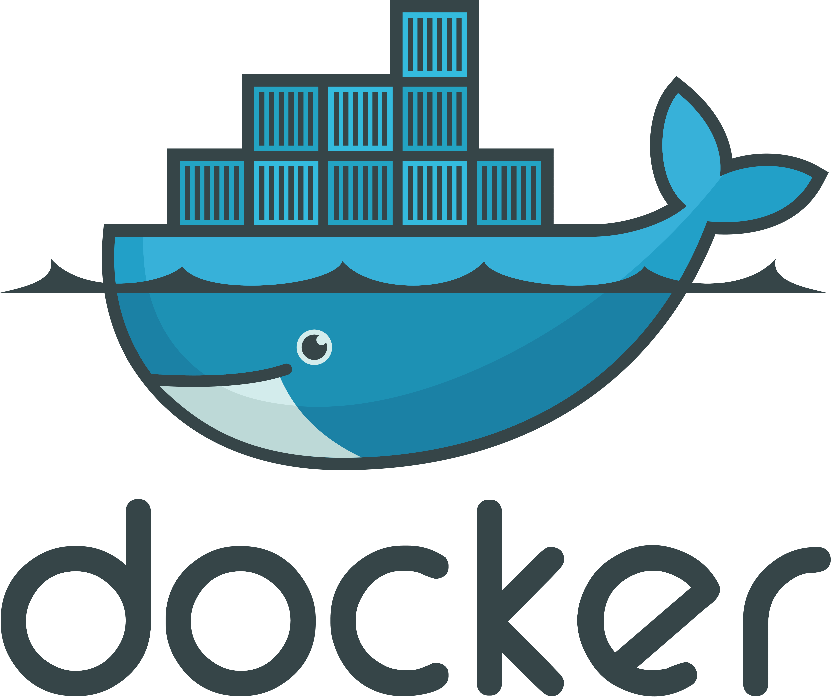
Dokumentation vom Modul 169 Tag 6

Erstellt von: Vladan Vranjes

Klasse Inf2022c

Kontakt: [vvr140992@stud.gibb.ch](mailto:vvr140992@stud.gibb.ch)



Contents

[Erläutern Sie was der Raft-Konsens-Algorithmus ist und warum es in einem Cluster immer eine ungerade Anzahl Server geben sollte 2](#_Toc161382977)

[Halten Sie fest, wie die App in Kubernetes läuft. 2](#_Toc161382978)

[Kurzer Eintrag zu Self Healing, Scale Down, Scale Up 2](#_Toc161382979)

[Self-Healing: 2](#_Toc161382980)

[Scale Down: 2](#_Toc161382981)

[Scale Up: 2](#_Toc161382982)

[PrintScreen, dass das Rolling Update funktioniert. Version 2 ist im Dashboard ersichtlich 2](#_Toc161382983)

[Eintrag zu Blue Green Deployment 2](#_Toc161382984)

# Erläutern Sie was der Raft-Konsens-Algorithmus ist und warum es in einem Cluster immer eine ungerade Anzahl Server geben sollte

Der Raft-Konsens-Algorithmus ist ein verteiltes Konsensprotokoll, das in verteilten Systemen wie verteilten Datenbanken oder zur Zustandsreplikation eingesetzt wird.

Er wurde entwickelt, um die Verwaltung von Konsistenz und Verfügbarkeit in einem Cluster von Servern zu vereinfachen.

