

Portfolio Modul 347
Dienst mit Container Anwenden
Version 1.0

Pietro Caroni, infw2023a

February 22, 2024



Contents

0.1	Vorwort	2
1	Tag 1	3
1.1	Was sind Container und deren Nutzen?	3
1.2	Was ist DevOps?	3
1.3	Virtualisierung vs. Containerisierung	4
1.4	Image vs. Container	4
	Zusammenfassung der gelernten Befehle	5
1.5	Screenshot OnlyOffice	6
2	Tag 2	7
2.1	To-do-App version 1	7
2.2	To-do-App version 1 auf Git	8
2.3	Benötigte Befehle für Git	9
2.4	To-do-App version 2	10
2.5	To-do-App version 2 auf Git	11
3	Tag 3	12
3.1	Docker Compose	12
3.2	Version2 mit Docker Compose	12
3.3	Portainer Installation	13
3.4	To-do-App mit Portainer	13
3.5	Shop Beispiel	14
4	Tag 4	16
4.1	Eigenes Projekt	16
4.2	Installationsanleitung	16

0.1 Vorwort

In diesem Modul werden wir uns intensiv mit Docker auseinandersetzen. Für mich persönlich stellt dies ein vollkommen neues Thema dar. Ich wusste nicht einmal, dass Docker so existiert. Entsprechend gross ist die Reise auf welche ich mich hier begeben. Ich hoffe sie wird faszinierend!

Kapitel 1

Tag 1

1.1 Was sind Container und deren Nutzen?

Hinter Docker steht grundsätzlich die Idee, die Verschiffung von Applikationen soweit zu vereinfachen, dass sie ähnlich einem Container um die Welt geschickt werden können. So wird zumindest die ursprüngliche Idee von dem Schöpfer Ben Golup dargestellt.

Docker ist eine Plattform zum Entwickeln, Versenden und Ausführen von Anwendungen in möglichst leichtgewichtigen Containern. In einen Container kann man eine Anwendung mit allen benötigten Bibliotheken und Abhängigkeiten packen und anschliessend diese als ein Paket versenden. Dadurch wird sichergestellt, dass die Anwendung auf jedem anderen Linux System ausführbar ist, unabhängig von den Einstellungen und Programmen welche dieses hat. Docker vereinfacht und beschleunigt somit den Arbeitsablauf bei der Entwicklung und Bereitstellung von Ressourcen.

1.2 Was ist DevOps?

DevOps steht für das Ziel Software schneller und qualitativ hochwertiger zu entwickeln sowie die Verteilung der Software zu beschleunigen. Die Agilität und Qualität sind Faktoren welche in der Software Entwicklung immer wieder zu sprechen geben. Meist (in meiner limitierten Erfahrung) scheint aus Kundensicht von beiden Konzepten nicht genug vorhanden zu sein. Dieser Problemstellung soll bei DevOps begegnet werden durch die Zusammenführung der Entwicklung, des IT-Betriebes und der Qualitätssicherung.

Konkreter heisst dies, dass man die Prozesse der Softwareentwicklung und des IT-Teams zusammen nimmt und in einem einzigen grossen Prozess vereint. Die Teams arbeiten dabei durchgehend zusammen.

Containerisierung kann in diesem Bereich im Besonderen helfen, da man den Wunsch nach Agilität

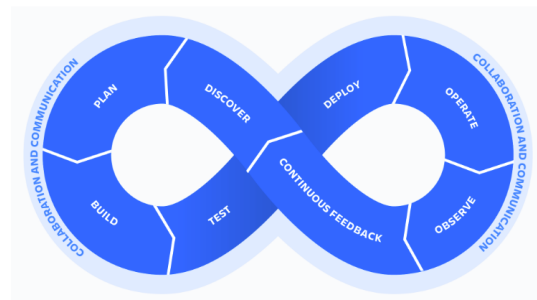


Abb. 1.1: DevOps-Zyklus

der Softwareentwicklung abdecken kann ohne dass das Hauptsystem verändert werden muss, was wiederum dem IT-Team entgegen kommt.

1.3 Virtualisierung vs. Containerisierung

Eine Virtuelle Maschine stellt ein komplettes System dar, welches auf einem schon bestehenden System läuft. Mithilfe eines Hostkernels wird die Kommunikation zwischen dem Virtuellen System und der physisch vorhandenen Hardware hergestellt. Mit einer Cloudsoftware welche nochmals eine Schicht über das System legt, kann man dabei sogar die Ressourcen von verschiedenen Computern ansprechen und verwenden.

