

Inhaltsverzeichnis

Einführung	1.1
Vorarbeiten und generelle Informationen	1.2
Basiswissen	1.3
Docker	1.3.1
Docker Compose	1.3.2
Aufbau einer Geodateninfrastruktur	1.4
Vorbereitung	1.4.1
PostGIS	1.4.2
GeoServer	1.4.3
OpenLayers über nginx	1.4.4
Musterlösung	1.4.5



Orchestrierung einer GDI über Docker

Herzlich Willkommen beim **Orchestrierung einer GDI über Docker** Workshop.

Dieser Workshop wurde für die Verwendung auf der [OSGeo-Live 16.0 DVD](#) entwickelt und soll Ihnen einen ersten Einblick in docker als Orchestrierungstool einer Geodateninfrastruktur (GDI) geben.

Information

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.
- [Aufbau einer Geodateninfrastruktur](#) Praktischer Aufbau einer exemplarischen GDI mit Docker und Docker Compose.

Autoren

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



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
```

```
(...)
```

Docker

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

```
Usage: docker compose [OPTIONS] COMMAND

Define and run multi-container applications with Docker.

(...)
```

Schlägt einer der obigen Befehle fehl, führen Sie bitte die folgenden Befehle aus:

```
sudo apt update
sudo apt install docker docker-compose-v2
sudo usermod -aG docker $USER
newgrp docker
```

Im [folgenden Abschnitt](#) werden wir mit Docker-Basiswissen fortfahren.

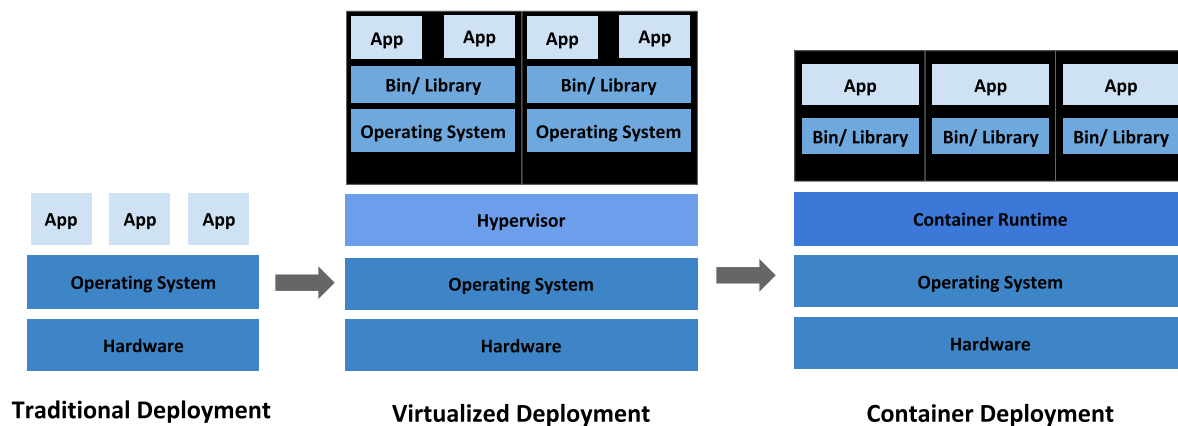
Basiswissen

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



Übersicht Docker

- Docker ist Freie Software (<https://github.com/docker>)
- 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

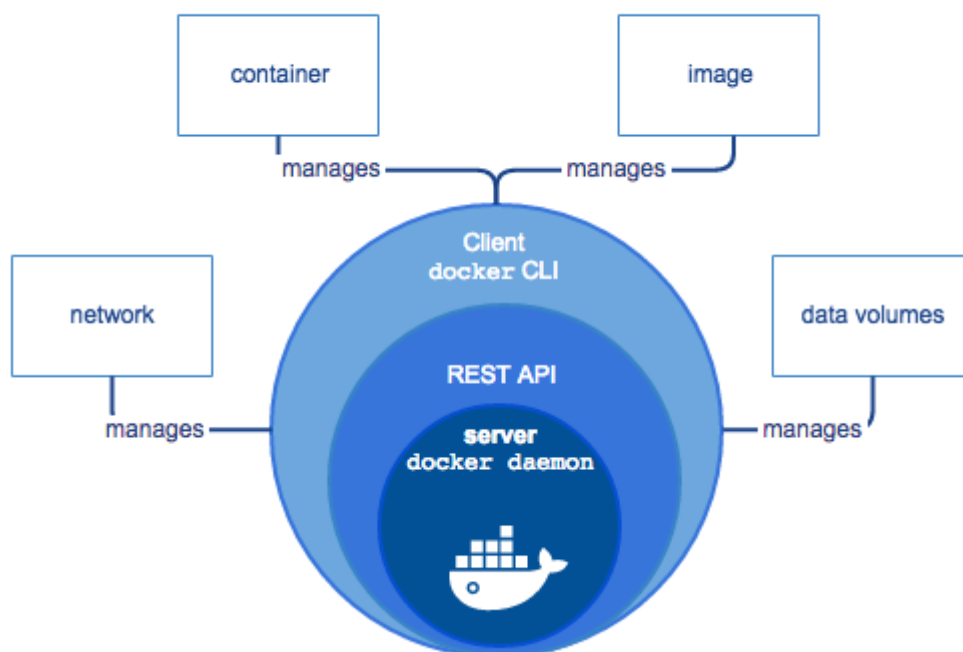
- **Dockerfile**
 - Textdatei mit Schritt-für-Schritt "Bauanleitung" für Docker-Images
- **Image**
 - Endprodukt des Bauens eines Dockerfiles, "Speicherabbild"
- **Container**
 - Konkret (laufende) Instanz eines Images
- **Registry**
 - Privates oder öffentliches Repository für Images, etwa <https://hub.docker.com/>

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 logs`
 - 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



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 letztlich die (geordnete) Sammlung von zahlreichen Layern.

Beispiel

Dockerfile

