Docker Merkblatt-1/3

www.datamics.com

Grundlagen

Installation

Installiert werden muss die für das Betriebssystem passende Version von Docker CE:

https://docs.docker.com/install/

Test

Nach Eingabe von

```
docker run hello-world
```

sollte eine Meldung erscheinen, dass die Installation erfolgreich war.

Busybox Container

BusyBox ist eine reduzierte Linux-Kommandozeile.

```
docker pull busybox
docker run busybox echo ”Hallo Welt!”
```

Mit letzterem Befehl führt Docker den BusyBox-Container aus, der wiederum **Hallo Welt!** in der Kommandozeile ausgibt.

Laufenden Container stoppen

```
docker stop NAME
```

NAME kann hierbei der Name oder die ID des Containers sein.

Container anzeigen

```
docker ps -a
```

zeigt alle ausgeführten Container mitsamt Status und ID (z.B. **305297d7a235**) an.

Interaktive Container

Der Parameter **-it** öffnet den interaktiven Modus, in dem der gestartete Container sequentiell Befehle ausführt:

```
docker run -it busybox
```

Container-Umgebung

Bei jedem Start eines Containers wird dessen Umgebung erneut erstellt. Wird beispielsweise BusyBox im interaktiven Modus gestartet und anschliessend

```
rm -rf bin
```

ausgeführt, funktionieren Befehle wie **ls** nicht mehr, da sie gelöscht wurden. Wird der Container beendet und neu gestartet, stehen sie allerdings wieder zur Verfügung.

Wichtige Befehle

Container entfernen

```
docker rm 305297d7a235 ff0a5c3750b9
docker container prune
```

Der erste Befehl enfernt Container mittels ID, der letzte Befehl entfernt alle beendeten Container.

Ports anzeigen

Docker Container bieten ihre Dienste üblicherweise auf festen Ports an:

```
docker port CONTAINER
```

Hierbei ist **CONTAINER** die ID des Containers. Ausgegeben werden alle Ports, die dieser Container verwendet.

Bind mount einrichten

Wenn ein Container persistente Daten speichern soll, kann ein Bind mount eingerichtet werden:

```
docker run –it –name bindmount -v
  PFAD busybox
```

PFAD ist hierbei der Pfad zu einer Datei oder einem Verzeichnis, an dem Daten abgelegt werden sollen.

Volume einrichten

Bevorzugt zu betriebssystemabhängigen Bind mounts werden Volumes. Diese müssen vor der Verwendung jedoch zuerst eingerichtet werden:

```
docker volume create NAME
docker run -it –name volume1
  –mount type=volume,source=NAME,
  target=PFAD busybox
```

PFAD ist hierbei der Pfad zu dem gewünschten Speicherort des Volumes und **NAME** dessen Name. Dabei können mehrere, parallel laufende Container auf das gleiche Volume zugreifen.

Volume löschen

```
docker volume rm NAME
docker volume prune
```

Mit dem zweiten Befehl werden alle Volumes, die nicht mit einem laufenden oder gestoppten Container verbunden sind, gelöscht.

Dateien kopieren

Kopiere eine Datei in einen Container:

```
docker cp DATEINAME
  CONTAINERNAME:/DATEINAME
```

Kopiere eine Datei aus einem Container:

```
docker cp CONTAINERNAME:/DATEINAME
  DATEINAME
```

Images

Alle Images anzeigen

Zeige alle lokal vorhandenen Images an:

```
docker image list
docker image list –filter dangling=true
```

Mit dem zweiten Befehl werden alle ungenutzten Images angezeigt.

Alle Images löschen

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

Wird **–force** als Parameter angehängt, werden auch Images gelöscht, die in Repositories aktiv verwendet werden.