Ein Docker-Container auf der anderen Seite verwendet zu grossen Teilen das Hostsystem und kann dadurch sehr leichtgewichtig sein. Der Docker-Container benötigt in sich nur die Teile welche zur Ausführung der Anwendung benötigt werden. Alle Prozesse laufen schlussendlich wieder auf dem Kernel des Grundsystems.

Bezüglich Sicherheit bietet Docker den Vorteil, dass er zum einen nur die wirklich notwendigen Dienste beinhaltet was die Möglichkeit für problematische Zugriffe stark einschränkt. Zusätzlich kann man über Namespaces die Zugriffsmöglichkeiten des Containers auf das System direkt steuern. Dadurch kann die Sicherheit gegenüber von Virtuellen Umgebungen weiter gesteigert werden.

1.4 Image vs. Container

Docker nutzt Images als Anleitungen wie ein bestimmter Container erstellt werden muss. Diese Images sind Schreibgeschützt und erstellen den Container somit entsprechend den genauen Vorgaben des Dockerfile. Das Image enthält alle Bibliotheken und Abhängigkeiten welche der Container benötigt um zu funktionieren.

Ein Docker Container wird aus einem Image erstellt. Der Container ist auf jedem System lauffähig solange dieses die Docker Software installiert hat. Der Container weis zur Laufzeit nichts von dem Betriebssystem auf welchem er ausgeführt wird sondern bezieht alle benötigten Ressourcen direkt aus sich selbst [1].

Zusammenfassung der gelernten Befehle

Alle hier aufgeführten Befehle besitzen weitere Argumente welche ihnen übergeben werden können. Dadurch verändert sich das Verhalten teils drastisch. Da eine solche Auflistung jedoch den Rahmen sprengen würde wird an dieser Stelle darauf verzichtet.

Befehl	Beschreibung
docker pull	Zieht ein image von einem Repository
docker run	Startet einen neuen Container. Images werden heruntergeladen, falls sie fehlen.
docker images	Anzeigen von Images
docker ps	Zeigt den Status der Container an
docker rm	Lösche einen Container
docker update	Updated den Containerinhalt dynamisch
docker start	Starte einen Container
docker stop	Stoppe einen Container
docker restart	Starte einen Container neu
docker pause	Pausiere einen Container
docker unpause	Startet einen pausierten Container
docker wait	Container wartet bis ein anderer stoppt
docker kill	Stoppt einen Container sofort
docker attach	Input / Output der Console wird an einen Kontainer gebunden
docker system prune	Löscht alles
docker rmi	Löscht ein Image
docker log	Zeigt die letzten Logdaten an
docker build	Baut ein Image von einem Dockerfile
docker push	Pushed ein Image
docker import	Erstellt ein Image aus einem .tar File
docker commit	erstellt ein Image aus einen Container
docker history	Zeigt die Geschichte an
docker network ls	Liste der Netzwerke
docker network create	Erstellt ein Docker Netzwerk

