

Portfolio Modul 347
Dienst mit Container Anwenden
Version 2.0

Pietro Caroni, infw2023a

27. März 2024



Inhaltsverzeichnis

1	Tag 1	1
1.1	Was sind Container und deren Nutzen?	1
1.2	Was ist DevOps?	1
1.3	Virtualisierung vs. Containerisierung	2
1.4	Image vs. Container	2
	Zusammenfassung der gelernten Befehle	3
1.5	Screenshot OnlyOffice	4
2	Tag 2	5
2.1	To-do-App version 1	5
2.2	To-do-App version 1 auf Git	6
2.3	Benötigte Befehle für Git	7
2.4	To-do-App version 2	8
2.5	To-do-App version 2 auf Git	9
3	Tag 3	10
3.1	Docker Compose	10
3.2	Version2 mit Docker Compose	10
3.3	Portainer Installation	11
3.4	To-do-App mit Portainer	11
3.5	Shop Beispiel	12
4	Tag 4	14
4.1	Eigenes Projekt	14
4.2	Installationsanleitung	14
5	Tag 5	16
5.1	Beschreibung Kubernetes	16
5.2	Beschreibung Microservices	16
5.3	Installation Kubernetes	17
5.4	Installation und Verbinden von Lens	17
6	Tag 6	18
6.1	Raft-Konsens-Algorithmus	18
6.2	App in Kubernetes	19
6.3	Self Healing	20
6.4	Scale Down	20

6.5	Scale Up	20
6.6	Funktionierendes Rolling Update	21
6.7	Blue/Green Deployment	22
7	Tag 7	23
7.1	Eintrag zu Cluster IP und Node IP	23
7.2	Eintrag zu LoadBalancer	23
7.3	TodoApp mit Ingress	24
7.4	404 bei Aufruf mit Localhost	24
7.5	Portainer auf Kubernetes	25
8	Tag 8	26
8.1	Learning Beispiel auf Kubernetes	27
8.2	Istio läuft	29
8.3	Kiali läuft	30
8.4	Grafana läuft	31
9	Tag 9	32
9.1	Installation des eigenen Projektes auf Kubernetes	32

Zusammenfassung

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 gebe. 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 vereinigt. Die Teams arbeiten dabei durchgehend zusammen.

