TECHNISCHE HOCHSCHULE NÜRNBERG
GEORG SIMON OHM
Fakultät efi
Studiengang Elektronische und Mechatronische Systeme

# Systemintegration for mobile Robotics

Masterarbeit von
Aayush Yadav
Matrikel-Nr.: 3549452

Sommersemester 2021
Abgabedatum: 15.12.2021

Betreuer:
Prof. Dr. Stefan May
Technische Hochschule Nürnberg

Michael Fiedler
SIEMENS

Schlagworte: Docker, Industrial Edge, ROS2

# Contents

# 1 Introduction

## 1.1 Motivation

## 1.2 Problem Statement

asdf fdsa
Nach [**Krypczyk2018**] müssen folgende Fragen nach der Analysephase des Softwareentwicklungsprozess beantwortbar sein:

- Was sind die entscheidenden Anforderungen an die zu erstellende Software?
- Welches Problem soll mithilfe des Anwendungssystems gelöst werden?
- Welche Wünsche haben Ihre Kunden und Nutzer an das System?

# 2 Background

This chapter gives an outline of the concepts and a detailed explanation of the various technologies that will be used later in this thesis.

## 2.1 Containers

The concept of container technology uses the same model as shipping containers in transportation. The idea is that before the invention of shipping containers, manufacturers had to ship goods in a variety of fashions which included ships, trains, airplanes, or trucks, all with different sized containers and packaging. With the standardization of shipping containers, products could be transported seamlessly without further preparation using different shipping methods. Before the arrival of this standard, shipping anything in volume was a complex, laborious process. The motivation behind software containers is the same. [1, P. 1]

Instead of shipping a complete operating system (OS) and the software (with necessary dependencies), we pack our code and dependencies into an image that can run anywhere. Furthermore, it enables the packaging of clusters of containers onto a single computer. In other words, a container consists of an entire runtime environment: an application, plus all the dependencies, libraries, and other binaries, and configuration files needed to run it, bundled into one package. The ability to have software code packaged in pre-built software containers means that code can be pushed to run on servers running different Linux kernels or be connected to run a distributed app in the cloud. This approach also has the advantage of speeding up the testing process and creating large, scalable cloud applications. This approach has been in software development communities for several years. It has recently gained in popularity with the growth of Linux and cloud computing. [1, P. 2]

### 2.1.1 Containerization vs Virtualization

Linux containers and virtual machines (VMs) are both package-based computing environments that combine several IT system components and keep them isolated from the rest of the system. Their main distinguishing features are scalability and portability. Containers are usually measured in megabytes, whereas VMs in gigabytes. [2]

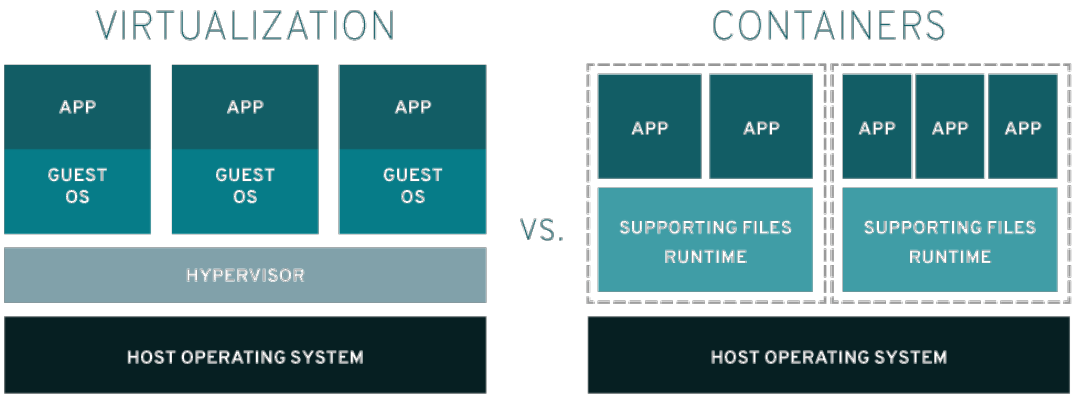


Figure 2.1: Differences between Virtualization and Containerization [2]