Table 1.1: Docker Befehle

1.5 Screenshot OnlyOffice

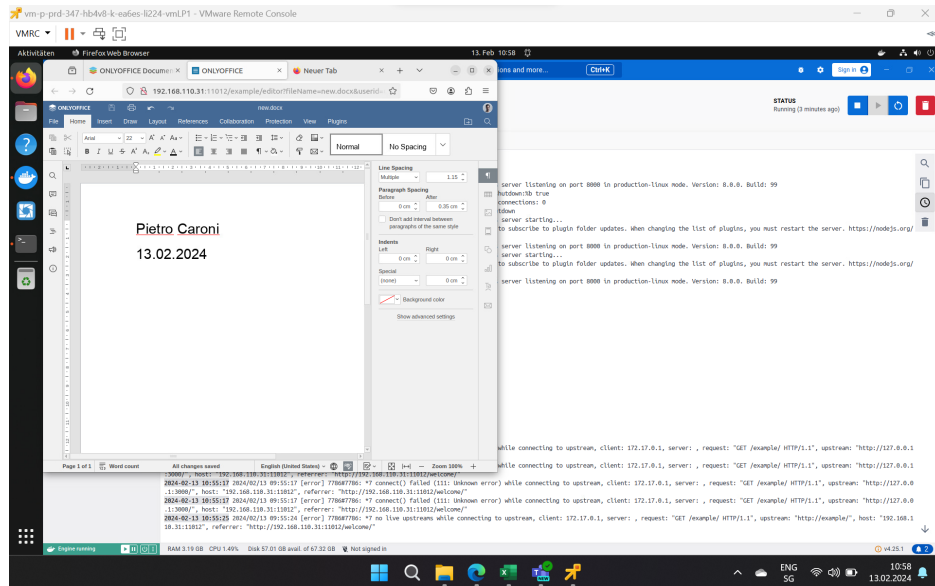


Abb. 1.2: Screenshot von OnlyOffice. Datum 13. Februar

Kapitel 2

Tag 2

2.1 To-do-App version 1

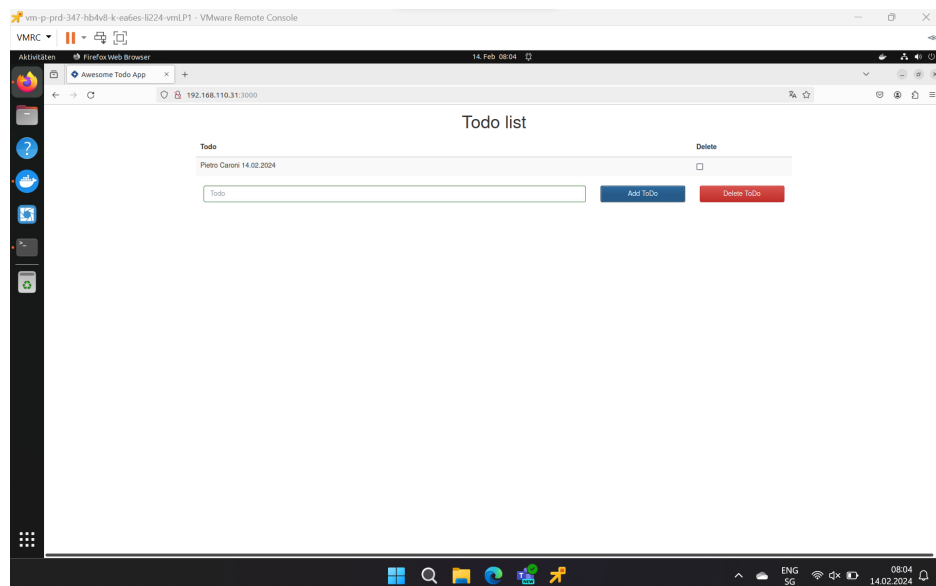


Abb. 2.1: To-do-App v1. Datum 19. Februar

2.2 To-do-App version 1 auf Git

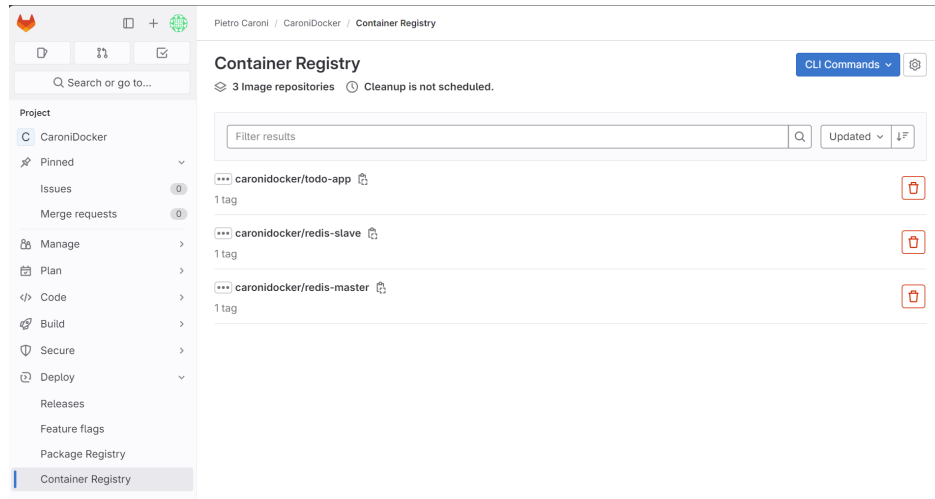


Abb. 2.2: To-do-App v1 auf Git gepusht. Datum 19. Februar

2.3 Benötigte Befehle für Git

Um ein Docker Image auf Git zu pushen, muss man in einem ersten Schritt das Image mit einem tag versehen. Dieses vorgehen steht in Kontrast zu dem Vorgehen welches verwendet wird, wenn ein Image auf die zentrale Docker Registrierung gebracht werden soll. Dort wird kein tag benötigt. Ein Docker tag stellt ein Alias für ein Image dar. Dieses wird benötigt um Docker mitzuteilen, wo er ein Image speichern soll.

