Warum Docker rockt

Mit Containern Zeit und Nerven sparen

Nicht nur für Admins und Webentwickler sind Docker-Container eine nützliche Sache. Jeder, der häufig Software ausprobiert, profitiert davon. Besonders gilt das für Webanwendungen, die oft mühselig aufzusetzen sind. Doch leider birgt Docker auch Gefahren. Wie so oft liegen Wohl und Wehe eng beieinander.

Von Peter Siering

Erfunden worden ist Docker, um Webentwicklern und Administratoren das Leben zu erleichtern: Sie können Anwendungen zu Paketen schnüren, Docker-Images genannt, die sich bequem verteilen lassen. Diese Images liegen in einer Registry, etwa auf den Servern der Firma Docker Inc, aus der sie sich der lokal laufende Docker-Daemon herunterlädt. Mit einem Befehl startet der Docker-Daemon die im Image enthaltene Software als Prozess in einem sogenannten Container. Ein Container verwendet spezielle Technik im Betriebssystem, die Prozesse stärker voneinander trennt als bisher; so erhält ein Container zum Beispiel eine eigene Netzwerkkarte und sieht fremde Prozesse sowie Daten nicht. Ein Image ist für sich vollständig, enthält also alle zum Ausführen der Anwendung nötigen Bibliotheken.

Längst ist Docker aber kein reines Admin- und Entwicklerglück für Linux-Liebhaber mehr. Moderne NAS-Geräte spannen Docker-Images als Funktionserweiterungen ein. Auf dem Raspberry Pi erleichtern sie zunehmend die Installation komplexer Softwareprojekte. Selbst für moderne Windows-Versionen stellt Microsoft höchstselbst Docker bereit: Dort gibt es Container sowohl mit Windows- als auch mit Linux-Inhalt. So hat sich Docker mehr und mehr zu einer Alternative fürs Einrichten und Betreiben von Software gemausert und tritt zumindest auf Serversystemen in Konkurrenz zu Paketmanagern und Vollvirtualisierung.

Profiteure

Von der Vereinfachung profitieren im besonderen Maße Anwendungen, die als Frontend einen Webbrowser bemühen, aber nicht nur. Die Serverseite solcher Webanwendungen hat oft Abhängigkeiten: Sie setzt einen Webserver in bestimmter Konfiguration voraus, greift auf komplexe Frameworks zurück und verlangt moderne spezifische Versionen und besondere Erweiterungen derselben. Das wächst sich bei regulärer Installation schnell in einen Wartungsalbtraum aus. Ein Docker-Image kann das alles zusammenfassen und als reproduzierbare Installation handhabbar machen. Dank Containerisierung laufen mehrere solcher Anwendungen trotz unterschiedlicher Abhängigkeiten friedlich nebeneinander.

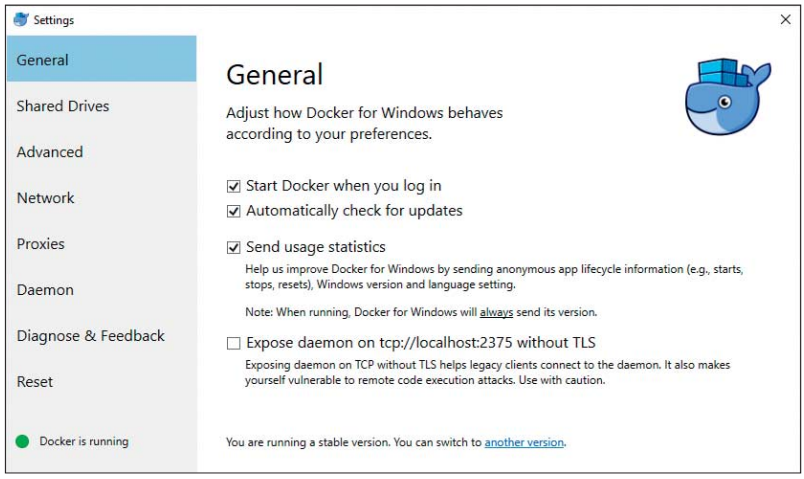
Einzelne Komponenten solcher Anwendungen, etwa eine benötigte Datenbank, steckt man üblicherweise nicht in dasselbe Image, sondern bemüht ein separates und lässt die Container dann übers Netzwerk miteinander reden. So können komplizierte Gebilde voneinander abhängiger Container entstehen, die zusammen eine Anwendung bilden. Das Einrichten ist trotzdem mit einer Befehlszeile getan, dank `docker-compose`. Es startet und verbindet mehrere Container und verarbeitet dazu eine YAML-Datei, in der die Abhängigkeiten untereinander beschrieben sind. Klingt alles prächtig und ist nicht schwer zu lernen [1].