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

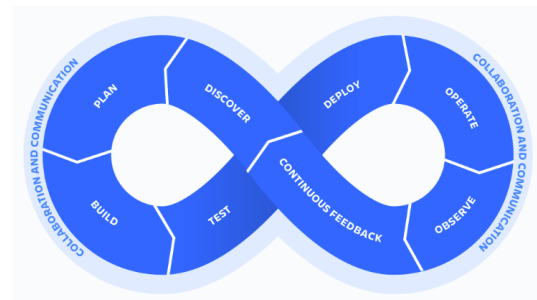


Abb. 1.1: DevOps-Zyklus

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

Tabelle 1.1: Docker Befehle

1.5 Screenshot OnlyOffice

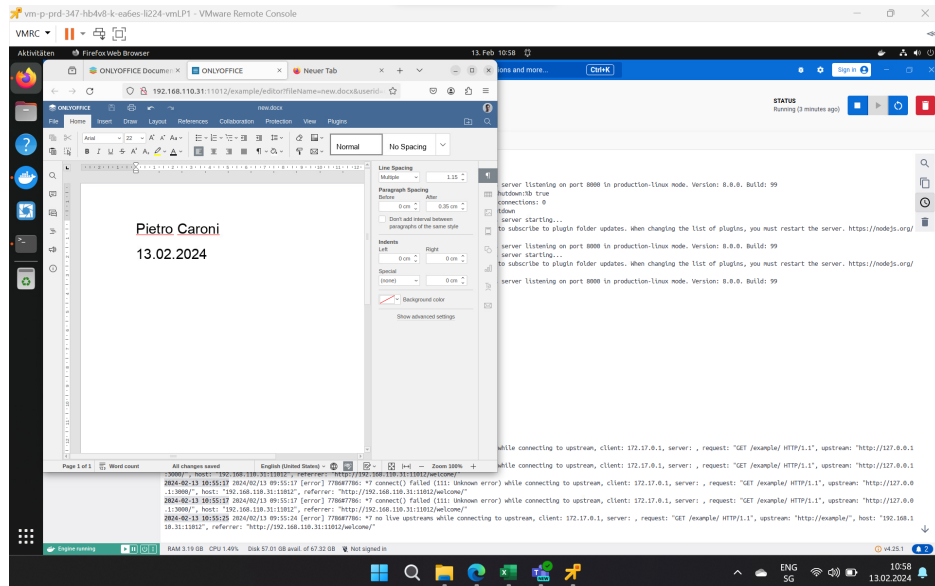


Abb. 1.2: Screenshot von OnlyOffice. Datum 13. Februar

Kapitel 2

Tag 2

2.1 To-do-App version 1

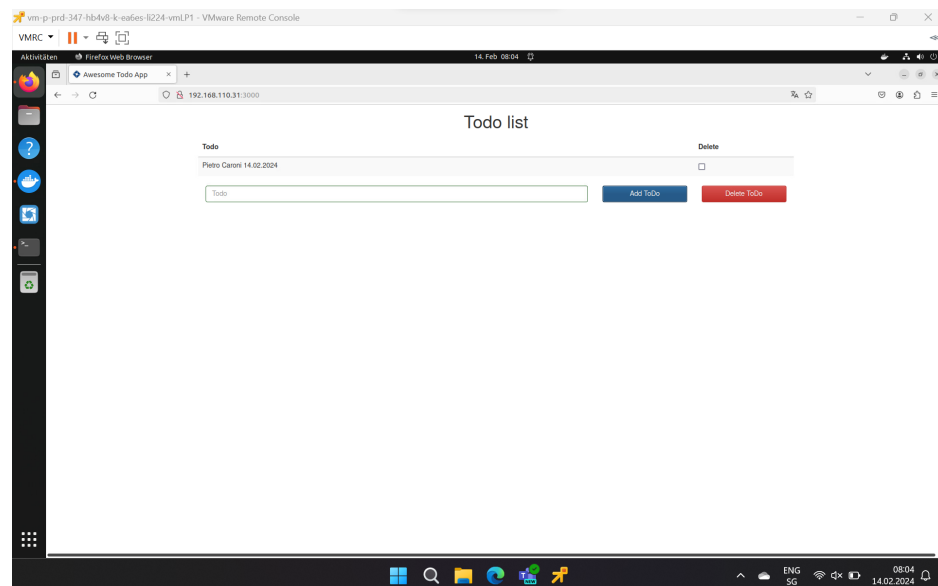


Abb. 2.1: To-do-App v1. Datum 19. Februar

2.2 To-do-App version 1 auf Git

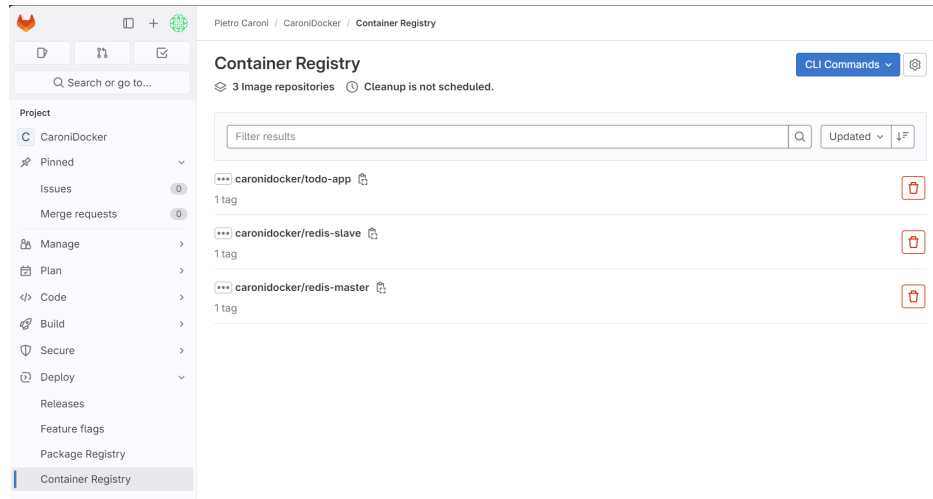


Abb. 2.2: To-do-App v1 auf Git gepusht. Datum 19. Februar

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

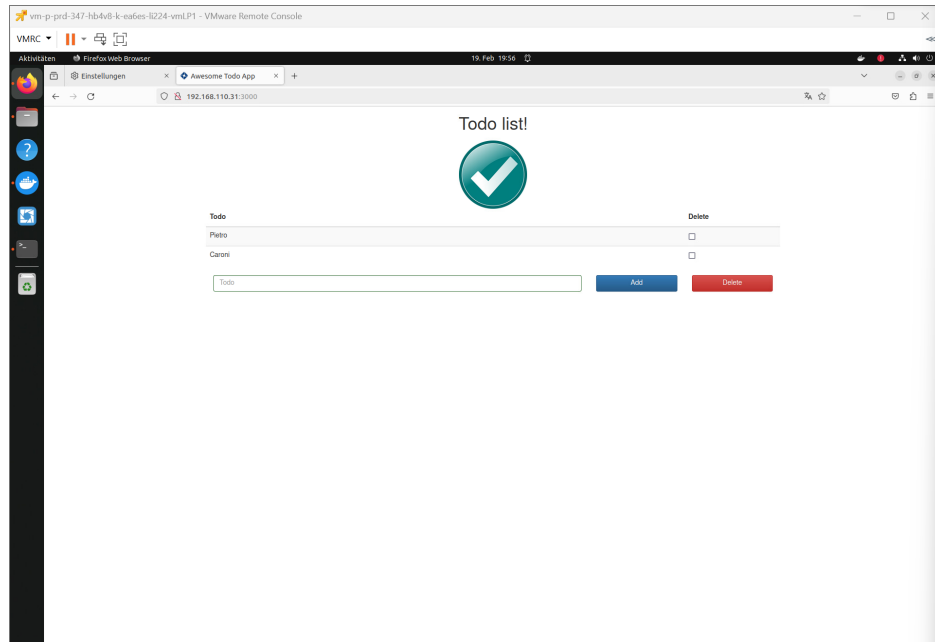


Abb. 2.3: To-do-App v2. Datum 20. Februar

2.5 To-do-App version 2 auf Git

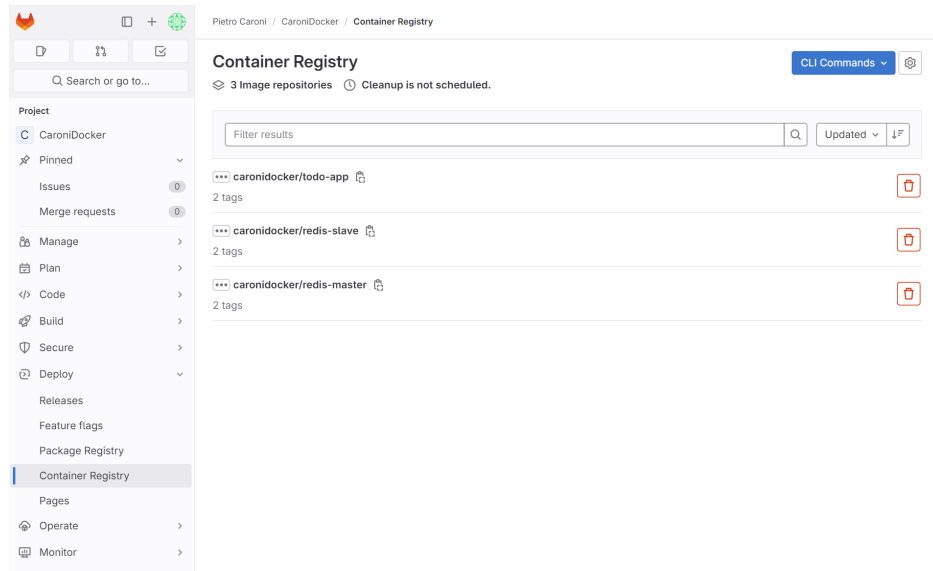


Abb. 2.4: To-do-App v2 auf Git gepusht. Datum 20. Februar

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.

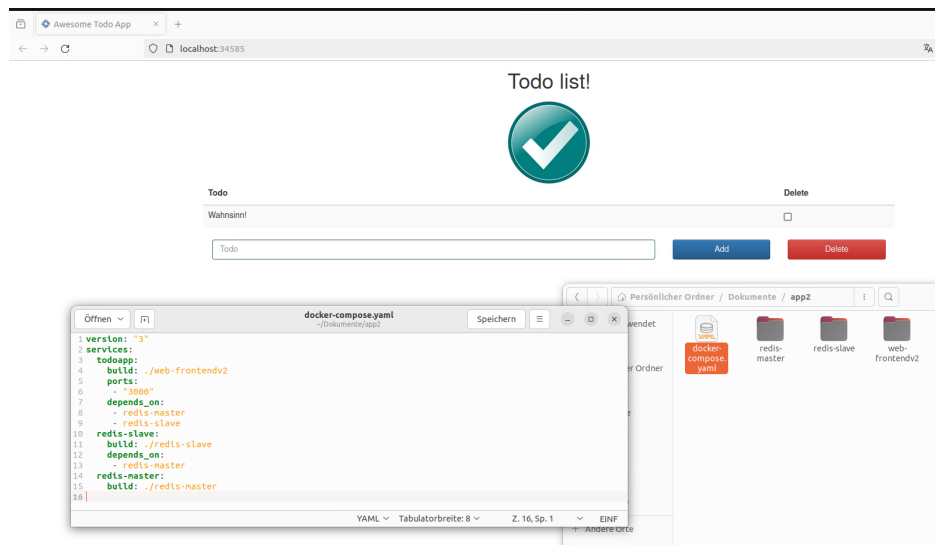


Abb. 3.1: Todo-app v2 mit Compose, Datum 20.Februar

3.3 Portainer Installation

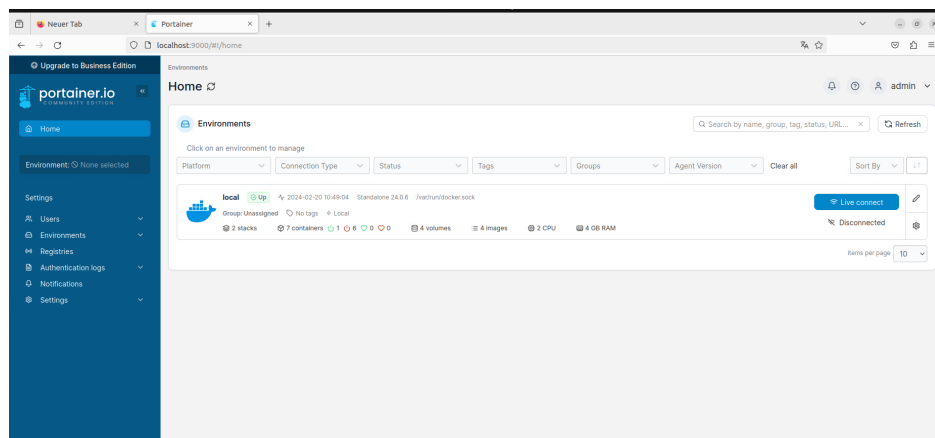


Abb. 3.2: Portainer Installation, Datum 20.Februar

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.

