IN DIESEM KAPITEL

- Was genau ist Docker?
- Grundlegende Elemente in Docker
- Unterschiede zwischen virtuellen Maschinen und Containern
- Vorteile und Nachteile von Containern gegenüber virtuellen Maschinen
- Einsatzgebiete von Docker

Kapitel 1
Container oder VM?

Containertechnologien und insbesondere *Docker* sind in aller Munde. In diesem Kapitel werden Sie erfahren, was es mit der Containertechnologie auf sich hat und wie sich diese von virtuellen Maschinen unterscheidet. Danach lernen Sie etwas darüber, welche Vorteile Container gegenüber virtuellen Maschinen haben und welche Einsatzgebiete es für Container gibt.

Was genau ist Docker?

Wenn Sie fünf IT-Experten fragen, was Docker bzw. die Containertechnologie ist, bekommen Sie mindestens sechs verschiedene Antworten. Die einen sehen Docker als virtuelle Maschine V 2.0 an, andere erklären, dass es sich bei Docker um eine Technologie handele, die das *Dateisystem* abstrahiere. Um etwas Klarheit zu schaffen, schauen Sie sich einmal die offizielle Beschreibung auf der Website des Herstellers an.

Container gewährleisten die Trennung und Verwaltung der auf einem Rechner genutzten Ressourcen. Das beinhaltet laut Aussage der Entwickler: Code, Laufzeitmodul, Systemwerkzeuge, Systembibliotheken – alles, was auf einem Rechner installiert werden kann. Das garantiert, dass die Software immer ausgeführt werden kann, egal auf welcher Umgebung.

`https://de.wikipedia.org/wiki/Docker_(Software)`

Schauen Sie sich die Definition etwas genauer an, fällt auf, dass es offenbar darum geht, Software zu kapseln, also vom Haupt-Betriebssystem zu trennen. Dies geschieht über die Kapselung des Dateisystems. Das gekapselte Dateisystem stellt dabei sämtliche Software zur Verfügung, die eine Anwendung benötigt, um ausgeführt zu werden. Dazu gehören auch bestimmte Versionen des Betriebssystems oder anderer Dienstprogramme sowie Software-Bibliotheken. Arbeitet eine Anwendung beispielsweise nur mit einer bestimmen Version einer Laufzeitbibliothek oder einem Framework, so ist es ohne Probleme möglich, genau die benötigten Versionen im Container zur Verfügung zu stellen, und zwar völlig unabhängig von den Laufzeitbibliotheken des Host-Systems sowie allen anderen Containern, die auf derselben Maschine laufen. So erreichen Sie eine komplette Kapselung der Umgebung, die eine Anwendung benötigt, um ausgeführt zu werden.

Wenn wir in diesem Buch von *Anwendungen* sprechen, meinen wir *Serveranwendungen*, also Anwendungen, die auf einem Server laufen und auf die Sie über eine definierte Schnittstelle wie beispielsweise das HTTP-Protokoll auf Port 80 o. Ä. zugreifen können. Es sind explizit keine Anwendungen wie Word oder Excel gemeint, die auf einem Client-Rechner ausgeführt werden.

Die Kapselung einer Anwendung mit allen Komponenten, die zum Ausführen der Anwendung benötigt werden, hat den großen Vorteil, dass die Anwendung unabhängig von beispielsweise Updates des Host-Systems immer ausgeführt werden kann. Stellen Sie sich einen Server vor, auf dem unterschiedliche Anwendungen installiert werden sollen. Oft kommt es in solchen Umgebungen zu Problemen, weil die eine Anwendung von einer bestimmten Version eines Frameworks abhängig ist, die andere Anwendung von einer anderen Version desselben Frameworks. Leider können Sie Frameworks im Server-Umfeld oft nicht parallel installieren, sodass die Installation der beiden Anwendungen auf demselben Server scheitert.

Das kann bei der Containertechnologie nicht passieren, da in unserem Beispiel die Anwendungen in zwei unterschiedlichen Containern installiert werden und in jedem Container die richtige Version des Frameworks installiert ist. Obwohl Sie dann zwei Container haben, wird weniger Speicherplatz benötigt als bei zwei getrennten virtuellen Maschinen, da in den Containern kein Betriebssystem vorhanden ist, sondern nur die unterschiedlichen Versionen des Frameworks.

Docker ist kostenlos – oder?