```
FROM busybox:1.37.0

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/*

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.

Information

Die Dokumentation zu Docker Compose findet sich auf <https://docs.docker.com/compose/>.

Docker Compose wird üblicherweise 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. Nutzernamen 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

Aufgaben

Erstellen Sie eine Datei `docker-compose.yml` in einem beliebigen Verzeichnis (z.B. `/home/user/docker-ws`) und fügen Sie folgenden Inhalt ein:

```

version: '3.8'
services:
  fossgis-geoserver:
    image: docker.osgeo.org/geoserver:2.26.2
    ports:
      - "8080:8080"
  fossgis-postgis:
    image: postgis/postgis:16-3.5-alpine
    ports:
      - "5433:5432"
    environment:
      POSTGRES_USER: fossgis
      POSTGRES_PASSWORD: fossgis

```

Speichern Sie das Dokument und wechseln Sie in das Terminal. Führen Sie dort den folgenden Befehl im Verzeichnis der `docker-compose.yml` aus:

```
docker compose up
```

Beobachten Sie den Terminal-Output, was fällt Ihnen aus? Öffnen Sie anschließend im Browser die Adresse <http://localhost:8080/geoserver> (Anmeldedaten `admin:geoserver`).

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.yml` definiert sind
- `docker compose up`
 - zum Starten der Services/Container
 - Mit dem Parameter `-f` lassen sich auch Dateien angeben, die nicht `docker-compose.yml` 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

Aufbau einer Geodateninfrastruktur

In diesem Kapitel werden wir Schritt für Schritt eine Geodateninfrastruktur (GDI) aufbauen. Ziel ist es, eine vollständige Umgebung zur Speicherung, Bereitstellung und Visualisierung von Geodaten zu erstellen. Dabei setzen wir auf bewährte Open-Source-Technologien, die in vielen professionellen GIS-Projekten weltweit zum Einsatz kommen.

