Inhaltsverzeichnis

Orchestrierung einer GDI über Docker

Herzlich Willkommen beim **Orchestrierung einer GDI über Docker** Workshop auf der FOSSGIS 2020 in Freiburg.

Dieser Workshop wurde für die Verwendung auf der OSGeo-Live 13.0 DVD entwickelt und soll Ihnen einen ersten Einblick in docker als Orchestrierungstool einer Geodateninfrastruktur (GDI) geben.

> Der Workshop kann hier als PDF-Version heruntergeladen werden.

Bitte stellen Sie sicher, dass Sie die Schritte der Vorarbeiten und generelle Informationen-Seite ausgeführt haben, um einen reibungslosen Ablauf zu gewährleisten.

Der Workshop ist aus einer Reihe von Modulen zusammengestellt. In jedem Modul werden Sie eine Reihe von Aufgaben lösen, um ein bestimmtes Ziel zu erreichen. Jedes Modul baut Ihre Wissensbasis iterativ auf.

Die folgenden Module werden in diesem Workshop behandelt:

- Vorarbeiten und generelle Informationen Grundlegende Informationen zur Workshop-Umgebung (OSGeoLive, Pfade, URLs, Credentials).
- Grundlagen Docker Grundlagenwissen zu Docker.
- Grundlagen docker-compose Grundlagenwissen zu docker-compose.
- Beispiel-GDI Praktischer Aufbau einer exemplarischen GDI mit Docker und docker-compose.

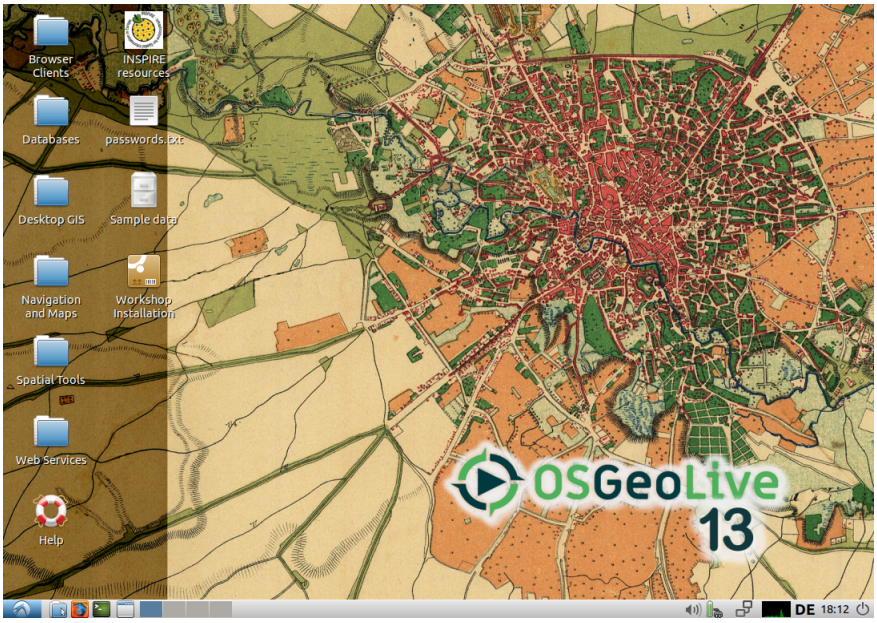
Autoren

- Jan Suleiman (suleiman@terrestris.de)
- Daniel Koch (koch@terrestris.de)

Vorarbeiten und generelle Informationen

Bevor wir mit dem Workshop starten können, führen Sie bitte die folgenden Schritte aus:

- Rechner mit OSGeoLive-Medium hochfahren
- Sprache auswählen (Deutsch für korrekte Tastaturbelegung)
- *Lubuntu ohne Installation ausprobieren* auswählen
- Benutzer: user; Passwort: user (wird vermutlich nicht benötigt)



Die Startansicht der OSGeo Live 13.0 auf Ihrem Rechner.