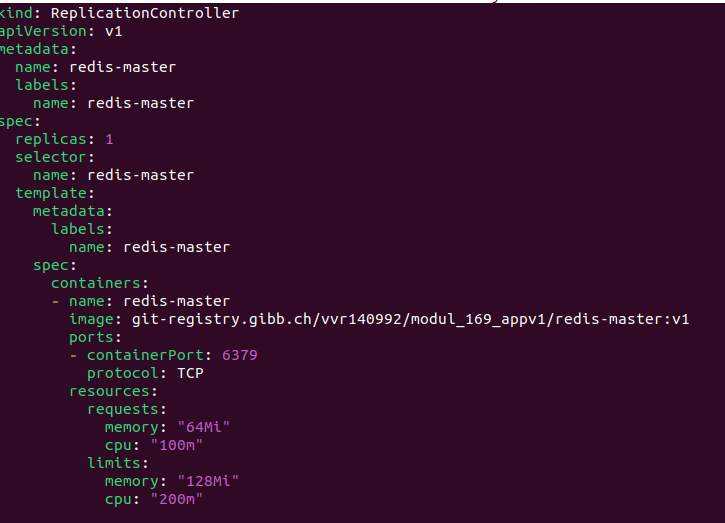
Es wird empfohlen, eine ungerade Anzahl von Servern in einem Raft-Cluster zu haben. Dies liegt daran, dass der Raft-Algorithmus in der Lage ist, Konsens zu erreichen, solange die Mehrheit der Server (die Hälfte plus eins) verfügbar ist. Mit einer ungeraden Anzahl von Servern wird vermieden, dass der Cluster in einem Pattzustand kommt, wenn eine Störung auftritt und genau die Hälfte der Server nicht erreichbar ist.

# Halten Sie fest, wie die App in Kubernetes läuft.

## Redis Master

Erstelle das File: Redis-master-controller.yaml

(Hier musst du dein image das du beim ersten Projekt genutzt hast aus dem Registry nutzen.)



Starte dann dieses File mit:

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

Wobei: Filename Namespace

Danach auch:

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

Namespaces anzeigen:

Kubectl get namespaces

Namespace ändern:

kubectl config set-context --current --namespace=

## Redis Slave

Hier dasselbe machen mit «redis-slave-controller»

Ein Bild, das Text, Screenshot enthält.

Automatisch generierte Beschreibung

Starte dann dieses File mit:

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

Danach auch:

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

## To do App

(Bild)

Startte das File mit:

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

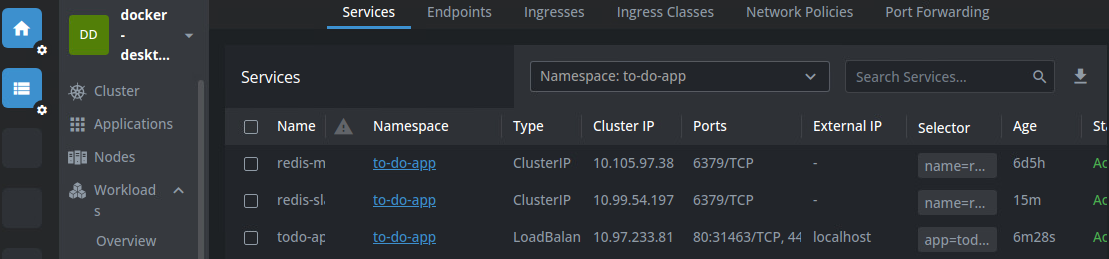
Teste ob die Pods am laufen sind:

kubectl get pods -l app=todo-app -n to-do-app

Sobald die Pods gestartet sind, richte die Services ein:

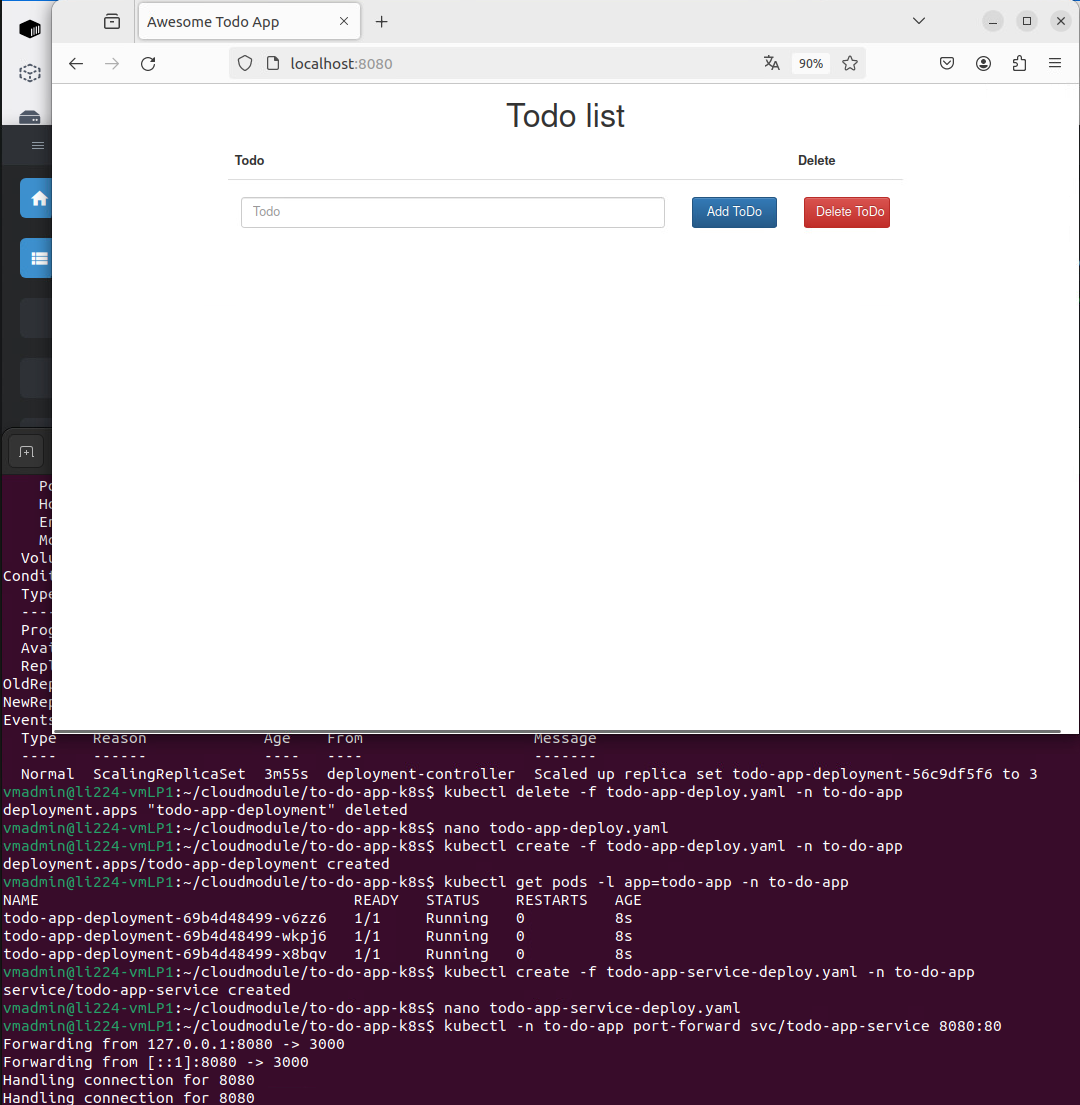
kubectl create -f todo-app-service-deploy.yaml -n to-do-app

Nun können wir den aktuellen Status des Clusters verifizieren. Die Endpoints sind Endpunkte, auf die, vorausgesetzt es handelt sich um "öffentlichte" bzw. erreichbare IP-Addressen, einfach zugegriffen werden kann mit Lens.



Nun sollten wir danach im Browser auf unsere App Zugriff haben:

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



# Kurzer Eintrag zu Self Healing, Scale Down, Scale Up

## Self-Healing:

Kubernetes bietet Mechanismen für die Selbstheilung von Anwendungen. Wenn ein Pod oder ein Container innerhalb eines Pods fehlschlägt, erkennt Kubernetes den Ausfall und startet automatisch einen neuen Pod oder Container, um den gewünschten Zustand wiederherzustellen.

