STAATLICHE BERUFSSCHULE LAUINGEN
MIT BERUFSFACHSCHULE FÜR TECHNISCHE ASSISTENTEN FÜR INFORMATIK

Staatliches
Berufliches Schulzentrum
Höchstädt an der Donau

seele

Projektarbeit

Aufbau eines Kubernetes Clusters für Hardwareressourcen-sparendes Software Deployment im Fassadenbau bei der Firma seele

Philipp Zwick

Kooperationspartner
seele GmbH
Dr. Fabian Schmid
Gutenbergstraße 6
86368 Gersthofen

Staatliches Berufliches Schulzentrum
Fachschule für Umweltschutztechnik
und regenerative Energien
Prinz-Eugen-Straße 13
89420 Höchstädt an der Donau

12/05/2021

Inhaltsverzeichnis

1 Einleitung

Im Unternehmensumfeld werden Anwendungen für gewöhnlich jeweils auf eigenen Servern oder virtuellen Maschinen betrieben, um Wechselwirkungen zwischen Programmen auszuschließen, die Sicherheit der Umgebung zu erhöhen oder die Wahrscheinlichkeit eines Ausfalls zu verringern. Hierbei werden allerdings, bedingt durch den Overhead eines kompletten Betriebssystems und die vorzuhaltenden Ressourcen für Zeiten der Spitzenlasten, viele Hardwareressourcen verschwendet bzw. nicht optimal genutzt. Um dieser Problematik entgegen zu wirken, gibt es sogenannte Container-Technologien. Container verbrauchen weniger Ressourcen als klassische Server-Betriebssysteme, da bspw. der Kernel mit dem Host geteilt wird und Daten in einem Container im Regelfall nicht persistent gespeichert werden. Zusätzlich können Container mithilfe eines Orchestrators auch skaliert oder z.B. gegen Ausfälle resistent gemacht werden. Durch die Verwendung eines Containers wird hierdurch eine erhöhte Ausfallsicherheit des Systems sowie eine Einsparung von Hardwareressourcen ermöglicht.

Diese Projektarbeit befasst sich mit dem Thema „Aufbau eines Kubernetes Clusters für Hardwareressourcen-sparendes Software Deployment im Fassadenbau bei der Firma seele“. Dabei wird grundlegendes Wissen in den Bereichen Netzwerktechnik, Linux sowie Container und deren Orchestrierung vorausgesetzt.
Die Firma seele GmbH verwirklicht als international tätiges, mittelständisches Unternehmen komplexe Glasbaukonstruktionen. Die Unternehmensgruppe beschäftigt über 1000 Mitarbeiter an 13 Standorten weltweit.

Um die Hardwareressourcen der Firma seele optimal zu nutzen, wird im Rahmen dieser Arbeit ein Kubernetes Cluster konfiguriert. Als erste Testanwendung wird anschließend das derzeit in der Firma verwendete Ticketing System „osTicket“ inklusive einer MySQL Datenbank in das aufgebaute Cluster überführt.
Als Container Runtime findet containerd Verwendung, wobei Kubernetes den Orchestrator für die Container darstellt. Zusätzlich soll die Möglichkeit bestehen, in Zukunft neben dem Ticketing System noch weitere Software in Kubernetes zu migrieren, um eine noch höhere Ressourceneinsparung zu ermöglichen. Im vorliegenden Projekt wird deshalb ein Kubernetes Cluster, welches sich aus insgesamt vier

virtuellen Maschinen zusammensetzt, realisiert. Der virtuelle Server hierfür existiert bereits. Zudem wird ein externer, nicht im Cluster installierter Server, der als sog. genannter Load Balancer fungiert, benötigt.

Im Rahmen der Projektarbeit galt es die folgenden Forschungsfragen zu beantworten:

- Welche Konfigurationen müssen bei der grundlegenden Serverinstallation der VMs erfolgen?
- Unter Verwendung welcher Kubernetes Komponenten kann die Migration der Applikation „osTicket“ erfolgen?
- Welche Komponenten werden innerhalb des Kubernetes Clusters konfiguriert, um das Cluster auch für zukünftige Anwendungen vorzubereiten?
- Welche Einsparungen ressourcentechnischer Art können nach der Umsetzung erzielt werden?
- Welche weiteren Vorteile ergeben sich durch die Migration der Software osTicket?

2 Leistungsbeschreibung

2.1 Ist-Zustand

Im Moment werden bei der Firma seele viele Applikationen jeweils auf einzelnen Servern gehostet. Vereinzelt werden zwar auch mehrere Anwendungen auf einem Server verwaltet, jedoch ist das im Hinblick auf eine effiziente Ressourcennutzung nicht optimal, da das auf dem Server installierte Betriebssystem selbst bereits einige Ressourcen in Anspruch nimmt. So wird auch für das Ticket System „osTicket“ momentan ein eigener Server verwendet. Aufgrund der suboptimalen Nutzung der Hardware Ressourcen bietet sich Kubernetes als Werkzeug für die operative IT an. Zu Beginn dieser Arbeit gibt es noch keine Erfahrungen oder Testsysteme mit Bezug auf Kubernetes.

2.2 Grundlagen

Im Rahmen dieses Kapitels werden die Grundlagen für das Verständnis dieses Projektes erläutert. Nachfolgend werden die wichtigsten Begriffe/Prinzipien erklärt.

2.2.1 Container

- Container = virtuelle Maschine, die einer kompletten Anwendung inkl. Konfiguration und Abhängigkeiten entspricht
- Container Orchestration[3] = Prozess der Verwaltung und Organisation der Funktionsweise verschiedener Komponenten und Anwendungsebenen von Containern
- Container Runtime = Laufzeitumgebung, die im Hintergrund den eigentlichen Container ausführt
- Container Runtime Interface(CRI)[4] = Schnittstelle zwischen Orchestrator und Container Runtime
- containerd = Container Runtime

2.2.2 Kubernetes

- Kubernetes = Orchestrator
- kubelet[5] = Dienst, der auf einer Kubernetes Node läuft und dort die Container erstellt und überwacht

- kubectl = ermöglicht die Kontrolle eines Kubernetes Clusters über die Kommandozeile
- kubeadm = Kommandozeilentool zur Erstellung von Kubernetes Clustern
- Namespace = stellt einen Namensbereich als Abgrenzung von anderen Benutzern zur Verfügung (vgl. Linux Namespace)
- Pod = kann einen oder mehrere Container beinhalten (kleinste Einheit in Kubernetes)
- ReplicaSet(RS) = sorgt für die vordefinierte garantierte Verfügbarkeit von Pods
- Secret = stellt eine Speichermöglichkeit für sensible Informationen wie Passwörter zur Verfügung
- PersistentVolume(PV) = stellt einen vom Administrator provisionierten persistenten Speicher als Ressource im Cluster zur Verfügung
- PersistentVolumeClaim(PVC) = erstellt eine Speicherplatzanfrage für ein PV mit entsprechenden Eigenschaften (z.B. Speicherplatz, Zugriffsart)
- Deployment = beschreibt einen angestrebten Zustand und erreicht diesen durch die Verwaltung von ReplicaSets und Pods (z.B. Einspielen eines Updates auf einem Pod)
- Service = ist eine logische Abstraktion, die auf Basis von Filtern einen oder auch mehrere Pods unter einer konstanten DNS-Adresse erreichbar machen kann
- horizontale Skalierung = Erstellung/Abschaltung von zusätzlichen Pods je nach Auslastung des Deployments
- vertikale Skalierung = automatisiertes Hinzufügen von zusätzlichen Ressourcen (CPU,RAM) zu einem Pod
- HorizontalPodAutoscaler(HPA) = dient zur automatischen horizontalen Skalierung, hier werden je nach Auslastung (z.B. CPU, RAM) zusätzliche Pods gestartet bzw. wieder abgeschaltet
- Ingress Controller = dient zur Verwaltung von Ingressen
- Ingress[6] = veröffentlicht HTTP(S) Routen von außerhalb des Clusters auf einen Service innerhalb
- Metrics Server = dient zur Live-Überwachung von Container/Node Metriken wie CPU und RAM
- Rancher[7] = Verwaltungsplattform für Kubernetes, wird in diesem Projekt lediglich als GUI eingesetzt

2.3 Soll-Zustand

Zum Abschluss des Projekts sollen das Kubernetes Cluster sowie der externe Load-Balancer implementiert und funktionsbereit sein. Ein weiteres Ziel ist zudem die Migration der Anwendung „osTicket“ in das Cluster. Des Weiteren soll das Cluster für die Migration zukünftig benötigter Softwareanwendungen vorbereitet werden. Somit können am Ende auch die bereits eingangs erwähnten Forschungsfragen beantwortet werden.

2.4 Produktumgebung

Die benötigten virtuellen Maschinen stehen bereits zur Verfügung, weshalb keine zusätzliche Hardwarebeschaffung notwendig ist. Da mit Kubernetes, Ubuntu und HAProxy als LoadBalancer ausschließlich kostenfreie Software zum Einsatz kommt, ist auch eine Lizenzbeschaffung nicht notwendig.

3 Planung des Projekts

Zu Beginn des Projekts fanden Absprachen bezüglich des Umfangs und der Ziele statt. Hier wurde auch definiert, dass das Cluster lediglich als Testsystem für die Zukunft fungieren soll. Im Anschluss daran wurde sichergestellt, dass der entsprechende virtuelle Host zur Verfügung stand.
Nach der grundlegenden Einarbeitung in die Themen Container und Kubernetes wurde mit der Recherche zur konkreten Umsetzung begonnen. Hier stellte sich sogleich heraus, dass es in einer On-Premise Umgebung einen externen LoadBalancer geben muss, wohingegen dieser in einer Cloud Umgebung bereits automatisch innerhalb des Kubernetes Clusters vom Cloud Provider zur Verfügung gestellt wird. Der externe LoadBalancer ist notwendig, um einen Zugriff multipler Anwendungen auf ihren jeweiligen nativen Ports unter Verwendung von verschiedenen URLs zu ermöglichen. Andernfalls müsste jede Anwendung einen eigenen eindeutigen Port zugewiesen bekommen. Da dies für den Endbenutzer ungeeignet ist, wird der LoadBalancer mit implementiert.
Ein detaillierter Projektablaufplan für das Projekt findet sich unter Anhang A.

4 Projektdurchführung

4.1 Cluster Übersicht

In Abbildung 1 wird zunächst der logische Aufbau des Clusters in einer Übersichtsgrafik veranschaulicht und zeigt die Komponenten die in der Zielsetzung zum Projektstart für den Aufbau des Clusters in einer Grobfassung festgelegt wurden. Hierbei wurden die Docker Registry und Rancher als einzelne Docker Container auf dem Kubernetes Control Server (Kubernetesmaster) eingerichtet. Die drei Nodes wiederum erhalten von der API des Kubernetesmaster ihre Aufgaben (z.B. die Ausführung eines Pods). Fällt ein Node aus, so übernehmen die anderen Nodes die fehlenden Pods.

