Dienst mit Container anwenden

Kompetenz

Erkennt Unterschiede der Containeranwendungsmöglichkeiten im eigenen beruflichen Alltag. Entwickelt mittels einer geeigneten Containerkomposition eine ICT-Lösung für den eigenen Betrieb oder für das betriebliche Umfeld.

Objekt

Virtualisierung einer Applikation mit Container.

Modul 347

1 Inhalt

2 Warum Container?

Die Container-Technik und Docker verändern den Umgang mit (Server)-Software. Von der neuen Flexibilität profitieren Unternehmensumgebungen genauso wie der eigene Heim- oder Webserver oder die Hausautomation.

Software-Container machen den Test und den Betrieb von Server-Diensten einfach: Ein Container enthält eine Software mit all ihren Abhängigkeiten. Gestartet wird ein solcher Container aus einem Abbild, dem Container-Image. Er verhält sich auf jedem Gerät exakt gleich – auf der Entwicklermaschine genauso wie auf dem Root-Server oder beim Cloud-Anbieter. Die Software im Container merkt nicht, dass sie im Container steckt – für sie sieht ihre Umgebung wie eine eigene Maschine aus. Löscht man einen Container, hinterlässt er keine Spuren, keine Konfigurationsreste im Betriebssystem. Das sind die Versprechen, mit denen eine Container-Umgebung wie Docker Administratoren und Entwickler überzeugen will.

Damit Container die Arbeit wirklich erleichtern, sollte man die zugrunde liegenden Ideen und Begrifflichkeiten verstanden haben. Ein wichtiges Paradigma der Docker-Welt lautet: ein Dienst pro Container. Man wirft also nicht etwa Datenbank und Webserver in dasselbe Image, um eine Webanwendung zu starten. Vielmehr stecken sowohl Datenbank als auch Webserver in ihren eigenen Images.

2.1 Grundbegriffe

Die wichtigste Information vorab: Docker-Container sind keine virtuellen Maschinen. Jeder laufende Container ist ein eigener Prozess innerhalb des Host-Betriebssystems. Hier gibt es keinen Hypervisor und keine virtuelle Hardware, auf der ein weiteres vollständiges Betriebssystem läuft. Die Container und das Host-Betriebssystem greifen auf denselben laufenden Linux-Kernel zurück. Das sieht man deutlich bei einem Blick in die Prozessliste des Host-Betriebssystems: Dort werden die in den Containern laufenden Prozesse genauso aufgelistet wie solche, die nicht in Containern stecken. Anders als eine virtuelle Maschine beansprucht ein Container also keine Ressourcen, um ein vollständiges Gastbetriebssystem laufen zu lassen. Container sind „nur“ durch Namespaces voneinander getrennte Prozesse – interessant wird Docker durch das Drumherum, also die Verwaltung von Images, Netzwerken und Volumes. Aber der Reihe nach.

Basis eines jeden Docker-Containers ist ein **Docker-Image**. Ein solches Image beinhaltet nur die grundlegenden Programme, Bibliotheken und Daten. Aus einem Image lassen sich beliebig viele Container erzeugen. Man kann sich ein Docker-Image wie eine Kopiervorlage vorstellen, die genutzt wird, um davon Container als Kopien herzustellen.

Images müssen Sie nicht immer per Hand selbst bauen. Dann wäre Docker nicht so erfolgreich geworden. Images entstehen aus einem Bauplan, dem Dockerfile. Es enthält Anweisungen, auf welchem anderen Image ein neues Image aufbauen soll, kopiert die Software und die Abhängigkeiten hinein und ergänzt Befehle, zum Beispiel Installationsskripte. Wer ein solches Dockerfile für eine Anwendung geschrieben hat, kann das Image in einer Registry veröffentlichen. Die grösste Registry ist der Docker-Hub, betrieben von der Docker Inc., der Firma hinter der Software Docker. Wenn man für eine Aufgabenstellung ein fertiges Container-Image sucht, wird man dort aber schnell erschlagen. Im Docker-Hub finden sich schon für ein und dieselbe Software durchaus mehrere hundert Images. Einige werden regelmässig gepflegt, andere seit Jahren vernachlässigt.

Betriebssystemumgebungen von Containern

Docker-Container erhalten die auszuführende Software und alle von ihren benötigten Bibliotheken aus Docker-Images. Diese setzen sich aus vielen Schichten zusammen, die sich wiederverwenden lassen – MySQL- und Wordpress-Schichten bauen beispielsweise auf derselben Debian-Schicht auf.

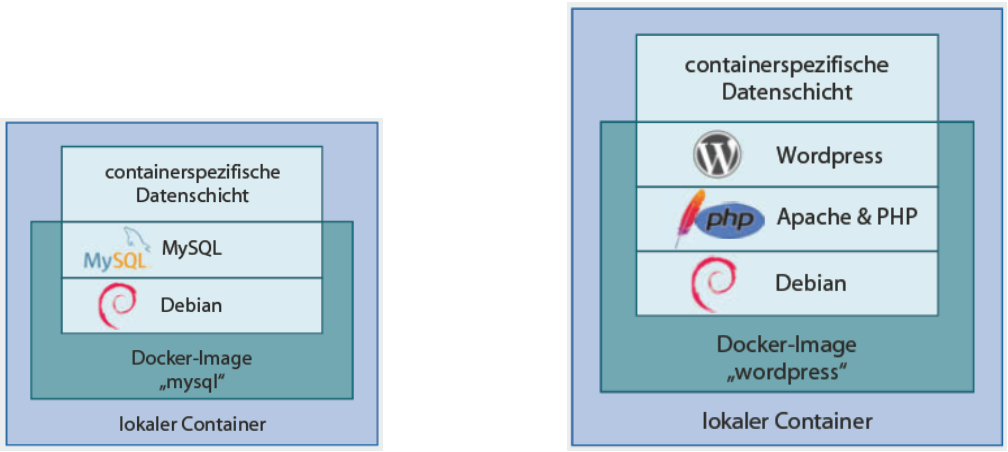

Abbildung 1: Betriebssystemumgebungen von Containern

2.2 Geschichtet

Wenn Docker auf Basis des Images einen Container startet, erzeugt es eine weitere beschreibbare containerspezifische Datenschicht. Alle Schichten darunter, die Teil des Images sind, bindet es nur lesbar ein. Der Prozess im Container merkt von der Schichtung nichts. Will der Prozess im Container in eine Datei schreiben, landen die geschriebenen Dateien in der containerspezifischen Datenschicht, denn nur sie ist für den Prozess beschreibbar. Diese Schicht ist aber nicht persistent, sie verschwindet, wenn dieser neu gestartet wird. Das ist bei der Arbeit mit Docker sehr nützlich: Hat ein Prozess ein Problem, reicht das erneute Erzeugen des Containers, damit wieder der ursprüngliche Zustand herrscht.

Für Daten, die im laufenden Betrieb anfallen, ist das aber nicht zielführend – die Daten, die zum Beispiel eine Datenbank schreibt, sollen ein Neuerstellen überleben. Docker nutzt dafür Volumes, also Ordner im lokalen Dateisystem, die in den Container übergeben werden.

2.3 Netzwerk

Docker kümmert sich auch darum, dass die Prozesse im Container eine Netzwerkverbindung bekommen. Darüber können sich Container untereinander erreichen – der Webserver-Container also zum Beispiel die Datenbank. Sie können auch einen Netzwerk Port Ihres Hostsystems mit einem Port im Container verbinden.

Man kann zum Beispiel mehrere Container in ein dediziertes Netzwerk verfrachten, um sie von den übrigen zu trennen. Dabei stellt Docker auf Wunsch nicht nur die automatische Vergabe von IP-Adressen, sondern auch eigene DNS-Dienste bereit, mittels denen Dienste zueinander finden können.

2.4 Für Entwickler

Software-Entwickler profitieren von Containerisierung gleich doppelt – unabhängig von der Programmiersprache. Auf der einen Seite können sie sich schnell eine Umgebung mit allen Abhängigkeiten zusammenstellen. Für viele Programmiersprachen gibt es Basis-Images, auf denen man die eigene

Anwendung aufsetzen kann. Wer mit Containern arbeitet, kann sich sehr sicher sein, dass das Programm darin genau die Bibliotheken nutzt, die man im Container installiert hat – nicht etwa eine alte

Version, die noch im Dateisystem der Entwicklermaschine hängengeblieben ist.

Auf der anderen Seite können Sie Ihre Anwendung im Container leicht an Kollegen, Kunden und andere Interessierte weitergeben. Den typischen Fehler „Works on my machine“ können Sie ausschliessen – der Container wird sich überall gleich verhalten. Besonders Open-Source-Projekte profitieren, weil man sie schnell aus dem Docker-Hub starten und ausprobieren kann.

2.5 Videos

Docker lernen: Eine Einführung https://www.youtube.com/watch?v=DESdVoKhIxY

Docker für Fortgeschrittene https://www.youtube.com/watch?v=3YP0eUHdLc8

3 Grundlegende Elemente in Docker

3.1 Image

Ein Image können Sie sich wie eine Vorlage für einen Docker-Container vorstellen. Sie erzeugen einen Docker-Container immer aus einem Image.