Installation docker/docker-compose

Bitte überprüfen Sie, ob `docker` und `docker-compose` korrekt installiert sind, indem Sie das Terminal öffnen und die Eingabe von

```
docker
```

die folgende Ausgabe (Auszug) erzeugt:

```
Usage:    docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

(…)
```

Prüfen Sie ebenfalls, ob die Eingabe von `docker-compose` die folgende Ausgabe (Auszug) erzeugt:

```
Define and run multi-container applications with Docker.

(…)
```

Schlägt einer der obigen Befehle fehl, führen Sie bitte die folgenden Befehle (als `root`) aus:

```
apt update
apt install docker.io docker-compose
usermod -aG docker $USER
newgrp docker
```

Im folgenden Abschnitt werden wir mit Docker-Basiswissen fortfahren.

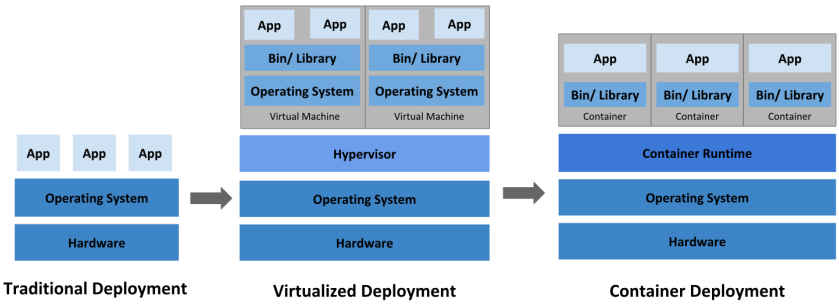
Basiswissen

Dieses Kapitel umfasst Erläuterungen der Grundkonzepte von Docker und docker-compose.

Docker

Docker/Containerisierung vs. Virtualisierung

- Virtuelle Maschinen (VM):
  - Isolation von Hardware
  - Jede VM hat ihren eigenen Kernel
  - Großer Overhead (OS, Treiber, ...)
- Containerisierung:
  - Isolation von Software
  - Mehrere Container teilen sich den Kernel und andere Ressourcen des Host-Systems
  - Leichtgewichtiger als VMs



https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Übersicht Docker

- Docker ist Freie Software (https://github.com/docker/docker-ce)
- Geschrieben in Go
- Isolierung von Software durch Containervirtualisierung
- Einfache und schnelle Bereitstellung von Anwendungen unabhängig vom Host-System
- Unterstützt Modularität (Microservices)
- Versionskontrolle/Rollbacks

Zentrale Docker Begriffe

- **Dockerfile**
  - Textdatei mit Schritt-für-Schritt "Bauanleitung" für Docker-Images
- **Image**
  - Endprodukt des Bauens eines Dockerfiles, "Speicherabbild"
- **Container**

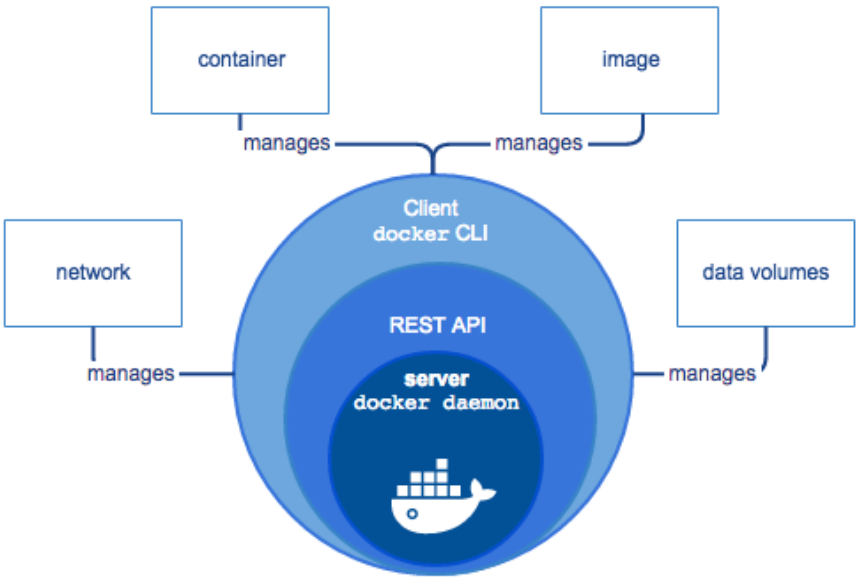
    - Konkret (laufende) Instanz eines Images
- **Registry**
    - Privates oder öffentliches Repository für Images, etwa https://hub.docker.com/

