Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Studiengang Medieninformatik

Bachelorarbeit

von

Albert **Hahn**

Konzeption und Implementierung einer Microservice Architektur in einem hybriden kubernetes Cluster für industrielle KI-Anwendungsfälle

Conceptual Design and Implementation of a Microservice Architecture in a Hybrid Kubernetes Cluster for Industrial AI Use Cases

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Studiengang Medieninformatik

Bachelorarbeit

von

Albert **Hahn**

Konzeption und Implementierung einer Microservice Architektur in einem hybriden kubernetes Cluster für industrielle KI-Anwendungsfälle

Conceptual Design and Implementation of a Microservice Architecture in a Hybrid Kubernetes Cluster for Industrial AI Use Cases

Bearbeitungszeitraum: von 4. Oktober 2021
bis 3. März 2022

1. Prüfer: Prof. Dr.-Ing. Christoph Neumann
2. Prüfer: Prof. Dr. Dieter Meiller

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Ostbayerische Technische Hochschule
Amberg-Weiden

Bestätigung gemäß § 12 APO

Name und Vorname
der Studentin/des Studenten: **Hahn, Albert**

Studiengang: **Medieninformatik**

Ich bestätige, dass ich die Bachelorarbeit mit dem Titel:

Konzeption und Implementierung einer Microservice Architektur in einem hybriden kubernetes Cluster für industrielle KI-Anwendungsfälle

selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Datum: 20. Januar 2022

Unterschrift:

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Ostbayerische Technische Hochschule
Amberg-Weiden

Bachelorarbeit Zusammenfassung

Studentin/Student (Name, Vorname): **Hahn, Albert**
Studiengang: Medieninformatik
Aufgabensteller, Professor: Prof. Dr.-Ing. Christoph Neumann
Durchgeführt in (Firma/Behörde/Hochschule): Krones AG, Neutraubling
Betreuer in Firma/Behörde: Ottmar Amann
Ausgabedatum: 4. Oktober 2021 Abgabedatum: 3. März 2022

Titel:

Konzeption und Implementierung einer Microservice Architektur in einem hybriden kubernetes Cluster für industrielle KI-Anwendungsfälle

---

Zusammenfassung:

Das Ziel dieser Bachelorarbeit ist es, eine flexible und nahtlose Lösung für ein Hybrides Cluster aus on-premise Edge Devices und Cloud Ressourcen bereitzustellen. Produktionslinienanwendungen/Microservices sollen zukünftig beliebig skalierbar und agil sein, dabei sollen für die Anwendungen generell keine Differenzierung zwischen offline und online Ressource getroffen werden. Im Zuge dessen wird die Umsetzbarkeit und Relevanz von cloudbasierten Microservices im Bereich der künstlichen Intelligenz auf einer zukünftigen Produktionsanlage untersucht.

Schlüsselwörter:

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Krones AG bietet Anlagen für die Getränkeindustrie als auch Nahrungsmittelhersteller, von der Prozesstechnik bis hin zur IT-Lösung. Die Komplettlinie beeinhaltet auch das bereitstellen von Software auf den einzelnen Produktionsanlagen. Hierfür werden eine Vielzahl von Produktionslinienanwendungen auf den Anlagen installiert, gewartet und verwaltet. Ein riesiger Aufwand der Fehleranfälligkeiten wie fehlende Frameworks, Bibliotheken und anderer Abhängigkeiten mit sich bringt. Eigene Server müssen für die Kommunikation der Anlagen verbaut und gewartet werden, was zusätzlich Ressourcen beansprucht und automatisch die Kosten für die Inbetriebnahme einer solchen Linien erhöhen. Die Weiterentwicklung der zukünftigen Bereitstellung von Produktionsanlagensoftware erfolgt mithilfe eines Proof of Concept (PoC), welcher die Möglichkeiten einer wartungsfreien Infrastruktur durch ein continuous delivery System evaluiert. Dies verläuft in Zusammenarbeit mit dem Kooperationspartner und Softwarteunternehmen SUSE GmbH, welches das wartungsfreie Betriebssystem SUSE Linux Enterprise Micro und die multi-cluster Orchestrierungsplattform Rancher anbietet.

