Docker-Praxis mit Linux

Wo Container punkten

Die Container-Technik Docker verändert gerade den Umgang mit Software-Infrastruktur. Von Dockers Flexibilität profitieren nicht nur Unternehmen, sondern auch der eigene Heim- oder Webserver.

Von Merlin Schumacher

Software-Container sind populär, denn sie sind flexibel und man kann sie fast überall einsetzen und reproduzieren. Vor allem im Entwickler- und Cloud-Umfeld hat Docker daher viele Fans gewonnen. Aber nicht nur dort lohnt sich der Einsatz der Containertechnik, auch im kleinen Stil kann man sich damit Arbeit ersparen. Die Docker-eigene Modularität nimmt selbst einem Server-Umzug den Schrecken. Eine große Community von Entwicklern hat viel Vorarbeit erledigt, auf die man aufbauen kann, um eigene maßgeschneiderte Container zu erzeugen.

Docker-Container, das gleich zu Anfang, 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. Bei virtuellen Maschinen gibt es zudem Redundanzen, da jeweils ein vollständiges Gastbetriebssystem läuft. Die Container sparen auch Festplattenplatz, denn sie verwenden mehrfach genutzte Abhängigkeiten in Form von Images wieder.

Images, Schichten und Container

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. Es ist also wie ein Musterhaus: Niemand wohnt darin und es hat nur eine grundlegende und unpersönliche Einrichtung. Erst das Wohnhaus, das auf Basis des Musterhauses gebaut wurde, wird bezogen und von seinen Bewohnern (den Prozessen) zum Eigenheim gemacht.

Ein Image, das die Grundlage für mehrere andere Images darstellt, muss nicht für jeden Container einzeln gespeichert werden. Möglich machen das Dateisystem-Overlays, die einander wie Schichten überlagern.

Erstellt man zum Beispiel ein neues Image auf Basis eines bestehenden MySQL-Images, werden die in der Bauanleitung des MySQL-Images referenzierten Schichten vom Docker Hub heruntergeladen und übereinander gemountet. Die eigenen Änderungen am MySQL-Image landen in einer weiteren Schicht. Die Kombination aller Schichten ist das eigene neue Image.

Wenn auf Basis des Images ein Container gestartet wird, erzeugt Docker eine weitere beschreibbare containerspezifische Datenschicht. Alle Schichten, die Teil des Images sind, werden nur lesbar eingebunden. Deshalb können sie in vielen Containern zugleich verwendet werden. Diese Schichten sind für den im Container laufenden Prozess vollkommen transparent – er merkt von der Schichtung nichts. Will ein laufender Prozess (etwa ein MySQL-Dienst), eine Datei lesen, geht der

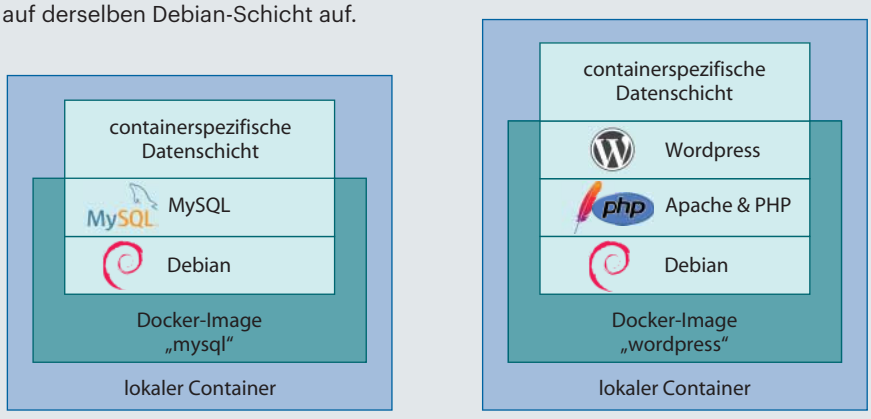
Docker in Windows

Auch in der Windows-Welt ist die Begeisterung für Docker angekommen. Microsoft preist die Segnungen sogenannter Windows-Container. Das Docker-Projekt bietet parallel dazu Docker for Windows an. Für ältere Windows-Versionen gibt es die Docker Toolbox. Gemeinsam haben alle Lösungen, dass unter Windows meist eine virtuelle Maschine zum Einsatz kommt, in der dann die Container laufen.