3.2 Container

Ein Container ist eine laufende Instanz eines Images. Mit der Anwendung innerhalb eines Containers kann der Anwender oder ein anderes System interagieren. Aus einem Image können Sie beliebig viele Container erstellen.

3.3 Container-Host

Der Container-Host ist der Rechner, auf dem die Container-Engine ausgeführt wird und auf dem dann schliesslich die Container laufen.

3.4 Container-Engine

Die Container-Engine ist die Software, die sich um die Bereitstellung der Container auf dem Container-Host kümmert. Falls Sie sich mit virtuellen Maschinen auskennen: Die Container-Engine ist am ehesten mit dem Hypervisor vergleichbar.

3.5 Container-Netzwerke

Auf dem Container-Host existieren Container-Netzwerke, über die die Container miteinander und auch mit der Aussenwelt verbunden werden. Es gibt ein Standard-Netzwerk, Sie können aber beliebig viele Netzwerke auf dem Container-Host erstellen.

3.6 Container-Registry

Die Container-Registry oder auch Docker-Registry ist eine Serveranwendung, auf der Images bereitgestellt werden, die Sie von dort beziehen können. Es gibt öffentliche Registries wie beispielsweise https://hub.docker.com. Natürlich können Sie auch Ihre eigene private Registry erzeugen.

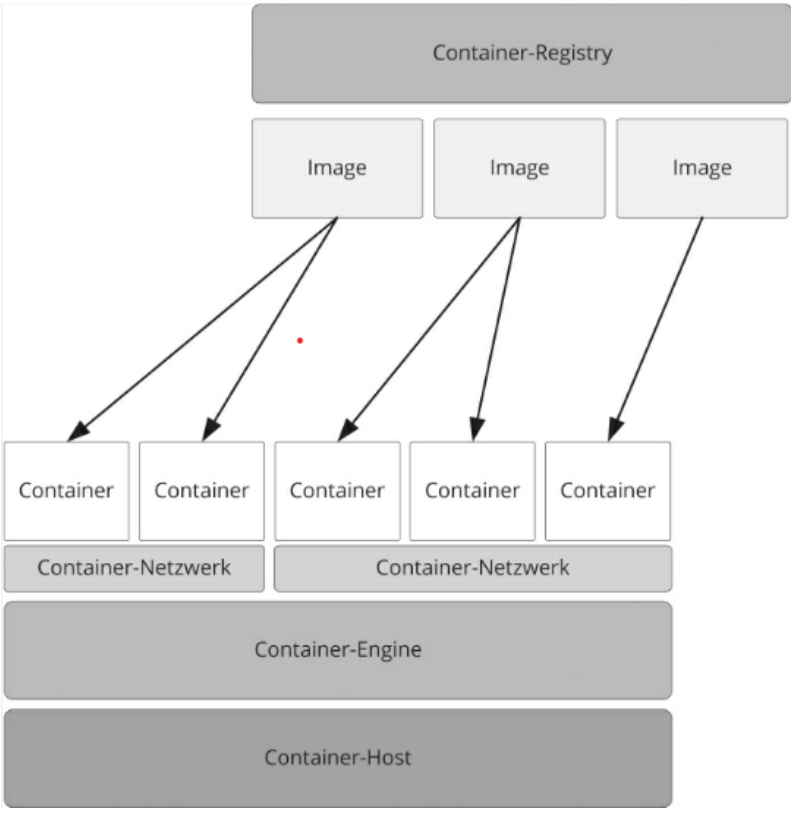

Abbildung 2: Docker Elemente

4 Vergleich Container und virtuelle Maschine

Da sich, von aussen betrachtet, Container und virtuelle Maschinen ähnlich verhalten, werden Container immer wieder mit virtuellen Maschinen verglichen oder gleichgestellt. Dies ist aber nicht richtig. Virtuelle Maschinen und Container sind zwei grundsätzlich unterschiedliche Konzepte, die aber ein ähnliches Ergebnis produzieren.

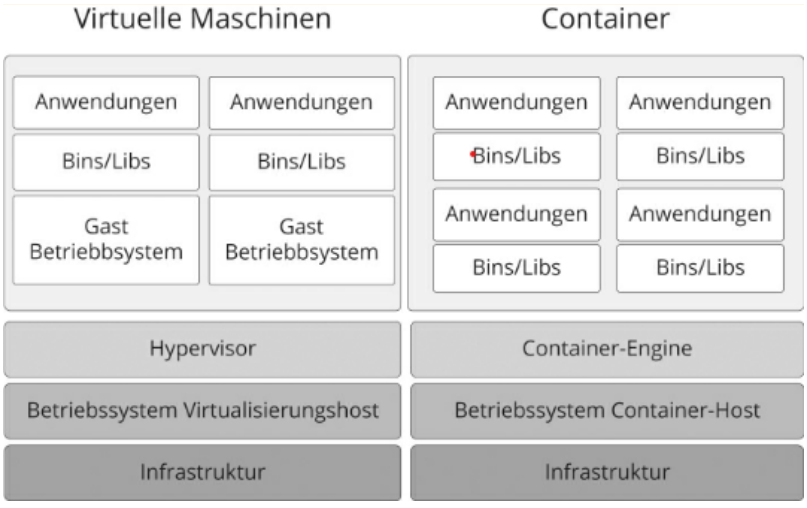

Abbildung 3: Vergleich zwischen virtuellen Maschinen und Containern

Die ersten beiden Schichten sind bei Systemen, die virtuelle Maschinen oder Container zur Verfügung stellen, gleich. Hier finden Sie in der ersten Schicht die Infrastruktur, also all das, was einen physischen Computer ausmacht wie Netzwerk, Prozessoren, Festplatten, Grafikkarten o. Ä. Die zweite Schicht bildet das Betriebssystem des Computers, auf dem die Systeme ausgeführt werden.

Ab der dritten Schicht beginnen die Unterschiede. Auf der linken Seite, bei den virtuellen Maschinen, finden Sie den Hypervisor. Das ist die Software, die sich um die virtuelle Umgebung kümmert, die die virtuellen Maschinen verwaltet. Auf der rechten Seite des Diagramms sehen Sie in der dritten Schicht die Container-Engine. Sie hat einen ähnlichen Job wie der Hypervisor – nur kümmert sie sich nicht um virtuelle Maschinen, sondern stellt die Container bereit.

Der grosse hellgraue Kasten oben im Diagramm stellt auf der linken Seite die Virtualisierungsumgebung dar, in der die virtuellen Maschinen betrieben werden, auf der rechten Seite stellt er die Containerumgebung dar, in der die Container betrieben werden. Eine virtuelle Maschine besteht aus drei Schichten, dem Gast-Betriebssystem, Binaries und Bibliotheken und schliesslich der Anwendung, die auf der Maschine betrieben wird.

Kommen wir nun zur rechten Seite, den Containern. Wie Sie in der Abbildung 2 sehen können, bestehen Container nur aus zwei Schichten – nämlich den Binaries und Bibliotheken und der eigentlichen Anwendung. Wie bereits weiter oben ausgeführt, handelt es sich bei einem Container nur um einen Prozess, der das Dateisystem abstrahiert und isoliert. Daher gibt es keine Schicht, in der ein Gast-Betriebssystem installiert ist. Im Container gibt es keine Kernel-Module, keine Treiber und keine andere hardwarenahe Software.

Das bedeutet aber wiederum auch, dass in einem Container nur das Betriebssystem des zugrunde liegenden Container-Hosts zur Verfügung steht. Auf einem Windows-Host können Sie also nur Windows-Container ausführen, auf einem Linux-Host nur Linux-Container und auf einem Macintosh-Host – nun ja, es gibt keine Macintosh-Container. - Natürlich gibt es auf Windows auch Linux-Container, und es gibt auch eine Docker-Version für den Mac.« Was soll das also? Die Erklärung ist ganz einfach. Bei diesen weiterführenden Lösungen kommt eine geschickte Kombination aus Virtualisierungsplattform und Container-Technologie zusammen.

Wie Docker arbeitet: https://www.youtube.com/watch?v=DisDkSqDCNc

4.1 Container sind unveränderlich, wohingegen virtuelle Maschinen einen bestimmten Status haben.

Grundsätzlich bedeutet das, dass die Software in Containern nicht aktualisiert wird, sondern dass, wenn eine neue Softwareversion herauskommt, der alte Container vernichtet wird und auf Basis eines Images ein neuer Container mit der neuen Softwareversion erzeugt wird. Betrachten Sie eine virtuelle Maschine, so wird die alte Software-Version auf die neue Software-Version aktualisiert, und die virtuelle Maschine ändert ihren Status.

4.2 Container haben im Vergleich zu virtuellen Maschinen eine kurze Lebenszeit.

Ein Container besteht nur so lange, wie er gebraucht wird. Benötigen Sie einen Container nicht mehr, wird er gelöscht, da es ja ziemlich einfach ist, sich aus einem Image einen neuen Container zu erstellen. Virtuelle Maschinen haben eine lange Lebensdauer, da sie ja im Prinzip nichts anderes als Software auf virtueller Hardware darstellen, zeitaufwendig zu installieren sind und somit eine kostbare Ressource darstellen.

