Containerbasiertes PaaS für individuelle VPS Infrastrukturen

Studienarbeit, Studiengang Informatik
OST – Ostschweizer Fachhochschule
Campus Rapperswil-Jona

Herbstsemester 2024

Autoren	Stefan Meyer, Jan Meier
Betreuer	Mirko Stocker, Dominic Klinger

Inhaltsverzeichnis

Abstract

Diese Studienarbeit befasst sich mit der Entwicklung von QuickStack, einer selbst gehosteten Platform-as-a-Service (PaaS)-Lösung, die auf einer Virtual Private Server (VPS)-Infrastruktur betrieben wird und die effiziente Durchführung von Build- und Deployment-Prozessen sowie die Orchestrierung von laufenden Container-Anwendungen ermöglicht. Eine von uns durchgeführte Marktanalyse ergab signifikante Kostenvorteile gegenüber kommerziellen Cloud PaaS Angeboten. Nach einem Vergleich verschiedener Container-Orchestrierungstools wurde die Kubernetes-Distribution k3s aufgrund ihres geringen Ressourcenbedarfs und der einfachen Installation als Basis für QuickStack ausgewählt. Longhorn dient als Shared Storage Provider für persistente Daten über alle VPS-Nodes hinweg. Zu den Kernfunktionen von QuickStack gehören die einfache Installation über einen einzigen Kommandozeilenbefehl, die Unterstützung von Deployments aus einem öffentlichen oder privaten Git-Repository, ein Monitoring-Dashboard zur Überwachung des Ressourcenverbrauchs, Tools zum Debuggen laufender Container, darunter der Live-Log-Stream und das Web-Terminal, sowie die Anbindung von persistentem Speicher an einen oder mehrere Container über alle VPS hinweg. Durch die Bereitstellung dieser Funktionen vereinfacht QuickStack den Aufbau und Betrieb von Anwendungen auf einer VPS-Infrastruktur erheblich. Zukünftige Arbeiten könnten automatisierte Backups, weitere Debugging-Möglichkeiten und Deployment-Templates für Datenbanken und gängige Open-Source-Anwendungen umfassen.

Management Summary

Ausgangslage

Die zunehmende Komplexität von Software-Deployments auf eigenen Infrastrukturen stellt Entwickler vor Herausforderungen. Während Platform-as-a-Service (PaaS)-Lösungen diese Komplexität reduzieren können, sind kommerzielle Cloud-PaaS-Angebote wie Azure App Service, AWS Beanstalk oder Digital Ocean App Platform oft kostenintensiv. Gleichzeitig mangelt es vielen Softwareentwicklern an fundierten Kenntnissen in der Serveradministration, um eine eigene Lösung auf einem Virtual Private Server (VPS) aufzusetzen. Ziel dieser Arbeit ist die Entwicklung einer kostengünstigen und einfach zu bedienenden Self-Hosted PaaS-Lösung für VPS-Infrastrukturen, um Build- und Deployment-Prozesse sowie die Orchestrierung von laufenden Container-Anwendungen zu vereinfachen. Darüber hinaus soll das Tool in einem Cluster aus mehreren VPS betrieben werden können, um auf unterschiedliche Skalierungsanforderungen reagieren zu können.

Vorgehen & Technologien

Eine Marktanalyse bestehender PaaS-Lösungen und VPS-Anbieter wurde durchgeführt, um Kosten und Funktionsumfang zu vergleichen. Anschliessend wurden verschiedene Container-Orchestrierungstools evaluiert. Die Wahl fiel auf die Kubernetes-Distribution k3s aufgrund des geringen Ressourcenbedarfs, der einfachen Installation und der integrierten Komponenten wie Traefik, Ingress und ServiceLB. Longhorn wurde aufgrund der mit Fio durchgeführten Performancetests und der einfacheren Inbetriebnahme als Shared Storage Provider für persistente Daten ausgewählt. QuickStack selbst wurde mit dem Fullstack-Framework Next.js, SQLite als Datenbank und verschiedenen JavaScript-Bibliotheken entwickelt. Die Architektur orientiert sich an Clean Code Prinzipien und verwendet Kubernetes und Longhorn APIs. Kaniko wird als Container Build Tool und Registry als Container Registry verwendet. Wie in der Grafik dargestellt, wird der Quellcode aus dem Git Repository geladen und dem Build Container zur Verfügung gestellt. Nach dem Build wird der Container in den k3s Cluster deployed und kann über den QuickStack Pod orchestriert werden. Die Container der Applikation werden je nach Konfiguration auf die verschiedenen VPS-Nodes des k3s Clusters verteilt.

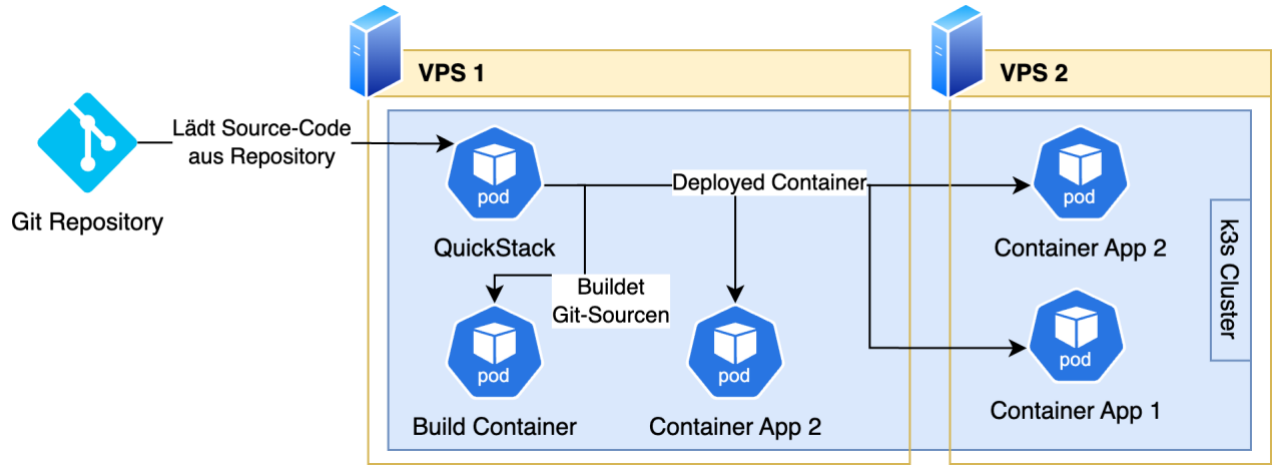


Abbildung 1: Architektur QuickStack vereinfacht

Ergebnisse & Ausblick

QuickStack bietet eine benutzerfreundliche Weboberfläche zur Verwaltung von Anwendungen und deren Deployments auf einem k3s-Cluster. Kernfunktionen sind die einfache Installation per Kommandozeile, Deployments aus Git-Repositories, Ressourcenüberwachung, SSL-Zertifikatsverwaltung, Live-Log-Streaming, Web-Terminal-Zugriff, Zwei-Faktor-Authentifizierung und die flexible Anbindung von persistenten Volumes. Damit vereinfacht QuickStack das Applikationsmanagement auf VPS-Infrastrukturen erheblich und bietet eine kostengünstige Alternative zu kommerziellen PaaS-Lösungen. Usability-Tests zeigten eine hohe Benutzerfreundlichkeit. Zukünftig könnte QuickStack um automatisierte Backups, weitere Container-

Debugging-Tools und Deployment-Templates für Datenbanken und gängige Open-Source-Anwendungen erweitert werden.

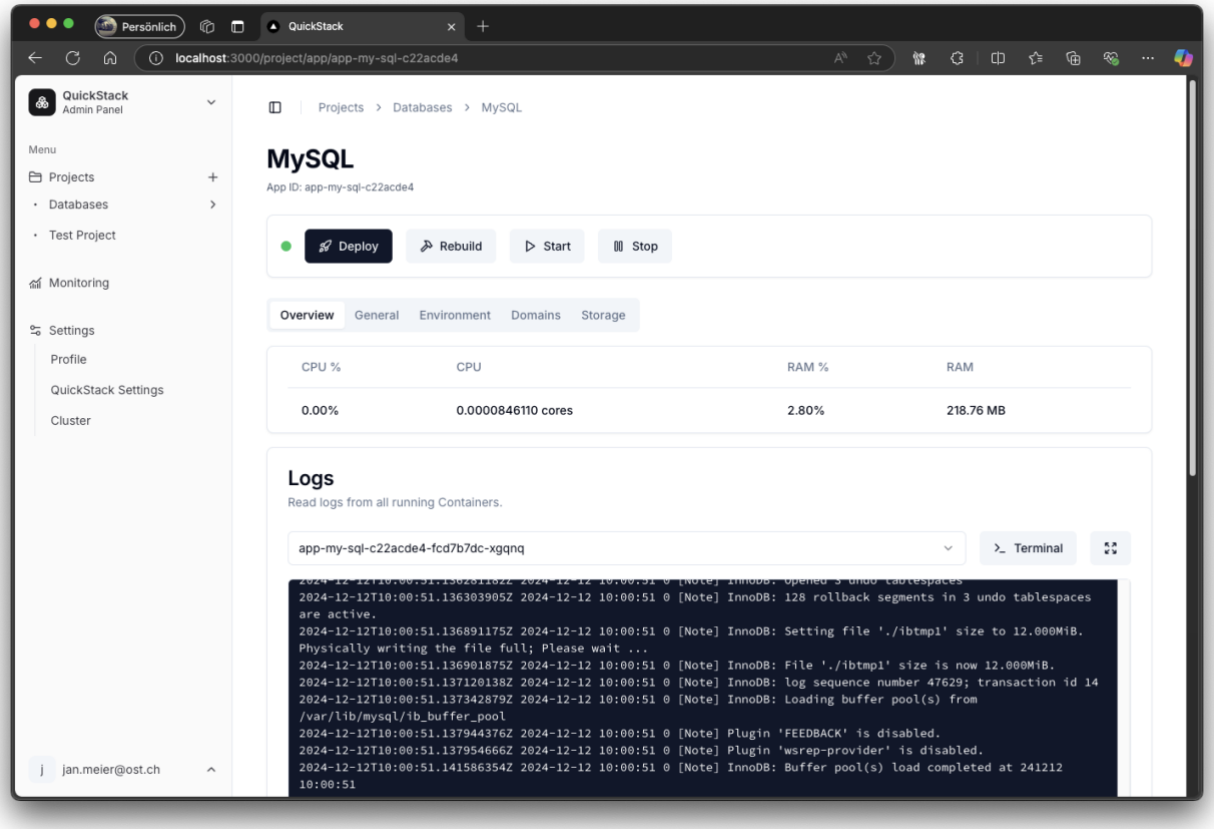


Abbildung 2: Screenshot QuickStack App Übersicht

1. Einleitung

1.1 Ausgangslage

Die steigende Komplexität von Software-Deployments auf eigenen Infrastrukturen, stellt Entwickler vor Herausforderungen. Während Platform-as-a-Service (PaaS) Lösungen diese Komplexität reduzieren können, sind kommerzielle Cloud-PaaS-Angebote oft kostenintensiv. Gleichzeitig mangelt es vielen Entwicklern an fundierten Kenntnissen in der Serveradministration, um eigene Lösungen aufzusetzen.

1.2 Ziel der Arbeit

Ziel dieser Arbeit war daher die Entwicklung einer einfach zu bedienenden Self-Hosted-PaaS-Lösung für VPS-Infrastrukturen, um Build- und Deployment-Prozesse sowie die Orchestrierung von Containeranwendungen zu vereinfachen.

Weitere Ziele sind:

- Die Software soll Docker-Container anhand einer Git-Repository-URL und eines Dockerfiles über eine Weboberfläche builden und starten können.
- Es soll möglich, sein Container über mehrere VPS hinweg zu orchestrieren (Cluster).
- Es soll möglich, sein pro containerbasierte Anwendung Serverressourcen (CPU, RAM, Speicher) zuzuweisen.
- Mit einem Log-Monitoring sollen Fehler und Abstürze identifiziert werden.

1.3 Anforderungen

An die Software bestehen die folgenden Anforderungen:

- Der Quellcode der Arbeit soll am Ende unter einer OpenSource Lizenz veröffentlicht werden.
- Die Software soll auf einem VPS innerhalb von 5 Minuten aufgesetzt werden können und der Benutzer muss maximal 5 Kommandozeilenbefehle ausführen, bis die Orchestrierungssoftware einsatzbereit ist.
- Die Software soll (wo möglich) bereits bestehende OpenSource Tools verwenden, um die Container zu Builden und zu Betreiben (verbreitete Tools und Standards verwenden, das Rad nicht neu erfinden)

2. Analyse, konkrete Anforderungen

Beim Hosting eines Services spielen viele Faktoren eine Rolle. Einige davon sind: Preis, Aufwand, benötigtes «Know-How» fürs Setup, Serverstandort, Verfügbarkeit und der Support. Da die meisten Softwareentwickler nur über begrenzte bis mittlere Kenntnisse im Bereich Server-Hosting und -Betrieb verfügen, entscheiden sich viele von ihnen für eine Platform-as-a-Service (PaaS) Lösung, um ihren Sourcecode einfach und effizient bereitzustellen. Bei der Wahl eines PaaS wird der gesamte Betrieb, die Wartung, die Sicherheit und das Monitoring von einem Dienstleister verwaltet. Der Softwareentwickler kann sich so fokussiert um die Entwicklung der Software kümmern.

Alternativ zu den PaaS Dienstleitern gibt es PaaS Software, welche auf einem eigenen Server installiert wird. Einstellungen für das Hosten von Services und Erstellen von Pipelines wird wie bei den PaaS Dienstleistern über ein UI gemacht und kann ohne grosses «Know-How» über Linux Services, Deployment Pipelines oder Netzwerkkonfigurationen erfolgen. Es ist ein PaaS, welches auf einem selbst betriebenen Server läuft (Self-Hosted PaaS). Ziel dieser SA ist ein solches Self-Hosted-PaaS zu erstellen, das einen ähnlichen Umfang an Funktionalitäten wie ein PaaS Dienstleister bietet.

Nachfolgend werden einige PaaS-Anbieter und Anbieter von Virtual Private Server (VPS) verglichen. Dieser Vergleich soll die Preisdifferenzen von PaaS Dienstleistern und dem Betrieb eines eigenen VPS mit Self-Hosted-PaaS aufzeigen.

2.1 Marktanalyse PaaS

Für das deployen von Software gibt es diverse PaaS, welche grosse Cloudanbieter, kleinere Firmen oder Startups anbieten. Nachfolgend eine Auswahl einiger dieser Services:

Service	Node-Unabhängiger	Git-Repo-Verknü	Docker Image	SSL-Zertifika	Kosten	Weitere Features
Azure App Service[1]	Ja	Ja	Ja	Ja	Abhängig von der Nutzung	Automatische Skalierung, Integrierte CI/CD, umfangreiche Azure-Integrationen
DigitalOcean App Platform[2]	Ja	Ja	Ja	Ja	Kostenlos für 3 statische Seiten, sonst kostenpflichtig	Schnelles Deployment, gute Integration mit anderen DO-Diensten, einfache Handhabung
Railway[3]	Ja	Ja	Ja	Ja	Kostenlose Testphase, Abhängig von Nutzung und gewähltem Abo	Einfacher Einstieg, schneller Deployment-Prozess, gute Dokumentation
AWS Elastic Beanstalk[4]	Ja	Ja	Ja	Ja	Abhängig von der Nutzung	Tiefgreifende Integration mit AWS-Diensten, automatisches Scaling, umfangreiche Konfigurationsmöglichkeiten
Heroku[5]	Ja	Ja	Ja	Ja	Kostenloser Plan, kostenpflichtig für mehr Ressourcen	Einfaches Deployment, umfangreiche Add-Ons, automatische Skalierung
Google App Engine[6]	Ja	Ja	Ja	Ja	Abhängig von der Nutzung	Automatische Skalierung, Integrierte GCP-Services, Unterstützung für mehrere Sprachen und Frameworks, Handhabung mit Config-Dateien in Repo eher mühsam und schlecht dokumentiert.
Render[7]	Ja	Ja	Ja	Ja	Kostenloser Plan, kostenpflichtig für mehr Ressourcen	Vollständig gemanagte Infrastruktur, einfache Handhabung, kontinuierliches Deployment

Tabelle 1: Marktanalyse PaaS

[1] https://azure.microsoft.com/de-de/products/app-service
[2] https://docs.digitalocean.com/products/app-platform/
[3] https://railway.app/
[4] https://aws.amazon.com/de/elasticbeanstalk/
[5] https://www.heroku.com/
[6] https://cloud.google.com/appengine
[7] https://render.com/

2.1.1.1 Kostenvergleich anhand von praktischem Beispiel

Um einen fairen Kostenvergleich zwischen den Angeboten von PaaS Services machen zu können, werden die Kosten pro Service für den Betrieb eines einfachen Backend Linux-Containers und einer PostgresSQL-Datenbank berechnet. Die Berechnungen werden alle für den Betrieb in einer Region in der EU gemacht. Falls bei einem Cloudprovider die Datenbankgrösse angegeben werden kann, wird von mindestens 10GB Speicher ausgegangen. Für alle Cloudprovider werden «Free-Trials» und Angebote für den nicht kommerziellen Gebrauch ignoriert. Ebenso werden Kosten für Traffic ignoriert.

Service	Kosten Backend	Kosten DB	Gesamtkosten
Azure App Service[8] [9]	13.14 $	13.77 $	26.91 $
DigitalOcean App Platform[10]	5 $	15 $	20 $
Railway[11]	-	-	5 $ Mit dem «hobby» Abo werden dem Konto monatlich Ressourcen im Wert von 5 $ gutgeschrieben. Dies reicht für den Betrieb einer einfachen Applikation inkl. DB. Für den Betrieb in der EU muss mindestens das «Pro» Abo für 20 $ monatlich abgeschlossen werden.
AWS Elastic Beanstalk[12]	3.5 $	64.59 $	68.09 $
Heroku[13]	Basic 7 $	Essential 0 5 $	12 $
Google App Engine[14]	Instance B1 27.38 $	Db-f1-micro 11.29 $	38.66 $
Render[15]	Starter 7 $	Standard 20 $	27 $ (Ausgehend von einem Hobby Abo)

Tabelle 2: Kostenvergleich PaaS anhand eines praktischen Beispiels

[8] https://azure.com/e/77e644854b304beb885334cdbe5b8521

[9] https://www.digitalocean.com/pricing/managed-databases

[10] https://www.digitalocean.com/pricing/app-platform

[11] https://railway.app/pricing

[12] https://calculator.aws/#/estimate?id=5b9568379559bca46f680b8e40b91a265400aceb

[13] https://www.heroku.com/pricing

[14] https://cloud.google.com/products/calculator/estimate-preview/CiQyOTMwMTQ3NS0zMzI0LTRkNjktYWFkMC00ZjVhMjJiODhhZTkQAQ==?hl=de

[15] https://render.com/pricing

Aus dem Preisvergleich geht heraus, dass der Betrieb eines einzigen Services zwischen 5 $ und 70 $ pro Monat kosten kann. Bei einem weiteren Service würde sich der Preis entsprechend verdoppeln. Einzige Ausnahme ist hier Railway, wo lediglich die gebrauchten CPU, RAM und Speicher Ressourcen abgerechnet werden, unabhängig davon wie viele Services betrieben werden.

2.2 Marktanalyse IaaS (VPS)

Eine alternative zu einem PaaS Dienstleister sind Virtual Private Server (VPS)[16], wo Applikationen direkt auf einem eigenen Server mit root Zugriff ausgeführt werden können. Nachfolgend eine Auflistung von Angeboten zu Linux VPS mit einer Mindestkapazität von 2 Cores, 4 GB RAM, 30 GB Speicher (Mindestanforderungen für Docker[17] oder Kubernetes[18]) und einer Public-IPv4-Adresse. Limitierungen und zusätzliche Kosten von Netzwerktraffic werden bei dieser Analyse ignoriert. Als Serverstandort wird die EU gewählt.

Anbieter	Monatlicher Preis	CPU-Cores	RAM	Speicher	Bemerkungen
Azure VM[19]	138.78 $	4	7 GB	32 GB	Die nächstkleinere VM hat 2 Cores und 3.5 GB RAM für 56.29 $ Mit Spot-Instances können Preise reduziert werden
AWS VM[20]	30.89 $	2	4 GB	30 GB	Instance t4g.medium Mit Spot-Instances können Preise reduziert werden
Google Cloud Platform VM[21]	75.15 $	2	4 GB	30 GB	Series N4 Mit Spot-Instances können Preise reduziert werden
Hetzner[22]	5.05 $	2	4 GB	40 GB	ARM Server in derselben Preiskategorie verfügbar
Linode[23]	25.92 $	2	4 GB	80 GB	
1&1 Ionos[24]	6 $	2	4 GB	160 GB	VPS Linux M
Contabo[25]	5.04 $	4	6 GB	100 GB	Eine zusätzliche Einrichtungsgebühr wird erhoben
Infomaniak[26]	9.50 CHF	2	4 GB	60 GB	Schweizer Firma mit Rechenzenter in der Schweiz
Hosttech[27]	19.90 CHF	4	4 GB	100 GB	Schweizer Firma mit CO2-Neutralen-Rechenzentren in der Schweiz

Tabelle 3: Marktanalyse IaaS (VPS)

[16] https://cloud.google.com/learn/what-is-a-virtual-private-server
[17] https://docs.docker.com/desktop/install/linux/
[18] https://kubernetes.io/docs/setup/
[19] https://azure.com/e/b5ad0cb36e3c4273b150c109b052c68d
[20] https://calculator.aws/#/estimate?id=d28d60ff558f74184832db4be365c1267395ef33
[21] https://cloud.google.com/products/calculator
[22] https://www.hetzner.com/cloud/
[23] https://www.linode.com/de/pricing/#compute-shared
[24] https://www.ionos.com/servers/vps
[25] https://contabo.com/de/pricing/
[26] https://www.infomaniak.com/de/hosting/vps-lite
[27] https://www.hosttech.ch/vserver/

Der Vergleich zeigt, dass die grossen Hyperscaler allesamt um einiges teurer sind als kleinere Anbieter. Im Vergleich zum Hosting mit einem PaaS wird ein monatlicher Fixpreis angeboten unabhängig davon ob die Ressourcen genutzt werden oder nicht. Zudem können auf VPS-Servern mehrere Services installiert und gleichzeitig betrieben werden, was im Vergleich zu den PaaS Angeboten um einiges günstiger ist.