Zentrale Docker Befehle

Häufige Docker Befehle für die Kommandozeile:

- `docker help`
    - docker help [command], z.B. `docker help build`
- `docker build`
    - zur Erzeugung eines Images aus einem Dockerfile
- `docker run`
    - zum Starten eines Containers auf Basis eines Images
- `docker log`
    - zum Einsehen der Logs eines laufenden Containers
- `docker stop`
    - zum Stoppen eines laufenden Containers
- `docker ps`
    - listet alle aktuell laufenden Container
- `docker exec`
    - ermöglicht die Ausführung von Befehlen in einem Container
- `docker images`
    - listet alle lokalen Images
- `docker pull`
    - lädt ein Image aus einer Registry
- `docker push`
    - lädt ein lokales Image in eine Registry

Architektur Docker Engine



https://jaxenter.de/einfuehrung-docker-tutorial-container-61528

Aufgaben:

- Öffnen Sie ein Terminalfenster und führen Sie folgenden Befehl aus:

```
docker run hello-world
```

- Versuchen Sie die einzelnen Schritte der Ausgabe dieses Befehls zu erläutern.

Dockerfile

Das Dockerfile beschreibt durch die Auflistung von Befehlen der Form [BEFEHL] [parameter] den Aufbau des Images, das aus ihm erzeugt wird.

Die wichtigsten Befehle in Dockerfiles:

- **`FROM`**
  - der erste Befehl und bestimmt das "Vater"-Image, das als Startpunkt dient
  - Beispiel: `FROM ubuntu:18.04`
- **`RUN`**
  - führt den übergebenen Parameter als
  - es können nur Befehle ausgeführt werden, die im Image möglich sind und somit vom Vater-Image abhängen
  - Beispiel: `RUN sudo apt-get install -y apache2`
- **`COPY (bzw. ADD)`**
  - kopiert lokale Dateien vom Host-System in das Image

    - Beispiel: `COPY local_image.jpg /opt/image.jpg`
- **`WORKDIR`**
    - wechselt im Image in das übergebene Verzeichnis
    - Beispiel: `WORKDIR /opt`
- **`CMD (bzw. ENTRYPOINT )`**
    - definiert den Standardbefehl, der später vom Container ausgeführt wird. Diesen Befehl kann es nur einmal pro Dockerfile geben (oder der letzte "gewinnt").
    - Beispiel: `CMD echo "Hello world"` oder `CMD myScript.sh`
- **`EXPOSE`**
    - dient der Dokumentation des Prozessports
    - Beispiel: `EXPOSE 8080`

Hinweis: RUN, COPY und ADD erzeugen immer einen neuen, eindeutigen "Layer". Das resultierende Image ist letzlich die (geordnete) Sammlung von zahlreichen Layern.

Beispiel

Dockerfile

```
FROM busybox

RUN echo "Hello World" > /fossgis.txt

RUN cat /fossgis.txt

CMD ["cat", "/fossgis.txt"]
```

Image bauen

```
docker build -t fossgis:1.0.0 .
```

Image starten

```
docker run --name fossgis-test fossgis:1.0.0
```

Aufgaben:

- Fügen Sie die obigen Inhalte der Beispiel-Dockerfile in eine neue Datei namens `Dockerfile` in einem beliebigen Verzeichnis ein, bauen Sie das Image und starten den Container.
- Welche Ausgabe erhalten Sie jeweils?

