Inhaltsverzeichnis

Einleitung 0

Vorarbeiten und generelle Informationen 1

Basiswissen 2

- Docker 2.1
- docker-compose 2.2

Administrationsoberfläche 3

- Serverstatus 3.1
- Daten 3.2
- Einstellungen und Sicherheit 3.3

Datenveröffentlichung 4

- Vektordatenquellen 4.1
- Rasterdatenquellen 4.2

Gruppenlayer 5

Styling 6

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)
- Motivation
- Grundlagen Docker
- Grundlagen docker-compose
- Beispiel-GDI

Autoren

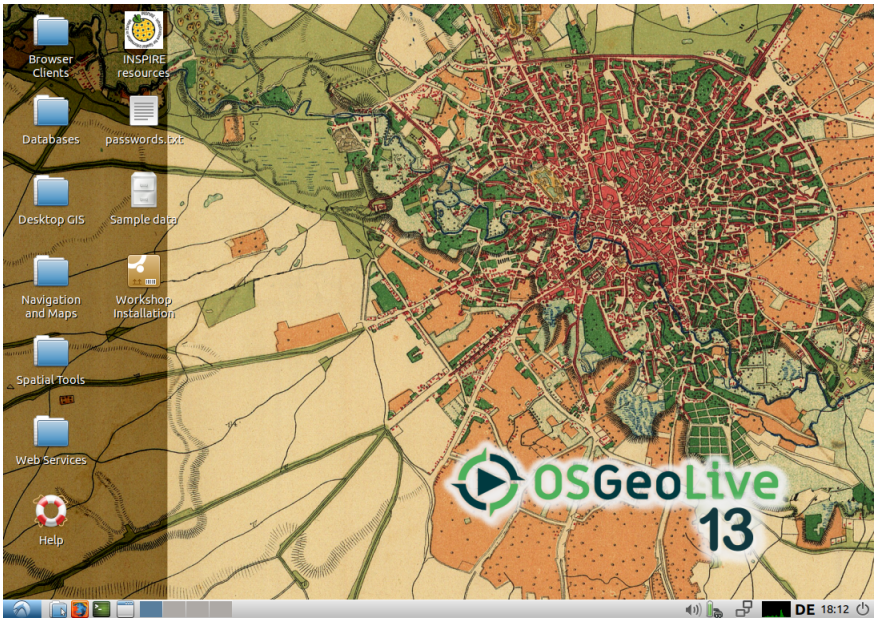
- Jan Suleimann (suleimann@terrestris.de)
- Daniel Koch (koch@terrestris.de)

(Die Autoren sind alphabetisch nach ihrem Nachnamen sortiert.)

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 aus:

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

Im folgenden Abschnitt werden wir mit Docker-Basiswissen fortfahren.

Docker

docker-compose

TODO

- Aufgaben:
  - `docker run hello-world` ausführen, Ausgabe analysieren

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:

TODO Beispiel vorher lokal erzeugtes Dockerfile und fertiges image erstellen

```
version: '3',
services:
  geoserver:
    build:
      context: geoserver
```

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

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