2.2.1 Konkurrenzanalyse

Aus der Marktanalyse geht heraus, dass der Betrieb eines eigenen VPS um kostengünstiger ist. Damit die Services so einfach wie möglich auf einem VPS betrieben werden können, gibt es einige Self-Hosted-PaaS welche direkt auf einem VPS installiert werden können und einen ähnlichen Umfang an Funktionalitäten wie ein PaaS Dienstleister bieten.

In der nachfolgenden Tabelle werden bereits vorhandene PaaS Tools aufgelistet und anhand der Features verglichen, welche in der SA umgesetzt werden sollen.

Tool	Clusterbetrieb möglich	Node-Unabhängiger Storage	Git-Repo-Verknüpfung	Docker Image Build	SSL-Zertifikate	Kosten	Open Source	Weitere Funktionen
Coolify[28]	Ja	Muss manuell konfiguriert werden mit externem Storage.	Ja	Ja	Ja	Wenn auf eigenen Servern gehosted, kostenlos	Ja	Datenbanken können «out-of-the-box» deployed werden, umfangreiche Dokumentation. Möglichkeit besteht PaaS auf Coolify Infrastruktur zu hosten.
CapRover[29]	Ja	Nein	Ja	Ja	Ja	Kostenlos	Ja	Einrichtung nicht intuitiv. Internes Container Registry, falls Cluster aktiviert ist.
Dokploy[30]	Ja (eingeschränkt, Multi-Node	Nein	Ja	Ja	Ja	Kostenlos	Ja	Datenbanken können «out-of-the-box» deployed werden, umfangreiche Dokumentation, Minimalistisches Design, einfach zu bedienen, teilweise

[28] https://coolify.io/
[29] https://caprover.com/
[30] https://dokploy.com/

	Setup experimentell)							Fehleranfällig, da noch neu auf dem Markt. Container Registry, falls Cluster aktiviert ist.
Easypanel[31]	Nein	Nein	Ja	Ja	Ja	Kostenlos (mit begrenzten Features), kostenpflichtige Pläne verfügbar	Nein	Datenbanken können «out-of-the-box» deployed werden, umfangreiche Dokumentation, Verwaltung über ansprechendes UI, einfaches Setup. Templates für 1-Click-Deployment, Daten von DB-Tabellen können über integriertes Web-UI angesehen/modifiziert werden.

Tabelle 4: Konkurrenzanalyse mit vorhandenen PaaS Tools

[31] https://easypanel.io/

2.3 Ergebnisse der Markt- und Konkurrenzanalyse

Die Analyse zeigt, dass der Betrieb von Self-Hosted-PaaS-Lösungen wesentlich günstiger im Vergleich zu den Angeboten von PaaS-Dienstleistern ist. Zu den wesentlichen Key Features der Self-Hosted-PaaS-Lösungen gehören das Builden von Docker-Images, das Ausführen von Docker Images, die Möglichkeit zur SSL-Zertifikatsverwaltung (anhand von zugewiesener Domain) und die Git-Repository-Integration, die bei allen verglichenen Tools vorhanden ist.

Coolify, CapRover und Dokploy bieten darüber hinaus teilweise Unterstützung für den Clusterbetrieb, was bei wachsender Anwendungskomplexität nützlich ist. Zudem ist die einfache Bereitstellung von Datenbanken (ohne grosse Konfiguration von Storage-Volumes, DB-Docker-Image) ein weiteres nützliches Feature. Easypanel bietet zudem Templates von bekannten Tools wie Nextcloud, Uptimekuma, Mattermost und weiteren Open Source Tools, welche mit einem One-Click-Deployment aufgesetzt werden können.

Ein grosser Punkt, bei welchem alle bestehenden Self-Hostes-PaaS-Lösungen nicht mit den PaaS-Dienstleistern mithalten können, ist die Storageverwaltung im Clusterbetrieb. Entweder können Dienste mit einer Storage-Abhängigkeit (z.B. Datenbanken) nur immer auf demselben Node betrieben werden oder es ist ein manuelles Setup eines Netzwerkstorage notwendig (z.B. GlusterFS, NFS).

2.4 Evaluierung des Container Orchestration Tools und eines Shared Storage Provider

Im nachfolgenden Abschnitt werden Container Orchestration Tools wie Docker oder Kubernetes verglichen und anhand von diesen Resultaten ein geeignetes Orchestrierungstool und Shared Storage Provider für geteilten persistierten Storage über mehrere Nodes ermittelt.

2.4.1 Nutzwertanalyse Container Orchestration Tools

Nachfolgend werden nur Container Orchestration Tools berücksichtigt, welche OpenSource sind und auf einem Linux Server betrieben werden können (Punktzahl: 6 → Super, 1 → Schlecht).

Als Quellen wurden die offiziellen Dokumentationen (Native Kubernetes[32], Docker Swarm[33], K3s[34], MicroK8s[3536], RKE2[37]) und Blogbeiträgen verwendet (Vergleich MicroK8s und K3s[3839], Vergleich Kubernetes und Docker Swarm[404142]).

Kriterium	Gewichtung in %	Kubernetes		Docker Swarm		K3s		MicroK8s		RKE 2	
		Punkte	Nutzwertpunkte	Punkte	Nutzwertpunkte	Punkte	Nutzwertpunkte	Punkte	Nutzwertpunkte	Punkte	Nutzwertpunkte
Einfachheit	30	1	30	4	120	5	150	5	150	5	150
Skalierbarkeit (Cluster)	10	6	60	5	50	5	50	5	50	5	50
Security Orchestrierungssoftware	10	5	50	4	40	5	50	5	50	5	50
Ressource Usage	5	2	10	5	25	5	25	4	20	4	20
Aufwand Updates Orchestrierungssoftware	10	2	20	6	60	4	40	6	60	4	40
Ressourcenmanagement Container	5	6	30	4	20	5	25	6	30	6	30
Fehlerüberwachung und Log-Monitoring Container	5	6	30	3	15	5	25	5	25	5	25
Unterstützung OpenSource Tools	15	6	90	5	75	5	75	5	75	5	75
Support / Community	10	5	50	5	50	5	50	5	50	5	50
Total	**100**	**39**	**370**	**41**	**455**	**44**	**490**	**46**	**510**	**44**	**490**

Tabelle 5: Nutzwertanalyse Container Orchestration Tool

Aus der Nutzwertanalyse geht hervor, dass die drei Kubernetes Distributionen K3s, Microk8s und RKE 2 am geeignetsten für die Anforderungen der SA sind. Im Weiteren werden K3s und MicroK8s evaluiert. Falls sich herausstellt, dass bei K3s einschneidende Limitierung hat, wird RKE 2 zur Evaluierung hinzugezogen. Im nachfolgenden Abschnitt werden die evaluierten Kubernetes Distributionen miteinander sowie mit shared Storage Providern verglichen und evaluiert.

[32] https://kubernetes.io/
[33] https://docs.docker.com/engine/swarm/
[34] https://k3s.io/
[35] https://microk8s.io/
[36] https://microk8s.io/compare
[37] https://docs.rke2.io/
[38] https://www.wallarm.com/cloud-native-products-101/k3s-vs-microk8s-lightweight-kubernetes-distributions
[39] https://www.civo.com/blog/k8s-vs-k3s
[40] https://circleci.com/blog/docker-swarm-vs-kubernetes/
[41] https://lumigo.io/kubernetes-monitoring/kubernetes-vs-docker-swarm-pros-cons-and-6-key-differences/
[42] https://betterstack.com/community/guides/scaling-docker/docker-swarm-kubernetes/

2.4.2 Kubernetes Distributionen

2.4.2.1 K3s

K3s ist eine optimierte Kubernetes Distribution, welche sich durch einen geringen Ressourcenverbrauch und Support für Edge und IoT auszeichnet. Einige Komponenten eines Kubernetes-Clusters werden «out-of-the-box» mitgeliefert, wie beispielsweise Traefik als Ingress Controller[43], Flannel (CNI)[44] und SeviceLB[45] als Loadbalancer.

Damit eine K3s Umgebung betrieben werden kann, müssen gemäss Dokumentation folgende Anforderungen [46] an einen Server erfüllt sein:

Spec	Minimum	Recommended
CPU	1 core	2 cores
RAM	512 MB	1 GB

Tabelle 6: Serveranforderungen k3s ohne High-Availability Cluster

Bei Nodes, welche in einem High-Availability Cluster betrieben werden sollen, sehen die Mindestvoraussetzungen[47] folgendermassen aus:

Deployment Size	Nodes	VCPUS	RAM
Small	Up to 10	2	4 GB
Medium	Up to 100	4	8 GB
Large	Up to 250	8	16 GB
X-Large	Up to 500	16	32 GB
XX-Large	500+	32	64 GB

Tabelle 7: Serveranforderungen k3s im High-Availability Cluster

2.4.2.2 MicroK8s

MicroK8s[48] ist eine auf einem Plugin System basierte vollständige Kubernetes Distribution, welche mit wenigen Commands hochgefahren werden kann. Einzelne Komponenten wie DNS[49], Cert-Manager[50], Kubernetes-Dashoard oder MetalLB[51] können via einem Plugin-Command einfach hinzugefügt werden. MicroK8s bietet eine vereinfachte Installation und Anbindung eines CEPH-Clusters. Sicherheitsupdates werden automatisch installiert. Die Kubernetes Distribution ist zudem so aufgebaut, dass sie durch die Installation mit Snap[52] das zugrundeliegende System nur minimal verändert und auch ein Entfernen von MicroK8s einfach ist.

[43] https://docs.k3s.io/networking/networking-services#traefik-ingress-controller
[44] https://github.com/flannel-io/flannel
[45] https://docs.k3s.io/networking/networking-services#how-servicelb-works
[46] https://docs.k3s.io/installation/requirements?os=debian
[47] https://docs.k3s.io/installation/requirements?os=debian
[48] https://microk8s.io/
[49] https://microk8s.io/docs/addon-dns
[50] https://microk8s.io/docs/addon-cert-manager
[51] https://microk8s.io/docs/addon-metallb
[52] https://ubuntu.com/core/services/guide/snaps-intro

Damit eine MicroK8s Umgebung betrieben werden kann, müssen gemäss Dokumentation folgende Anforderungen[53] an einen Server erfüllt sein:

Spec	Minimum	Recommended
CPU	n/a	n/a
RAM	540 MB	4 GB

Tabelle 8: Serveranforderungen MicroK8s

Zu den Mindestanforderungen für CPU wurde in der offiziellen Dokumentation nichts gefunden.

2.4.2.3 Vergleich K3s und MicroK8s

Die Zielgruppe von K3s liegt primär im IoT und Edge Bereich, während MicroK8s sich eher auf Entwicklungs- und Testumgebungen ausrichtet. Die Installation beider Kubernetes Distributionen ist in nur wenigen Commands erfolgt. K3s ist in der Regel ressourcenschonender, da es keine überflüssigen Komponenten beinhaltet, welche für den Grundbetrieb des Clusters nicht nötig sind. Während MicroK8s mehr oder weniger an das Host-System Ubuntu gebunden ist, kann K3s auf mehreren Linux-Distributionen installiert werden. Beide Cluster unterstützen High-Availability.

2.4.3 Shared Storage Provider

Im nachfolgenden Abschnitt werden zwei Storage Provider evaluiert: Longhorn[54] und CEPH[55]. Diese wurden gewählt, da beide bekannte Shared Storage Lösungen sind, der Quellcode Open Source ist und in Vorlesungen wie «Cloud Infrastructure» vorgestellt wurden. Weitere Storage Provider werden aus zeitlichen Gründen nicht vertieft angeschaut.

2.4.3.1 Longhorn

Longhorn[56] ist ein leichtgewichtiges verteiltes Blockspeichersystem für Kubernetes. Es setzt auf Container und Microservices, um verteilten Blockspeicher bereitzustellen. Für jedes Blockspeicher-Volume erstellt Longhorn einen dedizierten Speichercontroller und repliziert das Volume auf mehrere Replikate, die auf unterschiedlichen Nodes gespeichert werden. Der Speichercontroller und die Replikate werden von Kubernetes orchestriert. Es ist möglich wiederkehrende Snapshots und Backups zu erstellen, Backups auf

[53] https://microk8s.io/docs/getting-started
[54] https://longhorn.io/
[55] https://ceph.io/en/
[56] https://longhorn.io/docs/1.7.1/

NFS und S3 kompatiblen Storage hochzuladen und automatisierte Updates auszuführen. Zudem kann der Speicher über ein Web-UI verwaltet werden.

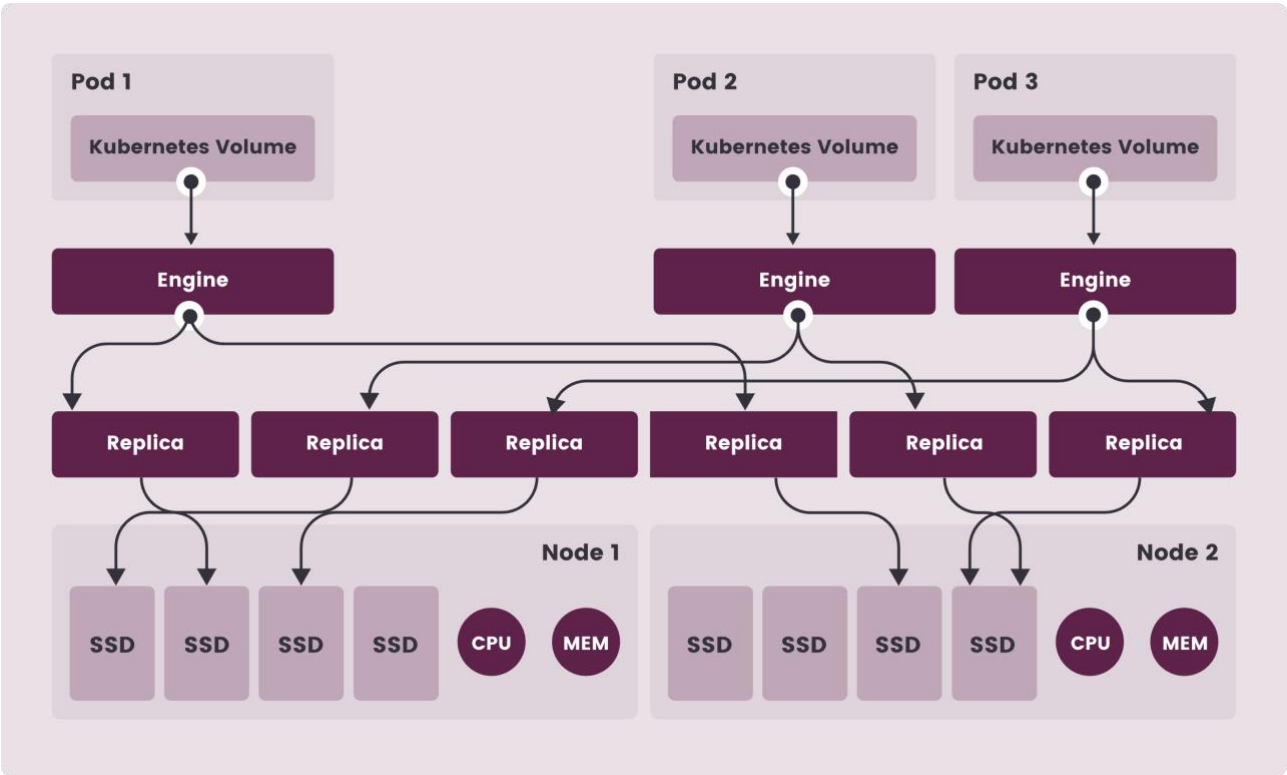


Tabelle 9: Übersicht Architektur Longhorn[57]

Die Mindestvoraussetzungen von Longhorn[58] sind folgende:

- 3 Nodes
- 4 vCPUs per node
- 4 GiB per node
- SSD/NVMe or similar performance block device on the node for storage (recommended)
- HDD/Spinning Disk or similar performance block device on the node for storage (verified)
    - 500/250 max IOPS per volume (1 MiB I/O)
    - 500/250 max throughput per volume (MiB/s)

Longhorn kann auch auf einem oder zwei Nodes installiert werden. Es müssen lediglich die Anzahl Replikas für ein Persistent Volume Claim angepasst werden. Ebenso ist auch der Betrieb mit nur 2 vCPU pro Node möglich [59].

2.4.3.2 CEPH (ROOK)

ROOK CEPH[60] ist eine Storage-Lösung für Kubernetes, die auf CEPH[61] basiert. Es ermöglicht die Bereitstellung von verteiltem Block-, File- und Objectstorage innerhalb eines Kubernetes-Clusters. ROOK

[57] Source: https://longhorn.io/
[58] https://longhorn.io/docs/1.7.1/best-practices/
[59] https://medium.com/@abdelrhmanahmed131/longhorn-distributed-block-storage-for-k8s-2ea11df400d1
[60] https://rook.io/docs/rook/latest-release/Getting-Started/intro/
[61] https://ceph.io/en/

nutzt Kubernetes-Operatoren, um das CEPH-Cluster anzusprechen und so zu verwalten und automatisch anzupassen. ROOK CEPH unterstützt die Erstellung von Snapshots, dynamischen Volumes und Replikationen, während die Verwaltung über Kubernetes-Ressourcen erfolgt. Für die Installation von ROOK CEPH muss separat ein CEPH-Cluster aufgebaut werden.

2.4.3.3 Vergleich Longhorn vs. ROOK CEPH

Grundsätzlich bietet ROOK CEPH eine breitere Palette von Funktionen an, während sich Longhorn im wesentlichen auf Shared-Storage in Kubernetes beschränkt. ROOK CEPH kann auch ausserhalb von Kubernetes verwendet werden. Während ROOK CEPH eine externe Abhängigkeit zu einem zusätzlich manuell aufgesetzten CEPH -Cluster hat, kann Longhorn direkt und ohne externe Installationen in einem Kubernetes Cluster erstellt werden. Longhorn bietet nur einen Blockstorage an, während CEPH zusätzlich einen Object- und Filestorage bietet. Die Einrichtung von Longhorn ist einfacher und intuitiver im Vergleich zu ROOK CEPH. Zu ROOK CEPH wurden auf der Website keine Mindestanforderungen zu CPU und RAM aufgeführt.

2.4.4 Setup Serverinfrastruktur für Testszenarien

Für die Tests von Kubernetes Distributionen und Shared-Storage-Providern wurde für jedes Szenario die folgende identische Testumgebung aufgesetzt. Jede Testumgebung besteht aus zwei VMs in der Azure Cloud mit den Spezifikationen:

- 2 Cores (size “Standard B2als v2”)
- 4 GB RAM
- 30 GB Standard HDD LRS (Max IOPS 500, Max throughput 60)
- Image: 0001-com-ubuntu-server-jammy | 22_04-lts-gen2
- Je eine public IP
- VMs über virtuelles Netzwerk miteinander verbunden

Die beiden VMs bilden ein Kubernetes Cluster mit je einem Master und einem Worker Node. Nachfolgend eine Visualisierung der provisionierten Komponenten in der Azure Cloud pro Testumgebung:

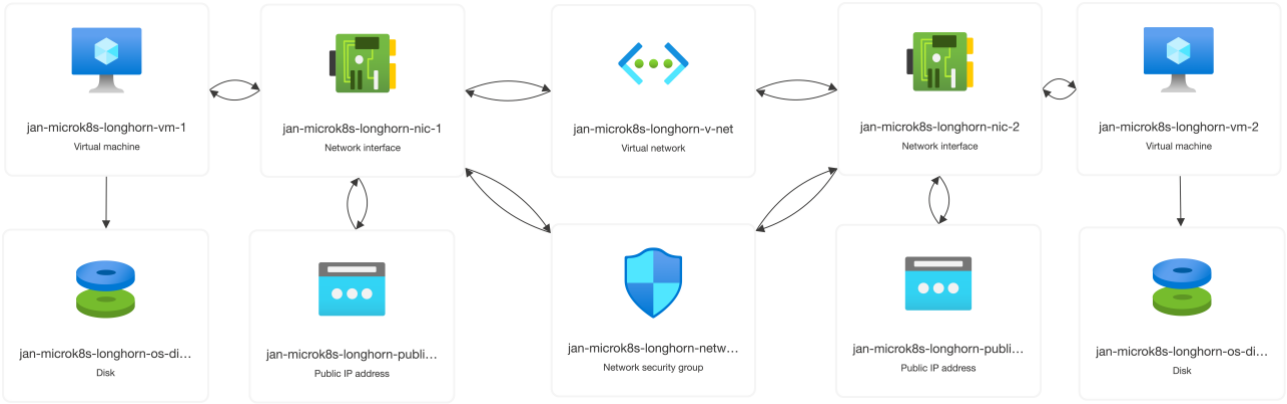


Abbildung 3: Visualisierung Azure Testumgebung pro Testszenario

Insgesamt werden 4 solche Testumgebungen aufgesetzt, um die Kombinationen von Storage-Provider und Kubernetes Distribution wie folgt zu testen:

- microK8s mit CEPH
- microK8s mit Longhorn
- k3s mit CEPH
- k3s mit Longhorn

2.5 Auswertung Kubernetes Distribution & Storage Provider

Für die finale Evaluierung einer Kubernetes Distribution und eines Storage Providers werden zwei Aspekte gemessen. Einerseits der Verbrauch von CPU, RAM und Storage Ressourcen sowie die Schreib- und Leseperformance des Storage Providers.

Um die Ressourcennutzung und Performance-Tests der Kubernetes Distributionen und Storage Provider zu messen, wird jede Testumgebung gemäss der «Abbildung 3: Visualisierung Azure Testumgebung pro Testszenario» aufgesetzt und folgende Konfigurationen vorgenommen:

- Installation der Kubernetes Distribution
- Installation des Storage Provider
- Deployen einer Wordpress Seite mit 2 Backend Pods und 1 DB Pod.
    - Storage von Backend Pods werden über ReadWriteMany PVC persistiert
    - Storage von DB Pod wird über ReadWriteOnce PVC persistiert

2.5.1.1 Ausschliessen von CEPH

CEPH wird aus folgenden Gründen bei der finalen Evaluierung nicht miteinbezogen:

- **microK8s mit CEPH:** Es ist ein einfacher «Step-by-Step»-Guide von Canonical[62] vorhanden, jedoch reichen für den Betrieb eines MicroK8s Cluster mit CEPH die Serverressourcen von 2 Cores und 4 GB RAM nicht aus. Da die anderen Kubernetes Distributionen und Storage Provider deutlich weniger Ressourcen für den Betrieb benötigen und die initial vorgegebenen 2 Cores und 4 GB RAM einhalten können, wird microK8s mit CEPH verworfen.
- **K3s mit CEPH:** CEPH bietet zwar eine umfangreiche Sammlung an Funktionalitäten, jedoch wird für die Verwendung in einem Kubernetes Cluster nur ein Teil davon verwendet. Für die Einrichtung des Storage Providers müssen Konfigurationen ausserhalb von Kubernetes manuell gemacht werden. Zudem ist die Einrichtung wesentlich komplexer als diejenige von Longhorn, wodurch K3s mit CEPH nicht weiter als Variante verfolgt wird.