**Containerization** is an alternative to standard virtualization that encapsulates an application in a container with its executing environment. Containers hold an application and everything it needs to run. Everything within a container is maintained on an image—a code-based file that includes all libraries and dependencies. These files are similar to a Linux distribution installation. An image comes with RPM packages and configuration files. Containers are so small compared to VMs, there are usually hundreds of them loosely coupled together.[2]

**Virtualization** is a way of sharing a single physical instance of a resource or an application to multiple organizations and clients. It utilizes software called a hypervisor that separates resources from their physical devices. It enables the partitioning of the resources and assigned to individual VMs. When a user issues a VM instruction that requires additional resources from the physical environment, the hypervisor sends the request to the physical system and saves the changes. VMs look and act like physical servers, which can multiply the drawbacks of application dependencies and large OS footprints—a footprint that's often not required to run a single app or microservice.[2]

Table 2.1 illustrates the key differences between the above two approaches concerning package-based computing environments.

| Parameters | Virtualization | Containerization |
|---|---|---|
| Isolation | Provides complete isolation from the host operating system and the other VMs | Provides lightweight isolation from the host and other containers, but doesn't provide a strict security boundary as a VM |
| Operating System | Runs a complete operating system including the kernel, thus requiring more system resources such as CPU, memory, and storage | Runs the user-mode portion of an operating system, and can be customized to include just the required services for your app utilizing fewer system resources |
| Compatibility | Runs just about any operating system inside the virtual machine | Runs on the same operating system version as the host |
| Deployment | Deploys individual VMs by using Hypervisor | Deploys single container by using Docker or deploy multiple containers by using an orchestrator such as Kubernetes |
| Persistent storage | Uses a Virtual Hard Disk (VHD) for local storage for a single VM or a Server Message Block (SMB) file share for storage shared by multiple servers | Uses local disks for local storage for a single node or SMB for storage shared by multiple nodes or servers |
| Networking | Uses virtual network adapters | Uses an isolated view of a virtual network adapter. Thus, providing a little less virtualization |
| Startup time | They take few minutes to boot up | They can boot up in few seconds |

Table 2.1: Differences between Virtualization and Containerization [3]

**Image** Ein Docker-Image ist ein Speicherabbild eines Containers. Das Image besteht aus mehreren Layern, die schreibgeschützt sind. Images können manuell erstellt werden, oder mit Verwendung eines dockerfiles automatisiert kompilliert werden.

**Container** Ein Container ist die aktive ausgeführte Instanz eines Images. Es läuft immer ein Prozess innerhalb des Containers. Somit wird ein Container beendet, sobald er kein Programm ausführt oder das Programm mit seinem Auftrag fertig ist.

**Dockerfile** Ein Dockerfile ist eine Textdatei mit Anweisungen zum erstellen eines Containers. Jeder Schritt erstellt einen neuen Layer, der dem Image hinzugefügt wird. Mit einem Dockerfile lässt sich das erstellen von Images automatisieren.

**Docker-Compose** Mit Docker-Compose lassen sich die Startparameter eines Containers in einer .yaml -Datei festlegen. Es lassen sich mehrere Docker-Container gleichzeitig starten, sowie umfangreiche Einstellungen bei dem Start des Containers treffen.

### 2.1.2 Netzwerke

Docker bietet auch die Netzwerkkapselung für die Container an. Somit lässt sich das Netzwerk eines Containers von dem Rest des Systems oder anderen Containern trennen. Es können auch eigene Netzwerke definiert werden, um mehrere Container miteinander Kommunizieren zu lassen. Sie spannen so ein eigenes Netzwerk auf, was vergleichbar mit einem lokalen Netzwerk ist. Zwei Netzwerktypen die Docker anbietet, werden nachfolgend erläutert.

**Bridged** Das Bridged Netzwerk kapselt das Netzwerk des Containers von dem des Host Systems, bietet aber noch eine Netzwerk-Brücke, damit der Container Teilnehmer in höherliegenden Netzen erreichen kann. Es lassen sich Ports des Host-Computers an einen Container weiterleiten, um so eine Erreichbarkeit des Containers aus einem höherliegendem Netz zu ermöglichen.

**Host** Die Netzwerkeinstellung Host verzichtet auf eine Kapselung der Netze. Hier verwendet der Container die IP Adresse des Host-Computers und ist entsprechend teil des Host-Netzes. Für weitere Teilnehmer des Netzwerkes, ist der Host-PC des Docker-Containers und der Container selbst, dasselbe Gerät. Eine Portweiterleitung zum ermöglichen von Diensten ist hier nicht notwendig.

