Kapitel 1

Einführung

1.1 Die Entstehung von Docker

Es war der 15. März 2013, als Solomon Hykes, der Gründer und Geschäftsführer von dotCloud (jetzt Docker, Inc.), das Projekt Docker erstmals – ohne Vorankündigung und großes Brimborium – in einer blitzartigen Fünf-Minuten-Präsentation (`http://youtu.be/wW9CAH9nSLs`) auf der Python Developers Conference in Santa Clara, Kalifornien, vorstellte. Zu diesem Zeitpunkt hatten lediglich rund 40 Leute außerhalb des Unternehmens dotCloud Gelegenheit gehabt, Docker auszuprobieren.

Schon wenige Wochen nach dieser Präsentation brach ein unerwartet großer Medienrummel über das Projekt herein. Es wurde umgehend auf GitHub (`https://github.com/moby/moby`) als Open Source zur Verfügung gestellt, damit jedermann es herunterladen und eigene Beiträge dazu leisten konnte. Im Laufe der darauffolgenden Monate gewann Docker in der Branche zunehmend an Bekanntheit, und man bescheinigte ihm, die bisherige Art und Weise der Softwareentwicklung, -verteilung und -nutzung revolutionieren zu können. Innerhalb eines Jahres hatte praktisch jeder Branchenkundige zumindest von Docker Notiz genommen, wenngleich vielen Menschen nicht ganz klar war, was es eigentlich genau leistet und warum das so spannend ist.

Docker ist ein Tool, das eine einfache Kapselung des Erstellungsprozesses von Artefakten zur Verteilung beliebiger Anwendungen verspricht. Dabei ist das Deployment für beliebige Umgebungen skalierbar, und der Workflow sowie die Reaktionsfähigkeit der betreffenden agilen Unternehmen werden optimiert.

1.2 Das Docker-Versprechen

Diejenigen, die noch keine Erfahrung mit Docker haben, verstehen Docker vordergründig als Virtualisierungsplattform. Es leistet tatsächlich aber viel mehr. Es umspannt eine Reihe von belebten Branchensegmenten, für die bereits Lösungsansätze in Form individueller Technologien wie KVM, Xen, OpenStack, Mesos, Capistrano, Fabric, Ansible, Chef, Puppet, SaltStack usw. existieren. Wie Sie vielleicht bemerkt haben, ist die Liste der Produkte, mit denen Docker in Konkurrenz tritt, schon ziemlich aufschlussreich. Viele Administratoren würden kaum behaupten, dass Virtualisierungslösungen im Wettbewerb mit Tools für die Konfi-

gurationsverwaltung stünden – dennoch wirkt sich Docker auf beide Technologien disruptiv aus. Dies hängt im Wesentlichen damit zusammen, dass Docker einen großen Einfluss auf die Arbeitsweise hat. Das wiederum wirkt sich stark auf die zuvor genannten traditionell voneinander isolierten Segmente in der DevOps-Pipeline aus.

Zudem werden den vorgenannten Technologien im Allgemeinen produktivitätssteigernde Eigenschaften nachgesagt, und genau deshalb wird ihnen auch so viel Aufmerksamkeit zuteil. Docker befindet sich sozusagen im Zentrum einiger der leistungsfähigsten Technologien des letzten Jahrzehnts und kann zu signifikanten Verbesserungen fast jedes Schritts der Pipeline führen.

Würden Sie Docker allerdings Punkt für Punkt mit den jeweiligen Platzhirschen der verschiedenen Einsatzbereiche vergleichen, stünde es höchstwahrscheinlich bloß wie ein mittelmäßiger Wettbewerber da. In manchen Bereichen kann Docker seine Stärken besser ausspielen als in anderen, insgesamt betrachtet decken die Features, die es mitbringt, aber ein sehr breites Anwendungsspektrum ab. Durch die Zusammenführung der Schlichtheit von Softwareverteilungstools wie Capistrano und Fabric mit der komfortablen Verwaltungshandhabung von Virtualisierungssystemen sowie optionalen Möglichkeiten zur problemlosen Implementierung der Automatisierung von Workflow und Orchestrierung stellt Docker einen sehr leistungsfähigen Satz an Features zur Verfügung.

Viele neue Technologien kommen und gehen, insofern ist eine gewisse Skepsis gegenüber den neuesten Trends durchaus angebracht. Oberflächlich betrachtet, könnte man auch Docker sicherlich einfach als eine weitere neue Technologie abtun, die sich irgendwelcher speziellen Probleme annimmt, mit denen Entwickler und Administratoren konfrontiert sind. Und wenn Sie es lediglich als Pseudo-Virtualisierungs- oder Deployment-Technologie sehen, erscheint es womöglich wirklich nicht besonders reizvoll. Allerdings leistet Docker sehr viel mehr als nur das, was es auf den ersten Blick zu erkennen gibt.

Die Kommunikation und Workflows mehrerer Teams zu koordinieren, ist oftmals selbst in kleineren Unternehmen und Organisationen schwierig und teuer. Dessen ungeachtet kommt dem detaillierten Informationsaustausch zwischen den Teams jedoch immer mehr Bedeutung für ein erfolgreiches Arbeiten zu. Die Entdeckung und Implementierung eines Tools, das diese Kommunikation erleichtert und gleichzeitig dazu beiträgt, stabilere Software zu produzieren, wäre also von enormem Nutzen – und genau deswegen ist Docker einen zweiten Blick wert. Natürlich ist es kein Wundermittel, und seine zielführende Implementierung will wohlüberlegt sein, trotzdem ist Docker ein guter Ansatz, um in der Praxis auftretende organisatorische Probleme zu lösen und einem Unternehmen das zügigere Deployment besserer Software zu ermöglichen. Abgesehen davon kann ein gut geplanter Docker-Workflow auch zufriedenere Teams und spürbare Kosteneinsparungen zur Folge haben.

Wo also treten die größten Schwierigkeiten auf? Software in dem Tempo auszuliefern, das heutzutage gefordert wird, ist nicht leicht – und wenn ein Unternehmen wächst und aus zwei oder drei Entwicklern mehrere Entwicklerteams werden, wird es auch zunehmend schwieriger, die Kommunikation hinsichtlich des Deployments neuer Softwareversionen zu koordinieren. Die Entwickler müssen möglichst weitreichende Kenntnisse von der Umgebung besitzen, in der ihre Software laufen soll, und die Administratoren müssen ihrerseits immer umfassender mit der internen Funktionsweise der ausgelieferten Software vertraut sein. Im Allgemeinen ist es durchaus vernünftig, diese Kenntnisse kontinuierlich weiter zu vertiefen, weil das letztlich zu einem besseren Verständnis der Softwareumgebung insgesamt führt und somit das Design stabilerer Software ermöglicht. Allerdings ist dieses Wissen nur sehr schwer skalierbar, wenn das Wachstum eines Unternehmens zunimmt.

Die spezifischen Details einer Softwareumgebung bedingen oft eine Menge Kommunikation, die für die beteiligten Teams nicht immer unmittelbar von Nutzen ist. Wenn beispielsweise ein Entwicklerteam darauf warten muss, dass ein Administratorenteam irgendeine Library in der Version 1.2.1 deployt, kommt es zunächst nicht weiter – und für das betroffene Unternehmen bedeutet das verlorene Arbeitszeit. Könnten die Entwickler die Version der verwendeten Library hingegen einfach selbst aktualisieren und dann ihren Code schreiben, testen und ausliefern, würde sich die Zeit bis zum Deployment deutlich verkürzen, und es bestünden darüber hinaus geringere Risiken beim Deployment der Änderung. Und wenn umgekehrt die Administratorenteams die Software auf dem Host aktualisieren könnten, ohne sich mit mehreren Entwicklerteams abstimmen zu müssen, ginge es ebenfalls schneller voran. Docker unterstützt die Möglichkeit, die Software so zu isolieren, dass weniger Kommunikation zwischen den beteiligten Teams erforderlich ist.

Docker reduziert aber nicht nur die notwendige Kommunikation, sondern wirkt sich auch hinsichtlich der Softwarearchitektur recht bestimmend aus und begünstigt Anwendungen, die besonders stabil ausgelegt sind. Seine architektonische Philosophie stellt *atomare Container* oder *Wegwerf-Container* in den Mittelpunkt. Beim Deployment wird die gesamte laufende Umgebung der alten Anwendung zusammen mit der Anwendung selbst weggeworfen. Nichts in der Umgebung der alten Anwendung überlebt länger als die Anwendung selbst – eine einfache Idee mit großem Effekt. Auf diese Weise wird es äußerst unwahrscheinlich, dass eine Anwendung versehentlich auf irgendwelche Überbleibsel einer vorhergehenden Version zugreift. Und auch beim Debuggen vorgenommene kurzlebige Änderungen können so keinen Eingang in künftige Versionen finden, indem sie vom lokalen Dateisystem eingelesen werden. Außerdem sind Anwendungen dadurch hochgradig portabel und lassen sich leicht auf einen anderen Server verschieben, denn alle Zustände müssen unveränderlicher Bestandteil des Deployment-Arte-

fakts sein oder an einen externen Speicherort wie eine Datenbank, einen Cache oder einen Dateiserver übergeben werden.

Die Anwendungen sind somit nicht nur besser skalierbar, sondern auch zuverlässiger – und die Instanzen eines Anwendungscontainers können kommen und gehen, ohne dass sich dies großartig auf die Verfügbarkeit des Frontends auswirken würde. Hierbei handelt es sich um erprobte architektonische Entscheidungen, die sich bei Anwendungen bewährt haben, in denen Docker nicht zum Einsatz kommt. Docker hat diese Designentscheidungen übernommen und gewährleistet somit, dass sich Anwendungen, in denen das Projekt genutzt wird, im Bedarfsfall ebenfalls dementsprechend verhalten – was nur vernünftig ist.

1.2.1 Vorteile des Docker-Workflows

Es ist nicht einfach, all die Möglichkeiten, die Docker mitbringt, in sinnvoll zusammenhängende Kategorien einzuordnen. Eine gut vollzogene Implementierung verschafft dem Unternehmen als Ganzes sowie den Teams, den Entwicklern und den Administratoren im Besonderen zahlreiche Vorteile. Sie vereinfacht architektonische Designentscheidungen, weil alle Anwendungen aus der Perspektive des Hostsystems von außen betrachtet im Wesentlichen gleich aussehen. Außerdem fällt es leichter, Toolings zu erstellen, die von mehreren Anwendungen gemeinsam genutzt werden können. Alles auf dieser Welt hat Vor- und Nachteile, im Fall von Docker ist es aber schon erstaunlich, wie sehr die Vorteile überwiegen. Im Folgenden sind einige der Vorteile beschrieben, die Docker Ihnen bietet:

- **Vorhandene Fertigkeiten der Entwickler fließen vollumfänglich in die Erstellung von Softwarepaketen mit ein.**
  In vielen Unternehmen mussten für die Erstellung von Softwarepaketen für die jeweils zu unterstützenden Plattformen eigens Stellen für Release und Build Engineers geschaffen werden, die über die erforderlichen Kenntnisse im Umgang mit den entsprechenden Toolings verfügen. Der Einsatz von Tools wie rpm, mock, dpkg oder pbuilder kann ziemlich kompliziert sein und muss jeweils individuell erlernt werden. Docker schnürt alle von Ihnen benötigten Bestandteile zu einem Paket zusammen, das aus nur einer einzigen Datei besteht.
- **Anwendungssoftware und erforderliche Betriebssystemdateien werden in einem standardisierten Image-Format gebündelt.**
  Früher musste ein Paket nicht nur die eigentliche Anwendung enthalten, sondern darüber hinaus auch viele weitere Dateien, auf die sie angewiesen war, wie z.B. Libraries oder Daemons. Trotzdem konnte man sich nie sicher sein, dass Ausführungs- und Entwicklungsumgebung zu 100 Prozent übereinstimmten. Für nativ kompilierten Code bedeutete dies, dass das Build-System exakt die gleichen Versionen der Shared Libraries haben musste wie die Produktivumgebung. Dies erschwerte die Paketierung und bereitete vielen Unter-

nehmen Probleme, verlässlich funktionierende Pakete zu liefern. Oftmals versuchen Entwickler, die die (auf Red Hat basierende) Linux-Distribution Scientific Linux verwenden, in der Hoffnung, dass beide Distributionen einander ähnlich genug sind, ein unter Red Hat getestetes Paket zum Laufen zu bekommen. Mit Docker wird die Anwendung allerdings bereits inklusive aller für die Ausführung erforderlichen Dateien zur Verfügung gestellt. Dank der mehrschichtigen Images (Overlays), die an dieser Stelle zum Einsatz kommen, handelt es sich hierbei um einen effizienten Vorgang, der gewährleistet, dass Ihre Anwendung in der erwarteten Umgebung läuft.

- **Pakete werden mit exakt gleichen Artefakten zum Testen und Ausliefern für alle Systeme in allen Umgebungen benutzt.**
  Wenn Entwickler über eine Versionsverwaltung Änderungen einchecken, kann ein neues Docker-Image erstellt werden, das den vollständigen Überprüfungsvorgang durchläuft und an die Produktivumgebung ausgeliefert wird, ohne dass dabei eine Neukompilierung oder die Erstellung eines neuen Pakets erforderlich wäre, es sei denn, dies ist speziell gewünscht.
- **Abstraktion von Softwareanwendungen von der Hardware, ohne Ressourcen zu opfern.**
  Wenn ein Abstraktionslayer zwischen der physischen Hardware und der darauf laufenden Softwareanwendung erforderlich ist, werden typischerweise herkömmliche Virtualisierungslösungen wie VMware eingesetzt – zum Preis eines höheren Ressourcenbedarfs: Der Hypervisor, der die VMs verwaltet, sowie alle laufenden VM-Kernel nutzen einen bestimmten Prozentsatz der Hardwareressourcen des Systems, die den Anwendungen dann nicht mehr zur Verfügung stehen. Ein Container hingegen ist einfach nur ein weiterer, direkt mit dem Linux-Kernel kommunizierender Prozess, der daher mehr Ressourcen in Anspruch nehmen kann, bis er an die systemseitig vorgegebene oder ihm zugewiesene Kontingentsgrenze stößt.

Zum Zeitpunkt des ursprünglichen Docker-Releases existierten Linux-Container bereits seit einigen Jahren, und auch viele der anderen Technologien, auf denen Docker beruht, sind nicht völlig neu. Allerdings bildet die einzigartige Mischung der nachhaltigen Designentscheidungen hinsichtlich Architektur und Workflow hier ein großes Ganzes, das erheblich leistungsfähiger ist als die Summe seiner Teile. Docker gewährt dem durchschnittlichen Software-Engineer Zugang zu den seit 2008 verfügbaren Linux-Containern. Es gestattet die relativ einfache Integration von Linux-Containern in den bereits vorhandenen Workflow bzw. die Prozessabläufe eines Unternehmens. Tatsächlich waren offenbar derart viele Menschen von den oben beschriebenen Schwierigkeiten betroffen, dass das Interesse an Docker schneller zunahm, als man vernünftigerweise hätte erwarten dürfen.

Seit seinem Start vor nur wenigen Jahren hat Docker bereits einige Iterationen gesehen, verfügt nun über eine Fülle an Funktionen und läuft weltweit in einer

großen Anzahl von Produktivumgebungen. Es entwickelte sich sehr schnell zu einem der Grundsteine jedes modernen verteilten Systems. Eine Vielzahl von Unternehmen ziehen Docker als Lösung für ernst zu nehmende Probleme in Betracht, mit denen sie sich im Rahmen des Deployments ihrer Anwendungen konfrontiert sehen.

1.3 Was Docker nicht ist

Docker ist für eine breite Palette von Problemstellungen geeignet, für deren Lösung in der Vergangenheit üblicherweise Tools anderer Kategorien herangezogen wurden. Diese Vielseitigkeit bedingt es oftmals aber auch, dass die Funktionalität in bestimmten Bereichen nicht allzu tief greifend ausgeprägt ist. So werden manche Unternehmen feststellen, dass sie komplett auf ihre Tools für die Konfigurationsverwaltung verzichten können, wenn sie Docker einsetzen. Es kann zwar durchaus einige der Aufgaben herkömmlicher Tools übernehmen – seine wahre Stärke zeigt sich jedoch darin, dass es in der Regel mit ihnen kompatibel ist oder sich durch sie ergänzen lässt. Im Folgenden stellen wir einige der Toolkategorien vor, die Docker zwar nicht unmittelbar ersetzt, die sich aber in Kombination verwenden lassen. Auf diese Weise können oft hervorragende Ergebnisse erzielt werden.

- **Enterprise-Virtualisierungsplattform (VMware, KVM usw.)**
  Ein Container ist keine virtuelle Maschine (kurz VM) im herkömmlichen Sinn. Virtuelle Maschinen enthalten ein vollständiges Betriebssystem, das von einem Hypervisor betrieben wird, der innerhalb des Host-Systems ausgeführt wird. Der größte Vorteil gegenüber der Hardwarevirtualisierung besteht darin, dass es problemlos möglich ist, mehrere virtuelle Maschinen mit völlig anderen Betriebssystemen auf einem einzelnen Host auszuführen. Bei einem Container teilen sich der Host und der Container denselben Kernel – das bedeutet, dass der Container weniger Ressourcen belegt als eine VM, aber auf demselben Betriebssystem (z.B. Linux) beruhen muss wie der Host.
- **Cloud-Plattform (OpenStack, CloudStack usw.)**
  Ebenso wie bei Enterprise-Virtualisierungsplattformen haben – oberflächlich gesehen – Container-Workflows und Cloud-Plattformen viele Gemeinsamkeiten. Beide werden typischerweise für die horizontale Skalierung eingesetzt, um wechselndem Leistungsbedarf Rechnung zu tragen. Docker ist allerdings keine Cloud-Plattform, sondern handhabt das Deployment, die Ausführung und die Verwaltung von Containern auf bereits vorhandenen Docker-Hosts. Es ist nicht möglich, neue Hosts (Instanzen), neue Objektspeicher, neuen Speicherplatz oder eine der anderen typischerweise auf einer Cloud-Plattform verfügbaren Ressourcen zu erstellen. Wenn Sie Docker-Tooling ausweiten, werden Sie mehr und mehr von den Vorteilen profitieren, die traditionell mit der Cloud assoziiert werden.

- **Konfigurationsverwaltung (Puppet, Chef usw.)**
  Docker kann zwar die Möglichkeiten zur Administration von Anwendungen und deren Abhängigkeiten erheblich verbessern, ist aber kein direkter Ersatz für eine herkömmliche Konfigurationsverwaltung. Dockerfiles legen fest, wie ein fertiger Container zur Build-Zeit aussehen soll, haben aber keinen Einfluss auf den aktuellen Zustand und können auch nicht verwendet werden, um das Docker-Host-System zu verwalten. Nichtsdestotrotz kann die Nutzung von Docker dazu führen, dass die Komplexität von Konfigurationsverwaltungscode deutlich reduziert wird. Allein dadurch, dass mehr und mehr Server einfach zu Docker-Hosts werden, wird der Code der Konfigurationsverwaltung viel kleiner, und Docker kann anschließend dafür verwendet werden, die komplexen Anforderungen einer Anwendung innerhalb von Docker-Images auszurollen.
- **Deployment-Framework (Capistrano, Fabric usw.)**
  Docker erleichtert viele Aspekte des Deployments, indem es eigenständige Container-Images erstellt, die alle Abhängigkeiten einer Anwendung kapseln und ohne weitere Anpassungen für alle Umgebungen geeignet sind. Es kann jedoch nicht verwendet werden, um die komplexen Deployments selbst zu automatisieren. Für die Aneinanderreihung umfangreicherer automatisierter Workflows werden im Allgemeinen zusätzliche Tools benötigt. Diese Methode verlangt, dass alle deployten Container auf allen Hosts konsistent ausgerollt werden. Ein einziger Deployment-Workflow würde für die meisten – wenn nicht sogar alle – Docker-basierten Anwendungen ausreichen.
- **Workload-Manager (Mesos, Kubernetes, Swarm usw.)**
  Ein Orchestrierungslayer (inklusive des eingebauten Swarm-Modus) muss genutzt werden, wenn man die Workload auf einen Pool von Docker-Hosts intelligent verteilen möchte. Er trackt den aktuellen Status aller Hosts und der genutzten Ressourcen und unterhält ein Inventar von allen laufenden Containern.
- **Entwicklungsumgebung (Vagrant usw.)**
  Vagrant ist ein Developer-Tool zur Verwaltung virtueller Maschinen, das häufig zur Simulation einer Serverumgebung genutzt wird, die der echten Produktivumgebung, in der eine Anwendung laufen soll, sehr ähnlich ist. Unter anderem vereinfacht Vagrant die Ausführung von Linux-Software auf Rechnern mit Windows oder macOS. Docker Machine ist für viele Standard-Docker-Workflows ausreichend. Es bringt allerdings nicht die breite Palette an Funktionen mit, die in Vagrant zu finden sind, da es sehr stark auf den speziellen Docker-Workflow ausgerichtet ist. Wie auch bei vielen der vorherigen Beispiele ist es allerdings nicht notwendig, eine breite Palette von Produktivumgebungen in der Entwicklungsumgebung zu simulieren, wenn Sie Docker einsetzen. Dies hängt damit zusammen, dass die meisten Produktivumgebungen einfache Docker-Server sein werden, die auch leicht lokal gebaut werden können.

1.4 Wichtige Begrifflichkeiten

Hier werden einige Begrifflichkeiten kurz genannt und erläutert, die im weiteren Verlauf des Buchs verwendet werden:

- **Docker-Client**
  Das ist der `docker`-Befehl, der im Docker-Workflow am meisten genutzt wird, um mit Docker-Servern zu arbeiten.
- **Docker-Server**
  Das ist der `dockerd`-Befehl, der genutzt wird, um den Docker-Server-Prozess zu starten, der über einen Client Container baut und startet.
- **Docker-Image**
  Docker-Images bestehen aus einem oder mehreren Dateisystemlayern, angereichert mit einigen wichtigen Metadaten, die alle Dateien repräsentieren, die eine dockerisierte Anwendung benötigt. Ein einzelnes Docker-Image kann auf zahlreiche Hosts kopiert werden. Ein Image hat üblicherweise sowohl einen Namen als auch ein Tag. Das Tag wird generell zur Identifizierung einer spezifizierten Version eines Images nutzt.
- **Docker-Container**
  Ein Docker-Container ist ein Linux-Container, der von einem Docker-Image instanziiert wird. Ein spezifischer Container existiert genau ein einziges Mal. Allerdings kann es mehrere Container von ein und demselben Image geben.
- **Atomic Host**
  Ein Atomic Host ist ein kleines Betriebssystemabbild wie Fedora CoreOS (`https://getfedora.org/en/coreos/`), das das Hosting von Containern erlaubt sowie Image-basierte (atomic) Betriebssystem-Upgrades ermöglicht.

1.5 Zusammenfassung

Sofern Sie nicht den entsprechenden beruflichen Hintergrund besitzen, ist es nicht ganz einfach, Docker zu verstehen. Im folgenden Kapitel geben wir Ihnen einen umfangreichen Überblick darüber, was Docker eigentlich leistet, wie es eingesetzt werden sollte und welche Vorteile Sie erzielen können, wenn Sie all das bei der Implementierung berücksichtigen.