2.5.1.2 Vergleich der verbrauchten Serverressourcen

Die Messwerte von CPU, RAM und Storage werden mithilfe des Tools «sar»[63] über einen Zeitraum von 5 Minuten gemessen und der Durchschnitt als Wert genommen. Der Wert für die Messung wird erst nachdem der Server mindestens eine Stunde ohne Änderungen an Infrastruktur oder Kubernetes Deployments läuft, ausgelesen.

- Befehl für Messung des CPU-Verbrauchs: `sar -u 1 300`
- Befehl für Messung des RAM-Verbrauchs: `sar -r 1 300`
- Befehl für Messung des freien Speichers: `df -h`

Kubernetes Distribution	Storage Provider	Master Node			Worker Node		
		CPU	RAM	Storage	CPU	RAM	Storage
MicroK8s	Longhorn	8.42%	1.83 GB von 4 GB	8.4 GB von 30 GB	5.97%	2.19 GB von 4 GB	8.8 GB von 30 GB
K3s	Longhorn	5.89%	1.96 GB von 4 GB	6.9 GB von 30 GB	4.01%	2.53 GB von 4 GB	8.1 GB von 30 GB

Tabelle 10: Verbrauchte Serverressourcen pro Kubernetes Distribution und Storage Provider

[62] https://microk8s.io/docs/how-to-ceph

[63] https://www.geeksforgeeks.org/sar-command-linux-monitor-system-performance/

2.5.1.3 Vergleich Performance der Storage Provider

Um die Performancemessungen mit den zwei Storageprovidern Longhorn und Ceph durchführen zu können, wird das Tool FIO[64] verwendet. Mit diesem Tool ist es möglich die Leistung der Storageprovider bezüglich verschiedener Kennzahlen miteinander zu vergleichen. Hier werden folgende Kennzahlen[65] ermittelt:

- IOPS (Input/Output Operations Per Second)
- Throughput
- Latency
- CPU-Auslastung

Um einen möglichst realistischen Wert zu erhalten, wird dieser Test jeweils in einem Pod auf den Testclustern ausgeführt. Neben dem Pod läuft eine deployte Wordpress Seite mit zwei Backend Pods und einem DB Pod. Die aufgezeichneten Werte können hier zwischen den Kubernetes Distributionen gut miteinander verglichen werden, da die Netzwerkumgebung und Konfiguration der VMs identisch ist und vom selben Hosting Provider (Azure). Die Zahlen können jedoch nicht als generell gültigen Wert interpretiert werden, da der ermittelte Wert auf vielen weiteren Faktoren basiert wie die zugrundeliegende Infrastruktur, die vom Hosting Provider bereits vorinstallierte Software oder die Auslastung des physischen Hosts.

Um die verschiedenen Kennzahlen auszulesen, wurden nachfolgende Abfragen mit Fio ausgeführt. Für jede Kennzahl gibt es jeweils einen «Schreiben» und einen «Lesen» Befehl.

IOPS Test

```
fio --rw=randwrite --name=IOPS-write --bs=4k --direct=1 --filename=/mnt/longhorn-volume/iops-write-test --size=1GB --numjobs=4 --ioengine=libaio --iodepth=32 --refill_buffers --group_reporting --runtime=60 --time_based
```

```
fio --rw=randread --name=IOPS-read --bs=4k --direct=1 --filename=/mnt/longhorn-volume/iops-read-test --size=1GB --numjobs=4 --ioengine=libaio --iodepth=32 --refill_buffers --group_reporting --runtime=60 --time_based
```

Throughput Test

```
fio --rw=write --name=IOPS-write --bs=1024k --direct=1 --filename=/mnt/longhorn-volume/throughput-write-test --size=1GB --numjobs=4 --ioengine=libaio --iodepth=32 --refill_buffers --group_reporting --runtime=60 --time_based
```

```
fio --rw=read --name=IOPS-read --bs=1024k --direct=1 --filename=/mnt/longhorn-volume/throughput-read-test --size=1GB --numjobs=4 --ioengine=libaio --iodepth=32 --refill_buffers --group_reporting --runtime=60 --time_based
```

Latency Test

```
fio --rw=randwrite --name=IOPS-write --bs=4k --direct=1 --filename=/mnt/longhorn-volume/latency-write-test --size=1GB --numjobs=1 --ioengine=libaio --iodepth=1 --refill_buffers --group_reporting --runtime=60 --time_based
```

[64] https://fio.readthedocs.io/en/latest/index.html

[65] https://www.thomas-krenn.com/de/wiki/Fio_Grundlagen#Wie_gro%C3%9F_sollen_die_Test-Dateien_werden

```
fio --rw=randread --name=IOPS-read --bs=4k --direct=1 --filename=/mnt/longhorn-volume/latency-read-test --size=1GB --numjobs=1 --ioengine=libaio --iodepth=1 --refill_buffers --group_reporting --runtime=60 --time_based
```

Die Resultate der Tests werden in der untenstehenden Tabelle zusammengefasst.

Anmerkung zur CPU-Auslastung: Die Daten für die CPU-Auslastung wurden vom Output der «Throughput Test»-Kategorie entnommen. Hierbei wurden die CPU-Werte von «usr» (Anteil User Prozesse) und «sys» (Anteil Kernel Prozesse) zusammengezählt. Die weisse Zeile bildet wie Write-Performance ab, die brombeer hinterlegte Zeile bildet die Read-Performance ab.

Kubernetes Distribution	**Storage Provider**	**Read Write Once**				**Read Write Many**			
		IOPS	Throughput (MiB/s)	Latency (msec)	CPU-Auslastung (%)	IOPS	Throughput (MiB/s)	Latency (msec)	CPU-Auslastung (%)
Microk8s	Longhorn	287	73	3.26	2.30	271	42	7.21	0.23
		5634	243	0.67	0.62	5746	471	0.65	2.05
K3s	Longhorn	432	87	2.53	0.40	350	55	0.01	0.35
		5899	232	0.60	0.31	9559	506	0.19	1.62

Tabelle 11: Auswertung FIO Performance Tests

2.5.2 Entscheid Kubernetes Distribution und Storage Provider

Nach diversen Tests, Analysen und Auswertungen wird die Arbeit mit K3s und Longhorn weitergeführt. Der Storage Provider Longhorn wird dem Provider CEPH vorgezogen, da dieses Produkt am besten die Anforderungen der SA erfüllt. CEPH bietet zwar eine grosse Palette von Funktionen an, welche aber betreffend SA nicht gebraucht werden und sich negativ auf die Einfachheit und die Wartbarkeit des Storage auswirken. Zudem reichen, im Gegensatz zu Longhorn, die vorgegebenen Ressourcen von 2 CPU-Cores und 4 GB RAM im Zusammenspiel mit Microk8s nicht aus.

Bei der Wahl der Kubernetes Distribution zeigte sich K3s über alle Tests hinweg als geeigneter im Vergleich zu Microk8s. Dies zum einen durch den kleineren Ressourcenverbrauch bei CPU und Storage, zum anderen auch wegen der besseren Performance im Zusammenspiel mit Longhorn. Bei den Read Write Once Tests befinden sich Microk8s und K3s auf einem ähnlichen Niveau, wobei K3s grösstenteils immer ein Stück besser abschneidet. Bei den Read Write Many Tests zeichnet sich das gleiche Bild ab. Hier ist hervorzuheben, dass die IOPS bei K3s deutlich höher und die Latenzzeiten beträchtlich kleiner ausfallen. Als weiteren positiven Aspekt ist zu erwähnen, dass K3s wie auch Longhorn von Rancher stammen und das Longhorn in der Dokumentation als geeigneter Storage Provider aufgeführt ist[66]. Die Kompatibilität dieser zwei Produkte sollte daher kein Problem sein.

2.6 Evaluierung einer Self-Hosted Container Registry

Da ein Teil der SA Software das Builden von Docker Containern beinhaltet, müssen diese nach dem Buildprozess zwischengespeichert werden, bevor die Build-Artefakte (Docker Container) im Kubernetes Cluster Deployed werden können. Dieser Zwischenspeicher ist eine «Container Registry»[67]. Als mögliche Tools für eine Self-Hosted Container Registry kommen folgende Open Source Tools in Frage: registry[68] [69], harbor[70] und Quay[71]. Wichtig bei dieser Evaluation ist, eine möglichst schlanke Lösung als Self Hosted

[66] https://docs.k3s.io/storage
[67] https://www.redhat.com/en/topics/cloud-native-apps/what-is-a-container-registry
[68] https://itnext.io/self-hosting-a-container-registry-85ce16c0b7b2
[69] https://hub.docker.com/_/registry
[70] https://goharbor.io/
[71] https://www.projectquay.io/

Container Registry zu haben, damit nicht unnötig viele Serverressourcen für den Betrieb der Registry verschwendet werden. Grundsätzlich soll das Container Registry Docker Images nur temporär Speichern und nach erfolgreichem Deployment können die gebuildeten Container wieder gelöscht werden.

Die Wahl fällt hier auf registry, da es eine offizielle Open-Source Container-Registry von Docker ist, welche grundlegende Funktionen zum Speichern und Verwalten von Docker-Images bietet, ohne dabei viel Ressourcen zu verbrauchen.

2.7 Evaluierung eines Container Build Tool

Damit ein Docker Image überhaupt erstellt werden kann, wird ein Build Tool (wie z.B. Docker) benötigt, welches aus einem Dockerfile einen Container erstellt. Im Kubernetes Umfeld bieten sich dazu Tools wie kaniko[72] oder buildah[73] an. Auch hier wird das Augenmerk für die Evaluierung auf eine schlanke und einfach zu bedienende Lösung gelegt.

Das Rennen machte das Tool «kaniko», welches von Google entwickelt wurde und somit auch eine entsprechend grosse Community besitzt. Kaniko braucht keine root Rechte, um seine Funktion als Container Image Build Tool auszuführen, dies ist ein sehr wichtiges Merkmal, wenn man Images innerhalb von Kubernetes (K3s) builden möchte. Kaniko wird über eine einfache CLI bedient und besitzt mit den Start-Parametern «--context» und «--destination» die für die SA benötigten Funktionalitäten. Mit «--context» hat man die Möglichkeit eine URL mitsamt eines Personal Access Tokens mitzugeben, um so Zugriff auf den Sourcecode eines private Repositories in GitHub oder GitLab zu bekommen. Mit «--destination» kann die URL der Container Registry angeben werden, in welcher das fertige Image abgelegt werden soll. Diese Parameter der Kaniko CLI können beim Starten des Containers über die Pod-Definition mitgegeben werden.

[72] https://github.com/GoogleContainerTools/kaniko

[73] https://buildah.io/

2.8 Evaluierung eines Tools für Ingress Controller

Im QuickStack k3s Cluster werden verschiedene Applikationen deployed, die alle von ausserhalb des Clusters erreicht werden müssen und je nach Fall auch ein SSL-Zertifikat besitzen müssen. Somit muss sich das um Reverse Proxy Konfigurationen, Routing, Loadbalancing und die Verwaltung von SSL-Zertifikaten gekümmert werden.

Folgende Kandidaten unterstützen diese Anforderungen[74]:

- NGINX[75]
- Traefik[76]
- HAProxy[77]
- Caddy[78]

Der Entscheid fiel auf Traefik, da dieser Ingress Controller schon standardmässig bei der Installation von K3s mitkommt und somit perfekt integriert ist. Zudem ist er einfacher einzurichten und schlanker zu betreiben, da er nicht so eine Fülle von Funktionen besitzt (die nicht benötigt werden), wie seine Mitbewerber. Da das Endprodukt der SA den Ansatz verfolgt schnell und einfach Applikationen im K3s Cluster zu deployen, ist die dynamische Erkennung von neuen Servicekonfigurationen in Traefik ein grosser Vorteil. Traefik passt seine Konfiguration automatisch an und es müssen keine manuellen Änderungen vorgenommen werden, um eine Applikation erreichbar zu machen. Damit Traefik über alle Nodes von aussen erreichbar ist, verwendet K3s ServiceLB[79]. ServiceLB konfiguriert die IP-Tables der Nodes und leitet standardmässig eingehenden Traffic von den Node-Ports 443 und 80 an den Traefik-Ingress-Controller weiter, welcher dann die Anfrage an den entsprechenden Kubernetes-Service sowie den entsprechenden Pod weiterleitet. Für die DNS-Auflösung innerhalb des Kubernetes Clusters wird CoreDNS verwendet.

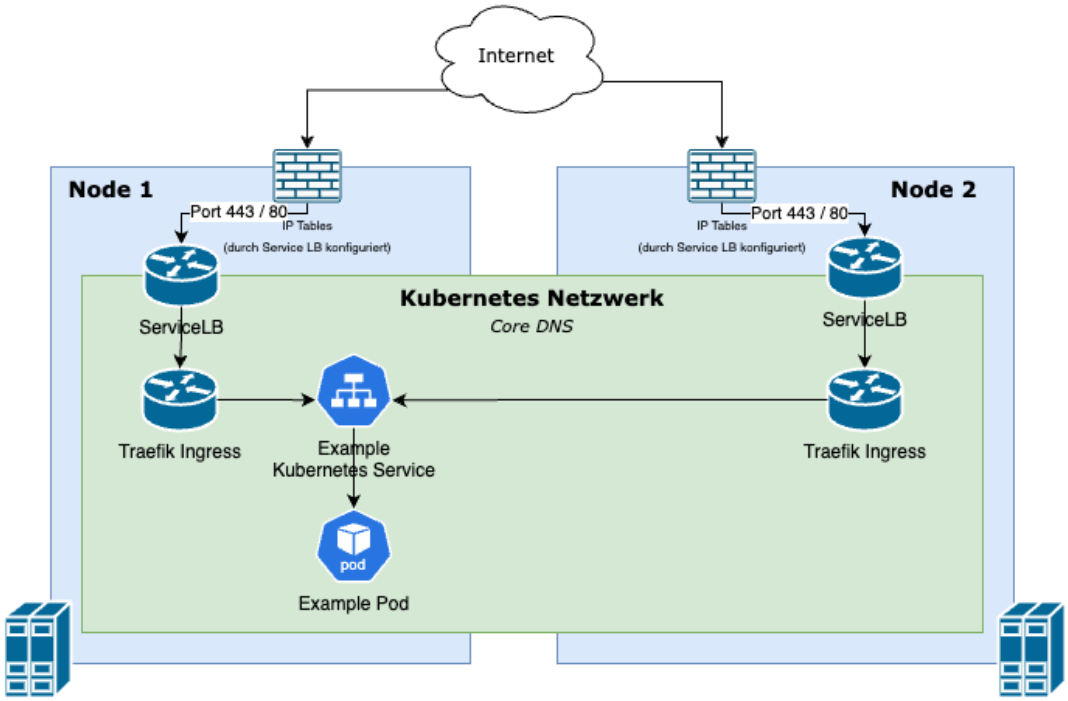


Abbildung 4: Netzwerkarchitektur k3s

[74] https://www.loft.sh/blog/nginx-vs-traefik-vs-haproxy-comparing-kubernetes-ingress-controllers
[75] https://nginx.org/
[76] https://traefik.io/
[77] https://www.haproxy.org/
[78] https://caddyserver.com/
[79] https://docs.k3s.io/networking/networking-services#how-servicelb-works

2.9 Evaluierung eines «Tech-Stack»

Die wichtigste Voraussetzung für den Tech-Stack ist, dass er einfach mit dem Kubernetes Cluster interagieren kann. Idealerweise geschieht das über eine API. Auf der offiziellen Kubernetes Seite finden sich diverse Kubernetes API SDKs [80]. Die SDK sind für folgende Programmiersprachen verfügbar und werden offiziell durch Kubernetes unterstützt: C[81], dotnet[82], Go[83], Haskell[84], Java[85], JavaScript (TypeScript)[86], Perl[87], Python[88] und Ruby[89]. Weitere SDKs werden durch die Community zur Verfügung gestellt, die werden für diese Arbeit jedoch aus Gründen der Wartbarkeit und Aktualität zur neusten Kubernetes Version ausgeschlossen.

Eine weitere Voraussetzung für den Tech-Stack ist ein einfacher, benutzerfreundlicher Zugang zur Applikation, idealerweise über das Web. Mit einer Webapplikation ist sichergestellt, dass die Benutzeroberfläche auf jedem Endgerät ohne Installation von zusätzlicher Software läuft.

Folgende Tech-Stacks werden daher in Betracht gezogen. Zu jedem Tech Stack werden Vor- und Nachteile aufgelistet:

- **TypeScript Frontend + Backend mit Next.js[90]**
  - Vorteile:
    - Gesamte Codelogik in einem Projekt und derselben Sprache
    - Viel Dokumentation und Beispiele sowie Support durch Community
    - Zahlreiche UI Libraries mit fertigen React Komponenten
    - Grosse Auswahl an npm Packages für diverse Use-Cases (Authentifizierung, UI, Utils-Libraries)
  - Nachteile:
    - Bei komplexem oder ressourcenintensivem Code ist man an die Limitierungen von JavaScript gebunden (z.B. Single Thread)

- **dotnet Frontend + Backend mit Blazor[91]**
  - Vorteile:
    - Gesamte Codelogik in einem Projekt und derselben Sprache
    - Kann gut komplexen Code im Backend verarbeiten
    - Viele Dokumentation durch Microsoft
  - Nachteile:
    - Verhältnismässig viele Daten hin und her geschoben zwischen Client und Server (dotnet Runtime wird in Browser geladen) → Performance Impact
    - Eher im Enterprise Bereich verbreitet
    - Eingeschränkte Frontend-Ressourcen (da eher neu, Auswahl an Frontend-Bibliotheken eher klein)

[80] https://kubernetes.io/docs/reference/using-api/client-libraries/
[81] https://github.com/kubernetes-client/c/
[82] https://github.com/kubernetes-client/csharp
[83] https://github.com/kubernetes/client-go/
[84] https://github.com/kubernetes-client/haskell
[85] https://github.com/kubernetes-client/java/
[86] https://github.com/kubernetes-client/javascript
[87] https://github.com/kubernetes-client/perl/
[88] https://github.com/kubernetes-client/python/
[89] https://github.com/kubernetes-client/ruby/
[90] https://Next.js.org/
[91] https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor

- **Angular Frontend[92] und Java (Spring[93]) Backend**
    - Vorteile:
        - Viel Dokumentation und Community Support, da Angular und Java Spring weit verbreitet
        - Klare Trennung von Frontend + Backend (kann auch ein Nachteil sein)
    - Nachteile:
        - Zwei verschiedene Programmiersprachen für Frontend und Backend
        - Schnittstelle zwischen Frontend und Backend zu erstellen im Verhältnis zu anderen Tech Stacks eher aufwändig (erstellen von APIs, DTOs, Parser, etc)
        - Spring kann höhere Konfiguration und mehr Ressourcen benötigen

Die ideale Wahl für den Tech-Stack ist Next.js, da es sich um ein vollständiges Fullstack-Framework handelt, das speziell für die Entwicklung flexibler Softwarelösungen konzipiert wurde. Es ermöglicht Entwicklern, Software schnell und effizient zu erstellen und bei Bedarf problemlos zu erweitern. Zudem beschränkt sich die SA damit auf eine Programmiersprache, welche von den Kubernetes SDKs unterstützt wird.

2.9.1 Datenbank

Um Daten wie Konfigurationsinformationen zu speichern, soll eine Datenbank verwendet werden. Ideal für dieses Projekt geeignet ist SQLite[94], da es eine (fast) vollwertige Relationale Datenbank abbildet, bei welcher die Daten in einer einzigen Datei gespeichert werden. Dadurch wird sichergestellt, dass über ein Schema auf Daten zugegriffen werden kann, es jedoch keine zusätzlichen Pods benötigt werden, um eine Datenbank im Kubernetes Cluster zu betreiben und somit auch keine Serverressourcen verschwendet werden.

2.10 Funktionale Anforderungen (FA)

In der nachfolgenden Tabelle sind alle funktionalen Anforderungen aufgelistet. Diese sind jeweils einer Kategorie zugeordnet. Ein App beschreibt ein Projekt, welches in dem Kubernetes Cluster von einem Benutzer deployed wird.

ID	Beschreibung	Kategorie
FA-1	Der Benutzer kann sich beim System registrieren und sich danach mit seinen Credentials anmelden.	Benutzerverwaltung
FA-2	Der Benutzer kann seine hinterlegten Benutzerdaten anschauen. (z.B. die hinterlegte Mailadresse)	Benutzerverwaltung
FA-3	Der Benutzer hat die Möglichkeit sein Passwort zu ändern.	Benutzerverwaltung
FA-4	Das System muss es ermöglichen, für jedes App einen separaten Namespace zu erstellen.	App-Deployment
FA-5	Der Namespace wird automatisch entfernt, sobald das App gelöscht wird.	App-Deployment
FA-6	Der Benutzer muss in der Lage sein, mehrere verschiedene Apps zu erstellen.	App-Deployment
FA-7	Auf dem Dashboard sollen alle Apps des Benutzers übersichtlich dargestellt werden. Mit Klick auf ein App werden die Details wie Git-URL, Service-Logs und Deployment Logs ersichtlich.	App-Deployment
FA-8	Der Benutzer muss den Port des Pods pro App konfigurieren können.	App-Deployment

[92] https://angular.dev/
[93] https://spring.io/
[94] https://www.sqlite.org/

FA-9	Das System muss es ermöglichen, Environment Variablen für jede App zu setzen.	App-Deployment
FA-10	Der Benutzer kann pro App eine oder mehrere Domains angeben, die vom Internet her erreichbar sind.	App-Deployment
FA-11	Benutzer müssen Ressourcenlimitierungen (CPU, RAM, Speicher) pro App festlegen können.	App-Deployment
FA-12	Der Benutzer muss die Anzahl der Replikas für eine App konfigurieren können.	App-Deployment
FA-13	Die Apps können den persistenten Speicher des Clusters nutzen und ein oder mehrere RWO- oder RWM-Volumes in jeden Container einer App mounten.	App-Deployment
FA-14	Jedes Deployment hat die Möglichkeit eine eigene Ingress-Konfiguration zu erhalten, um HTTP/HTTPS-Traffic auf den Pod weiterzuleiten.	App-Deployment
FA-15	Die Zertifikate für HTTPS müssen automatisch via Let's Encrypt und Certmanager ausgestellt und verwaltet werden können.	App-Deployment
FA-16	Vorlagen für verschiedene Datenbanken (z.B. PostgreSQL) müssen bereitgestellt und mit zwei Klicks deployed werden können.	App-Deployment
FA-17	Der Source-Code eines Apps muss über eine Git-URL und wenn nötig Credentials gepullt werden können während eines Build-Prozesses.	Build-Prozess
FA-18	Das System muss Artefakte des Build-Prozesses speichern und für spätere Deployments einer App wiederverwendbar machen.	Build-Prozess
FA-19	Die Logs des Build-Prozesses müssen angeschaut werden können.	Build-Prozess

ID	Beschreibung	Kategorie
FA-20	Das System muss Service Logs für jede App speichern und dem Benutzer zugänglich machen.	Logging und Monitoring
FA-21	Das System muss den Ressourcenverbrauch (CPU, RAM, Speicher) pro Deployment anzeigen.	Logging und Monitoring
FA-21	Das System muss eine Möglichkeit via Web UI bieten, die den Zugriff auf das Dateisystem der App ermöglicht.	Optionale Funktionen
FA-23	Der Benutzer muss Backups der Longhorn Volumes zu S3 Buckets erstellen können.	Optionale Funktionen
FA-24	Mit der Installation von QuickStack sollen im Hintergrund auch die Nodes des Clusters gehärtet werden. Dies wird mit Absetzen von entsprechenden Kommandozeilen bewerkstelligt und betrifft folgende Sicherheitsaspekte: Automatische Updates des Betriebssystems aktivieren, Firewall nur für notwendige Dienste öffnen, SSH-Passwortauthentifizierung deaktivieren und Default Port ändern, Benutzer ohne Rootrechte erstellen, Cron-Job für automatische Sicherheitsupdates und Paketupdates.	Optionale Funktionen

Tabelle 12: Funktionale Anforderungen (FA)

2.11 Nichtfunktionale Anforderungen (NFA)

In der nachfolgenden Tabelle sind alle nichtfunktionalen Anforderungen aufgelistet.