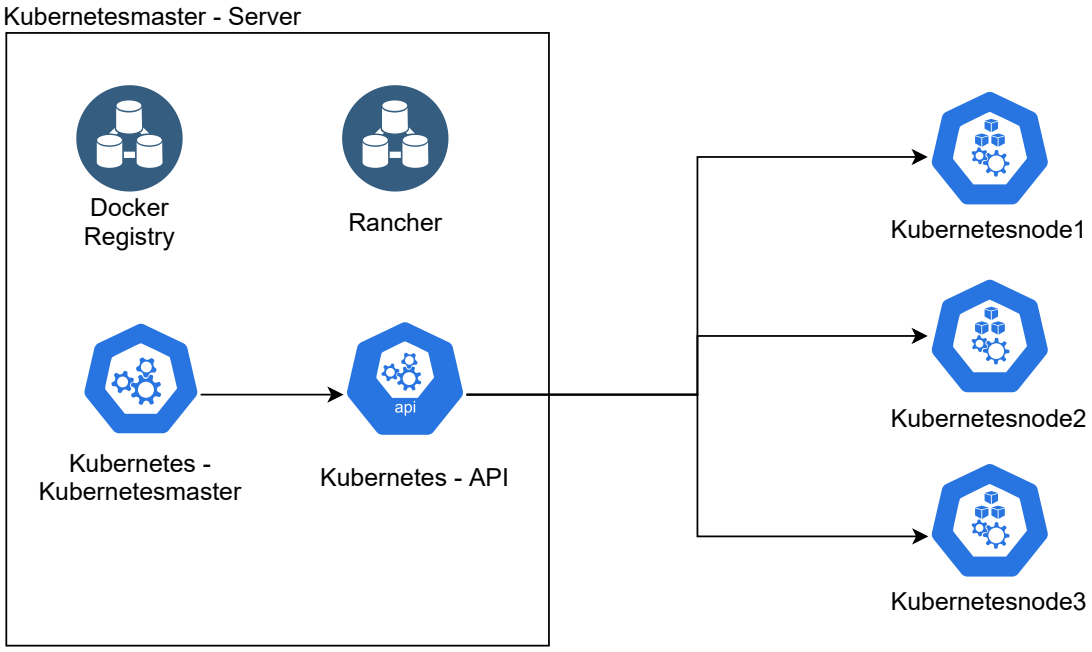


Abbildung 1: Logische Cluster Übersicht

Für eine Produtkivumgebung sollte die Docker Registry allerdings auf einem externen Server installiert werden, um eine höhere Verfügbarkeit sicherzustellen. Zudem ist es empfehlenswert, eine Hochverfügbarkeit des Kubernetesmasters durch einen zusätzlichen redundant bereitgestellten Control Server zu gewährleisten.

4.2 Implementierung

Während der Durchführung des Projektes stellte sich heraus, dass die ursprünglich geplante und zu diesem Zeitpunkt auch schon umgesetzte Konfiguration von Docker als Container Runtime von Seiten Kubernetes innerhalb des nächsten Jahres nicht mehr in der vorliegenden Form unterstützt wird. Um die Zukunftssicherheit des Systems und auch eine eventuelle Migration in ein späteres Produktivsystem nicht zu gefährden, wurde die Neuinstallation des Clusters unter Verwendung von containerd als Container Runtime beschlossen [8]. Nachfolgend wird nur die aktuelle Installation bzw. Konfiguration erläutert, da sich die Ursprüngliche nicht mehr im Einsatz befindet.
Diese unterteilt sich in:

- Grundinstallation
- Installation Docker auf Control Server
- Konfiguration Kubernetes
- Einrichtung Docker Registry
- Einrichtung Rancher
- Installation Metrics Server
- Installation Ingress Controller
- Installation/Einrichtung LoadBalancer
- Einrichtung NFS-Share
- Konfiguration osTicket

4.2.1 Grundinstallation

Im Folgenden wird nun die eingangs gestellte Fragestellung bearbeitet:
"Welche Konfigurationen müssen bei der grundlegenden Serverinstallation der VMs erfolgen?"
Nachfolgend werden die benötigten Komponenten und deren jeweilige Konfiguration spezifiziert.
Zu Beginn wurden auf dem virtuellen Host fünf Server mit Ubuntu 20.04 installiert und diese auf den neuesten Stand gebracht. Nachfolgende Schritte wurden nur auf vier von fünf Servern durchgeführt, da der fünfte Server als externer LoadBalancer konfiguriert wird und damit kein Kubernetes benötigt. Vor der Konfiguration von Kubernetes sind noch folgende Dinge zu beachten:

- Installation/Konfiguration von CRI-containerd
- Deaktivierung von SWAP
- Installation von kubelet, kubeadm und kubectl

4.2.1.1 Installation/Konfiguration von CRI-containerd

Die Installation von `containerd` erfolgte direkt nach den Vorgaben des Herstellers. Die detaillierte Installation kann dem Anhang B entnommen werden. Die Konfiguration wurde nun angepasst, damit der Daemon von `containerd` in Zukunft `systemd` statt `cgroupfs`, eine Docker-spezifische Implementierung eines control group managers, verwendet. Dies ist empfohlen, da die Hosts Ubuntu 20.04 als Betriebssystem verwenden und damit auf den Hosts systemd als init-Prozess ausgeführt wird. Deshalb wird vonseiten Kubernetes offiziell empfohlen, systemd anstelle von cgroupfs zu verwenden, da es zu Problemen kommen kann, wenn der Host von systemd verwaltet wird, aber das kubelet bzw. die Container Runtime mittels cgroupfs verwaltet werden.
Da selbst konfigurierte Container Images lokal erstellt und auf dem Control Server vorgehalten werden sollen, ist die Einbindung einer eigenen Registry vonnöten, welche aufgrund der Verwendung von HTTP manuell spezifiziert werden muss. HTTPS fand hier keine Verwendung, da diese Registry lediglich als Testsystem verwendet wird und nicht von extern erreichbar ist. Für die Übernahme der Konfiguration wurde der Dienst `containerd` neu gestartet.
Diese Einstellungen wurden in der Datei config.toml im Dateipfad `/etc/containerd/config.toml` angepasst. Wie in Abbildung 2 zu erkennen ist, werden lediglich zwei Bestandteile modifiziert:

- Verwenden von systemd statt cgroupfs
- Einbinden einer eigenen HTTP Registry

```
...
 [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc]
  ...
   [plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.
   options]
     SystemdCgroup = true
...
 [plugins."io.containerd.grpc.v1.cri".registry]
   [plugins."io.containerd.grpc.v1.cri".registry.mirrors]
    ...
     [plugins."io.containerd.grpc.v1.cri".registry.mirrors."Registry-
   IP:Port"]
            endpoint = ["http://Registry-IP:Port"]
...
```

Abbildung 2: Anpassungen in config.toml

4.2.1.2 Deaktivierung von SWAP

Hiernach wurde swap deaktiviert, da dies von Kubernetes nicht unterstützt wird. Dies geschah mittels folgender Befehle:

```
sudo swapoff -a
```

```
sudo sed -i 's/\/swap.img/#swap.img/g' /etc/fstab
```

Der erste Befehl sorgt für die Abschaltung von SWAP zur momentanen Laufzeit. Durch den zweiten Befehl wird SWAP auch über Neustarts hinweg deaktiviert. Siehe hierzu Anhang C.

4.2.1.3 Installation von kubelet, kubeadm und kubectl

Zum Abschluss der Grundinstallation wurden der Daemon `kubelet`, welcher bspw. für das Verwalten von Containern zuständig ist, und das Kommandozeilenprogramm `kubectl`, welches zur Verwaltung des Clusters dient, installiert. Zusätzlich war noch das Tool `kubeadm` zu installieren, um dem Cluster später die Nodes hinzufügen zu können. Die Installation erfolgte wiederum nach der offiziellen Dokumentation [9] und kann unter Anhang D gefunden werden.
Die bisher genannten Schritte wurden auf vier Servern ausgeführt, um eine identische Grundkonfiguration zu erhalten. Die Konfiguration des fünften Servers fand später statt, da die Funktionalitäten des Load Balancers für den Grundbetrieb des Clusters nicht erforderlich sind. Das NFS-Verzeichnis sollte auf dem Kubernetes Control Server gehostet werden, weshalb hier noch zusätzlich das Paket `nfs-kernel-server` eingerichtet wurde. Die Konfiguration dieses Pakets erfolgt erst unter 5.1.9 Einrichtung NFS-Share.

4.2.2 Installation Docker auf Control Server

Docker wurde nur auf dem Control Server installiert, da im späteren Verlauf die Docker Registry sowie Rancher, verwendet als GUI zur Verwaltung des Clusters, als Docker Container eingerichtet wurden. Die Konfiguration erfolgte wiederum nach den Vorgaben von Kubernetes [10], um eventuelle Kompatibilitätsprobleme auf dem Host ausschließen zu können, da Docker im Hintergrund auch auf containerd basiert. Hier war lediglich eine Erweiterung der Konfiguration des Docker Daemon um die geplante HTTP-Registry notwendig. Hierzu wurde dem Docker Daemon folgende Zeile in der JSON-Konfigurationsdatei hinzugefügt.

`"insecure-registries":["IP-CONTROL-SERVER:PORT"].`

Alle Anweisungen die zur Installation benötigt wurden, sind in Anhang E aufgeführt.

4.2.3 Konfiguration Kubernetes

Die für die Konfiguration von Kubernetes benötigten Abhängigkeiten `kubelet,kubectl` und `kubeadm` wurden bereits vollständig während der Grundinstallation eingerichtet. Die Konfiguration von Kubernetes selbst war nur auf dem Control Server notwendig. Die Einrichtung wurde nicht unter root ausgeführt, sondern mit einem anderen Benutzeraccount. Dieser wird aus Sicherheitsgründen nicht näher spezifiziert.
Zu Beginn musste mittels des Befehls `sudo mkdir -p $HOME/.kube` im Home Verzeichnis des Benutzers ein Ordner für das Abspeichern der Konfigurationsdatei angelegt werden.
Danach wurde mit `sudo kubeadm init` die Einrichtung des Clusters an Kubernetes übergeben. Hierfür ist die Konfiguration von `systemd` als Übergabeparameter sowie die Angabe des CRI-Sockets auf containerd notwendig, wie in Abbildung 3 dargestellt.

```
1 ...
2 nodeRegistration:
3   criSocket: /run/containerd/containerd.sock
4   ...
5
6 ---
7 ...
8 kind: KubeletConfiguration
9 cgroupDriver: systemd
```

Abbildung 3: Angepasste Konfigurationseinstellungen für kubeadm init

Mittels folgender Befehle wurde dann die Konfiguration (siehe Anhang F) an Kubernetes übergeben und das Cluster eingerichtet:

- Konfiguration des kubelet Daemons vor Einrichtung des Clusters

  ```
  sudo kubeadm init phase kubelet-start --config=kubeadmInit.yaml
  ```

- Neu laden des Daemons sowie neu starten des kubelets

  ```
  sudo systemctl daemon-reload && sudo systemctl restart kubelet
  ```

- Initialisierung des Clusters mit

  ```
  sudo kubeadm init --config=kubeadmInit.yaml
  ```

Nach erfolgreicher Installation des Clusters wird als Rückgabe ein Befehl zum Hinzufügen der Nodes in das Cluster ausgegeben (gekürzt):

`kubeadm join CONTROL-SERVER-IP:PORT --token abcdef.0xxxxxxxabcdef --discovery-token-ca-cert-hash sha256:sha256-Hash`