Als Grundlage hierfür dient das Open-Source-System Kubernetes, welches zur Automatisierung, Skalierung und Verwaltung von containerisierten Anwendungen bestimmt ist. Künftige Produktionsanlagen sollen mittels zusätzlichen Edge Devices als Knotenpunkte in einem Kubernetes Cluster fungieren, Ressourcen teilen, untereinander kommunizieren und Softwarepakete unkompliziert bereitstellen. Die Integration der kompakten Linux Rechner ermöglichen den Variablen Einsatz von Hardwareressourcen beim Kunden, der je nach Leistungsanspruch Knotenpunkte erweitern kann. Dabei soll es für die einzelnen Anwendungen möglich sein, sowohl auf cloudbasierten als auch auf on-premise Hardware zur Verfügung gestellt zu werden. Ein hybrides Kubernetes Cluster ermöglicht es somit lokale Rechenleistung oder öffentliche Cloudressourcen in der selben Softwareumgebung zu nutzen.

1.1 Motivation

Die Vorteile von Kubernetes und dem stetigen Paradigmenwechsel der Softwarelandschaft im Cloudbereich, welcher den Wechsel von monolithischen Architektur zu einer mehr flexibleren microservice Architektur bevorzugt, sind das Hauptmotiv der Auswertung neuer agiler Distributionsmöglichkeiten. Die Containerisierung von Anwendungen ermöglichen erst die Aufteilung großer Projekte in kleine unabängige Services die mittels Orchestrierungsplattformen sinnvoll gebündelt werden können. Namenhafte Unternehmen wie Netflix, Amazon und Uber entwickeln und verwenden bereits robuste und komplexe Microservices die containerisiert auf Plattformen verwaltet werden [1].

Durch die Flexibiltät einer solchen infrastuktur ist es möglich Anwendungsfälle im Bereich der künstlichen Intelligenz für die industrie zu testen. Die Anlage Linatronic AI der Krones AG nutzt bereits Deep-Learning-Technologie, um in der Linie mittels Vollinspektion Schäden, Dichtflächen oder Seitenwanddicken zu erkennen und Prozesse zu optimieren [2]. Allgemein sind Anwendungen mit künstlicher Intelligenz durch ihre Komplexität und vielzahl an Abhängigkeiten schwierig zu entwickeln und bereitzustellen. Eine passende Plattform für Anwendungsfälle mit Bezug zur künstlichen Intelligenz muss eine Vielzahl an Services anbieten. Verwaltung von Ressourcen wie Speicher, Rechenleistung und Verbindungsgeschwindigkeit für die Datenübertragung, bei der Ausführung einzelner Phasen von der Datenverarbeitung bis hin zur Evaluierung und Entwicklung [3].

1.2 Zielsetzung

Ziel dieser Arbeit ist die Entwicklung einer Microservice Architektur in einem hybriden Kubernetes Cluster. Das Endresultat soll eine Anwendung werden die mittels einer Weboberfläche, welche über eine Domain erreichbar ist, ein Login-Verfahren mittels einem backend Service ermöglichen der ein Authentifierzungsverfahren per Gesichtserkennung verwendet. Diese Daten sollen schließlich verarbeitet und persistent gespeichert werden, um bei erneuten Aufruf der Website bestehen zu bleiben. Die Konzeption der Anwendung findet containerisiert auf mehreren Software und Hardware Layern statt. Das ganze System wird auf einem Kubernetes Cluster bereitgestellt und verwaltet. Das bereitstellen von einem Service kann bei Vorkonfiguration auf on-premise oder cloudbasierten Ressourcen Ein Ingress Controller dient dabei als Loadbalancer und verteilt die Last beim Aufrufen der Website und der Kommunikation zwischen backend Service.