ID	NFA-1
Anforderung	Usability - Learnability
Beschreibung	Ein neuer Benutzer kann QuickStack mühelos verwenden.
Messung	Durchführung von Benutzertests mit mind. drei Testpersonen, welche QuickStack zum ersten Mal sehen. Diese Personen führen anhand eines abgegebenen Testbogens Aufgaben aus, wobei die messbaren Ausführungszeiten sowie auch die Resultate im Testbogen festgehalten werden. Vorgabe Testpersonen: Befinden sich zurzeit auch im Informatikstudium der OST und ist im dritten Semester oder höher.
Verifizierung	Anhand des Testbogens und den Rückmeldungen der Testpersonen kann bestimmt werden, ob QuickStack die nichtfunktionale Anforderung erfüllt.

ID	NFA-2
Anforderung	Performance Efficiency - Responsiveness
Beschreibung	Der Benutzer erfährt praktisch keine Wartezeiten während der Navigation durch QuickStack.
Messung	Die durchschnittliche Ladezeit in QuickStack für das Laden einzelner Seiten darf nicht länger als 3 Sekunden betragen. Dadurch wird eine positive Benutzererfahrung sichergestellt.
Verifizierung	Der Test kann mithilfe der «Google Chrome Developer Tools» vollzogen werden und wird voraussichtlich sowohl manuell als auch anhand eines vordefinierten Ablaufplans durchgeführt. Die Verwendung eines Ablaufplans gewährleistet die Reproduzierbarkeit des Tests.

ID	NFA-3
Anforderung	Performance Efficiency - Capacity
Beschreibung	QuickStack muss auch mit der gleichzeitigen Verwaltung von mehreren Apps ordnungsgemäss funktionieren und darf keine Fehler anzeigen.

	Abgrenzung: Diese nichtfunktionale Anforderung ist unabhängig von der Serverlast und bezieht sich nur auf die korrekte Orchestrierung der Anzahl von App durch QuickStack allein.
Messung	QuickStack soll in der Lage sein mindestens 10 Apps ohne Probleme zu bewältigen und zu verwalten. Die Serverleistung wird nicht berücksichtigt.
Verifizierung	Es werden gesamthaft 10 Apps deployed. Danach wird QuickStack vom Benutzer verwendet und anhand eines Drehbuchs werden verschiedene Funktionen und Abfragen via QuickStack durchgeführt. Anhand des subjektiven Eindrucks und den Rückmeldungen des Benutzers kann bestimmt werden, ob QuickStack die nichtfunktionale Anforderung erfüllt.

ID	NFA-4
Anforderung	Schnelle Installation von QuickStack
Beschreibung	QuickStack soll innerhalb von fünf Minuten auf dem VPS (Virtual Private Server) aufgesetzt werden können.
Messung	Die Installationszeit wird vom Absetzen des ersten Kommandozeilenbefehls bis zum ersten erfolgreichen Zugriff auf das Web UI gemessen.
Verifizierung	Anhand der gemessenen Installationszeit kann bestimmt werden, ob QuickStack die nichtfunktionale Anforderung erfüllt.

ID	NFA-5
Anforderung	Benutzer muss maximal fünf Kommandozeilenbefehle für Installation von QuickStack absetzen
Beschreibung	Für die Installation von QuickStack müssen maximal nur drei Kommandozeilenbefehle abgesetzt werden, bis QuickStack erfolgreich installiert und über das Web UI erreichbar ist.
Messung	Die Anzahl Kommandozeilenbefehle werden bis zum ersten erfolgreichen Zugriff auf das Web UI gezählt.
Verifizierung	Anhand der gezählten Kommandozeilenbefehle kann bestimmt werden, ob QuickStack die nichtfunktionale Anforderung erfüllt.

Tabelle 13: Nichtfunktionale Anforderungen (NFA)

3. Lösungskonzept

Im nachfolgenden Kapitel wird die Architektur mithilfe von Diagrammen dargestellt und beschrieben.

3.1 Datenbankschema

Das Datenbankschema kann in zwei Teile aufgegliedert werden. Der erste Teil des Schemas besteht aus Tabellen für das Authentifizierungsframework «NextAuth». Diese Tabellen werden vom Framework vorgegeben. Mit NextAuth wird die Authentifizierung und Authorisierung der Applikation sowie die

dazugehörigen Sicherheitsmechanismen an NextAuth ausgelagert, wodurch einerseits Zeit gespart und andererseits die Sicherheit durch eine standardisierte Library erhöht wird.

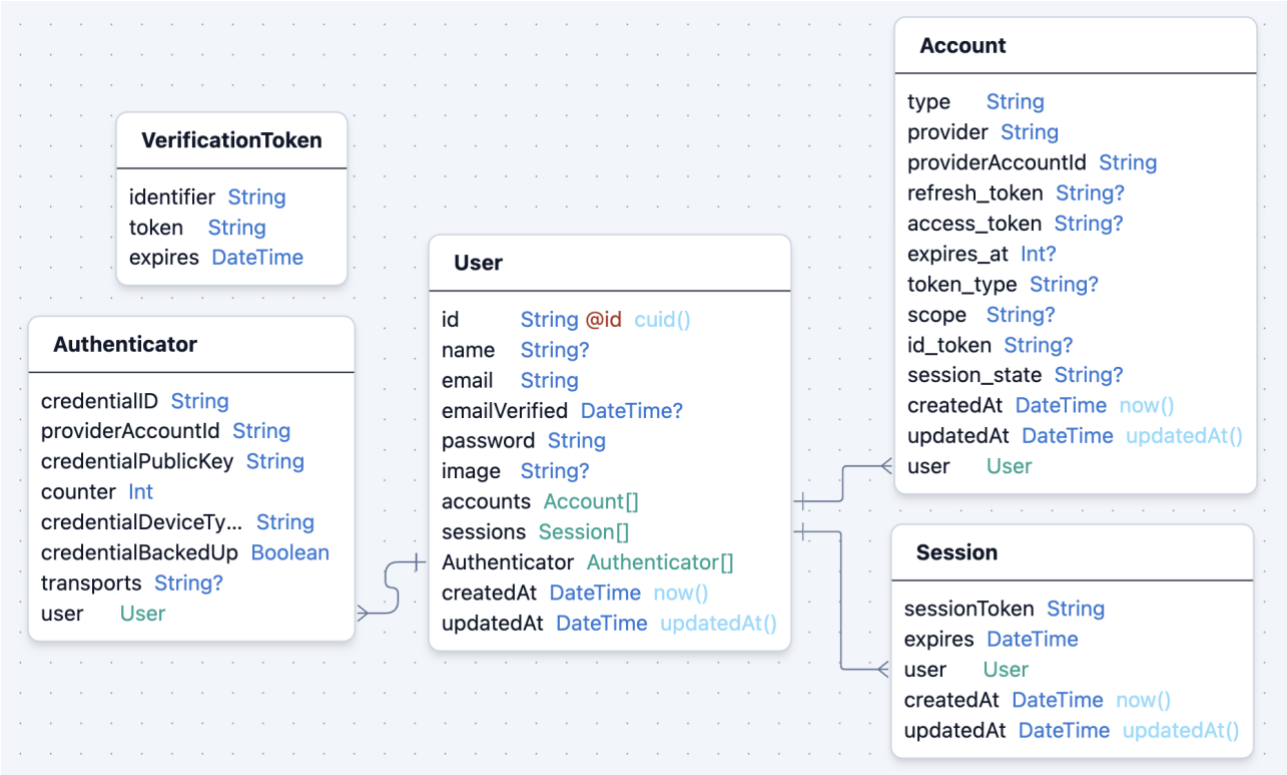


Abbildung 5: Datenbankschema für das Authentifizierungsframework «NextAuth»

Das Datenbankschema für die App Logik umfasst sechs Tabellen: Project, App, AppDomain, AppVolume AppPort und Parameter. Ein Projekt enthält mehrere Apps, welche zusammenhängen. Ein Beispiel hierzu wäre eine Worpress Seite, welche aus einem Backend-Pod (PHP) besteht und einem Datenbank-Pod (MySQL). Diese beiden Pods befinden sich in einem Projekt. Jede App in einem Projekt kann keine, eine oder mehrere Domains und App Volumes haben. Die Tabelle AppPort wird verwendet, um zusätzliche Ports angeben zu können, welche im internen Netzwerk (Kubernetes Service) von Kubernetes zur Verfügung gestellt werden sollen. Alle Entitäten enthalten nur die nötigsten Funktionen, um eine App in einen Kubernetes Cluster mit all seinen Kubernetes Objekten zu deployen. Die Entität Parameter enthält Konfigurationsinformationen,

wie beispielsweise, welche E-Mail für Let's Encrypt Zertifikate verwendet werden soll oder den Hostnamen für das QuickStack-Admin-Panel.

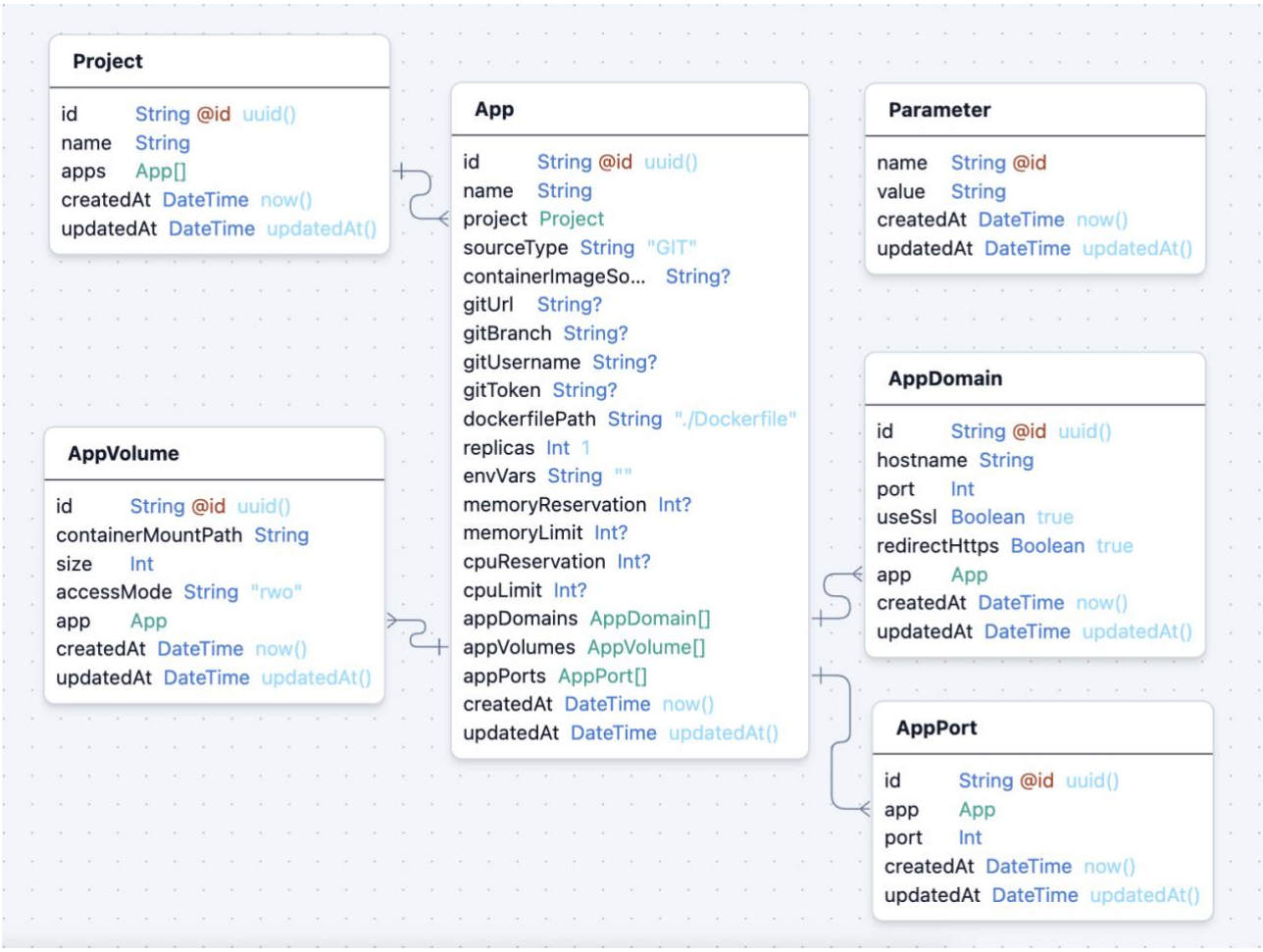


Abbildung 6: Datenbankschema für die App Logik

3.2 C4 Diagramm

Nachfolgend wird das Zusammenspiel der einzelnen Komponenten durch ein Level 1 und Level 2 in einem C4 Diagramm dargestellt.

3.2.1 Level 1: Kontext

Auf dem Level 1 des C4-Diagramms befinden sich ein Akteur und zwei Software-Systeme. Der Akteur ist ein QuickStack Benutzer, welcher über das Web UI auf die Applikation QuickStack zugreift. In dieser hat er die Möglichkeit Projekte anzulegen, Apps zu erstellen, Apps mit seinem Git Repository zu verknüpfen, einen Build-Prozess anzustossen und schlussendlich seine App zu deployen. Die einzigen externen Systeme, welche hierzu benötigt werden, sind das Git Repository, auf welchem sich der Sourcecode der zu deployenden App

befindet und Dockerhub, für öffentliche und bereits gebuildete Docker Images (z.B. Postgres Image für Datenbanken).

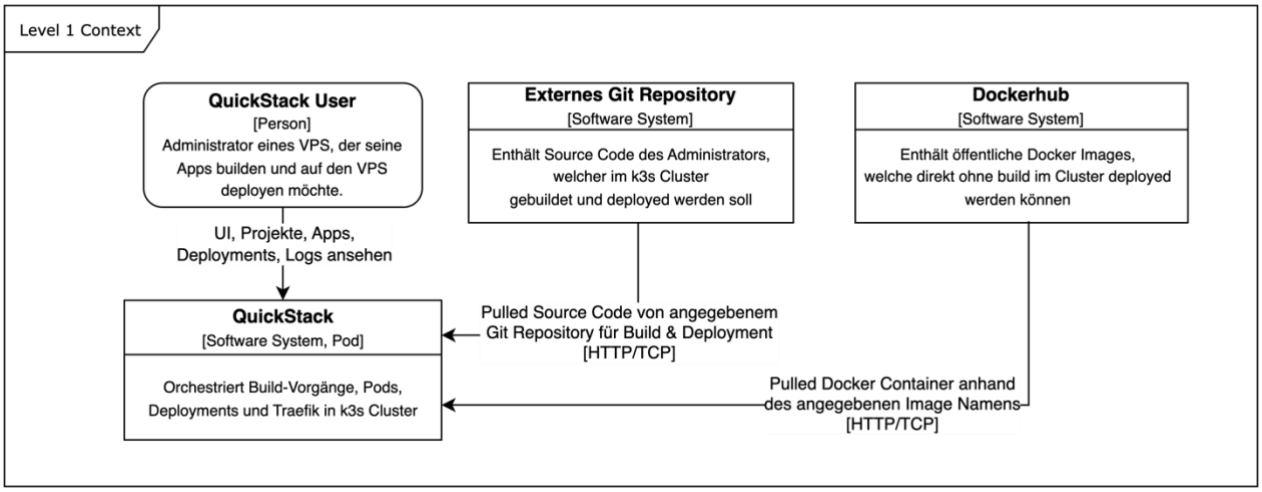


Abbildung 7: C4 Level 1 Kontext

3.2.2 Level 2: Containers

Das C4-Diagramm auf Level 2 visualisiert die detaillierte Container-Architektur des QuickStack-Systems, welches die Build- und Deployment-Vorgänge für Applikationen auf dem k3s-Cluster des jeweiligen VPS (auf welchem QuickStack installiert ist) orchestriert. Die zentrale Komponente in diesem Diagramm ist der QuickStack-Backend-Container, in welchem eine Next.js-Anwendung ausgeführt wird und sowohl als Orchestrator für den k3s-Cluster als auch als Schnittstelle für Benutzerinteraktionen fungiert.

Ein QuickStack-Benutzer (VPS-Administrator) besucht die QuickStack-Web-Oberfläche, um Apps zu verwalten. Eine App ist eine Software, welche vom Benutzer erstellt wurde und diese in den Cluster deployen möchte. Der ServiceLB empfängt den Traffic und leitet ihn über den Traefik Ingress Controller an die entsprechende App oder an QuickStack weiter. Wenn ein Benutzer ein Deployment für eine App ausführt, startet QuickStack einen Kubernetes Job über den Kubernetes API Server. Dieser Job beinhaltet ein Kaniko Container, welcher den Quellcode aus einem Git Repository zieht, einen Container baut und diesen Container in der internen Container Registry abspeichert (Registry Pod). Der gebuildete Container (App) wird im Anschluss durch QuickStack über die Kubernetes API von kubelet in das Cluster deployed. Falls die App eine Domain-Konfiguration hat, wird der Traefik Ingress aktualisiert, um eingehenden Traffic an die deployte App weiterzuleiten. Speicherinformationen von Nodes und einzelner Persistent Volume Claims (PVC), welche einem Pod angebunden sind, können über die Longhorn API abgerufen werde. Die gelben Komponenten im untenstehenden Diagramm sind Teile der QuickStack Software. Die grünen Komponenten können von QuickStack durch die Kubernetes internen Komponenten (blau, API + kubelet) angesprochen, gestartet und verwaltet werden. Es wurden bewusst nicht alle Komponenten von Kubernetes und Longhorn

aufgezeichnet, um das Diagramm übersichtlich zu halten und nur diejenigen Komponenten herauszuheben, welche direkt durch QuickStack angesprochen werden.

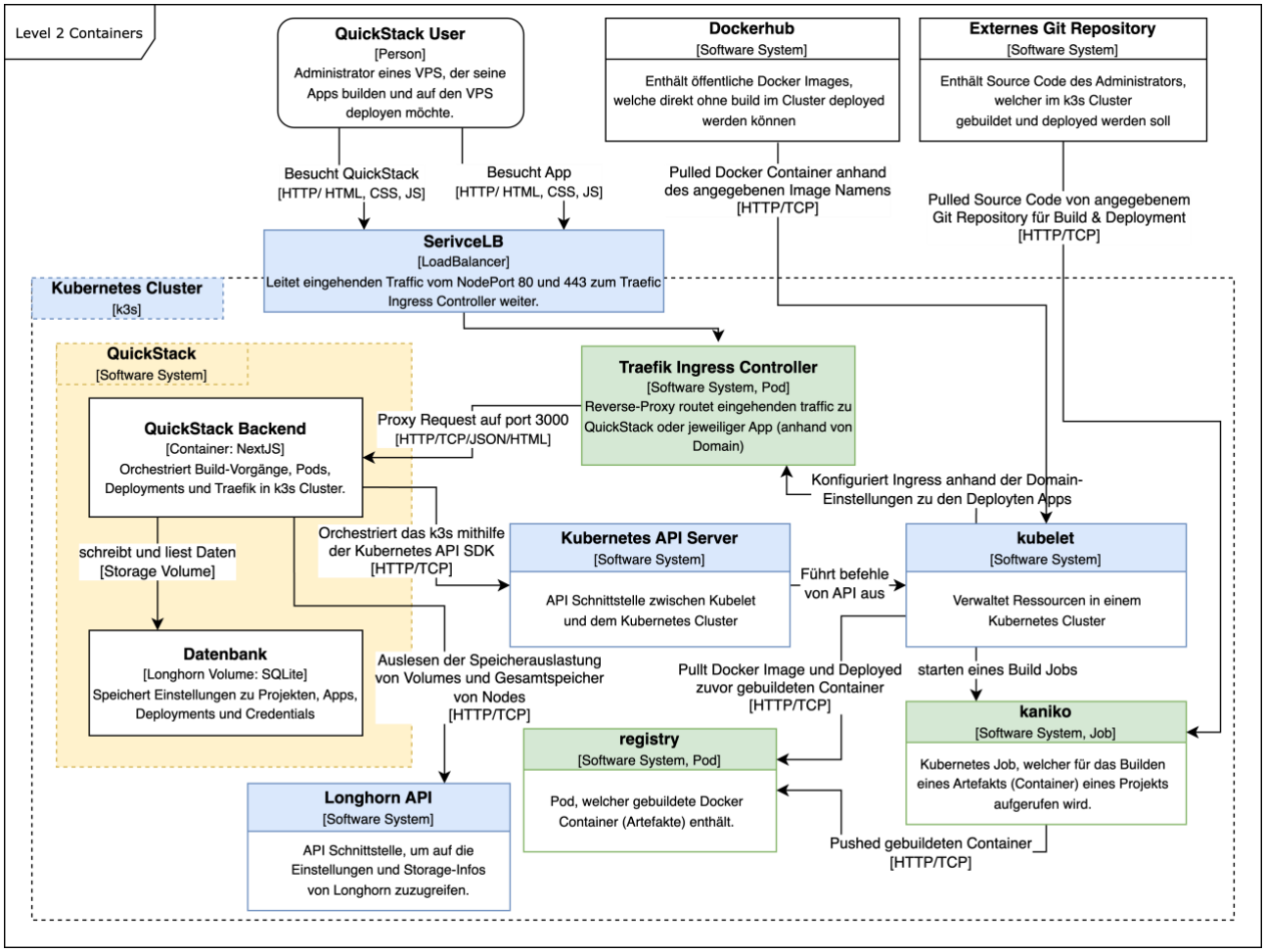


Abbildung 8: C4 Level 2 Containers

3.3 Codearchitektur

Der Code von QuickStack ist modular aufgebaut. Das Backend ist in Controller, Services und Models unterteilt. Im Backend sind drei «boundaries» definiert: die API-Grenze für die REST-API (Server Actions) zwischen dem Frontend und dem Backend sowie der Logik für den Aufbau der SSR-Pages, die Adapter-Grenze zum Laden von Informationen über die Kubernetes sowie Longhorn API und die DB-Grenze für die Datenbank. Das Frontend ist sehr eng mit dem Backend verknüpft, dies aufgrund der Technologiewahl von Next.js und kann daher nicht ohne weiteres einfach so ersetzt werden. Es gäbe die Möglichkeit weitere API-Routes im Next.js Backend zu definieren, wodurch ein anderes Frontend-Framework verwendet werden könnte.

