Kostenloses eBook

LERNEN Docker

Free unaffiliated eBook created from
Stack Overflow contributors.

#docker

Inhaltsverzeichnis

Über

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: docker

It is an unofficial and free Docker ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Docker.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Kapitel 1: Erste Schritte mit Docker

Bemerkungen

Docker ist ein Open-Source- Projekt, das die Bereitstellung von Anwendungen in Software-Containern automatisiert . Diese Anwendungscontainer ähneln leichtgewichtigen virtuellen Maschinen, da sie isoliert voneinander und vom ausgeführten Host ausgeführt werden können.

Docker erfordert Funktionen, die in aktuellen Linux-Kerneln vorhanden sind, damit er ordnungsgemäß funktioniert. Daher ist auf Mac OSX und Windows-Hosts eine virtuelle Maschine erforderlich, auf der Linux ausgeführt wird, damit Docker ordnungsgemäß funktioniert. Die Hauptmethode für die Installation und Einrichtung dieser virtuellen Maschine ist derzeit die Docker Toolbox , die VirtualBox intern verwendet. Es ist jedoch geplant, diese Funktionalität mithilfe der systemeigenen Virtualisierungsfunktionen des Betriebssystems in Docker selbst zu integrieren. Auf Linux-Systemen läuft das Docker nativ auf dem Host selbst.

Versionen

Ausführung	Veröffentlichungsdatum
17.05.0	2017-05-04
17.04.0	2017-04-05
17.03.0	2017-03-01
1.13.1	2016-02-08
1.12.0	2016-07-28
1.11.2	2016-04-13
1.10.3	2016-02-04
1.9.1	2015-11-03
1.8.3	2015-08-11
1.7.1	2015-06-16
1.6.2	2015-04-07
1.5.0	2015-02-10

Examples

Docker unter Mac OS X installieren

Voraussetzungen: OS X 10.8 "Mountain Lion" oder neuer ist erforderlich, um Docker auszuführen.

Während das Docker-Binärprogramm unter Mac OS X nativ ausgeführt werden kann, müssen Sie zum Erstellen und Hosten von Containern eine virtuelle Linux-Maschine auf der Box ausführen.

1.12.0

Seit Version 1.12 muss keine separate VM installiert werden, da Docker die native `Hypervisor.framework` Funktionalität von OSX verwenden kann, um eine kleine Linux-Maschine als Backend zu starten.

Um das Docker zu installieren, gehen Sie wie folgt vor:

1. Gehen Sie zu Docker für Mac
2. Laden Sie das Installationsprogramm herunter und führen Sie es aus.
3. Fahren Sie mit den Standardoptionen durch das Installationsprogramm fort, und geben Sie Ihre Kontoanmeldeinformationen ein, wenn Sie dazu aufgefordert werden.

Hier finden Sie weitere Informationen zur Installation.

1.11.2

Bis zur Version 1.11 ist die beste Methode zum Ausführen dieser Linux-VM die Installation von Docker Toolbox, mit der Docker, VirtualBox und der Linux-Gastcomputer installiert werden.

Um die Docker-Toolbox zu installieren, gehen Sie folgendermaßen vor:

1. Gehen Sie zur Docker Toolbox
2. Klicken Sie auf den Link für Mac und führen Sie das Installationsprogramm aus.
3. Fahren Sie mit den Standardoptionen durch das Installationsprogramm fort, und geben Sie Ihre Kontoanmeldeinformationen ein, wenn Sie dazu aufgefordert werden.

Dadurch werden die Docker-Binärdateien in `/usr/local/bin` installiert und alle vorhandenen Virtual Box-Installationen aktualisiert. Hier finden Sie weitere Informationen zur Installation.

So überprüfen Sie die Installation:

1.12.0

1. Starten Sie `Docker.app` im Anwendungsordner und stellen Sie sicher, dass es ausgeführt wird. Als nächstes öffnen Sie das Terminal.

1.11.2

1. Öffnen Sie das `Docker Quickstart Terminal` , wodurch ein Terminal geöffnet und für die Verwendung für Docker-Befehle vorbereitet wird.

2. Sobald das Terminal geöffnet ist

```
$ docker run hello-world
```

3. Wenn alles in Ordnung ist, sollte eine Willkommensnachricht ausgegeben werden, die bestätigt, dass die Installation erfolgreich war.

Docker unter Windows installieren

Voraussetzungen: 64-Bit-Version von Windows 7 oder höher auf einem Computer, der Hardware-Virtualisierungstechnologie unterstützt, und es ist aktiviert.

Während die Docker-Binärdatei unter Windows nativ ausgeführt werden kann, müssen Sie zum Erstellen und Hosten von Containern eine virtuelle Linux-Maschine auf der Box ausführen.

1.12.0

Seit Version 1.12 muss keine separate VM installiert werden, da Docker die native Hyper-V-Funktionalität von Windows verwenden kann, um eine kleine Linux-Maschine als Backend zu starten.

Um das Docker zu installieren, gehen Sie wie folgt vor:

1. Wechseln Sie zu Docker für Windows
2. Laden Sie das Installationsprogramm herunter und führen Sie es aus.
3. Fahren Sie mit den Standardoptionen durch das Installationsprogramm fort, und geben Sie Ihre Kontoanmeldeinformationen ein, wenn Sie dazu aufgefordert werden.

Hier finden Sie weitere Informationen zur Installation.

1.11.2

Bis zur Version 1.11 ist die beste Methode zum Ausführen dieser Linux-VM die Installation von Docker Toolbox, mit der Docker, VirtualBox und der Linux-Gastcomputer installiert werden.

Um die Docker-Toolbox zu installieren, gehen Sie folgendermaßen vor:

1. Gehen Sie zur Docker Toolbox
2. Klicken Sie auf den Link für Windows und führen Sie das Installationsprogramm aus.
3. Fahren Sie mit den Standardoptionen durch das Installationsprogramm fort, und geben Sie Ihre Kontoanmeldeinformationen ein, wenn Sie dazu aufgefordert werden.

Dadurch werden die Docker-Binärdateien in Programmdateien installiert und alle vorhandenen Virtual Box-Installationen aktualisiert. Hier finden Sie weitere Informationen zur Installation.

So überprüfen Sie die Installation:

1.12.0

1. Starten Sie `Docker` über das Startmenü, falls es noch nicht gestartet wurde, und vergewissern

Sie sich, dass es ausgeführt wird. Als nächstes ein beliebiges Terminal einrichten (entweder `cmd` oder PowerShell)

1.11.2

1. Suchen Sie auf Ihrem Desktop das Docker Toolbox-Symbol. Klicken Sie auf das Symbol, um ein Docker Toolbox-Terminal zu starten.

2. Sobald das Terminal geöffnet ist

```
docker run hello-world
```

3. Wenn alles in Ordnung ist, sollte eine Willkommensnachricht ausgegeben werden, die bestätigt, dass die Installation erfolgreich war.

Docker unter Ubuntu Linux installieren

Docker wird von den folgenden *64-Bit*- Versionen von Ubuntu Linux unterstützt:

- Ubuntu Xenial 16.04 (LTS)
- Ubuntu Wily 15.10
- Ubuntu Trusty 14.04 (LTS)
- Ubuntu Precise 12.04 (LTS)

Ein paar Anmerkungen:

> Die folgenden Anweisungen beziehen sich ausschließlich auf die Installation mit **Docker-** Paketen. **Dadurch wird** sichergestellt, dass Sie die neueste offizielle Version von **Docker erhalten** . Wenn Sie nur mit von `Ubuntu-managed` Paketen installieren müssen, lesen Sie die Ubuntu-Dokumentation (aus offensichtlichen Gründen nicht empfohlen).
>
> Ubuntu Utopic 14.10 und 15.04 sind im Docker-APT-Repository enthalten, werden jedoch aufgrund bekannter Sicherheitsprobleme nicht mehr offiziell unterstützt.

Voraussetzungen

- Docker funktioniert nur bei einer 64-Bit-Installation von Linux.
- Docker erfordert eine Linux-Kernel-Version 3.10 oder höher (außer `Ubuntu Precise 12.04` , für die Version 3.13 oder höher erforderlich ist). Kernel älter als 3.10 verfügen nicht über einige der Funktionen, die zum Ausführen von Docker-Containern erforderlich sind, und enthalten bekannte Fehler, die unter bestimmten Bedingungen zu Datenverlust und häufig zu Panik führen können. Überprüfen Sie die aktuelle Kernel-Version mit dem Befehl `uname -r` . Überprüfen Sie diesen Beitrag, wenn Sie Ihren `Ubuntu Precise (12.04 LTS)` -Kernel aktualisieren `Ubuntu Precise (12.04 LTS)` indem Sie weiter unten scrollen. In diesem WikiHow- Beitrag finden Sie die neueste Version für andere Ubuntu-Installationen.

Aktualisieren Sie die APT-Quellen

Dies ist erforderlich, um auf Pakete aus dem Docker-Repository zugreifen zu können.

1. Melden Sie sich als Benutzer mit `sudo` oder `root` Berechtigungen bei Ihrem Computer an.
2. Öffnen Sie ein Terminalfenster.
3. Aktualisieren Sie die Paketinformationen, und stellen Sie sicher, dass APT mit der https-Methode arbeitet und dass CA-Zertifikate installiert sind.

```
$ sudo apt-get update
$ sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    software-properties-common
```

4. Fügen Sie den offiziellen GPG-Schlüssel von Docker hinzu:

   ```
   $ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
   ```

   Stellen Sie sicher, dass der Schlüsselfingerabdruck **9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88 ist** .

   ```
   $ sudo apt-key fingerprint 0EBFCD88
   ```

```
      pub   4096R/0EBFCD88 2017-02-22
            Key fingerprint = 9DC8 5822 9FC7 DD38 854A  E2D8 8D81 803C 0EBF CD88
      uid                  Docker Release (CE deb) <docker@docker.com>
      sub   4096R/F273FCD8 2017-02-22
```

5. Suchen Sie in der folgenden Tabelle den Eintrag, der Ihrer Ubuntu-Version entspricht. Dies bestimmt, wo APT nach Docker-Paketen sucht. Führen Sie nach Möglichkeit eine Langzeit-Supportversion (LTS) von Ubuntu aus.

Ubuntu-Version	Repository
Genau 12,04 (LTS)	`deb https://apt.dockerproject.org/repo ubuntu-precise main`
Trusty 14.04 (LTS)	`deb https://apt.dockerproject.org/repo ubuntu-trusty main`
Wily 15.10	`deb https://apt.dockerproject.org/repo ubuntu-wily main`
Xenial 16.04 (LTS)	`deb https://apt.dockerproject.org/repo ubuntu-xenial main`

   Hinweis: Docker bietet keine Pakete für alle Architekturen an. Binäre Artefakte werden nachts erstellt und können von `https://master.dockerproject.org` heruntergeladen werden. Um das Docker auf einem System mit mehreren Architekturen zu installieren, fügen Sie dem Eintrag eine Klausel `[arch=...]` . Weitere Informationen finden Sie im Debian-Multiarch-Wiki .

6. Führen Sie den folgenden Befehl aus, und ersetzen Sie den Platzhalter `<REPO>` den Eintrag

Ihres Betriebssystems.

$ echo "" | Sudo tee /etc/apt/sources.list.d/docker.list

7. Aktualisieren Sie den Index des `APT` Pakets, indem `sudo apt-get update` ausführen.

8. Stellen Sie sicher, dass `APT` aus dem rechten Repository abruft.

Wenn Sie den folgenden Befehl ausführen, wird für jede Version von Docker, die für die Installation zur Verfügung steht, ein Eintrag zurückgegeben. Jeder Eintrag sollte die URL `https://apt.dockerproject.org/repo/` . Die aktuell installierte Version ist mit `***` Sehen Sie die Ausgabe des folgenden Beispiels.

```
$ apt-cache policy docker-engine

  docker-engine:
    Installed: 1.12.2-0~trusty
    Candidate: 1.12.2-0~trusty
    Version table:
   *** 1.12.2-0~trusty 0
          500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
          100 /var/lib/dpkg/status
       1.12.1-0~trusty 0
          500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
       1.12.0-0~trusty 0
          500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
```

Wenn Sie jetzt `apt-get upgrade` ausführen, wird `APT` aus dem neuen Repository abgerufen.

Voraussetzungen von Ubuntu Version

Installieren Sie für Ubuntu Trusty (14.04), Wily (15.10) und Xenial (16.04) die Kernel-Pakete `linux-image-extra-*` , mit denen Sie den `aufs` Speichertreiber verwenden können.

So installieren Sie die `linux-image-extra-*` -Pakete:

1. Öffnen Sie ein Terminal auf Ihrem Ubuntu-Host.

2. Aktualisieren Sie Ihren Paketmanager mit dem Befehl `sudo apt-get update` .

3. Installieren Sie die empfohlenen Pakete.

   ```
   $ sudo apt-get install linux-image-extra-$(uname -r) linux-image-extra-virtual
   ```

4. Fahren Sie mit der Docker-Installation fort

Für Ubuntu Precise (12.04 LTS) erfordert Docker die Kernel-Version 3.13. Wenn Ihre Kernel-Version älter als 3.13 ist, müssen Sie sie aktualisieren. In dieser Tabelle erfahren Sie, welche Pakete für Ihre Umgebung erforderlich sind:

Paket	Beschreibung
`linux-image-generic-lts-trusty`	Generisches Linux-Kernel-Image. Dieser Kernel hat `AUFS` eingebaut. Dies ist erforderlich, um Docker auszuführen.
`linux-headers-generic-lts-trusty`	Ermöglicht Pakete, wie `ZFS` und `VirtualBox guest additions` die von ihnen abhängig sind. Wenn Sie die Header für Ihren vorhandenen Kernel nicht installiert haben, können Sie diese Header für den `trusty` Kernel überspringen. Wenn Sie sich nicht sicher sind, sollten Sie dieses Paket zur Sicherheit beilegen.
`xserver-xorg-lts-trusty`	Optional in nicht grafischen Umgebungen ohne Unity / Xorg. **Erforderlich,** wenn Docker auf einem Computer mit einer grafischen Umgebung ausgeführt wird.
`libgl1-mesa-glx-lts-trusty`	Um mehr über die Gründe für diese Pakete zu erfahren, lesen Sie die Installationsanweisungen für zurückportierte Kernel, insbesondere den LTS-Aktivierungsstapel . Siehe Anmerkung 5 unter jeder Version.

Führen Sie folgende Schritte aus, um Ihren Kernel zu aktualisieren und die zusätzlichen Pakete zu installieren:

1. Öffnen Sie ein Terminal auf Ihrem Ubuntu-Host.
2. Aktualisieren Sie Ihren Paketmanager mit dem Befehl `sudo apt-get update` .
3. Installieren Sie die erforderlichen und optionalen Pakete.

```
$ sudo apt-get install linux-image-generic-lts-trusty
```

4. Wiederholen Sie diesen Schritt für andere Pakete, die Sie installieren müssen.
5. Starten Sie Ihren Host neu, um den aktualisierten Kernel mit dem Befehl `sudo reboot` .
6. Nach dem Neustart installieren Sie Docker.

Installieren Sie die neueste Version

Stellen Sie sicher, dass Sie die Voraussetzungen erfüllen, und befolgen Sie nur die folgenden Schritte.

> **Hinweis:** Bei Produktionssystemen wird empfohlen, eine bestimmte Version zu installieren, damit Docker nicht versehentlich aktualisiert wird. Sie sollten Upgrades für Produktionssysteme sorgfältig planen.

1. Melden Sie sich als Benutzer mit `sudo` Berechtigungen bei Ihrer Ubuntu-Installation an. (Möglicherweise `sudo -su`).
2. Aktualisieren Sie Ihren APT-Paketindex, indem Sie `sudo apt-get update` .

3. Installieren Sie Docker Community Edition mit dem Befehl `sudo apt-get install docker-ce` .
4. Starten Sie den `docker` Daemon mit dem Befehl `sudo service docker start` .
5. Stellen Sie sicher, dass das `docker` ordnungsgemäß installiert ist, indem Sie das Hello-World-Image ausführen.

```
$ sudo docker run hello-world
```

Dieser Befehl lädt ein Testbild herunter und führt es in einem Container aus. Wenn der Container ausgeführt wird, druckt er eine Informationsmeldung und wird beendet.

Verwalten Sie Docker als Benutzer ohne Rootberechtigung

Wenn Sie `sudo` bei Verwendung des Docker-Befehls nicht verwenden möchten, erstellen Sie eine Unix-Gruppe mit dem Namen `docker` und fügen Sie Benutzer hinzu. Wenn der `docker` Daemon gestartet wird, macht er den Besitz des Unix-Sockets für die Docker-Gruppe lesbar / schreibbar.

So erstellen Sie die `docker` Gruppe und fügen Ihren Benutzer hinzu:

1. Melden Sie sich als Benutzer mit `sudo` Berechtigungen bei Ubuntu an.
2. Erstellen Sie die `docker` Gruppe mit dem Befehl `sudo groupadd docker` .
3. Fügen Sie Ihren Benutzer der `docker` Gruppe hinzu.

```
$ sudo usermod -aG docker $USER
```

4. Melden Sie sich ab und wieder an, damit Ihre Gruppenmitgliedschaft erneut bewertet wird.
5. Stellen Sie sicher , dass Sie `docker` Befehle ohne `sudo` Erlaubnis.

```
$ docker run hello-world
```

Wenn dies fehlschlägt, wird ein Fehler angezeigt:

```
Cannot connect to the Docker daemon. Is 'docker daemon' running on this host?
```

Überprüfen Sie, ob die Umgebungsvariable `DOCKER_HOST` für Ihre Shell festgelegt ist.

```
$ env | grep DOCKER_HOST
```

Wenn es gesetzt ist, gibt der obige Befehl ein Ergebnis zurück. Wenn ja, setzen Sie es zurück.

```
$ unset DOCKER_HOST
```

Möglicherweise müssen Sie Ihre Umgebung in Dateien wie `~/.bashrc` oder `~/.profile` bearbeiten, um zu verhindern, dass die `DOCKER_HOST` Variable fehlerhaft festgelegt wird.

Docker unter Ubuntu installieren

Voraussetzungen: Docker kann auf jedem Linux mit einem Kernel von mindestens Version 3.10 installiert werden. Docker wird von den folgenden 64-Bit-Versionen von Ubuntu Linux unterstützt:

- Ubuntu Xenial 16.04 (LTS)
- Ubuntu Wily 15.10
- Ubuntu Trusty 14.04 (LTS)
- Ubuntu Precise 12.04 (LTS)

Einfache Installation

Hinweis: Wenn Sie Docker aus dem Standard-Ubuntu-Repository installieren, wird eine alte Version von Docker installiert.

Um die neueste Version von Docker mithilfe des Docker-Repositorys zu installieren, verwenden Sie `curl` , um das von Docker bereitgestellte Installationsskript zu packen und auszuführen:

```
$ curl -sSL https://get.docker.com/ | sh
```

Alternativ kann `wget` zur Installation von Docker verwendet werden:

```
$ wget -qO- https://get.docker.com/ | sh
```

Docker wird jetzt installiert.

Manuelle Installation

Wenn die Ausführung des Installationsskripts jedoch keine Option ist, können die folgenden Anweisungen zum manuellen Installieren der neuesten Version von Docker aus dem offiziellen Repository verwendet werden.

```
$ sudo apt-get update
$ sudo apt-get install apt-transport-https ca-certificates
```

Fügen Sie den GPG-Schlüssel hinzu:

```
$ sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 \
  --recv-keys 58118E89F3A912897C070ADBF76221572C52609D
```

Öffnen Sie anschließend die Datei `/etc/apt/sources.list.d/docker.list` in Ihrem bevorzugten Editor. Wenn die Datei nicht vorhanden ist, erstellen Sie sie. Entfernen Sie alle vorhandenen Einträge. Fügen Sie dann je nach Version die folgende Zeile hinzu:

- Ubuntu Precise 12.04 (LTS):

  ```
  deb https://apt.dockerproject.org/repo ubuntu-precise main
  ```

- Ubuntu Trusty 14.04 (LTS)

```
deb https://apt.dockerproject.org/repo ubuntu-trusty main
```

- Ubuntu Wily 15.10

  ```
  deb https://apt.dockerproject.org/repo ubuntu-wily main
  ```

- Ubuntu Xenial 16.04 (LTS)

  ```
  deb https://apt.dockerproject.org/repo ubuntu-xenial main
  ```

Speichern Sie die Datei und beenden Sie das Programm. Aktualisieren Sie dann Ihren Paketindex, deinstallieren Sie alle installierten Versionen von Docker und stellen Sie sicher, dass `apt` das korrekte Repo verwendet:

```
$ sudo apt-get update
$ sudo apt-get purge lxc-docker
$ sudo apt-cache policy docker-engine
```

Abhängig von Ihrer Ubuntu-Version sind möglicherweise einige Voraussetzungen erforderlich:

- Ubuntu Xenial 16.04 (LTS), Ubuntu Wily 15.10, Ubuntu Trusty 14.04 (LTS)

  ```
  sudo apt-get update && sudo apt-get install linux-image-extra-$(uname -r)
  ```

- Ubuntu Precise 12.04 (LTS)

  Diese Version von Ubuntu erfordert Kernel-Version 3.13. Abhängig von Ihrer Umgebung müssen Sie möglicherweise zusätzliche Pakete installieren:

  ```
  linux-image-generic-lts-trusty
  ```

  Generisches Linux-Kernel-Image. Dieser Kernel hat AUFS eingebaut. Dies ist erforderlich, um Docker auszuführen.

  ```
  linux-headers-generic-lts-trusty
  ```

  Ermöglicht Pakete, wie ZFS- und VirtualBox-Gastergänzungen, die von ihnen abhängig sind. Wenn Sie die Header für Ihren vorhandenen Kernel nicht installiert haben, können Sie diese Header für den `trusty` Kernel überspringen. Wenn Sie sich nicht sicher sind, sollten Sie dieses Paket zur Sicherheit beilegen.

  ```
  xserver-xorg-lts-trusty
  ```

  ```
  libgl1-mesa-glx-lts-trusty
  ```

  Diese beiden Pakete sind in nicht grafischen Umgebungen ohne Unity / Xorg optional. Erforderlich, wenn Docker auf einem Computer mit einer grafischen Umgebung ausgeführt wird.

  Um mehr über die Gründe für diese Pakete zu erfahren, lesen Sie die Installationsanweisungen für zurückportierte Kernel, insbesondere den LTS-Aktivierungsstapel - siehe Hinweis 5 unter jeder Version.

  Installieren Sie die erforderlichen Pakete und starten Sie den Host neu:

```
$ sudo apt-get install linux-image-generic-lts-trusty

$ sudo reboot
```

Aktualisieren Sie schließlich den `apt` Paketindex und installieren Sie Docker:

```
$ sudo apt-get update
$ sudo apt-get install docker-engine
```

Starten Sie den Daemon:

```
$ sudo service docker start
```

Stellen Sie nun sicher, dass das Andockfenster ordnungsgemäß ausgeführt wird, indem Sie ein Testbild starten:

```
$ sudo docker run hello-world
```

Dieser Befehl sollte eine Willkommensnachricht ausgeben, die bestätigt, dass die Installation erfolgreich war.

Erstellen Sie einen Docker-Container in Google Cloud

Sie können Docker ohne Cloud-Daemon (Engine) verwenden, indem Sie Cloud-Anbieter verwenden. In diesem Beispiel sollten Sie über eine `gcloud` (Google Cloud util) verfügen, die mit Ihrem Konto verbunden ist

```
docker-machine create --driver google --google-project `your-project-name` google-machine-type
f1-large fm02
```

In diesem Beispiel wird eine neue Instanz in Ihrer Google Cloud-Konsole erstellt. Mit Maschinenzeit `f1-large`

Installieren Sie Docker unter Ubuntu

Docker wird von den folgenden *64-Bit*- Versionen von Ubuntu Linux unterstützt:

- Ubuntu Xenial 16.04 (LTS)
- Ubuntu Wily 15.10
- Ubuntu Trusty 14.04 (LTS)
- Ubuntu Precise 12.04 (LTS)

Ein paar Anmerkungen:

> Die folgenden Anweisungen beziehen sich ausschließlich auf die Installation mit **Docker-** Paketen. **Dadurch wird** sichergestellt, dass Sie die neueste offizielle Version von **Docker erhalten** . Wenn Sie nur mit von `Ubuntu-managed` Paketen installieren müssen, lesen Sie die Ubuntu-Dokumentation (aus offensichtlichen Gründen nicht empfohlen).

Ubuntu Utopic 14.10 und 15.04 sind im Docker-APT-Repository enthalten, werden jedoch aufgrund bekannter Sicherheitsprobleme nicht mehr offiziell unterstützt.

Voraussetzungen

- Docker funktioniert nur bei einer 64-Bit-Installation von Linux.
- Docker erfordert eine Linux-Kernel-Version 3.10 oder höher (außer `Ubuntu Precise 12.04` , für die Version 3.13 oder höher erforderlich ist). Kernel älter als 3.10 verfügen nicht über einige der Funktionen, die zum Ausführen von Docker-Containern erforderlich sind, und enthalten bekannte Fehler, die unter bestimmten Bedingungen zu Datenverlust und häufig zu Panik führen können. Überprüfen Sie die aktuelle Kernel-Version mit dem Befehl `uname -r` . Überprüfen Sie diesen Beitrag, wenn Sie Ihren `Ubuntu Precise (12.04 LTS)` -Kernel aktualisieren `Ubuntu Precise (12.04 LTS)` indem Sie weiter unten scrollen. In diesem WikiHow- Beitrag finden Sie die neueste Version für andere Ubuntu-Installationen.

Aktualisieren Sie die APT-Quellen

Dies ist erforderlich, um auf Pakete aus dem Docker-Repository zugreifen zu können.

1. Melden Sie sich als Benutzer mit `sudo` oder `root` Berechtigungen bei Ihrem Computer an.
2. Öffnen Sie ein Terminalfenster.
3. Aktualisieren Sie die Paketinformationen, und stellen Sie sicher, dass APT mit der https-Methode arbeitet und dass CA-Zertifikate installiert sind.

```
$ sudo apt-get update
$ sudo apt-get install apt-transport-https ca-certificates
```

4. Fügen Sie den neuen `GPG` Schlüssel hinzu. Mit diesem `58118E89F3A912897C070ADBF76221572C52609D` der Schlüssel mit der ID `58118E89F3A912897C070ADBF76221572C52609D` vom Schlüsselserver `hkp://ha.pool.sks-keyservers.net:80` und dem `adv keychain` . Weitere Informationen finden Sie in der Ausgabe von `man apt-key` .

   ```
   $ sudo apt-key adv \
          --keyserver hkp://ha.pool.sks-keyservers.net:80 \
          --recv-keys 58118E89F3A912897C070ADBF76221572C52609D
   ```

5. Suchen Sie in der folgenden Tabelle den Eintrag, der Ihrer Ubuntu-Version entspricht. Dies bestimmt, wo APT nach Docker-Paketen sucht. Führen Sie nach Möglichkeit eine Langzeit-Supportversion (LTS) von Ubuntu aus.

Ubuntu-Version	Repository
Genau 12,04 (LTS)	`deb https://apt.dockerproject.org/repo ubuntu-precise main`
Trusty 14.04 (LTS)	`deb https://apt.dockerproject.org/repo ubuntu-trusty main`
Wily 15.10	`deb https://apt.dockerproject.org/repo ubuntu-wily main`

Ubuntu-Version	Repository
Xenial 16.04 (LTS)	`deb https://apt.dockerproject.org/repo ubuntu-xenial main`

> **Hinweis:** Docker bietet keine Pakete für alle Architekturen an. Binäre Artefakte werden nachts erstellt und können von `https://master.dockerproject.org` heruntergeladen werden. Um das Docker auf einem System mit mehreren Architekturen zu installieren, fügen Sie dem Eintrag eine Klausel `[arch=...]` . Weitere Informationen finden Sie im Debian-Multiarch-Wiki .

6. Führen Sie den folgenden Befehl aus, und ersetzen Sie den Platzhalter `<REPO>` den Eintrag Ihres Betriebssystems.

   $ echo "" | Sudo tee /etc/apt/sources.list.d/docker.list

7. Aktualisieren Sie den Index des `APT` Pakets, indem `sudo apt-get update` ausführen.

8. Stellen Sie sicher, dass `APT` aus dem rechten Repository abruft.

Wenn Sie den folgenden Befehl ausführen, wird für jede Version von Docker, die für die Installation zur Verfügung steht, ein Eintrag zurückgegeben. Jeder Eintrag sollte die URL `https://apt.dockerproject.org/repo/` . Die aktuell installierte Version ist mit `***` Sehen Sie die Ausgabe des folgenden Beispiels.

```
$ apt-cache policy docker-engine

  docker-engine:
    Installed: 1.12.2-0~trusty
    Candidate: 1.12.2-0~trusty
    Version table:
   *** 1.12.2-0~trusty 0
          500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
          100 /var/lib/dpkg/status
       1.12.1-0~trusty 0
          500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
       1.12.0-0~trusty 0
          500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
```

Wenn Sie jetzt `apt-get upgrade` ausführen, wird `APT` aus dem neuen Repository abgerufen.

Voraussetzungen von Ubuntu Version

Installieren Sie für Ubuntu Trusty (14.04), Wily (15.10) und Xenial (16.04) die Kernel-Pakete `linux-image-extra-*` , mit denen Sie den `aufs` Speichertreiber verwenden können.

So installieren Sie die `linux-image-extra-*` -Pakete:

1. Öffnen Sie ein Terminal auf Ihrem Ubuntu-Host.

2. Aktualisieren Sie Ihren Paketmanager mit dem Befehl `sudo apt-get update` .

3. Installieren Sie die empfohlenen Pakete.

```
$ sudo apt-get install linux-image-extra-$(uname -r) linux-image-extra-virtual
```

4. Fahren Sie mit der Docker-Installation fort

Für Ubuntu Precise (12.04 LTS) erfordert Docker die Kernel-Version 3.13. Wenn Ihre Kernel-Version älter als 3.13 ist, müssen Sie sie aktualisieren. In dieser Tabelle erfahren Sie, welche Pakete für Ihre Umgebung erforderlich sind:

Paket	Beschreibung
`linux-image-generic-lts-trusty`	Generisches Linux-Kernel-Image. Dieser Kernel hat `AUFS` eingebaut. Dies ist erforderlich, um Docker auszuführen.
`linux-headers-generic-lts-trusty`	Ermöglicht Pakete, wie `ZFS` und `VirtualBox guest additions` die von ihnen abhängig sind. Wenn Sie die Header für Ihren vorhandenen Kernel nicht installiert haben, können Sie diese Header für den `trusty` Kernel überspringen. Wenn Sie sich nicht sicher sind, sollten Sie dieses Paket zur Sicherheit beilegen.
`xserver-xorg-lts-trusty`	Optional in nicht grafischen Umgebungen ohne Unity / Xorg. **Erforderlich,** wenn Docker auf einem Computer mit einer grafischen Umgebung ausgeführt wird.
`libgl1-mesa-glx-lts-trusty`	Um mehr über die Gründe für diese Pakete zu erfahren, lesen Sie die Installationsanweisungen für zurückportierte Kernel, insbesondere den LTS-Aktivierungsstapel . Siehe Anmerkung 5 unter jeder Version.

Führen Sie folgende Schritte aus, um Ihren Kernel zu aktualisieren und die zusätzlichen Pakete zu installieren:

1. Öffnen Sie ein Terminal auf Ihrem Ubuntu-Host.
2. Aktualisieren Sie Ihren Paketmanager mit dem Befehl `sudo apt-get update` .
3. Installieren Sie die erforderlichen und optionalen Pakete.

```
$ sudo apt-get install linux-image-generic-lts-trusty
```

4. Wiederholen Sie diesen Schritt für andere Pakete, die Sie installieren müssen.
5. Starten Sie Ihren Host neu, um den aktualisierten Kernel mit dem Befehl `sudo reboot` .
6. Nach dem Neustart installieren Sie Docker.

Installieren Sie die neueste Version

Stellen Sie sicher, dass Sie die Voraussetzungen erfüllen, und befolgen Sie nur die folgenden Schritte.

Hinweis: Bei Produktionssystemen wird empfohlen, eine bestimmte Version zu installieren, damit Docker nicht versehentlich aktualisiert wird. Sie sollten Upgrades für Produktionssysteme sorgfältig planen.