Kapitel 2

Theoretische Grundlagen

Dieses Kapitel erläutert die Grundlagen die zum Verständnis dieser Arbeit notwendig sind. Zuerst werden bereits bestehende Technologien wie Docker und Kubernetes erklärt, dabei wird bausteinartig auf das Thema Microservices hingeführt.

2.1 Docker

In diesem Abschnitt wird die Technologie „Docker“ näher erläutert und nicht das Unternehmen „Docker, Inc.“, dass für die maßgebliche Entwicklung dessen verantwortlich ist. Angefangen mit der Terminologie, zum deutlicheren Verständnis der nächsten Abschnitte. Forgesetzt mit der aufsteigenden Erklärung der Architektur bis zum Aufbau eines Containers.

2.1.1 Architektur

Die Docker Technologie ist in der Programmiersprache „GO“ geschrieben und nutzt Funktionalitäten des Linux Kernels, wie cgroups und namespaces. Namespaces ermöglichen die Isolation von Prozessen in sogenannte Container, welche unabhängig voneinander arbeiten [4]. Diese beeinhalten alle nötigen Abhängigkeiten zur Ausführung der vordefinierten Anwendung. Container gewinnen dadurch an Portabilität, die ein bereitstellen auf Infrastrukturen mit der Docker Laufzeit ermöglichen. Die Laufzeit setzt sich aus „runc“ einer low-Level Laufzeit und „containerd“ einer higher-Level Laufzeit zusammen (vgl. Abbildung 2.1). Runc dient als Schnittstelle zum Betriebssystem und startet und stoppt Container. Containerd verwaltet die Lebenszyklen eines Container, ziehen von Images, erstellen von Netzwerken und Verwaltung von runc. Die Allgemeine Aufgabe des Docker Daemons ist es eine vereinfachte Schnittstelle für die Abstraktion der unterliegenden Schicht zu gewährleisten, wie zum Beispiel dem verwalten von Images, Volumes und Netzwerken [5]. Auf die Orchestrierung mit Swarm wird nicht weiter eingeganen, da sie zum Verständnis nicht nötig ist.

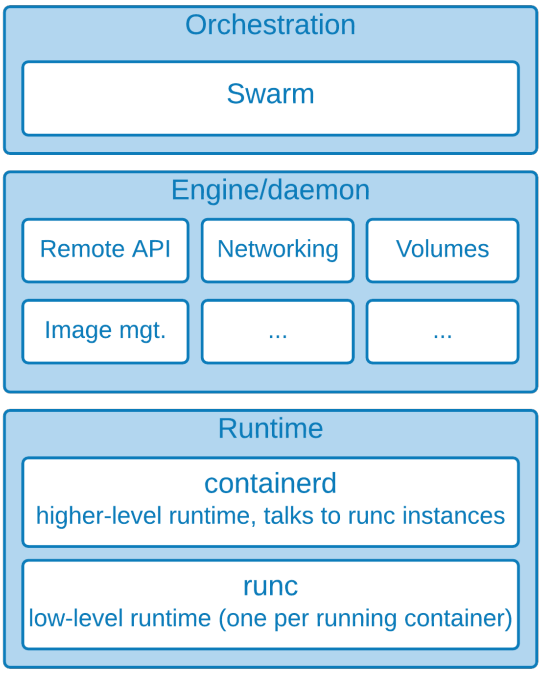

Abbildung 2.1: Docker Architektur [5]

2.1.2 Images und Container

Ein Docker Image ist ein Objekt das alle Abhängigkeiten wie Quellcode, Bibliotheken und Betriebssystem Funktionen für eine Anwendung beeinhaltet.

Registries

Das beziehen von Images erfolgt über sogenannte „Image Registries“. Bei Docker ist dies standardmäßig `https://hub.docker.com` und das eigene Lokale Registry. Es ist auch möglich eigene zu hosten oder die von Drittanbieter zu nutzen.