5 Was genau macht eine Virtualisierungsumgebung

Im Prinzip stellt die Virtualisierungsumgebung einen kompletten virtuellen Computer mit virtuellem RAM, virtuellem Prozessor, virtueller Festplatte und anderer virtueller Hardware zur Verfügung. Auf dieser virtuellen Maschine können Sie, genau wie auf einer physischen Maschine, ein beliebiges Betriebssystem installieren. Virtuelle Maschinen werden oft auch als VM abgekürzt. Das Betriebssystem muss noch nicht einmal mitbekommen, dass es auf einer virtuellen und nicht auf einer physischen

Maschine läuft. Aus Sicht des Betriebssystems ist die virtuelle Maschine genauso real wie eine physische Maschine.

Da die virtuelle Maschine eine simulierte Hardware zur Verfügung stellt, kann auf der virtuellen Maschine jedes beliebige Betriebssystem installiert werden, das die simulierte Prozessorarchitektur unterstützt. So ist es nicht ungewöhnlich, dass auf einem Windows Hypervisor ein Linux als virtuelles Betriebssystem läuft oder umgekehrt. Haben Sie das Betriebssystem einmal installiert, können Sie den virtuellen PC genauso verwenden wie einen physischen PC und jede beliebige Software, auch Client-Programme wie Office, dort installieren.

5.1 Hyper-V

Hyper-V ist die hauseigene Virtualisierungslösung von Microsoft. Neben der Servervariante steht Hyper-V auch unter Windows 10/11 als Professional und Enterprise Edition zur Verfügung.

5.2 Virtual Box

Virtual Box wird von Oracle betreut und kann unter Windows, Linux, Mac OS und Solaris installiert werden.

5.3 VMWare Workstation Player

VMWare Workstation Player ist die kostenlose und abgespeckte Clientlösung der bekannten Virtualisierungssoftware VMWare. Der VMWare Player kann beispielsweise keine Snapshots erstellen.

6 Installation

Die Website https://docker.com stellt Ihnen verschiedene Docker-Abos zur Auswahl, von Free über Pro, Team bis zu Large. Für viele Entwickler reicht die kostenlose Variante aus. Für uns reicht die kostenlose Docker-Version.

Das bedeutet nicht, dass die kommerziellen Docker-Angebote für professionelle Entwicklerteams uninteressant wären: Zahlende Kunden erhalten besseren Support, unlimitierten Zugriff auf Container-Images, die Möglichkeit, selbst private Image-Repositories auf dem Docker Hub einzurichten usw.

Auch wenn die Gestaltung der Docker-Website das Gegenteil vermuten lässt: Weder zur Installation noch zur Nutzung von Docker ist eine Registrierung erforderlich. Sie benötigen erst dann einen Docker-Account bzw. eine sogenannte Docker ID, wenn Sie selbst Images im Docker Hub anbieten möchten. Ein Docker-Account vergrössert zudem die Anzahl der zulässigen Zugriffe pro Stunde auf Images im Hub.

6.1 Was ist der Docker Desktop?

Für Windows und macOS bietet Docker den sogenannten Docker Desktop an. Das ist ein Komplettpaket, das neben der Docker-Infrastruktur auch eine grafische Benutzeroberfläche umfasst.

Unter Linux müssen Sie auf den Docker Desktop verzichten. Allzu schlimm ist das nicht: Egal, ob Sie unter macOS, Windows oder Linux arbeiten – in jedem Fall werden Sie Docker überwiegend durch Kommandos steuern. Diese Kommandos funktionieren auf allen Plattformen gleich. Der Docker Desktop vereinfacht die Installation, bietet plattformspezifische Konfigurationsmöglichkeiten (von denen viele für Linux gar nicht relevant sind) und bietet in manchen Fällen mehr Bequemlichkeit. Das Programm stellt Windows- oder macOS-Fans aber keine echten Zusatzfunktionen zur Verfügung.

6.2 Installation unter Windows

Die ersten Versionen von Docker für Windows verwendeten Hyper-V als Backend. 2019 stieg Docker dann auf das *Windows Subsystem for Linux* (WSL) um. Hyper-V wird zwar noch immer unterstützt,

aber nur mehr aus Gründen der Kompatibilität mit älteren Windows-Versionen. In diesem Modul wird vorausgesetzt, dass Sie Docker in Kombination mit WSL2 verwenden. Dazu benötigen Sie eine einigermassen aktuelle 64-Bit-Installation von Windows oder eine Windows-VM (beispielsweise der Windows10_Thin-Client aus der SL2223_VMBox).

Die aktuelle Version von Docker Desktop finden Sie hier zum Download:

https://docs.docker.com/docker-for-windows/install

Die Installation verläuft in der Regel vollkommen unkompliziert. Keine einzige Option ist einzustellen! Während der Installation wird automatisch WSL2 aktiviert, falls dies auf Ihrem Rechner noch nicht der Fall sein sollte. Nach der Installation finden Sie auf dem Desktop bzw. im Startmenü einen Eintrag zum Start von Docker Desktop. Dieses Programm bietet Ihnen beim ersten Mal die Möglichkeit, eine Einführungs-Tour zumachen.

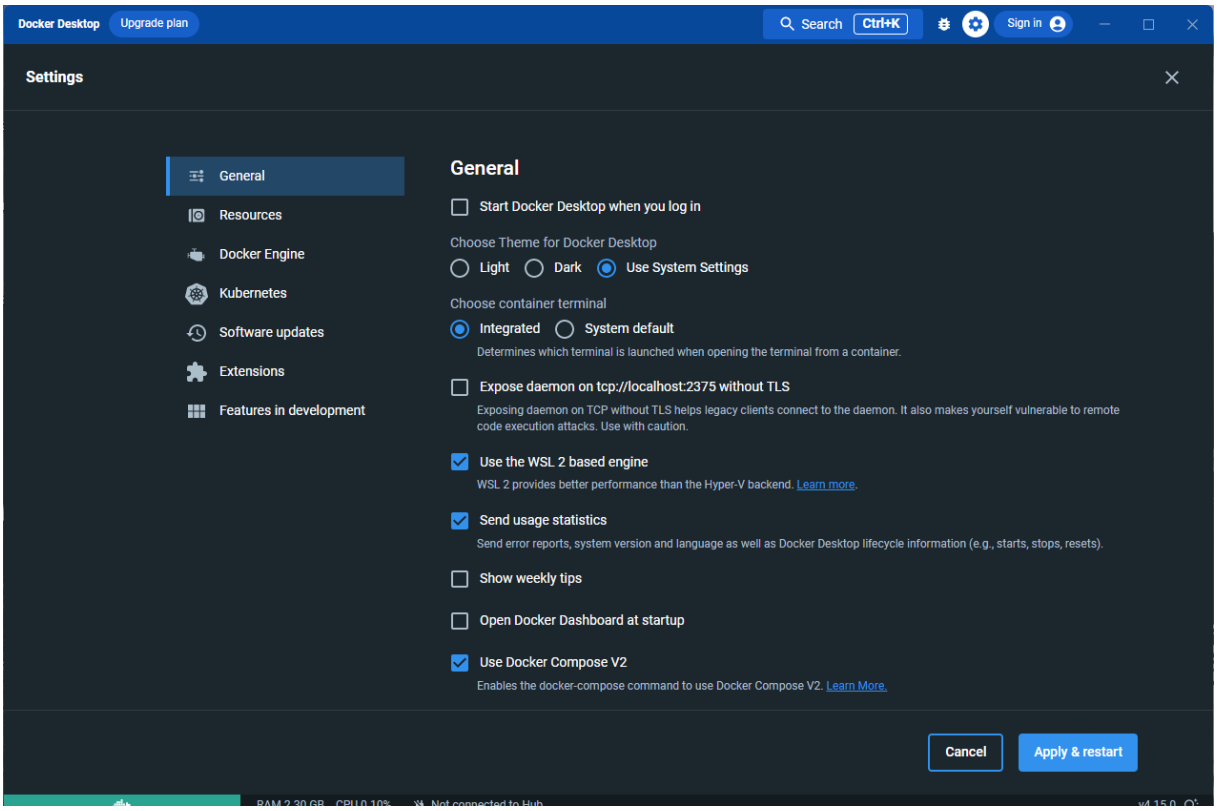

Abbildung 4: Grundeinstellungen im Docker Desktop

Die Konfiguration von WSL 2 siehe https://docs.docker.com/desktop/windows/wsl/

Ein weiteres Tutorial wie sie z.B. Ubuntu via WSL unter Windows nutzen können siehe hier.

Neben den von Docker-Desktop installierten WSL-Systeme können Sie auch im Microsoft Store nach Ubuntu suchen und den WSL hinzufügen.

6.3 Installation von Visual Studio Code

Installieren Sie Visual Studio Code auf Ihrer VM oder Laptop.

