Engineer

Docker

ein Programm samt allem Nötigen verpacken und ausliefern

Was es ist

Docker ist eine Werkzeugsammlung, die ein Programm, seine Bibliotheken, die Laufzeitumgebung seiner Programmiersprache und seine Dateien in ein Artefakt namens Image verpackt und dieses Image als isolierten Prozess namens Container ausführt. Das Image wird einmal nach einem Rezept, dem Dockerfile, gebaut. Dasselbe Image startet dann auf einem Laptop, einem Buildserver und in Produktion. Das ist das ganze Versprechen in einem Satz: Das Getestete ist Byte für Byte das, was läuft.

Vier Substantive prägen den größten Teil des Vokabulars. Ein Dockerfile ist eine Textdatei mit Anweisungen wie `FROM`, `COPY` und `RUN`. Ein Image ist eine daraus gebaute schreibgeschützte Vorlage, gespeichert als Schichtenstapel mit einer Schicht pro Anweisung. Ein Neubau nach einer kleinen Änderung baut so nur die geänderten Schichten neu. Ein Container ist eine laufende Instanz eines Images mit einer dünnen beschreibbaren Schicht darüber. Eine Registry wie Docker Hub oder die eines Cloudanbieters ist der Ort, an den Images gepusht und von dem sie gepullt werden.

Unter der Kommandozeile liegen Client und Server. Der Befehl `docker` ist ein Client. Er sendet Anfragen über eine REST-API, meist durch einen Unix-Socket, an einen dauerhaft laufenden Daemon namens `dockerd`. Der Daemon verwaltet Images, Container, Netzwerke und Volumes. Für den Containerlebenszyklus selbst gibt er die Arbeit an containerd weiter. containerd verwendet standardmäßig runc, das kleine Programm, das den Linux-Kernel um Namespaces und Kontrollgruppen bittet und den Prozess darin startet. Ein Container ist daher ein gewöhnlicher Prozess auf dem Hostkernel mit eingeschränkter Systemsicht. Er ist keine virtuelle Maschine. Auf macOS oder Windows betreibt Docker Desktop eine kleine virtuelle Linux-Maschine, um einen gemeinsam nutzbaren Linux-Kernel zu haben.

Das Imageformat wuchs über sein Ursprungsunternehmen hinaus. Im Juni 2015 gründeten Docker, CoreOS und andere die Open Container Initiative. Docker spendete der Initiative sein Imageformat und runc. Die daraus entstandenen Spezifikationen sind der Grund, warum ein mit Docker gebautes Image unverändert unter Podman, containerd oder einem verwalteten Clouddienst läuft.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist Docker?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. So verpackt, dass sie auf dem Laptop, dem Testserver und im Produktivbetrieb gleich funktioniert und bei höherer Auslastung mitwächst.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. 20 Docker-Apps auf Google Cloud und AWS betrieben — 20 bereitgestellte und überwachte Containeranwendungen

20 Docker-Apps auf Google Cloud und AWS betrieben Kontakt aufnehmen →

Warum es wichtig ist

Die nützliche Frage lautet nie abstrakt „Docker oder nicht“. Sie lautet, ob eine bestimmte Arbeitslast ein Image rechtfertigt oder ob ein einfacher Dienst auf dem Host dieselbe Aufgabe mit weniger Bestandteilen erledigen würde.

Ein Image lohnt sich, wenn das Programm eine eigene Umgebung mitbringt. Eine Python- oder R-Anwendung mit einem genau festgelegten Paketsatz, ein Altdienst, der auf eine alte Laufzeit festgelegt ist, zwei Programme mit widersprüchlichen Versionsanforderungen an dieselbe Bibliothek: All das ist von Hand mühsam und aus einem Image trivial zu reproduzieren. Ein Rollback bedeutet, das vorherige Image zu starten. Ein neuer Entwickler führt einen Befehl aus, statt einer Seite voller Einrichtungshinweise zu folgen. Der Vergleich zwischen Docker und einer virtuellen Maschine zeigt dieselbe Abwägung von der anderen Seite: Container starten in Augenblicken und teilen einen Kernel. Deshalb passen Dutzende auf einen Host, der nur wenige virtuelle Maschinen aufnehmen würde, bei schwächerer Isolation.

Der Preis ist echte Infrastruktur. Standardmäßig läuft der Daemon als root und verwaltet das Netzwerk; Docker unterstützt auch den Rootless-Modus. Es gibt eine zu betreibende oder zu bezahlende Registry, Basis-Images, für die sich Sicherheitskorrekturen ansammeln und die neu gebaut werden müssen, um sie

zu erhalten, und eine Speicherschicht, deren Regeln von denen des Hosts abweichen. Eine einzelne statische Binärdatei ohne Abhängigkeiten gewinnt dadurch wenig. Eine systemd-Unit mit Sandbox führt sie genauso gut aus.

Auch der Unterschied zwischen Docker und Podman zählt hier. Beide führen dieselben Images aus und akzeptieren größtenteils dieselben Befehle. Docker verwendet einen standardmäßig privilegierten Daemon; Podman startet Container direkt als Kindprozesse des Befehls oder von systemd. Auf einem Laptop ist der Unterschied klein. Auf einem Server mit bereits entworfener Firewall und Prozessüberwachung kann er die Entscheidung bestimmen.