Am Beispiel der Software für einen Magic Mirror [2] lassen sich die Vorzüge einer Installation via Docker schnell erkennen. Das übliche Einrichten des Projekts setzt allein wegen des Shell-Skript-Downloads entweder grenzenloses Vertrauen voraus oder viele einzelne Schritte, in denen unter anderem ein aktuelles Node.js installiert wird – all das hinterlässt im jeweiligen System unter Umständen Spuren abseits offizieller Installationsmechanismen, was erfahrene Admins aufgrund schlechter Wartbarkeit vermeiden. Mit Docker ist das Einrichten ein Einzeiler. Räumt man den Container nebst Image weg, sind alle Spuren getilgt.

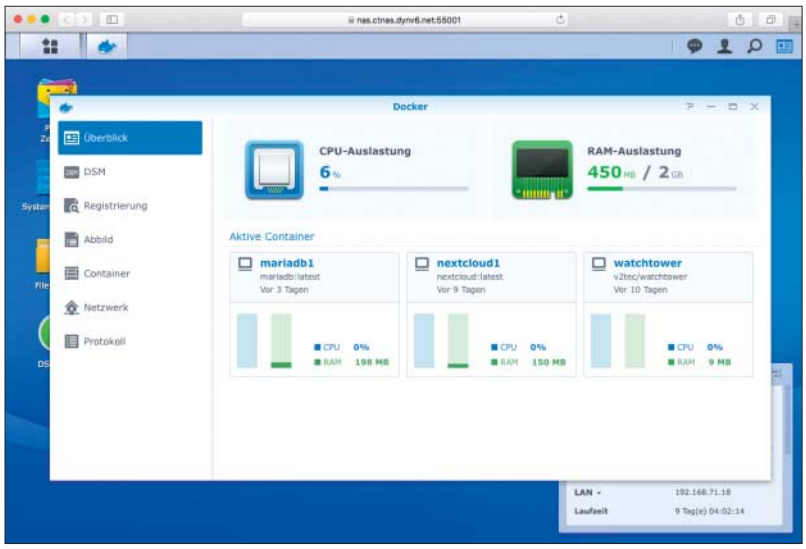
Unter der Haube steckt mehr: Ein Container besteht nicht nur aus den Dateien, die im Image stecken. Er bietet auch Platz für Nutz- und Konfigurationsdaten. Die liegen üblicherweise auf separat vom Docker-Daemon verwalteten Volumes, die unabhängig vom Container existieren. So kann man mit einem neuen Image Software aktualisieren, ohne dadurch die Nutz- oder Konfigurationsdaten zu verlieren. Um eventuellen Havarien vorzubeugen: Volumes sind nur so lange vor Aufräumprozessen gefeit, wie mindestens ein Container sie referenziert.

Praxis-Graben

Docker ist eigentlich zu mehr berufen als für den Einsatz als Sparvirtualisierung auf



Windows 10 kann Container als Feature aktivieren und man muss Docker dann von Hand installieren. Mit Linux-Images ist das durchaus nutzbar, Windows-Images bleiben Mutigen vorbehalten.



Auf x86-NAS-Systemen gehört Docker inzwischen zum guten Ton. Aufgrund der Anpassung an das herstellereigene Ökosystem scheitert der Einsatz allzu anspruchsvoller Images.

einem Host. Es wird gern als Baustein für die Industrialisierung der IT beschrieben: Es soll dafür sorgen, dass sich Anwendungen gut skalieren lassen, indem man bei wachsendem Bedarf einfach weitere Server-Knoten mit Containern beschickt. Dank automatisierter Prozesse für das kontinuierliche Zusammenführen und den Test neuer Features sollen Entwickler und Admins besser kooperieren. Hilfreich dabei sind Verwaltungswerkzeuge wie Kubernetes – für den Einsatz im Kleinen sind die allerdings Overkill.

Was auf den ersten Blick super klingt, nämlich die Möglichkeit, Docker auf verschiedenen Plattformen wie Linux, NAS und Windows zu nutzen, hat in der Praxis viele Tücken. Docker Inc. und Microsoft haben viel Mühe investiert, um sowohl Container mit Windows- als auch Linux-Inhalten ausführen zu können (bei letzteren läuft eine virtuelle Maschine mit Linux im Hintergrund). Doch der Erfolg bei Windows-Containern ist mäßig und der eigentlich beabsichtigte Mischbetrieb scheitert daran, dass sich die Welten im Detail fremd sind [3].