Für den Editor Visual Studio Code gibt es zwei Erweiterungen, die die Arbeit mit Docker vereinfachen: die Docker Extension , WSL und Remote Containers

Die offizielle **Docker Extension** von Microsoft bietet eine ganze Fülle von Funktionen:

- Mit einem Klick auf das Docker-Icon können Sie In der Seitenleiste (im »Explorer«) durch alle Container, Images, Volumes und Netzwerke scrollen. Per Mausklick können Sie Container starten und stoppen, Images aktualisieren (`pull`), aus Images neue Container erzeugen, ungenutzte Volumens löschen usw.
  Bei laufenden Containern können Sie sogar in das Dateisystem des Containers sehen, einzelne Dateien öffnen und direkt in VSCode ändern.
- VSCode unterstützt Sie beim Verfassen von `Dockerfiles` und Docker-Compose-Dateien durch Syntax-Highlighting und die automatische Vervollständigung von Schlüsselwörtern.
  Per Kontextmenü oder mit (F1) und DOCKER IMAGES: BUILD IMAGE können Sie ein Image zu einem `Dockerfile` erzeugen. Analog stehen beim Editieren einer Docker-Compose-Datei in der Seitenleiste Kontextmenükommandos wie COMPOSE UP zur Auswahl.

Auch die Erweiterung **Remote Containers** wurde von Microsoft entwickelt. Genau genommen handelt es sich dabei um eine Variante der wesentlich bekannteren Erweiterung `Remote SSH`, die es ermöglicht, in VSCode Dateien zu bearbeiten, die sich auf einem externen, via SSH zugänglichen Rechner befinden.

`Remote Containers` funktioniert so ähnlich. Die Erweiterung erlaubt es Ihnen, Dateien zu bearbeiten, die sich in einem (gerade laufenden) Container befinden. Neu ist allerdings das damit verbundene Konzept von `Development Containers`: Dabei wird ein Softwareentwicklungsprojekt um das Verzeichnis `.devcontainer` erweitert. Dieses Verzeichnis enthält ein `Dockerfile` und die Datei `devcontainer.json`. Diese beiden Dateien beschreiben das für die Entwicklung erforderliche Image und seine Anwendung. Dazu später mehr.

6.4 Docker Sandbox

Falls Sie mit der Installation Mühe haben, können Sie unter https://labs.play-with-docker.com/ einen Docker-Client simulieren. Um dieses Angebot nutzen zu können, müssen Sie einen kostenfreien Account unter https://hub.docker.com/ anlegen.

6.5 Der erste Container

Nach der Installation von Docker können Sie die Programme schnell testen. Ob der Docker-Daemon läuft, verrät > `docker version`. Ob Docker Compose funktioniert, verrät > `docker compose version`. Jetzt ist es an der Zeit für den ersten Container Ihrer Docker-Karriere. Für seinen Start soll Docker ein Image namens „hello-world“ vom Docker-Hub herunterladen und daraus einen Container erzeugen:

```
> docker run hello-world
```

Was war passiert? Der Docker Client hat sich mit dem Docker Daemon verbunden. Dieser hat das Image vom Docker Hub heruntergeladen. Der Docker Daemon hat aus dem Image einen neuen Container erstellt, der wiederum eine Anwendung ausführt, die den angezeigten Text ausgibt.

Nun laden wir eine Node.js-Umgebung herunter:

```
>docker run -it node
```

Führe folgenden Befehle aus:

```
>console.log("Hallo Welt")
```

```
>.exit
```

```
>docker run -it node:16
>console.log("Hallo Welt")
>.exit
```

Die wichtigsten Docker Befehle

Befehl	Funktion
`docker version`	zeigt die installierte Docker-Version an
`docker run <Image>`	startet einen Container und lädt das Image aus dem Docker-Hub
`docker container ls`	zeigt alle laufenden Container
`docker container ls -a`	zeigt alle Container, auch nicht laufende
`docker stop <container>`	hält den Container an
`docker start <container>`	startet den Container
`docker images -a`	zeigt alle Images
`docker rmi <Image>`	löscht das angegebene Image
`docker rm <container>`	löscht den angegebenen Container
docker run --help	sie erhalten Hilfe zum run-Befehl

Tabelle 1: Die wichtigsten Docker Befehle

6.6 Übungen

a) Starte in Docker einen Node.js Container. Node.js ist eine Laufzeitumgebung für JavaScript, um JavaScript ausserhalb eines Webbrowsers ausführen zu können.
Nachdem Sie den Node.js-Container gestartet haben, überprüfen Sie die Ausgaben von folgenden JavaScript-Befehlen.
   a. `3 > 2 > 1;`
   b. `typeof NaN;`
   c. `[1,2,3] + [4,5,6]`

b) Starten Sie in Docker einen neuen Ubuntu-Container.
   a. Lernen Sie das Ubuntu-Programm `tree` kennen
      i. Aktualisieren Sie den Paketmanager `apt` und installieren Sie dann das Programm `tree`
      ii. Wechseln Sie anschliessend in das Stammverzeichnis: `cd/`
      iii. Führen Sie dann den Befehl `tree` aus – was passiert, welchen Zweck erfüllt das Programm `tree`?
   b. Installieren Sie auch das Programm `cmatrix`
      i. Hier können Sie die Flag `-y` setzen, um eine manuelle Bestätigung zu überspringen
      ii. Während der Installation werden Sie mehrmals aufgefordert Zahlen zur Konfiguration einzutragen: es spielt inhaltlich keine Rolle, für welche der angegebenen Optionen Sie sich entscheiden
      iii. Was passiert dann, wenn Sie das Programm `cmatrix` ausführen?
      iv. `cmtrix` mit `Ctrl C` beenden, dann `exit` um das Ubuntu zu verlassen

7 Gute und schlechte Container-Images

Docker-Container sind nicht automatisch sicherer oder unsicherer als installierte Programme. Verwendbarkeit und Sicherheit hängen von der Qualität der eingesetzten Docker-Images ab. Wer Docker nutzt, sollte fremde Images sorgfältig prüfen – dafür muss man verstehen, wo sie eigentlich herkommen.

Ein Docker-Image ist die Grundlage für einen Container. Es enthält das auszuführende Programm und die nötigen Abhängigkeiten. Jeder, der Docker installiert hat, kann ein solches Image erstellen: Man legt ein Dockerfile mit der Bauanleitung an und führt den Befehl `docker build` aus. Wenn man das fertige Image auch auf anderen Systemen nutzen möchte, kann man ihm mit `docker tag` einen Namen im Format `benutzername/containername` geben und es mit `docker push` in eine Registry kopieren – als Standard ist bei Docker immer die Registry von docker.com eingerichtet. Wer dort ein Image abladen möchte, braucht ein Benutzerkonto, das schnell eingerichtet ist. Öffentlich verfügbare Images darf jeder kostenlos hochladen, nur für private Images muss man zahlen.

Weil es so einfach ist, eigene Werke in der Docker-Registry abzulegen und weil es keine Anforderungen oder Kontrolle gibt, landen dort nicht nur Images für den produktiven Einsatz. Nutzer laden sie aus unterschiedlichen Gründen hoch: nur zum Test, für den persönlichen Gebrauch oder gar, um Schadsoftware zu verbreiten. Immer mal wieder finden Anwender und Sicherheitsforscher Images, die neben ihrem eigentlichen Job zum Beispiel Kryptowährungen schürfen oder Hintertüren öffnen und tausendfach heruntergeladen wurden. Ein Blick unter die Haube und systematisches Vorgehen beim Auswählen von Images ist also angebracht.

7.1 Vertrauen ist gut

Die Suche nach einem Image für ein Programm sollte immer bei den sogenannten „Official Repositories“ beginnen. Diese werden von der Firma Docker Inc. selbst kuratiert und betreut und sind relativ sicher. Wer dem Paketmanager einer Linux-Distribution traut, kann das auch dem Kurator. Zu erkennen sind solche Images leicht am Namen – sie beginnen nicht mit einem Benutzernamen und einem Schrägstrich. Eine Übersicht aller offiziellen Images gibt es im Docker Hub unter https://hub.docker.com/explore. Hier finden sich viele häufig genutzte Docker Images. Darunter Webserver wie nginx, Datenbanken wie MySQL, Umgebungen für Programmiersprachen wie Node.js, aber auch fertige Anwendungen wie WordPress. Sie sind nicht nur vergleichsweise sicher, sondern im Docker Hub auch sehr gut dokumentiert. Die Macher erklären, welche Ports erreichbar sein sollen, in welche Ordner Konfigurations- und Nutzdaten gelegt werden, und liefern Beispiele mit.

Meist sind die im Container laufenden Programme in den offiziellen Images über Umgebungs-Variablen gut konfigurierbar. Ausserdem stehen sie fast immer in verschiedenen Ausführungen, sogenannten **Tags**, zur Verfügung. Die Tags unterscheiden sich in den Programmversionen und dem verwendeten Betriebssystem. Wer nichts angibt, bekommt den **Tag „latest“** – nicht immer ist das auch der Beste für jedes Einsatzgebiet.

7.2 Übungen

a) Starten Sie unter Windows PowerShell und geben Sie folgendes Kommando ein:
```
> docker search hello-world
```
Diskutieren Sie das Suchergebnis mit dem Tandem-Partner. Beachten Sie auch die Sterne!

b) Bauen Sie einen Container für «debian» Linux mit dem tag «jessie» und überprüfen Sie diesen.