Schichten

Docker Images bestehen aus mehreren Schichten, jede davon abhängig von der Schicht unter ihr und erkennbar durch IDs in Form von SHA256 Hashes (vgl. Abbildung 2.2). Docker kann dadurch beim bauen oder updaten von neuen Images vorhandene Schichten erneut verwenden. Die feste Reihenfolge ermöglicht eine ressourceneffiziente Verwaltung von Builds, indem man oft wechselnde Schichten oben platziert. Die Leistung beim erstellen und zusammenführen von Schichtem hängt vom Dateisystem des Hostsystems ab. Eine Schicht kann aus mehreren Dateien bestehen und einzelne Dateien aus der Unterliegenden Schicht mit einer neuen ersetzen.

Das starten eines Containers fügt auf die bereits bestehenden Schichten einen „Thin R/W layer“ oder auch „Container layer“ genannt hinzu, dieser gewährt Schreib- und Leserechte bei Laufzeit des Prozesses. Jeder dieser Container hat somit einen

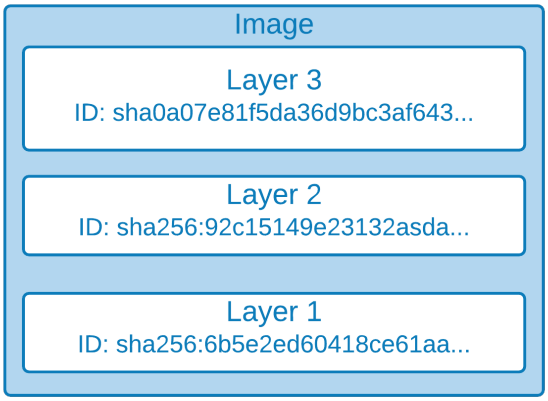

Abbildung 2.2: Image Layers

individuellen Zustand, der unähnlich vom abstammendem Image ist. Bei Löschung des Containers verschwindet auch die dazu gewonne Schicht. Das entfernen eines Images ist durch die Konzeption des Schichtensystem erst möglich, wenn alle darauf basierenden Container gelöscht sind [6].

Dockerfile

Zur Erstellung eines Docker Images wird ein Dockerfile benötigt, dies beeinhaltet alle Anweisungen zum Aufbau der einzelnen Schichten.

2.1.3 Containervirtualisierung

Ein Container ist eine laufende Instanz eines Images.

Dieses Konzept kommt dem einen oder anderen Leser vielleicht schon bekannt vor, vergleichbar ist dieses Konzept mit dem einer VM. Images ermöglichen ähnlich wie VM templates,das erstellen von mehreren Instanzen durch eine Vorkonfiguration. Mit dem großen Unterschied, dass die Einrichtung von VMs müheseliger ist und weitaus mehr Ressourcen beansprucht, da sie ein ganzes Betriebssystem ausführt. Containertechnologien bauen hingegen nur auf bestimmte Funktionalitäten des Kernels auf und sparen damit an Rechenleistung (vgl. Abbildung 2.3).