Grundsätzlich ist Docker ein Open-Source-Produkt und wird kostenlos zur Verfügung gestellt. Das heißt, Sie können Docker ohne Lizenzgebühren oder Ähnliches nutzen. Hergestellt wird die Docker-Technologie von Docker Inc., einem Unternehmen aus San Francisco. Doch ähnlich wie für einige Linux-Distributionen, die von kommerziellen Unternehmen betrieben werden, hat auch Docker Inc. ein Geschäftsmodell für Docker entwickelt. So gibt es neben Support- und Hosting-Angeboten auch die sogenannte »Enterprise Edition« von Docker (Docker Enterprise als Gegensatz zur Docker Desktop – der ehemaligen Community Edition).

Die kostenlose Version von Docker ist *Docker Desktop*. Hierbei handelt es sich um ein Open-Source-Produkt. Es gibt jedoch auch eine kostenpflichtige Version mit weiteren Features, *Docker Enterprise* (Docker EE).

Die Docker EE wird in einem Abo-Modell vertrieben und bietet zusätzlich zu den Features von Docker Desktop noch einige Funktionen, die gerade im produktiven Umfeld die Arbeit sehr erleichtern können. Dazu gehören beispielsweise eine auditierte, zertifizierte Infrastruktur und Plug-ins, eine verbesserte Verwaltung von Images mithilfe einer Weboberfläche (der *Docker Trusted Registry*, DTR) sowie Containern und Anwendungen in der sogenannten *Universal Control Pane* (UCP).

Für die Beispiele in diesem Buch verzichten wir auf Features, die Docker EE benötigen, und verwenden ausschließlich Docker Desktop. An einigen Stellen verweisen wir aber auf die Docker EE und darauf, was für Möglichkeiten dort existieren.

Grundlegende Elemente in Docker

Die grundlegenden Elemente in Docker sind:

- ✔ **Image**

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

- ✔ **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.

- ✔ **Container-Host**

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

- ✔ **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.

- ✔ **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.

✔ **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 `hub.docker.com`. Natürlich können Sie auch Ihre eigene private Registry erzeugen.

Um die Zusammenhänge noch einmal besser zu verstehen, schauen Sie sich bitte einmal Abbildung 1.1 an.