Best practices

- Reihenfolge der Befehle beachten, sie wirkt sich auf das Caching der Layer aus:

```
FROM debain

-COPY . /app

RUN apt-get update
RUN apt-get -y install openjdk-8-jdk ssh vim

+COPY . /app

CMD ["java", "-jar", "/app/target/app.jar"]
```

- Quellen möglichst explizit kopieren:

```
FROM debain

RUN apt-get update
RUN apt-get -y install openjdk-8-jdk ssh vim

-COPY . /app
+COPY target/app.jar /app

-CMD ["java", "-jar", "/app/target/app.jar"]
+CMD ["java", "-jar", "/app/app.jar"]
```

- `RUN` Befehle nach Möglichkeit bündeln:

```
FROM debain

-RUN apt-get update
-RUN apt-get -y install openjdk-8-jdk ssh vim
+RUN apt-get update && \
+    apt-get -y install \
+    openjdk-8-jdk \
+    ssh \
+    vim

COPY target/app.jar /app

CMD ["java", "-jar", "/app/app.jar"]
```

- Keine unnötigen Abhängigkeiten installieren:

```
FROM debain

-RUN apt-get update && \
-    apt-get -y install \
-    openjdk-8-jdk \
-    ssh \
-    vim
+RUN apt-get update && \
+    apt-get -y install --no-install-recommends \
+    openjdk-8-jdk

COPY target/app.jar /app

CMD ["java", "-jar", "/app/app.jar"]
```

- Paketmanager-Cache löschen:

```
FROM debain

-RUN apt-get update && \
-    apt-get -y install --no-install-recommends \
-    openjdk-8-jdk
+RUN apt-get update && \
+    apt-get -y install --no-install-recommends \
+    openjdk-8-jdk && \
+    rm -rf /var/lib/apt/lists/*

COPY target/app.jar /app

CMD ["java", "-jar", "/app/app.jar"]
```

- Nach Möglichkeit offizielle Images benutzen:

```
-FROM debain

-RUN apt-get update && \
-    apt-get -y install --no-install-recommends \
-    openjdk-8-jdk && \
-    rm -rf /var/lib/apt/lists/*

+FROM openjdk

COPY target/app.jar /app

CMD ["java", "-jar", "/app/app.jar"]
```

- Möglichst spezifische Tags nutzen:

```
-FROM openjdk
+FROM openjdk:8

COPY target/app.jar /app

CMD ["java", "-jar", "/app/app.jar"]
```

- Möglichst kleine Basisimages nutzen, die kompatibel sind:

```
REPOSITORY   TAG            SIZE
openjdk      8              624MB
openjdk      8-jre          443MB
openjdk      8-jre-slim     443MB
openjdk      8-jre-alpine   443MB
```

- Multi-Stage Builds verwenden:

```
FROM maven:3.6-jdk-8-alpine AS builder

WORKDIR /app

COPY pom.xml .

RUN mvn -e -B dependency:resolve

COPY src ./src

RUN mvn -e -B package

-CMD ["java", "-jar", "/app/app.jar"]

+FROM openjdk:8-jre-alpine

+COPY --from=builder /app/target/app.jar /

+CMD ["java", "-jar", "/app/app.jar"]
```

(https://www.docker.com/blog/intro-guide-to-dockerfile-best-practices/)

docker-compose

`docker-compose` bietet die Möglichkeit, verschiedene Dienste (Container) in einer YAML-Datei zu definieren (docker-compose.yml). Dies bietet den Vorteil, mehrere Container auf eine übersichtliche Art und Weise konfigurieren und orchestrieren zu können.

> Die Dokumentation zu `docker-compose` findet sich auf https://docs.docker.com/compose/.