1. Melden Sie sich als Benutzer mit `sudo` Berechtigungen bei Ihrer Ubuntu-Installation an. (Möglicherweise `sudo -su`).
2. Aktualisieren Sie Ihren APT-Paketindex, indem Sie `sudo apt-get update` .
3. Installieren Sie Docker mit dem Befehl `sudo apt-get install docker-engine` .
4. Starten Sie den `docker` Daemon mit dem Befehl `sudo service docker start` .
5. Stellen Sie sicher, dass das `docker` ordnungsgemäß installiert ist, indem Sie das Hello-World-Image ausführen.

```
$ sudo docker run hello-world
```

Dieser Befehl lädt ein Testbild herunter und führt es in einem Container aus. Wenn der Container ausgeführt wird, druckt er eine Informationsmeldung und wird beendet.

Verwalten Sie Docker als Benutzer ohne Rootberechtigung

Wenn Sie `sudo` bei Verwendung des Docker-Befehls nicht verwenden möchten, erstellen Sie eine Unix-Gruppe mit dem Namen `docker` und fügen Sie Benutzer hinzu. Wenn der `docker` Daemon gestartet wird, macht er den Besitz des Unix-Sockets für die Docker-Gruppe lesbar / schreibbar.

So erstellen Sie die `docker` Gruppe und fügen Ihren Benutzer hinzu:

1. Melden Sie sich als Benutzer mit `sudo` Berechtigungen bei Ubuntu an.
2. Erstellen Sie die `docker` Gruppe mit dem Befehl `sudo groupadd docker` .
3. Fügen Sie Ihren Benutzer der `docker` Gruppe hinzu.

```
$ sudo usermod -aG docker $USER
```

4. Melden Sie sich ab und wieder an, damit Ihre Gruppenmitgliedschaft erneut bewertet wird.
5. Stellen Sie sicher , dass Sie `docker` Befehle ohne `sudo` Erlaubnis.

```
$ docker run hello-world
```

Wenn dies fehlschlägt, wird ein Fehler angezeigt:

```
Cannot connect to the Docker daemon. Is 'docker daemon' running on this host?
```

Überprüfen Sie, ob die Umgebungsvariable `DOCKER_HOST` für Ihre Shell festgelegt ist.

```
$ env | grep DOCKER_HOST
```

Wenn es gesetzt ist, gibt der obige Befehl ein Ergebnis zurück. Wenn ja, setzen Sie es zurück.

```
$ unset DOCKER_HOST
```

Möglicherweise müssen Sie Ihre Umgebung in Dateien wie `~/.bashrc` oder `~/.profile` bearbeiten, um zu verhindern, dass die `DOCKER_HOST` Variable fehlerhaft festgelegt wird.

Docker-ce ODER Docker-ee auf CentOS installieren

Docker hat folgende Ausgaben angekündigt:

-Docker-ee (Enterprise Edition) zusammen mit Docker-ce (Community Edition) und Docker (Kommerzieller Support)

Dieses Dokument hilft Ihnen bei der Installation von Docker-ee und Docker-ce Edition in CentOS

Docker-ce-Installation

Im Folgenden finden Sie Schritte zum Installieren der Docker-Ce-Edition

1. Installieren Sie yum-utils, das das Dienstprogramm yum-config-manager enthält:

   ```
   $ sudo yum install -y yum-utils
   ```

2. Verwenden Sie den folgenden Befehl, um das Stable Repository einzurichten:

   ```
   $ sudo yum-config-manager \
    --add-repo \
    https://download.docker.com/linux/centos/docker-ce.repo
   ```

3. Optional: Aktivieren Sie das Edge-Repository. Dieses Repository ist in der obigen Datei docker.repo enthalten, jedoch standardmäßig deaktiviert. Sie können es neben dem stabilen Repository aktivieren.

   ```
   $ sudo yum-config-manager --enable docker-ce-edge
   ```

   - Sie können das Edge-Repository deaktivieren, indem Sie den Befehl `yum-config-manager` mit dem Flag `--disable` . Verwenden Sie zum `--enable` Flag `--enable` . Der folgende Befehl deaktiviert das Edge-Repository.

   ```
   $ sudo yum-config-manager --disable docker-ce-edge
   ```

4. Aktualisieren Sie den YUM-Paketindex.

   ```
   $ sudo yum makecache fast
   ```

5. Installieren Sie das Docker-ce mit folgendem Befehl:

```
$ sudo yum install docker-ce-17.03.0.ce
```

6. Bestätigen Sie den Docker-ce-Fingerabdruck

   `060A 61C5 1B55 8A7F 742B 77AA C52F EB6B 621E 9F35`

   Wenn Sie eine andere Version von docker-ce installieren möchten, können Sie den folgenden Befehl verwenden:

   ```
   $ sudo yum install docker-ce-VERSION
   ```

   `VERSION` die `VERSION` Nummer an

7. Wenn alles gut gelaufen ist, ist das Docker-ce jetzt in Ihrem System installiert. Verwenden Sie zum Starten den folgenden Befehl:

   ```
   $ sudo systemctl start docker
   ```

8. Testen Sie Ihre Docker-Installation:

   ```
   $ sudo docker run hello-world
   ```

   Sie sollten folgende Nachricht erhalten:

   ```
   Hello from Docker!
   This message shows that your installation appears to be working correctly.
   ```

-Docker-ee (Enterprise Edition) Installation

Für die Enterprise Edition (EE) ist eine Anmeldung erforderlich, um Ihre <DOCKER-EE-URL> abzurufen.

1. Um sich anzumelden, gehen Sie zu https://cloud.docker.com/ . Geben Sie Ihre Daten ein und bestätigen Sie Ihre E-Mail-ID. Nach der Bestätigung erhalten Sie eine <DOCKER-EE-URL>, die Sie in Ihrem Dashboard nach dem Klicken auf Setup sehen können.

2. Entfernen Sie alle vorhandenen Docker-Repositorys aus `/etc/yum.repos.d/`

3. Speichern Sie Ihre Docker EE-Repository-URL in einer Yum-Variablen in `/etc/yum/vars/` . Ersetzen Sie <DOCKER-EE-URL> durch die URL, die Sie im ersten Schritt notiert haben.

   ```
   $ sudo sh -c 'echo "<DOCKER-EE-URL>" > /etc/yum/vars/dockerurl'
   ```

4. Installieren Sie yum-utils, das das Dienstprogramm yum-config-manager bereitstellt:

```
$ sudo yum install -y yum-utils
```

5. Verwenden Sie den folgenden Befehl, um das stabile Repository hinzuzufügen:

```
$ sudo yum-config-manager \
--add-repo \
<DOCKER-EE-URL>/docker-ee.repo
```

6. Aktualisieren Sie den YUM-Paketindex.

```
$ sudo yum makecache fast
```

7. Installieren Sie Docker-ee

```
sudo yum install docker-ee
```

8. Sie können das Docker-ee mit folgendem Befehl starten:

```
$ sudo systemctl start docker
```

Erste Schritte mit Docker online lesen: https://riptutorial.com/de/docker/topic/658/erste-schritte-mit-docker

Kapitel 2: Behälter verbinden

Parameter

Parameter	Einzelheiten
`tty:true`	In docker-compose.yml sorgt das Flag `tty: true` dafür, dass der Befehl `sh` des Containers auf Eingabe wartet.

Bemerkungen

Die `host` und `bridge` Netzwerktreiber können Container auf einem einzigen Docker-Host verbinden. Erstellen Sie ein Overlay-Netzwerk, damit Container über eine Maschine kommunizieren können. Die Schritte zum Erstellen des Netzwerks hängen davon ab, wie Ihre Docker-Hosts verwaltet werden.

- `docker network create --driver overlay` : `docker network create --driver overlay`
- Docker / Swarm : erfordert einen externen Schlüsselwertspeicher

Examples

Docker-Netzwerk

Container im selben Docker-Netzwerk haben Zugriff auf freiliegende Ports.

```
docker network create sample
docker run --net sample --name keys consul agent -server -client=0.0.0.0 -bootstrap
```

Die Dockerfile von Consul stellt `8500` , `8600` und mehrere weitere Ports bereit . Führen Sie zur Demonstration einen anderen Container in demselben Netzwerk aus:

```
docker run --net sample -ti alpine sh
/ # wget -qO- keys:8500/v1/catalog/nodes
```

Hier wird der Konsul-Container aus `keys` aufgelöst, der Name wird im ersten Befehl angegeben. Docker bietet DNS-Auflösung in diesem Netzwerk, um Container anhand ihres `--name` zu finden.

Docker komponieren

Netzwerke können in einer Erstellungsdatei (Version 2) angegeben werden. Standardmäßig befinden sich alle Container in einem gemeinsam genutzten Netzwerk.

Beginnen Sie mit dieser Datei: `example/docker-compose.yml` :

```
version: '2'
services:
  keys:
    image: consul
    command: agent -server -client=0.0.0.0 -bootstrap
  test:
    image: alpine
    tty: true
    command: sh
```

Wenn Sie diesen Stack mit `docker-compose up -d example_default` wird ein Netzwerk erstellt, das nach dem übergeordneten Verzeichnis benannt wird, in diesem Fall `example_default` . Überprüfen Sie mit dem `docker network ls`

```
 > docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
719eafa8690b        example_default     bridge              local
```

Stellen Sie eine Verbindung zum Alpencontainer her, um zu überprüfen, ob die Container aufgelöst werden können und kommunizieren können:

```
 > docker exec -ti example_test_1 sh
/ # nslookup keys
...
/ # wget -qO- keys:8500/v1/kv/?recurse
...
```

Eine Erstellungsdatei kann über `networks:` verfügen `networks:` oberster Abschnitt, um den Netzwerknamen, den Treiber und andere Optionen des Docker-Netzwerkbefehls anzugeben.

Containerverknüpfung

Der Docker `--link` Argument und `link:` Abschnitte Docker-compose machen *Aliase* andere Behälter.

```
docker network create sample
docker run -d --net sample --name redis redis
```

Bei Verknüpfung löst entweder der ursprüngliche Name oder das Mapping den Redis-Container auf.

```
> docker run --net sample --link redis:cache -ti python:alpine sh -c "pip install redis &&
python"
>>> import redis
>>> r = redis.StrictRedis(host='cache')
>>> r.set('key', 'value')
True
```

Vor der Docker `1.10.0` Containerverbindung `1.10.0` auch die Netzwerkkonnektivität ein - Verhalten, das jetzt vom Docker-Netzwerk bereitgestellt wird. Links in späteren Versionen bieten nur `legacy` - Effekt auf dem Standard - Brücke Netzwerk.

Behälter verbinden online lesen: https://riptutorial.com/de/docker/topic/6528/behalter-verbinden

Kapitel 3: Beschränkung des Netzwerkzugriffs auf Container

Bemerkungen

Beispiel Andocknetzwerke, die den Verkehr blockieren. `--net` als Netzwerk verwendet, wenn der Container mit `--net` oder `docker network connect --net` .

Examples

Blockieren Sie den Zugriff auf LAN und Out

```
docker network create -o "com.docker.network.bridge.enable_ip_masquerade"="false" lan-restricted
```

- Blöcke
  - Lokales LAN
  - Internet
- Blockiert nicht
  - Host, der einen Docker-Daemon `10.0.1.10:22` (Beispielzugriff auf `10.0.1.10:22`)

Zugriff auf andere Container blockieren

```
docker network create -o "com.docker.network.bridge.enable_icc"="false" icc-restricted
```

- Blöcke
  - Container, die auf andere Container im selben `icc-restricted` Netzwerk zugreifen.
- Blockiert nicht
  - Zugriff auf den Host, der einen Docker-Daemon ausführt
  - Lokales LAN
  - Internet

Blockieren Sie den Zugriff von Containern auf den lokalen Host, der den Docker-Daemon ausführt

```
iptables -I INPUT -i docker0 -m addrtype --dst-type LOCAL -j DROP
```

- Blöcke
  - Zugriff auf den Host, der einen Docker-Daemon ausführt
- Blockiert nicht
  - Container zu Containerverkehr
  - Lokales LAN
  - Internet

- Benutzerdefinierte Docker-Netzwerke, die `docker0` nicht verwenden

Zugriff von Containern auf den lokalen Host blockieren, auf dem der Docker-Daemon ausgeführt wird (benutzerdefiniertes Netzwerk)

```
docker network create --subnet=192.168.0.0/24 --gateway=192.168.0.1 --ip-range=192.168.0.0/25
local-host-restricted
iptables -I INPUT -s 192.168.0.0/24 -m addrtype --dst-type LOCAL -j DROP
```

Erzeugt ein Netzwerk mit dem Namen " `local-host-restricted` , das:

- Blöcke
  - Zugriff auf den Host, der einen Docker-Daemon ausführt
- Blockiert nicht
  - Container zu Containerverkehr
  - Lokales LAN
  - Internet
  - Zugriff von anderen Docker-Netzwerken aus

Benutzerdefinierte Netzwerke haben Brückennamen wie `br-15bbe9bb5bf5` , daher verwenden wir stattdessen das Subnetz.

Beschränkung des Netzwerkzugriffs auf Container online lesen:
https://riptutorial.com/de/docker/topic/6331/beschrankung-des-netzwerkzugriffs-auf-container

Kapitel 4: Bilder bauen

Parameter

Parameter	Einzelheiten
--ziehen	Stellt sicher, dass das Basisimage (`FROM`) aktuell ist, bevor der Rest der Dockerfile erstellt wird.

Examples

Erstellen eines Bildes aus einer Dockerfile

Sobald Sie eine Docker-Datei haben, können Sie mit `docker build` ein Image daraus `docker build` . Die Grundform dieses Befehls lautet:

```
docker build -t image-name path
```

Wenn Ihre Docker-Datei nicht als `Dockerfile` , können Sie mit dem Flag `-f` den Namen der zu erstellenden Docker-Datei angeben.

```
docker build -t image-name -f Dockerfile2 .
```

So erstellen Sie beispielsweise ein Image mit dem Namen `dockerbuild-example:1.0.0` aus einer `Dockerfile` im aktuellen Arbeitsverzeichnis:

```
$ ls
Dockerfile Dockerfile2

$ docker build -t dockerbuild-example:1.0.0 .

$ docker build -t dockerbuild-example-2:1.0.0 -f Dockerfile2 .
```

Weitere `docker build` und Einstellungen finden Sie in der `docker build` Verwendungsdokumentation .

Ein häufiger Fehler ist das Erstellen einer Docker-Datei im Home-Verzeichnis des Benutzers (`~`). Dies ist eine schlechte Idee, weil während `docker build -t mytag .` Diese Meldung wird für lange Zeit angezeigt:

> Kontext hochladen

Die Ursache ist der Docker-Daemon, der versucht, alle Dateien des Benutzers (sowohl das Home-Verzeichnis als auch dessen Unterverzeichnisse) zu kopieren. Vermeiden Sie dies, indem Sie immer ein Verzeichnis für die Docker-Datei angeben.

Das Hinzufügen einer `.dockerignore` Datei zum Build-Verzeichnis ist eine gute Vorgehensweise . Die Syntax ähnelt den `.gitignore` Dateien und stellt sicher, dass nur gewünschte Dateien und Verzeichnisse als Kontext des `.gitignore` hochgeladen werden.

Eine einfache Dockerfile

```
FROM node:5
```

Die `FROM` Anweisung gibt ein Bild an, von dem aus begonnen werden soll. Es kann eine beliebige gültige Bildreferenz verwendet werden.

```
WORKDIR /usr/src/app
```

Die `WORKDIR` Direktive legt das aktuelle Arbeitsverzeichnis innerhalb des Containers fest. `WORKDIR` entspricht der Ausführung von `cd` im Container. (Hinweis: `RUN cd` wird *nicht* das aktuelle Arbeitsverzeichnis geändert werden .)

```
RUN npm install cowsay knock-knock-jokes
```

`RUN` führt den angegebenen Befehl innerhalb des Containers aus.

```
COPY cowsay-knockknock.js ./
```

`COPY` kopiert die Datei oder das Verzeichnis in dem ersten Argumente vom Build - Kontext angegeben (der *`path`* weitergegeben `docker build` *`path`*) zu dem Ort in dem Behälter durch das zweite Argument spezifiziert.

```
CMD node cowsay-knockknock.js
```

`CMD` gibt einen Befehl auszuführen , wenn das Bild ausgeführt und kein Befehl gegeben wird. Es kann überschrieben werden, indem ein Befehl an das `docker run` .

Es gibt viele andere Anweisungen und Optionen. Eine vollständige Liste finden Sie in der Dockerfile-Referenz .

Unterschied zwischen ENTRYPOINT und CMD

Es gibt zwei `Dockerfile` Direktiven, um anzugeben, welcher Befehl standardmäßig in erstellten Images ausgeführt werden soll. Wenn Sie nur `CMD` angeben, führt Docker diesen Befehl mit der Standardeinstellung `ENTRYPOINT` , `ENTRYPOINT /bin/sh -c` . Sie können entweder den Einstiegspunkt und / oder den Befehl überschreiben, wenn Sie das erstellte Image starten. Wenn Sie beides angeben, gibt `ENTRYPOINT` die ausführbare Datei Ihres Containerprozesses an, und `CMD` wird als Parameter dieser ausführbaren Datei angegeben.

Zum Beispiel, wenn Ihre `Dockerfile` enthält

```
FROM ubuntu:16.04
```

```
CMD ["/bin/date"]
```

Dann verwenden Sie die Standard- `ENTRYPOINT` Direktive von `/bin/sh -c` und führen `/bin/date` mit diesem Standard-Einstiegspunkt aus. Der Befehl Ihres Containerprozesses lautet `/bin/sh -c /bin/date` . Sobald Sie dieses Bild ausgeführt haben, wird standardmäßig das aktuelle Datum gedruckt

```
$ docker build -t test .
$ docker run test
Tue Jul 19 10:37:43 UTC 2016
```

Sie können `CMD` in der Befehlszeile überschreiben. In diesem Fall wird der von Ihnen angegebene Befehl ausgeführt.

```
$ docker run test /bin/hostname
bf0274ec8820
```

Wenn Sie eine `ENTRYPOINT` Direktive angeben, verwendet Docker diese ausführbare Datei, und die `CMD` Direktive gibt die Standardparameter des Befehls an. Wenn Ihre `Dockerfile` enthält:

```
FROM ubuntu:16.04
ENTRYPOINT ["/bin/echo"]
CMD ["Hello"]
```

Dann wird es laufen

```
$ docker build -t test .
$ docker run test
Hello
```

Sie können andere Parameter angeben, wenn Sie möchten, aber alle laufen mit `/bin/echo`

```
$ docker run test Hi
Hi
```

Wenn Sie den in Ihrem Dockerfile aufgeführten Eingangspunkt überschreiben möchten (dh, wenn Sie einen anderen Befehl als das `echo` in diesem Container ausführen möchten), müssen Sie den Parameter `--entrypoint` in der Befehlszeile angeben:

```
$ docker run --entrypoint=/bin/hostname test
b2c70e74df18
```

Im Allgemeinen verwenden Sie die `ENTRYPOINT` Direktive, um auf Ihre Hauptanwendung zu zeigen, die Sie ausführen möchten, und `CMD` auf die Standardparameter.

Freilegen eines Ports in der Dockerfile

```
EXPOSE <port> [<port>...]
```

Aus der Dokumentation von Docker:

> Die Anweisung `EXPOSE` informiert Docker darüber, dass der Container zur Laufzeit die angegebenen Netzwerkports abhört. `EXPOSE` macht die Ports des Containers für den Host nicht zugänglich. Dazu müssen Sie entweder das `-p` Flag zum Veröffentlichen eines `-P` oder das `-P` Flag zum Veröffentlichen aller freigelegten Ports verwenden. Sie können eine Portnummer freigeben und extern unter einer anderen Nummer veröffentlichen.

Beispiel:

In Ihrem Dockerfile:

```
EXPOSE 8765
```

Um auf diesen Port vom Host-Computer aus zuzugreifen, fügen Sie dieses Argument in Ihren `docker run` Befehl ein:

```
-p 8765:8765
```

ENTRYPOINT und CMD werden als Verb und Parameter betrachtet

Angenommen, Sie haben eine Dockerfile, die mit endet

```
ENTRYPOINT [ "nethogs"] CMD ["wlan0"]
```

wenn Sie dieses Bild mit einem erstellen

```
docker built -t inspector .
```

Starten Sie das mit einer solchen Docker-Datei erstellte Image mit einem Befehl wie

```
docker run -it --net=host --rm inspector
```

wird das Interface mit dem Namen wlan0 überwacht

Wenn Sie nun die Schnittstelle eth0 (oder wlan1 oder ra1 ...) überwachen möchten, müssen Sie Folgendes tun

```
docker run -it --net=host --rm inspector eth0
```

oder

```
docker run -it --net=host --rm inspector wlan1
```

Pushing und Ziehen eines Bildes an Docker Hub oder eine andere Registry

Lokal erstellte Bilder können an Docker Hub oder einen anderen Docker-Repo-Host, der als Registrierung bezeichnet wird, übertragen werden. Verwenden Sie die `docker login` , um sich bei

einem vorhandenen Docker-Hub-Konto anzumelden.

```
docker login

Login with your Docker ID to push and pull images from Docker Hub.
If you don't have a Docker ID, head over to https://hub.docker.com to create one.

Username: cjsimon
Password:
Login Succeeded
```

Durch Angabe eines Servernamens kann eine andere Docker-Registry verwendet werden. Dies funktioniert auch für private oder selbst gehostete Registries. Darüber hinaus ist die Verwendung eines externen Speichers für Berechtigungsnachweise zur Sicherheit möglich.

```
docker login quay.io
```

Sie können dann Bilder markieren und in die Registrierung verschieben, bei der Sie angemeldet sind. Ihr Repository muss als `server/username/reponame:tag` . Das Auslassen des Servers ist standardmäßig auf Docker Hub festgelegt. (Die Standardregistrierung kann nicht in einen anderen Anbieter geändert werden, und es ist nicht geplant , diese Funktion zu implementieren.)

```
docker tag mynginx quay.io/cjsimon/mynginx:latest
```

Verschiedene Tags können verwendet werden, um verschiedene Versionen oder Zweige desselben Bildes darzustellen. Ein Bild mit mehreren verschiedenen Tags zeigt jedes Tag im selben Repo an.

Verwenden Sie `docker images` , um eine Liste installierter Images anzuzeigen, die auf Ihrem lokalen Computer installiert sind, einschließlich des neu gekennzeichneten Images. Verwenden Sie dann Push, um es in die Registry hochzuladen, und ziehen Sie, um das Bild herunterzuladen.

```
docker push quay.io/cjsimon/mynginx:latest
```

Alle Tags eines Bildes können durch Angabe der Option `-a` abgerufen werden

```
docker pull quay.io/cjsimon/mynginx:latest
```

Erstellen mit einem Proxy

Oft , wenn ein Bild Docker Gebäude enthält die Dockerfile Anweisungen , die Programme ausführt Ressourcen aus dem Internet zu holen (`wget` zum Beispiel ein Programm binären Build auf GitHub zum Beispiel ziehen).

Es ist möglich, Docker anzuweisen, festgelegte Umgebungsvariablen zu übergeben, damit solche Programme diese Abrufe über einen Proxy durchführen:

```
$ docker build --build-arg http_proxy=http://myproxy.example.com:3128 \
```

```
              --build-arg https_proxy=http://myproxy.example.com:3128 \
              --build-arg no_proxy=internal.example.com \
              -t test .
```

`build-arg` sind Umgebungsvariablen, die nur zur Erstellungszeit verfügbar sind.

Bilder bauen online lesen: https://riptutorial.com/de/docker/topic/713/bilder-bauen

Kapitel 5: Bilder verwalten

Syntax

- Docker-Bilder [OPTIONEN] [WIEDERGABE [: TAG]]
- Docker inspizieren [OPTIONEN] CONTAINER | IMAGE [CONTAINER | IMAGE ...]
- Docker ziehen [OPTIONEN] NAME [: TAG | @DIGEST]
- docker rmi [OPTIONEN] BILD [BILD ...]
- Docker-Tag [OPTIONEN] IMAGE [: TAG] [REGISTRYHOST /] [Benutzername] /] NAME [: TAG]

Examples

Abrufen eines Bildes von Docker Hub

Normalerweise werden Bilder automatisch vom Docker Hub abgerufen . Docker versucht, ein beliebiges Bild von Docker Hub abzurufen, das auf dem Docker-Host noch nicht vorhanden ist. Wenn Sie beispielsweise `docker run ubuntu` wenn `ubuntu` Image nicht bereits auf dem Docker-Host vorhanden ist, wird Docker dazu veranlasst, das aktuelle `ubuntu` Image zu ziehen. Sie können ein Bild separat `docker pull` , indem Sie ein `docker pull` , um ein Bild manuell von Docker Hub abzurufen oder zu aktualisieren.

```
docker pull ubuntu
docker pull ubuntu:14.04
```

Zusätzliche Optionen zum Abrufen aus einer anderen Image-Registrierung oder zum Abrufen einer bestimmten Version eines Images sind vorhanden. Das Anzeigen einer alternativen Registrierung erfolgt unter Verwendung des vollständigen Image-Namens und der optionalen Version. Mit dem folgenden Befehl wird beispielsweise versucht, das `ubuntu:14.04` Image aus der `registry.example.com` Registry.example.com zu ziehen:

```
docker pull registry.example.com/username/ubuntu:14.04
```

Lokal heruntergeladene Bilder auflisten

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello-world         latest              693bce725149        6 days ago          967 B
postgres            9.5                 0f3af79d8673        10 weeks ago        265.7 MB
postgres            latest              0f3af79d8673        10 weeks ago        265.7 MB
```

Bilder referenzieren

Docker-Befehle, die den Namen eines Bildes annehmen, akzeptieren vier verschiedene Formen:

Art	Beispiel
Kurze ID	`693bce725149`
Name	`hello-world` *(Standardeinstellung `:latest` tag)*
Name + Tag	`hello-world:latest`
Verdauen	`hello-world@sha256:e52be8ffeeb1f374f440893189cd32f44cb166650e7ab185fa7735b7dc48d619`

Hinweis: Sie können auf ein Bild nur durch seinen Digest verweisen, wenn dieses Bild ursprünglich mit diesem Digest gezogen wurde. Um den Digest für ein Bild `docker images --digests` (falls vorhanden), führen Sie `docker images --digests` .

Bilder entfernen

Der `docker rmi` Befehl `docker rmi` wird zum Entfernen von Bildern verwendet:

```
docker rmi <image name>
```

Der vollständige Bildname muss verwendet werden, um ein Bild zu entfernen. Wenn das Image nicht zum Entfernen des Registrierungsnamens markiert wurde, muss es angegeben werden. Zum Beispiel:

```
docker rmi registry.example.com/username/myAppImage:1.3.5
```

Es ist auch möglich, Bilder nach ihrer ID zu entfernen:

```
docker rmi 693bce725149
```

Der Einfachheit halber ist es möglich, Bilder anhand ihrer Bild-ID zu entfernen, indem Sie nur die ersten paar Zeichen der Bild-ID angeben, sofern die angegebene Unterzeichenfolge eindeutig ist:

```
docker rmi 693
```

> **Hinweis:** Bilder können auch entfernt werden, wenn vorhandene Container dieses Bild verwenden. docker rmi "hebt" das Bild einfach auf.

Wenn kein Container ein Bild verwendet, wird Müll gesammelt. Wenn ein Container ein Bild verwendet, wird das Bild nach dem Entfernen aller Container, die es verwenden, entfernt. Zum Beispiel:

```
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED                  STATUS
PORTS               NAMES
5483657ee07b        hello-world         "/hello"            Less than a second ago   Exited
(0) 2 seconds ago                       small_elion
```

```
$ docker rmi hello-world
Untagged: hello-world:latest

$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED                  STATUS
PORTS               NAMES
5483657ee07b        693bce725149        "/hello"            Less than a second ago   Exited
(0) 12 seconds ago                       small_elion
```

Entfernen Sie alle Bilder ohne gestartete Container

Um alle lokalen Images zu entfernen, für die noch kein Container gestartet wurde, können Sie eine Auflistung der Images als Parameter bereitstellen:

```
docker rmi $(docker images -qa)
```

Alle Bilder entfernen

Wenn Sie Bilder entfernen möchten, unabhängig davon, ob sie einen gestarteten Container haben oder nicht, verwenden Sie das Force-Flag (`-f`):

```
docker rmi -f $(docker images -qa)
```

Entfernen Sie baumelnde Bilder

Wenn ein Bild nicht markiert ist und von keinem Container verwendet wird, ist es "baumelnd" und kann wie folgt entfernt werden:

```
docker images -q --no-trunc -f dangling=true | xargs -r docker rmi
```

Suchen Sie im Docker Hub nach Bildern

Sie können Docker Hub mit dem Suchbefehl nach Bildern durchsuchen :

```
docker search <term>
```

Zum Beispiel:

```
$ docker search nginx
NAME                     DESCRIPTION                                     STARS     OFFICIAL
AUTOMATED
nginx                    Official build of Nginx.                        3565      [OK]
jwilder/nginx-proxy      Automated Nginx reverse proxy for docker c...   717
[OK]
richarvey/nginx-php-fpm  Container running Nginx + PHP-FPM capable ...   232
[OK]
...
```

Bilder prüfen

```
docker inspect <image>
```

Die Ausgabe erfolgt im JSON-Format. Sie können das `jq` um nur die gewünschten Schlüssel zu analysieren und zu drucken.

```
docker inspect <image> | jq -r '.[0].Author'
```

Der obige Befehl zeigt den Autorennamen der Bilder.

Bilder kennzeichnen

Das Markieren eines Bildes ist nützlich, um verschiedene Bildversionen zu verfolgen:

```
docker tag ubuntu:latest registry.example.com/username/ubuntu:latest
```

Ein weiteres Beispiel für das Tagging:

```
docker tag myApp:1.4.2 myApp:latest
docker tag myApp:1.4.2 registry.example.com/company/myApp:1.4.2
```

Speichern und Laden von Docker-Bildern

```
docker save -o ubuntu.latest.tar ubuntu:latest
```

Dieser Befehl speichert das `ubuntu:latest` Abbild als Tarball-Archiv im aktuellen Verzeichnis mit dem Namen `ubuntu.latest.tar` . Dieses Archivarchiv kann dann auf einen anderen Host verschoben werden, z. B. mit `rsync` , oder im Speicher archiviert werden.

Nach dem Verschieben des Archivs erstellt der folgende Befehl ein Bild aus der Datei:

```
docker load -i /tmp/ubuntu.latest.tar
```

Es ist jetzt möglich, Container aus dem `ubuntu:latest` Image wie gewohnt zu erstellen.

Bilder verwalten online lesen: https://riptutorial.com/de/docker/topic/690/bilder-verwalten

Kapitel 6: Container debuggen

Syntax

- Docker-Statistiken [OPTIONEN] [CONTAINER ...]
- Andockprotokolle [OPTIONEN] CONTAINER
- Dockeroberteil [OPTIONEN] CONTAINER [ps OPTIONEN]

Examples

Eingabe in einen laufenden Container

Verwenden Sie zum Ausführen von Vorgängen in einem Container den Befehl `docker exec` . Manchmal wird dies als "Eintreten in den Container" bezeichnet, da alle Befehle innerhalb des Containers ausgeführt werden.

```
docker exec -it container_id bash
```

oder

```
docker exec -it container_id /bin/sh
```

Und jetzt haben Sie eine Shell in Ihrem laufenden Container. Listen Sie beispielsweise Dateien in einem Verzeichnis auf und verlassen Sie den Container:

```
docker exec container_id ls -la
```

Sie können das `-u flag` , um den Container mit einem bestimmten Benutzer `uid=1013` , z. B. `uid=1013` , `gid=1023` .