Docker File

Erzeuge eine Datei namens Dockerfile im Verzeichnis der Applikation, aus der ein Image erzeugt werden soll. Befülle die Datei mit folgendem Inhalt:

```
FROM python:3-onbuild
EXPOSE 5000
CMD [”python3”, ”./app.py”]
```

Die erste Zeile macht ein Python3-Image zum Basis-Image. Die zweite Zeile bestimmt, dass der Container auf Port 5000 zur Verfügung steht. Die dritte Zeile startet eine Applikation namens **app.py** mit Python3.

Image bauen

```
docker build -t
  BENUTZERNAME/CONTAINERNAME .
```

Mit **-t** wird DockerHub-Benutzername und Appname zugewiesen. Der letzte Parameter, **.**, ist der Speicherort der Applikation und des Dockerfiles.

Commit

Mit einem Commit kann aus einem veränderten Image ein neues Image erzeugt werden.

```
docker container commit
  CONTAINERNAME
```

Image publizieren

Soll ein eigenes Image der Öffentlichkeit zur Verfügung gestellt werden, kann es auf DockerHub veröffentlicht werden:

```
docker push BENUTZER/IMAGENAME
```

BENUTZER ist der DockerHub-Benutzername.

Images

Image History

Entwicklungsgeschichte eines Images:

```
docker history IMAGENAME
```

Image aktualisieren

```
docker pull IMAGENAME
```

Docker search

```
docker search BEGRIFF
```

sucht nach Images mit dem Term **BEGRIFF** auf DockerHub.

Docker save

Speichere ein oder mehrere Images als **.tar**-Datei:

```
docker save IMAGE
```

Docker import

Importiere ein oder mehrere Images aus einer **.tar**-Datei als Dateisystem:

```
docker import DATEI
```

Weitere Befehle

Docker diff

```
docker diff CONTAINERNAME
```

zeigt die Entwicklung von Dateien und Verzeichnissen eines Containers an.

Docker update

Aktualisiere die Konfiguration von Container **CONTAINER** mit

```
docker update CONTAINER
```

Docker attach

```
docker attach CONTAINER
```

hängt Input-, Output-, und Errorstreams der Konsole an **CONTAINER**.

Docker export

Container-Dateisysteme werden als **.tar**-Datei exportiert mit

```
docker export CONTAINER
```

DATAMICS
machine intelligence consulting services

Docker Merkblatt-2/3

www.datamics.com

Docker File

Mit den Anweisungen in einem Docker-File werden Images erzeugt.

FROM

Mit FROM wird das zu verwendende Basisimage definiert, beispielsweise ein Image, welches Python3 bereitstellt:

```
FROM python:3-onbuild
```

EXPOSE

EXPOSE bestimmt, auf welchem Port die Anwendung erreichbar ist:

```
EXPOSE 5000
```

CMD

Mittels CMD werden in dem Image auszuführende Befehle angegeben, beispielsweise der Start einer Python3-Applikation:

```
CMD [”python3”, ”./app.py”]
```

RUN

RUN dient dem Ausführen von Befehlen auf dem Image-Dateisystem:

```
RUN apt-get update
```

ENV

Mit ENV werden Systemvariablen gesetzt:

```
ENV name=Brian
```

WORKDIR

WORKDIR bestimmt das Arbeitsverzeichnis des Images:

```
WORKDIR /Verzeichnis
```

USER

Mit USER wird der Benutzer für das Image ausgewählt:

```
USER terry
```

VOLUME

Der Befehl VOLUME erzeugt ein Volume für das Image:

```
VOLUME /var/www/html/app
```

LABEL

Mittels LABEL werden einem Image beliebige Metadaten, wie zum Beispiel die Versionsnummer, angehängt:

```
LABEL version=”1.0”
```

COPY

COPY kopiert Dateien in ein Image:

```
COPY test.txt /Verzeichnis/
```

Docker Compose

Docker Compose erlaubt es, die Services mehrerer Container miteinander zu verbinden, beispielsweise eine das Flask-Webframework in einem Container und eine Redis-Datenbank in einem zweiten. Dazu wird ein Compose File im **yml**-Format verwendet.