`docker-compose` wird überlicherweise genutzt um Entwicklungsumgebungen aufzusetzen, automatisiertes Testen zu ermöglichen oder eben auch um GDIs mit geringem Aufwand zusammenzustellen. Ein großer Vorteil von `docker-compose` ist das gleichzeitige starten mehrerer Container mit nur einem Befehl. Zusätzlich können wir auch Abhängigkeiten zwischen den verschiedenen Containern definieren. Dazu später mehr.

In einer `docker-compose.yml` können sowohl lokale Dockerfiles, als auch veröffentlichte Images direkt eingebunden werden.

Dies sieht beispielsweise folgendermaßen aus:

```
version: '3',
services:
  my-local-dockerfile:
    build:
      context: relative-directory-containing-the-dockerfile/
  local-or-remote-image:
    image: image-name:version
```

Wichtige Konfigurations-Parameter

- **image**
  - Image, auf Basis dessen der Container gestartet werden soll
- **ports**
  - Ports, die von außerhalb der Container zugreifbar sein sollen. Syntax: `hostMachinePort:containerPort` (z.B. `8080:80` stellt Port 80 des Containers auf Port 8080 der Host Maschine frei)
- **environment**
  - Umgebungsvariablen die dem Container mitgegeben werden sollen (bspw. Nutzername und Passwort)
- **volumes**
  - Pfade, die von der Host Maschine in den Container eingehangen werden sollen
- **context**
  - Pfad zu einer Dockerfile, welche zum Erstellen des Images genutzt werden soll

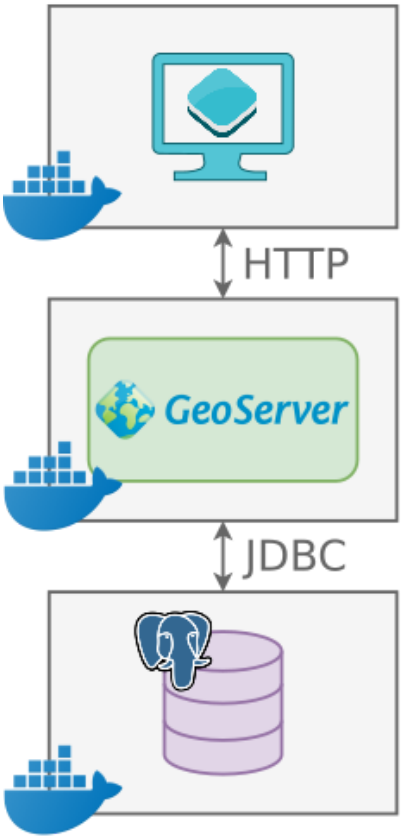
Wichtige Befehle

- **`docker-compose help`**
  - **`docker-compose help [command]`**, z.B. **`docker-compose help build`**
- **`docker-compose build`**
  - zur Erzeugung der Services, die in der Datei `docker-compose.yaml` definitiert sind
- **`docker-compose up`**
  - zum Starten der Services/Container
  - Mit dem Parameter **`-f`** lassen sich auch Dateien angeben, die nicht `docker-compose.yaml` heißen (z.B. unterschiedliche Konfiguration für verschiedene Umgebungen)
- **`docker-compose down`**
  - zum Stoppen der Services/Container
- **`docker-compose logs [service…]`**
  - zum Einsehen der Logs eines (oder mehrerer) Services/Container
- **`docker-compose restart [service…]`**
  - zum Neustarten eines (oder mehrerer) Services/Container

Beispiel-GDI

In diesem Kapitel werden wir eine exemplarische GDI bestehend aus den folgenden Komponenten aufbauen:

- PostGIS Datenbank zur Persistierung der Geodaten.
- GeoServer zur Veröffentlichung der Geodaten über OGC-Dienste.
- OpenLayers Applikation zur Anzeige der Geodaten über den Browser.