Um das Cluster auf dem Control Server mittels Kommandozeile (kubectl) verwalten zu können, mussten nun die Cluster Konfiguration in das vorher angelegte Verzeichnis kopiert und die Dateiberechtigungen angepasst werden. Die Berechtigungen wurden hierbei auf den ausführenden Benutzer sowie dessen Gruppe übertragen:

```
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Damit nun die Nodes in das Cluster hinzugefügt werden konnten, musste noch ein Netzwerkplugin in das Cluster eingefügt werden. Dieses übernimmt Aufgaben wie bspw. DNS innerhalb des Kubernetes Clusters und ist im Standardumfang von Kubernetes nicht enthalten. Hierfür fand das Project Calico Verwendung, da dies als einziges von Kubernetes offiziell unterstützt wird. Die Installation erfolgte via `kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml`. Diese Version der Project Calico Installation wurde ausgewählt, da diese für einen Einsatz von unter 50 Nodes gedacht ist [11].

Der Befehl `kubectl apply -f` ermöglicht die Anwendung einer oder mehrerer Dateien im .yaml Format innerhalb des Clusters. Hierbei werden von der Kommandozeilenapplikation die entsprechenden Parameter ausgelesen und in Kubernetes in die jeweiligen Abstraktionsebenen umgewandelt und umgesetzt.

Das Cluster war nun grundlegend konfiguriert und die Nodes wurden nun via `kubeadm join CONTROL-SERVER-IP:PORT --token abcdef.0xxxxxxxabcdef --discovery-token-ca-cert-hash sha256:sha256-Hash` (gekürzt) in das Cluster aufgenommen. Das komplette Skript findet sich unter Anhang G.

4.2.4 Einrichtung Docker Registry

Um selbst erstellte Docker Images und öffentliche Images offline zur Verfügung stellen und die Datenhoheit der erstellten Docker Images behalten zu können, war die Einrichtung einer Docker Registry notwendig. Diese ermöglicht den lokalen Up- und Download jeglicher Container-Abbilder.
Die Registry selbst läuft als einzelner Container direkt auf dem Kubernetes Control Server, da hier keine Hochverfügbarkeit benötigt wurde.
Die Erstellung der Registry wurde mit `sudo docker run -d -p 5000:5000 --restart=always -name registry registry:2` durchgeführt. Die Erläuterung der Parameter kann Tabelle 1 entnommen werden.

Einrichtung Registry	
Parameter	**Auswirkung/Bedeutung**
-d (detached)	Ausführung des Containers im Hintergrund
-p (port) 5000:5000	Mapping des Container-Ports 5000 auf den Host-Port 5000
--restart=always	Container startet immer automatisch neu (z.B. nach Host-Neustart)
--name=registry	Festlegen des Namens "registry" für Container
registry:2	Angabe des verwendeten Images

Tabelle 1: Aufschlüsselung des docker run Kommandos

Der Download der Images von der Registry erfolgt durch die Kubernetesnodes. Die Häufigkeit des Downloads hängt vom individuellen Deployment/Pod ab. Hierbei kann es sein, dass das Image bei jedem Start des Containers, bei Vorhandensein einer neueren Version oder nur wenn sich das Image noch nicht im Cache der entsprechenden Node befindet, heruntergeladen wird. Die Abbilder werden danach auf der jeweiligen Kubernetesnode lokal gecached.

4.2.5 Einrichtung Rancher

Um eine bessere Verwaltbarkeit sowie eine grafische Darstellung des Clusters zu erreichen, wurde die Software Rancher zusätzlich hinzugefügt. Diese läuft als Docker Container auf dem Control Server selbst. Die Einrichtung erfolgte mittels `sudo docker run -d --restart=unless-stopped -p 8080:443 --privileged rancher/rancher:latest` [12].
Die Parameter erklären sich analog zur Docker Registry. Zusätzlich jedoch musste hier noch der `--privileged` Parameter vergeben werden, der dem Container die glei-

chen Berechtigungen auf dem Host einräumt, wie der ausführende Benutzer des kubelet sie hat. Nun musste noch das Cluster zur Überwachung eingebunden werden. Dies erfolgte in der Weboberfläche über den Punkt „Add Cluster“. Hier wird von Rancher eine entsprechende .yaml Datei bereitgestellt, welche wiederum mittels `kubectl apply -f` auf dem Cluster eingebunden wurde. So wird bzw. werden nun auf jedem Node und dem Control Server ein bzw. mehrere Container zur Überwachung im Kubernetes Cluster erstellt. Danach ist das Cluster über Rancher verwaltbar. Das Aussehen der Oberfläche ist Anhang H bzw. I entnehmbar.

4.2.6 Installation Metrics Server

Die Installation des Kubernetes Metrics Servers ist erforderlich, da dieser die Grundlage für eine automatische Skalierung der Pods zur Verfügung stellt. Dies geschieht durch das Bereitstellen einer API, welche Daten zum CPU/RAM Verbrauch bereitstellt.

Die Installation erfolgte mittels:

```
kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download
/components.yaml
```

Eine weitere Konfiguration ist hier nicht erforderlich [13].

4.2.7 Installation Ingress Controller

Da in dem Cluster zukünftig verschiedenste Anwendungen laufen sollen, wurde ein Ingress Controller eingerichtet. Mittels Abbildung 4 wird nun der Aufbau beschrieben.

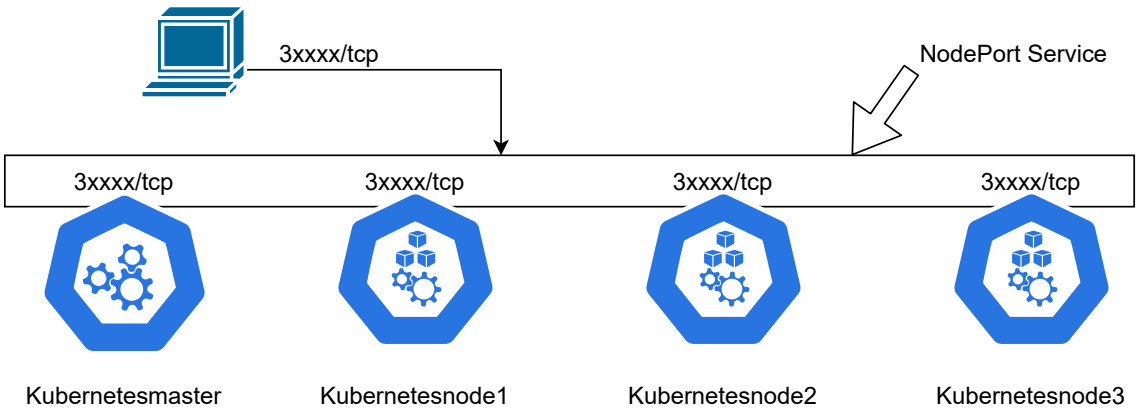


Abbildung 4: Ingress Controller

Wie aus der Grafik ersichtlich ist, wurde ein sog. NodePort Service im Portbereich 30000-32767 erstellt. Dieser NodePort Service dient zur Öffnung der Kubernetesnodes sowie des Kubernetesmaster nach außen. Zudem besteht damit nun die Möglichkeit, jeden Pod über jede Node zu erreichen, unabhängig davon wo dieser Pod gerade läuft. Dies wird dadurch ermöglicht, dass der NodePort Service die eingehenden Anfragen über einen Proxy Dienst verwaltet und dann an den entsprechenden Kubernetesnode weiterleitet. Damit muss allerdings der Client die Anfrage an den korrekten Port schicken und zudem muss dem Client eine externe IP eines Kubernetes Cluster Mitglieds bekannt sein.
Um diese Problematiken zu beseitigen, wurde, wie bereits eingangs erwähnt, ein externer LoadBalancer vor dem NodePort Service eingesetzt. Diese Vorgehensweise ist notwendig, da ein On-Premise Cluster nicht über die Kubernetes Funktionalität eines LoadBalancers verfügt. Dieses Feature ist lediglich in einer Cloud gehosteten Instanz verfügbar, sofern es vom Anbieter unterstützt wird. Zudem führt der Ingress Controller eine SSL-Termination durch, wodurch der externe Verkehr via SSL verschlüsselt wird, der Cluster-interne Datenverkehr unverschlüsselt erfolgen kann und sich somit die Konfiguration von Zertifikaten innerhalb des Clusters erübrigt. Die Installation des Ingress Controllers erfolgte via [14]:

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller
-v0.44.0/deploy/static/provider/baremetal/deploy.yaml
```

Eine weitere Konfiguration des Ingress Controllers ist nicht notwendig, jedoch wird bei der Konfiguration von osTicket ein Ingress erstellt.

4.2.8 Installation/Einrichtung LoadBalancer

Auf dem LoadBalancer(LB) wurde nun die Software HAProxy als LoadBalancer eingerichtet. Hierzu wurde zunächst das Repository via `sudo add-apt-repository ppa:vbernat/haproxy-2.2 --yes` hinzugefügt. Danach wurd via `sudo apt update` das eben eingebunden Repository aktualisiert und im Anschluss daran mittels `sudo apt install haproxy` die Software HAProxy installiert. Anschließend wurd die unter Anhang J spezifizierte Konfiguration unter `/etc/haproxy` eingebunden sowie der Service `haproxy` neu gestartet, um die Konfiguration zu übernehmen. Nachfolgend werden die angepassten Konfigurationsausschnitte kurz beschrieben.

Wie in Abbildung 5 zu sehen ist, werden alle auf Port 80 eingehenden HTTP Anfragen des Frontends automatisch an HTTPS auf Port 443 weitergeleitet. Zudem werden die Anfragen auf Layer 4 mitgeloggt und automatisch an das Kubernetes Backend weitergeleitet. Im Backend wird ein LoadBalancing anhand der Anzahl der momentan zur jeweiligen Node geöffneten Verbindungen durchgeführt. Die Node mit den wenigsten momentan bestehenden Verbindungen bekommt die neue Verbindung zugewiesen. Zudem wird ein SSL-Hello Paket an die Server bzw. den Ingress Controller geschickt, um die Erreichbarkeit via SSL zu prüfen. Auch wird die Verfügbarkeit der Nodes in einem regelmäßigen Zeitintervall überprüft. Der Control Server wird nicht verwendet, da dieser keine vom Benutzer erstellten Pods hostet.

```
frontend kubernetes_http
  bind *:80
  mode http
  redirect scheme https if !{ ssl_fc }

frontend kubernetes_https
  bind *:443
  option tcplog
  mode tcp
  default_backend kubernetes

backend kubernetes
  mode tcp
  balance leastconn
  option ssl-hello-chk
  server kubernetesnode1 IP:3xxxx check
  server kubernetesnode2 IP:3xxxx check
  server kubernetesnode3 IP:3xxxx check
```

Abbildung 5: Front-/Backend für HAProxy

In Abbildung 6 ist unter der URL `http://IP:8080/stats` eine GUI zur Live Überwachung der Auslastung sowie deren Verbindungsanzahl verfügbar. Hier erfolgt eine Authentifizierung mittels der spezifizierten Zugangsdaten. Für ein Bild der GUI siehe Anhang K.

```
listen stats  #Used for interface to view live statistics of HAProxy
  bind IP-LB:8080
  stats enable
  stats hide-version
  stats refresh 30s
  stats show-node
  stats auth user:password
  stats uri /stats
```

Abbildung 6: Einstellungen für Live Statistiken des HAProxy

Der Aufbau mit LoadBalancer stellt sich nun wie in Abbildung 7 gezeigt dar. Hierbei wird der LB lediglich vor das Cluster bzw. den NodePort Service geschalten und bewirkt ein Port Remapping der eingehenden Anfragen auf den Port 3xxxx (HTTPS) des NodePort Services. Zu beachten ist hierbei, dass ein Zugriff auf die Anwendungen des Kubernetesclusters über die IP des LB nicht erfolgen kann, da die Weiterleitung der Anfrage an den entsprechenden Service im Cluster vonseiten des Ingress Controllers auf Basis der vom Client angefragten URL erfolgt.
Die Vorteile dieser Konfiguration sind wie folgt:

- eine zentralisierte Schnittstelle nach außen
- Clients können über Standardports zugreifen
- Load Balancing der Anfragen auf das Cluster
- Weiterleitung der Anfragen auf URL-Basis

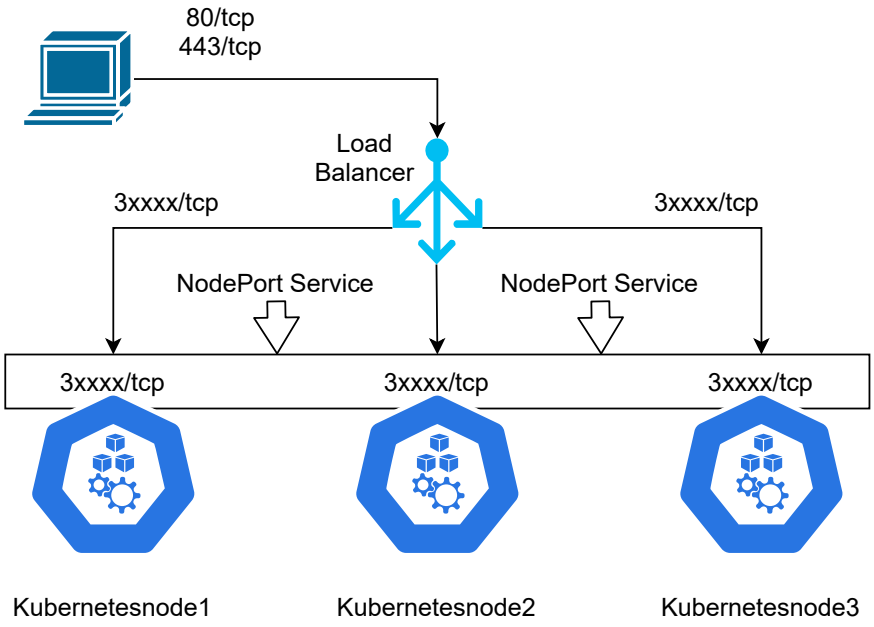


Abbildung 7: Ingress Controller mit Load Balancer

Exemplarischer Ablauf einer Anfrage:

1. Senden einer Anfrage eines Clients an LB in Form einer URL (bspw. https://ticket.osticket.info)
2. LB leitet Anfrage via LeastConnection LoadBalancing über eine geproxiete Node an entsprechenden Ingress Controller Port (3xxxx) weiter
3. Ingress Controller leitet anhand der URL die Anfrage an entsprechenden Service innerhalb des Clusters weiter
4. Service leitet die Anfrage an entsprechenden Pod weiter (bei multiplen Pods würde der Service selbst als LoadBalancer zwischen den Pods fungieren)

4.2.9 Einrichtung NFS-Share

Da die benötigten Komponenten für die Verwendung von NFS bereits installiert wurden, musste lediglich die Konfiguration durchgeführt werden. Hierzu war es ausreichend, die Datei `/etc/exports` anzupassen (siehe Abbildung 8) und danach den Befehl `sudo exportfs -ra` auszuführen, um die Konfiguration anzuwenden [15].

```
1 /mnt/nfs_share       IP1(rw,sync,no_subtree_check,no_root_squash)
2 /mnt/nfs_share       IP2(rw,sync,no_subtree_check,no_root_squash)
3 /mnt/nfs_share       IP3(rw,sync,no_subtree_check,no_root_squash)
4 /mnt/nfs_share       IP4(rw,sync,no_subtree_check,no_root_squash)
```

Abbildung 8: Ausschnitt Exports Datei für NFS-Share Konfiguration

Die komplette Datei `exports` findet sich unter Anhang L.