Um ein Image tag zu erstellen benötigt man:

- Den Pfad zum Repository
- Die Ordnerstruktur für die Images
- Den Namen des Images
- Den tag des Images

Das ganze setzt sich folgendermassen zusammen:

Docker image tag, Name des Images , tag des Images, Pfad zum Repository, Ordnerstruktur, Name des Images, tag des Images

Danach kann das Image mit den uns bekannten Methoden transferiert werden:

Docker push, Pfad zum Repository, Ordnerstruktur, Name des Images, tag des Images

2.4 To-do-App version 2

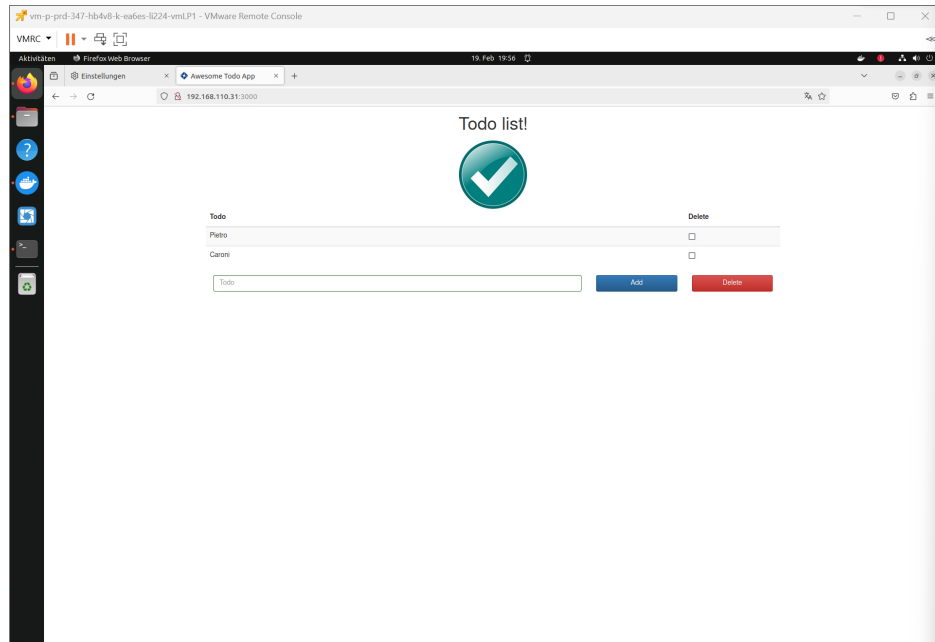


Abb. 2.3: To-do-App v2. Datum 20. Februar

2.5 To-do-App version 2 auf Git

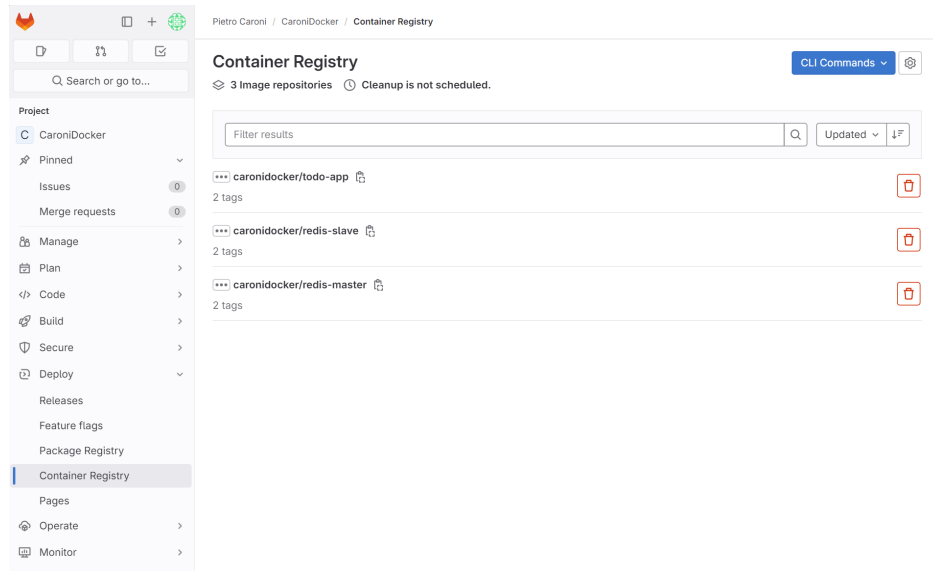


Abb. 2.4: To-do-App v2 auf Git gepusht. Datum 20. Februar

Kapitel 3

Tag 3

3.1 Docker Compose

Docker Compose ist ein Tool welches einen weiteren grossen Schritt auf den Entwickler und seine ihm inne wohnende Faulheit zugeht. Es erlaubt das gleichzeitige Starten aller konfigurierten Container und Dienste. Mit Docker Compose kann man somit das einzelne Starten von Containern umgehen. Um dies zu bewerkstelligen benutzt Docker Compose ein Yaml File welches die benötigten Konfigurationen beinhaltet. Docker Compose unterstützt den gesamten Lebenszyklus einer Anwendung. So kann Docker Compose einen Service starten, stoppen und neu bauen, den Status einer Anwendung einsehen, den Output einer Anwendung loggen oder auch nur ein einmaliges Kommando für einen Service laufen lassen. Zusammengefasst in den Worten von Docker selbst ist Compose ein Tool um Multicontainer Applikation zu definieren und laufen zu lassen. Es ist der Schlüssel für eine effiziente Entwicklung und Verteilung von Software [2].

3.2 Version2 mit Docker Compose

Da ich aus verschiedenen Gründen, wie zum Beispiel dem Cache meines Browsers, grösste Probleme hatte die Version 2 der To-do-App zum laufen zu bringen, hatte ich bereits grosse Sorgen beim Einstieg in die Thematik von Docker Compose. Jedoch verlief das ganze sehr viel besser als erwartet.

Als erster Schritt habe ich das Yaml File des Version 1 Beispiels in mein Verzeichnis der Version 2 kopiert. In einem weiteren Schritt wurde der Inhalt des Yaml Files angepasst. Spezifisch musste ich beim Punkt `todoApp`, Build den Namen des Files zu `./web-frontendv2` ändern. Danach konnte ich den Befehl: `docker-compose -f docker-compose.yaml up -d` auf der Kommandozeile (im richtigen Verzeichnis) ausführen und bereits war die Webseite funktional. Grossartig! Mit dem Befehl: `docker-compose -f docker-compose.yaml down` konnte ich die erstellten Container wieder beenden.