Da wir Prisma als ORM verwenden, ist die Datenbank vom Backend entkoppelt, was es erlaubt, die Datenbank durch eine andere wie Postgres, MySQL, SQLite oder sogar MongoDB zu ersetzen, ohne Änderungen an Datenbankabfragen machen zu müssen. (DB-Grenze).

Durch die Adapter-Grenze ist das Backend von der externen Longhorn API entkoppelt, was es ermöglicht, Longhorn API durch eine andere zu Datenquelle für Storageinformationen zu ersetzen. Durch die k8s SDK entsteht eine Entkopplung der Kubernetes Versionen. Die SDK kann jederzeit mit einer anderen SDK-Version ersetzt werden, um zukünftige Versionen von Kubernetes zu unterstützen.

Da durch Next.js gewisse Ordnerstrukturen vorgegeben werden, wird hier die Ordnerstruktur im Code-Repository im Zusammenspiel mit der Codearchitektur erläutert. In der Projektstruktur befinden sich der fürs Frontend relevante Code in den Ordnern «src/app», «src/components» und «src/frontend», Backend Code im

Ordner «src/backend» und alle Codeteile zu geteiltem Code (z.B. Models, Util-Klassen) im Ordner «src/shared». In Bezug auf das untenstehende Diagramm befinden sich Codeteile von «Services» im Ordner «src/server/services», Codeteile zu Adaptern in «src/server/adapters/>, Codeteile zu «Server Actions und SSR Pages» in «src/app». Die Architektur verfolgt Ansätze von Clean Code nach Robert C. Martin[95], jedoch nicht vollständig.

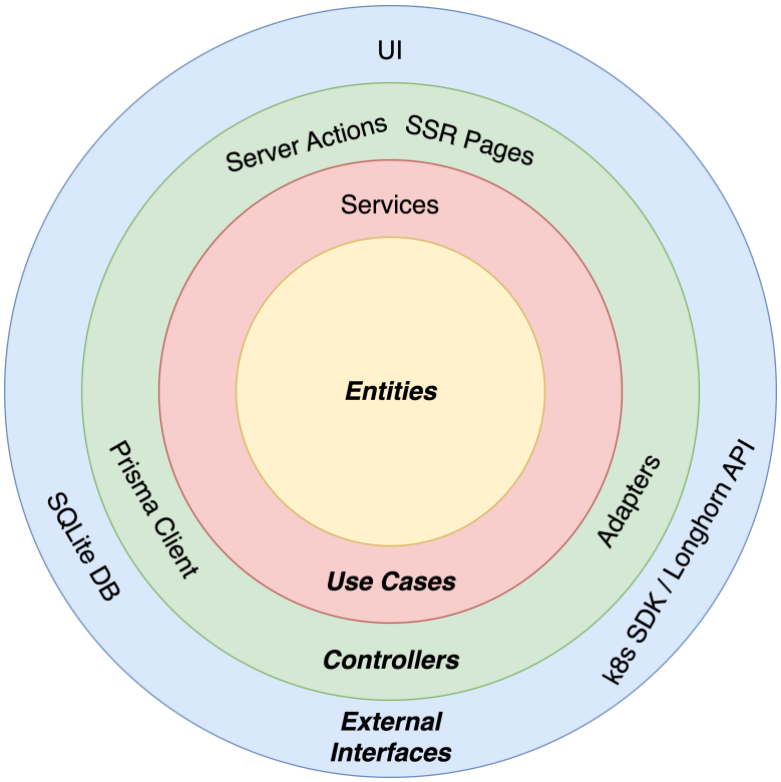


Abbildung 9: Diagramm Clean Code

3.4 Longhorn Schnittstelle

Die API-Daten von Longhorn können über den Service «longhorn-frontend» im Namespace «longhorn-system» innerhalb des Kubernetes Cluster abgerufen werden. Die verwendeten Endpunkte werden in den nachfolgenden Unterkapiteln kurz beschrieben.

3.4.1 PVC – benutzter Volumespeicher auslesen

Verwendeter Endpunkt: http://longhorn-frontend.longhorn-system.svc.cluster.local/v1/volumes/{pvcName}

Bei diesem GET-API-Call werden mithilfe des PVC-Namens alle Informationen des jeweiligen PVCs im JSON-Format zurückgegeben. Von diesen erhaltenen Informationen wird der Wert von «actualSize» extrahiert, welcher dann in QuickStack weiterverarbeitet wird. Diese Information ist dann im «App Monitoring» ersichtlich und zeigt an, wieviel Speicher eines Volumes belegt ist.

3.4.2 Node - verfügbarer und benutzter Gesamtspeicher auslesen

Verwendeter Endpunkt: http://longhorn-frontend.longhorn-system.svc.cluster.local/v1/nodes/{nodeName}

Bei diesem GET-API-Call werden mithilfe des Node-Namens alle Informationen des jeweiligen Nodes im JSON-Format zurückgegeben. Von diesen erhaltenen Informationen werden die Werte «storageMaximum» und «storageAvailable» extrahiert, welche dann in QuickStack weiterverarbeitet werden. Bei

[95] https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

«storageMaximum» handelt es sich um den gesamten vorhandenen Speicher der Node, welcher Longhorn zur Verfügung steht. Mit «storageAvailable» wird der Wert zurückgegeben, welcher anzeigt, wieviel Speicher der Node schon benutzt ist. Diese Werte sind dann unter «Monitoring» ersichtlich.

4. Umsetzung

Im nachfolgenden Kapitel wird auf den Tech-Stack der Applikation eingegangen, Details von einigen Key-Features werden beschrieben und Testergebnisse beschrieben.

4.1 Verwendete Libraries und Tools

Für die Implementierung der Software werden die Tools Next.js[96], Tailwindcss[97], Kubernetes API JS SDK[98], Shadcn/ui[99], Prisma ORM[100], SQLite[101], NextAuth[102], bcrypt[103], XTerm[104], optauth[105], qrcode[106], recharts[107], simple-git[108], socket.io[109], zod[110] und zustand[111]. In der nachfolgenden Tabelle werden die Tools beschrieben und eingeordnet, in welchem Bereich der Software das Tool Verwendung findet.

Tool	Beschreibung
Next.js	Ein React-basiertes Web-Framework, das serverseitiges Rendering und statische Webseiten-Generierung unterstützt. Es vereinfacht die Entwicklung von Webanwendungen mit Funktionen wie Seitenrouting und API-Handling. Next.js verwendet standardmässig weitere externe Libraries, welche bei der Installation automatisch mitinstalliert werden. Diese werden hier nicht mitaufgelistet.
Tailwindcss	Ein Utility-First CSS-Framework, das schnelle und massgeschneiderte Benutzeroberflächen ermöglicht. Es bietet eine grosse Anzahl an vordefinierten CSS-Klassen, um das Styling direkt im HTML-Code vorzunehmen und sorgt so für eine schnelle Entwicklung und saubere Wartung.
Kubernetes API JavaScript SDK	Eine JavaScript-Bibliothek, die die Interaktion mit der Kubernetes-API ermöglicht. Sie wird verwendet, um Kubernetes-Ressourcen zu erstellen, zu aktualisieren, zu löschen und zu verwalten.
Shadcn/ui	Eine auf Tailwind CSS basierende UI-Komponentenbibliothek, die anpassbare und einfach zu verwendende Komponenten für moderne React Web-Interfaces bereitstellt.
Prisma ORM	Ein ORM (Object-Relational Mapping)-Tool für Node.js und TypeScript, das den Datenbankzugriff vereinfacht. Prisma erlaubt eine intuitive Datenmodellierung und unterstützt Migrationen und die Generierung von TypeScript-Typen.
SQLite	Eine leichtgewichtige, eingebettete SQL-Datenbank, die keine Serverkomponente benötigt. Sie eignet sich ideal für Anwendungen mit geringem bis mittlerem Datenvolumen und erlaubt einen einfachen Zugriff auf relationale Daten. Die Daten der Datenbank werden lediglich in ein einziges File gespeichert.
NextAuth	NextAuth.js ist eine Open-Source-Bibliothek für die Authentifizierung in Next.js-Anwendungen. Sie unterstützt verschiedene Authentifizierungsprovider, darunter OAuth (z. B. Google, GitHub), E-Mail/Passwort sowie benutzerdefinierte Anmeldeverfahren. Die Library verwaltet Sitzungen und bietet sichere Mechanismen für den Zugriff auf Benutzerdaten, mit nahtloser Integration in Next.js.

96 https://Next.js.org/
97 https://tailwindcss.com/
98 https://github.com/kubernetes-client/javascript
99 https://ui.shadcn.com/
100 https://www.prisma.io/
101 https://www.sqlite.org/
102 https://next-auth.js.org/
103 https://github.com/kelektiv/node.bcrypt.js
104 https://github.com/xtermjs/xterm.js
105 https://github.com/hectorm/otpauth
106 https://github.com/soldair/node-qrcode
107 https://github.com/recharts/recharts
108 https://github.com/steveukx/git-js
109 https://github.com/socketio/socket.io/tree/main/packages/socket.io
110 https://zod.dev/
111 https://github.com/pmndrs/zustand

bcrypt	Verwendet für das hashen und abgleichen von Passwörtern.
XTerm	JavaScript Library zur einfachen Integration eines Web-Terminals in eine Website.
optauth	JavaScript Library zur Erstellung und Verifizierung von 2FA Tokens.
qrcode	JavaScript Library zur generierung von QR-Codes für die Einrichtung von 2FA.
recharts	React Library für die Anzeige von Charts in der Monitoring-Übersicht.
simple-git	NodeJS Library, um einen lokal installierten Git-Client anzusteuern. Wird verwendet, um Versionen von Git-Repositoriers zu vergleichen.
socket.io	JavaScript Library zur einfachen Integration von Websockets. Wird verwendet, um das Web-Terminal (XTerm) mit dem Pod zu verbinden.
zod	JavaScript Library für Schema-Validierung. Wird für die Validierung von Formularen (in Frontend & Backend) verwendet.
zustand	JavaScript Library für simples State-Management im Next.js Frontend.

Tabelle 14: Verwendete Libraries und Tools

4.2 Features

Im folgenden Abschnitt werden einige Key-Features der Software beschrieben und mithilfe von Diagrammen im Detail beleuchtet.

4.2.1 Installation von QuickStack auf einem VPS

Um QuickStack auf einem VPS installieren zu können, müssen die folgenden Komponenten installiert werden:

- **K3s:** Kubernetes Distribution für Orchestrierung von Containern
- **Longhorn:** Shared-Storage provider für Kubernetes, damit auf mehreren VPS-Nodes auf dieselben Persistent Volume Claim (PVC) zugegriffen werden können
- **NFS-Common:** Benötigtes Tool, damit ReadWriteMany PVCs erstellt werden können[112].
- **Cert-Manager:** Kubernetes Erweiterung, um Zertifikate verwalten zu können. In QuickStack für SSL-Zertifikate verwendet.
- **QuickStack:** Die Applikation QuickStack als Deployment innerhalb des Kubernetes Clusters.

Damit trotz all dieser Tools die Installation auf einem VPS möglichst einfach ist, wird ein Installationsscript verwendet, welches unter https://get.quickstack.dev/setup.sh heruntergeladen werden kann. Die nachfolgende Grafik zeigt den Ablauf der Installation von QuickStack.

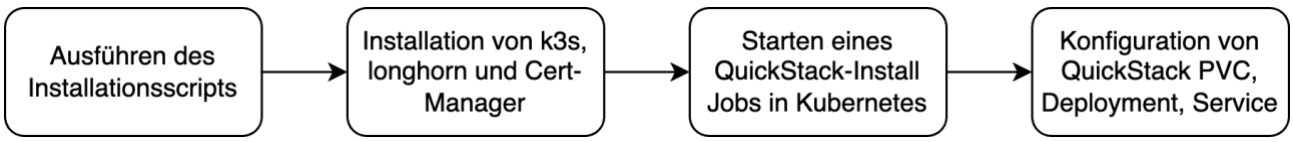


Abbildung 10: Diagramm Installationsablauf von QuickStack auf einem VPS

Nach dem Ausführen des Installationsscripts mittels *curl -sfL https://get.quickstack.dev/setup.sh | sh –* auf dem Terminal des VPS wird k3s, Longhorn und der Cert-Manager installiert. Sobald alle Installationen abgeschlossen sind und alle Pods in Kubernetes gestartet wurden, startet ein Kubernetes Objekt vom Typ «Job». Dieser Job startet den QuickStack Container mit einem Flag «START_MODE=setup». Durch dieses Flag wird nicht die reguläre QuickStack Applikation gestartet, sondern es wird überprüft, welche QuickStack Komponenten bereits im Kubernetes Cluster konfiguriert wurden und welche nicht. Jene, die noch nicht konfiguriert wurden, werden konfiguriert. Dies umfasst den Cert-Manager, ein PVC für die SQLite Datenbank und die Deployment Logs von QuickStack, ein Kubernetes-Deployment und ein Kubernetes Service. Dieser Job kann mehrmals ausgeführt werden und zudem bei Problemen in einem bestehenden QuickStack Cluster

112 https://longhorn.io/docs/1.7.2/deploy/install/#installing-nfsv4-client

verwendet werden. Nachdem die Konfigurationen mithilfe des Jobs ausgeführt wurden, wird der Job beendet und die QuickStack Applikation ist über das erstellte Kubernetes-Deployment über den Port 30000 auf allen VPS-Nodes erreichbar. Der QuickStack Benutzer erstellt in diesem Schritt das Benutzerkonto für die Authentifizierung an der QuickStack Applikation.

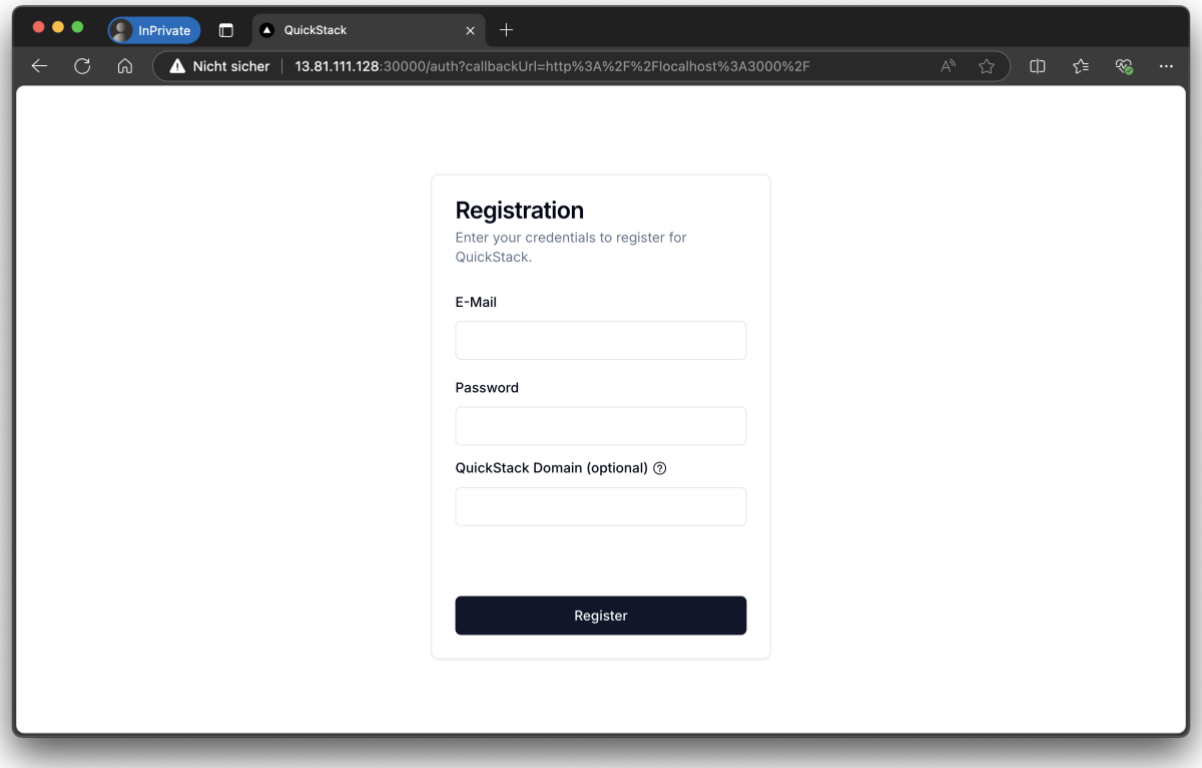


Abbildung 11: Screenshot Benutzerkonto in QuickStack erstellen

Nach der Registrierung wird im Hintergrund das Benutzerkonto angelegt und ein Kubernetes Objekt vom Typ «Cert-Issuer» angelegt. Der «Cert-Issuer» wird für Let's Encrypt konfiguriert und standardmässig wird die Registrierungs-E-Mail an Let's Encrypt für die Erstellung von SSL Zertifikaten weitergegeben. Nach der Registrierung ist QuickStack vollständig konfiguriert und bereit für die Verwendung.

4.2.1.1 Installation auf einem weiteren Node

K3s bietet die Möglichkeit das Kubernetes Cluster durch weitere Nodes (VPS) zu erweitern. Damit auf den weiteren Nodes neben k3s nfs-common installiert wird, verläuft die Installation auf einem weiteren VPS ebenfalls über ein Script. Um das Script zu starten, muss die IP-Adresse des ersten Nodes und ein Join-Key

mitgegeben werden. Den ganzen Befehl, um das Script zu starten, inklusive des Join-Tokens, befindet sich im UI von QuickStack unter «Einstellungen» und «Cluster».

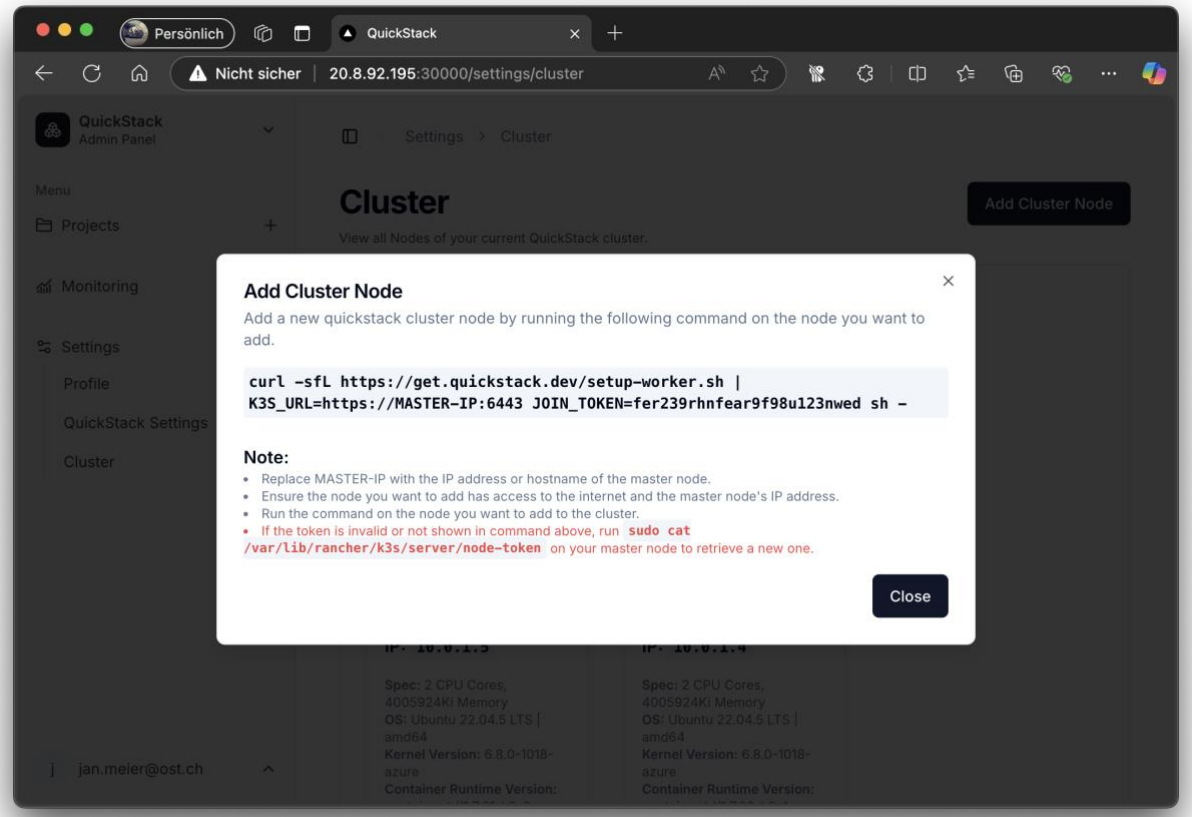


Abbildung 12: Screenshot Cluster Node in QuickStack hinzufügen

Nach erfolgreichem ausführen des Scripts wird die zusätzliche Node in den Einstellungen unter «Cluster» angezeigt.