Wer Microsofts Windows-Container einsetzt, kann dort ausschließlich Windows-Anwendungen ausführen. Für Linux-Anwendungen muss man unter Windows auf Docker for Windows zurückgreifen, dort laufen dann aber auch eben nur Linux-Anwendungen. Im Vergleich zu den Linux-basierten Docker-Containern ist die Begeisterung für Windows-Container und das entsprechende Angebot noch gering.

Betriebssystem-Umgebungen von Containern

Docker-Container erhalten die auszuführende Software und alle von ihr 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.



Linux-Kernel alle Schichten durch, bis er die Datei in einer Schicht findet. Will der Prozess jedoch in eine Datei schreiben, landen die geschriebenen Dateien in der containerspezifischen Datenschicht, denn nur sie ist für den Prozess beschreibbar. Diese Schicht ist fester Teil des Containers und verschwindet, wenn dieser gelöscht wird. (Wie Sie selbst eigene Images bauen, lesen Sie auf S. 110.)

Ein Netzwerk für sich allein

Verbunden sind die Container typischerweise über ein eigenes Netzwerk. Um die Verwaltung dieses Netzwerks kümmert sich Docker selbst. Aber auch hier kann man die Infrastruktur flexibel anpassen. Während früher das Verbinden der Container innerhalb eines Netzes üblich war, lassen sich mittels der neuen User Defined Networks komplexe Netzwerkstrukturen aufbauen (siehe S. 116). So kann man zum Beispiel mehrere Container in ein dediziertes Netzwerk verfrachten, um sie von den übrigen Containern zu trennen. Dabei stellt Docker auf Wunsch nicht nur die automatische Vergabe von IP-Adressen, sondern auch eigene DNS-Dienste bereit, mittels derer sich die Container finden können.

Ein wichtiges Paradigma der Docker-Welt lautet: ein Dienst pro Container. Man wirft also nicht etwa Datenbank und Webserver in dasselbe Image, um eine Web-Anwendung zu starten. Vielmehr steckt man sowohl Datenbank als auch Webserver in ihre eigenen Images.

Parzellen

Ein Blick in die Prozessliste des Host-Betriebssystems zeigt, dass die in den Containern laufenden Prozesse dort genauso aufgelistet werden wie solche, die nicht in Containern stecken. Der Unterschied ist jedoch, dass bestimmte Kernel-Mechanismen die Container vom Rest des Systems und voneinander isolieren. Mittels Control Groups (Cgroups) limitiert der Kernel Ressourcen wie Speicher und Rechenzeit. Namespaces schränken die für den Container zugänglichen auf einen kleinen Teil des Host-Systems ein. Die Prozesse sehen dank Mount-Namespaces zum Beispiel nur einen eingeschränkten Teil des Dateisystems – nur den, den sie zum Betrieb benötigen. Auf den ersten Blick erinnert das an eine chroot-Umgebung. Weitere Techniken wie IPC- und Prozess-Namespaces limitieren die Sichtbarkeit und Erreichbarkeit von Prozessen innerhalb des Containers. Dadurch sehen diese Prozesse keine deranderen Prozesse des Host-Systems oder gar die der anderen Container. Die Kommunikation mit den anderen Prozessen wird ebenfalls unterbunden. Für den Container gibt es daher nur das, was im Container ist. Die Namespaces und Cgroups machen Container erst möglich. Dockers Verdienst ist es, dass es den Umgang mit diesen Techniken so einfach wie möglich gestaltet.