## Scale Down:

Kubernetes ermöglicht das automatische Skalieren von Anwendungen basierend auf ihrer Auslastung. Wenn die Ressourcennutzung abnimmt und weniger Instanzen der Anwendung benötigt werden, skaliert Kubernetes die Anzahl der Pods oder Container automatisch nach unten, um Ressourcen zu sparen.

## Scale Up:

Umgekehrt skaliert Kubernetes bei steigender Auslastung automatisch die Anzahl der Pods oder Container nach oben, um eine ausreichende Kapazität und Leistungsfähigkeit der Anwendung sicherzustellen.

# Rolling Update funktioniert & Version 2 im Dashboard ersichtlich

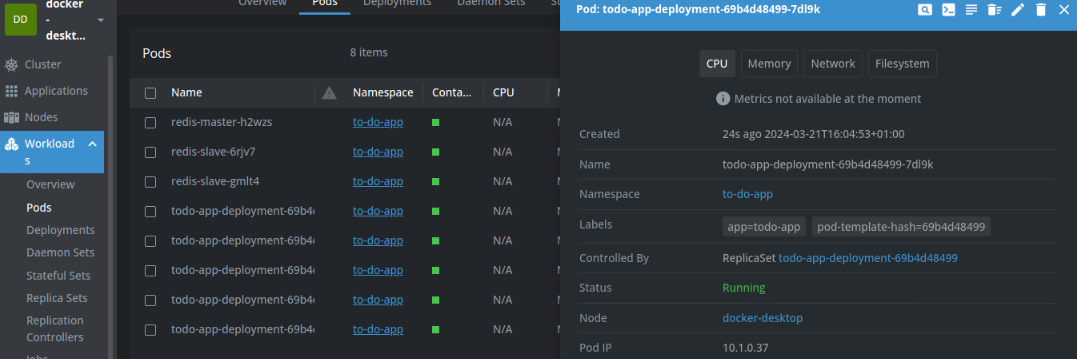
## Rolling Update

Bevor man eins ausführen kann, muss man zuerst die Anwendung hochskalieren.

kubectl scale --replicas=5 deployment todo-app-deployment -n to-do-app

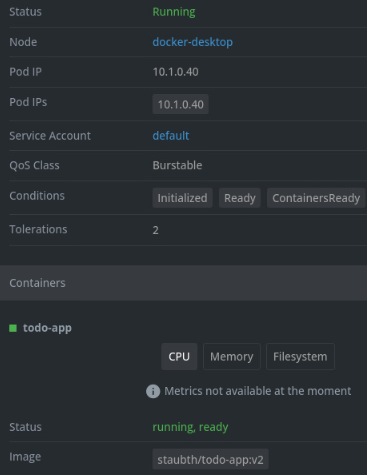
Dies erzeugt uns 5 Pods des Frontends.

Momentan läuft noch die Version 1 der App.



kubectl apply -f todo-app-deploy-v2.yaml -n to-do-app

Nun sehen wir wie nach und nach die Pots ersetzt werden.



Zuunterst steht nun auch bei Image: “v2”

# Eintrag zu Blue Green Deployment

Beim Blue-Green Deployment werden zwei identische Produktionsumgebungen parallel betrieben: eine "blaue" und eine "grüne" Umgebung. Die aktive Umgebung, die den Live-Verkehr erhält, wird als "blau" bezeichnet, während die neue Version der Anwendung in der "grünen" Umgebung bereitgestellt wird und getestet wird. Sobald die Tests abgeschlossen sind, wird der Verkehr von der "blauen" auf die "grüne" Umgebung umgeleitet, wodurch das Update ohne Downtime für die Benutzer durchgeführt wird.