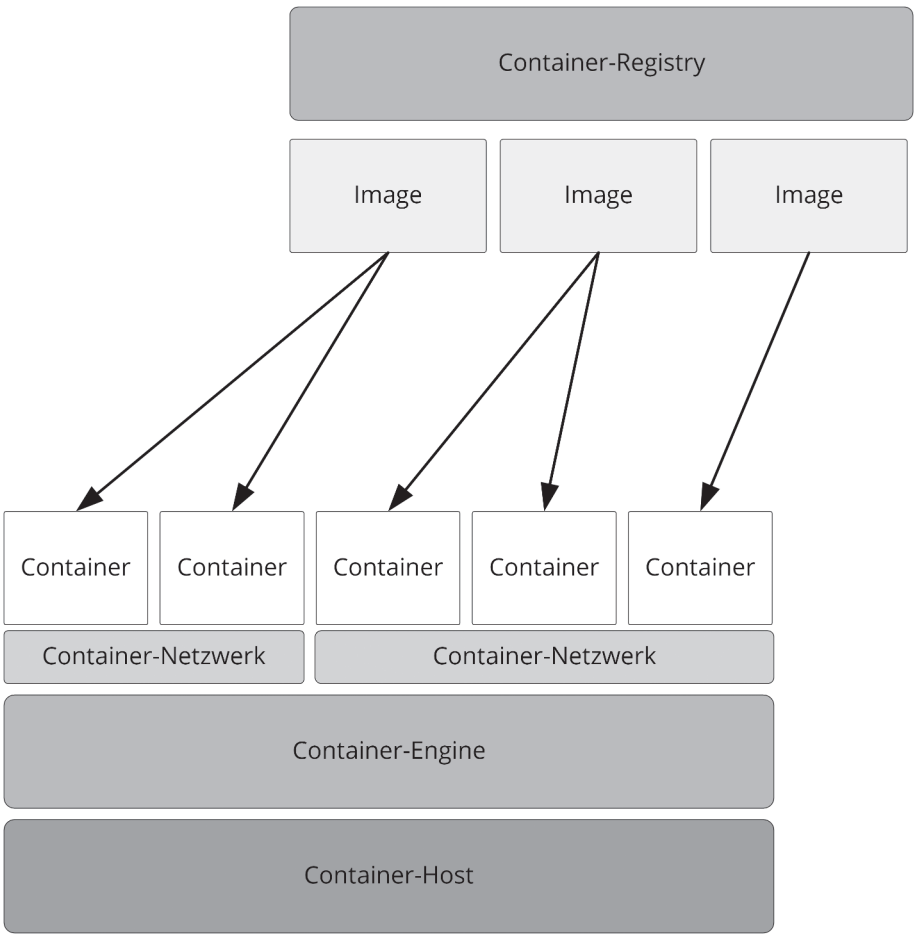

Abbildung 1.1: Überblick über die Docker-Elemente

In Abbildung 1.1 sehen Sie, dass der Container-Host die Basis für die Containertechnologie darstellt. Auf dem Container-Host wird die Container-Engine ausgeführt. Diese stellt wiederum Container-Netzwerke zur Verfügung, in denen dann Container bereitgestellt werden können. Die Container selbst werden aus Images erzeugt, die in einer Container-Registry liegen. Auch selbst erzeugte Images leiten Sie üblicherweise von einem sogenannten *parent image* ab. Die einzige Ausnahme hiervon ist, dass Sie ein Image von Grund auf selbst

erzeugen, dass Sie wiederum als *parent image* für andere eigene Images verwenden. Die Technologien hierfür stellen wir in Teil II des Buchs vor.

Auch wenn die einzelnen Komponenten von Docker im weiteren Verlauf des Buchs noch viel detaillierter vorgestellt werden, so wollen wir uns hier doch noch einmal kurz mit einigen dieser Komponenten beschäftigen.

Container

Ein Container ist keine virtuelle Maschine, sondern einfach nur ein Prozess, der auf dem Container-Host läuft. Stoppen Sie diesen Prozess, so wird der Container sofort beendet.

Container können im Gegensatz zu virtuellen Maschinen nur begrenzt auf Hardware zugreifen, das heißt, für einen Container steht nur die Hardware des Container-Hosts zur Verfügung. Im Gegensatz zu einer virtuellen Maschine können Sie Hardware nicht overcommitten.

Beim *Overcommitten* wird einer virtuellen Maschine mehr Hardware zugewiesen, als die eigentliche Maschine zur Verfügung stellt. So können Sie beispielsweise einer virtuellen Maschine 8 GB RAM zuweisen, obwohl der physische Rechner, auf dem die virtuelle Maschine läuft, nur 4 GB RAM hat. Nutzt die virtuelle Maschine mehr als die 4 GB RAM aus, so werden Daten auf die Festplatte ausgelagert.

Images

Genauso wenig wie ein Docker-Container eine virtuelle Maschine ist, ist ein Image eine virtuelle Festplatte.

Die offizielle Definition für ein Image ist: »Ein *Image* ist eine geordnete Menge an root-Dateisystemänderungen und den korrespondierenden Ausführungsparametern, die innerhalb einer Container-Laufzeitumgebung verwendet werden können.« Das ist natürlich eine sehr komplizierte und abstrakte Beschreibung. Grundsätzlich können Sie sich ein Image wie eine Blaupause oder einen Bauplan für einen Container vorstellen. Im Image stehen alle Änderungen am Grund-Dateisystem in geordneter Reihenfolge.

Die Definition besagt, dass es sich bei einem Image um Änderungen an einem Basis-Dateisystem handelt, die eine Ordnung, also eine Reihenfolge haben. Das heißt, ausgehend von einem Basis-Dateisystem zeichnet jede Schicht des Images eine Änderung an diesem Dateisystem oder der vorhergehenden Schicht auf. Damit alle Änderungen in der richtigen Reihenfolge durchgeführt werden – es kann ja sein, dass in einer Änderung eine Datei erzeugt wird, die in der nächsten Änderung verändert wird –, sind die Änderungen in einer geordneten Menge aneinandergereiht.

Aus der Definition wird weiterhin klar, dass ein Image ausführbare Dateien und deren Abhängigkeiten enthält. Was ein Image dagegen nicht enthält, ist ein komplettes Betriebssystem, ein Kernel oder Kernel-Module wie beispielsweise Treiber.

Je nachdem welche Änderungen am Basis-Dateisystem durchgeführt werden, kann ein Image sehr klein oder sehr groß sein. In einem Image kann nur eine einfache Textdatei als Änderung gegenüber dem Basis-Dateisystem enthalten sein, in einem anderen Image eine Gigabyte große Datenbankdatei eines Datawarehouses.

Container-Netzwerke

Damit Container untereinander kommunizieren können, werden sie über *Container-Netzwerke* miteinander verbunden. Standardmäßig richtet Docker drei Netzwerke auf dem Docker-Host ein. Diese werden als *bridge*, *host* und *none* bezeichnet. In diesem Abschnitt erhalten Sie eine kurze Übersicht über die Docker-Netzwerke, die in Kapitel 10 näher erklärt wird.

- **bridge**

  Das bridge-Netzwerk (wird auch manchmal als docker0 bezeichnet) ist das Standard-Docker-Netzwerk, in das jeder Docker-Container eingehängt wird, sofern nichts anderes definiert wurde.

Unter dem *Host-Adapter* des Docker-Hosts (also des Rechners, auf dem Docker ausgeführt wird) versteht man die Netzwerkkarte, die in diesem Rechner betrieben wird, also die Netzwerkkarte, über die der Docker-Host auf das Netzwerk zugreift.

- **host**

  Das *host-Netzwerk* stellt im Prinzip den Host selbst dar. Wenn Sie einen Container an das host-Netzwerk binden, binden Sie ihn direkt (ohne NAT) an die IP-Adresse des Container-Hosts.

- **none**

  Das Netzwerk *none* ist gar kein wirkliches Netzwerk, sondern eher die Einstellung, dass ein Container gar nicht mit einem Netzwerk verbunden ist. Im Prinzip ist das so, als wenn ein Rechner kein Netzwerkkabel in der Netzwerkkarte stecken hat.

Docker-Container, die sich in einem Netzwerk befinden, können direkt miteinander kommunizieren. Die Kommunikation zwischen Containern über das Container-Netzwerk verlässt niemals den Host. Standardmäßig sind alle externen Ports geschlossen. Möchten Sie von extern auf einen Container im Container-Netzwerk zugreifen, müssen Sie einen Port, auf dem eine Anwendung auf dem Container läuft, an einen Port des Container-Hosts binden.

Vergleich Container und virtuelle Maschinen

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.

In Abbildung 1.2 können Sie einen Vergleich zwischen virtuellen Maschinen und Containern sehen. Sehen Sie sich die Abbildung von unten nach oben an. 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.

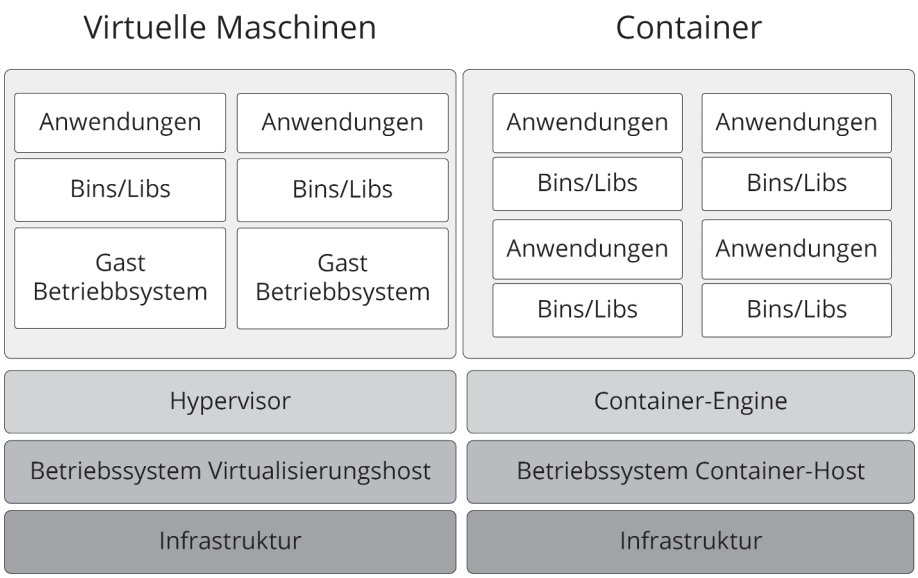

Abbildung 1.2: Vergleich zwischen virtuellen Maschinen und Containern

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.

Beim Vergleich von virtuellen Maschinen und Containern lassen sich grundsätzlich einige Unterschiede feststellen.

- ✔ **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. Sie finden in Kapitel 24 ein Beispiel, bei dem Sie genau so eine Datenbankanwendung aktualisieren. Beachten Sie eine virtuelle Maschine, so wird die alte Software-Version auf die neue Software-Version aktualisiert, und die virtuelle Maschine ändert ihren Status.

- ✔ **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.

- ✔ **Container können Sie im Vergleich zu virtuellen Maschinen recht schnell erzeugen und starten.**

  Im Gegensatz zu virtuellen Maschinen muss kein Betriebssystem gebootet werden, wenn der Container startet.

- ✔ **Virtuelle Maschinen können Sie gut in eine Active Directory Infrastruktur einbinden.**

  Im *Active Directory*, das ist das zentrale Netzwerk-Verzeichnis von Windows-Netzwerken, in dem alle User, Computer und weitere Netzwerk-Ressourcen verwaltet werden, besitzt jeder Computer, der zu dem Windows-Netzwerk gehört, eine eindeutige ID. Wegen der Langlebigkeit von virtuellen Maschinen macht es Sinn, diese in das Active Directory aufzunehmen. Bei schnelllebigen Containern ist das meist nicht so sinnvoll, da für jeden Container ein neues Computer-Objekt im Active Directory angelegt, beim Löschen des Containers aber nicht gelöscht wird, sodass sich dort nach einer Zeit unter Umständen viele Karteileichen sammeln.

- ✔ **Unter Docker stellen Sie pro Container eine Anwendung zur Verfügung.**

  Auf einer virtuellen Maschinen installieren Sie viele Anwendungen parallel, da es sich um eine teure Ressource handelt, die möglichst gut ausgenutzt werden muss.

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.

Früher war es tatsächlich der Fall, dass das Betriebssystem nicht »wusste«, dass es auf einer virtuellen und nicht auf einer physischen Maschine ausgeführt wurde. Inzwischen »wissen« die Betriebssysteme, ob sie virtuell oder physisch ausgeführt werden. Üblicherweise wird auf den virtuellen PCs eine Software installiert, mit der sie besser mit dem Virtualisierungshost kommunizieren können.

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.

In der folgenden Liste finden Sie typische Virtualisierungslösungen für Clientbetriebssysteme:

- ✔ **Virtual Box**

  Virtual Box wird von Oracle betreut und kann unter Windows, Linux, Mac OS und Solaris installiert werden. Nähere Informationen zu Virtual Box finden Sie unter `https://www.virtualbox.org/`

- ✔ **Hyper-V**

  Hyper-V ist die hauseigene Virtualisierungslösung von Microsoft. Neben der Servervariante steht Hyper-V auch unter Windows 10 als Professional und Enterprise Edition zur Verfügung. Weitere Informationen zu Hyper-V unter Windows 10 finden Sie unter `https://docs.microsoft.com/de-de/virtualization/hyper-v-on-windows/about/`

- ✔ **VMWare Workstation Player**

  VMWare Workstation Player ist die kostenlose und abgespeckte Clientlösung der bekannten Virtualisierungssoftware *VMWare*. Der VMWare Player kann beispielsweise keine Snapshots erstellen. Nähere Informationen zum VMWare Player bekommen Sie unter `https://www.vmware.com/de/products/workstation-player.html`

- ✔ **VMWare Workstation**

  VMWare Workstation, aktuell in der Version 15 Pro, ist die kostenpflichtige Variante von VMWare, die wesentlich mehr Funktionen als der VMWare Player bietet. Nähere Informationen zu VMWare Workstation finden Sie unter `https://www.vmware.com/de/products/workstation-pro.html`.

✔ **Parallels Desktop**

Parallels Desktop ist eine Client-Virtualisierungslösung speziell für den Mac. Interessant im Vergleich zu den anderen vorgestellten Virtualisierungslösungen ist, dass Parallels Desktop Fenster des virtualisierten Betriebssystems (zum Beispiel Windows) in den Desktop des Macs einblenden kann, sodass die Anwendungen in diesen Fenstern wie native Mac-Anwendungen aussehen. Nähere Informationen zu Parallels Desktop finden Sie unter `https://www.parallels.com/de/`

Sie werden sich an dieser Stelle sicherlich fragen, warum wir auf die Virtualisierungsumgebungen in einem Buch über Docker so intensiv eingehen. Das hat zwei Gründe. Zum einen möchten wir klarstellen, dass es sich bei Docker eben nicht um eine Virtualisierungslösung handelt, zum anderen werden uns einige der oben genannten Virtualisierungslösungen später in diesem Buch im Abschnitt über Docker auf dem Mac oder Docker auf Windows noch einmal über den Weg laufen.

Was ist ein Container?

Kommen wir nun zur rechten Seite von Abbildung 1.2, den Containern. Wie Sie in der Abbildung 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.

Wenn Sie sich schon mit Docker beschäftigt haben, werden Sie jetzt dieses Buch zur Seite legen und sich denken: »Was schreiben die denn da für einen Quatsch? 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 genau das funktioniert, können Sie in Kapitel 13 lernen.

Weiter bedeutet das auch, dass ein Container auf der wirklichen Hardware des Container-Hosts läuft, es ist also keine zwischengeschaltete Schicht aus virtueller Hardware vorhanden. Die Hardware des Container-Hosts kann verwendet werden, aber nicht mehr. Anders als bei einer virtuellen Maschine können Sie nicht mehr Hardware zur Verfügung stellen, als eigentlich vorhanden ist.

Einsatzgebiete von Docker

Da Docker in den letzten Jahren eine unglaubliche Popularität gewonnen hat – manche sagen, dass das aufgrund des niedlichen Wal-Maskottchens Moby Dock passiert ist, aber das glauben wir nicht – geht es in diesem Abschnitt darum, einmal herauszufinden, was denn die Einsatzgebiete von Docker sind und warum Docker in diesen Einsatzgebieten einer herkömmlichen virtuellen Maschine überlegen ist.

Wenn man sich mit Docker beschäftigt, trifft man unweigerlich auf Moby Dock (`https://blog.docker.com/2013/10/call-me-moby-dock/`), den kleinen blauen Wal. Moby Dock taucht im Logo von Docker auf, ist aber eigentlich nicht das Firmenmaskottchen. Das ist Gordon, die Schildkröte, die sogar einen eigenen Twitter-Account hat (`https://twitter.com/gordontheturtle` | @GordonTheTurtle). Gordon lebt auf dem Gelände von Docker Inc. und entwickelt sich genauso rasant wie Docker selbst: Als Docker 2013 veröffentlicht wurde, war Gordon eine kleine Baby-Schildkröte, die ohne Probleme auf eine Tastatur passte. Heute ist Gordon so groß, dass er nur noch von zwei Personen hochgehoben werden kann. Moby Dock hingegen ist das Logo von Docker und ist als Sieger aus einem Wettbewerb hervorgegangen.

✔ **Softwareverteilung herkömmlicher Anwendungen**

Die Installation von Serversoftware kann mitunter ein ziemlich aufwendiger und schmerzhafter Prozess sein. Ganz früher, als Rechner noch aus Holz geschnitzt wurden, wurden Serverapplikationen vom Softwarehersteller als Installationspakete zur Verfügung gestellt (okay, das ist heute auch noch oft so), und der Administrator, der die Software installieren sollte, musste sich alle Bibliotheken, Frameworks usw., auf denen die Software basierte, mühsam zusammensuchen und vor der eigentlichen Softwareinstallation installieren. Wenn Sie Glück hatten, gab es zumindest eine Anleitung, welche Voraussetzungen die spezielle Serversoftware benötigte, um ausgeführt zu werden. Dieser Prozess ist über die Jahre vereinfacht worden, sodass die Installationsprogramme von Serveranwendungen inzwischen dazu in der Lage sind – sofern der Server Zugang zum Internet hat –, die benötigten Softwarekomponenten selbstständig herunterzuladen.

Dies ist aber auch nicht unproblematisch. Werden beispielsweise die benötigten Bibliotheken in einer bestimmten Version heruntergeladen und überschreiben diese bereits vorhandene Bibliotheken in einer anderen Version, kann das dazu führen, dass eine andere Serveranwendung, die auf demselben Server läuft, nicht mehr ausgeführt werden kann.

Eine mögliche Lösung für dieses Problem stellen virtuelle Maschinen dar, über die Sie die Anwendungen auf derselben Serverhardware voneinander trennen können. In Bezug auf die Softwareverteilung und Bereitstellung haben virtuelle Maschinen aber mehrere entscheidende Nachteile. Zum einen sind die virtuellen Festplatten wegen

des installierten Gast-Betriebssystems recht groß, zum andern ist es lizenzrechtlich nicht immer so einfach, eine virtuelle Festplatte mit einem vorinstallierten Gast-Betriebssystem zu verteilen.

Genau hier kommt Docker ins Spiel. Serversoftware kann leicht über einen Docker-Container vorinstalliert vertrieben werden. Da kein Gast-Betriebssystem installiert ist, sind Docker-Images in der Regel relativ klein. Durch die Isolierung der Container voneinander kann in jedem Container die Laufzeitumgebung inklusive Bibliotheken, Frameworks usw. installiert sein, die die Serveranwendung in diesem Container zum Ausführen braucht. Um aus einem Docker-Image einen lauffähigen Container zu machen, benötigen Sie nur einen einzelnen Befehl, und schon steht die Serveranwendung zur Verfügung.

Wie Sie das alles genau machen und eigene Images mit vorinstallierten Serveranwendungen installieren, werden Sie im weiteren Verlauf dieses Buchs lernen.

✔ **Hybrid Cloud**

Aus vielen Gründen sind Cloud-Lösungen momentan sehr populär. Obwohl neben Infrastructure as a Service (IaaS) auch Platform as a Service (PaaS) oder Software as a Service (SaaS) angeboten werden, beruhen die meisten Cloud-Workloads immer noch auf IaaS. IaaS führt aber zu einem großen Problem, dem sogenannten Vendor-Lock-in.

Mit *Vendor-Lock-in* ist die Abhängigkeit vom Anbieter gemeint. Wenn Sie erst einmal Ihre Infrastruktur bei einem Cloud-Anbieter ausgerollt haben, ist es recht schwierig, diese Infrastruktur zu einem anderen Cloud-Anbieter zu verschieben. Der Grund dafür ist, dass für einen Transfer die Virtualisierungs-Technologie und damit die Rechenzentrums-Architektur dieselbe sein muss und das bei den großen Cloud-Anbietern nicht gegeben ist. Sie sind sozusagen bei Ihrem Cloud-Anbieter gefangen.

Auch dies ist ein Problem, für das Docker eine mögliche Lösung darstellen kann. Docker-Container basieren wie oben bereits beschrieben auf Images. Images sind Vorlagen, aus denen dann die Container instanziiert werden. Einen Docker-Container kann ich auf jedem Docker-Host laufen lassen, egal welches Betriebssystem auf dem Docker-Host ausgeführt wird. Das gilt natürlich auch für Cloud-Provider. Gefällt es mir bei meinem Cloud-Anbieter nicht mehr, dann lösche ich meine Container dort einfach und instanziiere sie bei einem anderen Anbieter.

Im Bezug auf Cloud-Dienste gibt es drei verschiedenen Cloud-Servicearten: *Infrastructure as a Service (IaaS)*, *Platform as a Service (PaaS)* und *Software as a Service (SaaS)*.

Was es damit auf sich hat, lässt sich schnell erklären. *IaaS* bedeutet, dass Sie Ihre virtuellen Maschinen nicht lokal bei sich im Rechenzentrum laufen lassen, sondern dass Sie die Maschinen bei einem Cloud-Anbieter hosten. Es gibt hier keinen wirklichen Unterschied zu lokalen virtuellen Maschinen; Sie können mit einer solchen Maschine all das tun, was Sie auch lokal tun können. Sie sind dann

aber auch für die Wartung und den Betrieb der Maschine zuständig. Wie schon der Onkel von Spiderman sagte: »Mit großer Macht kommt große Verantwortung.«

Bei *PaaS* wird Ihnen vom Cloud-Anbieter eine bestimmte Serveranwendung zur Verfügung gestellt, beispielsweise ein Datenbankserver. Sie nutzen diesen Datenbankserver als Datenbank, ohne dass Sie genau wissen, wie der Server installiert und konfiguriert ist. Natürlich steht Ihnen mit einer PaaS-Datenbank noch keine lauffähige Software zur Verfügung, sondern nur die Basis, auf der Sie eine Software bauen können, daher »Platform as a Service«.

Bei PaaS wird lediglich die Funktion der Datenbank zur Verfügung gestellt. Das hat den großen Vorteil, dass Sie sich um die Verwaltung der dahinterliegenden Infrastruktur nicht kümmern müssen und dass Sie die Systeme auch nicht warten müssen. Der Nachteil dieses Service ist, dass bestimmte Funktionalitäten des Datenbanksystems, die sehr betriebssystemnah ausgeführt werden (wie beispielsweise einen Shell-Befehl auf Betriebssystemebene ausführen), nicht zur Verfügung stehen. Die Cloud-Anbieter möchten ihr System, auf dem sich außer Ihnen noch andere Kunden tummeln, schützen.

SaaS können Sie sich so vorstellen, als wenn Sie ein E-Mail-Konto bei einem Anbieter wie Google oder Web.de einrichten. Sie melden sich bei diesem Dienst an, geben ein paar Basisinformationen ein und können dann sofort loslegen und E-Mails versenden und empfangen. Ihnen wird die komplette Funktionalität der Software zur Verfügung gestellt, ohne dass Sie irgendetwas über die technische Realisierung im Hintergrund wissen müssen.

✔ **Continuous Integration (CI) und DevOps**

Bei Continuous Integration und DevOps geht es um die Softwareentwicklung von Serversystemen. Unter *Continuous Integration* versteht man einen Prozess, bei dem viele Entwicklungsschritte wie das Kompilieren, Testen und Veröffentlichen automatisiert sind. Die von den Entwicklern geschriebenen Source-Komponenten werden in eine zentrale Quellcodeverwaltung eingecheckt. Der Eincheck-Vorgang stößt dann automatisch einen sogenannten Build-Prozess an, bei dem das System aus dem Quellcode eine lauffähige Anwendung erstellt. Ist der Build-Prozess erfolgreich abgeschlossen, wird die lauffähige Anwendung automatisch getestet.

In Continuous-Integration-Prozessen können Container auch sehr gut eingesetzt werden. Für den Anwendungstest wird ein neuer, frischer Container erzeugt, in dem die Anwendung bereitgestellt wird. Der Vorteil liegt darin, dass der Container immer einen definierten Ausgangszustand hat und der Software-Test nicht durch Nebeneffekte, die beispielsweise durch nicht vollständig entfernte Software verursacht werden, beeinflusst wird.

Außerdem kann ein CI-Prozess als Ergebnis automatisch einen Docker-Container erstellen, der für die Verteilung der Software an Kunden genutzt wird.

Das Wort *DevOps* setzt sich aus den Worten *Development* und *Operations* zusammen. Unter DevOps versteht man die enge Zusammenarbeit zwischen Entwicklern

(Development) und den IT-Administratoren, die die von den Entwicklern geschriebenen Anwendungen dann in der Produktivumgebung betreiben müssen (Operations). Früher gab es oft einen tiefen Graben zwischen diesen beiden Fronten, da die Zielsetzungen der beiden Gruppen grundsätzlich andere sind. Entwickler müssen agil arbeiten, neue Funktionen implementieren und schnell Lösungen schaffen, wohingegen IT-Administratoren die bestehende Umgebung bewahren wollen und dafür sorgen müssen, dass diese ohne Probleme läuft. Durch die immer weiter ansteigende Geschwindigkeit bei der Softwareentwicklung verschwimmen die Grenzen zwischen Entwicklern und IT-Administratoren, sodass Entwickler teilweise Operations-Aufgaben übernehmen und IT-Administratoren teilweise Development-Aufgaben. Um dieser Unschärfe Rechnung zu tragen, wurde der Begriff DevOps geschaffen. Docker unterstützt DevOps durch eine agile Plattform, auf der schnell Tests durchgeführt werden können und die auch bei Aufgaben wie dem Upgrade eines Systems schneller als jede andere bekannte Methode ist. Über diese Themen erfahren Sie in Kapitel 25 einiges mehr.

- ✔ **Microservices**

  Unter *Microservices* versteht man ein Software-Architekturmodell, bei dem eine komplexe Anwendung aus kleinen, voneinander unabhängigen Diensten zusammengestellt wird. Die Kommunikation zwischen diesen Diensten geschieht über Standardprotokolle, wie sie auch anderweitig im Internet verwendet werden. Jeder Microservice erledigt eine spezifische Aufgabe. Durch diese Architektur ist ein modularer Aufbau und damit die Skalierung von Anwendungen möglich.

  Durch die Isolation der einzelnen Docker-Container voneinander eignet sich Docker hervorragend für die Erstellung einer Anwendung auf Basis von Microservices. In jedem Container läuft ein Microservice, der nur über Standardprotokolle mit anderen Microservices in anderen Containern kommunizieren kann. Über Microservices erfahren Sie in Kapitel 15 mehr.

Verschiedene Ausführungsarten von Docker-Containern

Wie bereits oben beschrieben, ist ein Docker-Container nichts anderes als ein Prozess. Dieser Prozess wird beendet, wenn der Code, der im Docker-Container ausgeführt wird, ausgeführt wurde. Aufgrund dieser Eigenheit von Docker-Containern ergeben sich drei mögliche Modi wie Sie Docker-Container verwenden können

- ✔ **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. Ein gutes Beispiel für einen Task-Container ist ein Container, in dem sich ein PowerShell-Skript befindet, das eine Azure-Umgebung aufbaut. Dieses PowerShell-Skript hat viele Paketabhängigkeiten zu den entsprechenden PowerShell-Paketen, die auch alle im

Container installiert sind. Führen Sie den Container nun aus, wird dieser gestartet, und das PowerShell-Skript wird ausgeführt. Hierdurch wird die Azure-Umgebung erzeugt. Nachdem das Skript durchgelaufen ist, nachdem also der Task ausgeführt wurde, wird der Container wieder beendet. Der Vorteil, ein PowerShell-Skript so zu kapseln, ist, dass sämtliche Pakete, auf die das PowerShell-Skript zugreift, auch im Container installiert sind und daher nicht auf dem Rechner installiert werden müssen. Task-Container sind hervorragend für Automatisierung geeignet und können auch zusammen mit dem Task Scheduler genutzt werden.

- ✔ **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. Was genau ein Dockerfile ist, erfahren Sie in Kapitel 9. Interaktive Container werden so lange ausgeführt, bis Sie die interaktive Sitzung verlassen und das interaktiv gestartete Programm beenden.

- ✔ **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.