4.2.10 Konfiguration osTicket

Zu Beginn wurde für die Migration der Software osTicket das Datenbankverzeichnis aus `/var/lib/mysql` kopiert. Im Anschluss daran erfolgt nun die detaillierte Beschreibung der erstellten Kubernetes Komponenten. In diesem Zusammenhang wird auch die Forschungsfrage "Unter Verwendung welcher Kubernetes Komponenten kann die Migration der Applikation „osTicket“ erfolgen?" beantwortet.
Die nachfolgenden generellen Beispiele der einzelnen Komponenten stellen lediglich Auszüge aus den Möglichkeiten dar und erheben keinen Anspruch auf Vollständigkeit. Zudem finden nicht alle erwähnten Parameter in jeder Konfiguration Verwendung.
Sämtliche im Folgenden beschriebenen .yaml Konfigurationen wurden mit `kubectl apply -f` auf das Cluster angewandt.

4.2.10.1 Konfiguration osTicket - Namespace

Als erstes wurde ein eigener Namespace „osTicket“ erstellt (siehe Anhang M), um die Abtrennung von anderen zukünftigen Anwendungen zu gewährleisten und damit Zugriffe nur von in dem Namespace befindlichen Anwendungen zu erlauben. Ein generelles Beispiel stellt Tabelle 2 dar.

Attribut	Attributwert	Auswirkung
apiVersion:	v1	definiert API Version
kind:	Namespace	spezifiziert Art der Konfiguration
metadata:		legt Metadaten fest
name:	osticket	definiert den Namen

Tabelle 2: Aufschlüsselung einer Namespace .yaml Konfiguration

4.2.10.2 Konfiguration osTicket - Secret

Im Anschluss daran erfolgte die Konfiguration der Zugangsdaten für die MariaDB, einer relationalen Open-Source Datenbank, und für osTicket mittels eines Secrets. Eine detaillierte Beschreibung aller verwendeten Komponenten eines Secrets findet sich unter Tabelle 3.

Attribut	Attributwert	Auswirkung
apiVersion:	v1	definiert API Version
kind:	Secret	spezifiziert Art der Konfiguration
metadata:		legt Metadaten fest
name:	secret	definiert den Namen
namespace:	osticket	legt den Namespace fest
data:		definiert Inhalt des Secrets
variable:	Wert	weist Wert einer Variablen zu
variable2:	Wert2	weist Wert einer Variablen zu

Tabelle 3: Aufschlüsselung einer Secret .yaml Konfiguration

Hierbei wurden für osTicket und MariaDB zweimal dieselben Daten für `MYSQL_USER` und `MYSQL_PASSWORD` definiert (siehe Anhang N). Dies dient später dem Zugriff von osTicket auf die Datenbank. Für MariaDB wurde zusätzlich noch ein root Passwort in `MYSQL_ROOT_PASSWORD` spezifiziert. Aus Sicherheitsgründen wurden die Base64 codierten Passwörter aus dieser Dokumentation entfernt.

4.2.10.3 Konfiguration osTicket - PV/PVC

Bevor die Migration erfolgen konnte, mussten noch einige PersistentVolumes (PV) sowie die entsprechenden PersistentVolumeClaims (PVC) erstellt werden. Die konkrete Verwendung der PVCs wird beim Deployment der Anwendungen erläutert. Unter Tabelle 4 findet sich die Beschreibung der generellen verwendeten Komponenten eines PVs.

Attribut	Attributwert	Auswirkung
apiVersion:	v1	definiert API Version
kind:	PersistentVolume	spezifiziert Art der Konfiguration
metadata:		legt Metadaten fest
name:	volume	definiert den Namen
namespace:	osticket	legt den Namespace fest
spec:		spezifiziert den Inhalt
capacity:		definiert Kapazität
storage:		Festlegung des Speicherplatzes
volumeMode:	Filesystem	definiert Volume als Dateisystem
accessModes:	ReadWriteOnce (ReadWriteMany)	Lese-/Schreibzugriff für ein (mehrere) Gerät(e) zur selben Zeit
persistentVolumeReclaimPolicy:	Retain	Daten werden bei Löschung des Volumes beibehalten
storageClassName:	nfs	definiert NFS als StorageClass
mountOptions		Festlegung der Mounting Einstellungen
hard		bei Ausfall kontinuierliche Neuverbindung; Vermeidung von Datenkorruption
nfsver	4.1	Festlegung der NFS Version
nfs		NFS-spezifische Einstellungen
path	/nfs_share/osticket/xxx	Pfad zum NFS Share
server	IP	Definition IP des NFS-Servers

Tabelle 4: Aufschlüsselung einer PV .yaml Konfiguration

Das PV deklariert den Zugriff auf ein 10Gi großes NFS Share, welches extern gehostet wird. Dieser externe Zugriff ist von jeder Node erreichbar und ermöglicht dadurch das Starten des Pods auf jeder Kubernetesnode. Hierbei ist zu beachten, dass für MariaDB als Zugriffsmodus `ReadWriteOnce` implementiert wurde, um andere Zugriffe bereits auf dieser Ebene zu verhindern, da es bei Datenbanken ohne einen Datenbankbroker zu einer Dateninkonsistenz kommen kann.
Um allen osTicket Instanzen/Pods gleichzeitigen Zugriff zu ermöglichen, wurden diese PVs/PVCs mit `ReadWriteMany` konfiguriert. Unter Tabelle 5 findet sich die generelle Aufschlüsselung der verwendeten Komponenten eines PVCs.

Attribut	Attributwert	Auswirkung
apiVersion:	v1	definiert API Version
kind:	PersistentVolumeClaim	spezifiziert Art der Konfiguration
metadata:		legt Metadaten fest
name:	volume-claim	definiert den Namen
namespace:	osticket	legt den Namespace fest
spec:		spezifiziert den Inhalt
storageClassName:	nfs	definiert NFS als StorageClass
accessModes:	ReadWriteOnce (ReadWriteMany)	Lese-/Schreibzugriff für ein (mehrere) Gerät(e) zur selben Zeit
resources:		Spezifikation der Ressourcen
requests:		Festlegung des anzufragenden Speicherplatzes
storage:	10Gi	Festlegung des Speicherplatzes

Tabelle 5: Aufschlüsselung einer PVC .yaml Konfiguration

Für die zwei Deployments MariaDB und osTicket werden insgesamt drei verschiedene persistente NFS Speicher (PVs) sowie die zugehörigen PVCs benötigt:

- 10Gi für MariaDB Datenbank
- 1Gi für osTicket Plugins
- 1Gi für osTicket Languages

Unter Anhang O ist die exakte Konfiguration dieser drei persistenten NFS Speicher zu finden.

4.2.10.4 Konfiguration osTicket - MariaDB - Deployment

Damit die Container der Datenbank und des Ticketsystems automatisiert upgedated und skaliert werden können, wurde für MariaDB und osTicket jeweils ein eigenes Deployment konfiguriert. Unter Tabelle 6 ist die grundlegende Konfiguration eines Deployments ersichtlich.

Attribut	Attributwert	Auswirkung
apiVersion:	v1	definiert API Version
kind:	Deployment	spezifiziert Art der Konfiguration
metadata:		legt Metadaten fest
name:	deployment	definiert den Namen
namespace:	osticket	legt den Namespace fest
spec:		spezifiziert den Inhalt
selector:		definiert Filter
matchLabels:		definiert Filterbedingung (Label enthält)
app:	xxx	definiert Filterbedingung (Label enthält: app =xxx)
replicas:	xxx	Definition der Standardmäßigen Anzahl der Repliken
template:		Festlegung der Vorlage für Pod
metadata:		legt Metadaten fest
labels:		ermöglicht Label Erstellung
app:	xxx	legt Label fest
spec:		Festlegung der Pod-Spezifikationen
volumes:		festlegen der einzubindenden Volumes
- name:	xxx-storage	Angabe eines Volume-namens
persistentVolumeClaim:		Festlegung des zu verwendenden PVC
claimName:	xxx-claim	Definition des PVC Namens
containers:		Erstellung ein oder mehrer Container
- name:	xxx-container	Definition des Container Namens
image:	xxx-image	Definition des Container Images
ports:		Festlegung der zu öffnenden Container Ports
- containerPort:	xxx	Spezifikation des Ports
name:	Port für xxx	Spezifikation der Portbeschreibung
resources:		Festlegung der Ressourcen für Container
limits:		Definition eines CPU Limits
cpu:	x oder 0.x	Angabe CPU Kerne
requests:		Definition einer min. CPU
cpu:	x oder 0.x	Angabe CPU Kerne
volumeMounts:		einbinden des Volumes in Container
- mountPath:	"/xxx/xx"	Festlegung des Pfades im Container
name:	xxx-storage	Festlegung des Volumes (siehe oben festgelegtes Volume)
env:		Definition von Umgebungsvariablen
- name:	xxx-var	Angabe des Namens
valueFrom:		Quellenangabe des Wertes
secretKeyRef:		Referenz auf Kubernetes secret
name:	xxx-secret	Name des secrets
key:	xxx-key	Name des Schlüssels im Secret
- name:	xxx-var2	Angabe des Namens
value:	"var2-Inhalt"	manuelle Festlegung des Wertes

Tabelle 6: Aufschlüsselung einer Deployment .yaml Konfiguration

Das MariaDB Deployment findet sich in Anhang P(Zeilen 16-61). Hierbei werden folgende Haupteinstellungen gesetzt:

- Einbinden des `mysql-pv-claim` PVC
- Mounten des PVC nach `/var/lib/mysql`
- Festlegen des Docker Images auf `mariadb`
- Öffnen des SQL-Ports 3306
- Übergeben der Umgebungsvariablen `MYSQL_ROOT_PASSWORD`, `MYSQL_USER` sowie `MYSQL_DATABASE` mit Werten aus dem zuvor definierten Secret
- Manuelle Übergabe der Umgebungsvariable `MYSQL_DATABASE` mit dem Wert `osticket` zur Verwendung der Datenbank „osticket“

Mit diesen Einstellungen war das MariaDB Deployment nun komplett. Da die Datenbankdateien bereits in einem vorherigen Schritt in das NFS-Verzeichnis kopiert wurden, war hier der Zugriff nun bereits möglich.

4.2.10.5 Konfiguration osTicket - MariaDB - Service

Um das Deployment jedoch überhaupt intern im Cluster erreichen zu können, war die Erstellung eines Services vonnöten. Dieser leitet im vorhandenen Fall den ankommenden Datenverkehr von Port 3306 auf den Pod internen Port 3306 weiter. Der Typ `ClusterIP` des Services bedeutet lediglich, dass dieser Service nur innerhalb des Clusters erreichbar ist und nicht nach außen geroutet wird. Die exakte Konfiguration des Services befindet sich in Anhang P(Zeilen 1-14). In nachfolgender Tabelle 7 ist der grundlegende Aufbau eines Services beschrieben.