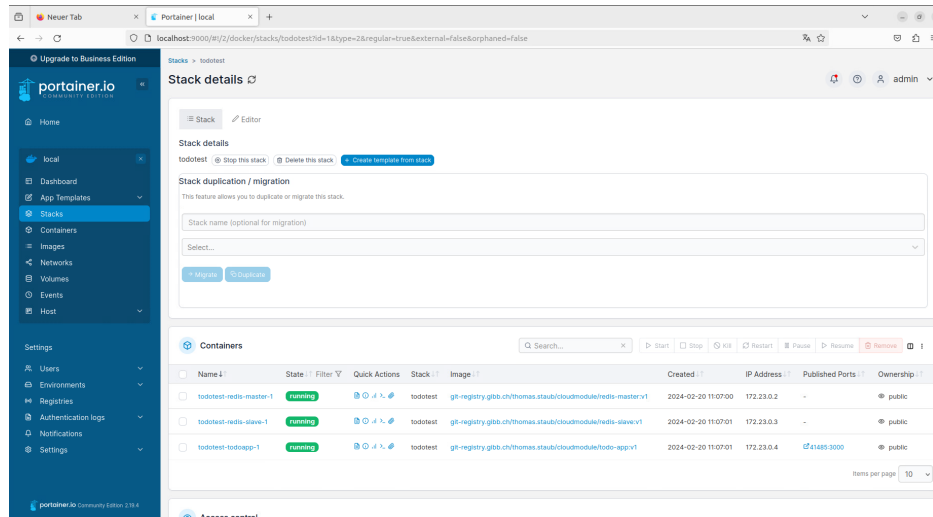


Abb. 3.3: In Portainer laufende To-do-App, Datum 20.Februar

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.

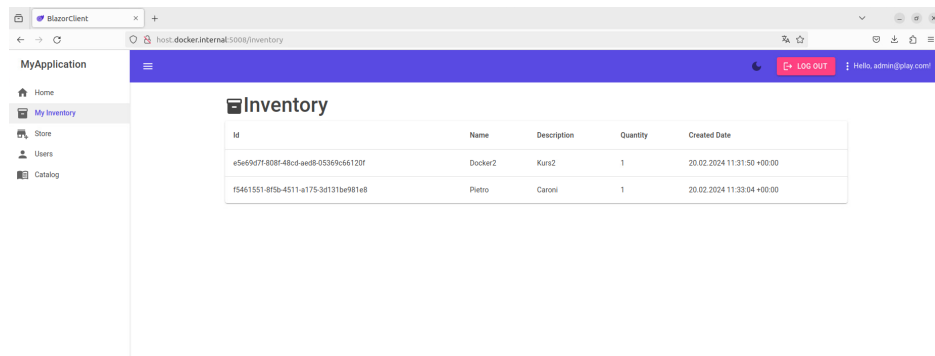


Abb. 3.4: Laufendes Shop Beispiel, Datum 20.Februar

Kapitel 4

Tag 4

4.1 Eigenes Projekt

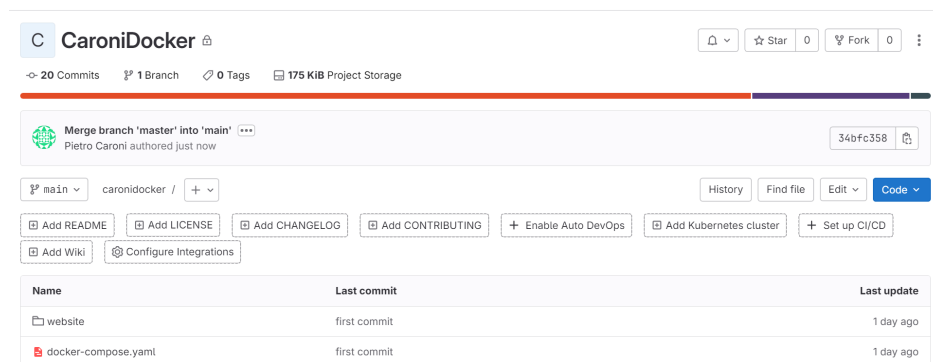


Abb. 4.1: Eigenes Projekt in Git, Datum 20.Februar

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.

Kapitel 5

Tag 5

5.1 Beschreibung Kubernetes

Bei Kubernetes (k8s) handelt es sich um ein von Google in 2014 bereitgestelltes Open Source Softwarepaket. Dieses soll Anwendungen verwalten welche durch einen Container bereitgestellt werden. Die Anwendung kann nicht nur Docker Container laufen lassen, sondern auch Container welche von einem anderen Dienst bereitgestellt werden. Kubernetes Netzwerke bestehen aus den folgenden teilen:

Eine oder mehrere Nodes welche entweder einen physischen PC oder eine virtuelle Maschine darstellen. Diese Node beinhaltet wiederum drei Teile:

- Ein kubelet - koordiniert die Pods in welchen Container laufen.
- Die Container Runtime - führt die eigentlichen Container aus.
- Ein Kube Proxy - beinhaltet die Netzwerkregeln der Node.

Ein Pod wiederum, koordiniert durch ein kubelet, stellt eine weitere Abstraktionsebene zwischen dem Betriebssystem und den Prozessen dar. In einem Pod können mehrere Container laufen. In der Regel ist es jedoch nur einer.

5.2 Beschreibung Microservices

Microservices stellen einen neuen Ansatz der Softwareentwicklung dar, welcher eine Abweichung zu der vorherrschenden Monolithischen Architekturweise darstellt. Bei einer monolithischen Softwarearchitektur werden alle Prozesse einer Anwendung in einem einzigen Service angeboten. Somit betreffen Änderungen welche an der Software vorgenommen werden, immer den gesamten Service. Dies kann zu Problemen führen bei zunehmend komplexer Software, da einzelne kleinere Änderungen einen grossen Einfluss auf Teile der Software haben können welche nicht im Fokus der Änderung lagen.

Bei dieser Problemstellung kommt der Ansatz von Microservices in das Spiel. Hier werden einzelne Komponenten erstellt welche mithilfe von schlanken APIs miteinander kommunizieren können. Jeder Service erfüllt dabei eine bestimmte Funktion und kann somit auch unabhängig aktualisiert

und skaliert werden. Somit sind Microservices agil, flexible in der Skalierung, einfach bereitzustellen, resilient gegenüber ausfällen und codetechnisch wiederverwendbar [3].

5.3 Installation Kubernetes

Um Kubernetes zu installieren, habe ich den direkten Weg über Docker gewählt auf Windows. Dies zum einen, da der Smartlearn Kurs dieses Vorgehen impliziert, aber auch weil wir von der vorhergehenden Folie wissen, dass nicht alle lightweight Installationen einer Kubernetes Umgebung gleichwertig sind.

Docker bietet uns die Möglichkeit Kubernetes direkt in sich selbst zu installieren. Dafür geht man einfach in die Einstellungen von Docker und aktiviert dort Kubernetes. Nach der Anwendung der neuen Einstellungen, installiert Docker Kubernetes selbständig und nach einem anschliessenden Neustart ist die Applikation einsatzbereit.