```
docker exec -it -u 1013:1023 container_id ls -la
```

Die uid und gid müssen nicht im Container vorhanden sein, der Befehl kann jedoch zu Fehlern führen. Wenn Sie einen Container starten und sofort hineingeben möchten, um etwas zu überprüfen, können Sie dies tun

```
docker run...; docker exec -it $(docker ps -lq) bash
```

Der Befehl `docker ps -lq` gibt nur die ID des zuletzt `-lq` Containers (l in `-lq`) aus. (Dies setzt voraus, dass Sie bash als Interpreter in Ihrem Container haben. Sie haben möglicherweise sh oder zsh oder einen anderen.)

Überwachung der Ressourcennutzung

Die Überprüfung der Verwendung von Systemressourcen ist ein effizienter Weg, um fehlerhafte

Anwendungen zu finden. Dieses Beispiel entspricht dem traditionellen Befehl `top` für Container:

```
docker stats
```

Um die Statistiken bestimmter Container zu verfolgen, listen Sie sie in der Befehlszeile auf:

```
docker stats 7786807d8084 7786807d8085
```

Docker-Statistiken zeigen die folgenden Informationen an:

```
CONTAINER     CPU %   MEM USAGE / LIMIT   MEM %    NET I/O                BLOCK I/O
7786807d8084  0.65%   1.33 GB / 3.95 GB   33.67%   142.2 MB / 57.79 MB   46.32 MB / 0 B
```

Standardmäßig zeigt `docker stats` die ID der Container an. Dies ist nicht sehr hilfreich. Wenn Sie die Namen der Container anzeigen möchten, tun Sie dies einfach

```
docker stats $(docker ps --format '{{.Names}}')
```

Prozesse in einem Container überwachen

Die Überprüfung der Verwendung von Systemressourcen ist eine effiziente Methode, um ein Problem in einer live laufenden Anwendung einzugrenzen. Dieses Beispiel entspricht dem traditionellen Befehl `ps` für Container.

```
docker top 7786807d8084
```

Um die Ausgabe zu filtern, fügen Sie der Befehlszeile `ps` Optionen hinzu:

```
docker top 7786807d8084 faux
```

Oder um die Liste der Prozesse als root abzurufen, was eine potenziell schädliche Praxis ist:

```
docker top 7786807d8084 -u root
```

Der `docker top` Befehl des Dockers ist besonders nützlich, wenn minimalistische Container ohne Shell oder mit dem Befehl `ps` .

An einen laufenden Container anhängen

"Anhängen an einen Container" ist das Starten einer Terminalsitzung innerhalb des Kontexts, in dem der Container (und alle darin enthaltenen Programme) ausgeführt wird. Dies wird hauptsächlich zu Debugging-Zwecken verwendet, kann aber auch erforderlich sein, wenn bestimmte Daten an Programme übergeben werden müssen, die im Container ausgeführt werden.

Der Befehl zum `attach` wird dazu verwendet. Es hat diese Syntax:

```
docker attach <container>
```

`<container>` kann entweder die Container-ID oder der Containername sein. Zum Beispiel:

```
docker attach c8a9cf1a1fa8
```

Oder:

```
docker attach graceful_hopper
```

Abhängig von Ihrem Benutzer und der Einrichtung des `sudo` die obigen Befehle möglicherweise erneut `sudo` .

> Anmerkung: Mit "Anhängen" kann nur eine einzelne Shell-Sitzung zu einem Zeitpunkt an einen Container angehängt werden.
>
> Achtung: *Alle* Tastatureingaben werden an den Container weitergeleitet. Wenn Sie `Strg-c` drücken *, wird* der Container gelöscht.

Um sich von einem angefügten Container zu lösen, `drücken Sie` nacheinander die Tasten `Strg-p und` dann `Strg-q`

Um mehrere Shellsitzungen an einen Container anzuhängen, oder einfach als Alternative, können Sie `exec` . Verwenden der Container-ID:

```
docker exec -i -t c8a9cf1a1fa8 /bin/bash
```

Verwenden des Containernamens:

```
docker exec -i -t graceful_hopper /bin/bash
```

`exec` führt ein Programm innerhalb eines Containers aus, in diesem Fall `/bin/bash` (eine Shell (vermutlich eine des Containers)). `-i` bezeichnet eine interaktive Sitzung, während `-t` einen Pseudo-TTY zuweist.

> Hinweis: Im Gegensatz zu *attach* wird durch Drücken von `Strg-c` der Befehl " *exec* " d nur bei interaktiver Ausführung beendet.

Protokolle drucken

Das Aufrufen der Protokolle ist die weniger aufdringliche Art, eine Live-Anwendung zu debuggen. In diesem Beispiel wird das Verhalten des traditionellen `7786807d8084 tail -f some-application.log` im Container `7786807d8084` .

```
docker logs --follow --tail 10 7786807d8084
```

Dieser Befehl zeigt im Wesentlichen die Standardausgabe des Containerprozesses (des Prozesses mit PID 1).

Wenn Ihre Protokolle keine Zeitmarke enthalten, können Sie das Flag `--timestamps` hinzufügen.

Es ist auch möglich, die Protokolle eines angehaltenen Containers anzuzeigen

- Starten Sie den fehlerhaften Container mit `docker run ... ; docker logs $(docker ps -lq)`
- Finden Sie die Container-ID oder den Namen mit

`docker ps -a`

und dann

`docker logs container-id` oder

`docker logs containername`

Es ist möglich, die Protokolle eines angehaltenen Containers anzuzeigen

Docker-Containerprozess-Debugging

Docker ist nur eine ausgefallene Möglichkeit, einen Prozess auszuführen, keine virtuelle Maschine. Daher ist das Debuggen eines Prozesses "in einem Container" auch "auf dem Host" möglich, indem der laufende Containerprozess als Benutzer mit den entsprechenden Berechtigungen zum Überprüfen dieser Prozesse auf dem Host (z. B. root) untersucht wird. Es ist beispielsweise möglich, jeden "Containerprozess" auf dem Host `ps` indem Sie ein einfaches `ps` als root `ps` :

```
sudo ps aux
```

Alle derzeit laufenden Docker-Container werden in der Ausgabe aufgeführt.

Dies kann während der Anwendungsentwicklung hilfreich sein, um einen in einem Container ausgeführten Prozess zu debuggen. Als Benutzer mit entsprechenden Berechtigungen können für den Containerprozess typische Debugging-Dienstprogramme verwendet werden, z. B. strace, ltrace, gdb usw.

Container debuggen online lesen: https://riptutorial.com/de/docker/topic/1333/container-debuggen

Kapitel 7: Container laufen lassen

Syntax

- Docker-Lauf [OPTIONEN] IMAGE [BEFEHL] [ARG ...]

Examples

Einen Container ausführen

```
docker run hello-world
```

Dadurch wird das neueste Hallo-Welt- Image vom Docker Hub abgerufen (falls noch nicht vorhanden), ein neuer Container erstellt und ausgeführt. Es sollte eine Meldung angezeigt werden, dass Ihre Installation anscheinend ordnungsgemäß funktioniert.

Einen anderen Befehl im Container ausführen

```
docker run docker/whalesay cowsay 'Hello, StackExchange!'
```

Dieser Befehl weist Docker an, einen Container aus dem `docker/whalesay` Image zu erstellen und den Befehl `cowsay 'Hello, StackExchange!'` drin. Es sollte ein Bild eines Wals ausdrucken, der `Hello, StackExchange!` sagt `Hello, StackExchange!` zu Ihrem Terminal.

Wenn der Einstiegspunkt im Image der Standard ist, können Sie jeden im Image verfügbaren Befehl ausführen:

```
docker run docker/whalesay ls /
```

Wenn es während des Image-Builds geändert wurde, müssen Sie den Standardwert wiederherstellen

```
docker run --entrypoint=/bin/bash docker/whalesay -c ls /
```

Einen Container nach dem Ausführen automatisch löschen

Normalerweise bleibt ein Docker-Container nach dem Beenden bestehen. Dadurch können Sie den Container erneut ausführen, sein Dateisystem überprüfen und so weiter. Manchmal möchten Sie jedoch einen Container ausführen und sofort nach dem Beenden löschen. Zum Beispiel, um einen Befehl auszuführen oder eine Datei aus dem Dateisystem anzuzeigen. Docker bietet zu diesem Zweck die `--rm` :

```
docker run --rm ubuntu cat /etc/hosts
```

Dadurch wird aus dem Image "ubuntu" ein Container erstellt, der Inhalt der Datei **/ etc / hosts wird** angezeigt und der Container wird unmittelbar nach dem Beenden gelöscht. Dies hilft zu vermeiden, dass Sie die Container nach dem Experimentieren aufräumen müssen.

> **Hinweis:** Das Flag `--rm` funktioniert nicht zusammen mit dem Flag `-d` (`--detach`) im Andockfenster <1.13.0.

Wenn `--rm` Flag `--rm` gesetzt ist, entfernt Docker auch die mit dem Container verknüpften Volumes, wenn der Container entfernt wird. Dies ähnelt dem Ausführen von `docker rm -v my-container` . ***Nur Volumes, die ohne Namen angegeben werden, werden entfernt*** .

Wenn Sie beispielsweise das `docker run -it --rm -v /etc -v logs:/var/log centos /bin/produce_some_logs` , wird der Datenträger von `/etc` entfernt, der Datenträger von `/var/log` jedoch nicht. Volumes, die über --volumes-from geerbt wurden, werden mit derselben Logik entfernt. Wenn das ursprüngliche Volume mit einem Namen angegeben wurde, wird es nicht entfernt.

Einen Namen angeben

Standardmäßig mit erstellten Behältern `docker run` werden einen zufälligen Namen wie angegeben `small_roentgen` oder `modest_dubinsky` . Diese Namen sind nicht besonders hilfreich, um den Zweck eines Containers zu ermitteln. Es ist möglich, einen Namen für den Container `--name` indem Sie die `--name` :

```
docker run --name my-ubuntu ubuntu:14.04
```

Namen müssen eindeutig sein. Wenn Sie einen Namen übergeben, den ein anderer Container bereits verwendet, gibt Docker einen Fehler aus, und es wird kein neuer Container erstellt.

Die Angabe eines Namens ist hilfreich, wenn auf den Container innerhalb eines Docker-Netzwerks verwiesen wird. Dies funktioniert sowohl für Docker-Container im Hintergrund als auch im Vordergrund.

Container im Standard-Bridge-Netzwerk **müssen** verknüpft werden, um über den Namen zu kommunizieren.

Binden eines Containerports an den Host

```
docker run -p "8080:8080" myApp
docker run -p "192.168.1.12:80:80" nginx
docker run -P myApp
```

Um auf dem Host - Ports zu verwenden , haben in einem Bild (über die ausgesetzt worden `EXPOSE` Dockerfile Direktive oder `--expose` Befehlszeilenoption für `docker run`), müssen diese Ports an den Host gebunden werden , um die Verwendung von `-p` oder `-P` Befehl Leitungsoptionen. Für die Verwendung von `-p` müssen der bestimmte Port (und die optionale Hostschnittstelle) angegeben werden. Wenn Sie die Befehlszeilenoption `-P` , wird Docker dazu gezwungen, *alle* freigelegten Ports im Image eines Containers an den Host zu binden.

Container-Neustartrichtlinie (Starten eines Containers beim Booten)

```
docker run --restart=always -d <container>
```

Standardmäßig startet Docker keine Container neu, wenn der Docker-Daemon neu gestartet wird, beispielsweise nach einem Neustart des Hostsystems. Docker stellt eine Neustartrichtlinie für Ihre Container `--restart` indem Sie die `--restart` . Die `--restart=always` einen Neustart eines Containers, nachdem der Docker-Daemon neu gestartet wurde. **Jedoch** , wenn dieser Behälter manuell gestoppt wird (zB mit `docker stop <container>`), wird der Neustart - Richtlinie nicht auf den Behälter aufgebracht werden.

`--restart` nach Anforderung können mehrere Optionen für `--restart` Option `--restart` angegeben werden (`--restart=[policy]`). Diese Optionen wirken sich auch darauf aus, wie der Container beim Booten gestartet wird.

Politik	Ergebnis
Nein	Der **Standardwert** Der Container wird nicht automatisch neu gestartet, wenn der Container gestoppt wird.
On-Failure [: max-retries]	Starten Sie nur dann neu, wenn der Container mit einem Fehler beendet wird (`non-zero exit status`). Um einen unbegrenzten Neustart (bei Problemen) zu vermeiden, können Sie die Anzahl der Neustartversuche des Docker-Dämons begrenzen.
immer	Starten Sie den Container unabhängig vom Exitstatus immer neu. Wenn Sie `always` angeben, versucht der Docker-Daemon, den Container unbegrenzt neu zu starten. Der Container wird auch beim Start des Daemons unabhängig vom aktuellen Status des Containers gestartet.
wenn nicht gestoppt	Starten Sie den Container unabhängig vom Exitstatus immer neu. Starten Sie ihn jedoch nicht beim Start des Daemons, wenn der Container zuvor in den Stoppzustand versetzt wurde.

Führen Sie einen Container im Hintergrund aus

`-d` Befehlszeilenoption `-d` während des Startens des Containers an, um einen Container im Hintergrund laufen zu lassen:

```
docker run -d busybox top
```

Die Option `-d` führt den Container im getrennten Modus aus. Es ist auch äquivalent zu `-d=true` .

Ein Container im getrennten Modus kann nicht automatisch entfernt werden, wenn er angehalten wird. Dies bedeutet, dass die Option --rm nicht zusammen mit der Option -d verwendet werden kann.

Weisen Sie einem Container ein Volume zu

Ein Docker-Volume ist eine Datei oder ein Verzeichnis, das über die Lebensdauer des Containers hinaus erhalten bleibt. Es ist möglich, eine Hostdatei oder ein Hostverzeichnis in einem Container als Volume einzuhängen (unter Umgehung des UnionFS).

Fügen Sie ein Volume mit der Befehlszeilenoption `-v` hinzu:

```
docker run -d -v "/data" awesome/app bootstrap.sh
```

Dadurch wird ein Volume erstellt und in den Pfad `/data` im Container eingebunden.

- Hinweis: Sie können das Flag `--rm` , um das Volume automatisch zu entfernen, wenn der Container entfernt wird.

Mounten von Host-Verzeichnissen

So hängen Sie eine Hostdatei oder ein Verzeichnis in einen Container ein:

```
docker run -d -v "/home/foo/data:/data" awesome/app bootstrap.sh
```

- **Bei der Angabe eines Host-Verzeichnisses muss ein absoluter Pfad angegeben werden.**

Dadurch wird das Hostverzeichnis `/home/foo/data` in `/data` im Container `/home/foo/data` . Dieses Volume, das an ein `mount --bind` Host-Verzeichnis `mount --bind` ist, ist dasselbe wie ein Linux-`mount --bind` und `mount --bind` das Host-Verzeichnis daher vorübergehend über den angegebenen Containerpfad für die Dauer des Containers. Änderungen am Volume des Hosts oder des Containers werden sofort auf dem anderen Datenträger angezeigt, da sie dasselbe Ziel auf der Festplatte sind.

UNIX-Beispiel zum Mounten eines relativen Ordners

```
docker run -d -v $(pwd)/data:/data awesome/app bootstrap.sh
```

Volumes benennen

Ein Volume kann benannt werden, indem anstelle eines Hostverzeichnispfads eine Zeichenfolge angegeben wird. Andernfalls erstellt Docker ein Volume mit diesem Namen.

```
docker run -d -v "my-volume:/data" awesome/app bootstrap.sh
```

Nach dem Erstellen eines benannten Datenträgers kann der Datenträger dann mit anderen Containern mit diesem Namen gemeinsam genutzt werden.

Umgebungsvariablen setzen

```
$ docker run -e "ENV_VAR=foo" ubuntu /bin/bash
```

Mit `-e` und `--env` können Umgebungsvariablen innerhalb eines Containers definiert werden. Es ist möglich, viele Umgebungsvariablen über eine Textdatei bereitzustellen:

```
$ docker run --env-file ./env.list ubuntu /bin/bash
```

Beispiel für Umgebungsvariablendatei:

```
# This is a comment
TEST_HOST=10.10.0.127
```

Das `--env-file` nimmt einen Dateinamen als Argument an und erwartet, dass jede Zeile das Format `VARIABLE=VALUE` , das das an `--env` Argument `--env` . Kommentarzeilen müssen nur mit `#` vorangestellt werden.

Unabhängig von der Reihenfolge dieser drei Flags werden zuerst die `--env-file` und dann `-e` / `--env` Flags verarbeitet. Auf diese Weise überschreiben alle Umgebungsvariablen, die einzeln mit `-e` oder `--env` werden, die in der `--env-var` Variablen.

Angabe eines Hostnamens

Standardmäßig erhalten Container, die mit dem Andocklauf erstellt wurden, einen zufälligen Hostnamen. Sie können dem Container einen anderen Hostnamen geben, indem Sie das Flag --hostname übergeben:

```
docker run --hostname redbox -d ubuntu:14.04
```

Führen Sie einen Container interaktiv aus

Um einen Container interaktiv auszuführen, übergeben Sie die Optionen `-it` :

```
$ docker run -it ubuntu:14.04 bash
root@8ef2356d919a:/# echo hi
hi
root@8ef2356d919a:/#
```

`-i` hält STDIN offen, während `-t` einen Pseudo-TTY zuweist.

Container mit Speicher- / Auslagerungslimits ausführen

Legen Sie das Speicherlimit fest und deaktivieren Sie das Swap-Limit

```
docker run -it -m 300M --memory-swap -1 ubuntu:14.04 /bin/bash
```

Legen Sie sowohl den Speicher als auch den Swap-Grenzwert fest. In diesem Fall kann der Container 300 MB Arbeitsspeicher und 700 MB Swap verwenden.

```
docker run -it -m 300M --memory-swap 1G ubuntu:14.04 /bin/bash
```

Melden Sie sich bei einem laufenden Container an

Ein Benutzer kann einen laufenden Container in einer neuen interaktiven Bash-Shell mit dem Befehl `exec` eingeben.

`jovial_morse` ein Container heißt `jovial_morse` Dann können Sie eine interaktive Pseudo-TTY-Bash-Shell erhalten, indem Sie `jovial_morse` :

```
docker exec -it jovial_morse bash
```

Melden Sie sich bei einem laufenden Container mit einem bestimmten Benutzer an

Wenn Sie einen Container als bestimmten Benutzer `--user` möchten, können Sie ihn mit dem Parameter `-u` oder `--user` . Der Benutzername muss im Container vorhanden sein.

> `-u, --user` Benutzername oder UID (Format: `<name|uid>[:<group|gid>]`)

Dieser Befehl `jovial_morse` mit dem Benutzer des `dockeruser` Benutzers bei `dockeruser`

```
docker exec -it -u dockeruser jovial_morse bash
```

Melden Sie sich als root bei einem laufenden Container an

Wenn Sie sich als Root anmelden möchten, verwenden Sie einfach den **`-u root`** Parameter **`-u root`** . Stammbenutzer existiert immer.

```
docker exec -it -u root jovial_morse bash
```

Loggen Sie sich in ein Bild ein

Sie können sich auch mit dem Befehl `run` in einem Image anmelden. Dies erfordert jedoch einen Image-Namen anstelle eines Containernamens.

```
docker run -it dockerimage bash
```

Anmelden bei einem Zwischenabbild (Debug)

Sie können sich auch bei einem Zwischenabbild anmelden, das während eines Dockerfile-Builds erstellt wird.

Ausgabe des `docker build .`

```
$ docker build .
Uploading context 10240 bytes
Step 1 : FROM busybox
Pulling repository busybox
 ---> e9aa60c60128MB/2.284 MB (100%) endpoint: https://cdn-registry-1.docker.io/v1/
Step 2 : RUN ls -lh /
 ---> Running in 9c9e81692ae9
total 24
drwxr-xr-x    2 root     root        4.0K Mar 12  2013 bin
drwxr-xr-x    5 root     root        4.0K Oct 19 00:19 dev
drwxr-xr-x    2 root     root        4.0K Oct 19 00:19 etc
drwxr-xr-x    2 root     root        4.0K Nov 15 23:34 lib
lrwxrwxrwx    1 root     root           3 Mar 12  2013 lib64 -> lib
dr-xr-xr-x  116 root     root           0 Nov 15 23:34 proc
lrwxrwxrwx    1 root     root           3 Mar 12  2013 sbin -> bin
dr-xr-xr-x   13 root     root           0 Nov 15 23:34 sys
drwxr-xr-x    2 root     root        4.0K Mar 12  2013 tmp
drwxr-xr-x    2 root     root        4.0K Nov 15 23:34 usr
 ---> b35f4035db3f
Step 3 : CMD echo Hello world
 ---> Running in 02071fceb21b
 ---> f52f38b7823e
```

Beachten Sie die `---> Running in 02071fceb21b` . Sie können sich bei diesen Bildern anmelden:

```
docker run -it 02071fceb21b bash
```

Stdin an den Container übergeben

In Fällen wie dem Wiederherstellen eines Datenbank-Dumps oder wenn Sie auf andere Weise einige Informationen vom Host durch eine Pipe pushen möchten, können Sie das Flag `-i` als Argument für das `docker run` oder `docker exec` .

Angenommen, Sie möchten einem in Container befindlichen Mariadb-Client einen Datenbankdump auf dem Host in einer lokalen `dump.sql` Datei `dump.sql` , können Sie den folgenden Befehl ausführen:

```
docker exec -i mariadb bash -c 'mariadb "-p$MARIADB_PASSWORD" ' < dump.sql
```

Im Algemeinen,

```
docker exec -i container command < file.stdin
```

Oder

```
docker exec -i container command <<EOF
inline-document-from-host-shell-HEREDOC-syntax
EOF
```

Trennen von einem Container

Wenn Sie an einen laufenden Container mit einem zugewiesenen Pty angeschlossen sind (`docker run -it ...`), können Sie `Control P` - `Control Q` drücken, um die Verbindung zu trennen.

Überschreibungsrichtlinie für Bilder überschreiben

```
docker run --name="test-app" --entrypoint="/bin/bash" example-app
```

Dieser Befehl überschreibt die `ENTRYPOINT` Direktive des `ENTRYPOINT` der `example-app` wenn die Container `test-app` `ENTRYPOINT` erstellt wird. Die `CMD` Direktive des Bildes bleibt unverändert, sofern nicht anders angegeben:

```
docker run --name="test-app" --entrypoint="/bin/bash" example-app /app/test.sh
```

In dem obigen Beispiel wurden sowohl `ENTRYPOINT` als auch `CMD` des Images überschrieben. Dieser Containerprozess wird zu `/bin/bash /app/test.sh` .

Hosteintrag zum Container hinzufügen

```
docker run --add-host="app-backend:10.15.1.24" awesome-app
```

Dieser Befehl fügt der Datei `/etc/hosts` des Containers einen Eintrag hinzu, der dem Format `--add-host <name>:<address>` folgt. In diesem Beispiel wird der Name `app-backend` in `10.15.1.24` . Dies ist besonders nützlich, wenn Sie verschiedene App-Komponenten programmatisch miteinander verbinden möchten.

Verhindern, dass der Container angehalten wird, wenn keine Befehle ausgeführt werden

Ein Container wird angehalten, wenn im Vordergrund kein Befehl ausgeführt wird. Wenn Sie die Option `-t` wird der Container nicht angehalten, selbst wenn die Option `-d` .

```
docker run -t -d debian bash
```

Einen Container anhalten

```
docker stop mynginx
```

Darüber hinaus kann die Container-ID auch verwendet werden, um den Container anstelle seines

Namens zu stoppen.

Dadurch wird ein laufender Container angehalten, indem das SIGTERM-Signal und ggf. das SIGKILL-Signal gesendet werden.

Außerdem kann mit dem Befehl kill sofort ein SIGKILL oder ein anderes angegebenes Signal mit der Option `-s` gesendet werden.

```
docker kill mynginx
```

Angegebenes Signal:

```
docker kill -s SIGINT mynginx
```

Das Stoppen eines Containers löscht ihn nicht. Verwenden Sie das `docker ps -a` , um den angehaltenen Container `docker ps -a` .

Führen Sie einen anderen Befehl für einen laufenden Container aus

Bei Bedarf können Sie Docker anweisen, zusätzliche Befehle für einen bereits ausgeführten Container mit dem Befehl `exec` auszuführen. Sie benötigen die Container-ID, die Sie mit dem `docker ps` .

```
docker exec 294fbc4c24b3 echo "Hello World"
```

Sie können eine interaktive Shell anhängen, wenn Sie die Option `-it` .

```
docker exec -it 294fbc4c24b3 bash
```

GUI-Apps in einem Linux-Container ausführen

Standardmäßig kann ein Docker-Container keine GUI-Anwendung *ausführen* .

Zuvor muss der X11-Socket zuerst an den Container weitergeleitet werden, damit er direkt verwendet werden kann. Die Umgebungsvariable *DISPLAY* muss ebenfalls weitergeleitet werden:

```
docker run -v /tmp/.X11-unix:/tmp/.X11-unix -e DISPLAY=unix$DISPLAY <image-name>
```

Dies wird zunächst fehlschlagen, da wir keine Berechtigungen für den X-Server-Host festgelegt haben:

```
cannot connect to X server unix:0
```

Der schnellste (aber nicht sicherste) Weg ist der direkte Zugriff mit:

```
xhost +local:root
```

Nach dem Beenden des Containers können wir mit folgenden Schritten in den ursprünglichen

Zustand zurückkehren:

```
xhost -local:root
```

Eine andere (sicherere) Methode ist die Vorbereitung einer Dockerfile, mit der ein neues Image erstellt wird, das die Anmeldeinformationen unserer Benutzer für den Zugriff auf den X-Server verwendet:

```
FROM <iamge-name>
MAINTAINER <you>

# Arguments picked from the command line!
ARG user
ARG uid
ARG gid

#Add new user with our credentials
ENV USERNAME ${user}
RUN useradd -m $USERNAME && \
        echo "$USERNAME:$USERNAME" | chpasswd && \
        usermod --shell /bin/bash $USERNAME && \
        usermod  --uid ${uid} $USERNAME && \
        groupmod --gid ${gid} $USERNAME

USER ${user}

WORKDIR /home/${user}
```

Beim Aufruf des `docker build` Builds über die Befehlszeile müssen die *ARG*- Variablen übergeben werden, die in der Docker-Datei angezeigt werden:

```
docker build --build-arg user=$USER --build-arg uid=$(id -u) --build-arg gid=$(id -g) -t <new-image-with-X11-enabled-name> -f <Dockerfile-for-X11> .
```

Bevor wir einen neuen Container erzeugen, müssen wir eine Xauth-Datei mit Zugriffsberechtigung erstellen:

```
xauth nlist $DISPLAY | sed -e 's/^..../ffff/' | xauth -f /tmp/.docker.xauth nmerge -
```

Diese Datei muss beim Erstellen / Ausführen in den Container gemountet werden:

```
docker run -e DISPLAY=unix$DISPLAY -v /tmp/.X11-unix:/tmp/.X11-unix -v
/tmp/.docker.xauth:/tmp/.docker.xauth:rw -e XAUTHORITY=/tmp/.docker.xauth
```

Container laufen lassen online lesen: https://riptutorial.com/de/docker/topic/679/container-laufen-lassen

Kapitel 8: Container verwalten

Syntax

- docker rm [OPTIONEN] CONTAINER [CONTAINER ...]
- Docker anhängen [OPTIONEN] CONTAINER
- docker exec [OPTIONEN] CONTAINER BEFEHL [ARG ...]
- Docker ps [OPTIONEN]
- Andockprotokolle [OPTIONEN] CONTAINER
- Docker inspizieren [OPTIONEN] CONTAINER | IMAGE [CONTAINER | IMAGE ...]

Bemerkungen

- Wenn in den obigen Beispielen Container ein Parameter des `<CONTAINER_NAME>` ist, wird er als `<container>` oder `container id` oder `<CONTAINER_NAME>` . In allen diesen Bereichen können Sie entweder einen Containernamen oder eine Container-ID übergeben, um einen Container anzugeben.

Examples

Container auflisten

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS                     NAMES
2bc9b1988080        redis               "docker-entrypoint.sh"   2 weeks ago         Up 2
hours          0.0.0.0:6379->6379/tcp    elephant-redis
817879be2230        postgres            "/docker-entrypoint.s"   2 weeks ago         Up 2
hours          0.0.0.0:65432->5432/tcp   pt-postgres
```

`docker ps` druckt allein nur die aktuell ausgeführten Container. Verwenden Sie zum Anzeigen aller Container (einschließlich der angehaltenen) das Flag `-a` :

```
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS                     NAMES
9cc69f11a0f7        docker/whalesay     "ls /"                   26 hours ago        Exited
(0) 26 hours ago                            berserk_wozniak
2bc9b1988080        redis               "docker-entrypoint.sh"   2 weeks ago         Up 2
hours                0.0.0.0:6379->6379/tcp    elephant-redis
817879be2230        postgres            "/docker-entrypoint.s"   2 weeks ago         Up 2
hours                0.0.0.0:65432->5432/tcp   pt-postgres
```

Verwenden Sie zum `-f` Containern mit einem bestimmten Status die Befehlszeilenoption `-f` , um die Ergebnisse zu filtern. Hier ein Beispiel für die Auflistung aller Container, die beendet wurden:

```
$ docker ps -a -f status=exited
```

```
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS                     NAMES
9cc69f11a0f7        docker/whalesay     "ls /"                   26 hours ago        Exited
(0) 26 hours ago
```

Es ist auch möglich, nur die Container-IDs mit der `-q` . Dies macht es sehr einfach, das Ergebnis mit anderen Unix-Dienstprogrammen (wie `grep` und `awk`) zu `awk` :

```
$ docker ps -aq
9cc69f11a0f7
2bc9b1988080
817879be2230
```

Wenn Sie einen Container mit `docker run --name mycontainer1` , geben Sie einen bestimmten Namen und nicht einen zufälligen Namen (in der Form mood_famous, z. B. nostalgic_stallman) an. Mit einem solchen Befehl können Sie diese leicht finden

```
docker ps -f name=mycontainer1
```

Container referenzieren

Docker-Befehle, die den Namen eines Containers annehmen, akzeptieren drei verschiedene Formen:

Art	Beispiel
Volle UUID	`9cc69f11a0f76073e87f25cb6eaf0e079fbfbd1bc47c063bcd25ed3722a8cc4a`
Kurze UUID	`9cc69f11a0f7`
Name	`berserk_wozniak`

Verwenden Sie `docker ps` , um diese Werte für die Container in Ihrem System anzuzeigen.

Die UUID wird von Docker generiert und kann nicht geändert werden. Sie können dem Container einen Namen geben, wenn Sie ihn `docker run --name <given name> <image>` . Docker generiert einen zufälligen Namen für den Container, wenn Sie beim Starten des Containers keinen Namen angeben.

***HINWEIS** : Der Wert der UUID (oder einer "kurzen" UUID) kann beliebig lang sein, solange der angegebene Wert für einen Container eindeutig ist*

Behälter starten und stoppen

So stoppen Sie einen laufenden Container:

```
docker stop <container> [<container>...]
```

Dadurch wird der Hauptprozess im Container ein SIGTERM gesendet, gefolgt von einem

SIGKILL, wenn er nicht innerhalb der Kulanzzeit stoppt. Der Name jedes Containers wird beim Stoppen gedruckt.

So starten Sie einen Container, der gestoppt ist:

```
docker start <container> [<container>...]
```

Dadurch wird jeder im Hintergrund übergebene Container gestartet. Der Name jedes Containers wird beim Start gedruckt. Um den Container im Vordergrund zu starten, übergeben Sie die `--attach -a` (`--attach`).

Listen Sie Container mit benutzerdefiniertem Format auf

```
docker ps --format 'table {{.ID}}\t{{.Names}}\t{{.Status}}'
```

Einen bestimmten Container finden

```
docker ps --filter name=myapp_1
```

Container-IP suchen

Um die IP-Adresse Ihres Containers herauszufinden, verwenden Sie:

```
docker inspect <container id> | grep IPAddress
```

oder Docker inspizieren

```
docker inspect --format '{{ .NetworkSettings.IPAddress }}' ${CID}
```

Docker-Container neu starten

```
docker restart <container> [<container>...]
```

Option **--time** : Sekunden bis zum Anhalten des Containers (Standardeinstellung 10)

```
docker restart <container> --time 10
```

Container entfernen, löschen und bereinigen

`docker rm` kann verwendet werden, um bestimmte Container wie `docker rm` zu entfernen:

```
docker rm <container name or id>
```

Um alle Container zu entfernen, können Sie diesen Ausdruck verwenden:

```
docker rm $(docker ps -qa)
```

Standardmäßig löscht das Andockfenster einen laufenden Container nicht. Jeder laufende Container erzeugt eine Warnmeldung und wird nicht gelöscht. Alle anderen Container werden gelöscht.

Alternativ können Sie `xargs` :

```
docker ps -aq -f status=exited | xargs -r docker rm
```

Wo `docker ps -aq -f status=exited` wird, wird eine Liste der Container-IDs von Containern mit dem Status "Exited" zurückgegeben.