7.3 Das Fundament

Nahezu jedes Image basiert auf dem Userland einer Linux-Distribution und jedes Dockerfile muss mit der Anweisung `FROM`, gefolgt vom Namen eines Basis-Images, beginnen. Meistens nutzt man hier eins der Betriebssysteme aus der Liste der offiziellen Images wie „debian“, „ubuntu“ oder „alpine“. Richtige und falsche Basis-Images gibt es nicht – gute Images gibt es aber in mehreren Varianten. Das spart vor allem Festplattenplatz für den Anwender – Docker geht den Download eines Images

schichtweise an und lädt nur das, was es noch nicht kennt. Hat man bereits einen Container mit debian-Basis im Einsatz, wird diese beim zweiten Container auf gleichem Fundament wiederverwendet.

Viele der offiziellen Images gibt es in einer „alpine“-Version. Alpine ist ein stark abgespecktes Linux (Details siehe https://alpinelinux.org/), optimiert auf Sicherheit und geringe Grösse. Anders als Ubuntu oder Debian soll es gar nicht als eigenständiges Server- oder Desktop-Betriebssystem funktionieren und enthält nur das Nötigste für den Container-Einsatz. Je weniger Komponenten ein Basis-Image mitschleppt, desto weniger Angriffsfläche bietet es auch. Auch von Debian gibt es die sparsame Container-Variante „debian:stable-slim“.

7.4 Wahlfreiheit

Auch auf den folgenden Schichten kann der Image-Ersteller in viele Richtungen abbiegen. Das offizielle WordPress-Image gibt es beispielsweise mit einem Apache-Webserver oder mit nginx, mit verschiedenen PHP-Versionen und jeweils mit Alpine und Debian-Grundlage.

Welche Tags ein offizielles Image mitbringt, hängt stark von der Software ab. Grundsätzlich gilt: Wer Tag-Bezeichnungen wie „latest“ ohne konkrete Versionsnummern verwendet, bekommt immer die aktuelle Version – das führt aber spätestens dann zu Problemen, wenn plötzlich eine nicht mehr kompatible neue Version zu „latest“ wird. Besser ist es, eine konkrete Versionsnummer festzulegen.

7.5 Gute Umgangsformen

Beim Bau von Dockerfiles gibt es einige Regeln, die den Anwendern später das Leben erleichtern und sehr gute Container ausmachen. Die Grundregel lautet „ein Prozess pro Container“. Das ist der Prozess, der am Ende des Dockerfiles mit `CMD` aufgerufen wird. Ein WordPress-Container zum Beispiel sollte nicht auch die MySQL-Datenbank enthalten – das sollte ein eigener Container sein. Hält man sich an die Ein-Prozess-Regel, kann Docker beim Start des Containers mit dem Parameter `--restart unless-stopped` dafür sorgen, dass der Container immer neugestartet wird, wenn der Prozess einmal aussteigt.

Da jeder Aufruf von `RUN`, `COPY` und `ADD` beim Bau des Images eine eigene Schicht erzeugt, sollte man Aufrufe möglichst zusammenfassen. Ein gutes Dockerfile ruft nicht für jede Paketinstallation `RUN` auf, sondern fasst alle Pakete zu einer Zeile zusammen und ruft den Paketmanager nur einmalig auf.

7.6 Gute Images finden

1) Überprüfen Sie, ob es ein offizielles Image für die Software gibt.
2) Wählen Sie einen geeigneten Tag. Abgespeckte Images sind kleiner und bieten weniger Angriffsfläche.
3) Gibt es kein offizielles Image, schauen Sie sich direkt beim Entwickler der Software um. Einige bieten eigene Docker-Images an. Achten Sie auf „Verified Publisher“.
4) Gibt es keine Images der Entwickler, suchen Sie im Hub oder in der Suchmaschine.
5) Abrufzahlen, Dokumentation und Sterne geben einen Hinweis, ob das Projekt aktiv benutzt wird.
6) Kennen Sie den Ersteller nicht, bevorzugen Sie Images mit automatischen Builds und schauen Sie ins Dockerfile.

7.7 Übungsaugabe

Erweitern Sie das Shell-Skript so, dass die Frage «*Whats your name?*» ausgegeben wird und anschliessend auf eine Tastatureingabe gewartet wird.

Nach der Tastatureingabe wird die Zeichenfolge: *Hello <+ eingegebener Text>* im Terminal angezeigt.

Wenn das funktioniert, erweitern Sie das Skript so, dass beliebig oft hintereinander ein Name eingegeben werden kann und danach jeweils der Text «*Hello <+ name>*» angezeigt wird.

Erst die Eingabe von «*q*» soll die Ausführung des Containers beenden.

Die folgenden bash Shell-Kommandos werden für diese Aufgabe benötigt:

Kommentare beginnen mit dem Hashtag (#) Zeichen. Alles, was rechts davon steht, wird ignoriert:
`# ein Kommentar`

Ausgabe eines Textes:
`echo <Textplatzhalter>` Beispiel: `echo "Hello M347"`

Wartet auf eine Tastatureingabe und liest die Eingabe in eine Systemvariable:
`read <Variablenplatzhalter>` Beispiel: `read INPUT`

Die while-Schleife:

```
while [Bedingung]
do
      <Kommando 1>
      <Kommando 2>
      …
done
```

Beispiel:

```
n=10
z=0
while [ $n -gt 0 ]
do
      z=`expr $z + $n`
      echo $z
      n=`expr $n - 1`
done
```

Bedingung für Zeichenketten:

```
<Textplatzhalter 1> = <Textplatzhalter 2> # True bei gleichen Zeichenketten
<Textplatzhalter 1> != <Textplatzhalter 2> # True bei ungleichen Zeichenk.
```

Wertzuweisung an Variablen, Beispiel:

```
a="Hallo M347"
echo $a
```

Variable auslesen: Dollarzeichen vor Variablenname (siehe oben)

7.8 Ausführungsarten von Docker-Container

Es gibt 3 mögliche Modi, wie Sie Docker-Container verwenden können.

7.8.1 Task-Container

Unter einem Task-Container versteht man einen Container, in dem Code mit allen Abhängigkeiten gekapselt ist und der nur dazu verwendet wird, diesen Code auszuführen.

7.8.2 Interaktive Container

Ein interaktiver Container wird mit dem Parameter -it gestartet. So können Sie sich interaktiv mit einer Anwendung im Container, beispielsweise einer Kommandozeile verbinden. Interaktive Container

sind sehr gut dazu geeignet, um auszuprobieren, was Sie innerhalb eines Containers so anstellen können. So können Sie so beispielsweise Befehle testen, die Sie später in einem Dockerfile verwenden möchten.

7.8.3 Hintergrund-Container

Hintergrund-Container, die auch als Background-Container bezeichnet werden, werden mit dem Parameter -d gestartet. Eine Grundvoraussetzung für einen Hintergrund-Container ist, dass das Programm, das im Hintergrund-Container läuft, nicht automatisch beendet wird, weil ansonsten auch der Container beendet wird. Besonders gut sind Serveranwendungen wie Webserver oder Datenbankserver für Hintergrund-Container geeignet, da diese über eine Endlosschleife verfügen, die die Serveranwendung und damit auch den Container am Leben hält. Wird der Prozess beendet, beispielsweise durch einen Fehler der Serveranwendung, so wird auch der Container beendet.

7.9 Image in Docker Hub veröffentlichen

```
> docker login
> docker push <Docker-ID>/hello-docker
```

… (detached-Modus – starten und stoppen)

7.10 Webseite mit NGINX

```
> docker run --name m347-nginx -d -p 8080:80 nginx
```

Erstellen Sie unter Ihrem Benutzerverzeichnis ein Verzeichnis mit dem Namen "Hello-Web" und wechseln Sie in dieses Verzeichnis.

Erstellen Sie ein Unterverzeichnis namens html im Verzeichnis.

Wir erstellen nun im Unterverzeichnis eine einfache HTML-Datei mit dem Namen index.html als Startseite für unsere eigene Webseite.