Jede Kopie ein Original

Docker ist bei Entwicklern so beliebt, weil es ein Leichtes ist, damit Software-Umgebungen zu reproduzieren: Ein von Docker erzeugter Container sieht auf jedem Computer gleich aus und hat die gleichen Inhalte. So lässt sich etwa eine anspruchsvolle Web-Anwendung mittels eines oder mehrerer Docker-Container auf jeder Maschine in kürzester Zeit aufsetzen. Dafür muss man keine Rücksicht auf Software-Abhängigkeiten, Bibliotheksversionen oder Paketquellen nehmen. Docker erschafft eine fertige Umgebung mit allem, was die Anwendung benötigt. Besteht die Anwendung aus mehreren Containern, etwa einem Webserver mit Datenbank und CMS, die untereinander Daten austauschen, greift man am besten zu Docker-Compose (siehe S. 120).

Die zentrale Verwaltung von Docker-Images übernimmt zumeist der Docker Hub. Dort finden sich zahllose Images von zahllosen Entwicklern. Genau deswegen sollte man mit Augenmaß vorgehen, wenn man von dort Images bezieht. Nicht alle Images werden regelmäßig aktualisiert oder überhaupt gepflegt. Deshalb bietet Docker sogenannte Official Repositories an. Hier finden Sie Images, auf deren Pflege und Aktualität ein von Docker finanziertes Team ein Auge hat. Das Team achtet laut Docker darauf, dass Sicherheits-Updates zeitnah in diese Images eingebunden werden. Greifen Sie im Zweifel immer zu einem Image aus einem Official Repository. Bei allen anderen Images sollten Sie sichergehen, dass Sie dem Anbieter vertrauen können. *(mls@ct.de)* c't

Docker und die Sicherheit

Container sollten die Sicherheit steigern, Anwendungsumgebungen kleinteilig trennen und die Kommunikation zwischen ihnen unterbinden können, indem sie bewährte Mechanismen des Linux-Kernels wie Prozess- und Netzwerk-Namespaces nutzen. Doch das Gefühl von mehr Sicherheit könnte an gleich zwei Stellen ein Trugschluss sein: Die Technik entwickelt sich noch massiv weiter; so arbeiten Kernel- und Docker-Entwickler mit den „User Namespaces" schon länger an einer Technik, um den Schutz zu verbessern – das Ganze wird standardmäßig aber noch nicht genutzt, denn es bestehen Zweifel an der Ausgereiftheit. Und: Docker erfindet letztlich neue Methoden der Software-Paketierung und -Verteilung – verglichen mit den eingespielten Paketierungs- und Installationstechniken von Linux-Distributionen stecken die noch in den Anfängen.

Für die Versorgung mit Sicherheits-Patches hängen die meisten Container-Images an der Nadel der Distributoren. Erst wenn die ein Update veröffentlicht haben, entsteht ein aktualisiertes Image. Für die „offiziellen" Images soll das binnen 24 Stunden gelingen, sagt Docker.

Bei „nicht offiziellen" Images stockt der Nachschub von Sicherheits-Updates noch ärger. Oft erhalten die Images erst beim nächsten regulären Update durch den Image-Maintainer solche Aktualisierungen und basieren so für lange Zeit auf einem löchrigen Basis-Image – ein Sicherheits-Update für die Basis triggert keinen Image-Neubau der aufbauenden Images.

Wünschenswert wäre es, wenn Software in Docker-Containern grundsätzlich nicht mit root-Rechten liefe. Leider missachten viele gängige Docker-Images die Empfehlung, den Hauptprozess direkt mit reduzierten Rechten zu starten. Dann hat ein Eindringling, der etwa eine Web-Anwendung im Container überrumpelt, ein zu leichtes Spiel: Die – wenn auch wenigen – zugänglichen Kernelschnittstellen kann der Angreifer mit maximalen Rechten benutzen. Der Widerstand, den die Schutzschicht leistet, ist größer als der einer chroot-Umgebung, aber sicher kleiner als der einer virtuellen Maschine. *(ps@ct.de)*