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]

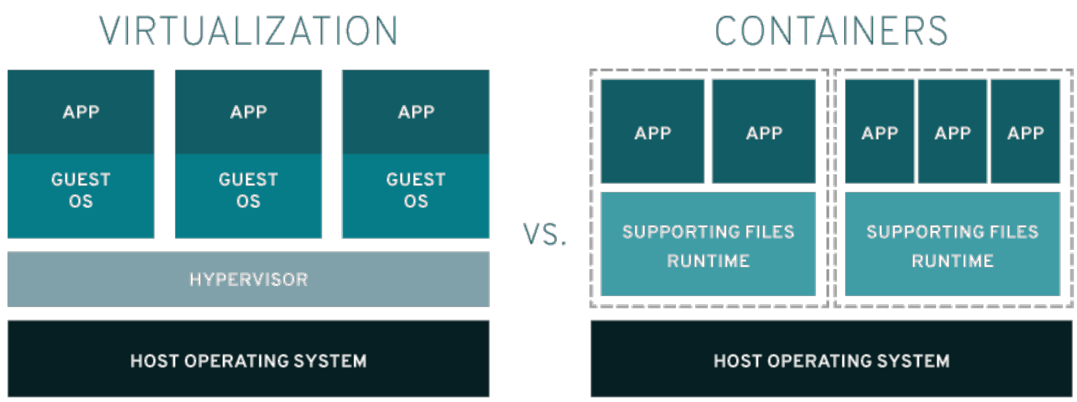

Figure 2.1: Differences between Virtualization and Containerization [2]

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]

The use of containers can decrease the required time for developing, testing, and deploying applications. It makes testing and fault detection less complex as there is no difference between running your application on a test environment and in production. It provides a cost-effective solution and can help reduce operational and development expenses. In most use-cases, container-based virtualization offers several advantages over traditional Virtual Machine based virtualization.

2.2 Docker

Docker is a person who works at a port whose job is to load goods onto and off container ships. [4]

Software Docker essentially does the same in the software context. Docker is a collection of open-source tools that quickly wraps up any application and all its unique dependencies in a lightweight, portable, self-sufficient container that can run virtually anywhere on any infrastructure.[5] Docker was launched as an open-source project by dotCloud, Inc. in 2013. it relies heavily on namespaces and cgroups to provide resource isolation and to package an application along with its dependencies. This bundling of dependencies into one package allows an application to run across different platforms and still support a level of portability. This provides flexibility to developers to develop in the desired language and platform. It has drawn a lot of interest in recent years.[1, P. 10]

Docker consists of several parts. The following section gives an overview of the main components of Docker.

2.2.1 The Docker Runtime and Orchestration Engine (Docker Engine)

The Docker engine is the software for the infrastructure that runs and orchestrates containers. All other Docker, Inc. and third-party products connect to and develop around the Docker Engine. It provides a workflow for building and managing the application stack. It builds and runs containers using other Docker components and services. It consists of the Docker daemon; a REST API that specifies the interfaces that programs can use to communicate with the daemon; and the CLI, the command-line interface that communicates with the Docker daemon via the API. Docker Engine creates and runs the Docker container from the Docker image file.

Following Diagramm illustrates the Docker System Architecture.