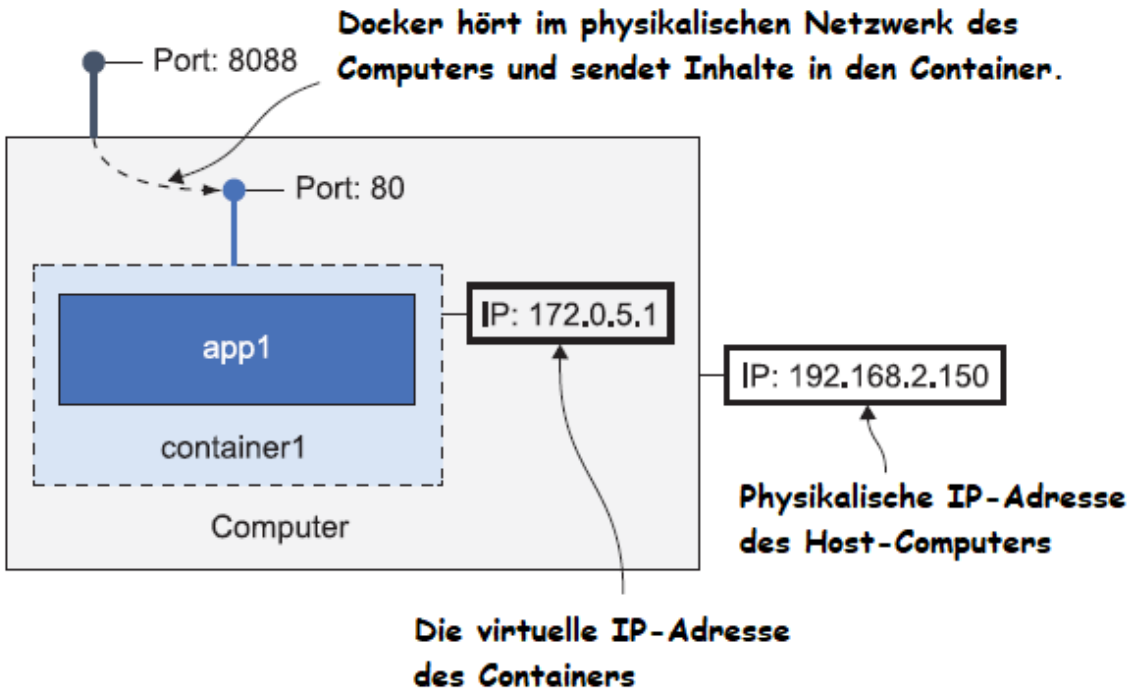

Abbildung 5: Eigene Webseite

7.11 Übung Eigene Webseite

Starten Sie den Website-Container, wie in diesem Kapitel eingeführt. Ersetzen Sie aber die index.html Datei durch eine eigene. Sie können einen beliebigen Inhalt verwenden.

7.12 Verstehen, wie Docker Container ausgeführt werden.

Wir wollen noch ein wenig mehr Hintergrundwissen anschauen, damit Sie ein solides Verständnis dafür haben was tatsächlich passiert, wenn Sie Anwendungen mit Docker ausführen. Installieren von Docker und das Ausführen von Containern ist täuschend einfach - es sind tatsächlich verschiedene Komponenten beteiligt.

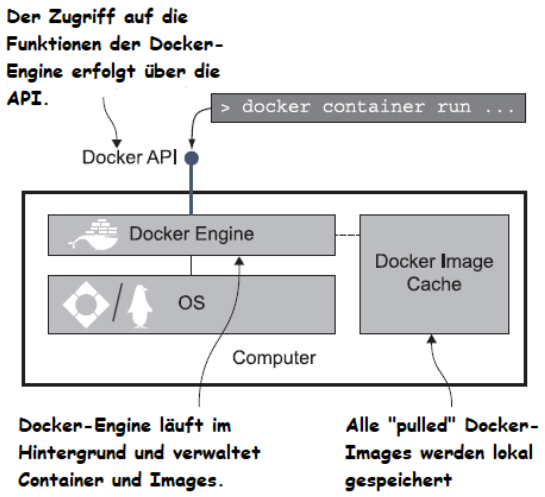

Abbildung 6: Docker Komponenten

- Die Docker Engine ist die Verwaltungskomponente von Docker. Sie kümmert sich um den lokalen Image-Cache, lädt Images herunter, wenn Sie sie brauchen, und verwendet sie wieder, wenn sie bereits heruntergeladen wurden. Sie arbeitet auch mit dem Betriebssystem zusammen, um Container, virtuelle Netzwerke und alle anderen Docker-Ressourcen zu erstellen. Die Engine ist ein Hintergrundprozess, der immer läuft (wie ein Linux-Daemon oder ein Windows-Dienst).
- Die Docker-Engine stellt alle Funktionen über die Docker-API zur Verfügung, die lediglich eine standardmässige HTTP-basierte REST-API ist. Sie können die Engine so konfigurieren, dass die API nur vom lokalen Computer aus zugänglich ist (dies ist die Standardeinstellung), oder sie für andere Computer in Ihrem Netzwerk verfügbar machen.
- Die Docker-Befehlszeilenschnittstelle *«Docker command-line interface»* (CLI) ist ein Client der Docker-API. Wenn Sie Docker-Befehle ausführen, sendet die CLI sie an die Docker-API, und die Docker-Engine erledigt die Arbeit.

Es ist gut, die Architektur von Docker zu verstehen. Die einzige Möglichkeit mit der Docker-Engine zu interagieren, erfolgt über die API, und es gibt verschiedene Optionen für den Zugang zur API. Die CLI funktioniert durch das Senden von Anfragen an die API.

Bisher haben wir die CLI verwendet, um Container auf demselben Computer zu verwalten, auf dem Docker ausgeführt wird, aber Sie können Ihre CLI auch auf die API auf einen entfernten Computer richten, auf dem Docker läuft und Container auf diesem Rechner steuern - so werden Sie Container in verschiedenen Umgebungen, z. B. Ihre Build-Server, Test- und Produktionsumgebungen verwalten können. Die Docker-API ist auf allen Betriebssystemen identisch, sodass Sie die CLI auf Ihrem

Windows-Laptop verwenden können, um Container auf Ihrem Raspberry Pi oder auf einem Linux-Server in der Cloud verwalten.

Die Docker-API hat eine veröffentlichte Spezifikation, und die Docker-CLI ist nicht der einzige Client. Es gibt mehrere grafische Benutzeroberflächen (Bsp. https://www.portainer.io/ siehe unten), die eine Verbindung zur Docker-API herstellen und Ihnen eine visuelle Möglichkeit bieten, um mit Ihren Containern zu interagieren. Die API stellt alle Details über Container, Images und die anderen von Docker verwalteten Ressourcen zur Verfügung.

7.13 Portainer

Portainer hilft bei der Verwaltung von Docker, Docker Swarm, Kubernetes und Azure ACI. Das Open-Source-Programm kann wahlweise als kostenlose Community Edition oder in einer Business Edition mit zusätzlichen Funktionen genutzt werden.

https://www.portainer.io/

Portainer unterstützt aktuell nur Ubuntu und Windows.

Bezeichnenderweise wird Portainer in Form eines Docker-Containers installiert. Normalerweise sind nur die ersten zwei Kommandos erforderlich. Damit wird Portainer als Webanwendung eingerichtet, die über Port 9000 genutzt werden kann.

```
docker volume create portainer_data
```

```
docker run -d -p 8000:8000 -p 9000:9000 --name=portainer `
--restart=always `
-v /var/run/docker.sock:/var/run/docker.sock `
-v portainer_data:/data `
portainer/portainer-ce
```

In Linux verwenden Sie statt ` (Backtick-Zeicen) ein Backslash \

Siehe auch: https://docs.portainer.io/start/install/server/docker

Nach dem Start des Portainer-Containers (nettes Wortspiel!) öffnen Sie im Webbrowser die Seite localhost:9000 und legen für den Benutzer admin ein Passwort fest. Im nächsten Schritt loggen Sie sich ein, geben an, dass Sie Docker (und nicht Docker Swarm oder Kubernetes) administrieren wollen. Damit gelangen Sie in die Weboberfläche, in der Sie nun Container, Stacks (üblicherweise durch docker compose erzeugte Container-Gruppen), Images, Volumes usw. administrieren können.

7.13.1 Aufräumen

Falls Portainer Sie nicht überzeugen kann, erfolgt die Deinstallation wie folgt:

```
docker stop portainer
docker rm portainer
docker volume rm portainer_data
```

7.14 Das Format von Dockerfiles

Die Zeilen eines Dockerfiles können entweder ein Kommando enthalten oder mit einer Raute (#) beginnen, dann handelt es sich um Kommentarzeilen. Wenn eine Zeile keine Kommentarzeile ist, dann ist sie immer nach dem Schema ANWEISUNG Argumente aufgebaut.

Docker kennt eine Reihe von Anweisungen für den Aufbau eines Container-Images. Diese Anweisungen ermöglichen Ihnen das Kopieren von Daten in ein Image, die Definition der beim Container-Start

auszuführenden Anwendung und die Modifikation einzelner Dateien und Verzeichnisse in Ihrem Image. Die Befehle, die Sie bei der Definition des Images angeben, werden dabei im Build-Prozess der Reihe nach von oben nach unten abgearbeitet. Die wichtigsten Befehle, die Ihnen hierfür zur Verfügung stehen, sind:

Anweisung	Bedeutung
ADD	Kopiert Dateien in das Dateisystem des Images.
CMD	Führt das angegebene Kommando beim Container-Start aus.
COPY	Kopiert Dateien aus dem Projektverzeichnis in das Image.
ENTRYPOINT	Führt das angegebene Kommando immer beim Container-Start aus.
ENV	Setzt eine Umgebungsvariable.
EXPOSE	Gibt die aktiven Ports des Containers an.
FROM	Gibt das Basis-Image an.
LABEL	Legt eine Zeichenkette fest.
RUN	Führt das angegebene Kommando aus.
USER	Gibt den Account für RUN, CMD und ENTRYPOINT an.
VOLUME	Gibt Volume-Verzeichnisse an.
WORKDIR	Legt das Arbeitsverzeichnis für RUN, CMD, COPY etc. fest.