Attribut	Attributwert	Auswirkung
apiVersion:	v1	definiert API Version
kind:	Service	spezifiziert Art der Konfiguration
metadata:		legt Metadaten fest
name:	service	definiert den Namen
namespace:	osticket	legt den Namespace fest
spec:		spezifiziert den Inhalt
selector:		definiert Filterbedingung
app:	xxx	definiert Filterbedingung (Label enthält: app = xxx)
ports:		definiert das PortMapping von Intern nach Extern
- protocol:	xxx	definiert das zu wählende Protokoll
port:	xxx	empfängt Daten auf Port xxx
targetPort:	xxx	leitet empfangene Daten an Port xxx weiter an Container, von denen Filterbedingung erfüllt wird
type:	xxx	setzt Service auf gewählten Typ

Tabelle 7: Aufschlüsselung einer Service .yaml Konfiguration

4.2.10.6 Konfiguration osTicket - Deployment

Auch für die Pods des Ticketsystems wurde ein eigenes Deployment benötigt. Die nachfolgende Konfiguration bezieht sich auch auf Tabelle 6 als Grundlage. Das osticket Deployment findet sich in Anhang Q(Zeilen 17-69). Hierbei werden folgende Haupteinstellungen gesetzt:

- Festlegen der Pod Anzahl(=Replicas) auf zwei
- Einbinden des `osticket-plugins-claim` PVC
- Mounten des `Plugin`-PVC nach `/data/upload/include/plugins`
- Einbinden des `osticket-languages-claim` PVC
- Mounten des `Languages`-PVC nach `/data/upload/include/i18n`
- Festlegen des Docker Images auf `campbellsoftwaresolutions/osticket`
- Öffnen des Standard Web Ports 80(http)
- Definition der CPU-Ressourcen für neue Pods auf 200m als Startanforderung bzw. 500m als Maximum für einen Pod
- Übergeben der Umgebungsvariablen `MYSQL_PASSWORD` und `MYSQL_USER` mit Werten aus dem zuvor definierten Secret `osticket-secret`
- Manuelle Übergabe der Umgebungsvariablen `MYSQL_HOST` mit dem Wert `mysql` zur Verwendung des zuvor erstellten Services für die Verbindung mit MySQL

- Angabe der Umgebungsvariable `INSTALL_SECRET` mit einem manuellen Wert, um eine eindeutige InstallationsID für das Ticketsystem zu haben und eine Neuinstallation bei einem Container Neustart zu vermeiden

4.2.10.7 Konfiguration osTicket - Service

Für die Cluster-interne Erreichbarkeit unter einer festgelegten URL wurde nun ein zusätzlicher Service mit dem Namen `osticket` erstellt. Hier wird der auf Port 80 beim Service ankommende Datenverkehr auf den Port 80 des Pods weitergeleitet. Die Konfiguration erfolgte hier analog zu der des MariaDB Services. Die detaillierten Einstellungen des Services finden sich in Anhang Q(Zeilen 1-15). Die generellen Eigenschaften eines Services sind wiederum in Tabelle 7 dargestellt.

4.2.10.8 Konfiguration osTicket - HPA

Um eine hohe Verfügbarkeit bei gleichzeitiger Ressourceneinsparung ermöglichen zu können, war ein sogenannter Autoscaler notwendig. Grundsätzlich existieren in Kubernetes zwei Möglichkeiten zur automatischen Skalierung von Pods:

- horizontal
- vertikal

Für die Skalierung von osTicket wird eine horizontale Skalierung [16] verwendet, da sich die vertikale Auto-Skalierung derzeit noch im Beta Status befindet.
Ein Horizontaler Pod Autoscaler soll anhand zuvor im Deployment definierter Ressourcen zusätzliche Pods, bspw. zum Kompensieren einer Spitzenlast, zur Verfügung stellen. Allerdings sollen nach einer aufgetretenen Spitze die zusätzlichen Pods auch wieder entfernt werden, um die Ressourcen freizugeben [17]. Die Aufschlüsselung einer HPA Konfiguration findet sich in Tabelle 8.

Attribut	Attributwert	Auswirkung
apiVersion:	autoscaling/v1	definiert API Version
kind:	HorizontalPodAutoscaler	spezifiziert Art der Konfiguration
metadata:		legt Metadaten fest
name:	autoscaler	definiert den Namen
namespace:	osticket	legt den Namespace fest
spec:		spezifiziert den Inhalt
maxReplicas:	xxx	maximale Anzahl der Pods/Repliken
minReplicas:	xxx	minimale Anzahl der Pods/Repliken
scaleTargetRef:		Definition des zu skalierenden Endpunktes
apiVersion:	v1	definiert API Version
kind:	Deployment	spezifiziert Art der Konfiguration
name:	xxx	Name des zu skalierenden Deployments
targetCPUUtilizationPercentage:	xxx	gibt an, wie hoch CPU ausgelastet sein soll bevor ein zusätzlicher Pod generiert wird

Tabelle 8: Aufschlüsselung einer HPA .yaml Konfiguration

Für das Deployment von osTicket wurde eine minimale Pod Anzahl von zwei definiert sowie eine maximale Anzahl von zehn. Ein neuer Pod sollte hinzugefügt werden, sobald die CPU Auslastung der bisherigen Pods 70% überschreitet. Nach längerem Unterschreiten der 70% wird vom HPA automatisch die Anzahl der Pods verringert, bis die minimale Anzahl von zwei Pods/Repliken erreicht ist. Dies wird im Zusammenspiel mit dem ReplicaSet des Deployments erreicht. Die exakte Konfiguration des HPA ist in Anhang Q(Zeilen 72-85) zu finden.

4.2.10.9 Konfiguration osTicket - Ingress

Damit die Anwendung auch von extern erreichbar ist, wurde auf dem zuvor eingerichteten Ingress Controller ein Ingress benötigt. Dieser Ingress sorgte dafür, dass die URL `https://ticket.osticket.info` an den Service `osticket` auf Port 80 weitergeleitet wird. In Tabelle 9 ist der generelle Aufbau eines Ingress ersichtlich. Die verwendete Konfiguration des Ingress ist in Anhang R beschrieben.

Attribut	Attributwert	Auswirkung
apiVersion:	v1	definiert API Version
kind:	Ingress	spezifiziert Art der Konfiguration
metadata:		legt Metadaten fest
name:	ingress	definiert den Namen
namespace:	osticket	legt den Namespace fest
spec:		spezifiziert den Inhalt
rules:		Festlegung eines Regelsatzes für eingehende Anfragen auf URL
- host:	xxx	definiert den Host/URL
http:		Umgang mit Aufbau der URL
paths:		Weiterleitung anhand des Pfades in der URL/Webseite
- pathType:	xxx	Weiterleitung anhand des URL pathType (bspw. Prefix)
path:	xxx	Angabe, welcher URL-Pfad an nachfolgenden Service weitergeleitet wird
backend:		Definition des Backends für Ingress
service:	xxx	Angabe des Services im Cluster
name:	xxx	Angabe des Service Namens
port:		Spezifikation des Service Ports
number:	xxx	Spezifikation des Service Ports

Tabelle 9: Aufschlüsselung einer Ingress .yaml Konfiguration

Im Anschluss daran mussten zusätzlich die zuvor bereits installierten Plugins `Authentication :: LDAP and Active Directory` und `Authentication :: HTTP Pass-Through` sowie die Sprachdateien für Deutsch und Englisch in den jeweiligen Shares hinterlegt werden. Der Download erfolgte hier direkt von der Website von osTicket. Die Konfigurationen der Plugins, Spracheinstellungen und genereller firmenspezifischer osTicket Einstellungen sind nicht Bestandteil dieser Dokumentation.

4.3 Zusammenfassung Implementierung

Zum Abschluss der Implementierung lässt sich im Bezug auf die Forschungsfrage "Welche Komponenten werden innerhalb des Kubernetes Clusters konfiguriert, um das Cluster auch für zukünftige Anwendungen vorzubereiten?" festhalten, dass folgende Cluster-Komponenten auch für eine zukünftige Erweiterbarkeit ausgelegt sind:

- Metrics Server
- Ingress Controller

Der Metrics Server kann hierbei für horizontale oder mit entsprechender Erweiterung auch für vertikale Skalierung bei anderen Deployments verwendet werden.

Der Ingress Controller stellt für alle zukünftigen Anwendungen die Möglichkeit eines externen Zugriffs auf die Standardports unter jeweils einer eigenen URL zur Verfügung.

Bei einer Erweiterung der Forschungsfrage auf nicht Kubernetes bezogene Konfigurationen wäre zudem eine Wiederverwendung der folgenden Komponenten möglich:

- Docker Registry: für die Speicherung von privaten Docker Images
- Rancher: zur Verwaltung des Clusters via GUI

Zudem können nun auch die letzten Forschungsfragen beantwortet werden: "Welche Einsparungen ressourcentechnischer Art können nach der Umsetzung erzielt werden?"

Aufgrund des Kubernetes Clusters werden momentan weit mehr Ressourcen verbraucht als der bisherige Server (2vCPUs, 2GB RAM) benötigt hat. Wenn jedoch das Cluster produktiv eingeführt wird, ist davon auszugehen, dass der Verbrauch des Deployments weitaus geringer ist. Allerdings kann hier zu gegebenem Zeitpunkt keine finale Aussage getroffen werden, da die Auslastung des Deployments auch durch Testing nicht mit der Anzahl der echten Anfragen im Produktivsystem verglichen werden kann. Diese Fragestellung kann somit final erst nach einer Überführung in ein Produktivsystem beantwortet werden. Zusätzlich ist hier aber auch zu berücksichtigen, dass hiermit bspw. automatisch die Möglichkeit von Rolling Updates sowie eine Ausfallsicherheit gegeben sind.

Sobald multiple andere Applikationen in das Cluster überführt werden, wird sich auch der Overhead des Clusters amortisieren. Ab diesem Zeitpunkt ist das Cluster dann ressourcenschonender bei einer gleichzeitigen Erhöhung der Funktionalitäten.

Zur Fragestellung "Welche weiteren Vorteile ergeben sich durch die Migration der Software osTicket?" hat sich gezeigt, dass bei vorhergehenden Updates manche Konfigurationen verloren gingen. Da in Zukunft diese extern auf einem PVC abgespeichert werden, sind diese nun auch über Updates hinweg persistent. Zudem kann die Software nun skaliert werden und ist des Weiteren auch noch ausfallsicher. Für die Zukunft könnten hier auch Rolling Updates implementiert werden, wodurch ein absolut störungsfreies Update der Software möglich wird. Zuletzt kann auch die SSL Konfiguration ignoriert werden, da SSL vonseiten des IngressControllers gehandhabt und durch den LoadBalancer weitergeleitet wird.

5 Tests

Die Funktionsweise des Clusters wurde anhand des auszurollenden Ticketsystems immer wieder aufs Neue überprüft. Da das Projekt als Testsystem zu verstehen ist, konnten die Tests immer direkt im Live-System durchgeführt werden.
Zu Beginn war das Deployment von „osTicket“ nicht erfolgreich, da es ein Problem mit dem Routing nach extern gab. Hier war die Problemstellung, dass sich ein von Kubernetes für die Pods intern genutzter Bereich mit einem Bereich des Firmennetzwerks überschnitt. Dies führte dazu, dass die internen IPs nicht nach extern kommunizieren konnten. Aufgrund dieser Tatsache musste der für die Pods intern genutzte IP-Bereich in ein anderes Netzwerksegment verlegt werden.
Zusätzlich schlug zu Beginn die automatische Skalierung des osticket Deployments fehl. Der Fehler lag hier an einer unvollständigen Konfiguration des Deployments, da hier keine minimale bzw. maximale Anzahl an gewünschten Repliken spezifiziert worden war. Die automatische Skalierung konnte durch eine Simulation von vielen gleichzeitigen Webanfragen recht einfach getestet werden und verlief erfolgreich.
Abschließend wurde noch geprüft, wie sich der Absturz eines Pods auf die jeweiligen Deployments auswirken würde. Hier hat sich gezeigt, dass bei einem Ausfall eines Pods im osticket Deployment aufgrund der minimalen Pod Anzahl von zwei keine Auswirkungen festgestellt werden konnten. Sofort nach Absturz eines Pods wurde zudem auch gleich ein weiterer vonseiten des Clusters erstellt. Aufgrund der fehlenden Redundanz beim Deployment der MariaDB war hier das System für ca. 20 Sekunden nicht erreichbar. Das automatisierte Neustarten des Pods funktionierte aber auch hier zuverlässig. Eventuell kann hier in Zukunft noch mithilfe eines Datenbankbrokers eine Redundanz der Datenbank Container erreicht werden.