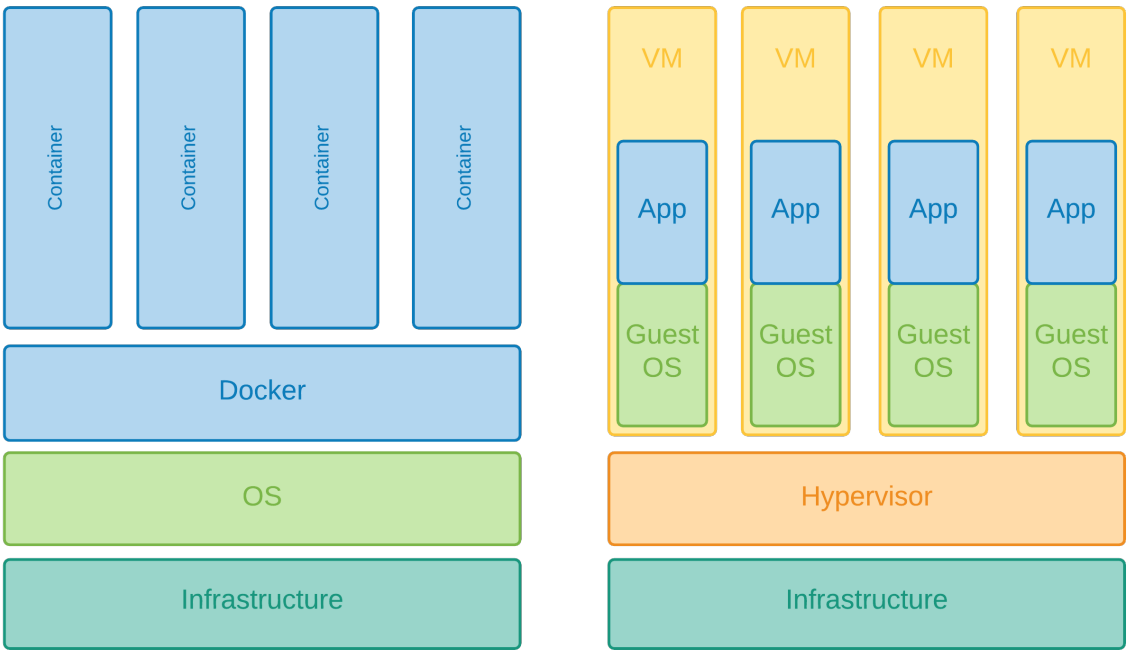

Abbildung 2.3: Container gegen virtuelle Maschinen

2.2 Kubernetes

2.2.1 Architektur

2.2.2 Lightweight Kubernetes

2.2.3 Hybrid Cloud

2.2.4 Rancher

2.3 Microservice

2.3.1 Aufbau

2.3.2 Entwicklung

2.3.3 Dezentrale Datenmanagement

Kapitel 3

Microservice Architektur

3.1 Anforderungen

3.2 Konzept

3.3 Herausforderungen

3.4 KI-Anwendungsfall

3.4.1 Gesichtserkennung

Kapitel 4

Implementierung der Architektur

4.1 Konfiguration und Einrichtung

4.1.1 Virtueller Privater Server

4.1.2 Domain

4.1.3 SSL-Verschlüsselung

4.2 Frameworks und Bibliotheken für Microservices

4.2.1 Flask

4.2.2 Gunicorn

4.2.3 SocketIO

4.2.4 OpenCV

4.2.5 MongoDB

4.3 Gesichtserkennung

4.3.1 Alignment

4.3.2 Training

4.3.3 Model

4.4 Containerisierung

4.4.1 Volumes

4.4.2 Netzwerk

4.4.3 Docker-Compose

4.4.4 DockerHub

Kapitel 5

Ergebnisse

5.1 Microservice

5.1.1 Frontend-Serivce

5.1.2 Backend-Serivce

5.1.3 Loadbalancer

5.1.4 Kubernetes Cluster

Kapitel 6

Diskussion und Ausblick

6.1 Dikussion

6.2 Ausblick

Literaturverzeichnis

[1] M. Villamizar, O. Garces, H. Castro, M. Verano, L. Salamanca, R. Casal- ´ las, and S. Gil, “Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud,” in 2015 10th

[2] Krones Linatronic 735,” Krones.com. [Online]. Available: https://www.krones.com/de/produkte/maschinen/leerflaschen-inspektionsmaschine-linatronic-735.php. [Accessed: 16-Jan-2022].

[3] Y. Zhou, Y. Yu and B. Ding, "Towards MLOps: A Case Study of ML Pipeline Platform,"2020 International Conference on Artificial Intelligence and Computer Engineering (ICAICE), 2020, pp. 494-500, doi: 10.1109/ICAICE51518.2020.00102.

[4] Docker overview,” Docs.docker.com. [Online]. Available: https://docs.docker.com/get-started/overview/. [Accessed: 20-Jan-2022].

[5] N. Poulton, “Docker,” in Docker Deep Dive: Zero to Docker in a single book: Nigel Poulton, 2020, pp. 12–13.

[6] About storage drivers,” Docs.docker.com. [Online]. Available: https://docs.docker.com/storage/storagedriver/. [Accessed: 20-Jan-2022].

Abbildungsverzeichnis

Tabellenverzeichnis