Tabelle 2: Wichtige Dockerfile-Schlüsselwörter

8 Kubernetes

Als Anfang 2013 Docker auf den Markt kam, dachten viele noch an einen neuen Hype, der schnell wieder in den Tiefen des Internets verschwindet. Mittlerweile hat sich Docker jedoch breitflächig durchgesetzt und die IT-Landschaft grundlegend verändert. Docker ist nicht mehr wegzudenken und geniesst eine immer grösser werdende Beliebtheit.

Enormen Einfluss auf diesen Erfolg hatte von Anfang an unter anderem Google, das mit Kubernetes ein Open-Source-Projekt zur Administration mehrerer Docker-Container veröffentlicht hat. Doch was ist Kubernetes genau und wie funktioniert es? Ich möchte in der nachfolgenden Einführung einen Überblick über Docker und Kubernetes geben und so ein Grundverständnis der Funktionalitäten schaffen.

8.1 Von Docker nach Kubernetes

Das Konzept der Container existiert seit über einem Jahrzehnt und wurde bereits in vielen bekannten Unix-basierten Betriebssystemen wie zum Beispiel Linux, Solaris oder FreeBSD eingesetzt. Jedoch war es erst **Docker**, welches die Containertechnologie für Entwickler und den IT-Betrieb im täglichen Umgang zugänglich und handhabbar gemacht hat. Docker bewies die Portabilität und Skalierbarkeit von Anwendungen, weshalb immer mehr Entwickler und der IT-Betrieb dazu übergingen, die Container für das Bauen von Software und Auflösen von Abhängigkeiten zu verwenden. Auch in DevOps-Prozessen spielen Container eine entscheidende Rolle. Sie sind zu einem integralen Bestandteil automatisierter Software-Builds und der kontinuierlichen Integration und Bereitstellung (Continuous Integration/Continuous Delivery) von Pipelines geworden.

Während man in kleinen Infrastrukturen ohne Probleme das Lifecycle-Management der Container manuell handhaben kann, verliert man in verteilten Rechenzentren mit hunderten Containern schnell den Überblick. Das Verwalten vieler unterschiedlicher Container in Entwicklungs- und Produktionsumgebungen mit unterschiedlichen Netzwerk- und Festplattenanforderungen erforderte eine Lösung. Eine der bekanntesten Lösungen ist **Kubernetes**.

Kubernetes ist ein Open-Source-System für das automatische Deployment, Skalieren und Verwalten von containerisierten Anwendungen. So ist Kubernetes auf seiner eigenen Webseite beschrieben. Urheber von Kubernetes ist Google, das Kubernetes als ein Google-Infrastructure-for-Everybody-Else-(GIFEE-)Projekt vollständig Open Source auf GitHub zur Verfügung gestellt hat. Neben Google als initialem und Hauptbeitragendem lebt das Projekt von vielen individuellen Entwicklern sowie beitragenden Firmen wie zum Beispiel Red Hat, Microsoft, NTT DATA oder Docker selbst. Kubernetes, auch bekannt unter dem Kürzel „k8s“, entstand nicht plötzlich aus dem Nichts. Google arbeitete bereits seit fünfzehn Jahren an einer Plattform zur Verwaltung von Containern. Damals hiess das Projekt noch „Borg“, wurde dann zu „Omega“ umbenannt und ist schliesslich als Open-Source-Lösung unter dem Namen „Kubernetes“ erschienen.

8.2 Die Kubernetes-Architektur

Master: Wie fast alle verteilten IT-Plattformen besteht auch Kubernetes aus mindestens einem Master und mehreren Nodes. Es ist möglich, beide Komponenten auf einem einzelnen System zu betreiben. Das wird unter anderem bei Minikube, einer lokalen Testumgebung, basierend auf VirtualBox, so zur Verfügung gestellt. Der Master verwaltet und veröffentlicht die Schnittstelle (API), über die er angesteuert werden kann, und überwacht den Zustand des vollständigen Clusters. Darüber hinaus plant und steuert der Master auch das Deployment auf den jeweiligen Nodes. Der Master kann zum Beispiel aus Gründen der Hochverfügbarkeit auch auf mehreren Systemen betrieben und hinter einen Load Balancer, der in Kubernetes integriert ist, gesetzt werden. Der Master definiert sich durch folgende drei Komponenten:

- Der **API-Server** ist die zentrale Anlaufstelle für alle Befehle des gesamten Clusters und erlaubt dem Administrator die Konfiguration der einzelnen Kubernetes-Objekte. Der API-Server überwacht ausserdem die Konsistenz zwischen den Parametern in etcd und den bereitgestellten Containern. Mit anderen Worten validiert und konfiguriert der API-Server die Daten der Pods, Services und ReplicaSets. Er weist Pods den einzelnen Nodes zu und synchronisiert die Pod-Information mit der Service-Konfiguration.
- Der **Service Controller Manager** verarbeitet die in den jeweiligen Replikationsaufgaben definierten Replikationsprozesse. Die dazu benötigten Informationen werden in der etcd festgehalten, die durch den Controller Manager auf Änderungen hin überwacht wird. Bei einer Änderung liest der Controller Manager die Informationen aus der Datenbank und führt die nötigen Schritte aus, um den gewünschten Zustand zu erreichen, zum Beispiel das Skalieren eines ReplicaSet oder das Löschen eines Pods. Auch der Controller Manager bedient sich hierbei des API.
- Der **Scheduler** organisiert innerhalb des Clusters die Lastverteilung auf einzelne Nodes. Das erreicht er, indem er die aktuellen Anforderungen erfasst, den aktuellen Zustand aller Nodes analysiert und schliesslich die Last auf einen oder mehrere Nodes neu verteilt.

kubectl ist ein Kommandozeilenprogramm, das für alle gängigen Betriebssysteme zur Verfügung steht. Es wird für die Kommunikation mit dem Kubernetes-API-Server benötigt. Darüber lassen sich alle verfügbaren Befehle des API-Servers ausführen, wie zum Beispiel das Hinzufügen und Löschen von Pods, ReplicaSets oder Services. Ausserdem kann man darüber auch auf die Logdateien zugreifen und den generellen Zustand des Kubernetes-Clusters überprüfen.

etcd ist eine verteilte Schlüssel-Wert-Datenbank von CoreOS, die als Single Source of Truth (SSOT) für alle Objekte innerhalb des Kubernetes-Clusters zur Verfügung steht. Diese Datenbank dient als Abstraktion zwischen den einzelnen Anwendungen und der eigentlichen Infrastruktur. Primär wird dieser Speicher durch den Master verwendet, um unterschiedliche Informationen zu den Nodes, Pods und den darin befindlichen Containern zu erhalten.

Nodes, auch Worker Nodes genannt, dienen nicht nur dem Master als Kommunikationspartner, sie dienen vielmehr den einzelnen Containern und stellen diesen CPU-, Arbeitsspeicher-, Netzwerk- und Festplattenressourcen zur Verfügung. Ein Node kann ein dediziertes System in einem Rechenzentrum, aber auch eine virtuelle Maschine (VM) bei einem Cloud-Provider sein. Auch das Mischen von virtuellen Maschinen und dedizierten Systemen über mehrere Rechenzentren hinweg ist innerhalb eines Clusters möglich. Auf einem Node sind für gewöhnlich folgende Dienste zu finden:

- Der Dienst **kubelet**, der den API-Dienst des Masters überwacht und sicherstellt, dass sich alle auf dem Node befindlichen Container im gewünschten Zustand befinden.
- Der Dienst **kube-proxy** dient den auf dem Node befindlichen Pods als einfacher Load Balancer und Netzwerk-Proxy.

Ein **Pod** besteht aus einem oder mehreren Containern. Pods stellen eine logische Gruppe für Container dar und stellen den einzelnen Containern die zugeteilten Ressourcen zur Verfügung. Dies ermöglicht, mehrere voneinander abhängige Container gemeinsam lauffähig zu halten. Pods können mittels ReplicaSets während der Laufzeit skaliert, also in ihrer Anzahl vergrössert oder verkleinert, werden. Pods sind die kleinste deploybare Einheit. Dabei wird die Konfiguration der einzelnen Kubernetes-Objekte über den Master abgewickelt. Dieser entscheidet anhand der konfigurierten Ressourcenanforderungen und der -verfügbarkeit auf den Nodes automatisch, auf welchem Node der Pod zur Verfügung gestellt wird. Der aktive Node lädt anschliessend von der konfigurierten Container Image Registry das Image und koordiniert die weitere Inbetriebnahme des Containers.

Namespaces helfen dabei, unterschiedliche Benutzer, Projekte oder Umgebungen wie zum Beispiel produktiv und Entwicklung voneinander zu trennen. Ein Namespace ist nur ein Präfix vor dem Namen einer Ressource und verhindert die Kollision von gleichen Objektnamen. Durch Namespaces kann es zum Beispiel jeweils einen Service backend-API geben, der mit den eben erwähnten Namespaces in produktiv:backend-api und entwicklung:backend-api unterschieden wird. In zukünftigen Kubernetes-Versionen werden Objekte eines Namespace standardmässig die gleichen Access Control Policies erhalten, die anschliessend modifiziert werden können.