6 Schluss

Im Laufe dieses Projektes hat sich herausgestellt, dass die Konfiguration eines Kubernetes Clusters sehr komplex ist und der damit einhergehende Aufwand sowie die Wartung sehr zeitintensiv sind. Nachdem diese Hindernisse durch gründliche Einarbeitung in die Thematik überwunden wurden, konnten mithilfe von Kubernetes sowohl viel Zeit als auch Ressourcen eingespart, sowie zusätzliche Funktionen für alle konfigurierten Applikationen umgesetzt werden. Unter der Prämisse, dass das Cluster in Zukunft mit weiteren Applikationen verwendet wird, ist eine hohe Ressourceneinsparung möglich. Zusätzlich gewährt das Cluster noch weitere Vorteile wie Skalierbarkeit, Ausfallsicherheit und eine einfachere Integration von Updates in den laufenden Betrieb, welche ansonsten im Regelfall bei kleineren Anwendungen nicht gegeben sind.

Im Moment befinden sich aufgrund der Initialisierung dieses Projekts bereits neue Anwendungsfälle wie eine Unternehmensschnittstelle in Portierung auf dieses System. Zudem wurde die Funktionalität des Clusters über die Projektarbeit hinaus um ein Monitoring System und eine CI/CD Pipeline für die Vorbereitung des produktiven Einsatzes erweitert. Durch CI/CD wiederum kann auch Entwicklungszeit eingespart, die Softwarequalität erhöht und die Bereitstellungszeit von Software verkürzt werden.

Zusammenfassend lässt sich sagen, dass sich das Projekt "Aufbau eines Kubernetes Clusters für Hardwareressourcen-sparendes Software Deployment im Fassadenbau bei der Firma seele" erfolgreich umsetzen ließ und bereits in der Testphase einen Mehrwert für die Softwareentwicklung in der Firma darstellte.

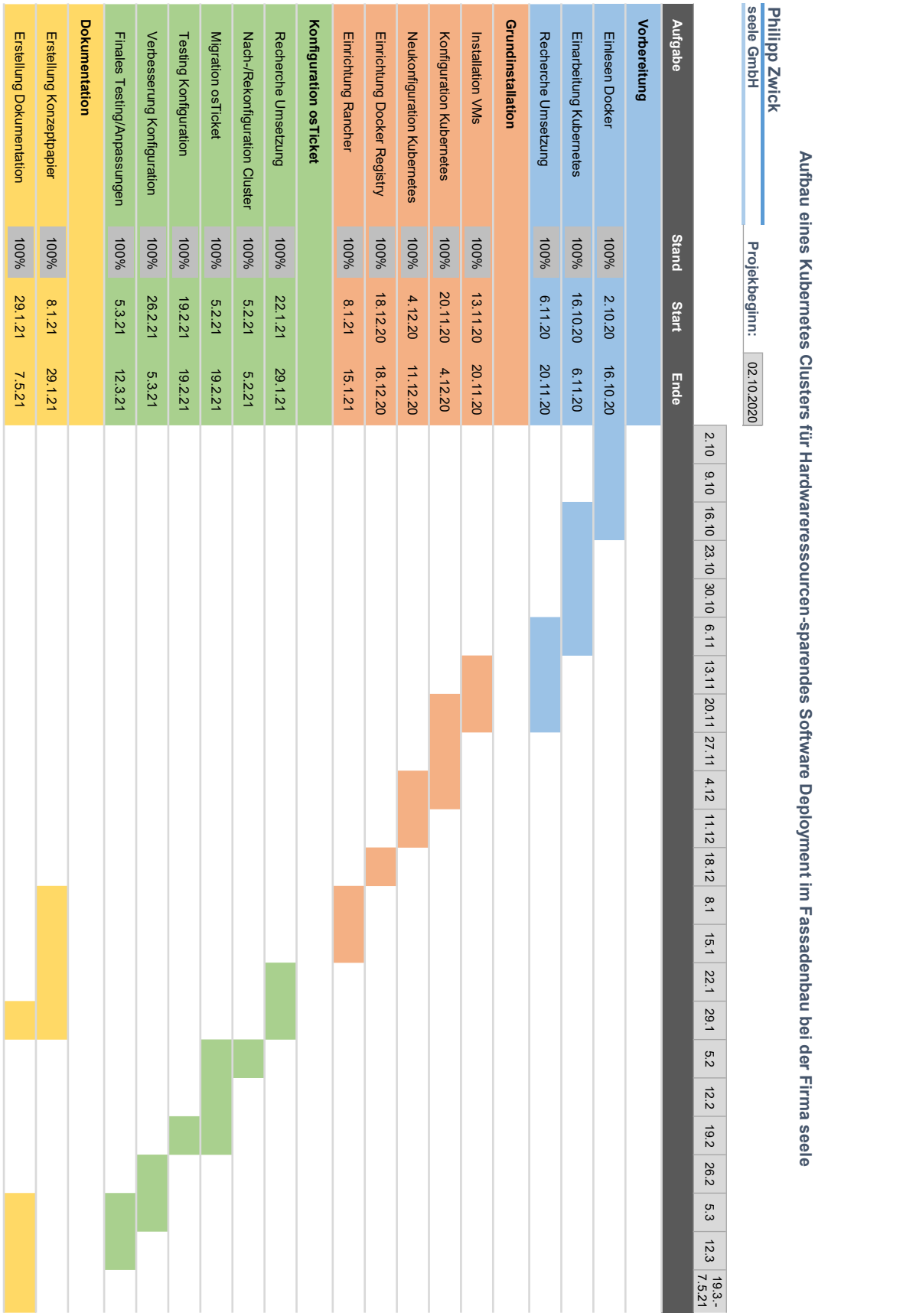
7 Definition

- MySQL= My Structured Query Language
- VM = virtuelle Maschine
- On-Premise (Hardware) = Betrieb des Servers im eigenen Rechenzentrum bzw. vor Ort
- LTS = Long Term Support
- DNS = Domain Name Service
- Daemon = Prozess unter unixartigen Systemen, welcher Dienste zur Verfügung stellt und im Hintergrund läuft
- init-Prozess = Prozess, der als erstes gestartet wird (meist Prozess-ID 1)
- systemd = Daemon, der als init-Prozess zum Beenden, Starten und Überwachen weiterer Prozesse dient (auch Cgroup Manager)
- cgroup = Feature des Linux Kernels, welches das Isolieren/Regulieren von Computer Ressourcen eines Prozesses oder einer Prozessgruppe ermöglicht
- cgroupfs = Cgroup Manager
- Overlay2 = von Docker Inc. speziell für Container entwickeltes Dateisystem
- NFS = Network File System
- yaml = YAML Ain't Another Markup Language (Auszeichnungssprache)
- GPG = GNU Privacy Guard
- CI/CD Pipeline = Continuous Integration / Continuous Delivery Pipeline
- Rancher = Software, die Verwaltung des Clusters über Webinterface ermöglicht
- JSON = JavaScript Object Notation
- swap = Auslagern von Dateien aus dem Arbeitsspeicher auf die lokale Festplatte
- CPU = Central Processing Unit (Prozessor)
- RAM = Random Access Memory (Arbeitsspeicher)
- HTTP(S) = Hypertext Transfer Protocol (Secure)
- GUI = Graphical User Interface
- URL = Uniform Resource Locator
- SSL = Secure Sockets Layer
- API = Application Programming Interface
- LB = LoadBalancer

Anhang

Anhang A:

Zeitplanung für das Projekt



Anhang B:

1containerdInstall.sh

```
#1/bin/bash
#Running prerequisite checks
#Enabling the modules overlay and br_netfilter within the kernel
cat <<EOF | sudo tee /etc/modules-load.d/containerd.conf
overlay
br_netfilter
EOF

#Checking if the modules have been loaded properly within the kernel
#Expecting return code
sudo modprobe overlay
sudo modprobe br_netfilter

# Setup required sysctl params, these persist across reboots.
cat <<EOF | sudo tee /etc/sysctl.d/99-kubernetes-cri.conf
net.bridge.bridge-nf-call-iptables  = 1
net.ipv4.ip_forward                 = 1
net.bridge.bridge-nf-call-ip6tables = 1
EOF

# Apply sysctl params without reboot
sudo sysctl --system

#Updating the system and installing CRI-containerd
sudo apt-get update && sudo apt-get install -y containerd

#Creating required directory and copying default configuration
sudo mkdir -p /etc/containerd
sudo containerd config default | sudo tee /etc/containerd/config.toml
#Manual reconfiguration of config.toml required

#Restarting service
sudo systemctl restart containerd
```

Anhang C:

2disableSwap.sh

```
#!/bin/bash
#Disabling swap, since necessary for kubernetes to run
#Disabling in current running system
sudo swapoff -a
#Disabling swap in order to persist across reboots
sudo sed -i 's/\/swap.img/#swap.img/g' /etc/fstab
```

Anhang D:

3kubernetesSetup.sh

```
#!/bin/bash

#Installing kubernetes (kubectl,kubeadm,kubelet)
sudo apt-get update && sudo apt-get install -y apt-transport-https
    curl
sudo curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg |
    sudo apt-key add -
sudo cat <<EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list
deb https://apt.kubernetes.io/ kubernetes-xenial main
EOF
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```

Anhang E:

4dockerInstall.sh

```
#!/bin/bash

#Installing Docker CE
#Setting up packages for repository to embed
sudo apt-get update && sudo apt-get install -y \
  apt-transport-https ca-certificates curl software-properties-common
    gnupg2

#Adding Dockers official GPG key
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo
   apt-key --keyring /etc/apt/trusted.gpg.d/docker.gpg add -

#Adding the Docker apt repository
sudo add-apt-repository \
  "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) \
  stable"

#Installing Docker CE
sudo apt-get update && sudo apt-get install -y \
  docker-ce=5:19.03.11~3-0~ubuntu-$(lsb_release -cs) \
        docker-ce-cli=5:19.03.11~3-0~ubuntu-$(lsb_release -cs)

#Setting up Docker Daemon
sudo cat <<EOF | sudo tee /etc/docker/daemon.json
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2",
  "insecure-registries":["IP-CONTROL-SERVER:PORT"]
}
EOF

# Create /etc/systemd/system/docker.service.d
sudo mkdir -p /etc/systemd/system/docker.service.d
```

```
37
38 # Restart Docker
39 sudo systemctl daemon-reload
40 sudo systemctl restart docker
41
42 #Enabling service to start on boot
43 sudo systemctl enable docker
```

Anhang F:

6kubeadmInit.yaml

```
1  apiVersion: kubeadm.k8s.io/v1beta2
2  bootstrapTokens:
3  - groups:
4    - system:bootstrappers:kubeadm:default-node-token
5    token: abcdef.xxxxxxxxxxxx
6    ttl: 24h0m0s
7    usages:
8    - signing
9    - authentication
10 kind: InitConfiguration
11 localAPIEndpoint:
12   advertiseAddress: CONTROL-SERVER-IP
13   bindPort: PORT
14 nodeRegistration:
15   criSocket: /run/containerd/containerd.sock
16   name: kubernetesmaster
17   taints:
18   - effect: NoSchedule
19     key: node-role.kubernetes.io/master
20 ---
21 apiServer:
22   timeoutForControlPlane: 4m0s
23 apiVersion: kubeadm.k8s.io/v1beta2
24 certificatesDir: /etc/kubernetes/pki
25 clusterName: kubernetes
```