Schwierig wird es besonders dann, wenn es auf die Details der Netzwerkkonfiguration ankommt – das gilt für Windows wie NAS gleichermaßen und für IPv6 sogar für Linux. Docker auf einem Linux-Host kennt verschiedene Spielarten für die Netzwerkanbindung der Container, für die es unter Windows nicht immer eine Entsprechung gibt. Auch die NAS-Hersteller kochen da ihr eigenes Süppchen. Wenn ein Container allzu sehr auf Details im Netz oder des Dateisystems pocht (etwa das Vorhandensein von ACLs), läuft er womöglich nur unter Linux und nicht mal auf einem NAS.

Images im Heuhaufen

Wenn man für eine Aufgabenstellung ein fertiges Container-Image sucht, wird man oft erschlagen. Im Docker-Hub oder im Docker-Store (dem Nachfolger des Hub) finden sich schon für ein und dieselbe Software durchaus mehrere hundert Images. Für deren Güte gibt es grobe Anhaltspunkte, etwa die Anzahl der Downloads. Obendrein sponsort die Firma Docker Inc. ein Team, das „offizielle“ Images besonders beliebter Software erstellt. Letzteres garantiert gewisse Qualitätsstandards – Docker Inc. spricht von „vorbildlichen Images“.

Im Umkehrschluss heißt das jedoch, dass ein Großteil der Images auf dem Docker Hub frei von jeglicher Kontrolle dorthin gelangt: Das nutzte seit Juli 2017 ein Nutzer namens „docker123321“ aus und stellte via Docker Hub Varianten gängiger Images wie mysql bereit (üblicherweise dann als docker123321/mysql), die zur Installation eines Krypto-Miners dienten. Fast ein Jahr lieferte der Docker Hub solche Images aus.

Argwohn ist letztlich des Docker-Fans höchste Tugend: Spätestens für den produktiven Einsatz sollte man Images sehr sorgfältig auf den Zahn fühlen. Über grundlegende Fragestellungen hinaus gibt es viele Details, die ein „vorbildliches“ Image beachten sollte. Manches ist auf den professionellen Einsatz gemünzt, etwa die zur Konfiguration unterstützten Techniken. Unsere Tipps zum Beurteilen von Images erhalten Sie ab Seite 114. Der unmittelbar folgende Artikel versorgt Sie mit einer Liste unserer Erfahrung nach besonders nützlichen Images. *(ps@ct.de)* c't

Literatur

[1] Merlin Schumacher, Docker-Praxis mit Linux, Wo Container punkten, c't 15/2017, S. 106
[2] Peter Siering, S-pi-eglein, S-pi-eglein ..., Raspberry beschreibt Spiegel, c't 6/2017, S. 88
[3] Jan Mahn, Friedliche Koexistenz, Linux- und Windows-Container parallel in Docker unter Windows, c't 5/2018, S. 160

Warum immer nur Docker?

Docker ist hip, aber es handelt sich längst nicht um die einzige Spielart zum Erzeugen und Betreiben von Containern. Letztlich bauen die gängigen Ansätze wie LXC/LXD rkt & Co. ohnehin auf denselben Techniken auf, die sich auch Docker zunutze macht: Prozesse mit Namespaces isolieren, Privilegien über Capabilities feiner regeln und Ressourcenverbrauch per Cgroups kontrollieren. Die Open-Source-Welt will sich aus der Abhängigkeit der Firma Docker Inc. lösen. Die rkt-Entwickler starteten mit dem Anspruch, in Docker fehlende Features zu implementieren. Red Hat tüftelt an kleinteiligen Alternativen und hat das, was Docker tut, in mehrere Befehle zerlegt. Beide kommen ohne ständig laufenden Daemon klar. Sieht man sich in der übrigen Welt um, entsteht der Eindruck, dass Docker-Images für viele Softwareprojekte so selbstverständlich sind wie Installer für Windows und Pakete für gängige Linux-Distributionen. Dass man für die anderen Container-Ökosysteme in freier Wildbahn keine eigene Images findet, liegt daran, dass sie Docker-Images verarbeiten können. Docker wird durch diese Entwicklungen unwichtiger und Firmen können ohne Docker Inc. Geld mit Containern verdienen.