4.2.2 Build und Deployment einer App

QuickStack bietet zwei Varianten von Deployments an. Mithilfe der ersten Variante können bestehende Docker Images direkt deployet werden (ohne Build Prozess). Die zweite Variante ist das Deployen eines Git-Repositories. Hierbei wird im ersten Schritt das Repository anhand einer Git-URL (http-Format) und den optionalen Anmeldeinformationen (für private Repositories) in einen temporären Ordner in den QuickStack Container gecloned. Falls bereits ein älteres Deployment besteht, wird anhand des letzten Git-Commit-Hash verglichen, ob nochmals ein Build Prozess gestartet werden muss (Bei Änderungen = JA, keine Codeänderungen = Nein). Der Build der zu deployenden App findet in einem separaten Container statt. Für den Build wird der «kaniko[113]»-Container von Google verwendet, welcher die Git-Credentials, den Pfad zu einem Dockerfile innerhalb des Repositories und die URL des Docker Registries enthält. Das Dockerfile muss im Vorhinein vom QuickStack Benutzer im Source Code Repository mitgegeben werden. Ohne Dockerfile kann kein Build gestartet werden. Als Docker Registry wird eine im Kubernetes-Cluster gehosteter Registry-Pod verwendet. Die Build-Artefakte werden innerhalb des Clusters gespeichert. QuickStack wartet im während des Buildvorgangs innerhalb einer «sleep-schleife» bis der Build im kaniko-Container beendet wurde. Bei

113

https://github.com/GoogleContainerTools/kaniko

erfolgreichem Beenden, werden Persistent Volume Claims (PVC) gemäss den App-Einstellungen in QuickStack erstellt. Im Anschluss wird ein Kubernetes Objekt vom Typ Deployment erstellt, worin alle dazugehörenden PVCs verknüpft und Container Image Source Daten aus dem Build Prozess, CPU / RAM Limits, Replica-Anzahl und Environment Variablen aus den App-Einstellungen eingetragen werden. Nach diesem Schritt läuft der gebuildete Container bereits im Kubernetes Cluster als Pod, jedoch wurden noch keine Service (Kubernetes internes Load Balancing) und Ingress Konfigurationen vorgenommen, welche den Pod von ausserhalb über eine Domain erreichbar machen. Im Anschluss an das Erstellen des Deployments, werden alte PVCs, welche nicht mehr verwendet werden (z.B. wenn der QuickStack Benutzer ein Volume aus seinen App Einstellungen entfernt) gelöscht. Als letztes folgt das Erstellen von Kubernetes Objekten vom Typ «Service» und «Ingress». Diese steuern das Routing von Anfragen an den Server über eine Domain und leiten den Traffic an den bestimmten Pod weiter. Zudem wird bei mehreren Replicas automatisch der Traffic via Load Balancing an alle Pods verteilt und mit dem Erstellen eines «Ingress» Objekts (inkl. TLS-Konfiguration) die Generierung eines SSL-Zertifikats angestossen.

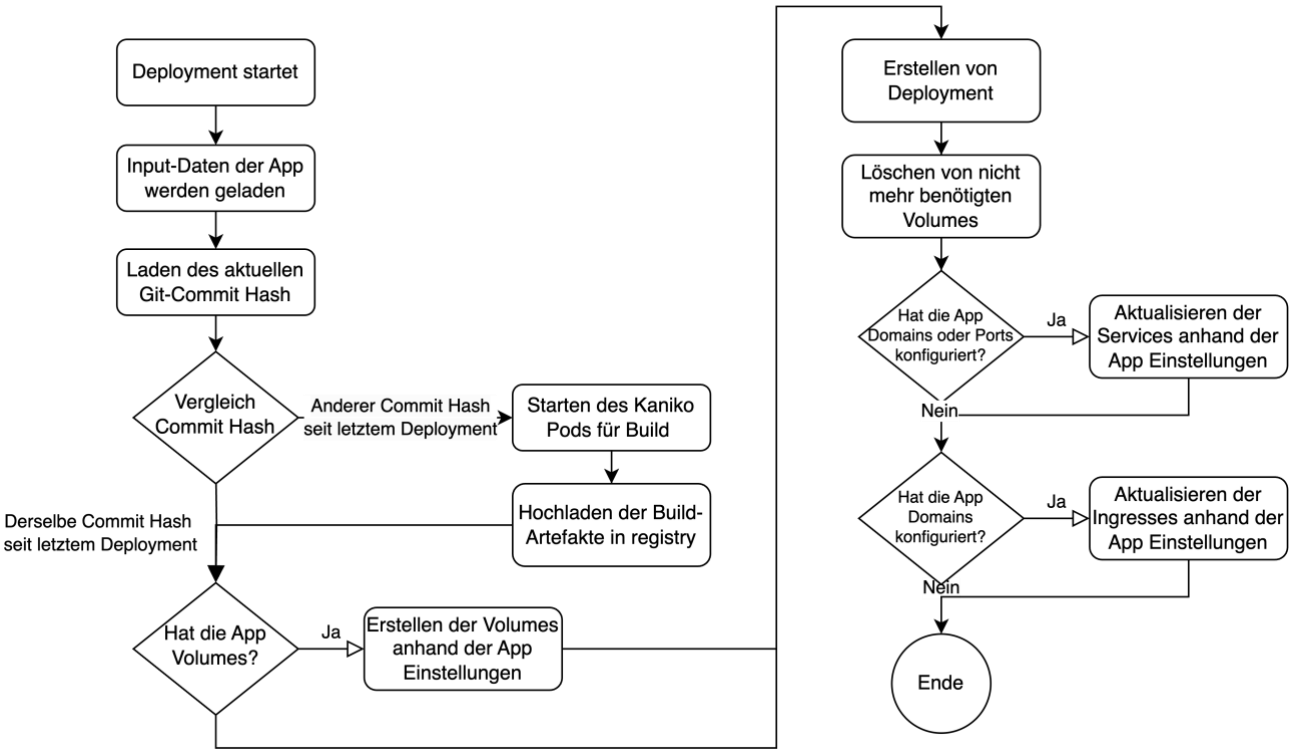


Abbildung 13: Flussdiagramm Deployment-Prozess

4.2.3 Deployment Log Management

Damit dem QuickStack Benutzer möglichst viele Informationen über das Deployment zur Verfügung stehen, wird der Output des Build-Prozesses und zusätzliche Informationen zum Deployment geloggt und auf QuickStack angezeigt. Da für das Deployment Logs aus verschiedenen Quellen gesammelt werden, müssen diese konsolidiert und zusammenhängend dem Benutzer angezeigt werden. Eine Log-Quelle ist der Output des kaniko Buildcontainers und eine Log-Quelle sind Informationen aus dem QuickStack Container während des Deployment Vorgangs (z.B. falls während dem Deployment in QuickStack ein Fehler auftritt oder die Ausgabe der Anzahl erkannten Environment Variablen). Für jedes Deployment wird in QuickStack eine Deployment-ID generiert, welche einerseits bei dem Deployment Kubernetes Objekt und dem Kaniko Build Container als Attribut mitgegeben wird und ein Log-File in einem persistierten Ordner innerhalb des QuickStack Containers erstellt. Alle Deployment relevanten Logs aus dem kaniko Build-Container und aus QuickStack werden direkt in das zugehörige log-File geschrieben. Das bietet die Möglichkeit zusätzlich Logs aus weiteren Quellen zusammen zu führen. Wenn ein QuickStack Benutzer auf dem Frontend die Logs einen Deployments ansieht, wird ein Stream an das backend eröffnet und vom backend ein «watch» auf das logfile

gemacht. Sobald eine Änderung am Logfile geschieht, werden die neuen Log-Zeilen via Server-Sent-Events an das Frontend gesendet.

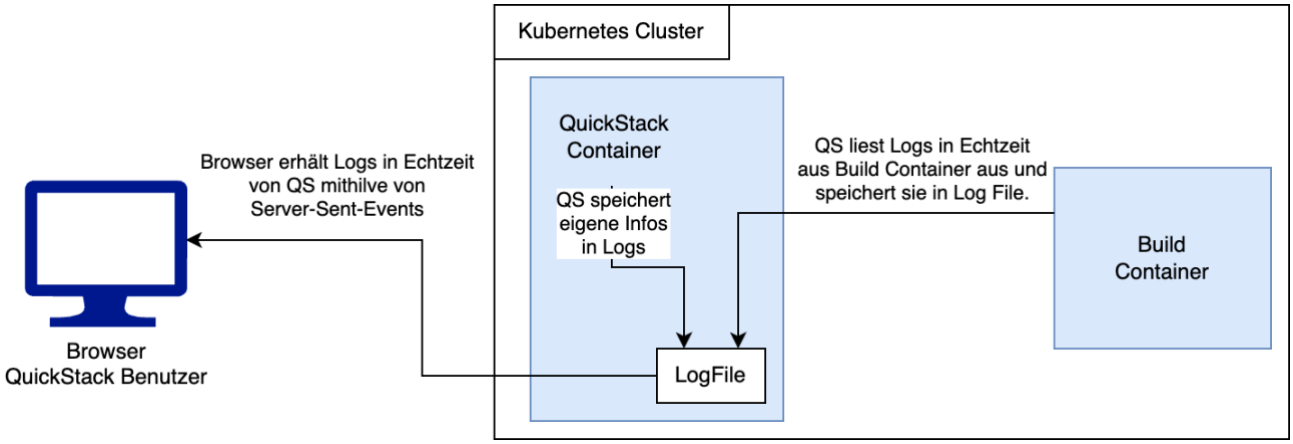


Abbildung 14: Diagramm Log-File-Management

Die Build-Logs können vom QuickStack Benutzer in der App Übersicht über ein Dialog angesehen werden. Ein Beispiel für ein Deployment eines Dockerhub Containers wird in der nachfolgenden Grafik angezeigt:

Deployment Logs

View the logs for the selected deployment 18.12.2024 16:03.

```
Stream opened, loading build logs...

------------------------------------------------
 Deployment:   dfe4b738-26ba-4d41-9f5b-bdd39faa02a7
 App:          app-wordpress-container-2dbd4702
 Project:      proj-wordpress-63040b77
------------------------------------------------
[2024-12-18T15:03:47.841Z]: Starting deployment of containter...
[2024-12-18T15:03:47.864Z]: Configured 1 Storage Volumes.
[2024-12-18T15:03:47.868Z]: Configured 7 Env Variables.
[2024-12-18T15:03:47.881Z]: Creating deployment...
[2024-12-18T15:03:47.954Z]: Updating service (HTTP) with ports 80...
[2024-12-18T15:03:47.962Z]: Updating ingress...
[2024-12-18T15:03:48.005Z]: Configuring Ingress with Domain https://wordpress.fallback.biersoeckli.ch --> app-wordpress-container-2dbd4702:80
[2024-12-18T15:03:48.027Z]: Deployment applied
```

Tabelle 15 Deployment Logs

4.2.4 Web-Terminal

Um zusätzliche Informationen aus einer laufenden Anwendung für Debugging-Zwecke zu extrahieren, ist es nützlich, wenn der QuickStack Benutzer auf das Terminal der laufenden App zugreifen kann. Für ein solches Web-Terminal wird eine bidirektionale Verbindung benötigt. Auf der einen Seite müssen Eingaben in das Terminal via dem QuickStack Backend an das Terminal im Pod gesendet werden und auf der anderen Seite alle Ausgaben des Terminals auf dem Pod via das QuickStack Backend an das Frontend gesendet werden. Für die Verbindung zwischen Pod und Backend kann die Kubernetes API SDK verwendet werden. Bei der Implementierung zwischen dem QuickStack Backend und dem Frontend im Browser wird eine Websocket-Verbindung mithilfe der Library socket.io verwendet. Next.js bietet hier keine eigene Lösung für Websocket Verbindungen. Beim Aufbau einer Terminal Session wird im QuickStack Backend die Websocket Verbindung aufgebaut und anschliessend überprüft, ob der angefragte Pod läuft. Falls dieser noch nicht online ist, wird maximal 600 Sekunden gewartet. Sobald der angefragte Pod online ist, wird die Verbindung zwischen dem

Backend und dem Pod aufgebaut und an das Frontend weitergeleitet. Der QuickStack Benutzer kann zwischen den Shells bash und sh auswählen, sofern diese im laufenden Container zur Verfügung stehen.

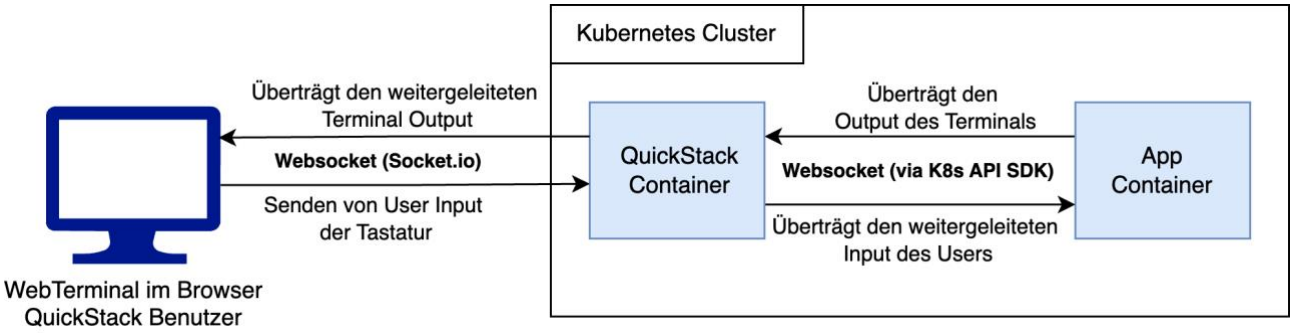


Abbildung 15: Diagramm Web-Terminal

Auf der Weboberfläche wird die Library XTerm verwendet, welche ein vollständiges Web-Terminal in die das QuickStack UI integriert. Im Browser sieht das wie folgt aus.

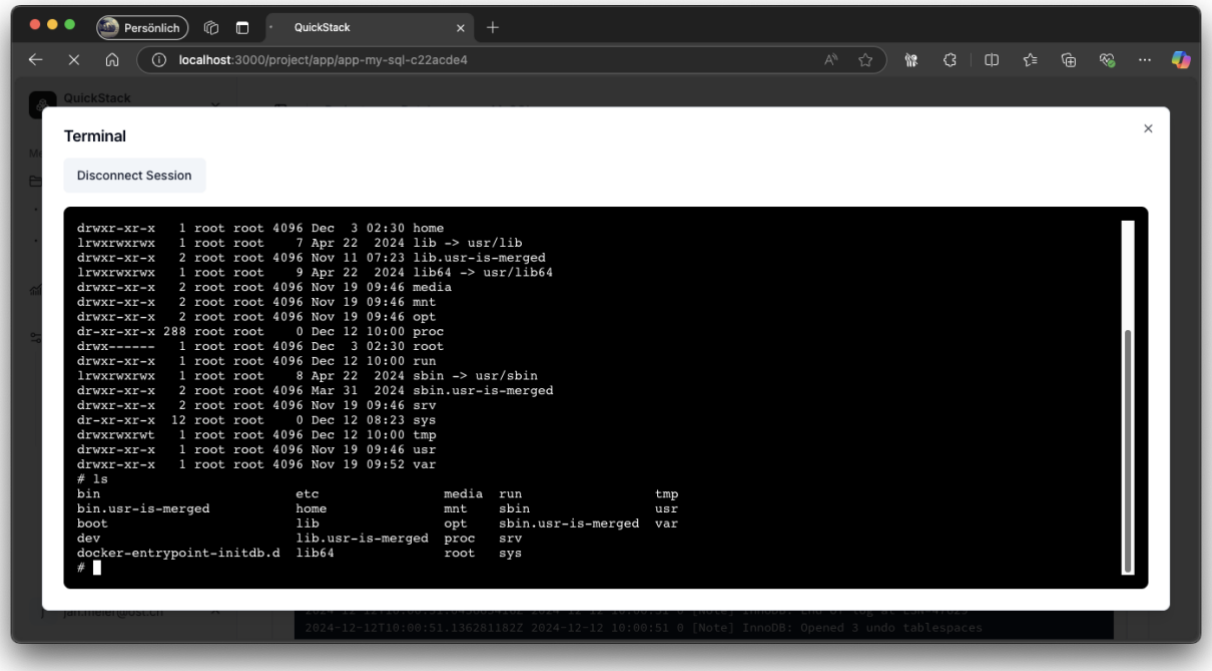


Abbildung 16: Screenshot Web-Terminal in QuickStack

4.2.5 Monitoring

Unter dem Menu «Monitoring» befinden sich die Verbrauchs- und Kapazitätswerte von CPU, RAM und Disk der verschiedenen Nodes, die zum Cluster gehören.

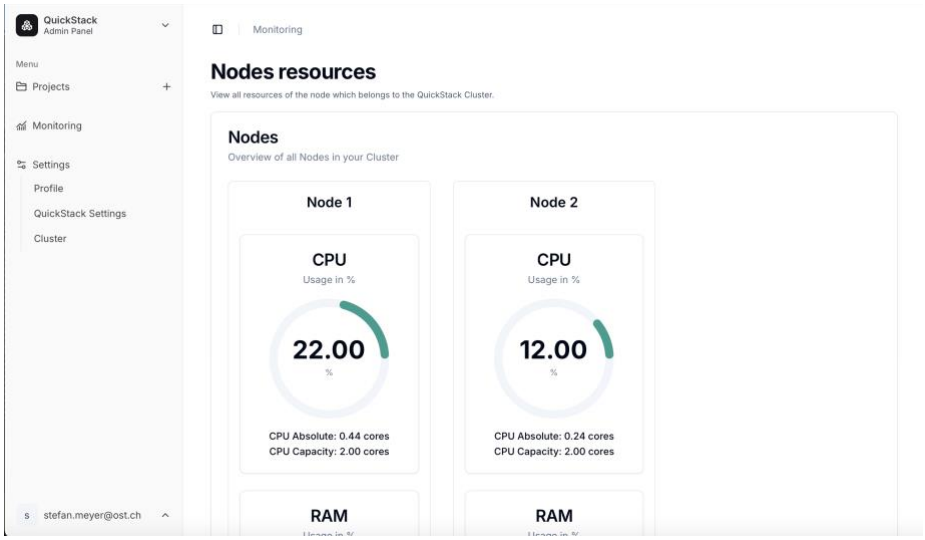


Abbildung 17: Screenshot Node Monitoring in QuickStack

Um die relevanten Werte für CPU und RAM zu erhalten, wurde auf die Kubernetes API zurückgegriffen, welche im Zusammenhang mit der Funktion «topNodes» die benötigten Informationen zurückgibt. Bei der Disk konnte nicht dieselbe API verwendet werden, da die zurückgegebenen Daten der Kubernetes SDK nicht der tatsächlichen Nutzung entsprechen. Aus diesem Grund musste hier auf die Longhorn API zurückgegriffen werden, die pro Node die benötigten Werte liefert. Da die verschiedenen Werte im JSON-Format zurückkommen, müssen die gewünschten Daten zuerst extrahiert werden, bevor sie weiterverarbeiten werden können. Zudem müssen diverse Umrechnungen vorgenommen werden, damit beispielsweise der RAM nicht in Bytes, sondern in Megabytes angezeigt wird.

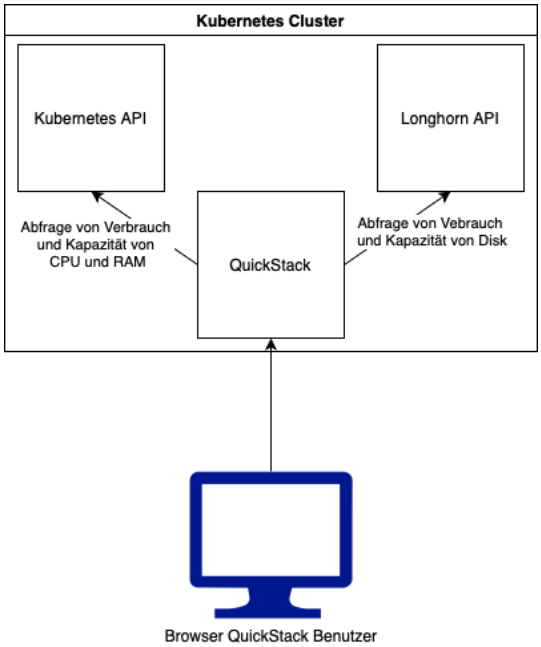


Abbildung 18: Diagramm Node Monitoring

Nachfolgend ist die entsprechende Schlüsselstelle im Code ersichtlich, bei welcher die Informationen in das jeweilige Modell geschrieben werden.

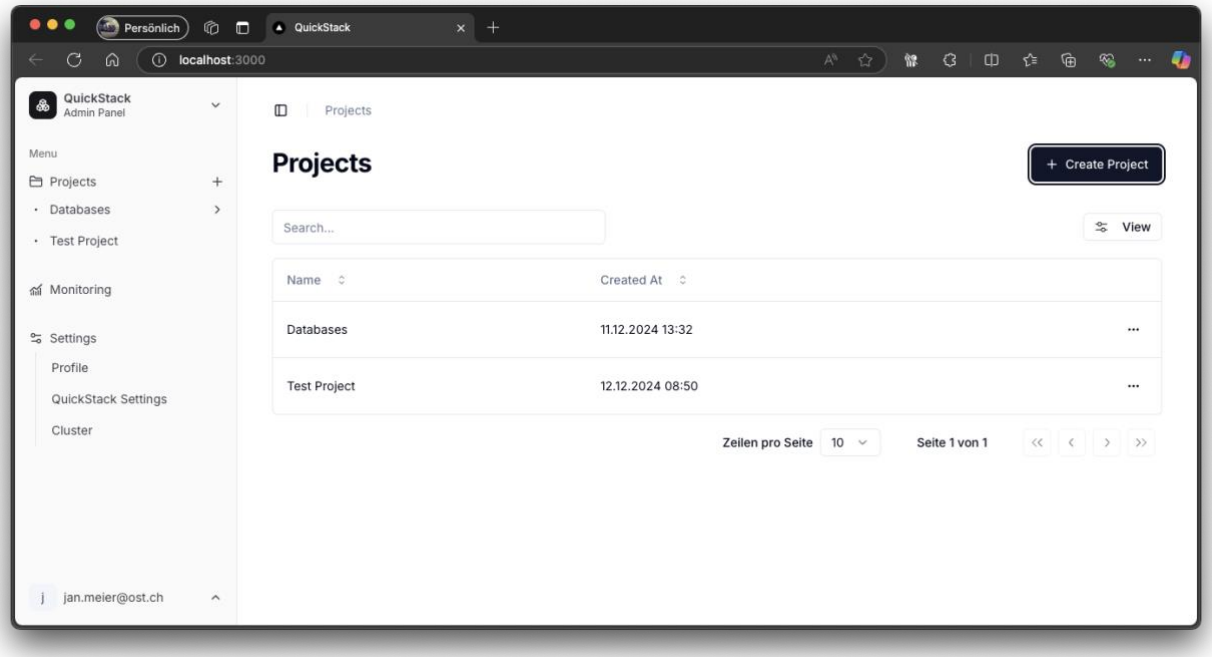
```
async getNodeResourceUsage(): Promise<NodeResourceModel[]> {
    const topNodes = await k8s.topNodes(k3s.core);

    return await Promise.all (topNodes.map(async (node) => {
        const diskInfo = await longhornApiAdapter.getNodeStorageInfo(node.Node.metadata?.name!);
        return {
            name: node.Node.metadata?.name!,
            cpuUsageAbsolut: Number(node.CPU?.RequestTotal!),
            cpuUsageCapacity: Number(node.CPU?.Capacity!),
            ramUsageAbsolut: Number(node.Memory?.RequestTotal!),
            ramUsageCapacity: Number(node.Memory?.Capacity!),
            diskUsageAbsolut: diskInfo.totalStorageMaximum - diskInfo.totalStorageAvailable,
            diskUsageCapacity: diskInfo.totalStorageMaximum,
        }}));
    }
```

Abbildung 19: Code Schlüsselstelle Node Monitoring

4.2.6 Caching der DB-Daten in Next.js

Durch die Verwendung von Next.js können integrierte Caching Features genutzt werden[114]. Mithilfe des «Unstable-Cache» müssen Daten nach dem Speichern nicht explizit nachgeladen werden. Stattdessen werden Sie direkt auf der bereits geladenen Seite aktualisiert sobald die damit verbundenen gecachten Daten revalidiert werden. Um das vereinfacht darzustellen, hier anhand eines Beispiels der Projektübersichtsseite.