```
controllerManager: {}
dns:
  type: CoreDNS
etcd:
  local:
    dataDir: /var/lib/etcd
imageRepository: k8s.gcr.io
kind: ClusterConfiguration
kubernetesVersion: v1.20.0
networking:
  dnsDomain: cluster.local
  serviceSubnet: IP/12
scheduler: {}
---
apiVersion: kubelet.config.k8s.io/v1beta1
kind: KubeletConfiguration
cgroupDriver: systemd
```

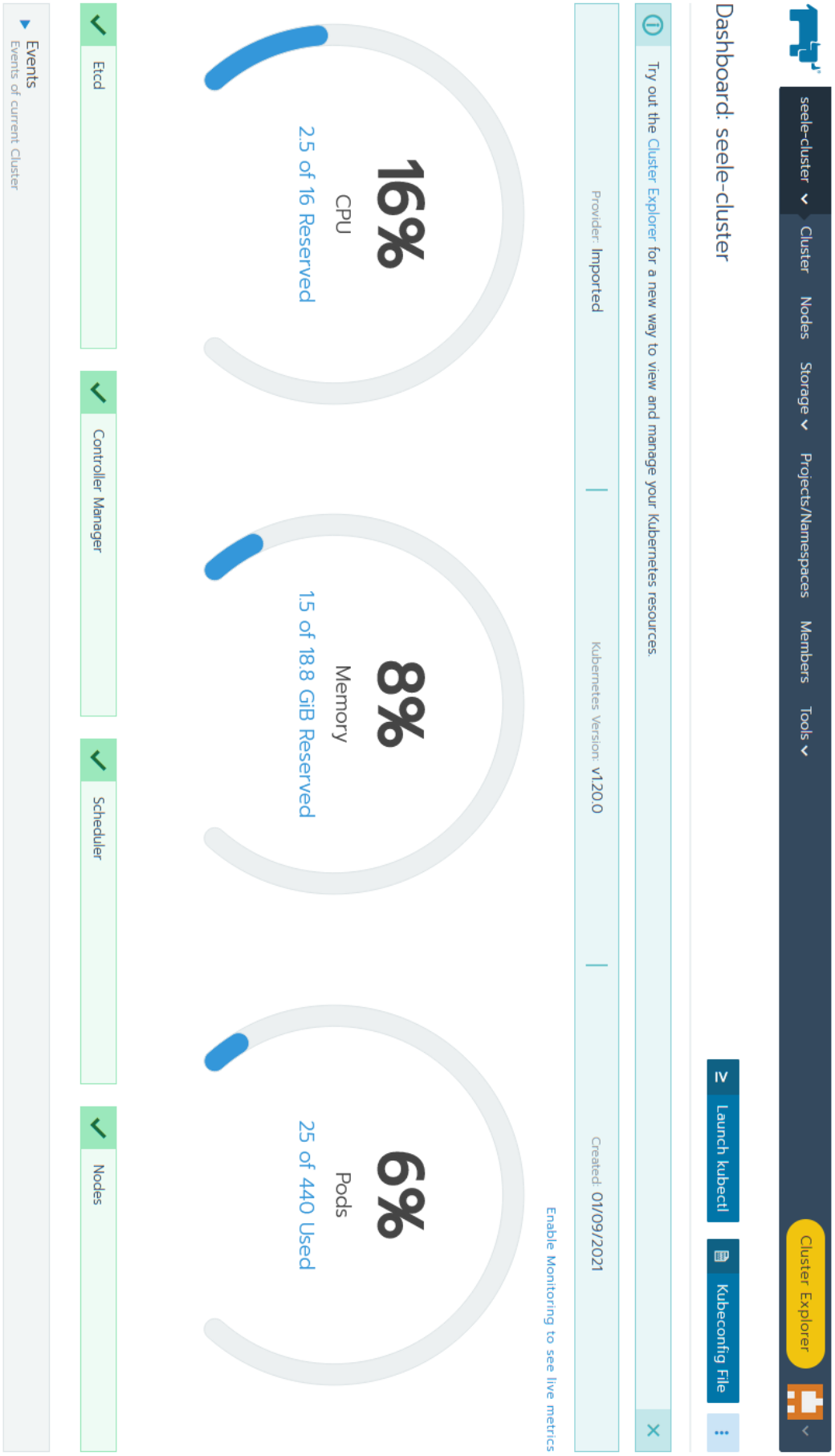
Anhang G:

5kubernetesInstall.sh

```
#!/bin/bash

#Creating Directory for kubectl config within normal user
sudo mkdir -p $HOME/.kube

#Creating Kubernetes Cluster
#Should to be done manually in order to retrieve the key for joining
   nodes to cluster
sudo kubeadm init phase kubelet-start --config=kubeadmInit.yaml
sudo systemctl daemon-reload && sudo systemctl restart kubelet
sudo kubeadm init --config=kubeadmInit.yaml

#Copying Config file to use for kubectl command
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

#Apply Calico networking (under 50 nodes)
kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml

#Adding Nodes to Cluster
kubeadm join CONTROL-SERVER-IP:PORT --token abcdef.0xxxxxxxabcdef \
    --discovery-token-ca-cert-hash sha256:sha256-Hash
```

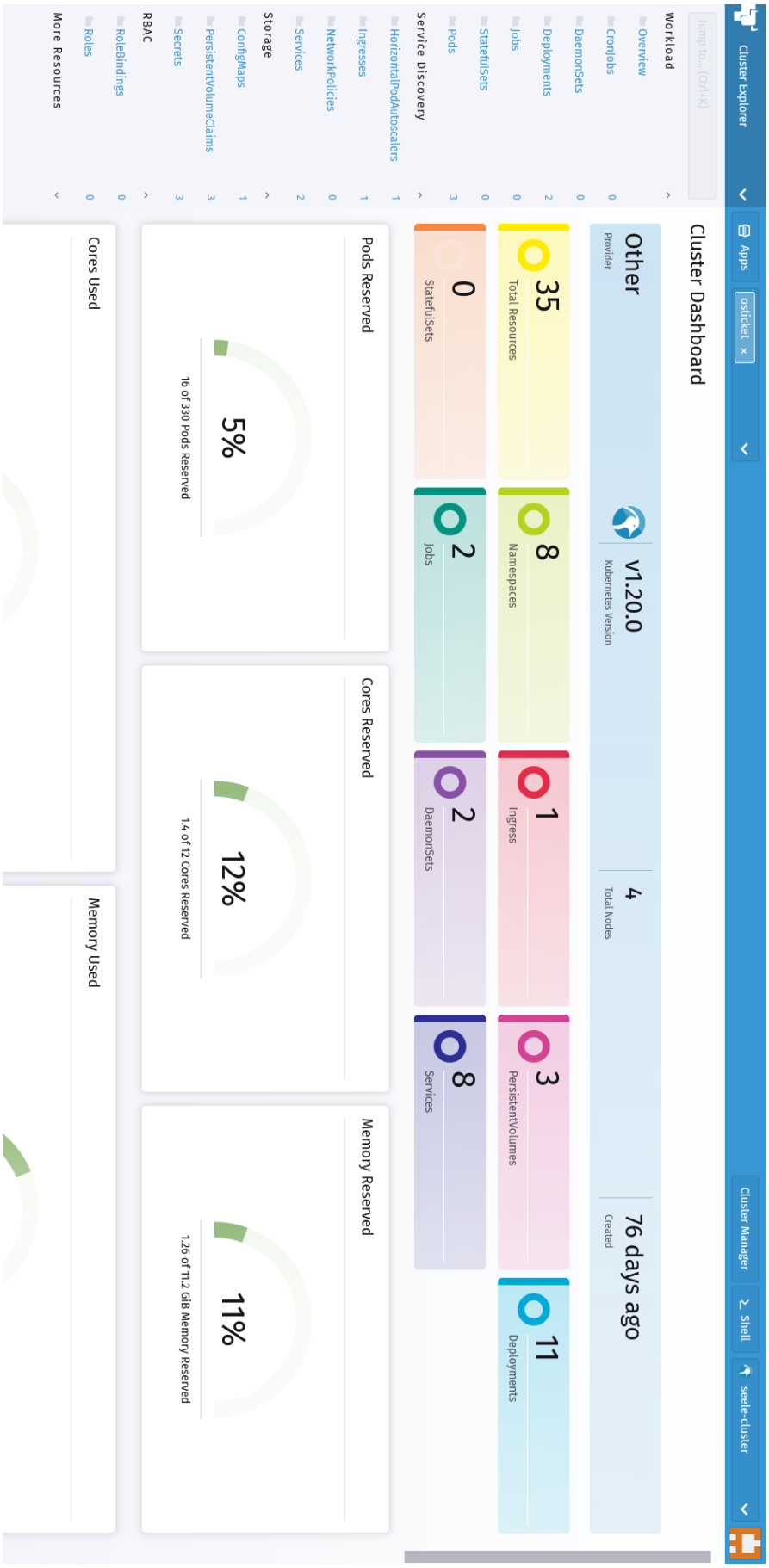
Anhang H:

Rancher Dashboard seele-cluster



Anhang I:

Ansicht des sog. "Cluster Explorers"



Anhang J:

haproxy.conf

```
global
  log /dev/log  local0
  log /dev/log  local1 notice
  chroot /var/lib/haproxy
  stats socket /run/haproxy/admin.sock mode 660 level admin expose-fd
    listeners
  stats timeout 30s
  user haproxy
  group haproxy
  daemon

  # Default SSL material locations
  ca-base /etc/ssl/certs
  crt-base /etc/ssl/private

  # See: https://ssl-config.mozilla.org/#server=haproxy&server-
   version=2.0.3&config=intermediate
        ssl-default-bind-ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-
   RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-
   AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20
   -POLY1305:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384
        ssl-default-bind-ciphersuites TLS_AES_128_GCM_SHA256:
   TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256
        ssl-default-bind-options ssl-min-ver TLSv1.2 no-tls-tickets

defaults
  log global
  mode  http
  option  httplog
  option  dontlognull
        timeout connect 5000
        timeout client  50000
        timeout server  50000
  errorfile 400 /etc/haproxy/errors/400.http
  errorfile 403 /etc/haproxy/errors/403.http
  errorfile 408 /etc/haproxy/errors/408.http
  errorfile 500 /etc/haproxy/errors/500.http
  errorfile 502 /etc/haproxy/errors/502.http
```

```
  errorfile 503 /etc/haproxy/errors/503.http
  errorfile 504 /etc/haproxy/errors/504.http

frontend kubernetes_http
  bind *:80
  mode http
  redirect scheme https if !{ ssl_fc }

frontend kubernetes_https
  bind *:443
  option tcplog
  mode tcp
  default_backend kubernetes

backend kubernetes
  mode tcp
  balance leastconn
  option ssl-hello-chk
  server kubernetesnode1 IP:3xxxx check
  server kubernetesnode2 IP:3xxxx check
  server kubernetesnode3 IP:3xxxx check

listen stats
  #Used for interface to view live statistics of HAProxy
  bind IP-LB:8080
  stats enable
  stats hide-version
  stats refresh 30s
  stats show-node
  stats auth user:password
  stats uri /stats
```

Anhang K:

Ansicht der HAProxy GUI

HAProxy

Statistics Report for pid 923 on haproxy

> General process information

pid = 923 (process #1, nbproc = 1, nbthread = 2)
uptime = 0d 1h15m09s
system limits: memmax = unlimited; ulimit-n = 524287
maxsock = 524287; **maxconn** = 262121; **maxpipes** = 0
current conns = 1; current pipes = 0/0; conn rate = 0/sec; bit rate = 1.631 kbps
Running tasks: 1/16; idle = 100 %

active UP
active UP, going down
active DOWN, going up
active or backup DOWN
active or backup DOWN for maintenance (MAINT)
active or backup SOFT STOPPED for maintenance
backup UP
backup UP, going down
backup DOWN, going up
not checked
Note: "NOLB"/"DRAIN" = UP with load-balancing disabled.

Display option:
- Scope :
- Hide 'DOWN' servers
- Enable refresh
- Refresh now
- CSV export
- JSON export (schema)

External resources:
- Primary site
- Updates (v2.2)
- Online manual

kubernetes_http

	Queue			Session rate			Sessions				Bytes		Denied		Errors			Warnings		Server									
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	0	-	0	0	262 121	0			0	0	0	0	0					OPEN								

kubernetes_https

	Queue			Session rate			Sessions				Bytes		Denied		Errors			Warnings		Server									
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	0	-	0	0	262 121	0			0	0	0	0	0					OPEN								

kubernetes

	Queue			Session rate			Sessions				Bytes		Denied		Errors			Warnings		Server									
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
kubernetesnode1	0	0	-	0	0	0	0	0	-	0	0	?	0	0	0		0	0	0	0	0	10m47s UP	L6OK in 0ms	1	Y	-	1	1	1h4m	-
kubernetesnode2	0	0	-	0	0	0	0	0	-	0	0	?	0	0	0		0	0	0	0	0	10m37s UP	L6OK in 1ms	1	Y	-	1	1	1h4m	-
kubernetesnode3	0	0	-	0	0	0	0	0	-	0	0	?	0	0	0		0	0	0	0	0	10m38s UP	L6OK in 1ms	1	Y	-	1	1	1h4m	-
Backend	0	0		0	0	0	0	0	26 213	0	0	?	0	0	0		0	0	0	0	0	10m47s UP		3	3	0		1	1h4m	

stats

	Queue			Session rate			Sessions				Bytes		Denied		Errors			Warnings		Server									
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				1	2	262 121	4						7 518	425 008	0	0	0					OPEN								
Backend	0	0		0	1	26 213	1	0	0	0s			7 518	425 008	0	0		1	0	0	0	1h15m UP		0	0	0		0		

Anhang L:

exports

