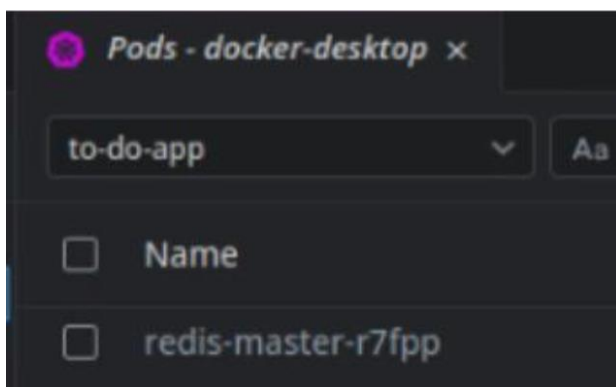
2. Den Redis-Master Replication-Controller starten:

```
kubectl create -f redis-master-controller.yaml -n to-do-app
```

```
redis-master-controller.yaml
1  kind: ReplicationController
2  apiVersion: v1
3  metadata:
4    name: redis-master
5    labels:
6      name: redis-master
7  spec:
8    replicas: 1
9    selector:
10     name: redis-master
11   template:
12     metadata:
13       labels:
14         name: redis-master
15     spec:
16       containers:
17         - name: redis-master
18           image: staubth/redis-master:v1
19           ports:
20             - containerPort: 6379
21             protocol: TCP
22
```

3. Überprüfen, ob der Redis-Master Pod gestartet wurde:

```
Kubectl get pods -n to-do-app
```



4. Redis-Master Service starten:

```
kubectl create -f redis-master-service.yaml -n to-do-app
```

5. Redis-Slave Controller starten:

```
kubectl create -f redis-slave-controller.yaml -n to-do-app
```

6. Redis-Slave Service starten:

```
kubectl create -f redis-slave-service.yaml -n to-do-app
```

7. Es sollte dann so aussehen:

```
vmadmin@li244-vmLP1:~/kubernetes/cloudmodule/to-do-app-k8s$ kubectl get service -n to-do-app
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
redis-master	ClusterIP	10.104.90.149	<none>	6379/TCP	111m
redis-slave	ClusterIP	10.96.169.214	<none>	6379/TCP	50s

8. To-do-app starten:

```
kubectl create -f todo-app-deploy.yaml -n to-do-app
```

```
todo-app-deploy.yaml
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: todo-app-deployment
5    labels:
6      app: todo-app
7  spec:
8    replicas: 3
9    selector:
10     matchLabels:
11       app: todo-app
12    template:
13     metadata:
14       labels:
15         app: todo-app
16     spec:
17       containers:
18       - name: todo-app
19         image: staubth/todo-app:v1
20         ports:
21         - containerPort: 3000
22         protocol: TCP
```

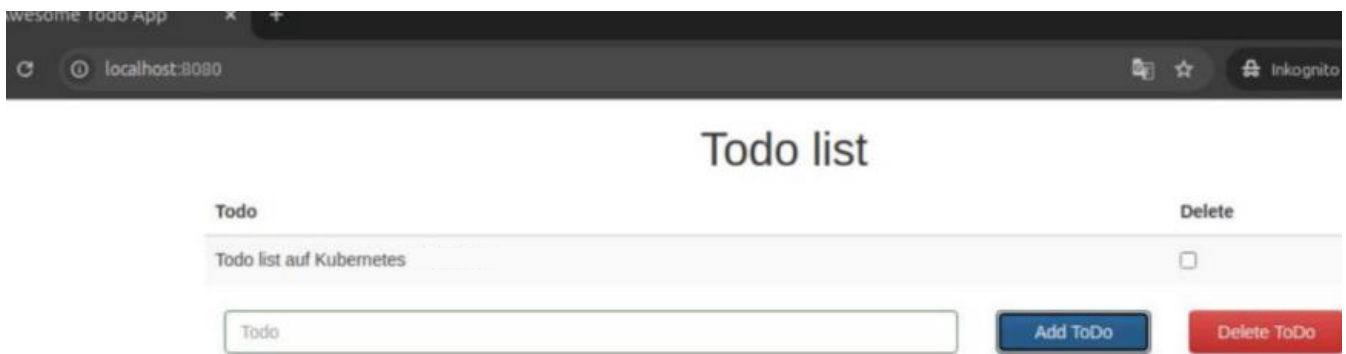
9. Service einrichten:

```
kubectl create -f todo-app-service-deploy.yaml -n to-do-app
```

10. To-do-app starten:

```
kubectl -n to-do-app port-forward svc/todo-app-service 8080:80
```

11. Im Browser localhost:8080 eingeben. Dann sollte das erscheinen:



8.3 Self-Healing

Self Healing bezieht sich auf die Fähigkeit eines Kubernetes-Clusters, automatisch Probleme zu erkennen und zu beheben, um sicherzustellen, dass die Anwendung immer verfügbar bleibt. Wenn beispielsweise ein Pod ausfällt oder einen Fehler aufweist, sorgt Kubernetes dafür, dass der Pod automatisch neu gestartet oder durch einen neuen ersetzt wird. Kubernetes überwacht kontinuierlich die **Pods** und **Nodes** und stellt sicher, dass die gewünschte Anzahl an Instanzen (Replikate) stets läuft.

8.4 Scale up und Scale down

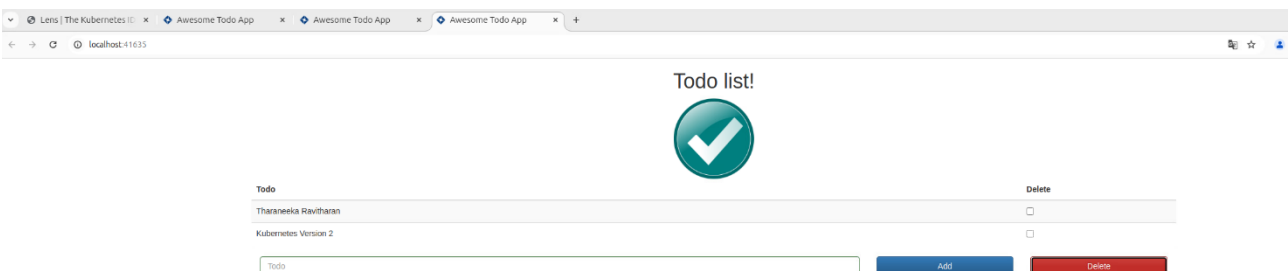
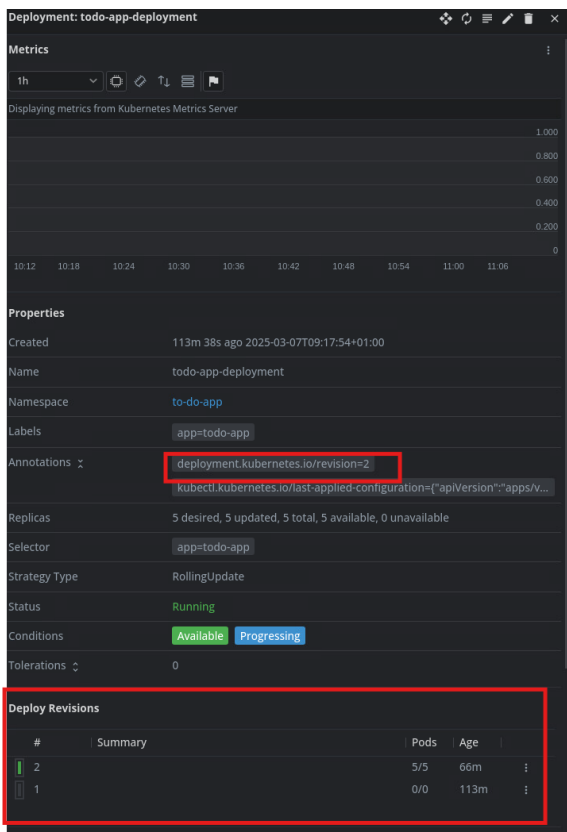
Scale Up und Scale Down sind Begriffe, die sich auf die Anpassung der Anzahl von Pods in einem Kubernetes-Cluster beziehen, um auf Laständerungen oder Anforderungen zu reagieren.

Scale Up: Dieser Vorgang erhöht die Anzahl der laufenden Pods, um die Verarbeitungskapazität zu steigern. Dies kann durch das Erhöhen der Replikate eines Deployments erfolgen. Ein Horizontal Pod Autoscaler (HPA) kann automatisch mehr Pods hinzufügen, wenn die Last steigt.

Scale Down: Dieser Vorgang reduziert die Anzahl der laufenden Pods, um Ressourcen zu sparen, wenn die Last sinkt. Auch dies kann manuell oder durch einen Horizontal Pod Autoscaler erfolgen, der Pods bei geringer Last entfernt.

8.5 Rolling Updates bei der Todo-App Version 2

Name	Namespace	Cont...	CPU	Memor	Restart	Controlled B	Node	QoS	Age	Status
redis-master-2qw5d	to-do-app		N/A	N/A	0	ReplicationCont	docker-desktop	BestEffort	121m	Running
redis-slave-6dzfg	to-do-app		N/A	N/A	0	ReplicationCont	docker-desktop	BestEffort	71m	Running
redis-slave-9942c	to-do-app		N/A	N/A	0	ReplicationCont	docker-desktop	BestEffort	71m	Running
redis-slave-n78tl	to-do-app		N/A	N/A	0	ReplicationCont	docker-desktop	BestEffort	71m	Running
redis-slave-nzhzr	to-do-app		N/A	N/A	0	ReplicationCont	docker-desktop	BestEffort	71m	Running
redis-slave-zcw59	to-do-app		N/A	N/A	0	ReplicationCont	docker-desktop	BestEffort	118m	Running
todo-app-deployment-5b88c7454	to-do-app		N/A	N/A	0	ReplicaSet	docker-desktop	BestEffort	64m	Running
todo-app-deployment-5b88c7454	to-do-app		N/A	N/A	0	ReplicaSet	docker-desktop	BestEffort	64m	Running
todo-app-deployment-5b88c7454	to-do-app		N/A	N/A	0	ReplicaSet	docker-desktop	BestEffort	62m	Running
todo-app-deployment-5b88c7454	to-do-app		N/A	N/A	0	ReplicaSet	docker-desktop	BestEffort	63m	Running
todo-app-deployment-5b88c7454	to-do-app		N/A	N/A	0	ReplicaSet	docker-desktop	BestEffort	63m	Running



8.6 Blue-Green Deployment

Blue-Green Deployment ist eine Strategie zur kontinuierlichen Bereitstellung von Software, bei der zwei separate Umgebungen (Blau und Grün) genutzt werden, um die Risiken bei der Veröffentlichung neuer Versionen einer Anwendung zu minimieren. Ziel ist es, eine Unterbrechung der Anwendung zu vermeiden und gleichzeitig sicherzustellen, dass eine stabile Version immer verfügbar ist.

Funktionsweise:

- 1. Blue Environment:** Die "blaue" Umgebung repräsentiert die derzeit in Produktion befindliche Version der Anwendung. Sie ist die aktive Umgebung, auf der die Benutzer zugreifen.
- 2. Green Environment:** Die "grüne" Umgebung enthält die neue Version der Anwendung, die bereit zur Veröffentlichung ist. Sie wird auf derselben Infrastruktur bereitgestellt, jedoch ohne aktiven Zugriff von den Endbenutzern.

3. **Umstellung (Switching):** Nachdem die grüne Umgebung erfolgreich getestet und validiert wurde, wird der Datenverkehr von der blauen auf die grüne Umgebung umgeschaltet. Das bedeutet, dass die grüne Umgebung jetzt die aktive Version wird, während die blaue Umgebung in den Hintergrund tritt.
4. **Rollback:** Falls nach dem Umschalten auf die grüne Umgebung Fehler auftreten, kann das System einfach wieder zur blauen Umgebung zurückgeschaltet werden, was ein Rollback ermöglicht. Dies stellt sicher, dass die Anwendung weiterhin stabil bleibt.

Vorteile des Blue-Green Deployments:

1. **Minimierte Ausfallzeiten:** Durch das Umschalten des Datenverkehrs zwischen den Umgebungen kann das Deployment nahezu ohne Ausfallzeiten erfolgen.
2. **Schnelles Rollback:** Bei Problemen mit der neuen Version kann schnell auf die alte Version zurückgeschaltet werden.
3. **Geringes Risiko:** Die grüne Umgebung kann gründlich getestet werden, bevor sie in den Live-Betrieb geht, was die Wahrscheinlichkeit von Fehlern reduziert.

Nachteile des Blue-Green Deployments:

1. **Zusätzliche Ressourcen:** Es müssen zwei Umgebungen gleichzeitig laufen (blau und grün), was mehr Ressourcen benötigt.
2. **Komplexität:** Das Verwalten von zwei parallelen Umgebungen und die automatische Umschaltung kann komplexer werden.

In Kubernetes kann Blue-Green Deployment mithilfe von **Deployments**, **Services** und **Ingresses** umgesetzt werden, wobei das Umschalten zwischen den Versionen oft durch das Ändern des Service-Targets oder durch die Nutzung von Ingress-Routing erfolgen kann.

9 Tag 7, KW 11

9.1 Cluster IP

In Kubernetes ist ClusterIP ein Servicetyp, der für die interne Kommunikation innerhalb eines Kubernetes-Clusters sorgt. Er ermöglicht es, eine Gruppe von Pods für andere Pods innerhalb desselben Clusters zugänglich zu machen, ist jedoch von außerhalb des Clusters nicht erreichbar. Dieser Servicetyp wird verwendet, wenn Dienste ausschließlich innerhalb des Clusters benötigt werden, beispielsweise für interne APIs oder Datenbanken.

Eine zentrale Eigenschaft von ClusterIP ist das automatische Load Balancing, wodurch der Datenverkehr gleichmäßig auf die Gruppe von Pods verteilt wird, die vom Service ausgewählt werden. Wird ein Service erstellt, ohne explizit einen Typ anzugeben, wird automatisch ClusterIP verwendet.

Ein ClusterIP-Service ist nur innerhalb des Clusters über seine DNS-Adresse oder IP erreichbar. Diese Art von Service wird häufig genutzt, um Microservices miteinander kommunizieren zu lassen, ohne sie für externe Zugriffe freizugeben.