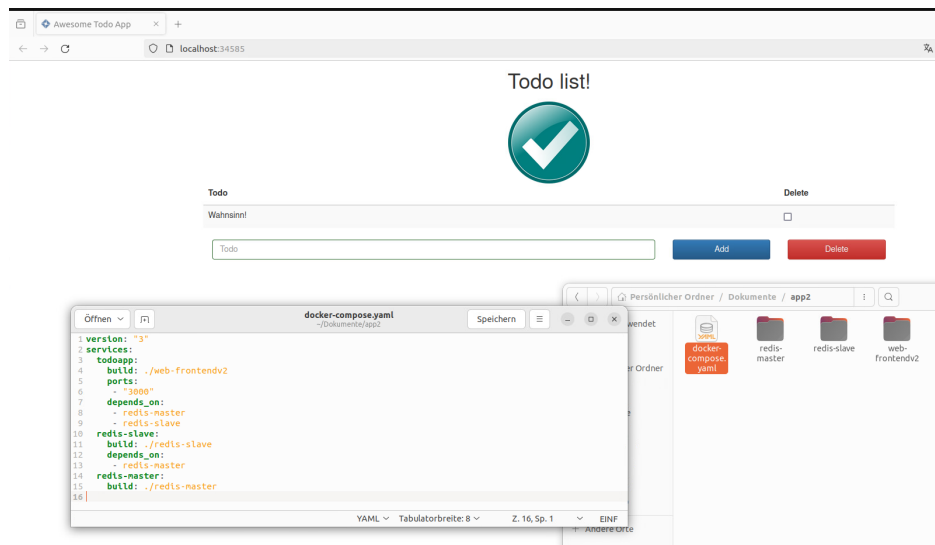


Abb. 3.1: Todo-app v2 mit Compose, Datum 20.Februar

3.3 Portainer Installation

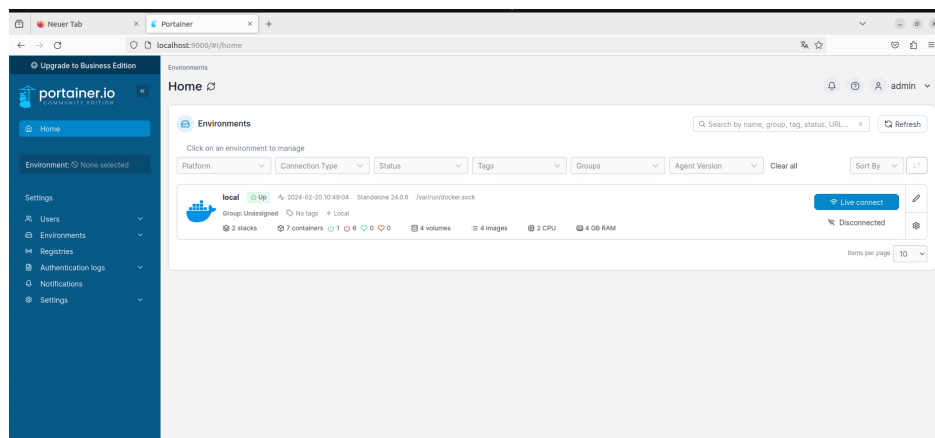


Abb. 3.2: Portainer Installation, Datum 20.Februar

3.4 To-do-App mit Portainer

Um die to-do-app im Portainer zu öffnen, musste ich diesen auf der Website aufrufen und als erstes ein Login erstellen. Danach konnte ich auf meine lokale Docker Installation zugreifen und somit auch auf die Images und Container. Im Menu Punkt Stacks konnte ich einen neuen Stack hinzufügen, wobei ich im Webeditor den Inhalt unseres Git Yaml Files einfügte. Danach konnte der Stack erstellt

werden. Somit kann der Container nun genutzt werden. Der gewünschte Container kann ebenfalls direkt aus dem Portainer gestartet werden. Somit bietet Portainer eine interessante Alternative zur Verwendung der Docker Software.

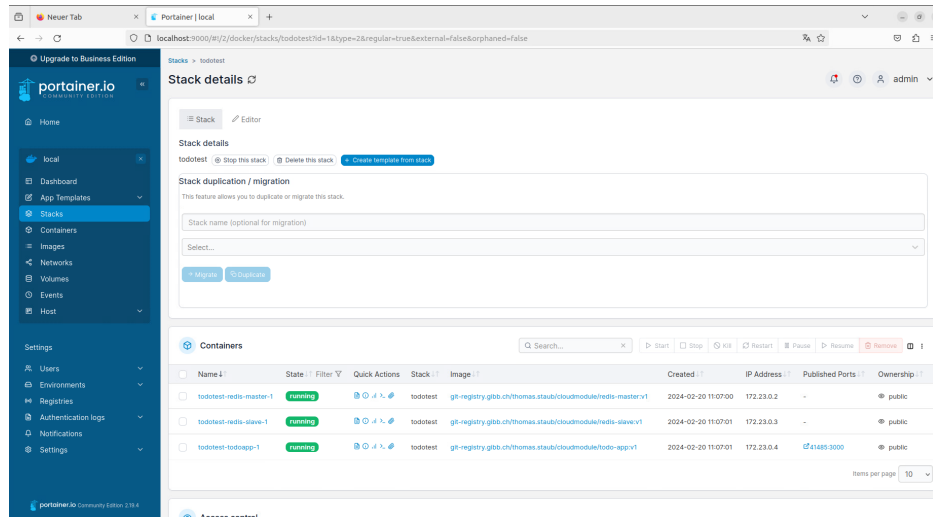


Abb. 3.3: In Portainer laufende To-do-App, Datum 20.Februar

3.5 Shop Beispiel

Um das Shop Beispiel zu installieren, musste ich zuerst in die Hostdatei die Zeile `host.docker.internal` für die IP `127.0.0.1` eintragen. Zu erwähnen ist an dieser Stelle, dass ich das Shop Beispiel in unserer Virtuellen Linux Maschine installierte. Unter Windows müsste die Hostdatei anderweitig angepasst werden. Nach dem das benötigte Repository heruntergeladen wurde, konnte das Image und die benötigten Container mit `docker-compose` erstellt werden. Dafür musste der Befehl natürlich in dem korrekten Verzeichnis ausgeführt werden. Dank der phantastischen Funktionalität einer `docker-compose` Datei erledigte sich der Rest selbst. Die Dauer um alle Container zu installieren und zu starten verunsicherte mich zuerst, jedoch verlief alles reibungslos. Anschliessend konnte das Shop Beispiel unter `http://host.docker.internal:5008/` aufgerufen werden und mit den weiteren Aufgaben in Smartlearn fortgefahren werden.