```
# /etc/exports: the access control list for filesystems which may be exported
#               to NFS clients.  See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes       hostname1(rw,sync,no_subtree_check) hostname2(ro, sync,no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs4        gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)
#

/mnt/nfs_share      IP1(rw,sync,no_subtree_check,no_root_squash)
/mnt/nfs_share      IP2(rw,sync,no_subtree_check,no_root_squash)
/mnt/nfs_share      IP3(rw,sync,no_subtree_check,no_root_squash)
/mnt/nfs_share      IP4(rw,sync,no_subtree_check,no_root_squash)
```

Anhang M:

01namespace.yaml

```
apiVersion: v1
kind: Namespace
metadata:
 name: osticket
```

Anhang N:

02secrets.yaml

```
#Creating a secret which contains credentials for osticket to access db
apiVersion: v1
kind: Secret
metadata:
 name: osticket-secret
 namespace: osticket
data:
```

```
  MYSQL_PASSWORD: ""
  MYSQL_USER: ""
---
#Creating a secret which contains credentials for mariadb to
    create db
apiVersion: v1
kind: Secret
metadata:
  name: mysql-secret
  namespace: osticket
data:
  MYSQL_ROOT_PASSWORD: ""
  MYSQL_USER: ""
  MYSQL_PASSWORD: ""
```

Anhang O:

03pv-pvc.yaml

```
#Creating a NFS PersistentVOLUME
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mysql-volume
  namespace: osticket
spec:
  capacity:
    storage: 10Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: nfs
  mountOptions:
    - hard
    - nfsvers=4.1
```

```
18   nfs:
19     path: /nfs_share/osticket/mysql
20     server: IP
21 ---
22 #Creating a PersistentVolumeCLAIM
23 apiVersion: v1
24 kind: PersistentVolumeClaim
25 metadata:
26   name: mysql-pv-claim
27   namespace: osticket
28 spec:
29   storageClassName: nfs
30   accessModes:
31     - ReadWriteOnce
32   resources:
33     requests:
34       storage: 10Gi
35 ---
36 #Creating a NFS PersistentVOLUME
37 apiVersion: v1
38 kind: PersistentVolume
39 metadata:
40   name: osticket-plugins-volume
41   namespace: osticket
42 spec:
43   capacity:
44     storage: 1Gi
45   volumeMode: Filesystem
46   accessModes:
47     - ReadWriteMany
48   persistentVolumeReclaimPolicy: Retain
49   storageClassName: nfs
50   mountOptions:
```

```
51    - hard
52    - nfsvers=4.1
53   nfs:
54    path: /nfs_share/osticket/plugins
55    server: IP
56  ---
57  #Creating a NFS PersistentVOLUME
58  apiVersion: v1
59  kind: PersistentVolume
60  metadata:
61   name: osticket-languages-volume
62   namespace: osticket
63  spec:
64   capacity:
65    storage: 1Gi
66   volumeMode: Filesystem
67   accessModes:
68    - ReadWriteMany
69   persistentVolumeReclaimPolicy: Retain
70   storageClassName: nfs
71   mountOptions:
72    - hard
73    - nfsvers=4.1
74   nfs:
75    path: /nfs_share/osticket/languages/
76    server: IP
77  ---
78  #Creating a PersistentVolumeCLAIM
79  apiVersion: v1
80  kind: PersistentVolumeClaim
81  metadata:
82   name: osticket-plugins-claim
83   namespace: osticket
```

```
spec:
 volumeName: osticket-plugins-volume
 storageClassName: nfs
 accessModes:
  - ReadWriteMany
 resources:
  requests:
   storage: 1Gi
---
#Creating a PersistentVolumeCLAIM
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: osticket-languages-claim
 namespace: osticket
spec:
 volumeName: osticket-languages-volume
 storageClassName: nfs
 accessModes:
  - ReadWriteMany
 resources:
  requests:
   storage: 1Gi
```

Anhang P:

04mysql-conf.yaml

```
#Creating a service
apiVersion: v1
kind: Service
metadata:
 name: mysql-svc
 namespace: osticket
spec:
 selector:
```

```
    app: mysql-local
  ports:
    - protocol: "TCP"
      port: 3306
      targetPort: 3306
  type: ClusterIP
---
#Creating a Deployment with MariaDB Image
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
  namespace: osticket
spec:
  selector:
    matchLabels:
      app: mysql-local
  template:
    metadata:
      labels:
        app: mysql-local
    spec:
      volumes:
        - name: mysql-pv-storage
          persistentVolumeClaim:
            claimName: mysql-pv-claim
      containers:
        - name: mysql-pv-container
          image: mariadb
          ports:
            - containerPort: 3306
              name: "mysql-ticket"
          volumeMounts:
```

```
42          - mountPath: "/var/lib/mysql"
43            name: mysql-pv-storage
44        env:
45          - name: MYSQL_ROOT_PASSWORD
46            valueFrom:
47              secretKeyRef:
48                name: mysql-secret
49                key: MYSQL_ROOT_PASSWORD
50          - name: MYSQL_USER
51            valueFrom:
52              secretKeyRef:
53                name: mysql-secret
54                key: MYSQL_USER
55          - name: MYSQL_PASSWORD
56            valueFrom:
57              secretKeyRef:
58                name: mysql-secret
59                key: MYSQL_PASSWORD
60          - name: MYSQL_DATABASE
61            value: "osticket"
```

Anhang Q:

05osticket-conf.yaml

```
1  #Creating a service to expose osticket pod on port 80
2  apiVersion: v1
3  kind: Service
4  metadata:
5   name: osticket
6   namespace: osticket
7  spec:
8   selector:
9    app: osticket
10  ports:
11   - name: web
```

```
12     protocol: "TCP"
13     port: 80
14     targetPort: 80
15   type: ClusterIP
16 ---
17 #Creating a deployment which manages the pods via a
     replicaset
18 apiVersion: apps/v1
19 kind: Deployment
20 metadata:
21   name: osticket
22   namespace: osticket
23 spec:
24   selector:
25     matchLabels:
26       app: osticket
27   replicas: 2
28   template:
29     metadata:
30       labels:
31         app: osticket
32     spec:
33       volumes:
34         - name: osticket-plugins-storage
35           persistentVolumeClaim:
36             claimName: osticket-plugins-claim
37         - name: osticket-languages-storage
38           persistentVolumeClaim:
39             claimName: osticket-languages-claim
40       containers:
41         - name: osticket
42           image: campbellsoftwaresolutions/osticket
43           ports:
```

```
44          - containerPort: 80
45        resources:
46          limits:
47            cpu: 500m
48          requests:
49            cpu: 200m
50        volumeMounts:
51          - mountPath: "/data/upload/include/plugins"
52            name: osticket-plugins-storage
53          - mountPath: "/data/upload/include/i18n"
54            name: osticket-languages-storage
55        env:
56          - name: MYSQL_PASSWORD
57            valueFrom:
58              secretKeyRef:
59                name: osticket-secret
60                key: MYSQL_PASSWORD
61          - name: MYSQL_USER
62            valueFrom:
63              secretKeyRef:
64                name: osticket-secret
65                key: MYSQL_USER
66          - name: MYSQL_HOST
67            value: "mysql"
68          - name: INSTALL_SECRET
69            value: "abcde"
70
71  ---
72  #Creating a horizontal pod autoscaler for automatic scaling
        of the pods
73  apiVersion: autoscaling/v1
74  kind: HorizontalPodAutoscaler
75  metadata:
```

```
  name: osticket
  namespace: osticket
spec:
  maxReplicas: 10
  minReplicas: 2
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: osticket
  targetCPUUtilizationPercentage: 70
```

Anhang R:

06ingress.yaml

```
#Creating an ingress to enable external access which listens on URL
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: nginx-osticket
 namespace: osticket
spec:
 rules:
  - host: ticket.osticket.info
    http:
     paths:
     - pathType: Prefix
       path: "/"
       backend:
        service:
         name: osticket
         port:
          number: 80
```

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[1] Bernd Öggl, Michael Kofler. *Docker - Das Praxisbuch für Entwickler und DevOps-Teams*. Rheinwerk-Computing, 1. Auflage 2018.

[2] Oliver Liebel. *Skalierbare Container-Infrastrukturen - Das Handbuch für Administratoren*. Rheinwerk-Computing, 2. Auflage 2020.

[3] Container Orchestration *https://www.hpe.com/de/de/what-is/container-orchestration.html*, Zugriff am 07.04.2021

[4] Container Runtime Interface *https://kubernetes.io/blog/2016/12/container-runtime-interface-cri-in-kubernetes/*, Zugriff am 07.04.2021

[5] Kubelet *https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/*, Zugriff am 07.04.2021

[6] Ingress *https://kubernetes.io/docs/concepts/services-networking/ingress/*, Zugriff am 07.04.2021

[7] Rancher *https://rancher.com/why-rancher/*, Zugriff am 07.04.2021

[8] Keine Docker-Unterstützung in Kubernetes *https://kubernetes.io/blog/2020/12/02/dont-panic-kubernetes-and-docker/*, Zugriff am 19.01.2021

[9] kubeadm Setup *https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/*, Zugriff am 27.03.2021

[10] Kubernetes Container Runtime Setup *https://kubernetes.io/docs/setup/production-environment/container-runtimes/*, Zugriff am 27.03.2021

[11] Kubernetes Network Plugin Calico
https://docs.projectcalico.org/getting-started/kubernetes/self-managed-onprem/onpremises/, Zugriff am 27.03.2021

[12] Rancher Installation
https://rancher.com/docs/rancher/v2.x/en/installation/other-installation-methods/single-node-docker/, Zugriff am 27.03.2021

[13] Installation Metrics Server
https://github.com/kubernetes-sigs/metrics-server, Zugriff am 14.02.2021

[14] Installation Ingress Controller
https://kubernetes.github.io/ingress-nginx/deploy/, Zugriff am 14.02.2021

[15] Konfiguration Network File System
https://phoenixnap.com/kb/ubuntu-nfs-server, Zugriff am 14.02.2021

[16] Konfiguration osTicket
https://github.com/CampbellSoftwareSolutions/docker-osticket, Zugriff am 14.02.2021

[17] Installation Horizontal Pod Autoscaler
https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/, Zugriff am 14.02.2021

[18] Definitionen
https://www.dev-insider.de, Zugriff am 16.01.2021

[19] Definitionen
https://www.wikipedia.de, Zugriff am 16.01.2021

Weiterführende Informationen

Einführung in Container:

`https://rancher.com/blog/2019/an-introduction-to-containers/`

Einführung in Kubernetes:

`https://www.cncf.io/blog/2020/12/14/kubernetes-101-an-introduction/`