Unsere GDI wird aus den folgenden Kernkomponenten bestehen:

- **OpenLayers** – Eine moderne, JavaScript-basierte Open-Source-Bibliothek, mit der wir interaktive Karten direkt im Browser anzeigen und mit ihnen interagieren können.
- **GeoServer** – Ein flexibler und weit verbreiteter Kartenserver, mit dem wir unsere Geodaten über standardisierte OGC-Dienste (z. B. WMS, WFS, WCS) bereitstellen können.
- **PostGIS** – Eine leistungsstarke Erweiterung für die PostgreSQL-Datenbank, die es uns ermöglicht, Geodaten effizient zu speichern, zu verwalten und abzufragen.

Während des Workshops werden wir die einzelnen Komponenten installieren, konfigurieren und in ein funktionierendes System integrieren. Am Ende hast du eine lauffähige GDI, mit der du eigene Geodaten verwalten und visualisieren kannst.

Zielarchitektur



Nächste Schritte

Im Folgenden werden wir die einzelnen Komponenten der Geodateninfrastruktur (GDI) Schritt für Schritt einrichten. Wir beginnen mit den Geodaten in der Datenbank, setzen mit dem GeoServer anschließend Geodienste darauf auf, um diese schließlich mit einer kleinen OpenLayers-Webanwendung zu visualisieren.

- [Vorarbeiten](#)
- [PostGIS](#)
- [GeoServer](#)
- [OpenLayers über nginx](#)

Folgen Sie den jeweiligen Links, um zur detaillierten Anleitung für die einzelnen Komponenten zu gelangen. 🚀

Vorarbeiten

Bevor wir mit der Einrichtung der Geodateninfrastruktur beginnen, müssen wir einige Vorbereitungen treffen.

1. Neues Verzeichnis anlegen

Erstellen Sie ein neues Verzeichnis namens `docker-gdi`, in dem alle notwendigen Dateien gespeichert werden:

```
mkdir -p ~/docker-gdi  
cd ~/docker-gdi
```

2. Materialien herunterladen und entpacken


Laden Sie das Archiv [materials.zip](#) in das `docker-gdi`-Verzeichnis herunter und entpacken Sie es:

```
wget https://github.com/terrestris/docker-ws/raw/refs/heads/main/materials/materials.zip  
unzip materials.zip
```

Das Archiv enthält einige wichtige Dateien, die wir später benötigen.

3. Neue Datei anlegen

Legen Sie eine Datei namens `docker-compose.yml` an.

 Hinweis: Die `docker-compose.yml` werden wir in den folgenden Abschnitten um unsere Dienste erweitern. Wir beginnen mit der Bereitstellung einer [PostGIS Datenbank](#).

Einrichten einer PostGIS-Datenbank

In diesem Abschnitt richten wir eine **PostGIS-Datenbank** über Docker Compose ein. Diese Datenbank bildet die Grundlage für unsere Geodateninfrastruktur.

PostGIS-Service hinzufügen


Fügen Sie in der `docker-compose.yml` eine neue service Konfiguration hinzu.

- Name: `fossgis-postgis`
- Image: `postgis/postgis:16-3.5-alpine`

Folgende Punkte sollten berücksichtigt werden:

- Der interne **PostgreSQL-Port 5432** soll auf den **Host-Port 5433** weitergeleitet werden, damit wir in der Lage sind Daten zu importieren.
- Das Datenverzeichnis der Datenbank (`/var/lib/postgresql/data`) sollte auf das Hostsystem **gemountet** werden.
- Ein **Datenbank-Benutzer** mit den Zugangsdaten `fossgis:fossgis` muss angelegt werden. Dies kann über das Setzen der folgenden Umgebungsvariablen erreicht werden:

```
POSTGRES_USER: fossgis
POSTGRES_PASSWORD: fossgis
```

 **Tipp:** Falls Sie unsicher sind, wie Sie das `docker-compose.yml` -File strukturieren sollen, werfen Sie einen Blick in die [Docker Compose Dokumentation](#).

Service starten

Starten Sie den PostGIS-Dienst mit folgendem Befehl:

```
docker compose up
```

Geodaten importieren

Um die Datenbank mit Geodaten zu füllen, importieren wir die weltweiten Landesgrenzen (`countries.sql` aus der entpackten `materials.zip`).