[114] https://Next.js.org/docs/app/building-your-application/data-fetching/fetching

Abbildung 20: Screenshot Projektübersicht in QuickStack

Alle in der Tabelle angezeigten Daten auf der Projektübersichtsseite werden einmal von der Datenbank geladen und danach auf unbestimmte Zeit gecached. Dies geschieht, indem die Datenbankabfrage in die Wrapper-Methode «unstable_chache» gesetzt wird. Als zusätzlich Information wird ein Tag angegeben, wodurch die gecachten Daten identifiziert werden können (im Bild der Rückgabewert aus der statischen Methode «Tags.projects()»). Es können mehrere gecachte Daten mit demselben Tag versehen werden.

```
async getAllProjects() {
    return await unstable_cache(() =>  dataAccess.client.project.findMany({
        include: {
            apps: true
        },
        orderBy: {
            name: 'asc'
        }
    }),
        [Tags.projects()], {
        tags: [Tags.projects()]
    })();
}
```

Abbildung 21: Code «unstable_chache» Datenbankabfrage

Sobald ein Benutzer Infos über ein Projekt speichert, wird im Hintergrund der Projekt-Chache revalidiert. Dies geschieht durch die Funktion «revalidateTag(Tags.projects());» wodurch alle Daten welche in der Projektübersicht geladen und mit dem Tag Tags.projects() gecachet wurden automatisch neu aus der Datenbank geladen und ans Frontend übermittelt werden. Das Ganze benötigt keine weitere Code Logik im Frontend für das Nachladen der Daten.

```
async save(item: Prisma.ProjectUncheckedCreateInput | Prisma.ProjectUncheckedUpdateInput) {
    let savedItem: Project;
    try {
        if (item.id) {
            savedItem = await dataAccess.client.project.update({
                where: {
                    id: item.id as string
                },
                data: item
            });
        } else {
            item.id = KubeObjectNameUtils.toProjectId(item.name as string);
            savedItem = await dataAccess.client.project.create({
                data: item as Prisma.ProjectUncheckedCreateInput
            });
        }
        await namespaceService.createNamespaceIfNotExists(savedItem.id);
    } finally {
        revalidateTag(Tags.projects());
    }
    return savedItem;
}
```

Abbildung 22: Code «unstable_chache» Tag revalidieren

4.2.7 Herunterladen von App-Volume Inhalt

Um bei einem Fehler einer App mit Persistent-Volume-Anbindung besser debuggen zu können, kann der Inhalt eines Volumes über QuickStack direkt heruntergeladen werden. Unter «Storage» ist ein Download-icon bei den Aktionen ersichtlich, wodurch der Download gestartet wird.

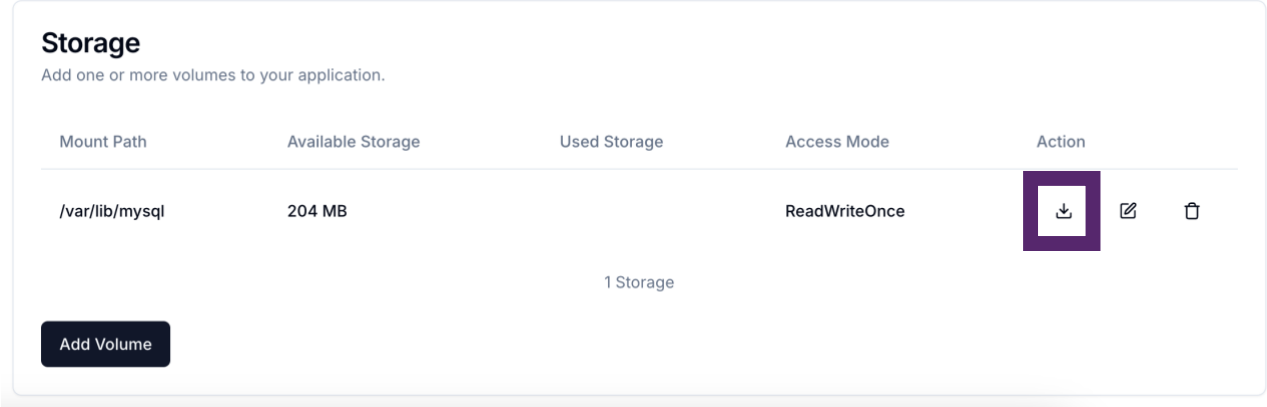


Abbildung 23: Screenshot App-Volume herunterladen in QuickStack

Im Hintergrund wird ein tar Command im Pod ausgeführt, welcher dann als Antwort einen Stream zwischen dem Pod und dem Backend öffnet und die gezippte Datei im QuickStack Backend in einem temporären Verzeichnis ablegt. Sobald die Übertragung des Zip abgeschlossen ist, wird dies ans Frontend zurückgemeldet und die Datei wird vom QuickStack Backend auf den PC des Benutzers heruntergeladen. Der gesamte Zip- und Downloadprozess wird in der nachfolgenden Grafik anhand eines Sequenzdiagramms genauer erläutert. Die einzelnen Sequenzen stellen das Frontend, Klassen innerhalb des Backends (blau) oder den Ziel-Pod dar.

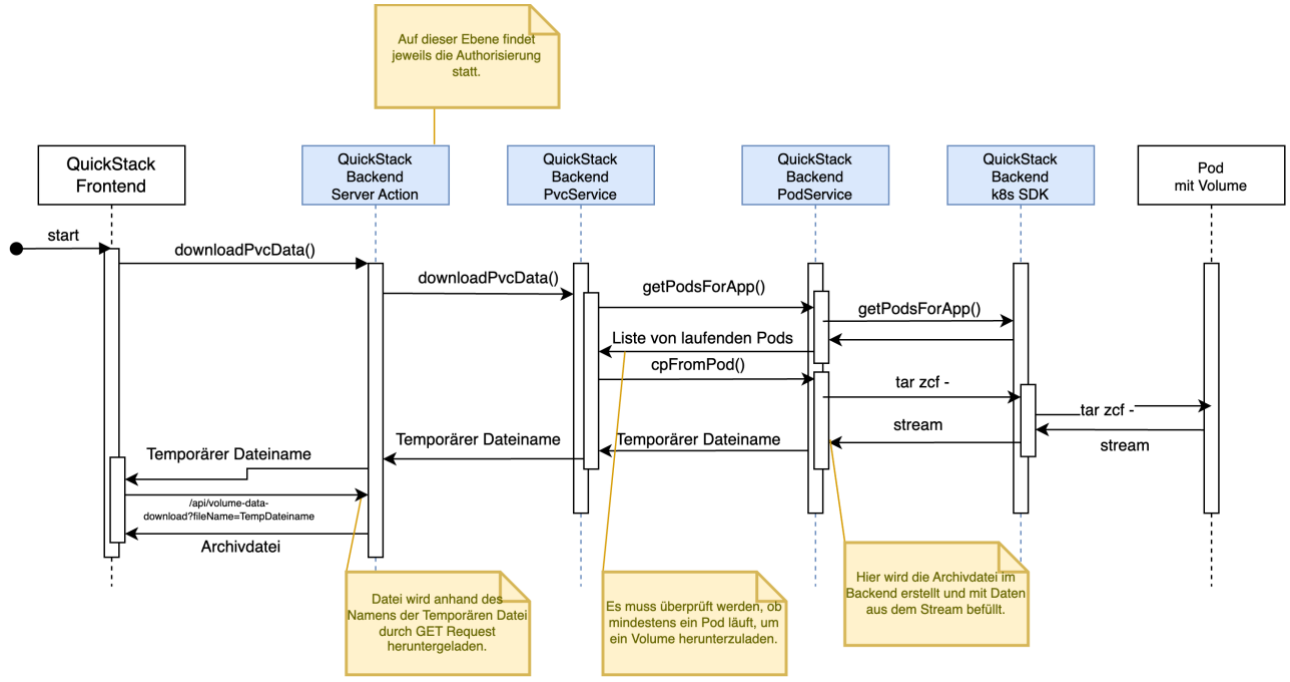


Abbildung 24: Sequenzdiagramm PVC Download

4.3 Testing

In dieser Arbeit werden drei verschiedene Testmethoden verwendet: Manuelle Tests, Automatisierte Unit Tests und Usability Test. Die entsprechenden Abläufe und Resultate werden in diesem Abschnitt beschrieben.

4.3.1 Manuelle Tests

Um QuickStack zu testen, wurden hauptsächlich Manuelle Tests durchgeführt, da das Setup für automatisierte Integrationstests im vorgegebenen Zeitrahmen zu umfangreich wäre. Für jeden Test besteht bereits ein Setup von einer aufgesetzten QuickStack Umgebung mit zwei VPS-Nodes und einer zugewiesenen Wildcard Domain (z.B. test.sa.biersoeckli.ch).

4.3.1.1 Testfälle

Nachfolgend sind alle Testfälle aufgelistet mit jeweiliger Beschreibung, den auszuführenden Schritten und dem erwarteten Resultat. Bei einem Testdurchlauf muss die Reihenfolge eingehalten werden.

Nr.	Beschreibung	Schritte	Erwartetes Resultat
1	Erfolgreiches Erstellen eines neuen Projekts.	1. Navigiere zur Projektübersicht. 2. Klicke auf «Create Project». 3. Gib einen Projektnamen «Testprojekt» ein. 4. Klicke auf «Create Project».	Das Projekt wird erfolgreich erstellt und in der Tabelle angezeigt. Eine Erfolgsmeldung wird eingeblendet.
2	Erfolgreiches Löschen eines bestehenden Projekts.	1. Navigiere zur Projektübersicht. 2. Klicke auf das Menü-Icon (drei Punkte) des zu löschenden Projekts. 3. Wähle «Delete Project». 4. Bestätige den Löschvorgang im Dialog.	Das Projekt wird erfolgreich gelöscht und aus der Tabelle entfernt. Eine Erfolgsmeldung wird eingeblendet.
3	Erfolgreiches Erstellen einer neuen App innerhalb eines Projekts.	1. Navigiere zur App-Übersicht eines Projekts. 2. Klicke auf «Create App». 3. Gib einen App-Namen ein (z.B. «Test App»). 4. Klicke auf «Create App».	Die App wird erfolgreich erstellt. Die Detailansicht der App wird geöffnet. Eine Erfolgsmeldung wird eingeblendet.
4	Erfolgreiches Löschen einer bestehenden App.	1. Navigiere zur App-Übersicht eines Projekts. 2. Klicke auf das Menü-Icon (drei Punkte) der zuvor erstellten App. 3. Wähle «Delete App». 4. Bestätige den Löschvorgang im Dialog.	Die App wird erfolgreich gelöscht und aus der Tabelle entfernt. Eine Erfolgsmeldung wird eingeblendet.
5	Erfolgreiches Bereitstellen einer App aus einem Git-Repository.	1. Erstelle eine neue App und wähle «Git» als Source Type. 2. Gib folgende Anmeldeinformationen ein: a. Git Repo URL: https://gitlab.ost.ch/sa-meier-meyer/test-frontend-app.git b. Git Username: jan.meier c. Git Token: sUkpzVWhA92Jzt7NLd4o d. Git Branch: main e. Path to Dockerfile: ./Dockerfile 3. Klicke auf «Save». 4. Klicke auf «Deploy».	Die App wird gebaut und bereitgestellt. Der Deployment-Status ändert sich entsprechend. In Lasche «Overview»: Im Deployment Log sollten keine Fehlermeldungen auftauchen. Die Logs sollten angezeigt werden.
6	Logs einer App ansehen	1. Navigiere zum Tab «Overview» der App. 2. Wähle im Log-Bereich einen Pod aus.	Die Logs des ausgewählten Pods werden im Log-Bereich angezeigt. Der Logstream verbindet sich und zeigt neue Logs an (inkl. Zeitstempel)
7	Erfolgreiches Bereitstellen einer	1. Erstelle eine neue App und wähle "Docker Container" als Source Type.	In Lasche «Overview»: Der Deployment-Status

	App aus einem Container-Image.	2. Gib einen gültigen Container-Image-Namen "nginx:latest" ein. 3. Klicke auf "Save". 4. Klicke auf "Deploy".	ändert sich auf «Succeded». Im Deployment Log sollten keine Fehlermeldungen auftauchen. Die Logs sollten angezeigt werden.
8	Erfolgreiches Hinzufügen einer Domain zu einer App.	1. Navigiere zum Tab "Domains" der App. 2. Klicke auf "Add Domain". 3. Gib einen Hostnamen «nginx.test.sa.biersoeckli.ch», Port «80» und die SSL-Einstellungen ein. 4. Klicke auf "Save".	Die Domain wird der App hinzugefügt und in der Tabelle angezeigt. Eine Erfolgsmeldung wird eingeblendet. Die Domain ist nach einem Deploy erreichbar.
9	Erfolgreiches Hinzufügen eines Volumes zu einer App.	1. Navigiere zum Tab "Storage" der App. 2. Klicke auf "Add Volume". 3. Gib den Mount-Pfad /tmp und eine 100 als Grösse ein. 4. Klicke auf "Save". 5. Klicke auf Deploy	Das Volume wird der App hinzugefügt und in der Tabelle angezeigt. Eine Erfolgsmeldung wird eingeblendet. Nach einem erneuten Deploy wird das Volume erstellt.
10	Erfolgreiches Setzen von Umgebungsvariablen für eine App.	1. Navigiere zum Tab "Environment" der App. 2. Gib folgendes in der Textarea ein: KEY=VALUE 3. Klicke auf "Save".	Die Umgebungsvariablen werden gespeichert. Eine Erfolgsmeldung wird eingeblendet. Nach einem erneuten Deploy sind die Umgebungsvariablen in der App verfügbar.
11	Erfolgreiches Skalieren einer App (Anzahl der Replicas ändern).	1. Navigiere zum Tab "General" → "Container Configuration" der App. 2. Ändere die Anzahl der Replicas auf 2. 3. Klicke auf "Save". 4. Klicke auf "Deploy".	Die App wird skaliert. Die Anzahl der Pods in der Kubernetes-Umgebung entspricht der eingestellten Anzahl von Replicas.
12	Erfolgreiches Setzen von CPU- und Memory-Limits für eine App.	1. Navigiere zum Tab "General" -> "Container Configuration" der App. 2. Setze CPU auf 100 und Memory Limit auf 100. 3. Klicke auf "Save". 4. Klicke auf "Deploy".	Die Ressourcenlimits werden in der Kubernetes-Umgebung angewendet.
13	Erfolgreiches Löschen einer Domain einer App.	1. Navigiere zum Tab "Domains" der App. 2. Klicke auf das Löschen-Icon der zuvor angelegten Domain. 3. Bestätige den Löschvorgang im Dialog. 4. Klicke auf "Deploy".	Die Domain wird aus der Datenbank und Kubernetes (Ingress) gelöscht und in der Tabelle nicht mehr angezeigt.
14	Erfolgreiches Löschen eines Volumes einer App.	1. Navigiere zum Tab "Storage" der App. 2. Klicke auf das Löschen-Icon des zuvor erstellen Volumes. 3. Bestätige den Löschvorgang im Dialog. 4. Klicke auf "Deploy".	Das Volume wird aus der Datenbank und Kubernetes gelöscht und in der Tabelle nicht mehr angezeigt.
15	Erfolgreiches Anzeigen der Logs eines Builds.	1. Navigiere zum Tab "Overview" -> "Deployments". 2. Wähle das letzte Deployment in der Tabelle aus, für welches build logs verfügbar sind.	Ein Dialogfenster öffnet sich und zeigt die Logs des Builds an.

		3. Klicke auf den Button "Show Logs".	
16	Erfolgreiches ändern des Benutzerpassworts.	1. Navigiere zu "Settings" -> "Profile". 2. Gib das aktuelle Passwort, das neue Passwort «12345678» und die Passwortbestätigung «12345678» ein. 3. Klicke auf "Change Password". 4. Klicke oben rechts auf den Logout Button 5. Melde dich mit der zur Verfügung gestellten Email und dem neuen Passwort an.	Das Passwort wird erfolgreich geändert. Das erneute anmelden mit dem neuen Passwort funktioniert.

Abbildung 25: Testfälle

4.3.1.2 Testresultate manuelle Tests

Nachfolgend die Testresultate aller manuellen Tests nach dem ersten Durchlauf.

Nr.	Resultat	Erwartetes Resultat	Kommentar
1	OK	Das Projekt wird erfolgreich erstellt und in der Tabelle angezeigt. Eine Erfolgsmeldung wird eingeblendet.	-
2	OK	Das Projekt wird erfolgreich gelöscht und aus der Tabelle entfernt. Eine Erfolgsmeldung wird eingeblendet.	-
3	OK	Die App wird erfolgreich erstellt. Die Detailansicht der App wird geöffnet. Eine Erfolgsmeldung wird eingeblendet.	-
4	OK	Die App wird erfolgreich gelöscht und aus der Tabelle entfernt. Eine Erfolgsmeldung wird eingeblendet.	-
5	OK	Die App wird gebaut und bereitgestellt. Der Deployment-Status ändert sich entsprechend. In Lasche «Overview»: Im Deployment Log sollten keine Fehlermeldungen auftauchen. Die Logs sollten angezeigt werden.	-
6	OK	Die Logs des ausgewählten Pods werden im Log-Bereich angezeigt. Der Logstream verbindet sich und zeigt neue Logs an (inkl. Zeitstempel)	-
7	OK	In Lasche «Overview»: Der Deployment-Status ändert sich auf «Succeded». Im Deployment Log sollten keine Fehlermeldungen auftauchen. Die Logs sollten angezeigt werden.	-
8	OK	Die Domain wird der App hinzugefügt und in der Tabelle angezeigt. Eine Erfolgsmeldung wird eingeblendet. Die Domain ist nach einem Deploy erreichbar.	-
9	OK	Das Volume wird der App hinzugefügt und in der Tabelle angezeigt. Eine Erfolgsmeldung wird eingeblendet. Nach einem erneuten Deploy wird das Volume erstellt.	-
10	OK	Die Umgebungsvariablen werden gespeichert. Eine Erfolgsmeldung wird eingeblendet. Nach einem erneuten Deploy sind die Umgebungsvariablen in der App verfügbar.	-
11	OK	Die App wird skaliert. Die Anzahl der Pods in der Kubernetes-Umgebung entspricht der eingestellten Anzahl von Replicas.	-
12	OK	Die Ressourcenlimits werden in der Kubernetes-Umgebung angewendet.	-
13	OK	Die Domain wird aus der Datenbank und Kubernetes (Ingress) gelöscht und in der Tabelle nicht mehr angezeigt.	-
14	OK	Das Volume wird aus der Datenbank und Kubernetes gelöscht und in der Tabelle nicht mehr angezeigt.	-
15	OK	Ein Dialogfenster öffnet sich und zeigt die Logs des Builds an.	-

16	OK	Das Passwort wird erfolgreich geändert. Das erneute anmelden mit dem neuen Passwort funktioniert.	-

Abbildung 26: Testresultate manuelle Tests

4.3.2 Automatisierte Tests

Die Automatisierten Tests in QuickStack beschränken sich auf Util-Klassen, welche Berechnungen oder String-Umformungen vornehmen. Es wurde bewusst auf Integrationstests zusammen mit der Kubernetes API und anderen automatisierten End-To-End Tests verzichtet, da diese den vorgegebenen Zeitrahmen sprengen würden. In der nachfolgenden Grafik ist die Testabdeckung der getesteten Files ersichtlich.

```
-----------------------------|---------|----------|---------|---------|-------------------
File                         | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-----------------------------|---------|----------|---------|---------|-------------------
All files                    |   95.95 |    93.79 |   97.91 |   95.95 |
 frontend/utils              |   89.41 |    86.36 |   88.88 |   89.41 |
  form.utilts.ts             |   78.12 |       50 |      50 |   78.12 | 19-20,27-31
  format.utils.ts            |     100 |      100 |     100 |     100 |
  toast.utils.ts             |   93.33 |    81.81 |     100 |   93.33 | 25-26
 server/utils                |   95.65 |    90.69 |     100 |   95.65 |
  fs.utils.ts                |   87.87 |    71.42 |     100 |   87.87 | 18-19,27-28,44-45,57-58
  kube-object-name.utils.ts  |     100 |      100 |     100 |     100 |
  memory-caluclation.utils.ts|     100 |      100 |     100 |     100 |
  path.utils.ts              |     100 |      100 |     100 |     100 |
 shared/utils                |   99.43 |    98.43 |     100 |   99.43 |
  list.utils.ts              |     100 |      100 |     100 |     100 |
  stream.utils.ts            |     100 |      100 |     100 |     100 |
  zod.utils.ts               |   99.06 |    97.77 |     100 |   99.06 | 106
-----------------------------|---------|----------|---------|---------|-------------------

Test Suites: 10 passed, 10 total
Tests:       91 passed, 91 total
Snapshots:   0 total
Time:        3.141 s
```

Abbildung 27: Screenshot Code Coverage

4.3.3 Usability Test

Mit dem Usability Test soll aufgezeigt werden, ob QuickStack von den Benutzern so benutzt wird, wie das anfänglich definiert wurde. Durch solche Tests werden allfällige Schwächen und Unklarheiten in der Webapplikation offengelegt und sind deshalb sehr wichtig, da direktes Feedback von den Benutzern erhalten werden kann.

