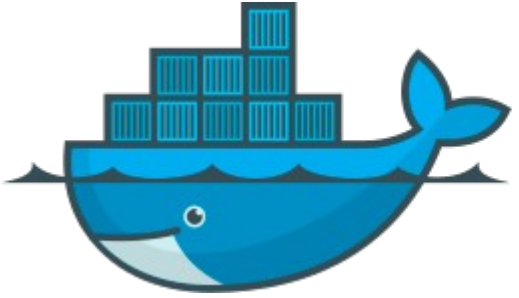
BGD2

Docker Compose



Andreas Scheibenpflug

Docker Compose

- Erlaubt die Definition, Erstellen und Ausführen von Multi-Container Anwendungen
- Definition über YAML Datei
- Definition Ports, Networks, Volumes,… als Konfiguration anstatt Argument für docker Befehl

Aufbau

```
version: '3'
services:
  service1:
    ...
  service2:
    ...

volumes:
  ...
networks:
  ...
```

- Ein Service entspricht einem Dockerfile/Image
- Unter jedem Service Eintrag findet sich die Konfiguration des Services (entspricht den Parametern von docker run/create)
- In den Abschnitten volumes und networks werden Konfigurationen für diese vorgenommen

Build

- Gibt den Pfad des zu bauenden Dockerfiles an
- Context ist ein Pfad oder ein git Repository
- Beispiele (zwei Formen):

```
build: ./dir

build:
      context: ./dir
      dockerfile: OtherDockerfile
      args:
        arg1: val1
```

Image

- Gibt ein Image für ein Service an
- Ist dieses nicht lokal vorhanden, wird es von einer Registry geladen
- Ist build spezifiziert, wird das Image gebaut
- Beispiel:
  - `image: ubuntu:18.10`

Environment

- Setzen von Umgebungsvariablen (`docker run -e / --env`)
- Beispiel:

```
environment:
  - DOTNET_VERSION=1.2
```

depends_on

- Ermöglicht die Definition von Abhängigkeiten zwischen Containern
- Ein Container, der ein `depends_on` definiert, wird erst nach den in `depends_on` angeführten Containern gestartet
- Beispiel

```
services:
    webapp:
        ...
        depends_on:
            - db
    db:
       ...
```

Command / Entrypoint

- Überschreibt `ENTRYPOINT` oder `CMD` aus dem Dockerfile
- Analog zu `docker run --entrypoint / docker run … [COMMAND]`
- Beispiel

```
services:
    db:
        command: --authentication=password
        …
```

Ports

- Freigabe von Ports des Containers zum Hostsystem
- Zugriff auf an Ports gebundene Anwendungen im Container
- Format: `hostport:containerport`
- Beispiel:

```
ports:
  - "8080:80"
```

Networks

- Definiert an welches Netzwerk ein Container gebunden ist
- User-defined bridge → Ports zum Hostsystem müssen explizit freigegeben werden
- Beispiel:

```
services:
  myservice:
    networks:
      - mynet

networks:
  - mynet:
```

Volumes

- Definiert Volumes für ein Service
- Namen von Volumes müssen in einem eigenen `volumes` Eintrag angeführt werden
- Konfiguration der Volumes erfolgt innerhalb eines Service Eintrags
- Beispiel:

```
volumes:
- myvolume:/root


volumes:
  myvolume:
```

```
volumes:

  myvolume:

     name: extVolume
```

Volumes – Beispiel Kurzform

```
services:
  myservice:
    volumes:
    - /tmp/data:/var/opt/data
    - datavolume:/var/opt/data
    - ./data:/var/opt/data
volumes:
  datavolume:
```

Volumes – Beispiel Langform

```
services:
  myservice:
    volumes:
     - type: volume
       source: datavolume
        target: /data
     - type: bind
       source: ./data
       target:  /var/opt/data
volumes:
  datavolume:
```

docker-compose

- `docker-compose build`
  - Baut alle Images (führt build Schritte in yaml aus)
- `docker-compose up`
  - Baut und startet alle Container (services)
  - `-d`: Führt Container im Hintergrund aus
- `docker-compose down`
  - Stoppt alle Container

docker-compose up

- Up erzeugt die Container beim ersten Start (`docker run`)
- Sind Image und Container vorhanden und unverändert, werden diese nur gestartet (`docker start`)
- Bei `start` bleiben Dateien in Containern erhalten
  - Kann zu schwer zu findenden Problemen bei Compose Dateien mit mehreren Services führen
- `--force-recreate`: Erzeugt Container immer neu

Beispiel

```
version: '3'
services:
  db:
    container_name: mydb
    image: postgres
    volumes:
      - ./tmp/db:/var/lib/postgresql/data
  web:
    build: .
    command: bundle exec rails s -p 3000 -b '0.0.0.0'
    volumes:
      - .:/myapp
    ports:
      - "3000:3000"
    depends_on:
      - db
```

https://docs.docker.com/compose/rails