5.4 Installation und Verbinden von Lens

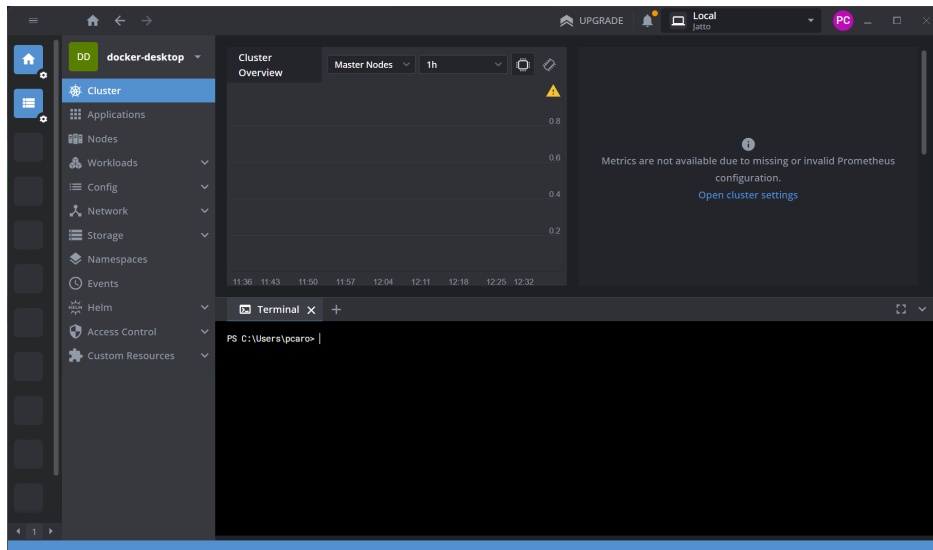


Abb. 5.1: Lens mit Server verbunden, Datum 27.Februar

Kapitel 6

Tag 6

6.1 Raft-Konsens-Algorithmus

Der Raft-Konsens-Algorithmus stellt eine Lösung für die Thematik der gemeinsamen Wahrheit dar welche, wie nun gelernt, nicht nur die Menschen beschäftigt sondern eben auch Computersysteme oder spezifischer, Serverclusters. Eine gängige Lösung welcher der Wahrheitsfindung dienen soll, ist die Arbeit mit einem Master Server und seinen Dienern. Der Master Server wird von einem Menschen erwählt und ist fortan zuständig für die Aufträge eines Klienten zur Veränderung seiner Datenbeständen. Nach getaner Arbeit informiert er die Diener welche somit ihre Datenbestände ebenfalls anpassen können.

Dies scheint soweit ein elegantes System zu sein. Jedoch stellt sich die Frage nach der Lösung einer Situation, in welcher der Master Server eines Systems ausfällt. In einer solcher Situation muss wiederum ein Mensch hinzukommen und einen neuen Master Server bestätigen sowie die Datenqualität sicherstellen. Als alternative zu dieser Situation kommt der Raft-Konsens Algorithmus zum Zug. Dieser wurde von Diego Ongaro und John Ousterhout vorgestellt.

Im Raft-Konsens Algorithmus wählen die Server selbständig einen Server zu Ihrem Master und fallen anschliessend automatisch in die Rolle des Dieners. Wichtig hierbei zu akzeptieren ist, dass es sich nicht um Demokratische Wahlen in unserem Verständnis handelt. Die einzelnen Server sind froh sobald sich ein Master abzeichnet und allfällige gleichzeitig auftretende Wünsche nach einer neuen Wahl werden sofort beendet sollte sich ein Master zeigen.

Erkannt wird das ganze anhand eines sogenannten Heartbeats, welcher anzeigt, dass es sich bei einem Server um einen Master handelt.

Die Bereitstellung einer solcher Serverstruktur sollte immer in ungeraden Zahlen erfolgen, wobei drei und sieben als Grenzwerte verstanden werden können. Dies ist dem Umstand geschuldet, dass die Wahl zum Master immer ein mehrheitsentscheid sein muss. Nehmen wir an, dass wir 4 Server hätten. In diesem Konstrukt kann keine Entscheidung mehr getroffen werden, sobald 2 Server ausgefallen sind. Es besteht also kein Unterschied zu einem Konstrukt mit 3 Servern. Sobald wir jedoch 5 Server haben, kann wieder eine Wahl mit 2 ausgefallenen Servern stattfinden. Somit lohnt es sich immer nur eine ungerade Anzahl von Servern bereit zu stellen. 7 stellt wiederum einen Grenzwert dar, da laut den Autoren ab dieser Anzahl von Servern das Konstrukt kaum mehr Vorteile bringt, jedoch stark an Komplexität zunimmt.