> Warnung: Bei allen obigen Beispielen werden nur "angehaltene" Container entfernt.

Um einen Container zu entfernen, unabhängig davon, ob er gestoppt ist oder nicht, können Sie das Force-Flag `-f` :

```
docker rm -f <container name or id>
```

So entfernen Sie alle Container unabhängig vom Status:

```
docker rm -f $(docker ps -qa)
```

Wenn Sie nur Container entfernen möchten, deren Status " `dead` " ist:

```
docker rm $(docker ps --all -q -f status=dead)
```

Wenn Sie nur Container mit einem `exited` Status entfernen möchten:

```
docker rm $(docker ps --all -q -f status=exited)
```

Dies sind alle Permutationen von Filtern, die beim Auflisten von Containern verwendet werden .

Um sowohl unerwünschte Container als auch baumelnde Images zu entfernen, die nach Version 1.3 Speicherplatz verwenden, verwenden Sie Folgendes (ähnlich dem Unix-Tool `df`):

```
$ docker system df
```

So entfernen Sie alle nicht verwendeten Daten:

```
$ docker system prune
```

Führen Sie den Befehl für einen bereits vorhandenen Docker-Container aus

```
docker exec -it <container id> /bin/bash
```

Es ist üblich, sich in einem bereits laufenden Container anzumelden, um ein paar schnelle Tests durchzuführen oder zu sehen, was die Anwendung macht. Häufig wird darauf hingewiesen, dass die Verwendung von Containern aufgrund von Protokollen schlecht ist und geänderte Dateien in

Volumes abgelegt werden sollten. In diesem Beispiel können wir uns im Container anmelden. Dies setzt voraus, dass / bin / bash im Container verfügbar ist. Dies kann / bin / sh oder etwas anderes sein.

```
docker exec <container id> tar -czvf /tmp/backup.tgz /data
docker cp <container id>:/tmp/backup.tgz .
```

Dieses Beispiel archiviert den Inhalt des Datenverzeichnisses in einem tar. Dann können Sie es mit `docker cp` abrufen.

Containerprotokolle

```
Usage:  docker logs [OPTIONS] CONTAINER

Fetch the logs of a container

  -f, --follow=false        Follow log output
  --help=false              Print usage
  --since=                  Show logs since timestamp
  -t, --timestamps=false    Show timestamps
  --tail=all                Number of lines to show from the end of the logs
```

Zum Beispiel:

```
$ docker ps
CONTAINER ID    IMAGE    COMMAND                  CREATED     STATUS       PORTS
ff9716dda6cb    nginx    "nginx -g 'daemon off"  8 days ago  Up 22 hours  443/tcp,
0.0.0.0:8080->80/tcp

$ docker logs ff9716dda6cb
xx.xx.xx.xx - - [15/Jul/2016:14:03:44 +0000] "GET /index.html HTTP/1.1" 200 511
"https://google.com" "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/50.0.2661.75 Safari/537.36"
xx.xx.xx.xx - - [15/Jul/2016:14:03:44 +0000] "GET /index.html HTTP/1.1" 200 511
"https://google.com" "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/50.0.2661.75 Safari/537.36"
```

Stellen Sie eine Verbindung zu einer Instanz her, die als Daemon ausgeführt wird

Es gibt zwei Möglichkeiten, dies zu erreichen, die erste und bekannteste ist die Folgende:

```
docker attach --sig-proxy=false <container>
```

Dieses fügt Ihre Bash buchstäblich an den Container Bash an, dh wenn Sie ein Skript ausführen, sehen Sie das Ergebnis.

`Geben Sie zum Lösen` einfach ein: `Ctl-P Ctl-Q`

Wenn Sie jedoch eine freundlichere Methode benötigen, um neue Bash-Instanzen erstellen zu können, führen Sie einfach den folgenden Befehl aus:

```
docker exec -it <container> bash
```

Datei von / in Container kopieren

vom Container zum Host

```
docker cp CONTAINER_NAME:PATH_IN_CONTAINER PATH_IN_HOST
```

vom Host zum Container

```
docker cp PATH_IN_HOST CONTAINER_NAME:PATH_IN_CONTAINER
```

Wenn ich jess / send aus benutze

https://hub.docker.com/r/jess/transmission/builds/bsn7eqxrkzrhxazcuytbmzp/

befinden sich die Dateien im Container in / transmission / download

und mein aktuelles Verzeichnis auf dem Host ist / home / $ USER / abc nach

```
docker cp transmission_id_or_name:/transmission/download .
```

Ich werde die Dateien kopieren lassen

```
/home/$USER/abc/transmission/download
```

Sie können nicht mit `docker cp` nur eine Datei kopieren, Sie kopieren den Verzeichnisbaum und die Dateien

Docker-Volumes entfernen, löschen und bereinigen

Docker-Volumes werden nicht automatisch entfernt, wenn ein Container angehalten wird. So entfernen Sie zugeordnete Volumes beim Stoppen eines Containers:

```
docker rm -v <container id or name>
```

Wenn das Flag `-v` nicht angegeben ist, verbleibt das Volume als 'hängendes Volume' auf der Festplatte. So löschen Sie alle baumelnden Datenträger:

```
docker volume rm $(docker volume ls -qf dangling=true)
```

Der `docker volume ls -qf dangling=true` Filter gibt eine Liste der `docker volume ls -qf dangling=true` Namen zurück, einschließlich nicht gekennzeichneter, die nicht an einen Container angehängt sind.

Alternativ können Sie `xargs` :

```
docker volume ls -f dangling=true -q | xargs --no-run-if-empty docker volume rm
```

Exportieren und Importieren von Docker-Container-Dateisystemen

Es ist möglich, den Dateisysteminhalt eines Docker-Containers in einer Tarball-Archivdatei zu speichern. Dies ist praktisch, wenn Sie Container-Dateisysteme auf andere Hosts verschieben möchten, z. B. wenn ein Datenbankcontainer wichtige Änderungen enthält und die Änderungen an anderer Stelle nicht repliziert werden können. **Beachten Sie,** dass es vorzuziehen ist, einen vollständig neuen Container aus einem aktualisierten Image mithilfe eines `docker run` oder einer `docker-compose.yml` Datei zu erstellen, anstatt das Dateisystem eines Containers zu exportieren und zu verschieben. Ein Teil von Dockers Macht ist die Überprüfbarkeit und Verantwortlichkeit seines deklarativen Stils zum Erstellen von Images und Containern. Durch den `docker export` und den `docker import` wird diese Leistung durch die Verschleierung von Änderungen im Dateisystem eines Containers gegenüber dem ursprünglichen Zustand beeinträchtigt.

```
docker export -o redis.tar redis
```

Der obige Befehl erstellt ein leeres Image und exportiert dann das Dateisystem des `redis` Containers in dieses leere Image. Verwenden Sie zum Importieren aus einem Tarball-Archiv:

```
docker import ./redis.tar redis-imported:3.0.7
```

Mit diesem Befehl wird das `redis-imported:3.0.7` Image erstellt, aus dem Container erstellt werden können. Es ist auch möglich, Änderungen beim Import anzulegen und eine Commit-Nachricht festzulegen:

```
docker import -c="ENV DEBUG true" -m="enable debug mode" ./redis.tar redis-changed
```

Die Dockerfile-Direktiven, die für die Verwendung der Befehlszeilenoption `-c` verfügbar sind, sind `CMD` , `ENTRYPOINT` , `ENV` , `EXPOSE` , `ONBUILD` , `USER` , `VOLUME` , `WORKDIR` .

Container verwalten online lesen: https://riptutorial.com/de/docker/topic/689/container-verwalten

Kapitel 9: Datenvolumen und Datencontainer

Examples

Nur-Daten-Container

Nur-Daten-Container sind veraltet und gelten jetzt als Anti-Pattern!

In früheren Tagen, vor dem Unterbefehls- `volume` Docker, und noch bevor benannte Volumes erstellt werden konnten, löschte Docker Volumes, wenn in keinem Container mehr Verweise darauf vorhanden waren. Nur-Daten-Container sind obsolet, da Docker jetzt die Möglichkeit bietet, benannte Volumes zu erstellen sowie mehr Dienstprogramm über den verschiedenen `docker volume` Unterbefehl zu erstellen. Aus diesem Grund werden Datencontainer jetzt als Anti-Pattern betrachtet.

Viele Ressourcen im Internet aus den letzten Jahren erwähnen die Verwendung eines Musters, das als "Nur-Daten-Container" bezeichnet wird. Hierbei handelt es sich einfach um einen Docker-Container, der nur vorhanden ist, um einen Verweis auf ein Datenvolumen zu erhalten.

Denken Sie daran, dass in diesem Zusammenhang ein "Daten-Volume" ein Docker-Volume ist, das nicht vom Host bereitgestellt wird. Zur Verdeutlichung ist ein "Daten-Volume" ein Volume, das entweder mit der `VOLUME` Dockerfile-Direktive oder mit der Option `-v` in der Befehlszeile in einem `docker run` Befehl erstellt wird, insbesondere mit dem Format `-v /path/on/container` . Ein "Nur-Daten-Container" ist daher ein Container, dessen einziger Zweck darin besteht, ein Daten-Volume `--volumes-from` Flag " `--volumes-from` in einem `docker run` Befehl verwendet wird. Zum Beispiel:

```
docker run -d --name "mysql-data" -v "/var/lib/mysql" alpine /bin/true
```

Wenn der obige Befehl ausgeführt wird, wird ein "Nur-Daten-Container" erstellt. Es ist einfach ein leerer Container, an den ein Datenvolumen angehängt ist. Es war dann möglich, dieses Volume in einem anderen Container zu verwenden:

```
docker run -d --name="mysql" --volumes-from="mysql-data" mysql
```

Der `mysql` Container enthält jetzt das gleiche Volume, das auch in `mysql-data` .

Da Docker jetzt den `volume` Unterbefehl und die benannten Datenträger bereitstellt, ist dieses Muster nun veraltet und wird nicht empfohlen.

Erste Schritte mit dem `volume` Unterbefehl und den benannten Volumes finden Sie unter Erstellen eines benannten Volumes

Datenvolumen erstellen

```
docker run -d --name "mysql-1" -v "/var/lib/mysql" mysql
```

Dieser Befehl erstellt einen neuen Container aus dem `mysql` Image. Es erstellt auch ein neues Daten-Volume, das dann in den Container unter `/var/lib/mysql` . Dieses Volume hilft, dass alle darin enthaltenen Daten über die Lebensdauer des Containers hinaus bestehen bleiben. Das heißt, wenn ein Container entfernt wird, werden auch seine Dateisystemänderungen entfernt. Wenn eine Datenbank Daten im Container gespeichert hat und der Container entfernt wurde, werden auch alle diese Daten entfernt. Volumes bleiben an einem bestimmten Ort auch dann erhalten, wenn der Container entfernt wird.

Es ist möglich, dasselbe Volume in mehreren Containern mit der `--volumes-from` :

```
docker run -d --name="mysql-2" --volumes-from="mysql-1" mysql
```

Dem `mysql-2` Container ist jetzt das Datenvolumen aus `mysql-1` angehängt, das auch den Pfad `/var/lib/mysql` .

Datenvolumen und Datencontainer online lesen:
https://riptutorial.com/de/docker/topic/3224/datenvolumen-und-datencontainer

Kapitel 10: Docker Engine-API

Einführung

Eine API, mit der Sie jeden Aspekt von Docker aus Ihren eigenen Anwendungen heraus steuern, Tools zum Verwalten und Überwachen von auf Docker laufenden Anwendungen erstellen und sogar Anwendungen auf Docker selbst erstellen können.

Examples

Aktivieren Sie den Remote-Zugriff auf die Docker-API unter Linux

Bearbeiten `/etc/init/docker.conf` die `DOCKER_OPTS` `/etc/init/docker.conf` und aktualisieren Sie die Variable `DOCKER_OPTS` wie folgt:

```
DOCKER_OPTS='-H tcp://0.0.0.0:4243 -H unix:///var/run/docker.sock'
```

Starten Sie den Docker-Deamon neu

```
service docker restart
```

Überprüfen Sie, ob die Remote-API funktioniert

```
curl -X GET http://localhost:4243/images/json
```

Aktivieren Sie den Remote-Zugriff auf die Docker-API unter Linux, auf dem systemd ausgeführt wird

Linux, auf dem Systemd ausgeführt wird, wie Ubuntu 16.04, und das Hinzufügen von `-H tcp://0.0.0.0:2375` zu `/etc/default/docker` hat nicht die `-H tcp://0.0.0.0:2375` Wirkung.

Erstellen Sie stattdessen eine Datei mit dem Namen `/etc/systemd/system/docker-tcp.socket` , um das Docker auf einem TCP-Socket an Port 4243 verfügbar zu machen:

```
[Unit]
Description=Docker Socket for the API
[Socket]
ListenStream=4243
Service=docker.service
[Install]
WantedBy=sockets.target
```

Dann aktivieren Sie den neuen Sockel:

```
systemctl enable docker-tcp.socket
systemctl enable docker.socket
```

```
systemctl stop docker
systemctl start docker-tcp.socket
systemctl start docker
```

Überprüfen Sie nun, ob die Remote-API funktioniert:

```
curl -X GET http://localhost:4243/images/json
```

Aktivieren Sie den Remote-Zugriff mit TLS auf Systemd

Kopieren Sie die Einheitendatei des Paketinstallationsprogramms nach / etc, wo die Änderungen bei einem Upgrade nicht überschrieben werden:

```
cp /lib/systemd/system/docker.service /etc/systemd/system/docker.service
```

Aktualisieren Sie /etc/systemd/system/docker.service mit Ihren Optionen in ExecStart:

```
ExecStart=/usr/bin/dockerd -H fd:// -H tcp://0.0.0.0:2376 \
  --tlsverify --tlscacert=/etc/docker/certs/ca.pem \
  --tlskey=/etc/docker/certs/key.pem \
  --tlscert=/etc/docker/certs/cert.pem
```

Beachten Sie, dass `dockerd` der Name des 1.12-Daemons ist, bevor es sich um einen `docker daemon` . Beachten Sie auch, dass 2376 der Standard-TLS-Port von Dockers ist, 2375 der unverschlüsselte Standardport. Auf dieser Seite finden Sie Schritte zum Erstellen Ihrer eigenen TLS-Zertifizierungsstelle, des Zertifikats und des Schlüssels.

Führen Sie nach dem Ändern der systemd-Einheitendateien die folgenden Schritte aus, um die systemd-Konfiguration erneut zu laden:

```
systemctl daemon-reload
```

Führen Sie dann Folgendes aus, um das Andockfenster neu zu starten:

```
systemctl restart docker
```

Es ist keine gute Idee, die TLS-Verschlüsselung zu überspringen, wenn der Docker-Port verfügbar gemacht wird, da jeder, der über Netzwerkzugriff auf diesen Port verfügt, auf dem Host über vollständigen Root-Zugriff verfügt.

Bild ziehen mit Fortschrittsbalken, geschrieben in Go

Hier ein Beispiel für das Ziehen von `docker pull your_image_name` mit der `Go` und `Docker Engine API` und den gleichen Fortschrittsbalken wie beim Ausführen des `docker pull your_image_name` in der `CLI` . Für die Zwecke der Fortschrittsbalken werden einige ANSI-Codes verwendet .

```
package yourpackage
```

```
import (
    "context"
    "encoding/json"
    "fmt"
    "io"
    "strings"

    "github.com/docker/docker/api/types"
    "github.com/docker/docker/client"
)

// Struct representing events returned from image pulling
type pullEvent struct {
    ID             string `json:"id"`
    Status         string `json:"status"`
    Error          string `json:"error,omitempty"`
    Progress       string `json:"progress,omitempty"`
    ProgressDetail struct {
        Current int `json:"current"`
        Total   int `json:"total"`
    } `json:"progressDetail"`
}

// Actual image pulling function
func PullImage(dockerImageName string) bool {
    client, err := client.NewEnvClient()

    if err != nil {
        panic(err)
    }

    resp, err := client.ImagePull(context.Background(), dockerImageName,
types.ImagePullOptions{})

    if err != nil {
        panic(err)
    }

    cursor := Cursor{}
    layers := make([]string, 0)
    oldIndex := len(layers)

    var event *pullEvent
    decoder := json.NewDecoder(resp)

    fmt.Printf("\n")
    cursor.hide()

    for {
        if err := decoder.Decode(&event); err != nil {
            if err == io.EOF {
                break
            }

            panic(err)
        }

        imageID := event.ID

        // Check if the line is one of the final two ones
        if strings.HasPrefix(event.Status, "Digest:") || strings.HasPrefix(event.Status,
```

```
"Status:") {
            fmt.Printf("%s\n", event.Status)
            continue
        }

        // Check if ID has already passed once
        index := 0
        for i, v := range layers {
            if v == imageID {
                index = i + 1
                break
            }
        }

        // Move the cursor
        if index > 0 {
            diff := index - oldIndex

            if diff > 1 {
                down := diff - 1
                cursor.moveDown(down)
            } else if diff < 1 {
                up := diff*(-1) + 1
                cursor.moveUp(up)
            }

            oldIndex = index
        } else {
            layers = append(layers, event.ID)
            diff := len(layers) - oldIndex

            if diff > 1 {
                cursor.moveDown(diff) // Return to the last row
            }

            oldIndex = len(layers)
        }

        cursor.clearLine()

        if event.Status == "Pull complete" {
            fmt.Printf("%s: %s\n", event.ID, event.Status)
        } else {
            fmt.Printf("%s: %s %s\n", event.ID, event.Status, event.Progress)
        }

    }

    cursor.show()

    if strings.Contains(event.Status, fmt.Sprintf("Downloaded newer image for %s",
dockerImageName)) {
        return true
    }

    return false
}
```

Zur besseren Lesbarkeit werden Cursoraktionen mit den ANSI-Codes in eine separate Struktur verschoben, die wie folgt aussieht:

```
package yourpackage

import "fmt"

// Cursor structure that implements some methods
// for manipulating command line's cursor
type Cursor struct{}

func (cursor *Cursor) hide() {
    fmt.Printf("\033[?25l")
}

func (cursor *Cursor) show() {
    fmt.Printf("\033[?25h")
}

func (cursor *Cursor) moveUp(rows int) {
    fmt.Printf("\033[%dF", rows)
}

func (cursor *Cursor) moveDown(rows int) {
    fmt.Printf("\033[%dE", rows)
}

func (cursor *Cursor) clearLine() {
    fmt.Printf("\033[2K")
}
```

Danach können Sie in Ihrem `PullImage` die `PullImage` Funktion `PullImage` , indem Sie den `PullImage` übergeben, den Sie ziehen möchten. Bevor Sie es aufrufen können, müssen Sie natürlich in der Docker-Registry angemeldet sein, wo sich das Image befindet.

Eine cURL-Anfrage mit der Übergabe einer komplexen Struktur erstellen

Wenn Sie `cURL` für einige Abfragen an die `Docker API` , kann es schwierig sein, einige komplexe Strukturen zu übergeben. Nehmen wir an, eine Liste von Bildern zu erhalten, ermöglicht die Verwendung von Filtern als Abfrageparameter, die eine `JSON` Darstellung von `map[string][]string` (Informationen zu den Maps in `Go` Sie hier).
So erreichen Sie das:

```
curl --unix-socket /var/run/docker.sock \
    -XGET "http:/v1.29/images/json" \
    -G \
    --data-urlencode 'filters={"reference":{"yourpreciousregistry.com/path/to/image": true},
"dangling":{"true": true}}'
```

Mit dem Flag `-G` wird angegeben, dass die Daten im Parameter `--data-urlencode` in einer `HTTP GET` Anforderung anstelle der `POST` Anforderung verwendet werden, die andernfalls verwendet würde. Die Daten werden mit einem `?` An die URL angehängt `?` Separator.

Docker Engine-API online lesen: https://riptutorial.com/de/docker/topic/3935/docker-engine-api

Kapitel 11: Docker erfasst alle laufenden Container

Examples

Docker erfasst alle laufenden Container

```
sudo docker stats $(sudo docker inspect -f "{{ .Name }}" $(sudo docker ps -q))
```

Zeigt die CPU-Auslastung aller laufenden Container an.

Docker erfasst alle laufenden Container online lesen:
https://riptutorial.com/de/docker/topic/5863/docker-erfasst-alle-laufenden-container

Kapitel 12: Docker in Docker

Examples

Jenkins CI Container mit Docker

In diesem Kapitel wird beschrieben, wie Sie einen Docker-Container mit Jenkins einrichten, der Docker-Befehle an die Docker-Installation (den Docker-Daemon) des Hosts senden kann. Docker effektiv in Docker verwenden. Um dies zu erreichen, müssen wir ein benutzerdefiniertes Docker Image erstellen, das auf einer beliebigen Version des offiziellen Jenkins Docker Image basiert. Die Dockerfile (Die Anleitung zum Erstellen des Image) sieht folgendermaßen aus:

```
FROM jenkins

USER root

RUN cd /usr/local/bin && \
curl https://master.dockerproject.org/linux/amd64/docker > docker  && \
chmod +x docker  && \
groupadd -g 999 docker && \
usermod -a -G docker jenkins

USER Jenkins
```

Diese Docker-Datei erstellt ein Image, das die Binärdateien des Docker-Clients enthält, über die dieser Client mit einem Docker-Daemon kommuniziert. In diesem Fall der Docker Daemon des Hosts. Die `RUN` Anweisung in dieser Datei erstellt auch eine UNIX-Benutzergruppe mit der UID 999 und fügt den Benutzer Jenkins hinzu. Warum genau dies notwendig ist, wird im weiteren Kapitel beschrieben. Mit diesem Image können wir einen Jenkins-Server ausführen, der Docker-Befehle verwenden kann. Wenn Sie jedoch nur dieses Image ausführen, kann der Docker-Client, den wir im Image installiert haben, nicht mit dem Docker-Daemon des Host kommunizieren. Diese beiden Komponenten kommunizieren über einen UNIX-Socket `/var/run/docker.sock` . Unter Unix ist dies eine Datei wie alles andere, sodass wir sie leicht in den Jenkins-Container einbinden können. Dies erfolgt mit dem Befehl `docker run -v /var/run/docker.sock:/var/run/docker.sock --name jenkins MY_CUSTOM_IMAGE_NAME` . Diese gemountete Datei gehört `docker:root` diesem Grund erstellt die Dockerfile diese Gruppe mit einer bekannten UID und fügt den Jenkins-Benutzer hinzu. Nun ist der Jenkins Container wirklich in der Lage, Docker auszuführen und zu verwenden. In der Produktion sollte der Befehl run `-v jenkins_home:/var/jenkins_home` um das Jenkins_home-Verzeichnis zu sichern, und natürlich ein Port-Mapping, um über ein Netzwerk auf den Server zuzugreifen.

Docker in Docker online lesen: https://riptutorial.com/de/docker/topic/8012/docker-in-docker

Kapitel 13: Docker inspizieren verschiedene Felder für Schlüssel: Wert und Elemente der Liste

Examples

verschiedene Docker inspizieren Beispiele

Ich finde, dass die Beispiele in der `docker inspect` Dokumentation magisch wirken, aber nicht viel erklären.

Docker Inspect ist wichtig, da dies die saubere Methode zum Extrahieren von Informationen aus einem laufenden Container `docker inspect -f ... container_id`

(oder alle laufenden Container)

```
docker inspect -f ... $(docker ps -q)
```

etwas Unzuverlässiges vermeiden

```
docker command | grep or awk | tr or cut
```

Wenn Sie eine `docker inspect` starten, können Sie die Werte einfach von der "obersten Ebene" mit einer einfachen Syntax wie für einen Container abrufen, auf dem htop ausgeführt wird (von https://hub.docker.com/r/jess/htop/). mit einer pid ae1

```
docker inspect -f '{{.Created}}' ae1
```

kann zeigen

```
2016-07-14T17:44:14.159094456Z
```

oder

```
docker inspect -f '{{.Path}}' ae1
```

kann zeigen

```
htop
```

Wenn ich nun einen Teil meines `docker inspect` ich

Aha

`"State": { "Status": "running", "Running": true, "Paused": false, "Restarting": false, "OOMKilled": false, "Dead": false, "Pid": 4525, "ExitCode": 0, "Error": "", "StartedAt": "2016-07-14T17:44:14.406286293Z", "FinishedAt": "0001-01-01T00:00:00Z"` So bekomme ich ein Wörterbuch, wie es hat `{ ...}` und viele Schlüssel: Werte

Also der Befehl

```
docker inspect -f '{{.State}}' ae1
```

gibt eine Liste zurück, wie z

```
{running true false false false false 4525 0 2016-07-14T17:44:14.406286293Z 0001-01-
01T00:00:00Z}
```

Ich kann den Wert von State.Pid leicht erhalten

```
docker inspect -f '{{ .State.Pid }}' ae1
```

Ich bekomme

```
4525
```

Manchmal gibt Docker Inspect eine Liste aus, die mit `[` beginnt und mit `]` endet.

ein anderes Beispiel mit einem anderen Container

```
docker inspect -f '{{ .Config.Env }}' 7a7
```

gibt

```
[DISPLAY=:0 PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin LANG=fr_FR.UTF-8
LANGUAGE=fr_FR:en LC_ALL=fr_FR.UTF-8 DEBIAN_FRONTEND=noninteractive HOME=/home/gg WINEARCH=win32
WINEPREFIX=/home/gg/.wine_captvty]
```

Um das erste Element der Liste zu erhalten, fügen wir den Index vor dem erforderlichen Feld und 0 (als erstes Element) danach hinzu

```
docker inspect -f '{{ index ( .Config.Env) 0 }}' 7a7
```

gibt

```
DISPLAY=:0
```

Wir erhalten das nächste Element mit 1 anstelle von 0 mit derselben Syntax

```
docker inspect -f '{{ index ( .Config.Env) 1 }}' 7a7
```

gibt

```
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

Wir können die Anzahl der Elemente dieser Liste ermitteln

```
docker inspect -f '{{ len .Config.Env }}' 7a7
```

gibt

```
9
```

und wir können das letzte Element der Liste erhalten, die Syntax ist nicht einfach

```
docker inspect -f "{{ index .Config.Cmd $[$(docker inspect -format '{{ len .Config.Cmd }}'
$CID)-1]}}" 7a7
```

Docker inspizieren verschiedene Felder für Schlüssel: Wert und Elemente der Liste online lesen: https://riptutorial.com/de/docker/topic/6470/docker-inspizieren-verschiedene-felder-fur-schlussel--wert-und-elemente-der-liste

Kapitel 14: Docker-Datenvolumen

Einführung

Docker-Datenmengen bieten eine Möglichkeit, Daten unabhängig vom Lebenszyklus eines Containers zu speichern. Volumes bieten eine Reihe hilfreicher Funktionen wie:

Mounten eines Hostverzeichnisses innerhalb des Containers, Freigeben von Daten zwischen Containern mithilfe des Dateisystems und Beibehalten der Daten, wenn ein Container gelöscht wird

Syntax

- Docker-Volume [OPTIONEN] [BEFEHL]

Examples

Mounten eines Verzeichnisses vom lokalen Host in einen Container

Es ist möglich, ein Host-Verzeichnis mit der Befehlszeilenoption `-v` oder `--volume` an einen bestimmten Pfad in Ihrem Container `--volume` . Im folgenden Beispiel wird `/etc` auf dem Host in `/mnt/etc` im Container `/mnt/etc` :

```
(on linux) docker run -v "/etc:/mnt/etc" alpine cat /mnt/etc/passwd
(on windows)  docker run -v "/c/etc:/mnt/etc" alpine cat /mnt/etc/passwd
```

Der Standardzugriff auf den Datenträger im Container lautet Lesen und Schreiben. Verwenden Sie das Suffix `:ro` um ein Volume schreibgeschützt in einem Container `:ro` .

```
docker run -v "/etc:/mnt/etc:ro" alpine touch /mnt/etc/passwd
```

Ein benanntes Volume erstellen

```
docker volume create --name="myAwesomeApp"
```

Die Verwendung eines benannten Datenträgers macht die Verwaltung von Datenträgern deutlich lesbarer. Es ist möglich, ein benanntes Volume mit dem oben angegebenen Befehl zu erstellen, aber es ist auch möglich, ein benanntes Volume innerhalb eines `docker run` mithilfe der Befehlszeilenoption `-v` oder `--volume` zu erstellen:

```
docker run -d --name="myApp-1" -v="myAwesomeApp:/data/app" myApp:1.5.3
```

Das Erstellen eines benannten Volumes in diesem Formular ähnelt dem Mounten einer Hostdatei / eines Hostverzeichnisses als Volume, mit der Ausnahme, dass anstelle eines gültigen Pfads der

Volume-Name angegeben wird. Nach der Erstellung können benannte Volumes für andere Container freigegeben werden:

```
docker run -d --name="myApp-2" --volumes-from "myApp-1" myApp:1.5.3
```

Nachdem Sie den obigen Befehl ausgeführt haben, wurde ein neuer Container mit dem Namen `myApp-2` aus dem Image `myApp:1.5.3` , der das mit `myAwesomeApp` benannte Volume für `myApp-1` . Das genannte Volume mit der Bezeichnung `myAwesomeApp` wird unter `/data/app` im `myApp-2` Container `myApp-2` , genauso wie es unter `/data/app` im `myApp-1` Container `myApp-1` .

Docker-Datenvolumen online lesen: https://riptutorial.com/de/docker/topic/1318/docker-datenvolumen

Kapitel 15: Docker-Ereignisse

Examples

Starten Sie einen Container und lassen Sie sich über verwandte Ereignisse benachrichtigen

Die Dokumentation für `docker events` enthält Details. Beim Debuggen kann es jedoch nützlich sein, einen Container zu starten und sofort über alle zugehörigen Ereignisse benachrichtigt zu werden:

```
docker run... & docker events --filter 'container=$(docker ps -lq)'
```

Im `docker ps -lq` steht das `l` für `last` und das `q` für `quiet` . Dadurch wird die `id` des zuletzt gestarteten Containers entfernt und sofort eine Benachrichtigung erstellt, wenn der Container stirbt oder ein anderes Ereignis eintritt.

Docker-Ereignisse online lesen: https://riptutorial.com/de/docker/topic/6200/docker-ereignisse

Kapitel 16: Dockerfile-Inhalte bestellen

Bemerkungen

1. Basisbilddeklaration (`FROM`)
2. Metadaten (zB `MAINTAINER` , `LABEL`)
3. Installation von Systemabhängigkeiten (zB `apt-get install` , `apk add`)
4. App-Abhängigkeitsdatei `bower.json` (z. B. `bower.json` , `package.json` , `build.gradle` , `requirements.txt`)
5. App-Abhängigkeiten `npm install` (zB `npm install` , `pip install`)
6. Kopieren der gesamten Codebasis
7. Standard-Laufzeitkonfigurationen einrichten (zB `CMD` , `ENTRYPOINT` , `ENV` , `EXPOSE`)

Diese Reihenfolge wird zur Optimierung der Buildzeit mithilfe des integrierten Cache-Mechanismus von Docker festgelegt.

Faustregel:

> Teile, die sich häufig ändern (z. B. Codebase), sollten sich in der Nähe der Unterseite der Dockerfile befinden und umgekehrt. Teile, die sich selten ändern (z. B. Abhängigkeiten), sollten oben platziert werden.

Examples

Einfache Dockerfile

```
# Base image
FROM python:2.7-alpine

# Metadata
MAINTAINER John Doe <johndoe@example.com>

# System-level dependencies
RUN apk add --update \
    ca-certificates \
    && update-ca-certificates \
    && rm -rf /var/cache/apk/*

# App dependencies
COPY requirements.txt /requirements.txt
RUN pip install -r /requirements.txt

# App codebase
WORKDIR /app
COPY . ./

# Configs
ENV DEBUG true
EXPOSE 5000
CMD ["python", "app.py"]
```

MAINTAINER wird in Docker 1.13 nicht mehr empfohlen und sollte mit LABEL ersetzt werden. (Quelle)

Beispiel: LABEL Maintainer = "John Doe johndoe@example.com"

Dockerfile-Inhalte bestellen online lesen: https://riptutorial.com/de/docker/topic/6448/dockerfile-inhalte-bestellen

Kapitel 17: Dockerfiles

Einführung

Dockerfiles sind Dateien, mit denen Docker-Images programmatisch erstellt werden. Sie ermöglichen es Ihnen, schnell und reproduzierbar ein Docker-Image zu erstellen und sind daher für die Zusammenarbeit hilfreich. Docker-Dateien enthalten Anweisungen zum Erstellen eines Docker-Images. Jede Anweisung ist in einer Zeile geschrieben und in der Form `<INSTRUCTION><argument(s)>` . Docker-Dateien werden verwendet, um Docker-Images mit dem `docker build` Befehl zu `docker build` .

Bemerkungen

Dockerfiles haben die Form:

```
# This is a comment
INSTRUCTION arguments
```

- Kommentare beginnen mit einem `#`
- Anweisungen sind nur in Großbuchstaben
- Die erste Anweisung einer Docker-Datei muss `FROM` , um das Basisabbild anzugeben

---

Beim Erstellen einer Docker-Datei sendet der Docker-Client einen "Build-Kontext" an den Docker-Daemon. Der Build-Kontext enthält alle Dateien und Ordner im selben Verzeichnis wie die Docker-Datei. `COPY` und `ADD` Vorgänge können nur Dateien aus diesem Kontext verwenden.

Einige Docker-Dateien beginnen mit:

```
# escape=`
```

Hiermit wird der Docker-Parser angewiesen, `` ` `` als Escape-Zeichen anstelle von `\` . Dies ist vor allem für Windows Docker-Dateien nützlich.

Examples

HelloWorld Dockerfile

Eine minimale Docker-Datei sieht folgendermaßen aus:

```
FROM alpine
CMD ["echo", "Hello StackOverflow!"]
```

Dadurch wird Docker angewiesen, ein Abbild auf der Grundlage von Alpine (`FROM`), einer

minimalen Verteilung für Container, zu erstellen und beim Ausführen des resultierenden Abbilds einen bestimmten Befehl (`CMD`) auszuführen.

Bauen Sie es auf und führen Sie es aus:

```
docker build -t hello .
docker run --rm hello
```

Dies wird ausgegeben:

```
Hello StackOverflow!
```

Dateien kopieren

Verwenden Sie zum Kopieren von Dateien aus dem Build-Kontext in einem Docker-Image die `COPY` Anweisung:

```
COPY localfile.txt containerfile.txt
```

Wenn der Dateiname Leerzeichen enthält, verwenden Sie die alternative Syntax:

```
COPY ["local file", "container file"]
```

Der Befehl `COPY` unterstützt Platzhalter. Es kann beispielsweise verwendet werden, um alle Bilder in das Verzeichnis `images/` zu kopieren:

```
COPY *.jpg images/
```

Hinweis: In diesem Beispiel sind möglicherweise keine `images/` vorhanden. In diesem Fall erstellt Docker es automatisch.

Einen Hafen freigeben

Um deklarierte Ports aus einer Dockerfile zu deklarieren, verwenden Sie die Anweisung `EXPOSE` :

```
EXPOSE 8080 8082
```

Die Einstellung der freigelegten Ports kann von der Docker-Befehlszeile aus überschrieben werden. Es ist jedoch empfehlenswert, sie explizit in der Docker-Datei festzulegen, da dies die Funktionsweise einer Anwendung erleichtert.

Dockerfiles beste Praktiken

Gemeinsame Operationen gruppieren

Docker erstellt Bilder als eine Sammlung von Ebenen. Jede Ebene kann nur Daten hinzufügen, selbst wenn diese Daten anzeigen, dass eine Datei gelöscht wurde. Jede Anweisung erstellt eine

neue Ebene. Zum Beispiel:

```
RUN apt-get -qq update
RUN apt-get -qq install some-package
```

Hat ein paar Nachteile:

- Es werden zwei Ebenen erstellt, wodurch ein größeres Bild erzeugt wird.
- Die alleinige Verwendung von `apt-get update` in einer `RUN` Anweisung führt zu Zwischenspeicherungsproblemen und anschließend können `apt-get install` **fehlschlagen** . Angenommen, Sie ändern `apt-get install` durch Hinzufügen zusätzlicher Pakete. Dann interpretiert das Andockfenster die ursprünglichen und geänderten Anweisungen als identisch und verwendet den Cache von den vorherigen Schritten. Daher wird der Befehl `apt-get update` **nicht** ausgeführt, da seine zwischengespeicherte Version während des Builds verwendet wird.

Verwenden Sie stattdessen:

```
RUN apt-get -qq update && \
    apt-get -qq install some-package
```

da dies nur eine Schicht erzeugt.

Erwähnen Sie den Betreuer

Dies ist normalerweise die zweite Zeile der Dockerfile. Es sagt, wer verantwortlich ist und helfen kann.