Zielarchitektur der GDI

Aufgaben

PostGIS-Service

- Legen Sie eine neue Datei namens `docker-compose.yml` in einem beliebigen Verzeichnis an.
- Fügen Sie dieser Datei einen neuen Service `fossgis-postgis` basierend auf dem `mdillon/postgis` Image in Version `11-alpine` hinzu.
  - Achten Sie beim Anlegen des Services auf das korrekte Weiterleiten des internen Ports (5432) auf den Host (5433) und legen Sie einen

    User mit den Zugangsdaten `fossgis:fossgis` an.
    - Mounten Sie das Datenverzeichnis der Datenbank (`/var/lib/postgresql/data`) auf das Hostsystem.
- Starten Sie den Service (über `docker-compose`) und verbinden Sie sich z.B. über `pgAdmin` mit dem Datenbank-Server und der Datenbank `fossgis` .
- Importieren Sie die Stadtgebiete Freiburgs (siehe `stadtteile.sql` aus Materialien) in die Datenbank.

GeoServer-Service

- Erweitern Sie die `docker-compose.yml` durch den Service `fossgis-geoserver` und nutzen Sie dabei das `terrestris/geoserver:2.15.2` Image.
    - Achten Sie auch hier auf das korrekte Mappen des internen Ports (8080) auf den Host (8080).
    - Mounten Sie das Datenverzeichnis des GeoServers (`/opt/geoserver_data`) auf das Hostsystem.
    - Bestimmen Sie zusätzlich die Startreihenfolge der Services mittels `depends_on` :
        1. `fossgis-postgis`
        2. `fossgis-geoserver`
- Stoppen Sie, falls noch nicht geschehen den bisherigen Service und starten Sie das compose Netzwerk neu.
- Öffnen Sie den GeoServer über die Adresse http://localhost:8080/geoserver im Browser. Nutzen Sie als Anmeldedaten `admin:geoserver` .
- Legen Sie einen neuen Arbeitsbereich `FOSSGIS` an.
- Legen Sie einen neuen Datenspeicher `POSTGIS` an. Wählen Sie dabei die Verbindungsparameter des `fossgis-postgis` Services an.
- Legen Sie anschließend einen neuen Layer `STADTTEILE` auf Basis des Datenspeichers `POSTGIS` und der Tabelle `stadtteile` an.
- Optional: Nutzen Sie den Stil `stadtteile.sld` der Materialien und weisen Sie diesen dem Layer zu.

nginx-Service (OpenLayers Anwendung)

- Erstellen Sie auf Ebene der `docker-compose.yml` ein neues Verzeichnis `fossgis-nginx` und dort eine neue Datei `Dockerfile` .
- Legen Sie das `client` -Verzeichnis sowie die `default.conf` der Materialien neben der `Dockerfile` ab.
- Öffnen Sie die `Dockerfile` und:
    - Wählen Sie als Basisimage die aktuelle Version des offiziellen nginx Images aus.
    - Kopieren Sie die Konfigurationsdatei `nginx.conf` in das Image und wählen Sie als Zielpfad `/etc/nginx/conf.d/default.conf` .
    - Kopieren Sie den Inhalt des entpackten Client-Archivs `client` in das Image und wählen Sie als Zielpfad `/etc/nginx/html` .
    - Geben Sie den Port (80) des nginx-Prozesses in der `Dockerfile` an.
- Fügen Sie der `docker-compose.yml` einen neuen Service `fossgis-nginx` hinzu.
    - Veröffentlichen Sie den Service-Port 80 auf dem 8000er Port des Hosts und wählen Sie als Build-Context die zuvor erstellte `Dockerfile` .

    - Achten Sie bei der Startreihenfolge darauf, dass der `nginx` Service zuletzt gestartet wird.
- Starten Sie anschließend alle Services neu und öffnen Sie http://localhost:8000 im Browser.

Bonus

- Verbinden Sie sich über das Terminal mit dem laufenden nginx-Container und ändern Sie den Wert des `<title>` Elements der `index.html` auf einen Wert Ihrer Wahl. Laden Sie anschließend die Applikation im Browser neu.
- Was passiert, wenn der Service neu gestartet wird? Was passiert, wenn der Container neu gebaut wird?