### 2.1.3 Datenspeicherung

Nach beenden des Containers werden die Daten, die während dem ausführen des Containers angefallen sind, gelöscht. Um eine Persistente Speicherung, zu ermöglichen, bietet Docker mehrere Möglichkeiten an, um über die Laufzeit des Containers hinaus Daten zu speichern, oder auch um auf Daten innerhalb des Containers zuzugreifen. Nachfolgend werden die wichtigsten kurz vorgestellt.

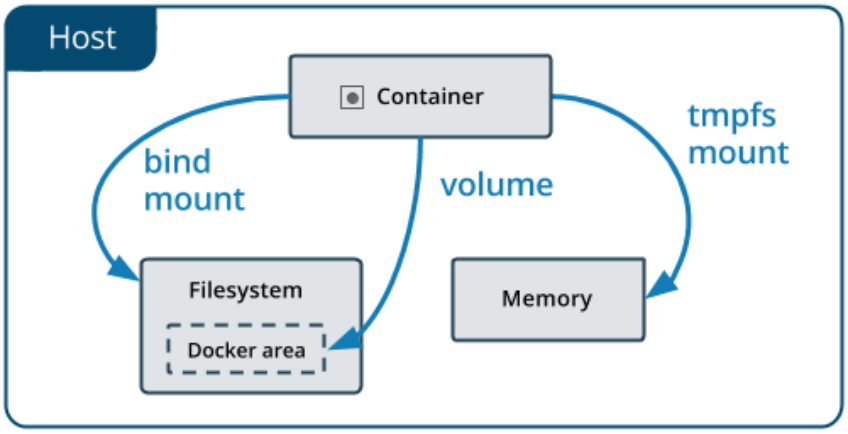


Figure 2.2: Verschiedene Datenspeichermöglichkeiten. [4]

**Volumes** Volumes werden auf dem Dateisystem des Hosts gespeichert (In Linux unter */var/lib/docker/volumes/*). Ein Volume kann in mehrere Container eingebunden werden, dadurch ist eine containerübergreifende Dateiablage möglich. Volumes werden nicht automatisch gelöscht, sobald alle Container gestoppt sind, welches das Volume eingebunden hatten. Sie eignen sich am besten, um Daten Persistent zu speichern. [4]

**Bind mounts** Bind mounts können überall auf dem Hostsystem gespeichert sein. So können wichtige Systempfade (zum Beispiel */dev/*) dem Container zugänglich gemacht werden. [4]

**tmpfs mounts** Hier werden die Dateien nicht auf dem Dateisystem des Hosts gespeichert, sondern auf dem Arbeitsspeicher des System. Dadurch entsteht eine nicht persistente Datenspeicherung, um geheime Informationen zwischen Containern auszutauschen. [4]

## 2.2 Industrial Edge

Industrial Edge ist eine Entwicklung von Siemens, die ermöglicht, im Prozess anfallende Daten bereits auf dem Gerät zu analysieren. Es wird lokales Engineering mit Cloud Engineering kombiniert. [5] Dadurch werden unnötige Datentransporte zwischen dem Endgerät und Server vermieden und erst bereits aufbereitete Daten zum Server geschickt. Die Rechenlast fließt so vom Server in Richtung des Endgerätes. Hierfür werden Applikationen (Apps) bereitgestellt, die einen festen Funktionsumfang enthalten und mit der Industrial Edge Infrastruktur auf das jeweilige Endgerät (Edge Device) gespielt werden können. Diese Apps basieren auf der Docker Technologie. Industiral Edge ermöglicht die zentrale Verwaltung von In der nachfolgenden Abbildung 2.3 ist eine typische Infrastruktur von dem obersten Server, bis zu den Endgeräten dargestellt, die im Nachfolgenden näher erläutert wird.