Die **ReplicaSets** kümmern sich darum, dass zum initialen Start und während der Laufzeit immer die gewünschte Anzahl an Pods zur Verfügung steht. Standardmässig ist ein Pod beziehungsweise ReplicaSet nicht von einem anderen Pod oder einem öffentlichen Interface aus zugänglich. Durch die Nutzung von Services können diese intern wie extern veröffentlicht werden.

Services: Dadurch, dass Pods zu jedem Zeitpunkt beendet werden können und nie wiederhergestellt, sondern durch das ReplicaSet dynamisch neu generiert werden, kann sich jederzeit die IP-Adresse des Pods verändern. Dies führt zu dem Problem, dass eine Gruppe von Pods, die zum Beispiel für die Datenbank zuständig ist, nicht über eine zentrale, gleichbleibende Adresse erreicht werden kann. Die Lösung dazu sind die Kubernetes Services. Diese erhalten eine feste Adresse, unter der sie erreichbar sind, und ermöglichen durch das Zuweisen von frei definierbaren Schlüssel-Wert-Paaren, sogenannten Labels, einen oder mehrere Pods mit einem Service zu verknüpfen. Hierbei wird dem Service der gewünschte Filter hinterlegt, wie zum Beispiel *app:webseite*, und alle Pods mit dem gleichen Label werden automatisch während der Laufzeit dem Service hinzugefügt. Zusätzlich bieten Services auch Load Balancing für die Pods, sodass nicht ein Pod auf einem Node alle Anfragen verarbeiten muss.

Labels sind Schlüssel-Wert-Paare, die an einzelne Objekte wie zum Beispiel Services oder Pods gebunden werden. Labels werden für die Organisation und Gruppierung von einzelnen Kubernetes-Objekten verwendet. Sie können zu jedem Zeitpunkt hinzugefügt, verändert oder ganz gelöscht werden. Ein Objekt muss kein Label erhalten, der Schlüssel hingegen muss pro Objekt gleich sein. In einem

Namespace kann es zum Beispiel *app:backend* und *app:frontend* geben. Einem Objekt kann hierbei nur eines der beiden _app-_Labels hinzugefügt werden, es kann nicht beides sein.

Ein **Volume** ist im Grunde genommen ein Verzeichnis, das von einem Container verwendet werden kann. Dabei wird das Volume in den Container eingebunden (mount) und dient dort als persistenter Speicher für die Anwendungsdaten. Kubernetes unterstützt zurzeit 26 verschiedene Typen von Volumes, darunter unter anderem Cloud Storages wie Amazon Elastic Block Store, Azure Disk Storage und GCE Persistent Disk, aber auch andere Arten wie *gitRepo*, *hostPath*, *iscsi* und *persistentVolumeClaim*, um nur ein paar aus der Liste zu nennen.

Am einfachsten ist hierbei das *hostPath* Volume, das ein Verzeichnis oder eine Datei des Host-Node-Dateisystems mit dem Pod verbindet. Darüber können zum Beispiel statische HTML-Dateien zur Verfügung gestellt werden oder eine Konfigurationsdatei, die von einem Docker Image benötigt wird.

Ein **Secret** ist, wie der Name schon sagt, ein Geheimnis innerhalb von Kubernetes. Es kann viele unterschiedliche Informationen enthalten, etwa SSL-Zertifikate, SSH-Schlüssel oder Benutzernamen und Passwörter für die Authentifizierung. Diese Secrets sind standardmässig nicht von den Pods aus zugänglich, sie müssen zum Beispiel über die Label einem Pod zugewiesen werden und können dort als Umgebungsvariable oder als Datei in einem zugewiesenen *mountPath* zur Verfügung gestellt werden.

Helm: Eine optionale Komponente, die nicht zur Grundausführung von Kubernetes gehört, hier aber dennoch eine kurze Erwähnung finden sollte, ist Helm. Helm ist ein Paketmanager für Kubernetes, um komplexe Anwendungen, Plug-ins oder andere Erweiterungen bequem installieren und aktualisieren zu können. Unter dem Namen Helm Charts oder Kubeapps findet man z. B. Grafana oder den Continuous-Delivery-Server GoCD.

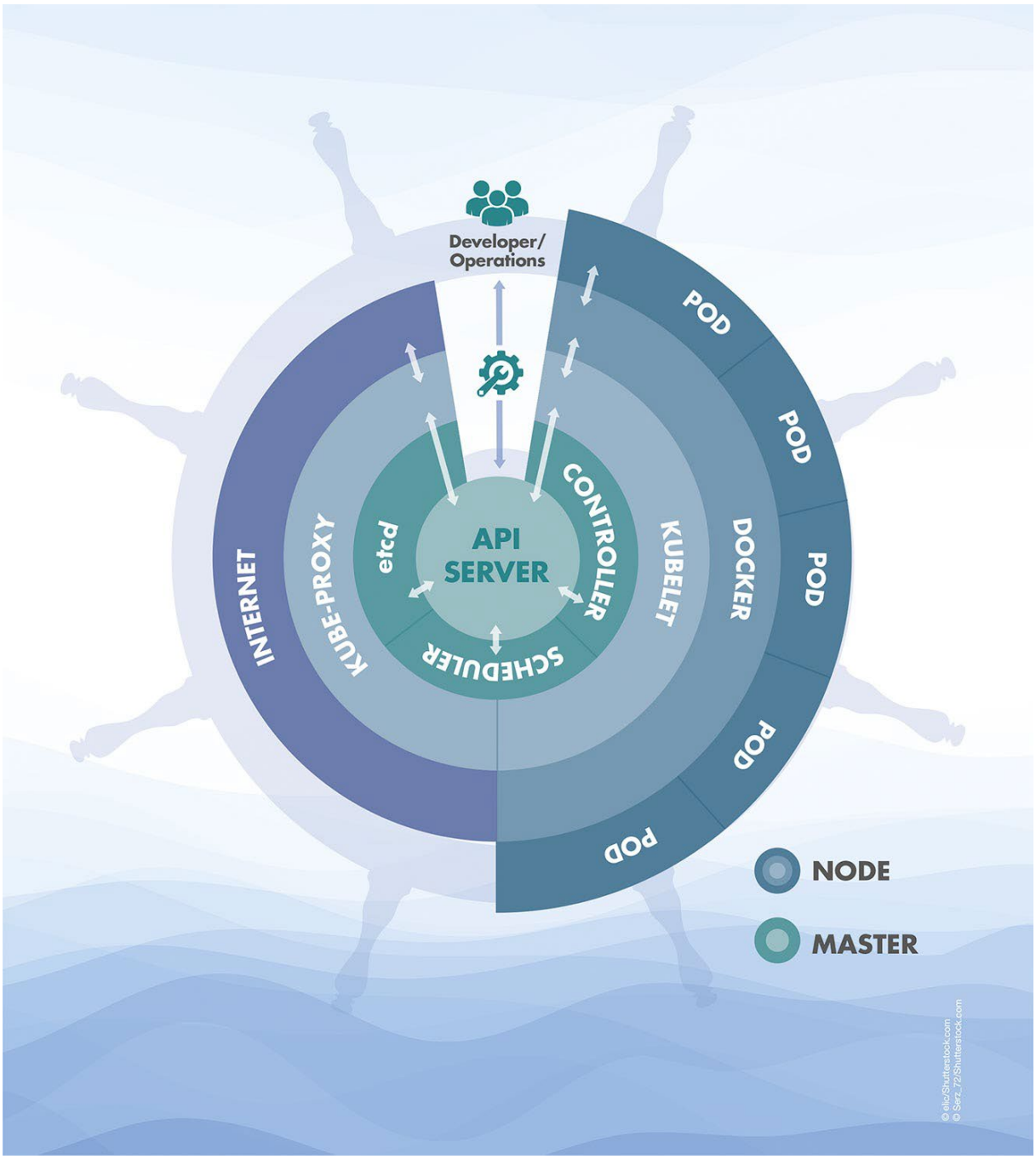

Abbildung 7: Kubernetes auf einen Blick

9 Literaturverzeichnis

- Docker docs: https://docs.docker.com/
- Kubernetes Documentation: https://kubernetes.io/docs/home/
- Kubernetes API: https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.20/
- Kubernetes Konzepte: https://kubernetes.io/docs/concepts/
- Heise online: https://www.heise.de/
- Docker für Dummies, Frank Geisler und Benjamin Kettner, Wiley-VCH Verlag, 2019
- Docker, Bernd Öggli und Michael Kofler, Reinwerk Verlag, 2021
- Kubernetes, Brendan Burns – Joe Beda – Kelsey Hightower, dpunkt Verlag, 2021
- Kubernetes in Action, Marko Lukša, Manning Verlag, 2017
- Skalierbare Container-Infrastrukturen, Oliver Liebel, Rheinwerk-Verlag, 2021

10 Abbildungsverzeichnis

11 Tabellenverzeichnis