Variante 1 – Nutzung von pgAdmin:

- Öffnen Sie pgAdmin
- Verbinden Sie sich mit der Datenbank `fossgis`
- Importieren Sie die Datei `countries.sql`



Variante 2 – Terminal-Befehl: Alternativ können Sie das folgende Kommando nutzen:

```
psql -U fossgis -h localhost -p 5433 -d fossgis -f countries.sql
```

Prüfung der Daten

Nach dem Import sollten die Landesgrenzen in der PostGIS-Datenbank sichtbar sein. Ein Beispiel in pgAdmin:

The screenshot shows the pgAdmin interface. On the left, the 'Servers' tree is expanded to show the 'public' schema. The 'countries' table is selected. The main pane displays the 'Query Editor' with the following SQL query:

```
1 SELECT * FROM public.countries
2 ORDER BY ogc_fid ASC LIMIT 100
3
```

Below the query editor, the 'Data Output' tab is active, showing a table with 12 rows of data. The columns are 'ogc_fid', 'wkb_geometry', and 'name'. The data is as follows:

ogc_fid	wkb_geometry	name
1	0106000020110F000003...	Fiji
2	0106000020110F000001...	Tanzania
3	0106000020110F000001...	W. Sahara
4	0106000020110F00001E...	Canada
5	0106000020110F00000A...	United States of America
6	0106000020110F000001...	Kazakhstan
7	0106000020110F000001...	Uzbekistan
8	0106000020110F000004...	Papua New Guinea
9	0106000020110F000000...	Indonesia
10	0106000020110F000002...	Argentina
11	0106000020110F000002...	Chile
12	0106000020110F000001...	Peru

Geschafft!

Ihre **PostGIS-Datenbank** läuft nun in **Docker** und enthält die ersten Geodaten. Im nächsten Schritt erweitern wir unsere Geodateninfrastruktur um den **GeoServer**.

➔ Weiter zu [GeoServer](#) 

Einrichten des GeoServer-Services

Nachdem wir die PostGIS-Datenbank erfolgreich eingerichtet haben, erweitern wir unsere **docker-compose.yml** nun um den **GeoServer**, um Geodaten als OGC-Dienste bereitzustellen.


GeoServer-Service hinzufügen

Erweitern Sie die bestehende `docker-compose.yml` um einen weiteren Service.

- Name: `fossgis-geoserver`
- Image: `docker.osgeo.org/geoserver:2.26.2`

Folgende Punkte sollten berücksichtigt werden:

- Port-Mapping: Der interne (HTTP-) **Port 8080** auf sollte auf den **Host-Port 8080** weitergeleitet werden.
- Datenverzeichnis **mounten**:
 - Host: Eigener Pfad für persistente Speicherung, z.B. `./geoserver_data`
 - Container: `/opt/geoserver_data`
- **Startreihenfolge** festlegen:
 - `depends_on` : `fossgis-postgis`

 **Hinweis:** Die `depends_on` -Option stellt sicher, dass der GeoServer erst gestartet wird, wenn die PostGIS-Datenbank bereit ist.

GeoServer starten

Falls nötig, beenden Sie den laufenden Docker-Compose-Dienst:

```
docker compose down
```

Starten Sie das Compose-Netzwerk neu:

```
docker compose up
```

Zugriff auf GeoServer

Nach dem erfolgreichen Start können Sie den **GeoServer** im Browser unter folgender Adresse aufrufen:

 <http://localhost:8080/geoserver>

 **Login-Daten:**

- Benutzer: `admin`
- Passwort: `geoserver`

Arbeitsbereich anlegen

1. Melden Sie sich im GeoServer an.
2. Navigieren Sie zu **Arbeitsbereiche** (bzw. **Workspaces**).
3. Erstellen Sie einen neuen Arbeitsbereich mit dem Namen `FOSSGIS`.