```
LABEL maintainer John Doe <john.doe@example.com>
```

Wenn Sie es überspringen, wird Ihr Bild nicht beschädigt. Aber es hilft auch nicht Ihren Benutzern.

Sei präzise

Halten Sie Ihre Dockerfile kurz. Wenn ein komplexes Setup erforderlich ist, sollten Sie ein dediziertes Skript verwenden oder Basisabbilder einrichten.

USER-Anweisung

```
USER daemon
```

Der `USER` - Befehl setzt die Benutzername oder UID zu verwenden , wenn das Bild ausgeführt wird und für jeden `RUN` , `CMD` und `ENTRYPOINT` Anweisungen , die es in dem folgen `Dockerfile` .

WORKDIR-Anweisung

```
WORKDIR /path/to/workdir
```

Die `WORKDIR` Anweisung legt das Arbeitsverzeichnis für alle Anweisungen `RUN` , `CMD` , `ENTRYPOINT` , `COPY` und `ADD` , die in der Docker-Datei folgen. Wenn das `WORKDIR` nicht existiert, wird es erstellt, auch wenn es nicht in einer nachfolgenden `Dockerfile` Anweisung verwendet wird.

Es kann mehrmals in einer `Dockerfile` . Wenn ein relativer Pfad angegeben wird, ist dieser relativ zum Pfad der vorherigen `WORKDIR` Anweisung. Zum Beispiel:

```
WORKDIR /a
WORKDIR b
WORKDIR c
RUN pwd
```

Die Ausgabe des letzten `pwd` Befehls in dieser `Dockerfile` wäre `/a/b/c` .

Die `WORKDIR` Anweisung kann Umgebungsvariablen auflösen, die zuvor mit `ENV` . Sie können nur Umgebungsvariablen verwenden, die explizit in der `Dockerfile` . Zum Beispiel:

```
ENV DIRPATH /path
WORKDIR $DIRPATH/$DIRNAME
RUN pwd
```

Die Ausgabe des letzten `pwd` Befehls in dieser Docker-Datei wäre `/path/$DIRNAME`

VOLUME-Anweisung

```
VOLUME ["/data"]
```

Mit der Anweisung `VOLUME` wird ein Mountpunkt mit dem angegebenen Namen erstellt und als für extern gemountete Volumes auf dem nativen Host oder anderen Containern markiert. Der Wert kann ein JSON-Array, `VOLUME ["/var/log/"]` oder eine einfache Zeichenfolge mit mehreren Argumenten sein, beispielsweise `VOLUME /var/log` oder `VOLUME /var/log /var/db` . Weitere Informationen / Beispiele und Installationsanweisungen über den Docker-Client finden Sie in der Dokumentation zum Freigeben von Verzeichnissen über Volumes.

Der `docker run` Befehl initialisiert den neu erstellten Datenträger mit allen Daten, die an der angegebenen Position im Basisabbild vorhanden sind. Betrachten Sie beispielsweise das folgende Dockerfile-Snippet:

```
FROM ubuntu
RUN mkdir /myvol
RUN echo "hello world" > /myvol/greeting
VOLUME /myvol
```

Diese Docker-Datei führt zu einem Bild, das das Andockfenster veranlasst, einen neuen Mount-Punkt unter / myvol zu erstellen und die Begrüßungsdatei in das neu erstellte Volume zu kopieren.

Hinweis: Wenn Build-Schritte die Daten innerhalb des Volumes ändern, nachdem sie deklariert wurden, werden diese Änderungen verworfen.

Anmerkung: Die Liste wird als JSON-Array analysiert. Das bedeutet, dass Sie doppelte Anführungszeichen (“) für Wörter verwenden müssen, nicht einfache Anführungszeichen (').

COPY-Anweisung

`COPY` hat zwei Formen:

```
COPY <src>... <dest>
COPY ["<src>",... "<dest>"] (this form is required for paths containing whitespace)
```

Die `COPY` Anweisung kopiert neue Dateien oder Verzeichnisse aus `<src>` und fügt sie dem Dateisystem des Containers unter dem Pfad `<dest>` .

Es können mehrere `<src>` -Ressourcen angegeben werden, sie müssen jedoch relativ zu dem Quellverzeichnis sein, das erstellt wird (dem Kontext des Builds).

Jedes `<src>` kann Platzhalter enthalten, und der Abgleich wird mit den `filepath.Match` Regeln von Go `filepath.Match` . Zum Beispiel:

```
COPY hom* /mydir/        # adds all files starting with "hom"
COPY hom?.txt /mydir/    # ? is replaced with any single character, e.g., "home.txt"
```

Der `<dest>` ist ein absoluter Pfad oder ein Pfad relativ zu `WORKDIR` , in den die Quelle in den `WORKDIR` kopiert wird.

```
COPY test relativeDir/   # adds "test" to `WORKDIR`/relativeDir/
COPY test /absoluteDir/  # adds "test" to /absoluteDir/
```

Alle neuen Dateien und Verzeichnisse werden mit einer UID und einer GID von 0 erstellt.

Hinweis: Wenn Sie mit stdin (`docker build - < somefile`) `docker build - < somefile` , gibt es keinen Build-Kontext, sodass `COPY` nicht verwendet werden kann.

`COPY` beachtet die folgenden Regeln:

- Der Pfad `<src>` muss sich im Kontext des Builds befinden. Sie können nicht `COPY` ../something / etwas, weil der erste Schritt eines Docker Build ist das Kontext Verzeichnis (und Unterverzeichnisse) mit der Docker - Daemon zu senden.
- Wenn `<src>` ein Verzeichnis ist, wird der gesamte Inhalt des Verzeichnisses einschließlich der Metadaten des Dateisystems kopiert. Hinweis: Das Verzeichnis selbst wird nicht kopiert, sondern nur der Inhalt.
- Wenn `<src>` eine andere Art von Datei ist, wird sie zusammen mit ihren Metadaten einzeln kopiert. Wenn in diesem Fall `<dest>` mit einem nachgestellten Schrägstrich / endet, wird dies als Verzeichnis betrachtet und der Inhalt von `<src>` wird in `<dest>/base(<src>)` .
- Wenn mehrere `<src>` -Ressourcen angegeben werden, entweder direkt oder aufgrund der Verwendung eines Platzhalters, muss `<dest>` ein Verzeichnis sein und mit einem

Schrägstrich `/` enden.

- Wenn `<dest>` nicht mit einem nachgestellten Schrägstrich endet, wird dies als reguläre Datei betrachtet und der Inhalt von `<src>` wird an `<dest>` .
- Wenn `<dest>` nicht vorhanden ist, wird es zusammen mit allen fehlenden Verzeichnissen in seinem Pfad erstellt.

Die ENV- und ARG-Anweisung

ENV

```
ENV <key> <value>
ENV <key>=<value> ...
```

Die Anweisung `ENV` setzt die Umgebungsvariable `<key>` auf den Wert. Dieser Wert befindet sich in der Umgebung aller "nachkommenden" Dockerfile-Befehle und kann auch in vielen Inline-Versionen ersetzt werden.

Die `ENV` Anweisung hat zwei Formen. Die erste Form `ENV <key> <value>` setzt eine einzelne Variable auf einen Wert. Die gesamte Zeichenfolge nach dem ersten Leerzeichen wird als `<value>` behandelt, einschließlich Zeichen wie Leerzeichen und Anführungszeichen.

Die zweite Form `ENV <key>=<value> ...` ermöglicht das gleichzeitige Setzen mehrerer Variablen. Beachten Sie, dass das zweite Formular das Gleichheitszeichen (=) in der Syntax verwendet, das erste Formular jedoch nicht. Wie bei der Befehlszeilenanalyse können Anführungszeichen und umgekehrte Schrägstriche verwendet werden, um Leerzeichen in Werte einzuschließen.

Zum Beispiel:

```
ENV myName="John Doe" myDog=Rex\ The\ Dog \
    myCat=fluffy
```

und

```
ENV myName John Doe
ENV myDog Rex The Dog
ENV myCat fluffy
```

führt im endgültigen Container zu den gleichen Nettoergebnissen, aber die erste Form wird bevorzugt, da sie eine einzelne Cache-Ebene erzeugt.

Die mit `ENV` festgelegten Umgebungsvariablen bleiben erhalten, wenn ein Container vom resultierenden Image aus ausgeführt wird. Sie können die Werte mithilfe der Docker- `docker run --env <key>=<value>` und mithilfe des `docker run --env <key>=<value>` .

ARG

Wenn Sie die Einstellung nicht beibehalten möchten, verwenden Sie stattdessen `ARG` . `ARG` setzt Umgebungen nur während des Builds. Zum Beispiel Einstellung

```
ENV DEBIAN_FRONTEND noninteractive
```

`apt-get` Benutzer können auf einem Debian-basierten Image verwirrt werden, wenn sie den Container in einem interaktiven Kontext über das `docker exec -it the-container bash` eingeben.

Verwenden Sie stattdessen:

```
ARG DEBIAN_FRONTEND noninteractive
```

Sie können alternativ auch einen Wert für einen einzelnen Befehl festlegen, indem Sie Folgendes verwenden:

```
RUN <key>=<value> <command>
```

EXPOSE Anweisung

```
EXPOSE <port> [<port>...]
```

Die Anweisung `EXPOSE` informiert Docker darüber, dass der Container zur Laufzeit die angegebenen Netzwerkports abhört. `EXPOSE` macht die Ports des Containers NICHT für den Host zugänglich. Dazu müssen Sie entweder das `-p` Flag zum Veröffentlichen eines `-P` oder das `-P` Flag zum Veröffentlichen aller freigelegten Ports verwenden. Diese Flags werden im `docker run [OPTIONS] IMAGE [COMMAND][ARG...]` , um den Port für den Host `docker run [OPTIONS] IMAGE [COMMAND][ARG...]` . Sie können eine Portnummer freigeben und extern unter einer anderen Nummer veröffentlichen.

```
docker run -p 2500:80 <image name>
```

Dieser Befehl erstellt einen Container mit dem Namen <image> und bindet den Port 80 des Containers an den Port 2500 der Hostmaschine.

Informationen zum Einrichten der Portumleitung auf dem Hostsystem finden Sie unter Verwenden des Kennzeichens `-P` . Die Docker-Netzwerkfunktion unterstützt das Erstellen von Netzwerken, ohne dass Ports innerhalb des Netzwerks verfügbar gemacht werden müssen (detaillierte Informationen finden Sie in der Übersicht dieser Funktion).

LABEL-Anweisung

```
LABEL <key>=<value> <key>=<value> <key>=<value> ...
```

Die `LABEL` Anweisung fügt einem Bild Metadaten hinzu. Ein `LABEL` ist ein Schlüssel-Wert-Paar. Verwenden Sie zum `LABEL` von Leerzeichen in einen `LABEL` Wert Anführungszeichen und `LABEL` Schrägstriche wie bei der Befehlszeilenanalyse. Einige Anwendungsbeispiele:

```
LABEL "com.example.vendor"="ACME Incorporated"
LABEL com.example.label-with-value="foo"
LABEL version="1.0"
LABEL description="This text illustrates \
that label-values can span multiple lines."
```

Ein Bild kann mehr als ein Label haben. Um mehrere Labels anzugeben, empfiehlt Docker, die Labels möglichst in einer einzigen `LABEL` Anweisung zu kombinieren. Jede `LABEL` Anweisung erzeugt eine neue Ebene, die zu einem ineffizienten Bild führen kann, wenn Sie viele Beschriftungen verwenden. Dieses Beispiel ergibt eine einzelne Bildebene.

```
LABEL multi.label1="value1" multi.label2="value2" other="value3"
```

Das oben Gesagte kann auch geschrieben werden als:

```
LABEL multi.label1="value1" \
      multi.label2="value2" \
      other="value3"
```

Labels sind additiv, einschließlich `LABEL` in `FROM` Bildern. Wenn Docker auf ein bereits vorhandenes Label / einen Schlüssel stößt, überschreibt der neue Wert alle vorherigen Labels mit identischen Schlüsseln.

Verwenden Sie den Andockbefehl, um die Beschriftungen eines Bildes anzuzeigen.

```
"Labels": {
    "com.example.vendor": "ACME Incorporated"
    "com.example.label-with-value": "foo",
    "version": "1.0",
    "description": "This text illustrates that label-values can span multiple lines.",
    "multi.label1": "value1",
    "multi.label2": "value2",
    "other": "value3"
},
```

CMD-Anweisung

Die `CMD` Anweisung hat drei Formen:

```
CMD ["executable","param1","param2"] (exec form, this is the preferred form)
CMD ["param1","param2"] (as default parameters to ENTRYPOINT)
CMD command param1 param2 (shell form)
```

Es kann nur eine `CMD` Anweisung in einer `Dockerfile` . Wenn Sie mehr als eine `CMD` `CMD` wird nur die letzte `CMD` wirksam.

Der Hauptzweck einer `CMD` besteht darin, Standardwerte für einen ausgeführten Container bereitzustellen. Diese Standardwerte können eine ausführbare Datei enthalten oder sie können die ausführbare Datei weglassen. In diesem Fall müssen Sie auch eine `ENTRYPOINT` Anweisung angeben.

Anmerkung: Wenn `CMD` verwendet wird, um Standardargumente für die Anweisung `ENTRYPOINT` , sollten sowohl die Anweisungen `CMD` als auch `ENTRYPOINT` mit dem JSON-Array-Format angegeben werden.

Hinweis: Das exec-Formular wird als JSON-Array analysiert. Das bedeutet, dass Sie doppelte Anführungszeichen (“) für Wörter verwenden müssen, nicht einfache Anführungszeichen (').

Hinweis: Im Gegensatz zum Shell-Formular ruft das Exec-Formular keine Befehls-Shell auf. Dies bedeutet, dass eine normale Shell-Verarbeitung nicht stattfindet. Beispielsweise führt `CMD [ "echo", "$HOME" ]` keine Variablensubstitution bei `$HOME` . Wenn Sie eine Shell-Verarbeitung wünschen, verwenden Sie entweder die Shell-Form oder führen Sie eine Shell direkt aus, zum Beispiel: `CMD [ "sh", "-c", "echo $HOME" ]` .

Bei Verwendung in Shell- oder Exec-Formaten legt der `CMD` Befehl fest, dass der Befehl ausgeführt wird, wenn das Image ausgeführt wird.

Wenn Sie die Shell-Form des `CMD` , wird der Befehl in `/bin/sh -c` :

```
FROM ubuntu
CMD echo "This is a test." | wc -
```

Wenn Sie Ihren Befehl ohne Shell ausführen möchten, müssen Sie den Befehl als JSON-Array ausdrücken und der ausführbaren Datei den vollständigen Pfad angeben. Diese Feldform ist das bevorzugte Format von `CMD` . Alle zusätzlichen Parameter müssen einzeln als Strings im Array ausgedrückt werden:

```
FROM ubuntu
CMD ["/usr/bin/wc","--help"]
```

Wenn Sie möchten, dass Ihr Container jedes Mal dieselbe ausführbare Datei `ENTRYPOINT` sollten Sie `ENTRYPOINT` in Kombination mit `CMD` in Betracht `ENTRYPOINT` . Siehe `ENTRYPOINT` .

Wenn der Benutzer Argumente für die Ausführung des Dockers angibt, überschreibt er den in `CMD` angegebenen Standard.

Hinweis: Verwechseln Sie `RUN` mit `CMD` . `RUN` tatsächlich einen Befehl zur Image-Erstellungszeit aus und überträgt das Ergebnis. `CMD` führt zum Erstellungszeitpunkt nichts aus, gibt jedoch den beabsichtigten Befehl für das Image an.

MAINTAINER-Anweisung

```
MAINTAINER <name>
```

Mit der Anweisung `MAINTAINER` können Sie das Autorenfeld der generierten Bilder festlegen.

DIE MAINTAINER-RICHTLINIE NICHT VERWENDEN

Gemäß der offiziellen Docker-Dokumentation ist die `MAINTAINER` Anweisung veraltet. Stattdessen sollte man die `LABEL` Anweisung verwenden, um den Autor der erzeugten Bilder zu definieren. Der

`LABEL` - Befehl ist flexibler, ermöglicht Metadaten Einstellung und können leicht mit dem Befehl betrachtet werden `docker inspect` .

```
LABEL maintainer="someone@something.com"
```

FROM Anweisung

```
FROM <image>
```

Oder

```
FROM <image>:<tag>
```

Oder

```
FROM <image>@<digest>
```

Die `FROM` Anweisung legt das Basisbild für nachfolgende Anweisungen fest. Daher muss eine gültige Docker-Datei `FROM` als erste Anweisung enthalten. Das Bild kann ein beliebiges gültiges Bild sein. Es ist besonders einfach, ein Bild aus den öffentlichen Repositorys zu ziehen.

`FROM` muss die erste Anweisung ohne Kommentar in der Dockerfile sein.

`FROM` kann in einer einzigen Docker-Datei mehrmals angezeigt werden, um mehrere Bilder zu erstellen. Notieren Sie sich vor jedem neuen `FROM` Befehl einfach die letzte vom Commit ausgegebene Bild-ID.

Die Tag- oder Digest-Werte sind optional. Wenn Sie eine der beiden Optionen weglassen, geht der Builder standardmäßig von einer neuen Version aus. Der Builder gibt einen Fehler zurück, wenn er nicht mit dem Tag-Wert übereinstimmen kann.

RUN-Anweisung

`RUN` hat 2 Formen:

```
RUN <command> (shell form, the command is run in a shell, which by default is /bin/sh -c on
Linux or cmd /S /C on Windows)
RUN ["executable", "param1", "param2"] (exec form)
```

Die `RUN` Anweisung führt alle Befehle in einer neuen Ebene über dem aktuellen Bild aus und schreibt die Ergebnisse fest. Das resultierende festgeschriebene Image wird für den nächsten Schritt in der `Dockerfile` .

Das Beschichten von `RUN` Anweisungen und das Generieren von Commits entspricht den Kernkonzepten von Docker, bei denen Commits billig sind und Container von jedem Punkt in der Historie eines Bildes erstellt werden können, ähnlich wie bei der Quellcodeverwaltung.

Die exec Form ermöglicht es , Shell String munging zu vermeiden und `RUN` Befehle ein Basisbild

verwenden , die nicht die angegebene ausführbare Shell enthält.

Die Standard-Shell für das Shell-Formular kann mit dem Befehl `SHELL` geändert werden.

In der Shell-Form können Sie einen `\` (Backslash) verwenden, um einen einzelnen `RUN` Befehl in der nächsten Zeile fortzusetzen. Betrachten Sie zum Beispiel diese beiden Zeilen:

```
RUN /bin/bash -c 'source $HOME/.bashrc ;\
echo $HOME'
```

Zusammen entsprechen sie dieser einzelnen Zeile:

```
RUN /bin/bash -c 'source $HOME/.bashrc ; echo $HOME'
```

Hinweis: Wenn Sie eine andere Shell als '/ bin / sh' verwenden möchten, verwenden Sie das Exec-Formular, das in der gewünschten Shell übergeben wird. Zum Beispiel `RUN ["/bin/bash", "-c", "echo hello"]`

Hinweis: Das exec-Formular wird als JSON-Array analysiert. Das bedeutet, dass Sie doppelte Anführungszeichen (`"`) für Wörter verwenden müssen, nicht einfache Anführungszeichen (`'`).

Hinweis: Im Gegensatz zum Shell-Formular ruft das Exec-Formular keine Befehls-Shell auf. Dies bedeutet, dass eine normale Shell-Verarbeitung nicht stattfindet. Zum Beispiel führt `RUN [ "echo", "$HOME" ]` keine Variablensubstitution bei `$HOME` . Wenn Sie eine Shell-Verarbeitung wünschen, verwenden Sie entweder die Shell-Form oder führen Sie eine Shell direkt aus, zum Beispiel: `RUN [ "sh", "-c", "echo $HOME" ]` .

Hinweis: Im JSON-Formular müssen umgekehrte Schrägstriche ausgeblendet werden. Dies ist insbesondere unter Windows relevant, wo der Backslash das Pfadtrennzeichen ist. Die folgende Zeile wird ansonsten als Shell-Form behandelt, da sie nicht gültig ist und unerwartet fehlschlägt: `RUN ["c:\windows\system32\tasklist.exe"]`

Die korrekte Syntax für dieses Beispiel lautet: `RUN ["c:\\windows\\system32\\tasklist.exe"]`

Der Cache für `RUN` Anweisungen wird beim nächsten Build nicht automatisch ungültig gemacht. Der Cache für eine Anweisung wie `RUN apt-get dist-upgrade -y` wird beim nächsten Build wiederverwendet. Der Cache für `RUN` Anweisungen kann mit dem Flag --no-cache ungültig gemacht werden, z. B. Docker-Build --no-cache.

Weitere Informationen finden Sie im Handbuch Dockerfile Best Practices.

Der Cache für `RUN` Anweisungen kann durch `ADD` Anweisungen ungültig gemacht werden. Details finden Sie unten.

ONBUILD-Anweisung

```
ONBUILD [INSTRUCTION]
```

Der `ONBUILD` Befehl fügt dem Bild einen Auslöserbefehl hinzu, der zu einem späteren Zeitpunkt

ausgeführt wird, wenn das Bild als Basis für einen anderen Build verwendet wird. Der Trigger wird im Kontext des Downstream-Builds ausgeführt, als ob er unmittelbar nach dem `FROM` Befehl in die Downstream-Docker-Datei eingefügt worden wäre.

Jede Build-Anweisung kann als Auslöser registriert werden.

Dies ist nützlich, wenn Sie ein Abbild erstellen, das als Basis zum Erstellen anderer Abbilder verwendet wird, z. B. eine Anwendungserstellungsumgebung oder einen Dämon, der mit einer benutzerspezifischen Konfiguration angepasst werden kann.

Wenn es sich bei Ihrem Image beispielsweise um einen wiederverwendbaren Python-Anwendungsersteller handelt, muss der Anwendungsquellcode in einem bestimmten Verzeichnis hinzugefügt werden. Möglicherweise muss danach ein Build-Skript aufgerufen werden. Sie können jetzt nicht einfach `ADD` und `RUN` aufrufen, da Sie noch keinen Zugriff auf den Anwendungsquellcode haben. Dieser wird für jeden Anwendungsaufbau unterschiedlich sein. Sie können Anwendungsentwickler einfach mit einer Boilerplate-Docker-Datei versehen, um sie in ihre Anwendung zu kopieren. Dies ist jedoch ineffizient, fehleranfällig und schwierig zu aktualisieren, da sie mit anwendungsspezifischem Code gemischt wird.

Die Lösung besteht darin, `ONBUILD` zu verwenden, um erweiterte Anweisungen für die spätere Ausführung während der nächsten `ONBUILD` zu registrieren.

So funktioniert das:

Wenn er auf eine `ONBUILD` Anweisung `ONBUILD` , fügt der Builder den Metadaten des zu `ONBUILD` einen Auslöser hinzu. Die Anweisung wirkt sich sonst nicht auf den aktuellen Build aus.

Am Ende des Builds wird im Image-Manifest eine Liste aller Trigger unter dem Schlüssel OnBuild gespeichert. Sie können mit dem `docker inspect` Befehl `docker inspect` . Später kann das Image mit der `FROM` Anweisung als Basis für einen neuen Build verwendet werden. Bei der Verarbeitung der `FROM` Anweisung sucht der Downstream-Builder nach `ONBUILD` Triggern und führt sie in der Reihenfolge aus, in der sie registriert wurden. Wenn einer der Trigger fehlschlägt, wird der `FROM` Befehl abgebrochen, wodurch der Build fehlschlägt. Wenn alle Trigger erfolgreich sind, wird die `FROM` Anweisung abgeschlossen und der Build wird wie gewohnt fortgesetzt.

Trigger werden nach ihrer Ausführung aus dem endgültigen Bild gelöscht. Mit anderen Worten, sie werden nicht von Enkelkindern vererbt.

Zum Beispiel könnten Sie so etwas hinzufügen:

```
[...]
ONBUILD ADD . /app/src
ONBUILD RUN /usr/local/bin/python-build --dir /app/src
[...]
```

Warnung: Das Verketten von `ONBUILD` Anweisungen mit `ONBUILD ONBUILD` ist nicht zulässig.

Achtung: Die `ONBUILD` Anweisung nicht auslösen können `FROM` oder `MAINTAINER` Anweisungen.

STOPSIGNAL-Anweisung

```
STOPSIGNAL signal
```

Die Anweisung `STOPSIGNAL` setzt das Systemaufrufsignal, das an den Container gesendet wird, zum Beenden. Dieses Signal kann eine gültige vorzeichenlose Zahl sein, die mit einer Position in der Syscall-Tabelle des Kernels übereinstimmt, beispielsweise 9, oder ein Signalname im Format SIGNAME, beispielsweise SIGKILL.

HEALTHCHECK-Anweisung

Der `HEALTHCHECK` Befehl hat zwei Formen:

```
HEALTHCHECK [OPTIONS] CMD command (check container health by running a command inside the
container)
HEALTHCHECK NONE (disable any healthcheck inherited from the base image)
```

Die Anweisung `HEALTHCHECK` teilt Docker mit, wie ein Container getestet werden kann, um zu überprüfen, ob er noch funktioniert. Dadurch können Fälle wie ein Webserver erkannt werden, der sich in einer Endlosschleife befindet und keine neuen Verbindungen verarbeiten kann, obwohl der Serverprozess noch ausgeführt wird.

Wenn für einen Container eine Integritätsprüfung angegeben wurde, hat er zusätzlich zum normalen Status einen Integritätsstatus. Dieser Status wird zunächst gestartet. Wenn eine Gesundheitsprüfung bestanden wird, wird sie gesund (in welchem Zustand sie zuvor war). Nach einer bestimmten Anzahl aufeinander folgender Fehler wird es ungesund.

Die Optionen, die vor `CMD` werden können, sind:

```
--interval=DURATION (default: 30s)
--timeout=DURATION (default: 30s)
--retries=N (default: 3)
```

Die Zustandsprüfung wird erst Sekunden nach dem Start des Containers ausgeführt und danach erneut Sekunden nach jeder vorherigen Prüfung.

Wenn ein einzelner Durchlauf der Prüfung länger dauert als Timeout-Sekunden, gilt die Prüfung als nicht bestanden.

Wiederholungen aufeinanderfolgender Fehler der Integritätsprüfung erfordern, dass der Container als fehlerhaft eingestuft wird.

Es kann nur eine `HEALTHCHECK` Anweisung in einer `Dockerfile` . Wenn Sie mehrere Listen `HEALTHCHECK` wird nur der letzte `HEALTHCHECK` wirksam.

Der Befehl nach dem `CMD` Schlüsselwort kann entweder ein Shell-Befehl (z. B. `HEALTHCHECK CMD /bin/check-running`) oder ein Exec-Array (wie bei anderen Dockerfile-Befehlen) sein (siehe `ENTRYPOINT` für Details).

Der Beendigungsstatus des Befehls zeigt den Integritätsstatus des Containers an. Die möglichen Werte sind:

- `0: success` - der Behälter ist gesund und einsatzbereit
- `1: unhealthy` - Der Container funktioniert nicht ordnungsgemäß
- `2: starting` - Der Container ist noch nicht betriebsbereit, funktioniert aber ordnungsgemäß

Wenn die Sonde 2 zurückgibt ("Start"), wenn der Container bereits den Status "Start" verlassen hat, wird er stattdessen als "ungesund" behandelt.

Um beispielsweise alle fünf Minuten zu überprüfen, dass ein Webserver die Hauptseite der Website innerhalb von drei Sekunden bereitstellen kann:

```
HEALTHCHECK --interval=5m --timeout=3s \
  CMD curl -f http://localhost/ || exit 1
```

Um das Debuggen fehlgeschlagener Tests zu erleichtern, wird der von stdout oder stderr geschriebene Ausgabetext (UTF-8-codiert) im Integritätsstatus gespeichert und kann mit `docker inspect` abgefragt werden. Diese Ausgabe sollte kurz gehalten werden (derzeit werden nur die ersten 4096 Bytes gespeichert).

Wenn sich der `health_status` eines Containers ändert, wird ein `health_status` Ereignis mit dem neuen Status generiert.

Die `HEALTHCHECK` Funktion wurde in Docker 1.12 hinzugefügt.

SHELL-Anweisung

```
SHELL ["executable", "parameters"]
```

Mit der `SHELL` Anweisung kann die Standard-Shell, die für die Shell-Form von Befehlen verwendet wird, überschrieben werden. Die Standardshell unter Linux ist `["/bin/sh", "-c"]` und unter Windows `["cmd", "/S", "/C"]` . Die `SHELL` Anweisung muss in einer Dockerfile in JSON-Form geschrieben werden.

Die `SHELL` Anweisung ist besonders nützlich unter Windows, wo es zwei häufig verwendete und recht unterschiedliche native Shells gibt: cmd und powershell sowie alternative Shells, einschließlich sh.

Die `SHELL` Anweisung kann mehrmals erscheinen. Jede `SHELL` Anweisung überschreibt alle vorherigen `SHELL` Anweisungen und wirkt sich auf alle nachfolgenden Anweisungen aus. Zum Beispiel:

```
FROM windowsservercore

# Executed as cmd /S /C echo default
RUN echo default

# Executed as cmd /S /C powershell -command Write-Host default
RUN powershell -command Write-Host default
```

```
# Executed as powershell -command Write-Host hello
SHELL ["powershell", "-command"]
RUN Write-Host hello

# Executed as cmd /S /C echo hello
SHELL ["cmd", "/S"", "/C"]
RUN echo hello
```

Die folgenden Anweisungen können von der `SHELL` Anweisung beeinflusst werden, wenn ihre Shell-Form in einer Docker-Datei verwendet wird: `RUN` , `CMD` und `ENTRYPOINT` .

Das folgende Beispiel ist ein allgemeines Muster unter Windows, das mit der `SHELL` Anweisung optimiert werden kann:

```
...
RUN powershell -command Execute-MyCmdlet -param1 "c:\foo.txt"
...
```

Der vom Docker aufgerufene Befehl lautet:

```
cmd /S /C powershell -command Execute-MyCmdlet -param1 "c:\foo.txt"
```

Dies ist aus zwei Gründen ineffizient. Zunächst wird ein nicht notwendiger Befehlsprozessor cmd.exe (auch als Shell bezeichnet) aufgerufen. Zweitens erfordert jede `RUN` Anweisung in der Shell-Form einen zusätzlichen Powershell-Befehl, der dem Befehl vorangestellt wird.

Um dies effizienter zu gestalten, kann einer von zwei Mechanismen eingesetzt werden. Verwenden Sie zum Beispiel die JSON-Form des `RUN` Befehls, z.