6.2 App in Kubernetes

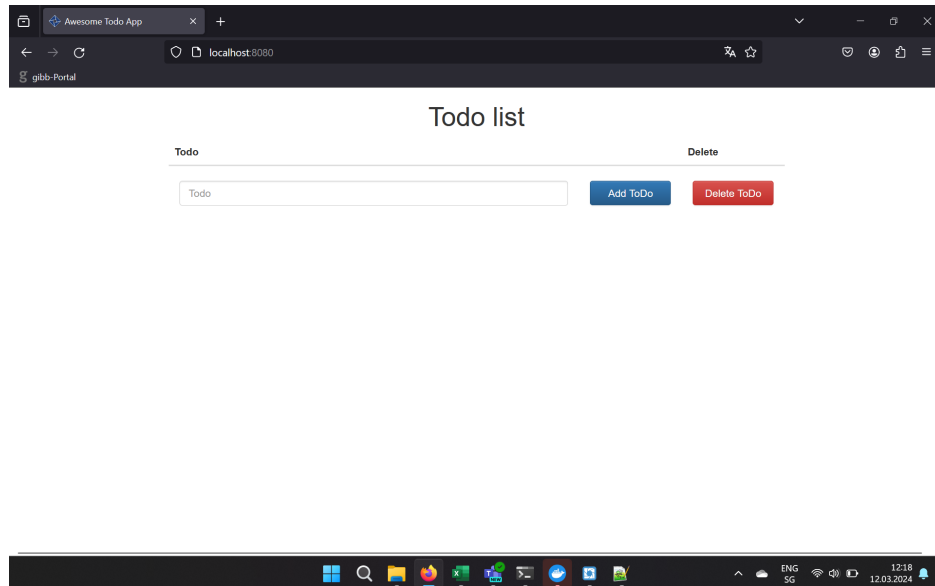


Abb. 6.1: App funktional über Kubernetes, Datum 12.März

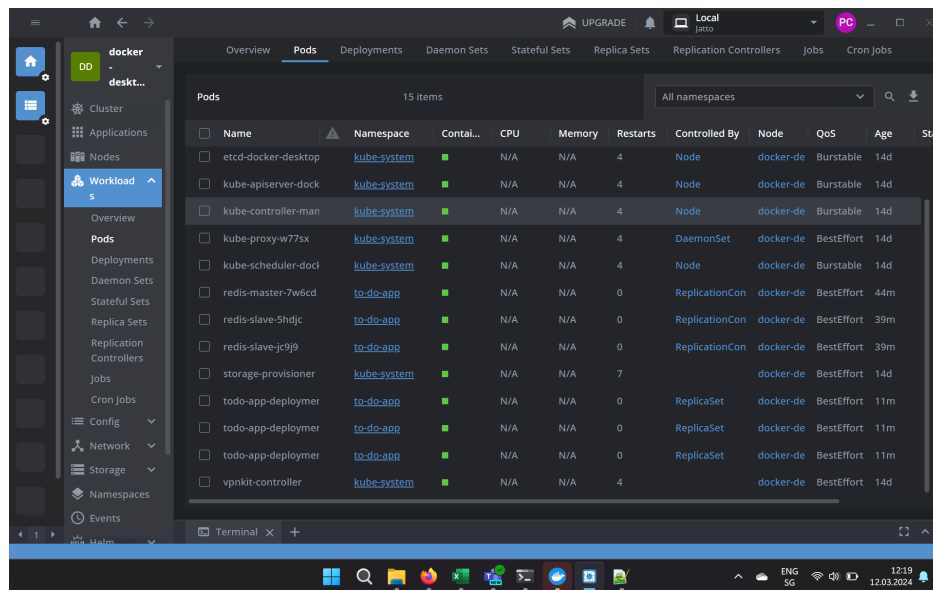


Abb. 6.2: Lens Ansicht der App, Datum 12.März

6.3 Self Healing

Self Healing bedeutet nichts weniger als die Erfüllung eines Traums. Ein richtig konfigurierter Kubernetes-Cluster kennt dank des eingestellten Befehls: `spec: replicas x` die benötigte Anzahl an Pods für einen Service. Durch regelmässige Kontrolle dieser Anzahl erkennt Kubernetes nun falls ein solcher Service nicht mehr richtig läuft oder sogar abgestürzt ist. Sollte ein solcher Zustand eintreffen, stellt Kubernetes den gewünschten Zustand wieder her, so dass wiederum die richtige Anzahl an Nodes vorhanden ist.

6.4 Scale Down

Beim Scale Down in Kubernetes geht es darum, die Anzahl der Replikate in einem Deployment zu reduzieren, um den aktuellen Anforderungen gerecht zu werden. Das Scale Down trägt dazu bei, die Ressourcennutzung zu optimieren und mögliche Kosten zu senken. Der Horizontal Pod Autoscaler (HPA) von Kubernetes kann diesen Prozess automatisieren, indem er zum Beispiel CPU- oder Speichernutzung überwacht und die Replikanzahl entsprechend anpasst. Eine manuelle Skalierung kann auch mit dem Befehl `kubectl scale` durchgeführt werden, wobei die gewünschte Anzahl von Replikaten angegeben wird. Effiziente Skalierungsstrategien sorgen für ein Gleichgewicht zwischen Ressourceneffizienz und der Möglichkeit, steigende Arbeitsbelastungen zu bewältigen. Doch dazu mehr im folgenden Kapitel bezüglich des Scale Ups.

6.5 Scale Up

Bei dem Scale Up in Kubernetes geht es darum, die Anzahl der Replikas in einem Deployment zu erhöhen, um einer höheren Arbeitslast gerecht zu werden. Dadurch wird sichergestellt, dass die An-

wendung auch bei steigendem Datenverkehr schnell und verfügbar bleibt. Kubernetes unterstützt die automatische Skalierung durch den Horizontal Pod Autoscaler (HPA), der die Anzahl der Pods basierend auf vordefinierten Merkmalen wie CPU- oder Speichernutzung anpasst. Eine manuelle Skalierung ist, gleich wie bei dem Scale Down, auch mit dem Befehl „kubectl scale“ möglich. Effektive Upscaling-Strategien tragen dazu bei, Leistung und Verfügbarkeit aufrechtzuerhalten, ohne Ressourcen zu überdimensionieren.

6.6 Funktionierendes Rolling Update

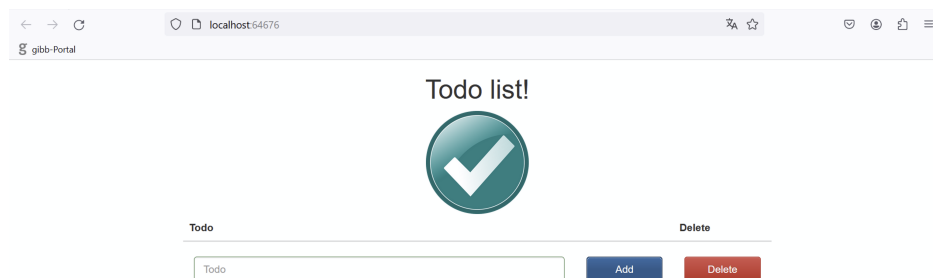


Abb. 6.3: Ansicht der Todo-AppV2 nach dem Rollingupdate, Datum 24.März

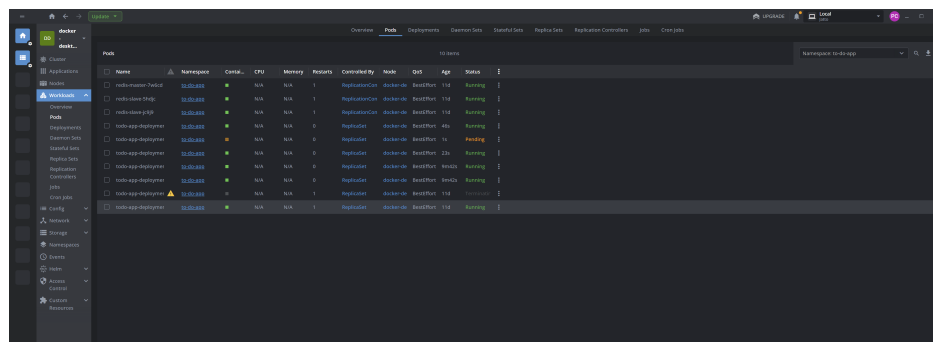


Abb. 6.4: Ansicht des laufenden Rollingupdates in Lens, Datum 24.März

6.7 Blue/Green Deployment

Das Blue / Green Deployment ist eine Strategie in Kubernetes, die Ausfallzeiten und Risiken minimieren soll, indem zwei identische Umgebungen ausgeführt werden. Dies unterscheidet das Blue / Green Deployment von dem normalen Scale Up, wo die Pods stückweise ersetzt werden. In einem Blue / Green Deployment gibt es eine Blaue Umgebung, in der die aktuelle Version der Anwendung gehostet wird, und eine Grüne Umgebung, in der die neue Version gehostet wird. Während der Bereitstellung wird der gesamte Datenverkehr zunächst an die Blaue Umgebung weitergeleitet. Sobald die Grüne Umgebung bereit und getestet ist, wird der Datenverkehr umgeschaltet. Dieser Wechsel kann mit minimaler bis gar keiner Ausfallzeit durchgeführt werden und bietet dem Nutzer einen nahtlosen Übergang. Ein weiterer Vorteil der Blue/Green-Bereitstellung ist die Möglichkeit eines schnellen Rollbacks auf die Blaue Umgebung, wenn Probleme mit der Grünen Version auftreten, wodurch eine hohe Verfügbarkeit und Zuverlässigkeit gewährleistet wird.

Kapitel 7

Tag 7

7.1 Eintrag zu Cluster IP und Node IP

„ClusterIP“ und „NodePort“ sind Servicetypen in Kubernetes welche die Kommunikation zu einem Cluster und innerhalb eines Clusters ermöglichen.