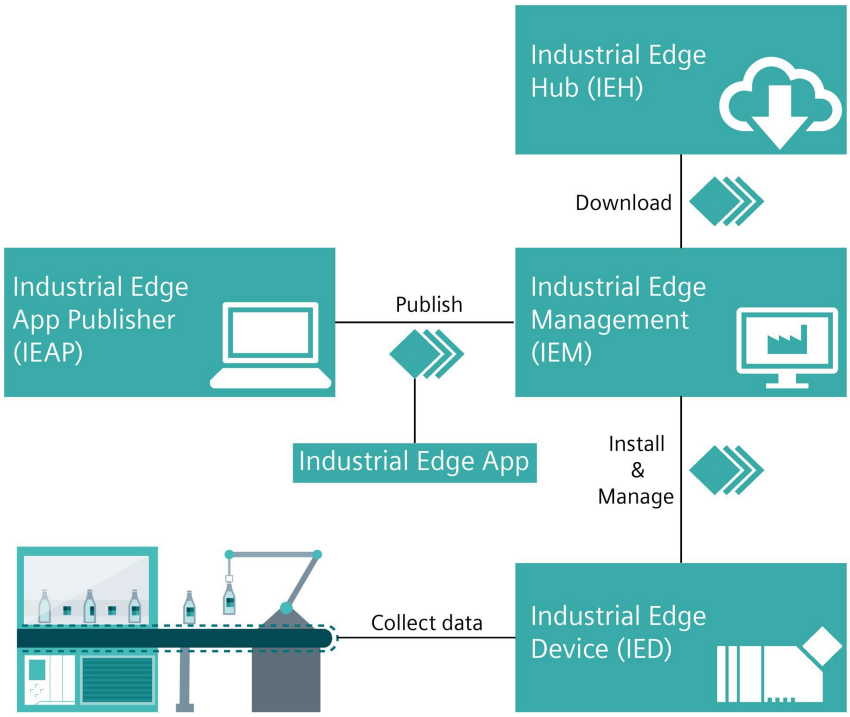


Figure 2.3: Überblick über Industrial Edge. [5]

**Industrial Edge Hub** Das Industrial Edge Hub (IEH) ist die oberste, von Siemens bereitgestellte Serverebene. Hier bietet Siemens eigene entwickelte Applikationen an, die gegen eine Lizenzgebühr auf das Industrial Edge Management (IEM) geladen werden können. Von hier aus lassen sich ebenfalls benötigte Softwarepakete und Dokumentation herunterladen, die das Entwickeln von eigenen Apps und das Aufsetzen und betreiben von einem eigenen IEM ermöglicht. [5]

**Industrial Edge Management System** Das IEM ist ein Server, welcher in einem Lokalen Netz selbstständig betrieben werden kann. Der Anwender hat hiermit die Möglichkeit, einen Server zu erstellen, damit z.B. sensible Daten, die zwischen dem Edge Device und dem IEM ausgetauscht werden, nicht über das Internet transportiert werden müssen. Zudem lassen sich die Endgeräte über diesen Server verwalten, es können Apps sowie Softwareupdates aufgespielt werden, sowie lassen sich weitere Analysen durchführen. Eine selbst entwickelte App lässt sich auf das IEM aufspielen, welches Diese wiederum an die Edge Devices verteilen kann. [5]

**Industrial Edge App** Eine Edge App dient zur intelligenten Verarbeitung von Automatisierungsaufgaben. [5] Neben den von Siemens im IEH angebotenen Apps lassen sich auch eigene Applikationen entwickeln. Diese und eigene entwickelte Apps sind Docker basierte Images, welche auf dem Industrial Edge Device (IED) ausgeführt werden. Als oberste Beschreibungsebene wird eine docker-compose Datei angelegt, in welcher beschrieben wird, wie die docker Images zu starten sind, sowie welche Netzwerkkonfiguartion (Kapitel 2.1.2) und welche Art der Datenspeicherung (Kapitel 2.1.3) verwendet wird.