```
...
RUN ["powershell", "-command", "Execute-MyCmdlet", "-param1 \"c:\\foo.txt\""]
...
```

Das JSON-Formular ist zwar eindeutig und verwendet nicht die nicht benötigte cmd.exe, es erfordert jedoch mehr Ausführlichkeit durch doppelte Anführungszeichen und Escape-Anweisungen. Der alternative Mechanismus besteht darin, die `SHELL` Anweisung und die Shell-Form zu verwenden, um eine natürlichere Syntax für Windows-Benutzer zu schaffen, insbesondere in Kombination mit der Escape-Parser-Direktive:

```
# escape=`

FROM windowsservercore
SHELL ["powershell","-command"]
RUN New-Item -ItemType Directory C:\Example
ADD Execute-MyCmdlet.ps1 c:\example\
RUN c:\example\Execute-MyCmdlet -sample 'hello world'
```

Ergebend:

```
PS E:\docker\build\shell> docker build -t shell .
```

```
Sending build context to Docker daemon 3.584 kB
Step 1 : FROM windowsservercore
 ---> 5bc36a335344
Step 2 : SHELL powershell -command
 ---> Running in 87d7a64c9751
 ---> 4327358436c1
Removing intermediate container 87d7a64c9751
Step 3 : RUN New-Item -ItemType Directory C:\Example
 ---> Running in 3e6ba16b8df9


Directory: C:\


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         6/2/2016   2:59 PM                Example


 ---> 1f1dfdcec085
Removing intermediate container 3e6ba16b8df9
Step 4 : ADD Execute-MyCmdlet.ps1 c:\example\
 ---> 6770b4c17f29
Removing intermediate container b139e34291dc
Step 5 : RUN c:\example\Execute-MyCmdlet -sample 'hello world'
 ---> Running in abdcf50dfd1f
Hello from Execute-MyCmdlet.ps1 - passed hello world
 ---> ba0e25255fda
Removing intermediate container abdcf50dfd1f
Successfully built ba0e25255fda
PS E:\docker\build\shell>
```

Die `SHELL` Anweisung kann auch verwendet werden, um die Funktionsweise einer Shell zu ändern. Bei Verwendung von `SHELL cmd /S /C /V:ON|OFF` unter Windows könnte die Erweiterungssemantik der verzögerten Umgebungsvariablen geändert werden.

Die `SHELL` Anweisung kann auch unter Linux verwendet werden, wenn eine alternative Shell wie zsh, csh, tcsh und andere erforderlich ist.

Die `SHELL` Funktion wurde in Docker 1.12 hinzugefügt.

Debian / Ubuntu-Pakete installieren

Führen Sie die Installation mit einem einzigen Ausführungsbefehl aus, um das Update zusammenzuführen und zu installieren. Wenn Sie später weitere Pakete hinzufügen, wird das Update erneut ausgeführt und alle benötigten Pakete installiert. Wenn das Update separat ausgeführt wird, wird es zwischengespeichert, und die Paketinstallation kann fehlschlagen. Das Setzen des Frontends auf nicht interaktiv und das Übergeben von -y zur Installation ist für Skriptinstallationen erforderlich. Durch das Reinigen und Löschen am Ende der Installation wird die Größe der Ebene minimiert.

```
FROM debian

RUN apt-get update \
 && DEBIAN_FRONTEND=noninteractive apt-get install -y \
```

```
    git \
    openssh-client \
    sudo \
    vim \
    wget \
&& apt-get clean \
&& rm -rf /var/lib/apt/lists/*
```

Dockerfiles online lesen: https://riptutorial.com/de/docker/topic/3161/dockerfiles

Kapitel 18: Docker-Maschine

Einführung

Fernverwaltung mehrerer Docker-Engine-Hosts.

Bemerkungen

`docker-machine` verwaltet Remote-Hosts, auf denen Docker ausgeführt wird.

Das `docker-machine` Befehlszeilentool verwaltet den gesamten Lebenszyklus der Maschine mithilfe von anbieterspezifischen Treibern. Damit kann eine "aktive" Maschine ausgewählt werden. Nach der Auswahl kann eine aktive Maschine als lokale Docker Engine verwendet werden.

Examples

Erhalten Sie aktuelle Informationen zur Docker Machine-Umgebung

All dies sind Shell-Befehle.

`docker-machine env` , um die aktuelle Standard-Docker-Machine-Konfiguration abzurufen

`eval $(docker-machine env)` , um die aktuelle Konfiguration der Docker-Maschine abzurufen und die aktuelle Shell-Umgebung für die Verwendung dieser Docker-Maschine einzurichten.

Wenn Ihre Shell für die Verwendung eines Proxy eingerichtet ist, können Sie die Option --no-proxy angeben, um den Proxy bei der Verbindung mit Ihrem Docker-Computer zu umgehen: `eval $(docker-machine env --no-proxy)`

Wenn Sie über mehrere Docker-Maschinen verfügen, können Sie den Computernamen als Argument angeben: `eval $(docker-machine env --no-proxy machinename)`

SSH in eine Docker-Maschine

All dies sind Shell-Befehle

- Wenn Sie sich direkt an einem laufenden Docker-Computer anmelden müssen, können Sie Folgendes tun:

`docker-machine ssh` an ssh in die Standard-Docker-Maschine

`docker-machine ssh machinename` in ssh in eine nicht standardmäßige Docker-Maschine

- Wenn Sie nur einen einzelnen Befehl ausführen möchten, können Sie dies tun. Um die `uptime` auf dem Standard-Docker-Computer auszuführen, um zu sehen, wie lange er läuft, führen Sie den `docker-machine ssh default uptime`

Erstellen Sie eine Docker-Maschine

Die `docker-machine` ist die beste Methode, um Docker auf einer Maschine zu installieren. Dabei werden automatisch die besten verfügbaren Sicherheitseinstellungen angewendet, einschließlich der Erzeugung eines eindeutigen Paares von SSL-Zertifikaten für die gegenseitige Authentifizierung und SSH-Schlüssel.

So erstellen Sie einen lokalen Computer mit Virtualbox:

```
docker-machine create --driver virtualbox docker-host-1
```

Verwenden Sie den `generic` Treiber, um Docker auf einem vorhandenen Computer zu installieren:

```
docker-machine -D create -d generic --generic-ip-address 1.2.3.4 docker-host-2
```

Die Option `--driver` teilt Docker mit, wie die Maschine erstellt werden soll. Eine Liste der unterstützten Treiber finden Sie unter:

- offiziell unterstützt
- dritte Seite

Docker-Maschinen auflisten

Die Auflistung von Docker-Maschinen gibt den Status, die Adresse und die Version von Docker aller Docker-Maschinen zurück.

```
 docker-machine ls
```

Druckt etwas wie:

```
NAME             ACTIVE   DRIVER    STATE     URL                         SWARM   DOCKER
ERRORS
docker-machine-1 -        ovh       Running   tcp://1.2.3.4:2376                  v1.11.2
docker-machine-2 -        generic   Running   tcp://1.2.3.5:2376                  v1.11.2
```

Um die laufenden Maschinen aufzulisten:

```
docker-machine ls --filter state=running
```

Fehlerlisten auflisten:

```
docker-machine ls --filter state=
```

Verwenden Sie den Golang-Filter, um Maschinen aufzulisten, deren Name mit "side-project-" beginnt.

```
docker-machine ls --filter name="^side-project-"
```

Um nur die Liste der Maschinen-URLs abzurufen:

```
docker-machine ls --format '{{ .URL }}'
```

Die vollständige Befehlsreferenz finden Sie unter https://docs.docker.com/machine/reference/ls/ .

Aktualisieren Sie eine Docker-Maschine

Das Aktualisieren einer Docker-Maschine bedeutet Ausfallzeiten und erfordert möglicherweise eine Planung. Führen Sie zum Aktualisieren einer Docker-Maschine Folgendes aus:

```
docker-machine upgrade docker-machine-name
```

Dieser Befehl hat keine Optionen

Rufen Sie die IP-Adresse einer Docker-Maschine ab

Um die IP-Adresse einer Docker-Maschine zu erhalten, können Sie dies mit diesem Befehl tun:

```
docker-machine ip machine-name
```

Docker-Maschine online lesen: https://riptutorial.com/de/docker/topic/1349/docker-maschine

Kapitel 19: Docker-Net-Modi (Bridge, Hots, zugeordneter Container und keiner).

Einführung

Fertig machen

Bridge-Modus Dies ist eine Standardeinstellung und ist an die Docker0-Bridge angeschlossen. Platzieren Sie den Container in einem vollständig separaten Netzwerk-Namespace.

Hostmodus Wenn Container nur ein Prozess ist, der in einem Host ausgeführt wird, hängen wir den Container an die Host-NIC an.

Zugeordneter Containermodus Dieser Modus ordnet einen neuen Container im Wesentlichen einem vorhandenen Container-Netzwerkstapel zu. Es wird auch als "Container im Containermodus" bezeichnet.

Keine Gibt an, dass Docker den Container ohne Konfiguration in einen eigenen Netzwerkstapel setzt

Examples

Brückenmodus, Hostmodus und zugeordneter Containermodus

Brückenmodus

```
$ docker run -d --name my_app -p 10000:80 image_name
```

Beachten Sie, dass wir **--net = bridge** nicht angeben **mussten,** da dies der Standardarbeitsmodus für Docker ist. Auf diese Weise können mehrere Container auf demselben Host ausgeführt werden, ohne dass ein dynamischer Port zugewiesen werden muss. Im **BRIDGE-**Modus wird also vermieden, dass die Ports kollidieren, und es ist sicher, da jeder Container seinen eigenen privaten Netzwerk-Namespace ausführt.

Host-Modus

```
$ docker run -d --name my_app -net=host image_name
```

Da der Host-Netzwerk-Namespace verwendet wird, ist keine spezielle Konfiguration erforderlich, kann jedoch zu Sicherheitsproblemen führen.

Zugeordneter Containermodus

Dieser Modus ordnet einen neuen Container im Wesentlichen einem vorhandenen Container-Netzwerkstapel zu. Dies bedeutet, dass Netzwerkressourcen wie IP-Adresse und

Anschlusszuordnungen des ersten Containers vom zweiten Container gemeinsam genutzt werden. Dies wird auch als "Container in Container" -Modus bezeichnet. Angenommen, Sie haben zwei Konten wie web_container_1 und web_container_2, und wir führen web_container_2 im zugeordneten Containermodus aus. Lassen Sie uns zuerst web_container_1 herunterladen und mit folgendem Befehl in den getrennten Modus ausführen.

```
$ docker run -d --name web1 -p 80:80 USERNAME/web_container_1
```

Sobald es heruntergeladen ist, schauen wir uns das an. Hier haben wir nur einen Port einem Container zugeordnet, der im Standard-Bridge-Modus ausgeführt wird. Lassen Sie uns nun einen zweiten Container im zugeordneten Containermodus ausführen. Das machen wir mit diesem Befehl.

```
$ docker run -d --name web2 --net=container:web1 USERNAME/web_container_2
```

Wenn Sie nun einfach die Schnittstelleninformationen zu beiden Konten erhalten, erhalten Sie dieselbe Netzwerkkonfiguration. Dies beinhaltet eigentlich den HOST-Modus, der genaue Informationen des Hosts enthält. Der erste Container wurde im Standard-Bridge-Modus ausgeführt, und der zweite Container wird im zugeordneten Containermodus ausgeführt. Sehr ähnliche Ergebnisse können wir erzielen, indem der erste Container im Hostmodus und der zweite Container im zugeordneten Containermodus gestartet werden.

Docker-Net-Modi (Bridge, Hots, zugeordneter Container und keiner). online lesen: https://riptutorial.com/de/docker/topic/9643/docker-net-modi--bridge--hots--zugeordneter-container-und-keiner--

Kapitel 20: Docker-Netzwerk

Examples

So finden Sie die Host-IP des Containers

Sie müssen die IP-Adresse des Containers ermitteln, der im Host ausgeführt wird, damit Sie beispielsweise eine Verbindung zum Webserver herstellen können, der auf dem Host ausgeführt wird.

`docker-machine` wird unter MacOSX und Windows verwendet.

Listen Sie zunächst Ihre Maschinen auf:

```
$ docker-machine ls

NAME      ACTIVE   DRIVER       STATE     URL                         SWARM
default   *        virtualbox   Running   tcp://192.168.99.100:2376
```

Wählen Sie dann eine der Maschinen (die Standardmaschine heißt Standard) und:

```
$ docker-machine ip default

192.168.99.100
```

Ein Docker-Netzwerk erstellen

```
docker network create app-backend
```

Dieser Befehl erstellt ein einfaches überbrücktes Netzwerk mit dem Namen `appBackend` . Standardmäßig sind keine Container an dieses Netzwerk angeschlossen.

Netzwerke auflisten

```
docker network ls
```

Dieser Befehl listet alle Netzwerke auf, die auf dem lokalen Docker-Host erstellt wurden. Es umfasst das Standard-Bridge- `bridge` Netzwerk, das Host- `host` Netzwerk und das Null- `null` Netzwerk. Alle Container sind standardmäßig mit dem Standard-Bridge- `bridge` Netzwerk verbunden.

Container zum Netzwerk hinzufügen

```
docker network connect app-backend myAwesomeApp-1
```

Dieser Befehl verbindet den `myAwesomeApp-1` Container mit dem `app-backend` Netzwerk. Wenn Sie einem benutzerdefinierten Netzwerk einen Container hinzufügen, ermöglicht der eingebettete DNS-Auflöser (der kein voll ausgestatteter DNS-Server ist und nicht exportierbar ist), dass jeder Container im Netzwerk jeden anderen Container im selben Netzwerk auflösen kann. Dieser einfache DNS-Resolver ist im Standard-Bridge- `bridge` Netzwerk nicht verfügbar.

Container vom Netzwerk trennen

```
docker network disconnect app-backend myAwesomeApp-1
```

Dieser Befehl `myAwesomeApp-1` den `myAwesomeApp-1` Container vom `app-backend` Netzwerk. Der Container kann nicht mehr mit anderen Containern in dem Netzwerk kommunizieren, von dem er getrennt wurde, und der eingebettete DNS-Resolver kann keine anderen Container in dem Netzwerk suchen, von dem er getrennt wurde.

Entfernen Sie ein Docker-Netzwerk

```
docker network rm app-backend
```

Dieser Befehl entfernt das benutzerdefinierte `app-backend` Netzwerk vom Docker-Host. Alle Container im Netzwerk, die nicht über ein anderes Netzwerk verbunden sind, verlieren die Kommunikation mit anderen Containern. Das Standard-Bridge- `bridge` Netzwerk, das `host` Host-Netzwerk oder das `null` Null-Netzwerk kann nicht entfernt werden.

Überprüfen Sie ein Docker-Netzwerk

```
docker network inspect app-backend
```

Dieser Befehl gibt Details zum `app-backend` Netzwerk aus.

Die Ausgabe dieses Befehls sollte folgendermaßen aussehen:

```
[
    {
        "Name": "foo",
        "Id": "a0349d78c8fd7c16f5940bdbaf1adec8d8399b8309b2e8a969bd4e3226a6fc58",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1/16"
                }
            ]
        },
        "Internal": false,
```

```
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

Docker-Netzwerk online lesen: https://riptutorial.com/de/docker/topic/3221/docker-netzwerk

Kapitel 21: Docker-Registrierung

Examples

Ausführen der Registrierung

`registry:latest` nicht verwenden `registry:latest` ! Dieses Bild zeigt auf die alte v1-Registrierung. Dieses Python-Projekt wird nicht mehr entwickelt. Die neue v2-Registry ist in Go geschrieben und wird aktiv gewartet. Wenn sich Leute auf eine "private Registry" beziehen, beziehen sie sich auf die v2-Registry, *nicht auf* die v1-Registry!

```
docker run -d -p 5000:5000 --name="registry" registry:2
```

Der obige Befehl führt die neueste Version der Registrierung aus, die sich im Docker Distribution-Projekt befindet .

Weitere Beispiele für Bildverwaltungsfunktionen, z. B. Kennzeichnen, Ziehen oder Pushing, finden Sie im Abschnitt zum Verwalten von Bildern.

Konfigurieren Sie die Registrierung mit AWS S3 Storage Backend

Das Konfigurieren einer privaten Registry für die Verwendung eines AWS S3- Backends ist einfach. Die Registry kann dies automatisch mit der richtigen Konfiguration durchführen. Hier ist ein Beispiel, was in Ihrer `config.yml` Datei enthalten sein sollte:

```
storage:
    s3:
        accesskey: AKAAAAAACCCCCCCBBBDA
        secretkey: rn9rjnNuX44iK+26qpM4cDEoOnonbBW98FYaiDtS
        region: us-east-1
        bucket: registry.example.com
        encrypt: false
        secure: true
        v4auth: true
        chunksize: 5242880
        rootdirectory: /registry
```

Die `accesskey` und `secretkey` Felder sind IAM - Anmeldeinformationen mit spezifischen S3 - Berechtigungen (siehe die Dokumentation für weitere Informationen). Es kann genauso einfach Anmeldeinformationen mit der `AmazonS3FullAccess` Richtlinie verwenden . Die `region` ist die Region Ihres S3-Buckets. Der `bucket` ist der Bucket-Name. Sie können Ihre Bilder mit `encrypt` . Das `secure` Feld gibt die Verwendung von HTTPS an. Sie sollten `v4auth` im Allgemeinen auf true `v4auth` , obwohl der Standardwert false ist. Das `chunksize` Feld ermöglicht es Ihnen, die S3-API-Anforderung einzuhalten, dass die Größe der hochgeladenen Uploads mindestens fünf Megabyte beträgt. Schließlich gibt `rootdirectory` ein Verzeichnis an, das unter Ihrem S3-Bucket verwendet werden soll.

Es gibt andere Speicher-Backends , die genauso einfach konfiguriert werden können.

Docker-Registrierung online lesen: https://riptutorial.com/de/docker/topic/4173/docker-registrierung

Kapitel 22: Docker-Schwarm-Modus

Einführung

Ein Schwarm ist eine Reihe von Docker Engines (oder *Knoten*), die *Dienste* gemeinsam bereitstellen. Swarm wird verwendet, um die Verarbeitung auf viele physische, virtuelle oder Cloud-Maschinen zu verteilen.

Syntax

- Initialisieren eines Schwarms : Docker-Schwarm-Init [OPTIONEN]
- Treten Sie einem Knoten als Knoten und / oder Manager bei : Docker-Schwarm tritt [OPTIONS] HOST: PORT bei
- Erstellen Sie einen neuen Dienst : Andockdienst Erstellen Sie [OPTIONEN] IMAGE [BEFEHL] [ARG ...].
- Anzeigen detaillierter Informationen zu einem oder mehreren Diensten : Andockdienst prüfen [OPTIONEN] SERVICE [SERVICE ...]
- Liste Dienste : Docker Service ls [Optionen]
- Entfernen Sie einen oder mehrere Dienste : docker service rm SERVICE [SERVICE ...]
- Skalieren Sie einen oder mehrere replizierte Dienste : Dockerservicemaßstab SERVICE = REPLICAS [SERVICE = REPLICAS ...]
- Auflisten der Aufgaben eines oder mehrerer Dienste : Andockdienst ps [OPTIONEN] SERVICE [SERVICE ...]
- Aktualisieren Sie einen Service : Docker-Service-Update [OPTIONEN] SERVICE

Bemerkungen

Der Schwarmmodus implementiert die folgenden Funktionen:

- Cluster-Management in Docker Engine integriert
- Dezentrales Design
- Deklaratives Servicemodell
- Skalieren
- Gewünschter Staatenabgleich
- Multi-Host-Netzwerk
- Service Discovery
- Lastverteilung
- Standardmäßig sicheres Design

- Aktualisierungen

Weitere offizielle Docker-Dokumentation zu Swarm finden Sie unter: Swarm-Modusübersicht

CLI-Befehle für den Schwarmmodus

Klicken Sie zur Beschreibung auf die Befehlsbeschreibung

Einen Schwarm initialisieren

```
docker swarm init [OPTIONS]
```

Treten Sie einem Knoten als Knoten und / oder Manager bei

```
docker swarm join [OPTIONS] HOST:PORT
```

Erstellen Sie einen neuen Dienst

```
docker service create [OPTIONS] IMAGE [COMMAND] [ARG...]
```

Detaillierte Informationen zu einem oder mehreren Diensten anzeigen

```
docker service inspect [OPTIONS] SERVICE [SERVICE...]
```

Services auflisten

```
docker service ls [OPTIONS]
```

Entfernen Sie einen oder mehrere Dienste

```
docker service rm SERVICE [SERVICE...]
```

Skalieren Sie einen oder mehrere replizierte Dienste

```
docker service scale SERVICE=REPLICAS [SERVICE=REPLICAS...]
```

Listen Sie die Aufgaben eines oder mehrerer Dienste auf

```
docker service ps [OPTIONS] SERVICE [SERVICE...]
```

Aktualisieren Sie einen Dienst

```
docker service update [OPTIONS] SERVICE
```

Examples

Erstellen Sie einen Schwarm unter Linux mit Docker-Machine und VirtualBox

```
# Create the nodes
# In a real world scenario we would use at least 3 managers to cover the fail of one manager.
docker-machine create -d virtualbox manager
docker-machine create -d virtualbox worker1

# Create the swarm
# It is possible to define a port for the *advertise-addr* and *listen-addr*, if none is
defined the default port 2377 will be used.
docker-machine ssh manager \
    docker swarm init \
    --advertise-addr $(docker-machine ip manager)
    --listen-addr $(docker-machine ip manager)

# Extract the Tokens for joining the Swarm
# There are 2 different Tokens for joining the swarm.
MANAGER_TOKEN=$(docker-machine ssh manager docker swarm join-token manager --quiet)
WORKER_TOKEN=$(docker-machine ssh manager docker swarm join-token worker --quiet)


# Join a worker node with the worker token
docker-machine ssh worker1 \
    docker swarm join \
    --token $WORKER_TOKEN \
    --listen-addr $(docker-machine ip worker1) \
    $(docker-machine ip manager):2377
```

Finde heraus, wie Arbeiter und Manager Token beitreten

Wenn Sie die Bereitstellung neuer Knoten für einen Schwarm automatisieren, müssen Sie wissen, was der richtige Join-Token für den Schwarm ist, sowie die angegebene Adresse des Managers. Sie können dies herausfinden, indem Sie die folgenden Befehle auf einem der vorhandenen Managerknoten ausführen:

```
# grab the ipaddress:port of the manager (second last line minus the whitespace)
export MANAGER_ADDRESS=$(docker swarm join-token worker | tail -n 2 |  tr -d '[[:space:]]')

# grab the manager and worker token
export MANAGER_TOKEN=$(docker swarm join-token manager -q)
export WORKER_TOKEN=$(docker swarm join-token worker -q)
```

Die Option -q gibt nur das Token aus. Ohne diese Option erhalten Sie den vollständigen Befehl zur Registrierung bei einem Schwarm.

Auf neu bereitgestellten Knoten können Sie dann mit dem Schwarm beitreten.

```
docker swarm join --token $WORKER_TOKEN $MANAGER_ADDRESS
```

Hallo Weltanwendung

Normalerweise möchten Sie einen Stapel von Services erstellen, um eine replizierte und orchestrierte Anwendung zu bilden.

Eine typische moderne Webanwendung besteht aus Datenbank, API, Frontend und Reverse Proxy.

Beharrlichkeit

Die Datenbank benötigt Persistenz, daher benötigen wir ein Dateisystem, das von allen Knoten eines Schwarms gemeinsam genutzt wird. Es kann sich dabei um NAS, NFS-Server, GFS2 oder etwas anderes handeln. Das Einrichten ist hier nicht möglich. Momentan enthält Docker keine Persistenz in einem Schwarm und verwaltet diese auch nicht. In diesem Beispiel wird davon `/nfs/` dass `/nfs/` shared location auf allen Knoten bereitgestellt ist.

Netzwerk

Um miteinander kommunizieren zu können, müssen sich Dienste in einem Schwarm im selben Netzwerk befinden.

Wählen Sie einen IP-Bereich (hier `10.0.9.0/24`) und den Netzwerknamen (`hello-network`) und führen Sie einen Befehl aus:

```
docker network create \
  --driver overlay \
  --subnet 10.0.9.0/24 \
  --opt encrypted \
  hello-network
```

Datenbank

Der erste Service, den wir brauchen, ist eine Datenbank. Verwenden wir postgresql als Beispiel. Erstellen Sie einen Ordner für eine Datenbank in `nfs/postgres` und führen Sie diesen aus:

```
docker service create --replicas 1 --name hello-db \
       --network hello-network -e PGDATA=/var/lib/postgresql/data \
       --mount type=bind,src=/nfs/postgres,dst=/var/lib/postgresql/data \
       kiasaki/alpine-postgres:9.5
```

Beachten Sie, dass wir die Optionen `--network hello-network` und `--mount` haben.

API

Das Erstellen einer API ist außerhalb des Anwendungsbereichs dieses Beispiels. Nehmen wir also an, Sie haben ein API-Image unter `username/hello-api` .

```
docker service create --replicas 1 --name hello-api \
       --network hello-network \
       -e NODE_ENV=production -e PORT=80 -e POSTGRESQL_HOST=hello-db \
       username/hello-api
```

Beachten Sie, dass wir einen Namen unseres Datenbankdienstes übergeben haben. Der Docker-Schwarm verfügt über einen integrierten Round-Robin-DNS-Server, sodass die API unter Verwendung ihres DNS-Namens eine Verbindung zur Datenbank herstellen kann.

Reverse Proxy

Lassen Sie uns einen Nginx-Service erstellen, um unsere API für eine äußere Welt bereitzustellen. Erstellen Sie Nginx-Konfigurationsdateien an einem freigegebenen Speicherort und führen Sie Folgendes aus:

```
docker service create --replicas 1 --name hello-load-balancer \
       --network hello-network \
       --mount type=bind,src=/nfs/nginx/nginx.conf,dst=/etc/nginx/nginx.conf \
       -p 80:80 \
       nginx:1.10-alpine
```

Beachten Sie, dass wir die Option `-p` , um einen Port zu veröffentlichen. Dieser Port steht jedem Knoten in einem Schwarm zur Verfügung.

Verfügbarkeit der Knoten

Verfügbarkeit des Schwarmmodusknotens:

- Aktiv bedeutet, dass der Scheduler einem Knoten Aufgaben zuweisen kann.
- Pause bedeutet, dass der Scheduler dem Knoten keine neuen Aufgaben zuweist, aber vorhandene Aufgaben bleiben aktiv.
- Entleeren bedeutet, dass der Scheduler dem Knoten keine neuen Aufgaben zuweist. Der Scheduler fährt alle vorhandenen Aufgaben herunter und plant sie auf einem verfügbaren Knoten.

So ändern Sie die Verfügbarkeit des Modus:

```
#Following commands can be used on swarm manager(s)
docker node update --availability drain node-1
#to verify:
docker node ls
```

Schwarmknoten fördern oder herabstufen

Führen Sie zum `docker node promote` eines Knotens oder einer Knotengruppe einen `docker node promote` von einem Managerknoten aus aus:

```
docker node promote node-3 node-2

Node node-3 promoted to a manager in the swarm.
Node node-2 promoted to a manager in the swarm.
```

Führen Sie zum Herabstufen eines Knotens oder einer Gruppe von Knoten einen `docker node demote` von einem Manager-Knoten aus aus:

```
docker node demote node-3 node-2

Manager node-3 demoted in the swarm.
Manager node-2 demoted in the swarm.
```

Den Schwarm verlassen

Arbeiterknoten:

```
#Run the following on the worker node to leave the swarm.

docker swarm leave
Node left the swarm.
```

Wenn der Knoten die *Manager*- Rolle hat, erhalten Sie eine Warnung, dass das Quorum der Manager beibehalten wird. Sie können --force verwenden, um auf dem Manager-Knoten zu bleiben:

```
#Manager Node

docker swarm leave --force
Node left the swarm.
```

Knoten, die den Swarm verlassen haben, werden weiterhin in der Ausgabe des `docker node ls` .

So entfernen Sie Knoten aus der Liste:

```
docker node rm node-2

node-2
```

Docker-Schwarm-Modus online lesen: https://riptutorial.com/de/docker/topic/749/docker-schwarm-modus

Kapitel 23: Einrichten eines Drei-Knoten-Mongo-Replikats mit Docker Image und Bereitstellen mit Chef

Einführung

In dieser Dokumentation wird beschrieben, wie ein Mongo-Replikensatz mit drei Knoten mit Docker Image erstellt und mit Chef automatisch bereitgestellt wird.