Ein „ClusterIP“-Service stellt eine virtuelle IP-Adresse bereit, über die innerhalb des Clusters auf den Dienst zugegriffen werden kann. Dies ist der standardmäßige Kubernetes-Service, der die interne Kommunikation zwischen Pods über verschiedene Knoten hinweg erleichtert. Wichtig hierbei ist, dass durch diesen Service die Pods der Aussenwelt nicht zur Verfügung gestellt werden. Somit ist der Service ideal für Anwendungen, auf die andere Teile desselben Clusters zugreifen müssen. Falls doch eine Verbindung von aussen möglich sein soll, kann dies durch einen Kubernetes-Proxy erreicht werden.

Ein „NodePort“-Service auf der anderen Seite erweitert die Fähigkeiten von „ClusterIP“ dahingehend, dass er die Anwendung über einen bestimmten Port auf allen Knoten des Clusters von außerhalb zugänglich macht. Wenn ein „NodePort“-Service erstellt wird, weist Kubernetes einen Port aus einem vordefinierten Bereich zu (Standard: 30000-32767). Als Nutzer kann ich anschliessend auf den Service mit `NodeIP:NodePort` zugreifen.

7.2 Eintrag zu LoadBalancer

In Kubernetes bietet ein LoadBalancer-Service die Möglichkeit, eine Anwendung dem externen Internet zugänglich zu machen. Es bestehen somit Überschneidungen zu den bisher erwähnten Services. Der LoadBalancer weist jedoch automatisch eine öffentliche IP-Adresse zu, leitet den externen Datenverkehr an den gegebenen Dienst weiter und verteilt ihn anschliessend auf die Pods. Zusätzlich prüft der LoadBalancer die Verfügbarkeit von Pods mit der Kubernetes API um sicherzustellen, dass diese ansprechbar sind. Diese Art von Service wird in der Regel von Cloud-Anbietern im Rahmen ihrer Kubernetes-Angebote bereitgestellt und ermöglicht den einfachen Zugriff auf Anwendungen von außerhalb des Kubernetes-Clusters. Der LoadBalancer-Service fungiert als Einstiegspunkt für externen Datenverkehr und kümmert sich um die Verteilung von Anforderungen, um eine hohe Verfügbarkeit und Skalierbarkeit von Anwendungen sicherzustellen [4].

7.3 TodoApp mit Ingress

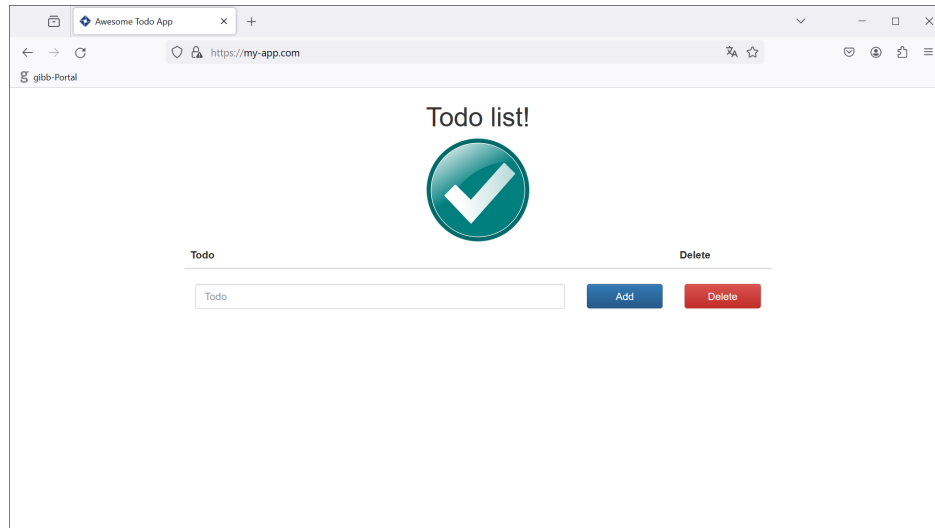


Abb. 7.1: Todo-App unter my-app.com, Datum 24.März

7.4 404 bei Aufruf mit Localhost

Der Ingress-Controller ist nun dafür verantwortlich, externen anfragen an die entsprechenden Dienste in unserem Cluster. Dieses Routing basiert auf dem Domänennamen welchen wir mit my-app.com festgelegt haben. Der Ingress-Controller verlässt sich also auf den Host-Header um zu wissen wohin die Anfrage weitergeleitet werden muss. Wenn wir nun direkt über 127.0.0.1, respektive localhost auf die Seite zu greifen möchten, kann der Ingress-Controller damit nichts mehr anfangen.

Um die Anwendung trotzdem über die Adresse „127.0.0.1“ zu erreichen, müssten wir den „Host“-Header in die Anfrage aufnehmen. Dies könnte man mit einem Tool wie „curl“ erreichen. Da wir aber spezifisch dieses Verhalten erreichen wollten, macht ein solches Vorgehen in unserem fälle wohl kaum Sinn.

7.5 Portainer auf Kubernetes

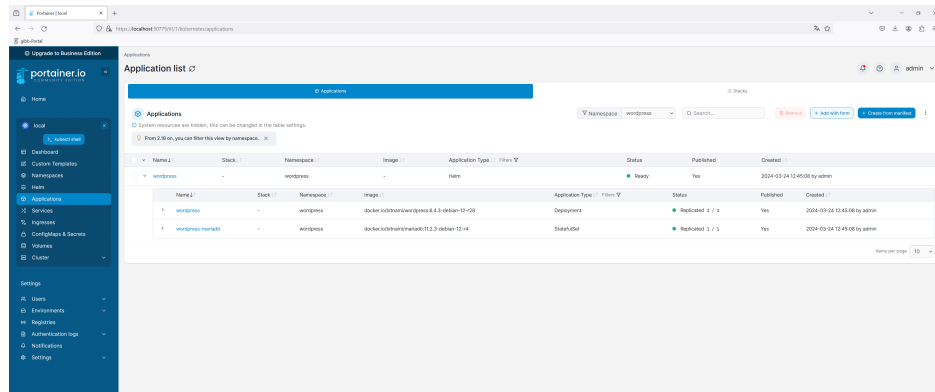


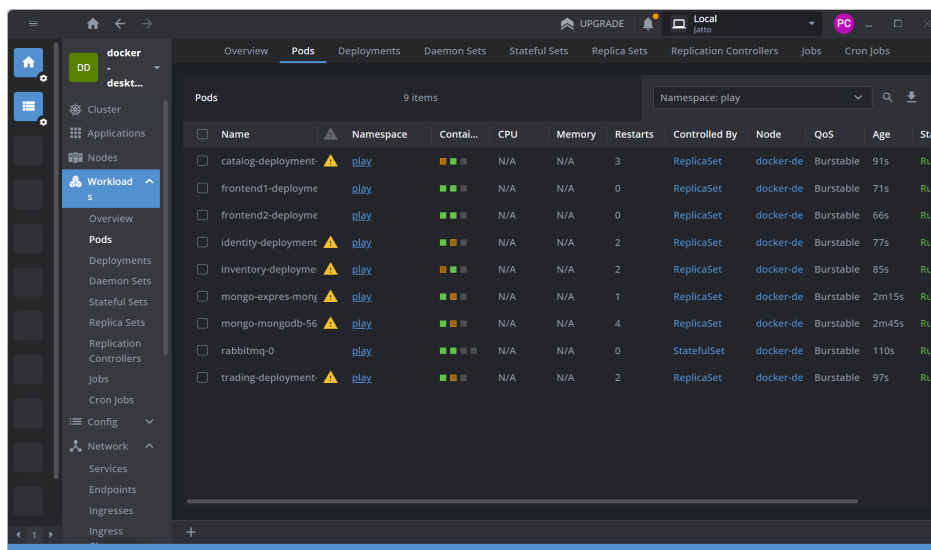
Abb. 7.2: Portainer auf Kubernetes Installiert mit laufendem Wordpress, Datum 24.März

Kapitel 8

Tag 8

8.1 Learning Beispiel auf Kubernetes

Bezüglich des Play Beispiels gilt es zu erwähnen, dass diese nicht vollständig funktional ist. Auch nach exzessivem probieren und Rücksprachen mit den anderen Teilnehmern der Klasse konnten die einzelnen Teile des Programms nicht korrekt gestartet werden. Es ist anzunehmen, dass es sich hierbei um ein Versionierung Problem handelt. Spezifischer geht es höchstwahrscheinlich um die Mongo-express Installation. Die genannte Version kann nicht funktional installiert werden. Wenn man eine neuere Version installiert funktioniert dies, jedoch scheinen die restlichen Pods des Docker-play Beispiels nicht damit klar zu kommen. Im sinne der Abgabe habe ich mich dazu entschieden, den Aufgaben weiter zu folgen und die nicht funktionalen Teile zu übergehen.