Compose File

Compose Files sind Versionsabhängig, weshalb in der ersten Zeile die Version angegeben werden sollte. Mit dem folgenden Compose File werden zwei Services bereitgestellt, ein Service mit Flask und ein Service mit Redis.

```
version: ’3’

services:
  app:
    build: .
    image: takacsmark/flask-redis:1.0
    environment:
      - FLASK_ENV=development
    ports:
      - 5000:5000
    redis:
      image: redis:4.0.11-alpine
```

ports mappt hierbei den Container-Port 5000 auf den Port 5000 des Hosts. Zusätzlich wird eine Umgebungsvariable **FLASK_ENV** erstellt.

Compose ausführen

Ausgeführt wird der Compose File mit

```
docker-compose up
```

Applikation testen

Speichert einen Parameter in der Flask-Applikation:

```
curl –header ”Content-Type:
application/json”\
–request POST \
–data ’{”name”:”John”}’ \
localhost:5000
```

Folgendermaßen wird der Parameter wieder abgerufen:

```
curl localhost:5000
```

Python Anbindung

Hiermit wird auf den Redis-Service aus einem Python-Script zugegriffen:

```
redis = Redis(host=”redis”, db=0,
  db=0, socket_timeout=5,
  charset=”utf-8”,
  decode_responses=True)
```

Docker Swarm

Mit Docker Swarm werden Container in einem Computercluster kontrolliert. Dabei werden Nodes in Manager zur Verwaltung und Worker für das eigentliche Ausführen der Aufgaben unterteilt.

Schwarm initialisieren

Starte den Schwarm:

```
docker swarm init
```

Worker hinzufügen

Um den Schwarm Manager- oder Worker-Nodes hinzuzufügen, sind Schwarm-spezifische Token erforderlich:

```
docker swarm join-token manager
docker swarm join-token worker
```

Gibt die Befehle zum Hinzufügen von Manager- oder Worker-Nodes mit einem Schwarm-spezifischen Token **XXX** aus:

```
docker swarm join –token XXX
```

Führe den Befehl zum Hinzufügen von Workern auf den anderen Computern im Cluster aus.

Schwarm anzeigen

Zeigt alle Nodes im Netz auf dem Manager-Computer an:

```
docker node ls
```

Node entfernen

Manager-Nodes entfernt andere Nodes aus dem Schwarm mittels der Node-ID:

```
docker node rm ID
```

Laufende Container anzeigen

Zeigt alle laufenden Container des Nodes an:

```
docker node ps
```

Service starten

Startet einen Service (hier Apache als Beispiel) auf einem Node:

```
docker service create httpd
```

Service beenden

Beendet einen Service auf einem Node:

```
docker service rm httpd
```

Service skalieren

Mit einem Schwarm können Services beliebig skaliert werden. Mit folgendem Befehl wird Apache auf zwei Nodes ausgeführt:

```
docker service skale httpd=2
```

Routing Mash

Ein Routing Mash stellt einen Service für alle Nodes im Schwarm bereit, hier auf Port 8080:

```
docker service create –name testhttpd \
  –publish published=8080,target=80 \
  httpd
```

Docker run

Docker run stellt vielfältige Parametrisierungen zur Verfügung.

Detached

Das Flag **-d** bewirkt, dass ein Docker-Prozess im Hintergrund gestartet wird:

```
docker run -d image service nginx start
```

PID

Das Flag **–cidfile** schreibt die Prozess-ID eines Images in eine Datei:

```
–cidfile=”dateiname”
```

Tag

Wird ein Tag an den Imagenamen angehängt, so wird die spezifizierte Version ausgeführt:

```
docker run ubuntu:14.04
```

IPC

Ein Container kann entweder über einen eigenen Namespace verfügen oder diesen mit anderen Containern teilen. Dies wird mit dem IPC Flag bestimmt:

```
–ipc=”MODUS”
```

Dabei kann **MODUS** unter anderem **none**, **private** oder **shareable** sein.