**Edge Device** Ein IED führt die einzelnen Edge Apps aus. Sie können Automatisierungsdaten lokal speichern und nach Bedarf abrufen. Ebenfalls lassen sich diese Daten an eine Cloud-Infrastruktur übermitteln. Ein IED muss bei einem IEM aktiviert werden. [5]

**Industrial Edge Publisher** Der Industrial Edge Publishers ist ein Tool, mit dem Docker Images in Edge Apps überführt und in das IEM hochgeladen werden. Es lassen sich ebenfalls bereits erstellte Apps verwalten, verändern oder löschen. [6]

## 2.3 ROS 2

Ein asdfasdasd [7] Robot Operating System 2 (ROS2)

### 2.3.1 Nachrichtentypen

- Message (topics)
- Service
- Action

### 2.3.2 ROS 2-web Bridge

- Stellt einen websocket bereit, der mittels JSON requests verarbeitet. - Basiert auf RclNodeJS, welches eine brücke zwischen javascript und ros2 bildet
- Basiert auf NodeJS. NodeJs ist eine JavaScript Runtime, die JS Code serverseitig ausführen lässt. Das ist hier nicht weiter von relevanz.
- Verwendet wird für die Anwendung ebenfalls roslibjs. Dies ist eine javascript bibliothek, welche in der html eingebunden wird. Sie stellt Befehle zum erstellen von ROS Subscriber/Publisher und Services bereit.

### 2.3.3 roslibjs

## 2.4 DDS

- Grundlage für Kommunikation von ROS2
- Realisiert die eigentliche Kommunikation
- Wenn das hier geht, geht auch ROS2! - Verschiedene Systemanbieter, näher wird RTI und Fastrtps untersucht: https://ros.org/reps/rep-2000.html (Beide TIER 1)

ROS2 verwendet als Kommunikationsschnittstelle DDS. DDS funktioniert gut in lokalen Netzwerken, sobald allerdings ein NAT-Router dazwischen ist, funktioniert der standardmäßige Discovery Mechanismus nicht. Da das Ziel eine Edge Applikation ist, soll das Verhalten von DDS in der Docker Umgebung, bzw. in unterschiedlichen Netzwerken untersucht werden.

# 3 Approach

Im Rahmen dieser Masterarbeit werden mehrere Kommunikationswege über verschiedene Geräte verwendet. So ist es ein Ziel, das Framework ROS2 zu verwenden.

## 3.1 Testsetup

Mithilfe einer Testumgebung können verschiedene Konfigurationen getestet und entwickelt werden. Notwendig ist, dass das Testsetup entscheidende Merkmale des Netzwerkes in der realen Umgebung beibehält. So ist die Aufteilung der Teilnehmer in zwei verschiedene Netzwerke wichtig, da die zu entwickelnde Edge App auf die Netzwerkeinstellung "host" verzichten soll. Ebenfalls wird dadurch eine sehr viel höhere Flexibilität der Netzwerkstruktur ermöglicht.

### 3.1.1 Host Only

Das Erste der beiden Testnetzwerke erstellt drei Virtuelle Maschinen: VM A, VM B und VM Router. Die Virtuelle Maschine VM A ist dem Subnetz 192.168.64.0/24 zugeordnet. VM B dem Subnetz 192.168.80.0/24. Als Brücke beider Subnetze dient VM Router, welche eine Verbindung zu beiden Netzwerken aufbaut und sämtliche Pakete an das jeweils andere Netz schickt.

# 4 Design and Implementation

## 4.1 Struktur

- Präsentation, die ich laura gezeigt habe

# 5 Result

- Schnittstelle für TIA Projekte definieren
- Funtionserweiterung für TIA Openess
- Device seite unabhängig von Master Client

# 6 Discussion

Mal schauen was so kommt :-)

# Literaturverzeichnis

[1] Jangla Kinnary. *Accelerating Development Velocity Using Docker*. Apress, 2018. ISBN: 978-1-4842-3935-3. URL: https://doi.org/10.1007/978-1-4842-3936-0.

[2] Red Hat Inc. URL: https://www.redhat.com/en/topics/containers/containers-vs-vms (visited on 05/29/2021).

[3] Tarnum Java SRL(Baeldung). URL: https://www.baeldung.com/cs/virtualization-vs-containerization (visited on 05/29/2021).

[4] Docker Inc. URL: https://docs.docker.com/storage/ (visited on 05/20/2021).

[5] Siemens AG. *Industrial Edge Management – Getting Started, V1.2.0, A5E50177815-AC*. 2021.

[6] Siemens AG. *Industrial Edge App Publisher – Bedienung, V1.2.0, A5E50177958-AC*. 2021.

[7] Open Source Robotics Foundation Inc. URL: https://docs.ros.org/en/foxy/index.html (visited on 05/25/2021).

# List of Figures

# List of Tables

# Listings