Name	Namespace	Contai...	CPU	Memory	Restarts	Controlled By	Node	QoS	Age	St
catalog-deployment	play	■ ■ ■	N/A	N/A	3	ReplicaSet	docker-de	Burstable	91s	Rt
frontend1-deployme	play	■ ■ ■	N/A	N/A	0	ReplicaSet	docker-de	Burstable	71s	Rt
frontend2-deployme	play	■ ■ ■	N/A	N/A	0	ReplicaSet	docker-de	Burstable	66s	Rt
identity-deployment	play	■ ■ ■	N/A	N/A	2	ReplicaSet	docker-de	Burstable	77s	Rt
inventory-deployme	play	■ ■ ■	N/A	N/A	2	ReplicaSet	docker-de	Burstable	85s	Rt
mongo-express-mong	play	■ ■ ■	N/A	N/A	1	ReplicaSet	docker-de	Burstable	2m15s	Rt
mongo-mongodb-56	play	■ ■ ■	N/A	N/A	4	ReplicaSet	docker-de	Burstable	2m45s	Rt
rabbitmq-0	play	■ ■ ■	N/A	N/A	0	StatefulSet	docker-de	Burstable	110s	Rt
trading-deployment	play	■ ■ ■	N/A	N/A	2	ReplicaSet	docker-de	Burstable	97s	Rt

Abb. 8.1: Verschiedene Probleme bei erstellen der Pods für das Learning Beispiel, Datum 24.März

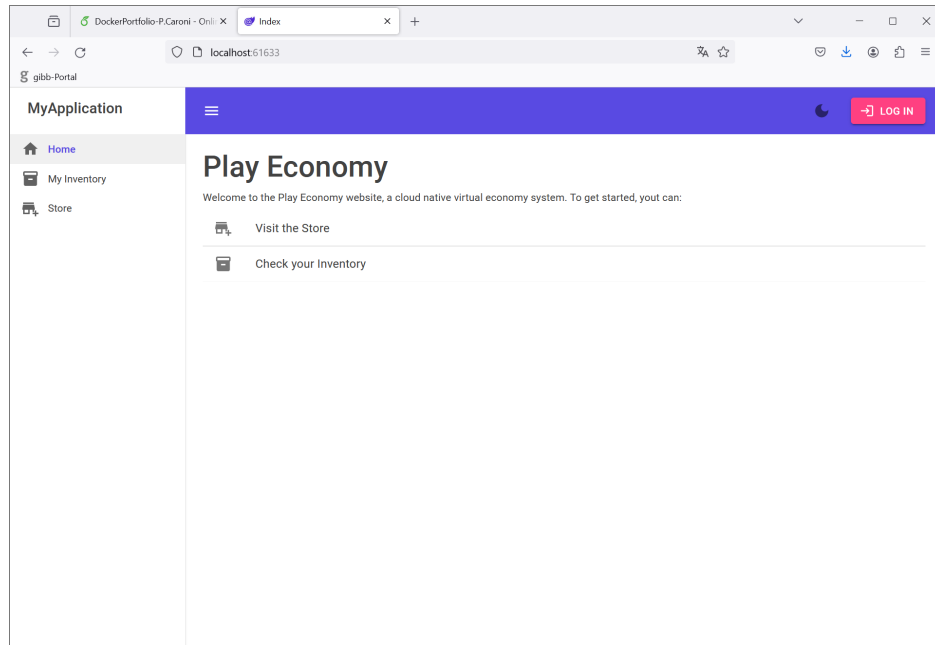


Abb. 8.2: Learning Beispiel teilweise funktional auf Kubernetes Installiert, Datum 24.März

8.2 Istio läuft

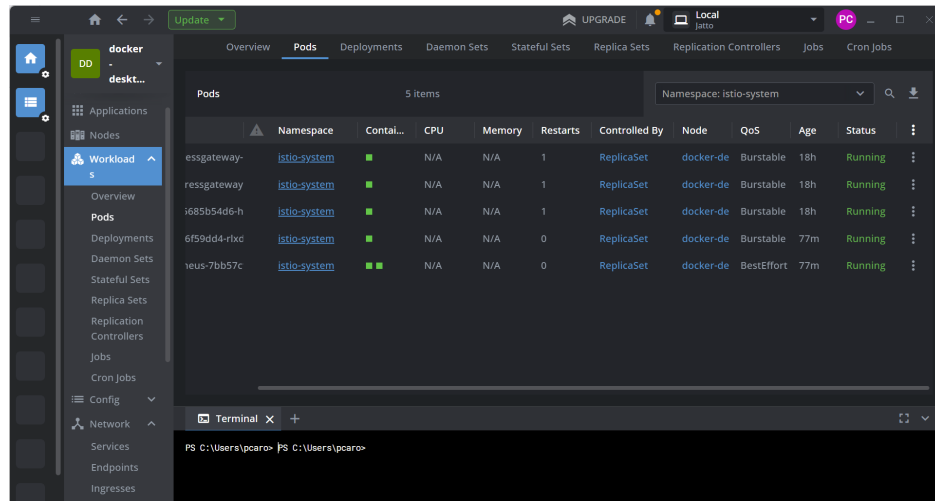


Abb. 8.3: Istio installiert, Datum 25.März

8.3 Kiali läuft

Kiali funktioniert im folgenden Bild ordnungsgemäss. Jedoch haben wir hier erneut das Problem des nicht funktionalen Docker-play Beispiels. Somit sind keine Daten für Kiali verfügbar.