Netzwerk

run erlaubt eine vielfältige Netzwerkparametrisierung, u.a.:

```
–dns=[]
–add-host=
–ip=
–network=”MODUS”
```

Dabei kann **MODUS** unter anderem **none**, **bridge**, **container** oder **host** sein oder zu einem manuell definierten Netzwerk verbinden.

Neustartverhalten

Über den **–restart**-Flag wird das Verhalten für Neustarts bestimmt:

```
docker run –restart=always redis
docker run –restart=on-failure:10 redis
```

Der erste Befehl startet einen Redis-Container jedesmal neu, wenn der Container stoppt, der zweite Befehl nur zehn mal.

DATAMICS
machine intelligence consulting services

Docker Merkblatt-3/3

www.datamics.com

Glossar

Docker Daemon

Programm im Hintergrund, welches das bauen, starten und verteilen von Docker-Containern kontrolliert.

Docker Client

Kommandozeilenwerkzeug, durch das der Benutzer mit dem Docker Daemon interagieren kann.

Bind Mount

Ein Bind Mount ist ein einem laufenden Container zugeordneter, persistenter Speicherort im Dateisystem des Computers. Bind Mounts sind Betriebssystemabhängig.

Volume

Ein Volume ist ein persistenter Datenspeicher für die von Containern gespeicherte Daten. Sie können von verschiedenen Containern parallel genutzt werden und sind Betriebssystemunabhängig.

Registrierung

Eine Sammlung von Images. Der Docker-Hub ist die größte Sammlung öffentlich verfügbarer Images. Viele Unternehmen, die mit Docker arbeiten, haben private verfügbare Registrierungen für selbst entwickelte Images.

Host (etwa: Wirt)

In Bezug auf Docker ist ein Host ein Computer, der Docker-Container ausführt. Docker unterstützt die Betriebssysteme Windows, macOS und Linux.

Cluster (etwa: Schwarm)

Ein Cluster bezeichnet eine Sammlung von Docker-Hosts, die zusammen einen einzelnen, virtuellen Host darstellen. Dies dient der Skalierung: weitere Instanzen der Anwendungen werden als Hosts dem virtuellen Host hinzugefügt. Dienste hierzu sind unter anderem Kubernetes, Docker Swarm und Azure Service Fabric.

Node (Knoten)

Eine Instanz eines Docker-Swarm, die einen Container ausführt.

Stack

Stack bezeichnet die Menge der von einem Schwarm bereitgestellten Services.

Glossar

Image (etwa: Abbild)

Ein Image enthält alle erforderlichen Abhängigkeiten und Informationen zum Erstellen eines Containers. Ebenfalls beinhaltet ist die von einer Containerruntime verwendete Konfiguration. Üblicherweise besteht ein Image aus mehreren, in Schichten organisierten Sub-Images, die das Dateisystem des Containers darstellen. Images können multiple Architekturen unterstützen, indem sie bei ihrer Erzeugung Sub-Images für die Rechnerarchitektur benötigte Versionen von Sub-Images anfordert. Nach dem Erstellen kann ein Image nicht mehr geändert werden.

Container

Container bezeichnet eine Instanz eines Docker-Images. Nach Start eines Containers stellt dieser einen Dienst dem System zur Verfügung. Zum Skalieren dieses Dienstes müssen mehrere Instanzen eines Containers gestartet werden, entweder manuell oder automatisiert mit einem Batchauftrag, der jeder Instanz eigene Startparameter übergibt.

Docker-Datei (Docker-File)

Die Anweisungen zum Erstellen eines Docker-Images werden in einer Docker-Datei zusammengefasst. Die erste Zeile bezeichnet das Basisimage, die folgenden Zeilen beinhalten Anweisungen zur Installation der benötigten Programme und Dateien, bis alle erforderlichen Funktionalitäten vorhanden sind.

Build (Aufbau)

Mit Build wird ein Containerimage basierend auf der Docker-Datei und zusätzlich benötigter Dateien aufgebaut. Der Befehl hierzu ist **docker build**.