9.2 Node IP

Die Node-IP ist die IP-Adresse eines einzelnen Nodes (Rechners) innerhalb eines Kubernetes-Clusters. Jede Node im Cluster hat eine eigene IP-Adresse, die für die interne Kommunikation zwischen den Nodes und für den Zugriff auf bestimmte Dienste, wie z.B. einen NodePort-Service, verwendet wird. Diese IP kann entweder eine private oder öffentliche IP-Adresse sein, abhängig von der Netzwerkinfrastruktur und der Umgebung, in der der Cluster betrieben wird.

Im Falle eines NodePort-Services ermöglicht die Node-IP den externen Zugriff auf einen Dienst über einen bestimmten Port, der auf allen Nodes des Clusters verfügbar gemacht wird. Die Node-IP ist somit eine wichtige Adresse, um direkt mit einem Node zu kommunizieren, insbesondere in Szenarien, in denen externe Systeme oder Benutzer auf einen Service im Cluster zugreifen müssen.

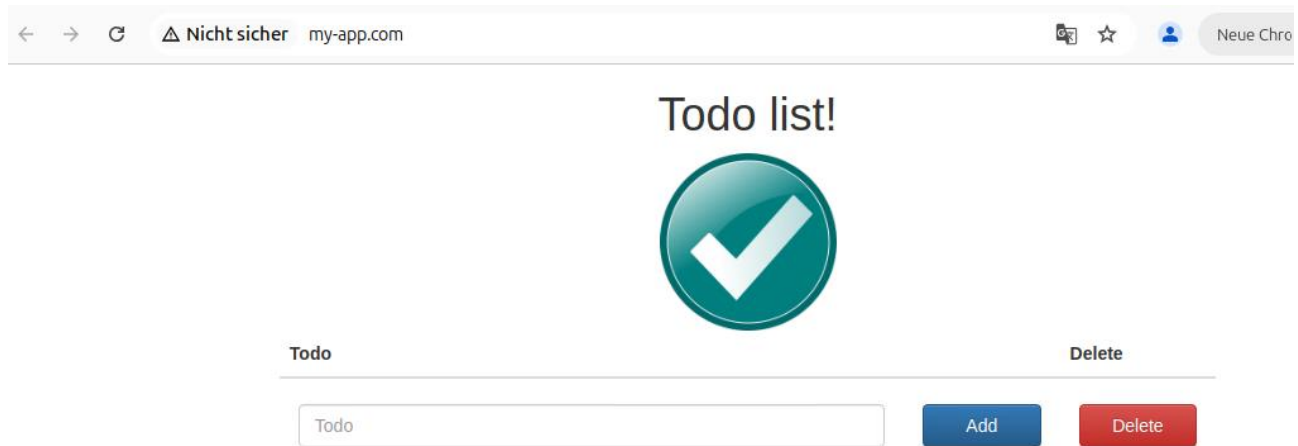
9.3 LoadBalancer

Ein LoadBalancer in Kubernetes ist ein Servicetyp, der es ermöglicht, einen Dienst sowohl intern als auch extern über eine öffentliche IP-Adresse zugänglich zu machen. Er stellt einen externen Load Balancer bereit, der den eingehenden Datenverkehr auf verschiedene Pods verteilt, um eine hohe Verfügbarkeit und Skalierbarkeit zu gewährleisten. Der Load Balancer sorgt dafür, dass der Datenverkehr gleichmäßig auf die verfügbaren Pods verteilt wird, was zu einer besseren Performance und Ausfallsicherheit führt.

Der LoadBalancer-Service wird häufig in Cloud-Umgebungen verwendet, da er automatisch mit einem Cloud-Load-Balancer integriert wird. In Plattformen wie AWS, GCP oder Azure kann dieser Load Balancer von den Cloud-Anbietern verwaltet werden, was die Bereitstellung und Skalierung des Dienstes erleichtert.

Im Vergleich zu anderen Servicetypen wie ClusterIP oder NodePort, die entweder nur intern oder über eine Node-IP zugänglich sind, ermöglicht der LoadBalancer-Service eine direkte externe Erreichbarkeit des Dienstes, was ihn ideal für Webanwendungen oder APIs macht, die eine hohe Verfügbarkeit und eine zuverlässige Lastverteilung benötigen.

9.4 Ingress



9.4.1 Warum funktioniert der Zugriff über 127.0.0.1 oder localhost nicht?

Der Zugriff über 127.0.0.1 (localhost) führt zu einem Fehler, da die Anfrage nicht auf die richtige Weise an den Kubernetes-Cluster weitergeleitet wird. Der Grund dafür ist wie folgt:

1. **Ingress funktioniert nur über den Ingress-Controller:**
 - Der **Ingress-Controller** ist für die Verarbeitung von HTTP- und HTTPS-Anfragen verantwortlich, die auf Basis von Domainnamen (z. B. **my-app.com**) und definierten Regeln an den richtigen Service weitergeleitet werden.
 - **127.0.0.1** und **localhost** verweisen jedoch nur auf den lokalen Host-Computer (z. B. den Laptop oder Server), auf dem die Anfrage ausgeführt wird. Diese IP-Adresse hat keine direkte Verbindung zum Ingress-Controller oder dem Kubernetes-Cluster, wenn der Ingress-Controller ordnungsgemäß eingerichtet ist.
2. **DNS-Auflösung:**
 - Die Domain **my-app.com** muss korrekt auf die öffentliche IP-Adresse des Ingress-Controllers (z. B. einen Load Balancer oder eine externe IP des Kubernetes-Clusters) aufgelöst werden.
 - **127.0.0.1** verweist ausschließlich auf den lokalen Rechner, wodurch die Anfrage nicht an die externe IP des Clusters weitergeleitet werden kann.
3. **Lokale IP ist nicht verbunden:**
 - Wenn **127.0.0.1** oder **localhost** verwendet wird, versucht die Anfrage, die Anwendung lokal auf der Maschine zu erreichen. Der Service jedoch, der durch den Ingress-Controller bereitgestellt wird, ist nicht lokal erreichbar. Er kann nur über die Domain und die damit verbundene öffentliche IP-Adresse des Ingress-Controllers bzw. Load Balancers zugänglich gemacht werden.

9.5 Portainer auf Kubernetes

The screenshot shows the Portainer web interface for a Kubernetes cluster. On the left, a list of services is shown, with 'portainer' selected. The main panel displays the details for the 'portainer' service in the 'portainer' namespace. The 'Properties' section shows the service was created 3m 40s ago, has 6 labels, and 2 annotations. The 'Selector' is 'app.kubernetes.io/instance=portainer' and 'app.kubernetes.io/name=portainer'. The 'Type' is 'LoadBalancer' and 'Session Affinity' is 'None'. The 'Connection' section shows the cluster IP is 10.103.42.92, cluster IPs are 10.103.42.92, IP families are IPv4, IP family policy is 'SingleStack', and external IPs are 'localhost'. The 'Ports' section shows three ports: 9000-33678/TCP, 9443-32164/TCP, and 8000-32422/TCP. The 'Endpoints' section shows the 'portainer' endpoint with IP addresses 10.1.0.70-9443, 10.1.0.70-9000, and 10.1.0.70-8000. The 'Vulnerabilities' and 'Events' sections are also visible.

The screenshot shows the Portainer.io Community Edition web interface. The left sidebar contains navigation links: Home, Environment (None selected), Administration, User-related, Environment-related, Registries, Logs, Notifications, and Settings. The main panel displays the 'Home' page with a 'Latest News From Portainer' section. Below this, the 'Environments' section shows a list of environments. The first environment is 'local', which is 'Up' and was created on 2025-03-14 11:08:53. It is a 'Standalone 27.5.1' environment using '/var/run/docker.sock'. The environment has 4 stacks, 21 containers (15 green, 5 red, 0 yellow, 0 blue), 25 volumes, 27 images, 2 CPU, and 2 GB RAM.

10 NAS auf Kubernetes

10.1 Voraussetzungen

- **Docker & Kubernetes:** Es wird vorausgesetzt, dass Docker und Kubernetes (z. B. Minikube oder k3s) auf dem System installiert sind.
- **kubectl:** Die Installation von kubectl ist erforderlich, um das Kubernetes-Cluster zu verwalten.
- **Grundkenntnisse in YAML und Kubernetes-Konzepten:** Es sind grundlegende Kenntnisse in YAML sowie in den Kubernetes-Konzepten wie Deployments, Services und Persistent Volumes erforderlich.

10.2 Projekt Aufbau

1. Namespace erstellen:

```
kubectl create namespace nas-system
```

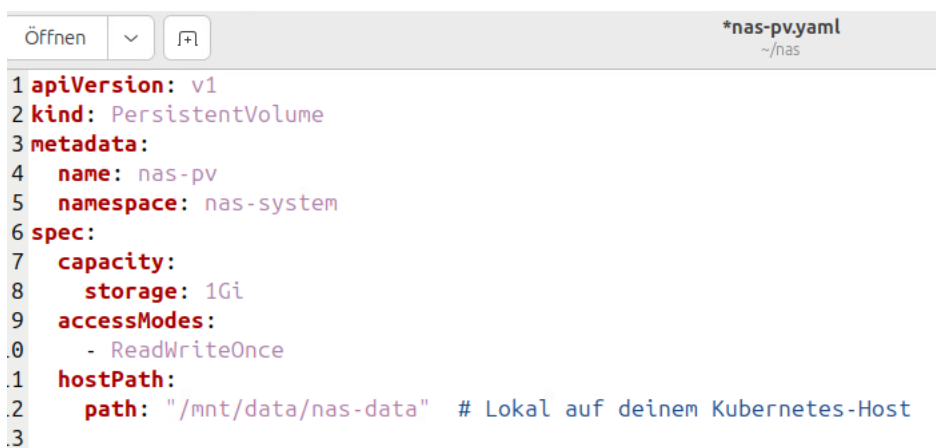
2. Verzeichnis erstellen:

```
sudo mkdir -p /mnt/data/nas-data
```

3. Dem Verzeichnis Schreibrechte geben, damit Kubernetes darauf zugreifen kann:

```
sudo chmod -R 777 /mnt/data/nas-data
```

4. Erstellen eines Persistent Volumes und Speichern der YAML-Datei als `nas-pv.yaml`:



```
1 apiVersion: v1
2 kind: PersistentVolume
3 metadata:
4   name: nas-pv
5   namespace: nas-system
6 spec:
7   capacity:
8     storage: 1Gi
9   accessModes:
10     - ReadWriteOnce
11 hostPath:
12   path: "/mnt/data/nas-data" # Lokal auf deinem Kubernetes-Host
13
```

5. Den Persistent Volume erstellen:

```
kubectl apply -f nas-pv.yaml
```

6. Erstellen eines Persistent Volumes und Speichern der YAML-Datei als `nas-pvc.yaml`:

```
Öffnen  ▼  [🔍]  nas-pvc.yaml
~/nas
1 apiVersion: v1
2 kind: PersistentVolumeClaim
3 metadata:
4   name: nas-pvc
5   namespace: nas-system
6 spec:
7   accessModes:
8     - ReadWriteOnce
9   resources:
10    requests:
11     storage: 1Gi
```

7. Den PVC erstellen:

```
Öffnen  ▼  [🔍]  nas-pvc.yaml
~/nas
1 apiVersion: v1
2 kind: PersistentVolumeClaim
3 metadata:
4   name: nas-pvc
5   namespace: nas-system
6 spec:
7   accessModes:
8     - ReadWriteOnce
9   resources:
10    requests:
11     storage: 1Gi
```

8. Erstellen eines Nextcloud Deployments und Speichern der YAML-Datei als `nextcloud-deployment.yaml`:

```
Öffnen  ▼  [🔍]  *nextcloud-deployment.yaml
~/nas
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: nextcloud
5   namespace: nas-system
6 spec:
7   replicas: 1
8   selector:
9     matchLabels:
10      app: nextcloud
11   template:
12     metadata:
13       labels:
14         app: nextcloud
15     spec:
16       containers:
17         - name: nextcloud
18           image: nextcloud
19           ports:
20             - containerPort: 80
21           volumeMounts:
22             - name: nextcloud-storage
23               mountPath: /var/www/html/data # Mount des Verzeichnisses für Nextcloud-Daten
24       volumes:
25         - name: nextcloud-storage
26           persistentVolumeClaim:
27             claimName: nas-pvc # Verwende deinen PersistentVolumeClaim 'nas-pvc'
```

9. Deployment erstellen mit:

```
kubectl apply -f nextcloud-deployment.yaml
```

10. Erstellen eines Services und Speichern der YAML-Datei als `nextcloud-service.yaml`:

```
Öffnen  nextcloud-service.yaml
~/nas

1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: nextcloud-service
5   namespace: nas-system
6 spec:
7   selector:
8     app: nextcloud
9   ports:
10    - protocol: TCP
11      port: 80
12      targetPort: 80
13   type: NodePort
```

11. Service erstellen mit:

```
kubectl apply -f nextcloud-service.yaml
```

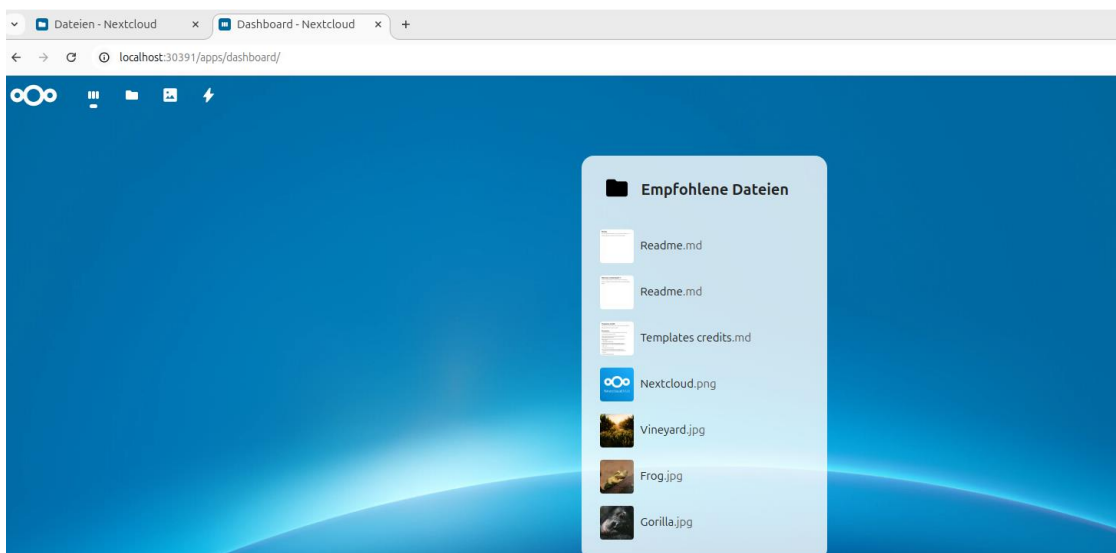
12. Den NodePort finden mit:

```
kubectl get services -n nas-system
```

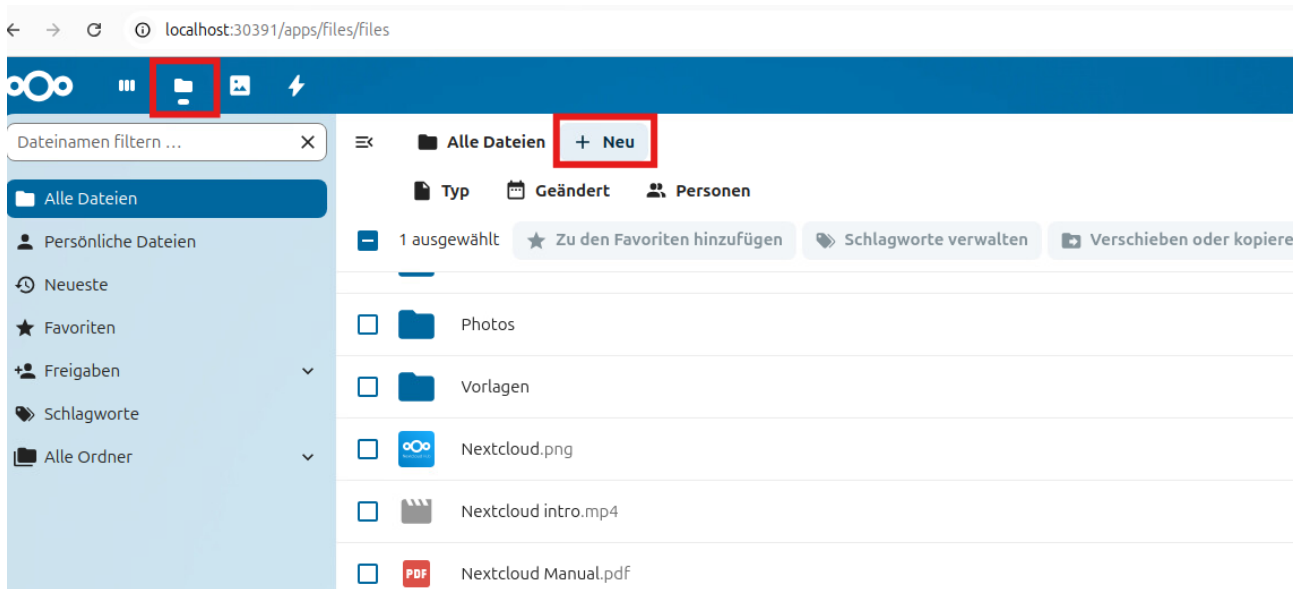
```
vmadmin@li244-vmLP1:~$ kubectl get services -n nas-system
NAME                TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
nextcloud-service   NodePort    10.106.153.81 <none>         80:30391/TCP     3h
```

13. Im Browser `localhost:30391` eingeben. Danach solle die Nextcloud Registrierung erscheinen. Dort muss man sich mit einem Adminnamen und Passwort registrieren.

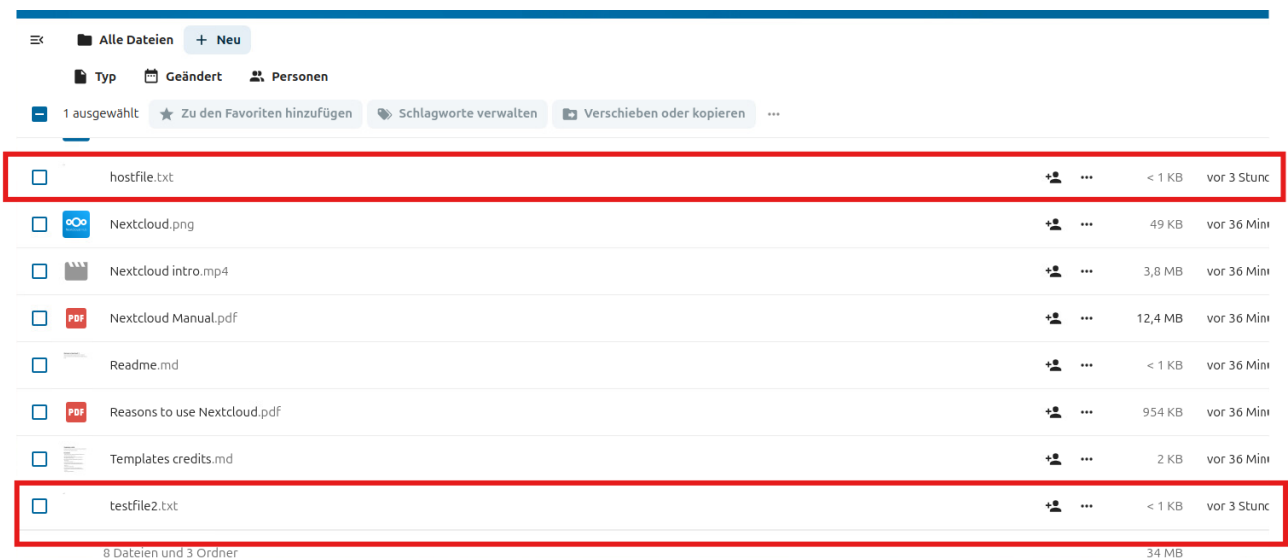
14. So solle dann Nextcloud Dashboard aussehen:



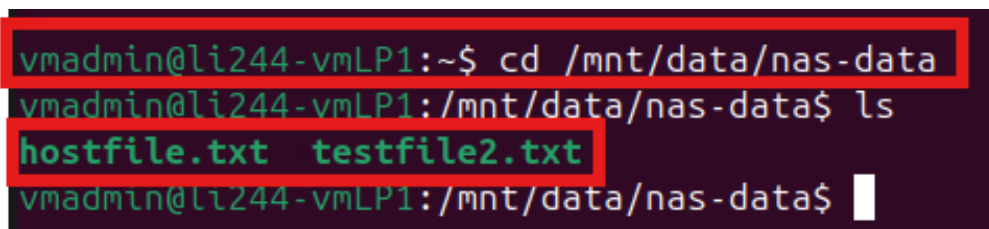
15. Auf Nextcloud ein paar Dateien hochladen:



16. Diese beiden Files wurde hochgeladen:



17. In dieses Verzeichnis wechseln und ls eingeben. Dort sollten dann die Dateien, welche auf Nextcloud geladen wurden, zusehen sein:



11 To-do-app auf Podman zum Laufen bringen

1. Podman herunterladen:

```
sudo apt update  
sudo apt install podman
```

2. Podman Desktop installieren:

```
flatpak install flathub io.podman_desktop.PodmanDesktop
```

```
vmadmin@li244-vmLP1:~$ flatpak install flathub io.podman_desktop.PodmanDesktop  
Suchen nach Übereinstimmungen ...  
Required runtime for io.podman_desktop.PodmanDesktop/x86_64/stable (runtime/org.freedesktop.Platform/x86_64/24.08) found in remote flathub  
Möchten Sie es installieren? [Y/n]: Y  
  
io.podman_desktop.PodmanDesktop Berechtigungen:  
ipc network x11 dri file access [1] dbus access [2]  
  
[1] /run/docker.sock, home, xdg-run/containers/auth.json, xdg-run/podman:create  
[2] org.freedesktop.Flatpak, org.freedesktop.Notifications, org.kde.StatusNotifierWatcher  
  
KENNUNG                                Zweig                                Op                                Gegenstelle                          Herunterladen  
1. [✓] org.freedesktop.Platform.GL.default 24.08                                i                                flathub                             156.3 MB / 156.8 MB  
2. [✓] org.freedesktop.Platform.GL.default 24.08extra                            i                                flathub                             25.2 MB / 156.8 MB  
3. [✓] org.freedesktop.Platform.Locale      24.08                                i                                flathub                             24.2 MB / 380.4 MB  
4. [✓] org.freedesktop.Platform.openh264    2.5.1                                i                                flathub                             913.7 KB / 971.4 KB  
5. [✓] org.gtk.Gtk3theme.Yaru                3.22                                i                                flathub                             139.3 KB / 191.5 KB  
6. [✓] org.freedesktop.Platform             24.08                                i                                flathub                             210.2 MB / 264.5 MB  
7. [✓] io.podman_desktop.PodmanDesktop      stable                               i                                flathub                             133.6 MB / 140.0 MB  
  
Installation abgeschlossen.
```

3. Python3 installieren:

```
sudo apt install python3-venv
```

4. Eine virtuelle Umgebung mit dem Namen podman-env im aktuellen Verzeichnis erstellen:

```
python3 -m venv podman-env
```

5. Aktivierung der virtuellen Python Umgebung:

```
source podman-env/bin/activate
```

6. Podman Compose installieren:

```
pip install podman-compose
```

7. Die FROM Zeile in den Dockerfiles von Master, Slave und To-do-app ändern zu:

```
Dockerfile  
~/cloudmodule-main/to-do-appv1/redis-master  
Öffnen  [Icon]  Speichern  [Icon]  [Icon]  [Icon]  
1 FROM docker.io/redis:alpine3.16  
2 MAINTAINER Johannes M. Scheuermann <johannes.scheuermann@inovex.de>  
3  
4 COPY start-redis-master.sh /start-redis-master.sh  
5 RUN chmod +x /start-redis-master.sh  
6 CMD /start-redis-master.sh
```

8. In das richtige Verzeichnis wechseln:

```
(podman-env) vmadmin@li244-vmLP1:~/cloudmodule-main/to-do-appv1/web-frontent$
```

9. Auf Podman die Images bauen:

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

10. Netzwerk erstellen:

```
podman network create todoapp_network
```

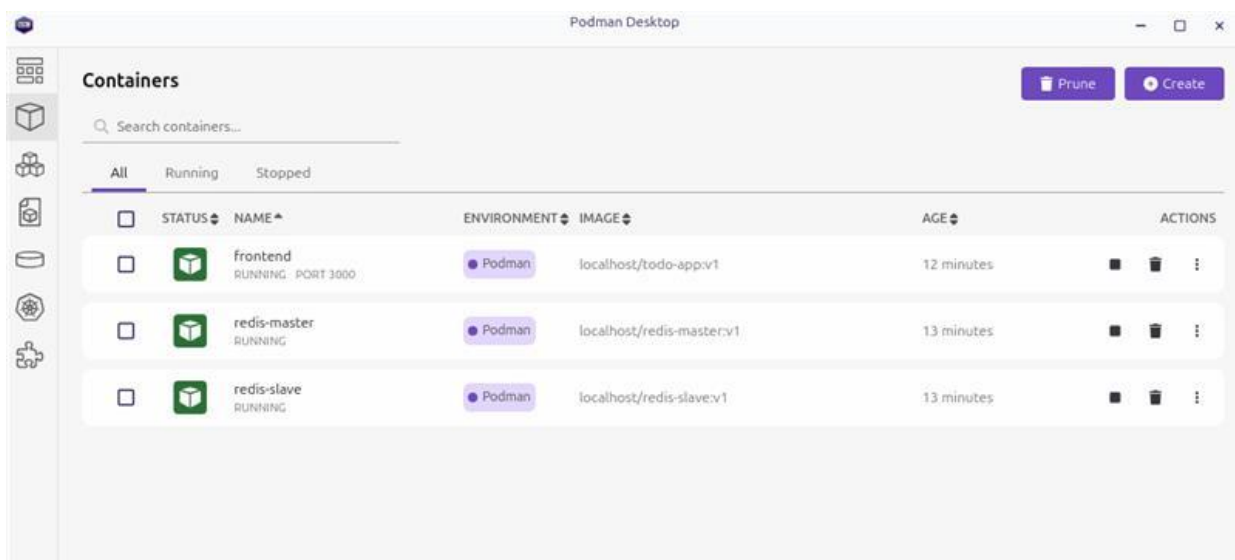
11. Redis Master Container, Redis Slave Container und Frontend Container starten :

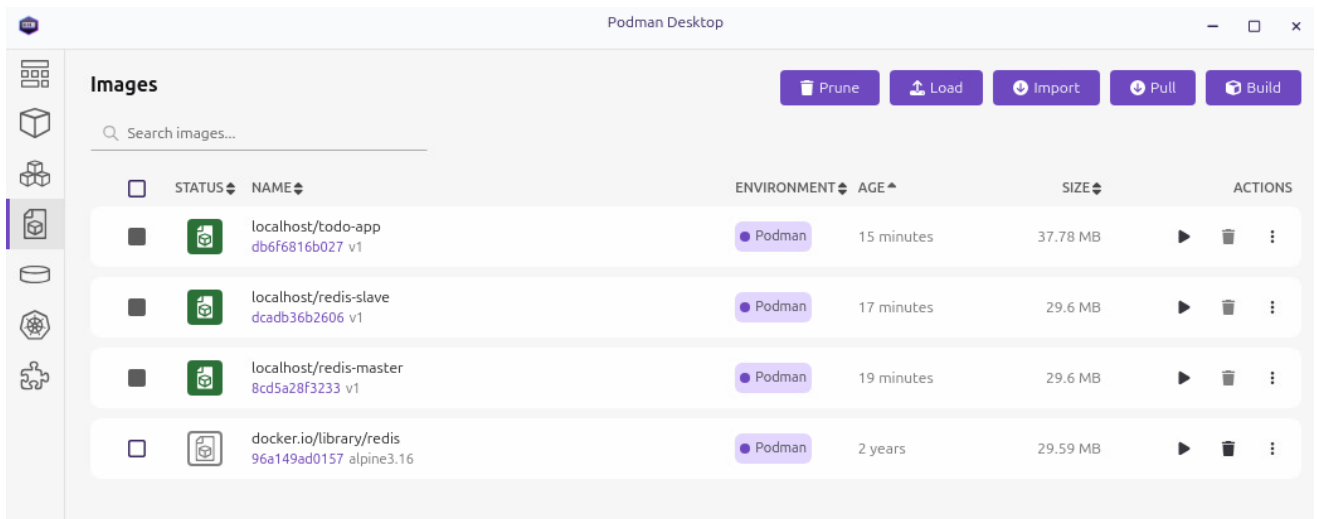
```
podman run --network=todoapp_network --name=redis-master -d redis-master:v1
podman run --network=todoapp_network --name=redis-slave -d redis-slave:v1
podman run --network=todoapp_network --name=frontend -d -p 3000:3000
todo-app:v1
```

12. Mit diesem Befehl sieht man dann, ob die Container gestartet wurden:

```
(podman-env) vmadmin@li244-vmLP1:~/cloudmodule-main/to-do-appv1/web-frontent$ podman ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
1e7ef10a4d0c   localhost/redis-master:v1          /bin/sh -c /start...    43 seconds ago Up 43 seconds                redis-master
3085bd6fd92e   localhost/redis-slave:v1          /bin/sh -c /start...    28 seconds ago Up 29 seconds                redis-slave
12fc447e8c1b   localhost/todo-app:v1             ./todo-app              18 seconds ago Up 19 seconds    0.0.0.0:3000->3000/tcp        frontend
```

13. So sehen die Container und Images auf Podman aus:





14. Im Browser localhost:3000 eingeben. Dann sollte die To-do-App erscheinen:

