Vorwort

Die weitverbreitete technische Evolution, die sich um uns herum vollzieht, konzentriert sich auf ein scheinbar einfaches Tool: den Container. Etwas vom Design her so Kleines und Leichtgewichtiges hat einen gewaltigen Einfluss auf die Softwareentwicklung in allen Branchen. Und das innerhalb kürzester Zeit.

Containerisierung ist allerdings nicht neu und war es auch 2013 nicht, als Docker zum ersten Mal vorgestellt wurde. Allerdings war die Containerisierung vor Docker bei den meisten Softwareentwicklern kaum auf dem Radar. Selbst die Low-Level-Konzepte hinter Containern wurden überwiegend nur von denen verstanden, die ein tiefes Verständnis vom Linux-Kernel hatten oder bei einigen der Tech-Giganten wie Sun oder Google arbeiteten. Windows-Entwickler oder Systemadministratoren wurden generell außen vor gelassen. Heutzutage ist es schwer, ein Gespräch über eine Software zu führen, ohne Docker zu erwähnen. Aber wie sind wir zu diesem Punkt gekommen, und wo geht die Reise noch hin?

Wir nutzen Docker nicht mehr, weil es neu ist, oder nur wegen der reinen Technologie. Es wird hauptsächlich verwendet, weil es den Entwicklungsprozess beschleunigt, die Ausgaben für Infrastruktur und Overhead verringert, das Onboarding neuer Entwickler erleichtert und sogar die Barriere zwischen den Entwicklungs- und Administratorenteams reduziert. Windows-Nutzer können dank der Arbeit von Microsoft, Docker und weiteren zahlreichen Open-Source-Software-(OSS-)Anbietern nun ebenfalls von den Vorteilen von Docker profitieren.

Trotz all seiner Vorteile ist Docker jedoch nicht immer ein einfaches Tool. Man muss es richtig verstanden haben, um Cloud-Native-Anwendungen bauen und administrieren zu können. Cloud-Native-Anwendungen sind hochverfügbare, skalierbare Anwendungen, die auf Managed-Cloud-Infrastrukturen laufen. Um diese Resilienz und Skalierbarkeit zu erreichen, muss man auf Container-Orchestrierungstechnologien wie Kubernetes setzen. Zusätzlich dazu sind Cloud-Native-Anwendungen in der Regel serviceorientiert oder folgen dem Microservice-Architektur-Ansatz.

Ich werde oft gefragt, ob Docker virtuelle Maschinen (VMs) ersetzt, ob Microservice-Architekturen eine Voraussetzung sind oder ob Unternehmen lieber alles über Docker vergessen und stattdessen auf Serverless Computing setzen sollten, was immer populärer wird. Die Antwort lautet immer: Nein! Tools im Cloud-Native-Ökosystem sind ein Zusatzmittel und nicht exklusiv. Docker und VMs ste-

hen nicht in Konkurrenz zueinander, sondern sollten gemeinsam genutzt werden, um den maximalen Nutzen herauszuziehen. Serverless Computing ist dann am sinnvollsten, wenn es mit Containern genutzt wird. Ich würde sogar behaupten, dass Serverless Computing überhaupt nicht so populär wäre, wenn es keine kurzlebigen und leichtgewichtigen Container gäbe. Microservices sind auch keine Voraussetzung für Container. Allerdings ziehen Sie mehr Vorteile aus Containern, wenn Ihre Architektur kleinere Services erlaubt.

Der Einsatz von Docker erlaubt den Entwicklern, ihre Zuständigkeiten in den Bereich der Administration auszudehnen und für das, was sie entwickelt haben, die Verantwortung zu übernehmen. Das kann die Zusammenarbeit über Abteilungsgrenzen hinweg fördern, da Details wie Abhängigkeiten auch in den Verantwortungsbereich des Entwicklungsteams fallen statt ausschließlich in den der Administratoren. Darüber hinaus sind Teams so in der Lage, bessere Artefakte zu generieren, die als Eckpunkte für die Dokumentation genutzt werden können: Ein *Dockerfile* und eine *docker-compose.yml* können zusammen die Rolle eines Leitfadens für die Inbetriebnahme des Projekts übernehmen. Neue Entwickler im Team oder Entwickler in Open-Source-Projekten können in wenigen Schritten produktiv arbeiten, wenn diese Dateien existieren. In der Vergangenheit war es oft eine mehrtägige Aufgabe, eine Entwicklungsumgebung aufzusetzen. Heutzutage können wir das mit einem einfachen, reproduzierbaren Workflow ersetzen: Docker installieren, das Repository klonen und `docker-compose up` laufen lassen.

Ähnlich wie das Cloud-Native-Ökosystem ein Hilfsmittel ist, sind auch viele Docker-eigene Tools praktische Hilfsmittel, und das Beherrschen der Grundlagen wird Ihnen helfen, erfolgreicher zu sein. In diesem Buch sollten Sie Kapitel 4 ganz besondere Aufmerksamkeit widmen. Dieses Kapitel beschäftigt sich mit den Container-Images inklusive der wichtigsten Datei im Projekt: dem Dockerfile. Diese Datei ist, neben den anderen Images, die Sie möglicherweise direkt von einer Image-Registry herunterladen, die Basis für Ihre Anwendung. Jeder weitere Layer, wie Container-Orchestrierung, basiert darauf, dass Sie Ihre Anwendung mittels eines Dockerfiles in einem Image paketiert haben. Sie lernen, dass jeder Anwendungscode, unabhängig von Alter, Framework, Sprache oder Architektur, in ein Container-Image paketiert werden kann.

In diesem Buch teilen Sean und Karl ihr umfangreiches kollektives Wissen, um Ihnen das breite theoretische und taktische Verständnis von Docker und dem dazugehörigen Ökosystem zu vermitteln mit dem Ziel, Ihnen zu einem erfolgreichen Start in die Containerisierung zu verhelfen. Während Docker die Entwicklungsprozesse vereinfachen und optimieren kann, ist es aber auch ein mächtiges Tool mit zahlreichen Komplexitätsschichten. Sean und Karl haben dieses Buch sorgfältig zusammengestellt, um sich auf das Wesentliche zu konzentrieren und Ihnen dabei zu helfen, schnell produktiv zu werden und trotzdem die wichtigen Grundlagen zu verstehen.

Saugen Sie das Wissen und die Erfahrung, die auf diesen Seiten geteilt werden, auf und behalten Sie dieses Buch als Nachschlagewerk. Sie werden es nicht bereuen.

-- Laura Frank Tacho

Docker Captain und Director of Engineering, CloudBees

Twitter: @rhein_wein

Über die Autoren

Sean P. Kane ist zur Zeit Lead Site Reliability Engineer bei New Relic. Er war lange als IT-Techniker tätig und hat in sehr breit gefächerten Industriebranchen (Biotechnologie, Verteidigungswesen, Hightechunternehmen) viele verschiedene Posten bekleidet. Zusätzlich zu seinen Tätigkeiten als Autor, Tutor und Speaker über moderne Systemadministration in produktiven Umgebungen ist er begeisteter Urlauber, Wanderer und Camper. Er lebt mit seiner Frau, seinen Kindern und Hunden im pazifischen Nordwesten der USA.

Karl Matthias ist Director of Cloud and Platform Services bei InVision. Zuvor war er Principal Systems Engineer bei Nitro Software und war in den letzten 20 Jahren als Entwickler, Systemadministrator und Netzwerktechniker für Start-ups und verschiedene Fortune-500-Unternehmen tätig. Nach einigen Jahren in Start-ups in Deutschland und Großbrittannien, mit einem kurzem Aufenthalt in seiner Heimat in Portland, Oregon, wohnt er mit seiner Familie in Dublin in Irland. Wenn er sich nicht gerade mit digitalen Dingen beschäftigt, verbringt er seine Zeit mit seinen zwei Töchtern, mit dem Fotografieren mit Vintage Kameras oder fährt eines seiner Fahrräder.