PostGIS-Datenquelle einbinden

1. Navigieren Sie zu **Datenspeichern** (bzw. **Stores**) und erstellen Sie einen neuen Datenspeicher `POSTGIS`.
2. Verwenden Sie folgende Verbindungsparameter:

Parameter	Wert
Host	<code>fossgis-postgis</code>
Port	5432
Database	<code>fossgis</code>
Schema	public
User	<code>fossgis</code>
Password	<code>fossgis</code>

Neuen Layer hinzufügen

1. Neuen Layer `COUNTRIES` anlegen

Docker

- Quelle: Datenspeicher `POSTGIS`
- Tabelle: `countries`

2. Optional: Stil für den Layer festlegen

- Verwenden Sie den Stil `countries.sld` aus der `materials.zip` .

Geschafft!


Ihr **GeoServer** ist **nun eingerichtet** und veröffentlicht Geodaten als Web Map Services (WMS). 🎉 Im nächsten Schritt werden wir **OpenLayers über nginx** einbinden.

➡ Weiter zu [OpenLayers über nginx](#) 🚀

Einrichten des nginx-Services


Im letzten Schritt unseres Workshops richten wir einen **nginx-Service** ein, der als HTTP-Server eine **OpenLayers-Anwendung** bereitstellt. Dieser Service wird als dritter und letzter Bestandteil unserer Geodateninfrastruktur in die `docker-compose.yml` integriert.

Verzeichnisstruktur & Dateien vorbereiten

 Neues Verzeichnis für den nginx-Service anlegen:

```
mkdir fossgis-nginx
```

 Erstellen Sie in diesem Verzeichnis eine neue Datei namens `Dockerfile`

 Dateien aus der `materials.zip` einbinden:

- Kopieren Sie das `client` -Verzeichnis nach `fossgis-nginx` .
- Kopieren Sie die `default.conf` nach `fossgis-nginx` .

Dockerfile erstellen

Öffnen Sie das `Dockerfile` in einem Editor und definieren Sie ein aktuelles nginx Docker-Image als Basis. Nutzen Sie die aktuelle Version des offiziellen [nginx-Images](#).

Erweitern Sie das Dockerfile um folgende Schritte:

- Kopieren Sie die nginx-Konfiguration `default.conf` in das Image:
 - Zielpfad: `/etc/nginx/conf.d/default.conf`
- Kopieren Sie den Inhalt des `client` -Verzeichnisses in das Image:
 - Zielpfad: `/etc/nginx/html`
- Geben Sie den Port **80** für den nginx-Prozess an.

nginx-Service hinzufügen

Erweitern Sie die bestehende `docker-compose.yml` um einen weiteren Service.

- Name: `fossgis-geoserver`

Folgende Punkte sollten berücksichtigt werden:

- Statt eines Image einen Build-Context nutzen, also das Dockerfile im `fossgis-nginx` -Unterverzeichnis.
- Port-Mapping:
 - Container-Port: `80`
 - Host-Port: `8000`
- Startreihenfolge (`depends_on`) beachten:
 - `fossgis-geoserver`

 **Hinweis:** nginx wird als **letzter Dienst** gestartet, damit die Geodaten bereits verfügbar sind.

Alle Services neu starten

Docker

Falls nötig, stoppen Sie alle Services:

```
docker compose down
```

Starten Sie anschließend das gesamte Compose-Netzwerk neu:

```
docker compose up --build
```

OpenLayers-Anwendung aufrufen

Nach dem erfolgreichen Start können Sie die **OpenLayers-Anwendung** im Browser öffnen:

 <http://localhost:8000>

Geschafft!

Ihre komplette Geodateninfrastruktur läuft nun mit Docker.

✓ **PostGIS-Datenbank** speichert die Geodaten

✓ **GeoServer** stellt OGC-Dienste bereit

✓ **nginx** hostet die OpenLayers-Anwendung

 Weiter zur [Musterlösung](#) 

Musterlösung

Eine Musterlösung für die Beispiel-GDI finden Sie [hier](#).

Startansicht des Kartenclients