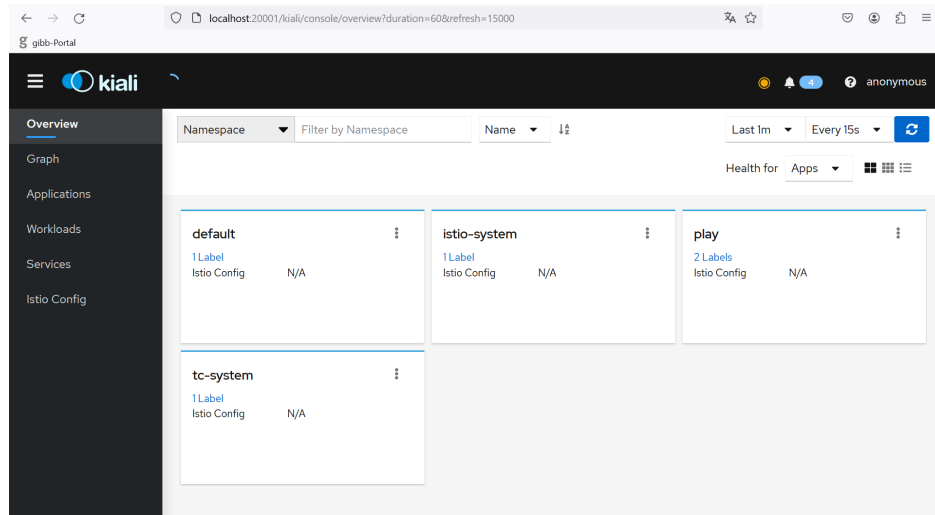


Abb. 8.4: Kiali installiert, Datum 25.März

8.4 Grafana läuft

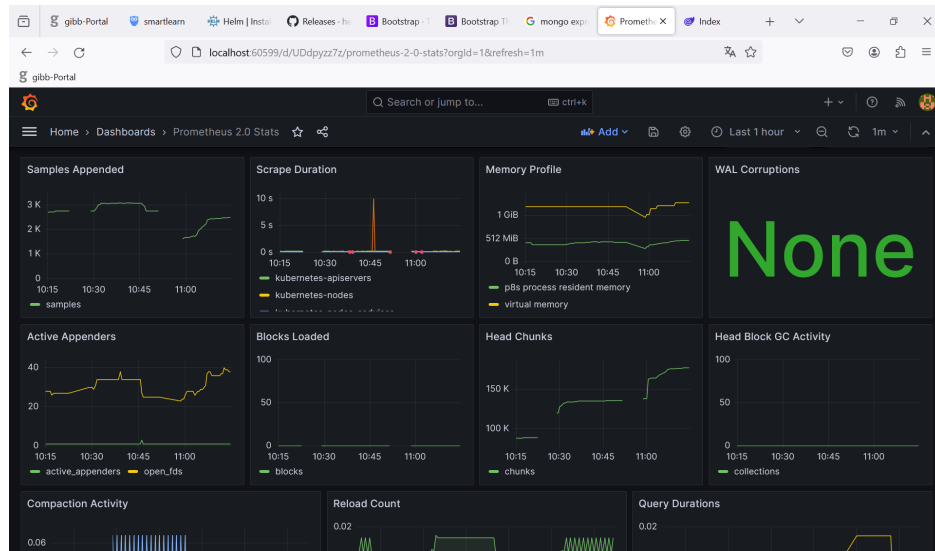


Abb. 8.5: Grafana überwacht die Nodes, Datum 25.März

Kapitel 9

Tag 9

9.1 Installation des eigenen Projektes auf Kubernetes

Zur installation des eigenen Projektes wurde zuerst ein Image aus der Website erstellt mit der Hilfe von docker build.

Im Anschluss wurde das Image in mein Git Repository gepushed mit dem Tag: website:v1.

Nachdem ich ein yaml File für das Aufsetzen der Anwendung erstellt habe (siehe Bild rechts) konnte diese ausgeführt werden.

Dazu erstellen wir zuerst einen Namespace für das Projekt:

```
kubectl create namespace website
```

Danach führen wir das erstellte yaml mit dem folgenden Befehl aus:

```
kubectl create -f website-deploy.yaml -n website
```

Nun können wir die Website über die Ports, welche wir in Lens finden können, aufrufen.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: website-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: website
  template:
    metadata:
      labels:
        app: website
    spec:
      containers:
        - name: website
          image: git-registry.gibb.ch/pca150880/caronidocker/website:v1
          ports:
            - containerPort: 80
```

Abb. 9.1: website-deploy.yaml

9.1. INSTALLATION DES EIGENEN PROJEKTES AUF KUBERNETES KAPITEL 9. TAG 9

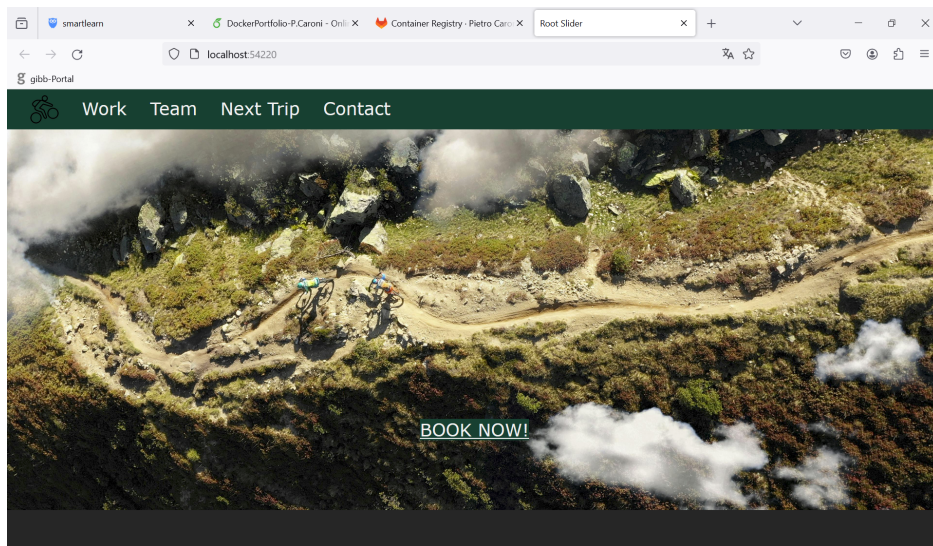


Abb. 9.2: Das eigene Projekt, eine Website, funktional, Datum 25.März

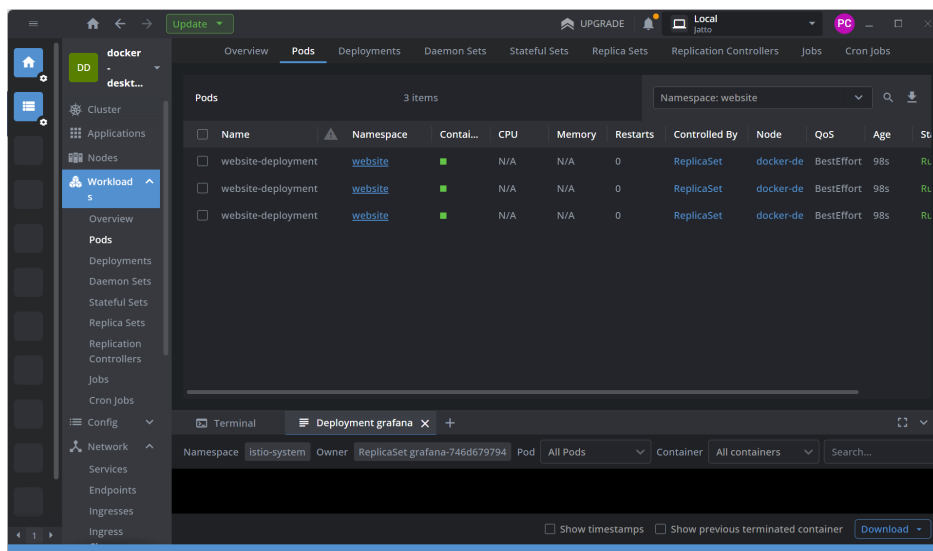


Abb. 9.3: Das eigene Projekt abgebildet auf Lens, Datum 25.März

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.>
- [3] Was sind Microservices?, URL: <https://aws.amazon.com/de/microservices/>
- [4] Kubernetes Load Balancer Definition <https://avinetworks.com/glossary/kubernetes-load-balancer/>