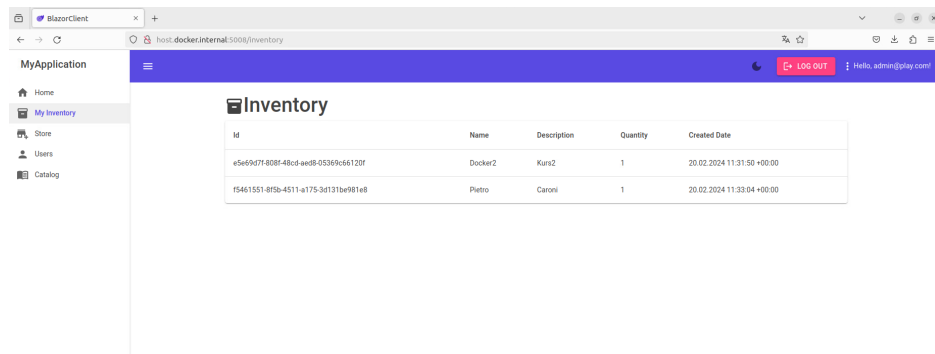


Abb. 3.4: Laufendes Shop Beispiel, Datum 20.Februar

Kapitel 4

Tag 4

4.1 Eigenes Projekt

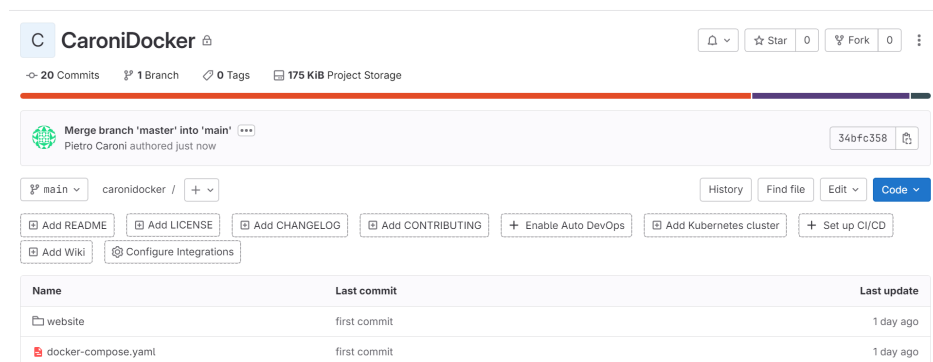


Abb. 4.1: Eigenes Projekt in Git, Datum 20.Februar

4.2 Installationsanleitung

Um mein eigenes Projekt, CaroniDocker, zu installieren, laden Sie zuerst den Main Branch meines Projektes herunter oder Clonen Sie das Projekt per Git in Ihre eigene Arbeitsumgebung.

Nach dem Sie die Datei entpackt haben, wechseln Sie in das Verzeichnis meines Projektes.

Ihr Pfad sollte nun mit /caronidocker-main enden.

Stellen Sie sicher, dass Docker Compose auf Ihrem Gerät installiert ist.

Führen Sie den folgenden Befehl aus:

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

Anschliessend können Sie die Webseite unter <http://localhost:8080/> aufrufen.

Bitte beachten Sie, dass es sich dabei um unsere erste Webseite im Modul 293 handelt. Diese sollte möglichst schlicht und ohne Javascript auskommen. Somit sind die Knöpfe auf der Webseite nutzbar, jedoch führen sie zu keiner Zustandsveränderung. Auch das Formular welches sich auf der Contact Seite befindet versendet keine Daten.

Quellenverzeichnis

- [1] Was ist der Unterschied zwischen Docker-Images und Containern?, URL: <https://aws.amazon.com/de/compare/the-difference-between-docker-images-and-containers/>
- [2] Docker Compose overview, URL: <https://docs.docker.com/compose/#:~:text=Docker%20Compose%20is%20a%20tool,efficient%20development%20and%20deployment%20experience.>