Examples

Schritt bauen

Schritte:

1. Generieren Sie eine Base 64-Schlüsseldatei für die Authentifizierung des Mongo-Knotens. Diese Datei in chef data_bags ablegen
2. Gehen Sie zum Chefkochfachgeschäft und laden Sie das Docker-Kochbuch herunter. Erstellen Sie ein benutzerdefiniertes Kochbuch (z. B. custom_mongo) und fügen Sie der Metadata.rb Ihres Kochbuchs 'docker', '~> 2.0' hinzu
3. Erstellen Sie Attribute und Rezepte in Ihrem benutzerdefinierten Kochbuch
4. Initialisieren Sie Mongo, um einen Rep Set-Cluster zu bilden

Schritt 1: Schlüsseldatei erstellen

Erstellen Sie data_bag mit dem Namen mongo-keyfile und das Element keyfile. Dies befindet sich im Verzeichnis data_bags im Chef. Artikelinhalt wird wie folgt sein

```
openssl rand -base64 756 > <path-to-keyfile>
```

Keyfile-Elementinhalt

```
{
  "id": "keyfile",
  "comment": "Mongo Repset keyfile",
  "key-file": "generated base 64 key above"
}
```

Schritt 2: Laden Sie das Docker-Kochbuch vom Chef-Supper-Markt herunter und erstellen Sie anschließend das Kochbuch custom_mongo

```
knife cookbook site download docker
knife cookbook create custom_mongo
```

in metadat.rb von custom_mongo add

```
depends          'docker', '~> 2.0'
```

Schritt 3: Attribut und Rezept erstellen

Attribute

```
default['custom_mongo']['mongo_keyfile'] = '/data/keyfile'
default['custom_mongo']['mongo_datadir'] = '/data/db'
default['custom_mongo']['mongo_datapath'] = '/data'
default['custom_mongo']['keyfilename'] = 'mongodb-keyfile'
```

Rezept

```
#
# Cookbook Name:: custom_mongo
# Recipe:: default
#
# Copyright 2017, Innocent Anigbo
#
# All rights reserved - Do Not Redistribute
#

data_path = "#{node['custom_mongo']['mongo_datapath']}"
data_dir = "#{node['custom_mongo']['mongo_datadir']}"
key_dir = "#{node['custom_mongo']['mongo_keyfile']}"
keyfile_content = data_bag_item('mongo-keyfile', 'keyfile')
keyfile_name = "#{node['custom_mongo']['keyfilename']}"

#chown of keyfile to docker user
execute 'assign-user' do
 command "chown 999 #{key_dir}/#{keyfile_name}"
 action :nothing
end

#Declaration to create Mongo data DIR and Keyfile DIR
%W[ #{data_path} #{data_dir} #{key_dir} ].each do |path|
directory path do
  mode '0755'
  end
end

#declaration to copy keyfile from data_bag to keyfile DIR on your mongo server
file "#{key_dir}/#{keyfile_name}" do
  content keyfile_content['key-file']
  group 'root'
  mode '0400'
  notifies :run, 'execute[assign-user]', :immediately
end

#Install docker
docker_service 'default' do
  action [:create, :start]
```

```
end

#Install mongo 3.4.2
docker_image 'mongo' do
  tag '3.4.2'
  action :pull
end
```

Erstellen Sie eine Rolle namens Mongo-Rolle im Rollenverzeichnis

```
{
  "name": "mongo-role",
  "description": "mongo DB Role",
  "run_list": [
    "recipe[custom_mongo]"
  ]
}
```

Fügen Sie der drei Mongo-Knoten-Ausführungsliste oben eine Rolle hinzu

```
knife node run_list add FQDN_of_node_01 'role[mongo-role]'
knife node run_list add FQDN_of_node_02 'role[mongo-role]'
knife node run_list add FQDN_of_node_03 'role[mongo-role]'
```

Schritt 4: Initialisieren Sie die drei Knoten Mongo, um den Repset zu bilden

Ich gehe davon aus, dass die obige Rolle bereits auf alle drei Mongo-Knoten angewendet wurde. Starten Sie Mongo nur mit Knoten 01 mit --auth, um die Authentifizierung zu aktivieren

```
docker run --name mongo -v /data/db:/data/db -v /data/keyfile:/opt/keyfile --hostname="mongo-
01.example.com" -p 27017:27017 -d mongo:3.4.2 --keyFile /opt/keyfile/mongodb-keyfile --auth
```

Greifen Sie auf die interaktive Shell des ausgeführten Docker-Containers auf Knoten 01 und Create admin user zu

```
docker exec -it mongo /bin/sh
    mongo
    use admin
    db.createUser( {
         user: "admin-user",
         pwd: "password",
         roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
       });
```

Root-Benutzer erstellen

```
db.createUser( {
         user: "RootAdmin",
         pwd: "password",
         roles: [ { role: "root", db: "admin" } ]
       });
```

Stoppen und löschen Sie den Docker-Container, der oben auf Knoten 01 erstellt wurde. Dies hat

keine Auswirkungen auf die Daten und die Schlüsseldatei im Host-DIR. Nach dem Löschen starten Sie Mongo erneut auf Knoten 01, diesmal jedoch mit Repset-Flag

```
docker rm -fv mongo
docker run --name mongo-uat -v /data/db:/data/db -v /data/keyfile:/opt/keyfile --
hostname="mongo-01.example.com" -p 27017:27017 -d mongo:3.4.2 --keyFile /opt/keyfile/mongodb-
keyfile --replSet "rs0"
```

Starten Sie nun den Mongo auf den Knoten 02 und 03 mit dem Rep-Set-Flag

```
docker run --name mongo -v /data/db:/data/db -v /data/keyfile:/opt/keyfile --hostname="mongo-
02.example.com" -p 27017:27017 -d mongo:3.4.2 --keyFile /opt/keyfile/mongodb-keyfile --replSet
"rs0"
docker run --name mongo -v /data/db:/data/db -v /data/keyfile:/opt/keyfile --hostname="mongo-
03.example.com" -p 27017:27017 -d mongo:3.4.2 --keyFile /opt/keyfile/mongodb-keyfile --replSet
"rs0"
```

Authentifizieren Sie sich mit dem Root-Benutzer auf Knoten 01 und initiieren Sie den Replikatsatz

```
use admin
db.auth("RootAdmin", "password");
rs.initiate()
```

Fügen Sie auf Knoten 01 Knoten 2 und 3 zum Replikatsatz hinzu, um den repset0-Cluster zu bilden

```
rs.add("mongo-02.example.com")
rs.add("mongo-03.example.com")
```

Testen

Führen Sie auf der Primärdatenbank db.printSlaveReplicationInfo () aus, und beobachten Sie SyncedTo und Behind die Primärzeit. Die spätere sollte 0 s wie unten sein

Ausgabe

```
rs0:PRIMARY> db.printSlaveReplicationInfo()
   source: mongo-02.example.com:27017
           syncedTo: Mon Mar 27 2017 15:01:04 GMT+0000 (UTC)
           0 secs (0 hrs) behind the primary
   source: mongo-03.example.com:27017
           syncedTo: Mon Mar 27 2017 15:01:04 GMT+0000 (UTC)
           0 secs (0 hrs) behind the primary
```

Ich hoffe das hilft jemandem

Einrichten eines Drei-Knoten-Mongo-Replikats mit Docker Image und Bereitstellen mit Chef online lesen: https://riptutorial.com/de/docker/topic/10014/einrichten-eines-drei-knoten-mongo-replikats-mit-docker-image-und-bereitstellen-mit-chef

Kapitel 24: Erstellen eines Dienstes mit Persistenz

Syntax

- Docker-Volume erstellen - Name <Volume-Name> # Erstellt ein Volume mit dem Namen <Volume-Name>
- Docker-Run -v <Volume_Name>: <Mount_point> -d crramirez / limesurvey: latest # Mounten Sie das Volume <volume_name> im Verzeichnis <mount_point> im Container

Parameter

Parameter	Einzelheiten
--name <Datenträgername>	Geben Sie den zu erstellenden Datenträgernamen an
-v <Datenträgername>: <Einhängepunkt>	Geben Sie an, wo das benannte Volume im Container bereitgestellt werden soll

Bemerkungen

Persistenz wird in Docker-Containern mit Volumes erstellt. Docker haben viele Möglichkeiten, mit Datenträgern umzugehen. Benannte Bände sind sehr bequem durch:

- Sie bleiben auch dann bestehen, wenn der Container mit der Option -v entfernt wird.
- Die einzige Möglichkeit, ein benanntes Volume zu löschen, besteht darin, einen expliziten Aufruf an das Docker-Volume rm durchzuführen
- Die benannten Volumes können von Containern ohne Verknüpfung oder der Option --volumes-from gemeinsam genutzt werden.
- Sie haben keine Berechtigungsprobleme, die von Host-bereitgestellten Volumes verursacht werden.
- Sie können mit dem Docker-Volume-Befehl bearbeitet werden.

Examples

Persistenz mit benannten Datenträgern

Persistenz wird in Docker-Containern mit Volumes erstellt. Lassen Sie uns einen Limesurvey-Container erstellen und die Datenbank, den hochgeladenen Inhalt und die Konfiguration in einem benannten Volume beibehalten:

```
docker volume create --name mysql
```

```
docker volume create --name upload

docker run -d --name limesurvey -v mysql:/var/lib/mysql -v upload:/app/upload -p 80:80
crramirez/limesurvey:latest
```

Sichern Sie einen benannten Volume-Inhalt

Wir müssen einen Container erstellen, um das Volume bereitzustellen. Dann archivieren Sie es und laden Sie das Archiv auf unseren Host herunter.

Erstellen wir zunächst ein Datenvolumen mit einigen Daten:

```
docker volume create --name=data
echo "Hello World" |  docker run -i --rm=true -v data:/data ubuntu:trusty tee /data/hello.txt
```

Lassen Sie uns die Daten sichern:

```
docker run -d --name backup -v data:/data ubuntu:trusty tar -czvf /tmp/data.tgz /data
docker cp backup:/tmp/data.tgz data.tgz
docker rm -fv backup
```

Lass uns testen:

```
tar -xzvf data.tgz
cat data/hello.txt
```

Erstellen eines Dienstes mit Persistenz online lesen:
https://riptutorial.com/de/docker/topic/7429/erstellen-eines-dienstes-mit-persistenz

Kapitel 25: geheime Daten an einen laufenden Container übergeben

Examples

Möglichkeiten, Geheimnisse in einem Container weiterzugeben

Der nicht sehr sichere Weg (da `docker inspect` zeigt) ist das Übergeben einer Umgebungsvariablen an

```
docker run
```

sowie

```
docker run -e password=abc
```

oder in einer Datei

```
docker run --env-file myfile
```

wo meine datei enthalten kann

```
password1=abc password2=def
```

Es ist auch möglich, sie in einem Volume abzulegen

```
docker run -v $(pwd)/my-secret-file:/secret-file
```

einige bessere Möglichkeiten verwenden

keywhiz https://square.github.io/keywhiz/

Tresor https://www.hashicorp.com/blog/vault.html

etcd mit Crypt https://xordataexchange.github.io/crypt/

geheime Daten an einen laufenden Container übergeben online lesen: https://riptutorial.com/de/docker/topic/6481/geheime-daten-an-einen-laufenden-container-ubergeben

Kapitel 26: Iptables mit Docker

Einführung

In diesem Thema wird beschrieben, wie Sie den Zugriff auf Ihre Docker-Container mithilfe von iptables von außerhalb der Welt einschränken.

Für ungeduldige Menschen können Sie die Beispiele überprüfen. Für die anderen lesen Sie bitte den Bemerkungsabschnitt, um zu erfahren, wie Sie neue Regeln erstellen.

Syntax

- iptables -I DOCKER [RULE ...] [ACCEPT | DROP] // Eine Regel oben in der DOCKER-Tabelle hinzufügen
- iptables -D DOCKER [RULE ...] [ACCEPT | DROP] // Eine Regel aus der DOCKER-Tabelle entfernen
- ipset restore </etc/ipfriends.conf // Um Ihre ipset *ipfriends* neu zu konfigurieren

Parameter

Parameter	Einzelheiten
ext_if	Ihre externe Schnittstelle auf dem Docker-Host.
XXX.XXX.XXX.XXX	Eine bestimmte IP-Adresse, auf die Docker-Container zugreifen, sollte angegeben werden.
YYY.YYY.YYY.YYY	Eine weitere IP, auf die Docker-Container zugreifen sollen, sollte angegeben werden.
ipfriends	Der ipset-Name, der die IPs definiert, die zum Zugriff auf Ihre Docker-Container berechtigt sind.

Bemerkungen

Das Problem

Das Konfigurieren von iptables-Regeln für Docker-Container ist etwas schwierig. Zuerst würden Sie denken, dass "klassische" Firewall-Regeln den Trick tun sollten.

Nehmen wir beispielsweise an, Sie haben einen nginx-proxy-Container + mehrere Service-Container konfiguriert, um einige persönliche Web-Services über HTTPS verfügbar zu machen. Dann sollte eine Regel wie diese nur für IP XXX.XXX.XXX.XXX Zugriff auf Ihre Webdienste

gewähren.

```
$ iptables -A INPUT -i eth0 -p tcp -s XXX.XXX.XXX.XXX -j ACCEPT
$ iptables -P INPUT DROP
```

Es funktioniert nicht, Ihre Container sind für jeden zugänglich.

In der Tat sind Docker-Container keine Hostdienste. Sie sind auf ein virtuelles Netzwerk in Ihrem Host angewiesen, und der Host fungiert als Gateway für dieses Netzwerk. In Bezug auf Gateways wird gerouteter Verkehr nicht von der INPUT-Tabelle verarbeitet, sondern von der FORWARD-Tabelle, die die Veröffentlichung der Regel als unwirksam macht.

Aber es ist noch nicht alles. In der Tat erstellt der Docker-Daemon eine Vielzahl von iptables-Regeln, wenn er seine Magie in Bezug auf die Netzwerkkonnektivität von Containern beginnt. Insbesondere wird eine DOCKER-Tabelle erstellt, um Regeln für Container zu behandeln, indem der Verkehr von der FORWARD-Tabelle an diese neue Tabelle weitergeleitet wird.

```
$ iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination
DOCKER-ISOLATION  all  --  anywhere             anywhere
DOCKER     all  --  anywhere             anywhere
ACCEPT     all  --  anywhere             anywhere             ctstate RELATED,ESTABLISHED
ACCEPT     all  --  anywhere             anywhere
ACCEPT     all  --  anywhere             anywhere
DOCKER     all  --  anywhere             anywhere
ACCEPT     all  --  anywhere             anywhere             ctstate RELATED,ESTABLISHED
ACCEPT     all  --  anywhere             anywhere
ACCEPT     all  --  anywhere             anywhere

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain DOCKER (2 references)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             172.18.0.4           tcp dpt:https
ACCEPT     tcp  --  anywhere             172.18.0.4           tcp dpt:http

Chain DOCKER-ISOLATION (1 references)
target     prot opt source               destination
DROP       all  --  anywhere             anywhere
DROP       all  --  anywhere             anywhere
RETURN     all  --  anywhere             anywhere
```

Die Lösung

Wenn Sie die offizielle Dokumentation (https://docs.docker.com/v1.5/articles/networking/) lesen , wird eine erste Lösung angegeben, um den Zugriff des Docker-Containers auf eine bestimmte IP-Adresse zu beschränken.

```
$ iptables –I DOCKER –i ext_if ! –s 8.8.8.8 –j DROP
```

Das Hinzufügen einer Regel oben in der DOCKER-Tabelle ist eine gute Idee. Die von Docker automatisch konfigurierten Regeln werden nicht beeinträchtigt. Dies ist einfach. Aber zwei Hauptmängel:

- Erstens, was ist, wenn Sie von zwei IP-Adressen aus auf eine zugreifen müssen? Hier kann nur eine Src-IP akzeptiert werden, eine andere wird verworfen, ohne dass dies verhindert werden kann.
- Zweitens, was ist, wenn Ihr Docker Zugang zum Internet benötigt? Praktisch keine Anfrage wird erfolgreich sein, da nur der Server 8.8.8.8 darauf reagieren könnte.
- Was ist schließlich, wenn Sie weitere Logiken hinzufügen möchten? Geben Sie beispielsweise jedem Benutzer Zugriff auf Ihren Webserver, der über das HTTP-Protokoll bereitgestellt wird, beschränken Sie jedoch alles andere auf eine bestimmte IP-Adresse.

Für die erste Beobachtung können wir *ipset verwenden* . Anstatt eine IP in der obigen Regel zuzulassen, lassen wir alle IPs aus dem vordefinierten ipset zu. Als Bonus kann das ipset aktualisiert werden, ohne dass die iptable-Regel neu definiert werden muss.

```
$ iptables –I DOCKER –i ext_if –m set ! --match-set my-ipset src –j DROP
```

Für die zweite Beobachtung ist dies ein kanonisches Problem für Firewalls: Wenn Sie einen Server über eine Firewall kontaktieren dürfen, sollte die Firewall den Server dazu berechtigen, auf Ihre Anfrage zu antworten. Dies kann durch Autorisieren von Paketen geschehen, die sich auf eine bestehende Verbindung beziehen. Für die Docker-Logik gibt es:

```
$ iptables –I DOCKER –i ext_if –m state --state ESTABLISHED,RELATED –j ACCEPT
```

Die letzte Beobachtung konzentriert sich auf einen Punkt: iptables-Regeln sind wesentlich. Tatsächlich muss eine zusätzliche Logik zum ACCEPT-Aktivieren einiger Verbindungen (einschließlich der Verbindungen, die ESTABLISHED-Verbindungen betreffen) vor der DROP-Regel an die Spitze der DOCKER-Tabelle gestellt werden, die alle verbleibenden Verbindungen ablehnen, die nicht zum ipset passen.

Da wir die Option -I von iptable verwenden, die Regeln oben in der Tabelle einfügt, müssen die vorherigen iptables-Regeln in umgekehrter Reihenfolge eingefügt werden:

```
// Drop rule for non matching IPs
$ iptables –I DOCKER –i ext_if –m set ! --match-set my-ipset src –j DROP
// Then Accept rules for established connections
$ iptables –I DOCKER –i ext_if –m state --state ESTABLISHED,RELATED –j ACCEPT
$ iptables –I DOCKER –i ext_if ... ACCEPT // Then 3rd custom accept rule
$ iptables –I DOCKER –i ext_if ... ACCEPT // Then 2nd custom accept rule
$ iptables –I DOCKER –i ext_if ... ACCEPT // Then 1st custom accept rule
```

In Anbetracht dessen können Sie nun die Beispiele überprüfen, die diese Konfiguration veranschaulichen.

Examples

Beschränken Sie den Zugriff auf Docker-Container auf eine Reihe von IP-Adressen

Installieren Sie zuerst *ipset,* falls erforderlich. Bitte beziehen Sie sich auf Ihre Distribution, um zu erfahren, wie es geht. Als Beispiel ist hier der Befehl für Debian-ähnliche Distributionen.

```
$ apt-get update
$ apt-get install ipset
```

Erstellen Sie anschließend eine Konfigurationsdatei, um ein ipset zu definieren, das die IPs enthält, für die Sie Zugriff auf Ihre Docker-Container öffnen möchten.

```
$ vi /etc/ipfriends.conf
# Recreate the ipset if needed, and flush all entries
create -exist ipfriends hash:ip family inet hashsize 1024 maxelem 65536
flush
# Give access to specific ips
add ipfriends XXX.XXX.XXX.XXX
add ipfriends YYY.YYY.YYY.YYY
```

Laden Sie dieses ipset.

```
$ ipset restore < /etc/ipfriends.conf
```

Stellen Sie sicher, dass Ihr Docker-Dämon ausgeführt wird: Nach Eingabe des folgenden Befehls sollte kein Fehler angezeigt werden.

```
$ docker ps
```

Sie können nun Ihre iptables-Regeln einfügen. Sie **müssen** die Reihenfolge respektieren.

```
// All requests of src ips not matching the ones from ipset ipfriends will be dropped.
$ iptables -I DOCKER -i ext_if -m set ! --match-set ipfriends src -j DROP
// Except for requests coming from a connection already established.
$ iptables -I DOCKER -i ext_if -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Wenn Sie neue Regeln erstellen möchten, müssen Sie vor dem Einfügen der neuen Regeln alle benutzerdefinierten Regeln entfernen, die Sie hinzugefügt haben.

```
$ iptables -D DOCKER -i ext_if -m set ! --match-set ipfriends src -j DROP
$ iptables -D DOCKER -i ext_if -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Konfigurieren Sie den Einschränkungszugriff beim Starten des Docker-Daemons

In Arbeit

Einige benutzerdefinierte iptables-Regeln

In Arbeit

Iptables mit Docker online lesen: https://riptutorial.com/de/docker/topic/9201/iptables-mit-docker

Kapitel 27: Konsul in Docker 1.12 Schwarm laufen lassen

Examples

Führen Sie Konsul in einem Docker 1.12-Schwarm aus

Dies setzt voraus, dass das offizielle Konsul-Docker-Image den Konsul im Cluster-Modus in einem Docker-Schwarm mit dem neuen Schwarm-Modus in Docker 1.12 ausführt. Dieses Beispiel basiert auf http://qnib.org/2016/08/11/consul-service/ . Kurz gesagt besteht die Idee darin, zwei Docker-Schwarmdienste zu verwenden, die miteinander kommunizieren. Dadurch wird das Problem gelöst, dass Sie die IPS der einzelnen Konsulcontainer nicht im Voraus kennen können und Sie sich auf die Docking-Swarm-Dockingstationen verlassen können.

Dies setzt voraus, dass Sie bereits über einen Docker 1.12-Schwarmcluster mit mindestens drei Knoten verfügen.

Sie können einen Protokolltreiber auf Ihren Docker-Daemons konfigurieren, damit Sie sehen können, was passiert. Ich habe den syslog-Treiber dafür verwendet: setze die Option `--log-driver=syslog` auf dockerd.

Erstellen Sie zunächst ein Overlay-Netzwerk für Konsul:

```
docker network create consul-net -d overlay
```

Booten Sie jetzt den Cluster mit nur einem Knoten (der Standardwert --replicas ist 1):

```
docker service create --name consul-seed \
  -p 8301:8300 \
  --network consul-net \
  -e 'CONSUL_BIND_INTERFACE=eth0' \
  consul agent -server -bootstrap-expect=3  -retry-join=consul-seed:8301 -retry-join=consul-
cluster:8300
```

Sie sollten jetzt einen 1-Knoten-Cluster haben. Rufen Sie jetzt den zweiten Dienst auf:

```
docker service create --name consul-cluster \
  -p 8300:8300 \
  --network consul-net \
  --replicas 3 \
  -e 'CONSUL_BIND_INTERFACE=eth0' \
  consul agent -server -retry-join=consul-seed:8301 -retry-join=consul-cluster:8300
```

Sie sollten jetzt einen Konsul-Cluster mit vier Knoten haben. Sie können dies überprüfen, indem Sie einen der Docker-Container ausführen:

```
docker exec <containerid> consul members
```

Konsul in Docker 1.12 Schwarm laufen lassen online lesen:
https://riptutorial.com/de/docker/topic/6437/konsul-in-docker-1-12-schwarm-laufen-lassen

Kapitel 28: Kontrollpunkt und Wiederherstellungscontainer

Examples

Docker mit Checkpoint und Wiederherstellung (Ubuntu) kompilieren

Um Docker zu kompilieren, wird empfohlen, dass Sie mindestens **2 GB RAM haben** . Trotzdem fällt es manchmal aus, so dass es besser ist, stattdessen **4 GB zu** verwenden.

1. Stellen Sie sicher, dass git und make installiert sind

```
sudo apt-get install make git-core -y
```

2. Installieren Sie einen neuen Kernel (mindestens 4.2)

```
sudo apt-get install linux-generic-lts-xenial
```

3. Starten Sie den Computer neu, um den neuen Kernel zu aktivieren

```
sudo reboot
```

4. kompiliere `criu` das benötigt wird, um den `docker checkpoint` auszuführen

```
sudo apt-get install libprotobuf-dev libprotobuf-c0-dev protobuf-c-compiler protobuf-
compiler python-protobuf libnl-3-dev libcap-dev -y
wget http://download.openvz.org/criu/criu-2.4.tar.bz2 -O - | tar -xj
cd criu-2.4
make
make install-lib
make install-criu
```

5. Prüfen Sie, ob alle Anforderungen erfüllt sind, um criu auszuführen

```
sudo criu check
```

6. experimentelles Docker kompilieren (wir benötigen Docker, um Docker zu kompilieren)

```
cd ~
wget -qO- https://get.docker.com/ | sh
sudo usermod -aG docker $(whoami)
```

- **An diesem Punkt müssen wir uns abmelden und erneut anmelden, um einen Docker-Daemon zu haben. Nach dem relog fahren Sie mit dem Kompilierungsschritt fort**

```
git clone https://github.com/boucher/docker
cd docker
git checkout docker-checkpoint-restore
make #that will take some time - drink a coffee
DOCKER_EXPERIMENTAL=1 make binary
```

7. Wir haben jetzt ein kompiliertes Docker. Lässt die Binärdateien verschieben. Stellen Sie sicher, dass Sie `<version>` durch die installierte Version ersetzen

```
sudo service docker stop
sudo cp $(which docker) $(which docker)_ ; sudo cp ./bundles/latest/binary-client/docker-
<version>-dev $(which docker)
sudo cp $(which docker-containerd) $(which docker-containerd)_ ; sudo cp
./bundles/latest/binary-daemon/docker-containerd $(which docker-containerd)
sudo cp $(which docker-containerd-ctr) $(which docker-containerd-ctr)_ ; sudo cp
./bundles/latest/binary-daemon/docker-containerd-ctr $(which docker-containerd-ctr)
sudo cp $(which docker-containerd-shim) $(which docker-containerd-shim)_ ; sudo cp
./bundles/latest/binary-daemon/docker-containerd-shim $(which docker-containerd-shim)
sudo cp $(which dockerd) $(which dockerd)_ ; sudo cp ./bundles/latest/binary-
daemon/dockerd $(which dockerd)
sudo cp $(which docker-runc) $(which docker-runc)_ ; sudo cp ./bundles/latest/binary-
daemon/docker-runc $(which docker-runc)
sudo service docker start
```

Machen Sie sich keine Sorgen - wir haben die alten Binärdateien gesichert. Sie sind immer noch da, jedoch mit einem Unterstrich (`docker_`).

Herzlichen Glückwunsch, Sie haben jetzt ein experimentelles Andockgerät mit der Möglichkeit, einen Container zu überprüfen und ihn wiederherzustellen.

Bitte beachten Sie, dass experimentelle Funktionen NICHT für die Produktion bereit sind

Prüfpunkt und Wiederherstellen eines Containers

```
# create docker container
export cid=$(docker run -d --security-opt seccomp:unconfined busybox /bin/sh -c 'i=0; while
true; do echo $i; i=$(expr $i + 1); sleep 1; done')

# container is started and prints a number every second
# display the output with
docker logs $cid

# checkpoint the container
docker checkpoint create $cid checkpointname

# container is not running anymore
docker np

# lets pass some time to make sure

# resume container
docker start $cid --checkpoint=checkpointname

# print logs again
docker logs $cid
```

Kontrollpunkt und Wiederherstellungscontainer online lesen:
https://riptutorial.com/de/docker/topic/5291/kontrollpunkt-und-wiederherstellungscontainer

Kapitel 29: Konzept der Docker-Volumes

Bemerkungen

Leute, die Docker neu kennen, wissen oft nicht, dass Docker-Dateisysteme standardmäßig temporär sind. Wenn Sie ein Docker-Image starten, erhalten Sie einen Container, der sich auf der Oberfläche wie eine virtuelle Maschine verhält. Sie können Dateien erstellen, ändern und löschen. Im Gegensatz zu einer virtuellen Maschine gehen jedoch alle Änderungen verloren, wenn Sie den Container anhalten und erneut starten. Alle zuvor gelöschten Dateien sind nun wieder vorhanden, und alle neuen Dateien oder Änderungen, die Sie vorgenommen haben, sind nicht vorhanden.

Volumes in Docker-Containern ermöglichen persistente Daten und die gemeinsame Nutzung von Host-Maschinendaten in einem Container.

Examples

A) Starten Sie einen Container mit einem Volumen

```
[root@localhost ~]# docker run -it -v  /data  --name=vol3   8251da35e7a7 /bin/bash
root@d87bf9607836:/# cd /data/
root@d87bf9607836:/data# touch abc{1..10}
root@d87bf9607836:/data# ls
```

abc1 abc10 abc2 abc3 abc4 abc5 abc6 abc7 abc8 abc9

B) Drücken Sie nun [cont + P + Q], um den Container zu verlassen, ohne die Überprüfung des Containers auf den laufenden Container zu beenden

```
[root@localhost ~]# docker ps
```

CONTAINER ID BILDBEFEHL ERSTELLTE STATUSPORTS NAMEN d87bf9607836 8251da35e7a7 "/ bin / bash" Vor etwa einer Minute.

C) Führen Sie 'Docker Inspect' aus, um weitere Informationen zum Volume anzuzeigen

```
[root@localhost ~]# docker inspect  d87bf9607836
```

"" "" "" Anzahl "" "Laufstür. "Modus": "", "RW": wahr

D) Sie können ein laufendes Container-Volume an andere Container anhängen

```
[root@localhost ~]# docker run -it  --volumes-from  vol3  8251da35e7a7  /bin/bash

root@ef2f5cc545be:/# ls
```

bin boot data dev etc home lib lib64 medien mnt opt prozz rennlauf sbin srv sys tmp usr var

`root@ef2f5cc545be:/# ls` / data abc1 abc10 abc2 abc3 abc4 abc5 abc6 abc7 abc8 abc9

E) Sie können auch Ihr Basisverzeichnis in einem Container einhängen

```
[root@localhost ~]# docker run -it  -v  /etc:/etc1 8251da35e7a7 /bin/bash
```

Hier: / etc ist das Host-Maschinenverzeichnis und / etc1 ist das Ziel innerhalb des Containers

Konzept der Docker-Volumes online lesen: https://riptutorial.com/de/docker/topic/5908/konzept-der-docker-volumes

Kapitel 30: Laufende Dienste

Examples

Erstellen eines fortgeschritteneren Dienstes

Im folgenden Beispiel erstellen wir einen Dienst mit dem *Namenvisualisierer* . Wir geben ein benutzerdefiniertes Label an und ordnen den internen Port des Dienstes von 8080 auf 9090 neu zu. Außerdem werden wir ein externes Verzeichnis des Hosts an den Dienst mounten.

```
docker service create \
        --name=visualizer \
        --label com.my.custom.label=visualizer \
        --publish=9090:8080 \
        --mount type=bind,source=/var/run/docker.sock,target=/var/run/docker.sock \
        manomarks/visualizer:latest
```

Einen einfachen Service erstellen

Dieses einfache Beispiel wird einen Hallo-Welt-Webdienst erstellen, der auf dem Port 80 zu hören ist.

```
docker service create \
    --publish 80:80 \
    tutum/hello-world
```

Service entfernen

Dieses einfache Beispiel entfernt den Dienst mit dem Namen "visualizer":

```
docker service rm visualizer
```

Skalieren eines Dienstes

In diesem Beispiel wird der Dienst auf 4 Instanzen skaliert:

```
docker service scale visualizer=4
```

Im Docker Swarm-Modus stoppen wir keinen Dienst. Wir verkleinern es auf null:

```
docker service scale visualizer=0
```

Laufende Dienste online lesen: https://riptutorial.com/de/docker/topic/8802/laufende-dienste

Kapitel 31: Mehrere Prozesse in einer Containerinstanz

Bemerkungen

Normalerweise sollte jeder Container einen Prozess enthalten. Falls Sie mehrere Prozesse in einem Container benötigen (z. B. einen SSH-Server, um sich bei Ihrer laufenden Containerinstanz anzumelden), könnten Sie die Idee haben, ein eigenes Shellskript zu schreiben, das diese Prozesse startet. In diesem Fall mussten Sie sich um die Behandlung des `SIGNAL` kümmern (z. B. das Weiterleiten eines erfassten `SIGINT` an die untergeordneten Prozesse Ihres Skripts). Das ist nicht wirklich das, was du willst. Eine einfache Lösung besteht darin, `supervisord` als Container-Root-Prozess zu verwenden, der sich um die `SIGNAL` Verarbeitung und die Lebensdauer der `SIGNAL` Prozesse kümmert.

Beachten Sie jedoch, dass dies nicht der "Docker-Weg" ist. Um dieses Beispiel auf die Docker-Art zu erhalten, müssen Sie sich beim `docker host` (der Maschine, auf der der Container ausgeführt wird) anmelden und das `docker exec -it container_name /bin/bahs` . Dieser Befehl öffnet eine Shell im Container, wie dies bei ssh der Fall ist.

Examples

Dockerfile + supervisord.conf

Um mehrere Prozesse auszuführen, z. B. einen Apache-Webserver zusammen mit einem SSH-Dämon im gleichen Container, können Sie `supervisord` .