Der Aufbau des Usability Tests ist so gestaltet, dass den Benutzern ein Skript vorgelegt wird, welches verschiedene Aufgaben enthält, die in QuickStack umgesetzt werden müssen. Hierbei wird das Ziel vorgegeben, welches erreicht werden soll. Es soll keine «Schritt für Schritt» Anleitung sein, jedoch die benötigten Informationen enthalten (z.B. Source Informationen eines Git Repositories) um voranzukommen. Die Benutzer werden während des Tests von den Entwicklern beobachtet, die sich dadurch Notizen über mögliche Verbesserungen machen können. Am Ende werden die User noch kurz persönlich über die Usability von QuickStack befragt. Jedem Benutzer wird eine eigene Umgebung vorgegeben mit zwei VPS-Nodes, deren IP-Adressen und zwei Domains, welche bereits mit den IP-Adressen konfiguriert wurden.

4.3.3.1 QuickStack aufsetzen

Nr.	Aufgabe	Bemerkungen

1	• Im ersten Schritt soll QuickStack auf einem neuen Ubuntu Server installiert werden. Es sollte ein Server zur Verfügung gestellt worden sein.
	• Login auf den Ubuntu Server via SSH
	  ○ IP Adresse: (vorläufig zur Verfügung gestellt)
	  ○ Username azureadmin
	  ○ Passwort: 9Fi0Vpihg1kLQq96NB52asdliiskdafjz6i71pG9cRWhiwqIkG86Wpw3tyxcvvthqj8
	• Für die Installation von QuickStack mit dem Server via SSH verbinden und folgenden Command eingeben:
	curl -sfL https://get.quickstack.dev/setup.sh | sh -
	• Die Installation könnte ein paar Minuten dauern	

2	• Sobald die Installation abgeschlossen ist, im Browser die folgende URL öffnen: http://IP-Adresse:30000/auth
	Bei der URL muss IP-Adresse mit der IP-Adresse des Servers ersetzt werden.
	• Sobald sich die Registrier-Seite öffnet, muss eine E-Mail und ein Passwort angegeben werden. Folgende Daten sollen gewählt werden:
	  ○ Benutzername: test@ost.ch
	  ○ Passwort: weLoveOst69
	• Loge dich mit dem neu erstellen User ein.	

3	• Überprüfe in den Einstellungen, ob die OST-Mailadresse für die SSL-Zertifikate verwendet wird. Falls nicht, rechts ankreuzen.	
4	• Kontrolliere in den Einstellungen, wie es um die Ressourcen (CPU, RAM, Disk) von der Master Node steht. Hat der Server genügend Ressourcen / wird eine Meldung angezeigt. Falls nicht, rechts ankreuzen.	

Tabelle 16: Usability Test QuickStack aufsetzen

4.3.3.2 App aus Repository deployen

Nr.	Aufgabe	Bemerkungen
1	• Erstelle ein Projekt mit dem Namen «Frontend».	
2	• Erstelle nun in diesem Projekt eine weitere App mit dem Namen «KI».	

3	• Als Source verwenden wir ein Git-Repository mit folgenden Daten:
	  ○ Git Repo URL: https://gitlab.ost.ch/sa-meier-meyer/test-frontend-app.git
	  ○ Git Username: jan.meier
	  ○ Git Token: sUkpzVWhA92Jzt7NLd4o
	  ○ Git Branch: main
	  ○ Path to Dockerfile: ./Dockerfile
	• Die App soll folgende Ressourcen-Eigenschaften haben:
	  ○ Replica Count: 1
	  ○ Memory Limit (MB): 100
	  ○ Memory Reservation (MB): 50
	  ○ CPU Limit (m): 100
	  ○ CPU Reservation (m): 50
	• Füge die Domain 1 hinzu, die auf Port 3000 weiterleitet. Aktiviere zusätzlich die Optionen «use HTTPS». (Die Domain wurde schon korrekt auf dem DNS-Server hinterlegt)	

4	• Deploy nun die App und begutachte die Logs.	

	• Bei den Deployments kannst du erkennen, wann die App ready ist. • Überprüfe, ob du via über die Domain 1 von Browser aus auf die Applikation zugreifen kannst.	
5	• Lösche nun das erstellte Projekt «Frontend».	

Tabelle 17: Usability Test App aus Repository deployen

4.3.3.3 Wordpress App deployen

Nr.	Aufgabe	Bemerkungen
1	• Unter «Projects» erstellen wir nun eine kleine Wordpress-Umgebung. • Erstelle ein neues Projekt mit dem Namen «Wordpress».	
	• Innerhalb von diesem Projekt erstellen wir zuerst eine App mit dem Namen «Wordpress DB»	
2	• Als Source verwenden wir einen Docker Container von Dockerhub mit dem Imagename «mariadb:latest». • Folgende Environment Variablen werden verwendet: MYSQL_ROOT_PASSWORD=mariadb MYSQL_USER=mariadb MYSQL_PASSWORD=mariadb MYSQL_DATABASE=defaultdb • Zusätzlich wird noch ein Volume mit dem Mount Path «/var/lib/mysql», der Grösse 500 MB und dem Access Mode «ReadWriteMany» hinzugefügt. • Konfigurieren Sie den Internal-Port 3306, damit über diesen Port die Datenbank erreicht werden kann	
3	• Leider haben wir beim Volume einen Fehler gemacht. Es handelt sich um den Access Mode «ReadWriteOnce». Bitte korrigiere diesen Fehler.	
4	• Deploy nun die App und begutachte die Logs. • Bei den Deployments kannst du erkennen, wann die App ready ist. • Kopiere unter der Lasche «Domains» den internen Hostname in die Zwischenablage, da dieser danach im Wordpress Frontend gebraucht wird.	
5	• Erstelle nun im gleichen Projekt eine weitere App mit dem Namen «Wordpress Backend».	
6	• Als Source verwenden wir einen Docker Container von Dockerhub mit dem Imagename «wordpress:latest». • Erhöhe die Anzahl Replikation auf 2. • Folgende Environment Variablen werden verwendet (ACHTUNG: Ersetze den Wordpress_DB_Host mit dem kopierten Hostnamen der letzten Aufgabe): WORDPRESS_DB_HOST=*INTERNEN HOSTNAME VON DATABASE EINFÜGEN*:3306 WORDPRESS_DB_USER=mariadb WORDPRESS_DB_PASSWORD=mariadb WORDPRESS_DB_NAME=defaultdb WORDPRESS_TABLE_PREFIX=wp_ WORDPRESS_HOSTNAME=wordpress.sa.biersoeckli.ch WORDPRESS_PROTOCOL=https • Zusätzlich wird noch ein Volume mit dem Mount Path «/var/www/html», der Grösse 100 MB und dem Access Mode «ReadWriteMany» hinzugefügt. (ReadWriteMany, weil zwei Replikas gleichzeitig auf das Volume zugreifen) • Füge die Domain «Domain 2» hinzu, die auf Port 80 weiterleitet. Aktiviere zusätzlich die Optionen «use HTTPS» und «Redirect http to HTTPS». (Die Domain wurde schon korrekt auf dem DNS-Server hinterlegt)	

7	• Deploy nun die App und begutachte die Logs. • Bei den Deployments kannst du erkennen, wann die App ready ist. • Überprüfe, ob du via Domainname (Domain 2) von aussen auf die Wordpress Applikation zugreifen kannst (Es sollte ein Setup Screen angezeigt werden). Möglicherweise dauert es ein paar sekunden, bis das Wordpress App vollständig startet.	
8	• Lösche nun das erstellte Projekt «Wordpress».	
9	• Logge dich aus.	

Tabelle 18: Usability Test Wordpress App deployen

4.3.3.4 Rückmeldungen Usability Tests

Insgesamt wurden die Usability Tests mit fünf Personen durchgeführt. Vier von fünf Personen konnten alle Aufgaben ohne Probleme und nachfragen lösen. Eine Person hatte ein Problem mit dem deployment von Wordpress, jedoch hat hier der Wordpress Container intern ein Problem das durch neustarten des Containers gelöst werden konnte. Die Berichte der einzelnen Usability Tests befinden sich im Anhang.

Folgende Verbesserungsvorschläge wurden durch die Usability Tests ermittelt und verbessert:

- Sidenav muss geöffent sein nach erstem Login
- Feld «Docker Container Name» zu «Docker Image Name» umbenennen
- «Deactivate Node» darf nicht möglich sein auf Master Node
- Wenn eine Domain, ein Volume oder Internal Port zweimal editiert wird, wird jeweils immer der alte wert angezeigt.
- Monitoring: Cores mit 2 kommastellen anzeigen und bei hover über die Zahl wird in Tooltipp die volle Zahl anzeigen.

Die nachfolgenden Punkte wurden aufgenommen, konnten jedoch nicht mehr im Rahmen der SA umgesetzt werden:

- Knopf "View Deployment" in «success-toast» anzeigen, sobald "deploy" gedrückt wurde --> Absprung zu laufendem Deployment
- Ctrl+S als Shortcut unterstützen
- Warnung anzeigen, wenn Eingaben in Felder gemacht wurden und dann auf einen anderen Tab geklickt wird, ohne zu speichern.
- Klarer anzeigen, welche Felder optional sind und welche nicht.

5. Ergebnisdiskussion mit Ausblick

In der nachfolgenden Kapitelstruktur wird zuerst auf die Ergebnisse der Studienarbeit eingegangen. Hierbei wird der konkrete Bezug zu den Anforderungen (FA und NFA) hergestellt und auf dieser Basis analysiert, was erreicht bzw. was nicht erreicht wurde. Im zweiten Kapitel «Schlussfolgerung und Ausblick» werden die Ergebnisse reflektiert und bewertet. Dies geschieht anhand eines Abgleichs mit der Aufgabenstellung sowie mit den Zielen der Arbeit. Ganz am Schluss erfolgt noch ein Blick auf mögliche Weiterentwicklungen bzw. allfällige Verbesserungen der Arbeit.

5.1 Ergebnisse

Sowohl die funktionalen als auch die nicht-funktionalen Anforderungen wurden zu einem grossen Teil erfolgreich umgesetzt, während nur in einigen wenigen Bereichen Abstriche gemacht werden mussten.

Die Kernfunktionen des Systems, wie Benutzerverwaltung, App-Deployment sowie Logging und Monitoring sind weitestgehend realisiert. Benutzer können sich registrieren, ihre Daten einsehen und ändern. Im Bereich App-Deployment sind fast alle Anforderungen erfüllt: Benutzer können Namespaces anlegen, Ports und Umgebungsvariablen konfigurieren, Ressourcenlimits setzen und Replicas verwalten. Das Dashboard

ermöglicht eine übersichtliche Darstellung der Apps und bietet Zugriff auf Service- und Deployment-Logs. Die Anzeige des Ressourcenverbrauchs pro Deployment sowie die Integration von persistentem Speicher wurden ebenfalls erfolgreich umgesetzt.

Die nicht-funktionalen Anforderungen wurden ebenfalls erfüllt. Die Benutzbarkeit wurde durch Benutzertests validiert, die Ladezeiten des Systems bleiben unter der vorgegebenen Schwelle und die gleichzeitige Verwaltung von mindestens zehn Apps funktioniert zuverlässig. Die Installation von QuickStack erfolgte in der vorgegebenen Zeit und mit der maximal erlaubten Anzahl von Befehlen.

Eine Ausnahme bildet FA-16 «Vorlagen für verschiedene Datenbanken (z.B. PostgreSQL) müssen bereitgestellt und mit zwei Klicks deployed werden können.». Dieser Punkt wurde nicht umgesetzt und ist deshalb nicht erfüllt.

Die optionalen Funktionen (FA-21 bis FA-24) wurden ebenfalls nicht realisiert. Hierzu zählen der direkte Zugriff auf das Dateisystem der Apps, die Erstellung von Backups der Longhorn-Volumes und die automatische Security Härtung der Cluster-Nodes.

5.2 Schlussfolgerung und Ausblick

Wie im vorherigen Kapitel beschrieben, konnten fast alle funktionalen und nicht-funktionalen Anforderungen erfolgreich erreicht und umgesetzt werden. FA-16 wurde nicht umgesetzt, da entschieden wurde, die Zeitressourcen für andere «wichtigere» Funktionen zu verwenden, die sich im Laufe der Studienarbeit herauskristallisiert haben. Eine dieser wichtigen Funktionen ist das Terminal Interface, welches es dem Benutzer ermöglicht sich direkt mit dem Terminal einer laufenden Anwendung zu verbinden. Auch die drei optionalen Funktionen wurden nicht umgesetzt. Dies ist jedoch nicht schlimm, da der Fokus bewusst auf der sauberen und vollständigen Umsetzung der priorisierten Kernfunktionen lag. In diesem Zusammenhang ist auch zu erwähnen, dass es dafür möglich ist, den Inhalt eines gemountetes Volume einer App direkt herunterzuladen.

Betrachtet man die «Aufgabenstellung» und das «Ziel der Arbeit», so kann mit Freude festgestellt werden, dass alle Punkte zur Zielerreichung umgesetzt wurden. So konnte ein neues Open Source[115] Produkt entwickelt werden, das den Softwareentwicklern ermöglicht, ein PaaS auf einer eigenen VPS-Infrastruktur zu betreiben. Im Gegensatz zu den bestehenden kommerziellen Cloud-PaaS wurde eine Lösung entwickelt, die einerseits kostenlos ist und andererseits nicht viele Serverressourcen verschlingt, was letztendlich den Geldbeutel der QuickStack Nutzer schont. Dies war ein grosses Ziel dieser Arbeit und konnte erfolgreich erreicht werden. Grosses Augenmerk wurde zudem auf die einfache und schnelle Einrichtung bzw. Bedienung gelegt. So können sich Softwareentwickler vollkommen auf die Entwicklung ihrer Applikation konzentrieren und müssen sich nicht um Serverkonfigurationen kümmern. Das Erreichen dieses Ziels konnte durch die Durchführung von Usability-Tests bestätigt werden. Darüber hinaus wurde viel Zeit investiert, um mögliche Schwierigkeiten beim Deployment der Apps frühzeitig zu erkennen. Dazu wurden verschiedene Logs und Monitoring Funktionalitäten integriert, damit Probleme schnell und effizient eingegrenzt und behoben werden können.

Natürlich gibt es auch bei diesem Produkt wünschenswerte Funktionen bzw. Verbesserungen, die bei einer Weiterentwicklung berücksichtigt werden sollten. Vor allem im Bereich Backup und Sicherheit kann noch viel umgesetzt werden. Hier sind zum Beispiel automatische Updates von k3s und Longhorn sowie ein CIS-Hardening der Cluster Nodes zu nennen. Des Weiteren wäre eine Backupfunktionalität der Longhorn Volumes wünschenswert, die z.B. mit einem S3 Bucket realisiert werden könnte. Auch auf Seiten der Usability gibt es noch Ideen, um die Benutzererfahrung weiter zu vereinfachen. Dazu gehört zum Beispiel das automatische Auslesen von Ports und Volumes aus den Dockerfiles. Dadurch muss der Benutzer nur noch die Daten überprüfen und keine Werte mehr manuell eingeben. Dies reduziert die Wahrscheinlichkeit von Fehleingaben. Auch eine Information über den Status eines angeforderten SSL-Zertifikats ist wünschenswert, damit der Benutzer weiss, ob das Deployment des Zertifikats korrekt funktioniert hat. Eine Komfortfunktion für den Benutzer ist schliesslich auch die Hinterlegung von vordefinierten App-Templates. Damit können häufig

[115] https://github.com/biersoeckli/QuickStack

benötigte Anwendungen, wie z.B. verschiedene Datenbanken-Typen, schnell und mit wenigen Klicks deployed werden.

6. Glossar

Fachbegriff	Umschreibung
API	Application Programming Interface. Eine Schnittstelle, die es verschiedenen Softwareanwendungen ermöglicht, miteinander zu kommunizieren.
Build Tool	Ein Werkzeug oder eine Suite von Werkzeugen, mit denen ein Dockerfile zu einem Docker Image übersetzt wird.
Cluster	Gruppe von Servern, die gemeinsam arbeiten und als Einheit verwaltet werden.
Cluster Node	Einzelner VPS innerhalb eines Clusters.
Container	Eine isolierte Umgebung zum Ausführen von Anwendungen, die auf Docker Images basieren.
Container Image	Datei, die alles enthält, was zum Ausführen eines Containers benötigt wird, einschliesslich Code, Bibliotheken und Konfigurationen.
Container Registry	Ein Speicherort für Container-Images. Sie ermöglicht es Entwicklern, ihre Container-Images zu verwalten, zu versionieren und für die Bereitstellung im Kubernetes Cluster verfügbar zu machen.
Container-Orchestrierung	Prozess, um den Betrieb, die Skalierung und das Management von Containeranwendungen zu automatisieren.
Continuous Deployment (CD)	Automatisierter Prozess, um getestete Software direkt in die Produktion zu bringen.
Continuous Integration (CI)	Entwicklungspraktik, bei der Quellcode regelmässig integriert und getestet wird.
CoreDNS	DNS-Server für Kubernetes, der die Namensauflösung innerhalb des Clusters ermöglicht.
Debugging	Prozess der Fehlerfindung und -behebung in einer Anwendung.
Deployment	Ein Kubernetes-Objekt, das das Deployment von Applikationen verwaltet, indem es Replica Sets kontrolliert und Updates für die Pods ermöglicht.
Docker	Eine Plattform zur Erstellung, Ausführung und Verwaltung von Containern.
Docker Image	Eine schreibgeschützte Vorlage, die ein ausführbares Anwendungspaket enthält, inklusive Code, Runtime, Systemtools und Bibliotheken.
Dockerfile	Textdatei, die die Anweisungen enthält, um ein Docker-Image zu erstellen.
FIO (Flexible I/O Tester)	Tool zur Messung der Leistung von Speichersystemen in Bezug auf IOPS, Durchsatz und Latenz.
Git	Versionskontrollsystem, das Änderungen an Dateien und Quellcode verwaltet und verfolgt.
Git Repository	Ein Speicherort für Softwareprojekte, der die Versionskontrolle mit Git ermöglicht.
High-Availability Cluster	Gruppe von Servern, die so konfiguriert sind, dass sie eine hohe Verfügbarkeit und Ausfallsicherheit gewährleisten.
IaaS	Infrastructure-as-a-Service. Ein Cloud-Service-Modell, das Rechen-, Speicher- und Netzwerkressourcen auf Abruf bereitstellt, wobei der Benutzer mehr Kontrolle über die Infrastruktur hat.

Ingress Controller	Eine Komponente in Kubernetes, die eingehenden Netzwerkverkehr an die entsprechenden Services und Pods weiterleitet.
IoT (Internet of Things)	Netzwerk verbundener Geräte, die Daten sammeln, austauschen und oft autonom reagieren können.
Live-Log-Streaming	Anzeige von Protokolldaten in Echtzeit, um Anwendungsprobleme zu diagnostizieren.
Load Balancer	Verteilt eingehenden Netzwerkverkehr auf mehrere Server, um die Last zu verteilen und Ausfälle zu vermeiden.
Monitoring	Überwachung der Systemleistung und -verfügbarkeit, oft durch Dashboards dargestellt.
NFS	Network File System. Ein Protokoll zum Zugriff auf Dateien über ein Netzwerk.
One-Click-Deployment	Funktion, mit der Anwendungen mit minimalem Aufwand bereitgestellt werden können.
Open Source	Software, deren Quellcode öffentlich zugänglich ist und von der Community verändert und weitergegeben werden kann.
ORM	Object-Relational Mapper. Ein Tool zum Abbilden von Objekten in Datenbanktabellen.
PaaS	Platform-as-a-Service. Ein Cloud-Service-Modell, das eine Plattform zum Entwickeln, Ausführen und Verwalten von Anwendungen bereitstellt, ohne dass der Nutzer sich um die zugrunde liegende Infrastruktur kümmern muss.
Persistent Volume Claim (PVC)	Anfrage von Benutzern in Kubernetes, um Speicher mit bestimmten Eigenschaften zu beanspruchen.
Pod	Die kleinste Einheit in Kubernetes, die einen oder mehrere Container beinhaltet.
qrcode	Eine JavaScript Library zur generierung von QR-Codes für die Einrichtung von 2FA.
ReadWriteMany (RWX)	Speicherzugriffstyp, bei dem mehrere Pods gleichzeitig auf dieselbe Speicherressource zugreifen können.
ReadWriteOnce (RWO)	Speicherzugriffstyp, bei dem nur ein Pod zu einer Zeit auf eine Speicherressource zugreifen kann.
ReplicaSet	Ein Kubernetes-Objekt, das die Anzahl der identischen Pods festlegt, welche zur Verfütung stehen soll. Es sorgt dafür, dass die gewünschte Anzahl von Pods ausgeführt wird.
Repository	Speicherort für Quellcode und Dateien, oft in Verbindung mit Git genutzt, z. B. GitHub oder GitLab.
Reverse Proxy	Server, der Anfragen von Clients an Backend-Server weiterleitet.
S3-kompatibler Speicher	Speicherlösung, die das Amazon S3-Protokoll verwendet, z. B. für Backups oder Objektspeicher.
SDK	Software Development Kit. Eine Sammlung von Tools, Bibliotheken und Dokumentationen, die Entwickler zum Erstellen von Anwendungen nutzen können.
Self-Hosted	Software, die auf der eigens verwalteten Infrastruktur betrieben wird.
Self-Hosted PaaS	Eine PaaS-Lösung, die auf eigener Infrastruktur (z.B. auf einem VPS) gehostet wird, im Gegensatz zu kommerziellen Cloud-PaaS-Diensten.
Shared Storage Provider	Ein System oder eine Dienstleistung, die Speicherressourcen bereitstellt, die von mehreren Containern oder Anwendungen gemeinsam genutzt werden können.
Snapshots	Momentaufnahme eines Systems oder Datenzustands, der zu einem späteren Zeitpunkt wiederhergestellt werden kann.
SSL-Zertifikat	Zertifikat, das verschlüsselte Verbindungen über HTTPS ermöglicht und die Identität einer Website bestätigt.
TypeScript	Eine superset of Javascript, welches statisch Typen hinzufügt.

Usability Testing	Testverfahren, um die Benutzerfreundlichkeit einer Software oder Anwendung zu bewerten.
Volume	Speicherbereich, der von Containern genutzt wird, um persistente Daten zu speichern.
VPS	Virtual Private Server. Ein virtueller Server, der auf einem physischen Server gehostet wird und eigene Ressourcen und Betriebssystem bietet.
Web-Terminal	Browserbasiertes Interface, das Zugriff auf die Kommandozeile eines Servers ermöglicht.

7. Abbildungsverzeichnis

8. Tabellenverzeichnis

Ostschweizer Fachhochschule, Standort Rapperswil