Mehrstufige Builds

Seit Docker 17.05 kann mit mehrstufigen Builds die Größe des finalen Images verringert werden. Beispielsweise kann so ein großes Basisimage mit komplettem SDK (software development kit) verwendet werden, um die Anwendung zu kompilieren. Im finalen Image wird dann statt dessen jedoch ein kleines Basisimage, das lediglich die Laufzeitumgebung (Runtime Environment) enthält.

Glossar

Repositoryname (Repo)

Der Repositoryname ist Auflistung der verwendeten Docker-Images mit einem Tag, das die Version des Images angibt. Dabei kann ein Repository mehrere Varianten eines Images enthalten, wie zum beispiel SDK- und Runtime-Varianten oder Images für Linux und Windows.

Azure Container Registry

Registrierung von Docker-Images zur Arbeit mit MicroSoft Azure, welche die Verwendung von Azure Active Directory-Gruppen und -Berechtigungen gestattet.

Docker-Hub

Docker-Hub ist die größte öffentliche Registrierung von Images und im Besitz des Unternehmens Docker. Dabei bietet Docker-Hub unter anderem Docker-Image-Hosting, öffentliche oder private Registrierungen, Buldtrigger, Web-Hooks und Integration mit GitHub.

Docker Trusted Registry (DTR)

Eine im Produkt Docker Datacenter enthaltene Registrierung zur Verwaltung privater Images im lokalen Netzwerk.

Docker Community Edition (CE)

Softwarepacket zum Erzeugen, Ausführen und Testen von Containern für Windows und macOS. Linux-Container können mit Docker CE mit der Hyper-V-VM (Virtuelle Maschine) entwickelt werden. Die macOS-Variante basiert auf dem Hypervisor-Framework und dem Xhyve-Hypervisor. Docker CE ist der Nachfolger der Docker Toolbox.

Docker Enterprise Edition (EE)

Auf Unternehmen ausgerichtete Version von Docker für die Linux- und Windows-Entwicklung.

Tag (Bezeichner)

Vom Nutzer gewählter Bezeichner eines Images, die verwendet wird, um Images oder Image-Versionen unabhängig von Versionsnummer oder Zielumgebung identifizieren zu können.

Glossar

Compose (etwa: komponieren)

Ein Kommandozeilenwerkzeug, mit dem basierend auf Konfigurationsdateien im YAML-Dateiformat und weiteren Metadaten Anwendungen mit mehreren Containern ausgeführt werden können. Dabei wird eine einzelne Anwendungen mit ein oder mehreren YAML-Konfigurationsdateien und allen zugehörigen Images definiert, die dann alle mit dem Befehl **docker-compose-up** konvertiert werden.

Orchestrator

Ein Orchestrator ist ein Werkzeug zur Vereinfachung der Verwaltung von Clustern und Docker-Hosts. Images, Container und Hosts werden über die Kommandozeile oder eine graphische Schnittstelle verwaltet. Dies erlaubt die Kontrolle von Containernetzwerken, des Gerätestatus, des Lastenausgleichs sowie der Container- und Hostconfiguration. Orchestratoren werden beispielsweise von Kubernetes und Azure Service Fabric angeboten.

Layer (Ebene)

Ein Layer bezeichnet über ein Image gelegte Modifikationen. Wird ein Image erneut aufgebaut, werden nur geänderte Layer aktualisiert.

Service (Dienst)

Ein Service bezeichnet den Umfang eines Docker-Swarm bezüglich eines einzelnen Images.

Virtuelle Maschine (Virtual machine)

Eine virtuelle Maschine simuliert eine Laufzeitumgebung in einem nicht-nativen System. Container sind eine leichtgewichtige Form virtueller Machinen, die nur die absolut nötigsten Betriebssystemsoperationen bereitstellen.

Base-Image (Basis Image)

Grundlegendes, unterstes Image eines Containers. Wird beispielsweise ein Container für eine Python-Applikation entworfen, so stellt das Base Image mindestens den benötigten Python-Interpreter zur Verfügung.