Erstellen Sie Ihre `supervisord.conf` Konfigurationsdatei wie folgt:

```
[supervisord]
nodaemon=true

[program:sshd]
command=/usr/sbin/sshd -D

[program:apache2]
command=/bin/bash -c "source /etc/apache2/envvars && exec /usr/sbin/apache2 -DFOREGROUND"
```

Dann erstellen Sie eine `Dockerfile` wie:

```
FROM ubuntu:16.04
RUN apt-get install -y openssh-server apache2 supervisor
RUN mkdir -p /var/lock/apache2 /var/run/apache2 /var/run/sshd /var/log/supervisor
COPY supervisord.conf /etc/supervisor/conf.d/supervisord.conf
CMD ["/usr/bin/supervisord"]
```

Dann kannst du dein Image erstellen:

```
docker build -t supervisord-test .
```

Danach kannst du es ausführen:

```
$ docker run -p 22 -p 80 -t -i supervisord-test
2016-07-26 13:15:21,101 CRIT Supervisor running as root (no user in config file)
2016-07-26 13:15:21,101 WARN Included extra file     "/etc/supervisor/conf.d/supervisord.conf"
during parsing
2016-07-26 13:15:21,112 INFO supervisord started with pid 1
2016-07-26 13:15:21,113 INFO spawned: 'sshd' with pid 6
2016-07-26 13:15:21,115 INFO spawned: 'apache2' with pid 7
...
```

Mehrere Prozesse in einer Containerinstanz online lesen:
https://riptutorial.com/de/docker/topic/4053/mehrere-prozesse-in-einer-containerinstanz

Kapitel 32: Private / sichere Registrierung für API mit API v2

Einführung

Eine private und sichere Dockerregistrierung anstelle eines Docker Hub. Grundlegende Dockerfähigkeiten sind erforderlich.

Parameter

Befehl	Erläuterung
sudo docker run -p 5000: 5000	Starten Sie einen Docker-Container und binden Sie den Port 5000 vom Container an den Port 5000 der physischen Maschine.
--name Registrierung	Containername (zur besseren Lesbarkeit von "docker ps").
-v 'pwd' / certs: / certs	Binden Sie CURRENT_DIR / certs des physischen Computers an / certs des Containers (wie ein "freigegebener Ordner").
-e REGISTRY_HTTP_TLS_CERTIFICATE = / certs / server.crt	Wir geben an, dass die Registry die Datei /certs/server.crt zum Starten verwenden soll. (Umgebungsvariable)
-e REGISTRY_HTTP_TLS_KEY = / certs / server.key	Gleiches für den RSA-Schlüssel (server.key).
-v / root / images: / var / lib / registry /	Wenn Sie alle Registrierungsabbilder speichern möchten, sollten Sie dies auf dem physischen Computer tun. Hier speichern wir alle Bilder in / root / images auf dem physischen Computer. Wenn Sie dies tun, können Sie die Registrierung anhalten und erneut starten, ohne Bilder zu verlieren.
Registrierung: 2	Wir geben an, dass wir das Registrierungsabbild vom Docker-Hub (oder lokal) ziehen möchten, und fügen "2" hinzu, weil wir die Version 2 der Registrierung installieren möchten.

Bemerkungen

So installieren Sie eine Docker-Engine (in diesem Lernprogramm als Client bezeichnet)

So erstellen Sie ein selbstsigniertes SSL-Zertifikat

Examples

Zertifikate generieren

Generieren Sie einen privaten RSA-Schlüssel: `openssl genrsa -des3 -out server.key 4096`

Openssl sollte in diesem Schritt nach einer Passphrase fragen. Beachten Sie, dass wir nur das Zertifikat für die Kommunikation und Authentifizierung ohne Passphrase verwenden. Verwenden Sie einfach 123456.

Generieren Sie die Zertifikatsignierungsanforderung: `openssl req -new -key server.key -out server.csr`

Dieser Schritt ist wichtig, da Sie nach Informationen zu Zertifikaten gefragt werden. Die wichtigsten Informationen sind "Common Name", dh der Domänenname, der für die Kommunikation zwischen der privaten Docker-Registrierung und allen anderen Computern verwendet wird. Beispiel: mydomain.com

Entfernen `cp server.key server.key.org && openssl rsa -in server.key.org -out server.key` **aus dem privaten RSA-Schlüssel:** `cp server.key server.key.org && openssl rsa -in server.key.org -out server.key`

Wie gesagt, konzentrieren wir uns auf das Zertifikat ohne Passphrase. Seien Sie also vorsichtig mit allen Schlüsseldateien (.key, .csr, .crt) und bewahren Sie sie an einem sicheren Ort auf.

Generieren Sie das selbstsignierte Zertifikat: `openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt`

Sie haben jetzt zwei wichtige Dateien, *server.key* und *server.crt* , die für die private Registrierungsauthentifizierung erforderlich sind.

Führen Sie die Registrierung mit einem selbstsignierten Zertifikat aus

Um die private Registry (sicher) auszuführen, müssen Sie ein selbstsigniertes Zertifikat erstellen. Sie können sich auf das vorherige Beispiel beziehen, um es zu generieren.

Für mein Beispiel habe ich *server.key* und *server.crt* in / root / certs eingefügt

Vor dem Lauf Docker Befehl sollten Sie platziert (verwenden werden `cd`) in das Verzeichnis , das *certs* - Ordner enthält. Wenn dies nicht der Fall ist und Sie versuchen, den Befehl auszuführen, erhalten Sie eine Fehlermeldung wie

> level = fatal msg = "/certs/server.crt öffnen: keine solche Datei oder Verzeichnis"

Wenn Sie (`cd /root` in meinem Beispiel) sind, können Sie die sichere / private Registrierung grundsätzlich mit folgendem `sudo docker run -p 5000:5000 --restart=always --name registry -v `pwd`/certs:/certs -e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/server.crt -e REGISTRY_HTTP_TLS_KEY=/certs/server.key -v /root/Documents:/var/lib/registry/ registry:2` starten:
`sudo docker run -p 5000:5000 --restart=always --name registry -v `pwd`/certs:/certs -e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/server.crt -e REGISTRY_HTTP_TLS_KEY=/certs/server.key -v /root/Documents:/var/lib/registry/ registry:2`
Erläuterungen zum Befehl finden Sie im Parameter-Teil.

Ziehen oder drücken Sie einen Docker-Client

Wenn Sie eine funktionierende Registry ausführen, können Sie Bilder darauf laden oder verschieben. Dazu benötigen Sie die *server.crt* -Datei in einem speziellen Ordner auf Ihrem Docker-Client. Mit dem Zertifikat können Sie sich bei der Registrierung authentifizieren und anschließend die Kommunikation verschlüsseln.

Kopieren Sie *server.crt* von der Registrierungsmaschine in die Datei /etc/docker/certs.d/mydomain.com:5000/ auf Ihrer Clientmaschine. `mv /etc/docker/certs.d/mydomain.com:5000/server.crt /etc/docker/certs.d/mydomain.com:5000/ca-certificates.crt` dann in *ca-certificate.crt um* : `mv /etc/docker/certs.d/mydomain.com:5000/server.crt /etc/docker/certs.d/mydomain.com:5000/ca-certificates.crt`

An dieser Stelle können Sie Bilder aus Ihrer privaten Registry ziehen oder verschieben:
PULL: `docker pull mydomain.com:5000/nginx` oder
DRÜCKEN :

1. Holen Sie sich ein offizielles Image von hub.docker.com: docker `docker pull nginx`
2. `docker tag IMAGE_ID mydomain.com:5000/nginx` dieses Bild, bevor Sie es in die private Registry verschieben: `docker tag IMAGE_ID mydomain.com:5000/nginx` (verwenden Sie `docker images` , um die IMAGE_ID zu erhalten)
3. Schieben Sie das Bild in die Registry: `docker push mydomain.com:5000/nginx`

Private / sichere Registrierung für API mit API v2 online lesen:
https://riptutorial.com/de/docker/topic/8707/private---sichere-registrierung-fur-api-mit-api-v2

Kapitel 33: Protokollierung

Examples

Konfigurieren eines Protokolltreibers im systemd-Dienst

```
[Service]

# empty exec prevents error "docker.service has more than one ExecStart= setting, which is
only allowed for Type=oneshot services. Refusing."
ExecStart=
ExecStart=/usr/bin/dockerd -H fd:// --log-driver=syslog
```

Dies ermöglicht die Syslog-Protokollierung für den Docker-Daemon. Die Datei sollte im entsprechenden Verzeichnis mit dem Eigentümer root erstellt werden. `/etc/systemd/system/docker.service.d` normalerweise `/etc/systemd/system/docker.service.d` auf Ubuntu 16.04.

Überblick

Die Protokollierungsmethode von Docker besteht darin, dass Sie Ihre Container so erstellen, dass Protokolle in die Standardausgabe (Konsole / Terminal) geschrieben werden.

Wenn Sie bereits über einen Container verfügen, der Protokolle in eine Datei schreibt, können Sie ihn umleiten, indem Sie einen symbolischen Link erstellen:

```
ln -sf /dev/stdout /var/log/nginx/access.log
ln -sf /dev/stderr /var/log/nginx/error.log
```

Anschließend können Sie verschiedene Protokolltreiber verwenden, um Ihre Protokolle dort zu platzieren, wo Sie sie benötigen.

Protokollierung online lesen: https://riptutorial.com/de/docker/topic/7378/protokollierung

Kapitel 34: Sicherheit

Einführung

Um unsere Bilder für die Sicherheitspatches auf dem neuesten Stand zu halten, müssen wir wissen, von welchem Basis-Image wir abhängig sind

Examples

Wie Sie herausfinden können, von welchem Bild unser Bild stammt

Betrachten wir als Beispiel einen Wordpress-Container

Die Docker-Datei beginnt mit FROM php: 5.6-apache

so gehen wir zum Dockerfile oben https://github.com/docker-library/php/blob/master/5.6/apache/Dockerfile

und wir finden FROM debian: jessie Das bedeutet, dass ein Sicherheitspatch für Debian jessie erscheint. Wir müssen unser Image erneut aufbauen.

Sicherheit online lesen: https://riptutorial.com/de/docker/topic/8077/sicherheit

Kapitel 35: Simple Node.js-Anwendung ausführen

Examples

Ausführen einer Basic Node.js-Anwendung in einem Container

Das Beispiel, das ich besprechen werde, setzt voraus, dass Sie eine Docker-Installation haben, die in Ihrem System funktioniert, und ein grundlegendes Verständnis für die Arbeit mit Node.js. Wenn Sie wissen, wie Sie mit Docker arbeiten müssen, sollte klar sein, dass das Node.js-Framework nicht auf Ihrem System installiert werden muss. Stattdessen verwenden wir die `latest` Version des von Docker verfügbaren `node` . Daher können Sie das Bild bei Bedarf vorher mit dem Befehls- `docker pull node` herunterladen. (Der Befehl `pulls` automatisch die neueste Version des `node` aus dem Andockfenster.)

1. Fahren Sie mit der Erstellung eines Verzeichnisses fort, in dem sich alle Ihre aktiven Anwendungsdateien befinden würden. Erstellen `package.json` in diesem Verzeichnis eine `package.json` Datei, die Ihre Anwendung sowie die Abhängigkeiten beschreibt. Ihre `package.json` Datei sollte `package.json` so aussehen:

```
{
  "name": "docker_web_app",
  "version": "1.0.0",
  "description": "Node.js on Docker",
  "author": "First Last <first.last@example.com>",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
  "dependencies": {
    "express": "^4.13.3"
  }
}
```

2. Wenn wir mit Node.js arbeiten müssen, erstellen wir normalerweise eine `server` Datei, die eine Webanwendung definiert. In diesem Fall verwenden wir das `Express.js` Framework (ab Version `4.13.3`). Eine grundlegende `server.js` Datei würde `server.js` so aussehen:

```
var express = require('express');
var PORT = 8080;
var app = express();
app.get('/', function (req, res) {
  res.send('Hello world\n');
});

app.listen(PORT);
console.log('Running on http://localhost:' + PORT);
```

3. Für diejenigen, die mit Docker vertraut sind, hätten Sie eine Docker- `Dockerfile` . Eine

`Dockerfile` Datei ist eine Textdatei, die alle Befehle enthält, die zum Erstellen eines benutzerdefinierten Abbilds erforderlich sind, das auf Ihre Anwendung zugeschnitten ist.

Erstellen Sie eine leere Textdatei mit dem Namen `Dockerfile` im aktuellen Verzeichnis. Die Methode zum Erstellen ist in Windows unkompliziert. Unter Linux möchten Sie möglicherweise `touch Dockerfile` in dem Verzeichnis ausführen, das alle für Ihre Anwendung erforderlichen Dateien enthält. Öffnen Sie die Docker-Datei mit einem beliebigen Texteditor und fügen Sie die folgenden Zeilen hinzu:

```
FROM node:latest
RUN mkdir -p /usr/src/my_first_app
WORKDIR /usr/src/my_first_app
COPY package.json /usr/src/my_first_app/
RUN npm install
COPY . /usr/src/my_first_app
EXPOSE 8080
```

- `FROM node:latest` weist den Docker-Dämon an, von welchem Image wir erstellen wollen. In diesem Fall verwenden wir die `latest` Version des offiziellen Docker-Image- `node` die im Docker Hub verfügbar ist.
- In diesem Bild erstellen wir ein Arbeitsverzeichnis, das alle erforderlichen Dateien enthält, und weisen den Dämon an, dieses Verzeichnis als das gewünschte Arbeitsverzeichnis für unsere Anwendung festzulegen. Dazu fügen wir hinzu

  ```
  RUN mkdir -p /usr/src/my_first_app
  WORKDIR /usr/src/my_first_app
  ```

- Anschließend fahren wir mit der Installation von Anwendungsabhängigkeiten fort, indem wir zunächst die `package.json` Datei (die die Anwendungsinformationen einschließlich der Abhängigkeiten angibt) in das Arbeitsverzeichnis `/usr/src/my_first_app` im Image verschieben. Wir machen das durch

  ```
  COPY package.json /usr/src/my_first_app/
  RUN npm install
  ```

- Wir geben dann `COPY . /usr/src/my_first_app` , um alle Anwendungsdateien und Quellcode zum Arbeitsverzeichnis im Image hinzuzufügen.
- Dann verwenden wir die Anweisung `EXPOSE` , um den Daemon anzuweisen, Port `8080` des resultierenden Containers sichtbar zu machen (über eine Container-zu-Host-Zuordnung), da die Anwendung an Port `8080` bindet.
- Im letzten Schritt `node server.js` wir den Dämon an, den Befehlsknoten `node server.js` im Image auszuführen, indem Sie den `node server.js` `npm start` ausführen. Wir verwenden dazu die `CMD` Direktive, die die Befehle als Argumente übernimmt.

  ```
  CMD [ "npm", "start" ]
  ```

4. Wir erstellen dann eine `.dockerignore` -Datei in demselben Verzeichnis wie die `Dockerfile` , um zu verhindern, dass unsere von Node.js-Systeminstallation verwendete Kopie von `node_modules` und Protokollen in das Docker-Image kopiert wird. Die `.dockerignore` Datei muss den folgenden Inhalt haben:

```
node_modules
npm-debug.log
```

5. # Bauen Sie Ihr Bild auf

Navigieren Sie zu dem Verzeichnis, das die Docker- `Dockerfile` enthält, und führen Sie den folgenden Befehl aus, um das Docker-Image zu erstellen. Mit der Markierung `-t` können Sie Ihr Bild mit einem Tag versehen, sodass Sie es später leichter finden können, indem Sie den Befehl docker images verwenden:

```
    $ docker build -t <your username>/node-web-app .
```

Ihr Bild wird jetzt von Docker aufgelistet. Zeigen Sie Bilder mit dem folgenden Befehl an:

```
$ docker images

REPOSITORY                         TAG         ID              CREATED
node                               latest      539c0211cd76    10 minutes ago
<your username>/node-web-app       latest      d64d3505b0d2    1 minute ago
```

6. # Ausführen des Bildes

Wir können jetzt das soeben erstellte Image mit den Inhalten der Anwendung, dem `node` Basis-Image und der `Dockerfile` . Wir führen nun unser neu erstelltes Image `<your username>/node-web-app` . Durch die Option `-d` für den `docker run` Befehl wird der Container im separaten Modus ausgeführt, sodass der Container im Hintergrund ausgeführt wird. Das Flag `-p` leitet einen öffentlichen Port an einen privaten Port innerhalb des Containers weiter. Führen Sie das zuvor erstellte Image mit diesem Befehl aus:

```
$ docker run -p 49160:8080 -d <your username>/node-web-app
```

7. Drucken Sie die Ausgabe Ihrer App, indem Sie `docker ps` auf Ihrem Terminal `docker ps` . Die Ausgabe sollte ungefähr so aussehen.

```
 CONTAINER ID        IMAGE                          COMMAND             CREATED
STATUS              PORTS                      NAMES
 7b701693b294      <your username>/node-web-app   "npm start"         20 minutes ago
Up 48 seconds       0.0.0.0:49160->8080/tcp    loving_goldstine
```

`docker logs <CONTAINER ID>` Sie die Anwendungsausgabe ab, indem Sie `docker logs <CONTAINER ID>` . In diesem Fall handelt es sich um `docker logs 7b701693b294` .

Ausgabe: `Running on http://localhost:8080`

8. Von der `docker ps` Ausgabe von `docker ps` lautet das erhaltene Port-Mapping `0.0.0.0:49160->8080/tcp` . Daher ordnete Docker den `8080` Port im Container dem Port 49160 auf der Hostmaschine zu. Im Browser können wir jetzt `localhost:49160` eingeben `localhost:49160` .

Wir können unsere App auch mit `curl` aufrufen:

```
$ curl -i localhost:49160

HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 12
Date: Sun, 08 Jan 2017 14:00:12 GMT
Connection: keep-alive

Hello world
```

Simple Node.js-Anwendung ausführen online lesen:
https://riptutorial.com/de/docker/topic/8754/simple-node-js-anwendung-ausfuhren

Kapitel 36: Überprüfen eines laufenden Containers

Syntax

- Docker inspizieren [OPTIONEN] CONTAINER | IMAGE [CONTAINER | IMAGE ...]

Examples

Containerinformationen abrufen

Um alle Informationen zu einem Container zu erhalten, können Sie Folgendes ausführen:

```
docker inspect <container>
```

Holen Sie sich bestimmte Informationen aus einem Container

Sie können bestimmte Informationen aus einem Container abrufen, indem Sie Folgendes ausführen:

```
docker inspect -f '<format>' <container>
```

Zum Beispiel können Sie die Netzwerkeinstellungen abrufen, indem Sie Folgendes ausführen:

```
docker inspect -f '{{ .NetworkSettings }}' <container>
```

Sie können auch nur die IP-Adresse erhalten:

```
docker inspect -f '{{ .NetworkSettings.IPAddress }}' <container>
```

Der Parameter -f bedeutet formatiert und erhält eine Go-Vorlage als Eingabe, um das zu erwartende Format zu formatieren. Dies wird jedoch nicht zu einer schönen Rendite führen.

```
docker inspect -f '{{ json .NetworkSettings }}' {{containerIdOrName}}
```

Das Json-Schlüsselwort gibt die Rückgabe als JSON zurück.

Zum Abschluss noch ein kleiner Tipp: Verwenden Sie Python, um die Ausgabe-JSON zu formatieren:

```
docker inspect -f '{{ json .NetworkSettings }}' <container> | python -mjson.tool
```

Und voila, Sie können alles auf dem Docker inspizieren abfragen und es in Ihrem Terminal schön

aussehen lassen.

Es ist auch möglich, ein Dienstprogramm namens " jq " zu verwenden, um die `docker inspect` Befehlsausgabe bei der Befehlsausgabe zu unterstützen.

```
docker inspect -f '{{ json .NetworkSettings }}' aa1 | jq [.Gateway]
```

Der obige Befehl gibt die folgende Ausgabe zurück:

```
[
  "172.17.0.1"
]
```

Diese Ausgabe ist eigentlich eine Liste, die ein Element enthält. Manchmal zeigt die `docker inspect` eine Liste mit mehreren Elementen an, und Sie möchten möglicherweise auf ein bestimmtes Element verweisen. Wenn `Config.Env` beispielsweise mehrere Elemente enthält, können Sie mit `index` auf das erste Element dieser Liste verweisen:

```
docker inspect --format '{{ index (index .Config.Env) 0 }}' <container>
```

Das erste Element ist bei Null indiziert, das heißt, das zweite Element dieser Liste befindet sich am Index `1` :

```
docker inspect --format '{{ index (index .Config.Env) 1 }}' <container>
```

Mit `len` es möglich, die Anzahl der Elemente der Liste abzurufen:

```
docker inspect --format `{{ len .Config.Env }}` <container>
```

Bei negativen Zahlen kann auf das letzte Element der Liste verwiesen werden:

```
docker inspect -format "{{ index .Config.Cmd $[$(docker inspect -format `{{ len .Config.Cmd
}}` <container>)-1]}}" <container>
```

Einige `docker inspect` Informationen werden als Schlüsselwörterbuch bezeichnet: Wert, hier ist ein Auszug aus einem `docker inspect` Inspect eines Jess / Spotify-Containers

```
"Config": { "Hostname": "8255f4804dde", "Domainname": "", "User": "spotify", "AttachStdin":
false, "AttachStdout": false, "AttachStderr": false, "Tty": false, "OpenStdin": false,
"StdinOnce": false, "Env": [ "DISPLAY=unix:0",
"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin", "HOME=/home/spotify" ],
"Cmd": [ "-stylesheet=/home/spotify/spotify-override.css" ], "Image": "jess/spotify", "Volumes":
null, "WorkingDir": "/home/spotify", "Entrypoint": [ "spotify" ], "OnBuild": null, "Labels": {}
},
```

also bekomme ich die Werte des ganzen Config-Abschnitts

```
docker inspect -f '{{.Config}}' 825

{8255f4804dde spotify false false false map[] false false false [DISPLAY=unix:0
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin HOME=/home/spotify] [-
stylesheet=/home/spotify/spotify-override.css] false jess/spotify map[] /home/spotify [spotify]
```

```
false [] map[] }
```

aber auch ein einzelnes Feld, wie der Wert von Config.Image

```
docker inspect -f '{{index (.Config) "Image" }}' 825
```

```
jess/spotify
```

oder Config.Cmd

```
docker inspect -f '{{.Config.Cmd}}' 825
```

```
[-stylesheet=/home/spotify/spotify-override.css]
```

Überprüfen Sie ein Bild

Um ein Bild zu untersuchen, können Sie die Bild-ID oder den Bildnamen verwenden, der aus Repository und Tag besteht. Angenommen, Sie haben das CentOS 6-Basisbild:

```
➜ ~ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
centos              centos6             cf2c3ece5e41        2 weeks ago         194.6 MB
```

In diesem Fall können Sie eine der folgenden Aktionen ausführen:

- ➜ ~ docker inspect cf2c3ece5e41
- ➜ ~ docker inspect centos:centos6

Beide Befehle geben Ihnen alle Informationen, die in einem JSON-Array verfügbar sind:

```
[
    {
        "Id": "sha256:cf2c3ece5e418fd063bfad5e7e8d083182195152f90aac3a5ca4dbfbf6a1fc2a",
        "RepoTags": [
            "centos:centos6"
        ],
        "RepoDigests": [],
        "Parent": "",
        "Comment": "",
        "Created": "2016-07-01T22:34:39.970264448Z",
        "Container": "b355fe9a01a8f95072e4406763138c5ad9ca0a50dbb0ce07387ba905817d6702",
        "ContainerConfig": {
            "Hostname": "68a1f3cfce80",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
            ],
            "Cmd": [
                "/bin/sh",
                "-c",
```

```
                "#(nop) CMD [\"/bin/bash\"]"
            ],
            "Image":
"sha256:cdbcc7980b002dc19b4d5b6ac450993c478927f673339b4e6893647fe2158fa7",
            "Volumes": null,
            "WorkingDir": "",
            "Entrypoint": null,
            "OnBuild": null,
            "Labels": {
                "build-date": "20160701",
                "license": "GPLv2",
                "name": "CentOS Base Image",
                "vendor": "CentOS"
            }
        },
        "DockerVersion": "1.10.3",
        "Author": "https://github.com/CentOS/sig-cloud-instance-images",
        "Config": {
            "Hostname": "68a1f3cfce80",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
            ],
            "Cmd": [
                "/bin/bash"
            ],
            "Image":
"sha256:cdbcc7980b002dc19b4d5b6ac450993c478927f673339b4e6893647fe2158fa7",
            "Volumes": null,
            "WorkingDir": "",
            "Entrypoint": null,
            "OnBuild": null,
            "Labels": {
                "build-date": "20160701",
                "license": "GPLv2",
                "name": "CentOS Base Image",
                "vendor": "CentOS"
            }
        },
        "Architecture": "amd64",
        "Os": "linux",
        "Size": 194606575,
        "VirtualSize": 194606575,
        "GraphDriver": {
            "Name": "aufs",
            "Data": null
        },
        "RootFS": {
            "Type": "layers",
            "Layers": [
                "sha256:2714f4a6cdee9d4c987fef019608a4f61f1cda7ccf423aeb8d7d89f745c58b18"
            ]
        }
    }
```

```
]
```

Spezifische Informationen drucken

Docker `inspect` Stützen Vorlagen gehen über die `--format` Option. Dies ermöglicht eine bessere Integration in Skripts, ohne auf traditionelle Pipes / sed / grep-Tools zurückgreifen zu müssen.

Eine interne IP-Adresse des Containers drucken :

```
docker inspect --format '{{ .NetworkSettings.IPAddress }}' 7786807d8084
```

Dies ist nützlich für den direkten Netzwerkzugriff der automatischen Konfiguration von Load-Balancer.

Drucken Sie eine Container- *Init*- PID :

```
docker inspect --format '{{ .State.Pid }}' 7786807d8084
```

Dies ist nützlich für tiefere Inspektionen mit `/proc` oder Tools wie `strace` .

Fortgeschrittene Formatierung :

```
docker inspect --format 'Container {{ .Name }} listens on {{ .NetworkSettings.IPAddress }}:{{
range $index, $elem := .Config.ExposedPorts }}{{ $index }}{{ end }}' 5765847de886 7786807d8084
```

Wird ausgegeben:

```
Container /redis listens on 172.17.0.3:6379/tcp
Container /api listens on 172.17.0.2:4000/tcp
```

Debuggen der Containerprotokolle mit Docker inspect

`docker inspect` Befehl `docker inspect` kann zum Debuggen der Containerprotokolle verwendet werden.

Stdout und Stderr des Containers können überprüft werden, um den Container zu debuggen, dessen Position mithilfe von `docker inspect` .

Befehl: `docker inspect <container-id> | grep Source`

Es gibt die Position der Container stdout und stderr an.

Stdout / stderr eines laufenden Containers untersuchen

```
docker logs --follow <containerid>
```

Dadurch wird die Ausgabe des laufenden Containers angepasst. Dies ist nützlich, wenn Sie auf dem Docker-Daemon keinen Protokolltreiber eingerichtet haben.

Überprüfen eines laufenden Containers online lesen:
https://riptutorial.com/de/docker/topic/1336/uberprufen-eines-laufenden-containers

Kapitel 37: Wie debuggen, wenn der Andockaufbau fehlschlägt

Einführung

Wenn ein `docker build -t mytag .` schlägt mit einer Nachricht wie `---> Running in d9a42e53eb5a The command '/bin/sh -c returned a non-zero code: 127` (127 bedeutet "Befehl nicht gefunden, aber 1) kann durch 6 oder irgendetwas ersetzt werden;) Es kann nicht trivial sein, den Fehler in einer langen Zeile zu finden

Examples

grundlegendes Beispiel

Als letzte Ebene erstellt von

```
docker build -t mytag .
```

gezeigt

```
---> Running in d9a42e53eb5a
```

Sie starten einfach das zuletzt erstellte Image mit einer Shell und den Befehl, und Sie erhalten eine klarere Fehlermeldung

```
docker run -it d9a42e53eb5a /bin/bash
```

(Dies setzt voraus, dass / bin / bash verfügbar ist. Es kann sich auch um / bin / sh oder etwas anderes handeln.

Mit der Eingabeaufforderung starten Sie den letzten fehlgeschlagenen Befehl und sehen, was angezeigt wird

Wie debuggen, wenn der Andockaufbau fehlschlägt online lesen:
https://riptutorial.com/de/docker/topic/8078/wie-debuggen--wenn-der-andockaufbau-fehlschlagt

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit Docker	abaracedo, Aminadav, Braiam, Carlos Rafael Ramirez, Community, ganesshkumar, HankCa, Josha Inglis, L0j1k, mohan08p, Nathaniel Ford, schumacherj, Siddharth Srinivasan, SztupY, Vishrant
2	Behälter verbinden	Jett Jones
3	Beschränkung des Netzwerkzugriffs auf Container	xeor
4	Bilder bauen	cjsimon, ETL, Ken Cochrane, L0j1k, Nathan Arthur, Nathaniel Ford, Nour Chawich, SztupY, user2915097, Wolfgang
5	Bilder verwalten	akhyar, Björn Enochsson, dsw88, L0j1k, Nathan Arthur, Nathaniel Ford, Szymon Biliński, user2915097, Wolfgang, zygimantus
6	Container debuggen	allprog, Binary Nerd, foraidt, L0j1k, Nathaniel Ford, user2915097, yadutaf
7	Container laufen lassen	abaracedo, Adri C.S., AlcaDotS, atv, Binary Nerd, BMitch, Camilo Silva, Carlos Rafael Ramirez, cizixs, cjsimon, Claudiu, ElMesa, Emil Burzo, enderland, Felipe Plets, ganesshkumar, Gergely Fehérvári, ISanych, L0j1k, Nathan Arthur, Patrick Auld, RoyB, ssice, SztupY, Thomasleveil, tommyyards, VanagaS, Wolfgang, zinking
8	Container verwalten	akhyar, atv, Binary Nerd, BrunoLM, Carlos Rafael Ramirez, Emil Burzo, Felipe Plets, ganesshkumar, L0j1k, Matt, Nathaniel Ford, Rafal Wiliński, Sachin Malhotra, serieznyi, sk8terboi87 ツ, tommyyards, user2915097, Victor Oliveira Antonino, Wolfgang, Xavier Nicollet, zygimantus
9	Datenvolumen und Datencontainer	GameScripting, L0j1k, melihovv
10	Docker Engine-API	Ashish Bista, atv, BMitch, L0j1k, Radoslav Stoyanov, SztupY

11	Docker erfasst alle laufenden Container	Kostiantyn Rybnikov
12	Docker in Docker	Ohmen
13	Docker inspizieren verschiedene Felder für Schlüssel: Wert und Elemente der Liste	user2915097
14	Docker-Datenvolumen	James Hewitt, L0j1k, NRKirby, Nuno Curado, Scott Coates, t3h2mas
15	Docker-Ereignisse	Nathaniel Ford, user2915097
16	Dockerfile-Inhalte bestellen	akhyar, Philip
17	Dockerfiles	BMitch, foraidt, k0pernikus, kubanczyk, L0j1k, ob1, Ohmen, rosysnake, satsumas, Stephen Leppik, Thiago Almeida, Wassim Dhif, yadutaf
18	Docker-Maschine	Amine24h, kubanczyk, Nik Rahmel, user2915097, yadutaf
19	Docker-Net-Modi (Bridge, Hots, zugeordneter Container und keiner).	mohan08p
20	Docker-Netzwerk	HankCa, L0j1k, Nathaniel Ford
21	Docker-Registrierung	Ashish Bista, L0j1k
22	Docker-Schwarm-Modus	abronan, Christian, Farhad Farahi, Jilles van Gurp, kstromeiraos, kubanczyk, ob1, Philip, Vanuan
23	Einrichten eines Drei-Knoten-Mongo-Replikats mit Docker Image und Bereitstellen mit Chef	Innocent Anigbo
24	Erstellen eines Dienstes mit Persistenz	Carlos Rafael Ramirez, Vanuan
25	geheime Daten an einen laufenden Container übergeben	user2915097
26	Iptables mit Docker	Adrien Ferrand
27	Konsul in Docker 1.12	Jilles van Gurp

	Schwarm laufen lassen	
28	Kontrollpunkt und Wiederherstellungscontainer	Bastian, Fuzzyma
29	Konzept der Docker-Volumes	Amit Poonia, Rob Bednark, serieznyi
30	Laufende Dienste	Mateusz Mrozewski, Philip
31	Mehrere Prozesse in einer Containerinstanz	h3nrik, Ohmen, Xavier Nicollet
32	Private / sichere Registrierung für API mit API v2	bastien enjalbert, kubanczyk
33	Protokollierung	Jilles van Gurp, Vanuan
34	Sicherheit	user2915097
35	Simple Node.js-Anwendung ausführen	Siddharth Srinivasan
36	Überprüfen eines laufenden Containers	AlcaDotS, devopskata, Felipe Plets, h3nrik, Jilles van Gurp, L0j1k, Milind Chawre, Nik Rahmel, Stephen Leppik , user2915097, yadutaf
37	Wie debuggen, wenn der Andockaufbau fehlschlägt	user2915097