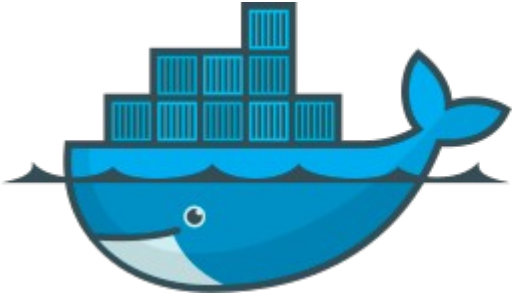
BGD2

Docker: Befehle, Storage, Networking



Andreas Scheibenpflug

Befehle zu Images

Build

- `docker build <path>`
- Baut ein Image aus einem Dockerfile und speichert dieses lokal
- Optionen
  - `-t / --tag`: Vergibt einen Tag, z.B.
  - `docker build -t myimage:latest .`

Images

- `docker images`
- `docker image ls`
- Listet alle lokalen Docker Images
- z.B. gespeichert in `/var/lib/docker/image/`
- Optionen:
  - `-a / --all`: Auch intermediate images

Save

- `docker [image] save image > imageFile.tar`
- `docker [image] save image -o imageFile.tar`
- Speichert ein Image in ein tar Archive

Load

- `docker [image] load < image.tar`
- `docker [image] load -i image.tar`
- Lädt ein Image aus einem tar Archive

Rmi

- `docker rmi <image>`
- `docker image rm <image>`
- Löscht ein Image

Pull und Push

- `docker image pull <[url/]tag>`
  - Lädt ein Image aus einem Repository
- `docker image push <[url/]tag>`
  - Lädt ein Image in ein Repository

Befehle zu Containern

Run (1/2)

- `docker run <tag> [command] [args]`
- Führt ein Image und, optional, einen Befehl im Container aus
- Erzeugt den Writable Layer und startet den Container
- Optionen:
  - `-i/--interactive`: Lässt STDIN offen
  - `-a`: Öffnet STDIN/STDOUT/STDERR
  - `-t/--tty`: Öffnet eine Shell für Ein- und Ausgabe
  - `-v/--volume`: Mounted Verzeichnisse/Dateien/Volumes

Run (2/2)

- Optionen:
  - `-p/--publish`: Gibt Ports eines Containers zum Hostsystem frei
  - `-d/--detach`: Führt den Container im Hintergrund aus
  - `-env`: Setzt Umgebungsvariablen für den Container
  - `--name`: Setzt einen Namen, der bei anderen Befehlen verwendet werden kann (start, stop,…)
  - `--init`: Startet ein Init im Container mit PID 1
- Viele weitere Optionen zur Einschränkung von Ressourcen

Container create

- Erstellt den Writable Layer des Containers für ein Image
- Kann mit `docker start` gestartet werden
- Optionen: siehe `docker run`

Start/Stop

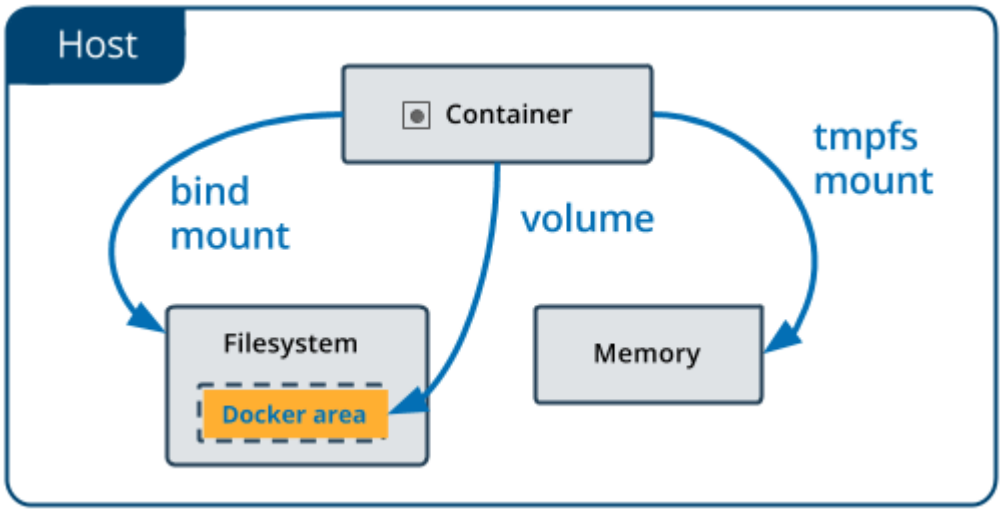
- `docker stop <container>`
  - Stoppt einen laufenden Container
  - Sendet SIGTERM zu den Prozessen im Container, nach einem Timeout SIGKILL
- `docker kill <container>`
  - Sendet SIGKILL an alle Prozesse im Container
- `docker start <container>`
  - Startet einen zuvor gestoppten Container
- `docker pause <container>`
  - Pausiert Prozesse in Container (→ cgroups freezer)
- `docker unpause <container>`

Container cp

- Erlaubt das Kopieren von Dateien aus dem Container in das Dateisystem des Hostsystems
- `docker [container] cp <container>:src dest`
- Beispiel
  - `docker cp container1:/root/notes.txt notes.txt`

Mounts

- Erlaubt
  - Persistieren von Daten (Volumes)
  - Zugriff/Austausch von Daten mit dem Hostsystem (Bind Mount)
  - Temporären Storage (tmpfs)



https://docs.docker.com/storage/volumes/

Volumes

- Volumes werden durch Docker verwaltet
- Werden in Container gemountet
- Auf Volumes geschriebene Daten sind persistent über Container Restarts
- Format: `volumename:containerdir[:ro]`
- Beispiel:
  - `docker volume create myvolume`
  - `docker run -v myvolume:/root …`

Befehle zu Volumes

- `docker volume`
  - `create <name>`: Erstellen eines benannten Volumes
  - `rm <name>`: Löschen eines Volumes
  - `ls`: Anzeigen der Volumes auf dem System
  - `Prune`: Löschen aller gerade nicht verwendeten Volumes

Bind Mounts

- Mounten von Dateien oder Verzeichnissen des Hostsystems in einen Container
- Format: `hostdir:containerdir[:ro]`
- Format: `hostfile:containerfile[:ro]`
- Beispiel:
  - `docker run -v $PWD/note.txt:/root/note.txt …`
- Das Hostverzeichnis/datei muss ein absoluter Pfad sein
- Anstelle -v kann auch --mount verwendet werden:
  - `docker run --mount type=bind,source=$PWD/note.txt,target=/root/note.txt ...`

Tmpfs

- Erlaubt das Mounten von einem In-Memory Dateisystem außerhalb des Writable Layers des Containers
- Beim Stoppen des Containers gehen die Daten im tmpfs verloren
- tmpfs mounts können nicht zwischen Containern geteilt werden
- tmpfs mounts funktionieren nur unter Linux
- Beispiel:
  - `docker run --tmpfs /logs ...`

Ports

- Freigabe von Ports des Containers zum Hostsystem
- Zugriff auf an Ports gebundene Anwendungen im Container
- Format: `hostport:containerport`
- Beispiel:
  - `docker run -p 1234:80 ...`

Ps

- `docker ps`
- Listet alle aktuell laufenden Container
- Optionen:
  - `-a`: alle Container, auch jene die gerade nicht laufen
  - `-l`: Zeigt den zuletzt gestarteten Container an
  - `-s`: Größe des Containers
    - Size: Größe des letzten Dateisystem Layers (Writable Layer)
    - Virtual: Größe des Images + Size

Attach/Exec

- `docker attach <container>`
  - Verbindet sich zu STDIN, STDOUT und STDERR des Containers
- `docker exec <container> CMD [ARG…]`
  - Führt einen Befehl in einem laufenden Container aus

Top / Stats

- `docker top <container>`
  - Zeigt die im Container laufenden Prozesse
- `docker stats <container>`
  - Zeigt Ressourcenverbrauch eines Containers
  - Festplattenspeicher, CPU, Memory, Network

Networking

- Mehrere Optionen für Networking, erweiterbar durch Plugins
- Default: Bridge
  - Standard Netzwerk für Container wenn nicht anders spezifiziert
  - Keine offenen Ports
    - Zwischen Containern
    - Zwischen Container und Hostsystem
  - Keine Auflösung von Hostnamen zwischen Containern
    - Nur über IP Adressen Zugriff möglich

User-defined Bridges

- Zuordnung von Containern zu einem Netzwerk
- Die Ports zwischen den Containern sind alle offen
- Ports zum Hostsystem müssen explizit freigegeben werden
- DNS ist verfügbar (Container Name == DNS Name)
- Container können zur Laufzeit zu einer User-defined Bridge hinzugefügt und entfernt werden

Network

- `docker network create <network>`
  - Erzeugt ein neues Netzwerk (default: Bridge)
- `docker network [dis]connect <network> <container>`
  - Verbindet oder entfernt Container mit/von Netzwerk
- `docker network rm, ls, prune`
  - Analog zu Volumes
- `docker run --net <network> …`
  - Startet einen Container in dem angegebenen Netzwerk