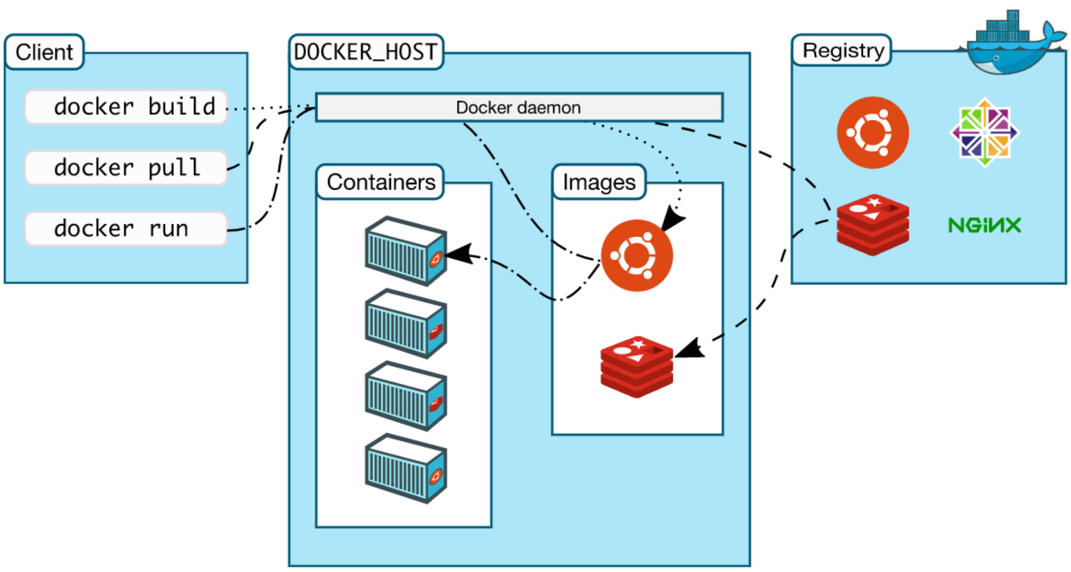

Figure 2.2: Docker System Architecture [6]

Docker daemon (dockerd) is a server process that runs in the background. It continuously listens to the REST API interface and listens for incoming requests and manages Docker objects (images, containers, networks, and volumes). A daemon also has the ability to communicate with other daemons to manage Docker services.[6]

Docker client represents the primary means for most users to interface with Docker. The commands run through the command-line interface are sent to the Docker daemon through the Docker API interface. The Docker daemon(dockerd) then executes these commands. The Docker client has the ability to connect with multiple Docker daemons.[1, P. 32]

Docker registry The images created by the Docker daemon are stored in the Docker registry. Docker looks for images in the Docker Hub by default, but it is possible to have a self hosted private registry. Docker Hub is a public registry and is freely accessible.[1, P. 33]

2.2.2 Docker Objects

Images

Containers

Networks

Volumes

Dockerfiles

Docker-Compose

2.3 Industrial Edge

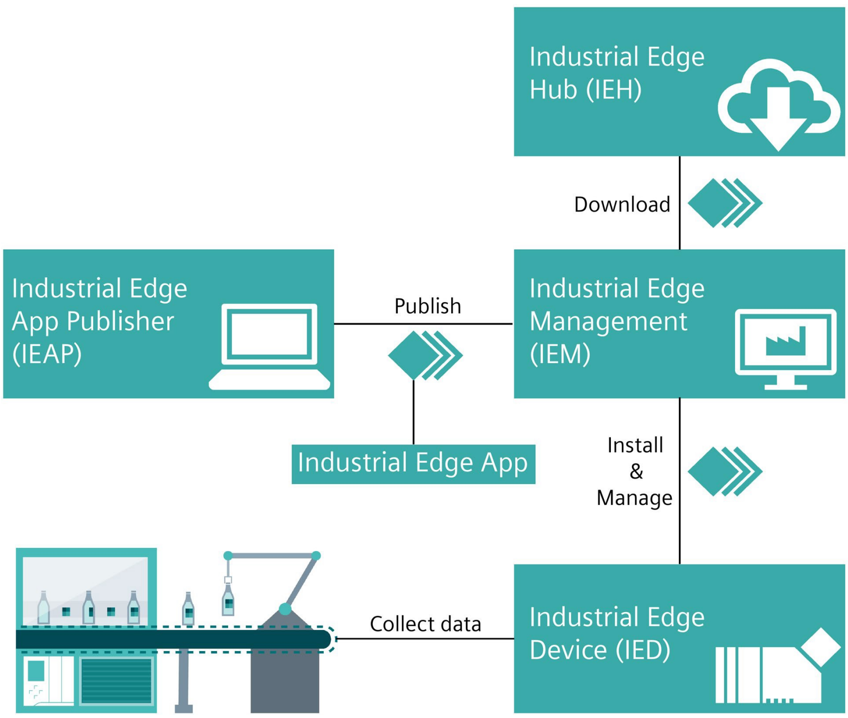

Figure 2.3: Überblick über Industrial Edge. [8]

Industrial Edge Hub Das Industrial Edge Hub (IEH)

Industrial Edge Management System Das Industrial Edge Management (IEM)

Industrial Edge App

Edge Device

Industrial Edge Publisher

2.4 ROS 2

Ein asdfasdasd [10] Robot Operating System 2 (ROS2)

2.4.1 Nachrichtentypen

- Message (topics)
- Service
- Action

2.4.2 ROS 2-web Bridge

2.4.3 roslibjs

2.5 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)

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] Cambridge english dictionary. URL: https://dictionary.cambridge.org/dictionary/english/docker (visited on 05/29/2021).

[5] MAUREEN O'GARA. URL: https://web.archive.org/web/20190913100835/http://maureenogara.sys-con.com/node/2747331 (visited on 05/29/2021).

[6] Docker Inc. URL: https://docs.docker.com/get-started/overview/ (visited on 05/29/2021).

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

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

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

[10] 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