Wie es in der Praxis aussieht

Docker taucht in vielen Arbeiten dieser Website auf. Eines der Projekte ist die bewusste Entscheidung, darauf zu verzichten.

Zwischen 2022 und 2024 liefen zwanzig containerisierte Anwendungen auf Google Cloud und AWS, darunter Analyseanwendungen in R, bei denen der genaue Paketsatz das Schwierige an der Reproduktion ist. Die zwanzig wurden mit einheitlicher Konfiguration und Monitoring bereitgestellt und betreut. Das hielt sie stabil und reproduzierbar und beschleunigte die Problemdiagnose über die Clouds hinweg.

In größerem Maßstab liefen hundert Docker-Anwendungen auf Ubuntu-Hosts hinter einer Mischung aus Apache und Nginx. Jede brauchte ein gültiges TLS-Zertifikat. Ein ACME-basierter Ablauf stellte diese Zertifikate automatisch aus, erneuerte sie und stellte sie bereit. So behielten alle hundert gültige Zertifikate, und die manuelle Zertifikatsarbeit entfiel.

Der dritte Fall weist in die andere Richtung. Auf einem Host, dessen Firewallregeln aus einer Härtungsrolle stammen, wurde die Containerschicht auf Podman und Quadlet statt Docker aufgebaut. Denn Dockers Standardnetzwerk schreibt die Paketfilterregeln des Kernels um und veröffentlicht Containerports vor den Regeln der Firewall selbst. Der Grund ist zusammen mit der Entscheidung dokumentiert. So weiß die nächste Person am Host, dass die Standardwahl abgewogen und nicht vergessen wurde.

Wo es schiefgeht

Docker macht den ersten Container leicht. Die Probleme kommen später, meist in einer dieser vier Formen.

Ein von der Firewall gesperrter Port ist aus dem Internet erreichbar. Ein veröffentlichter Port bindet standardmäßig an jede Adresse des Hosts. Die von Docker geschriebenen Paketfilterregeln leiten den Verkehr zum Container, bevor ein Werkzeug wie ufw ihn überhaupt sieht. Die Firewall meldet den Port als geschlossen, obwohl er offen ist. Braucht nur ein lokaler Reverse Proxy den Container, veröffentlichen Sie ihn an der Loopbackadresse:

```
# reachable from anywhere, whatever ufw reports
docker run -d -p 8080:80 nginx:1.27

# reachable only from the host itself
docker run -d -p 127.0.0.1:8080:80 nginx:1.27
```

Daten verschwinden mit dem Container. Alles, was in die eigene beschreibbare Schicht des Containers geschrieben wird, wird beim Entfernen des Containers gelöscht. Genau das geschieht bei einem Upgrade. Datenbanken, Uploads und alles andere, das überleben muss, gehören auf ein benanntes Volume oder einen Bind-Mount. Dieses Volume braucht dann eine eigene Sicherung.

Ein Neubau erzeugt etwas anderes als letzte Woche. Ein Tag wie `python:3` oder `latest` ist ein beweglicher Verweis. Dasselbe Dockerfile kann deshalb an einem anderen Tag ein anderes Image bauen. Fixieren Sie Basis-Images auf eine bestimmte Version oder auf einen Digest, wenn der Build exakt wiederholbar sein muss. Bauen Sie bewusst neu, wenn das Basis-Image Sicherheitskorrekturen veröffentlicht.

Zugangsdaten stecken im Image. Buildargumente und Umgebungsvariablen bleiben im fertigen Image erhalten. Eine in eine Schicht kopierte Datei bleibt dort, selbst wenn eine spätere Anweisung sie löscht. Jeder mit Zugriff zum Pullen des Images kann sie lesen. Übergeben Sie Geheimnisse beim Build durch einen Secret-Mount. Er macht sie einem einzigen `RUN` -Schritt zugänglich, ohne sie in eine Schicht zu schreiben.

Weiterführende Lektüre

- Docker Docs — die offiziellen Handbücher, Anleitungen und Referenzen für Engine, Kommandozeile und Compose.
- Docker-Überblick — Client, Daemon, Images und Container, vom Projekt selbst erklärt.
- Ports veröffentlichen und zuordnen — warum veröffentlichte Ports standardmäßig von außen offen sind und wie sie sich an localhost binden lassen.
- Paketfilterung und Firewalls — wie Docker iptables- oder nftables-Regeln schreibt und wo ufw den Verkehr nicht mehr sieht.
- Geheimnisse beim Build — Zugangsdaten an einen Build übergeben, ohne sie im Image zu hinterlassen.
- OCI-Imageformatspezifikation — der offene Standard, mit dem ein Image unter Docker, Podman und containerd gleichermaßen läuft.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Bereitstellungsweg, die Betriebsrisiken und die Bereitschaftsverantwortlichen. Wir können eine Änderung mit Prüfungen, einem Wiederherstellungsweg und klaren Zuständigkeiten abgrenzen.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist Docker?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/docker/