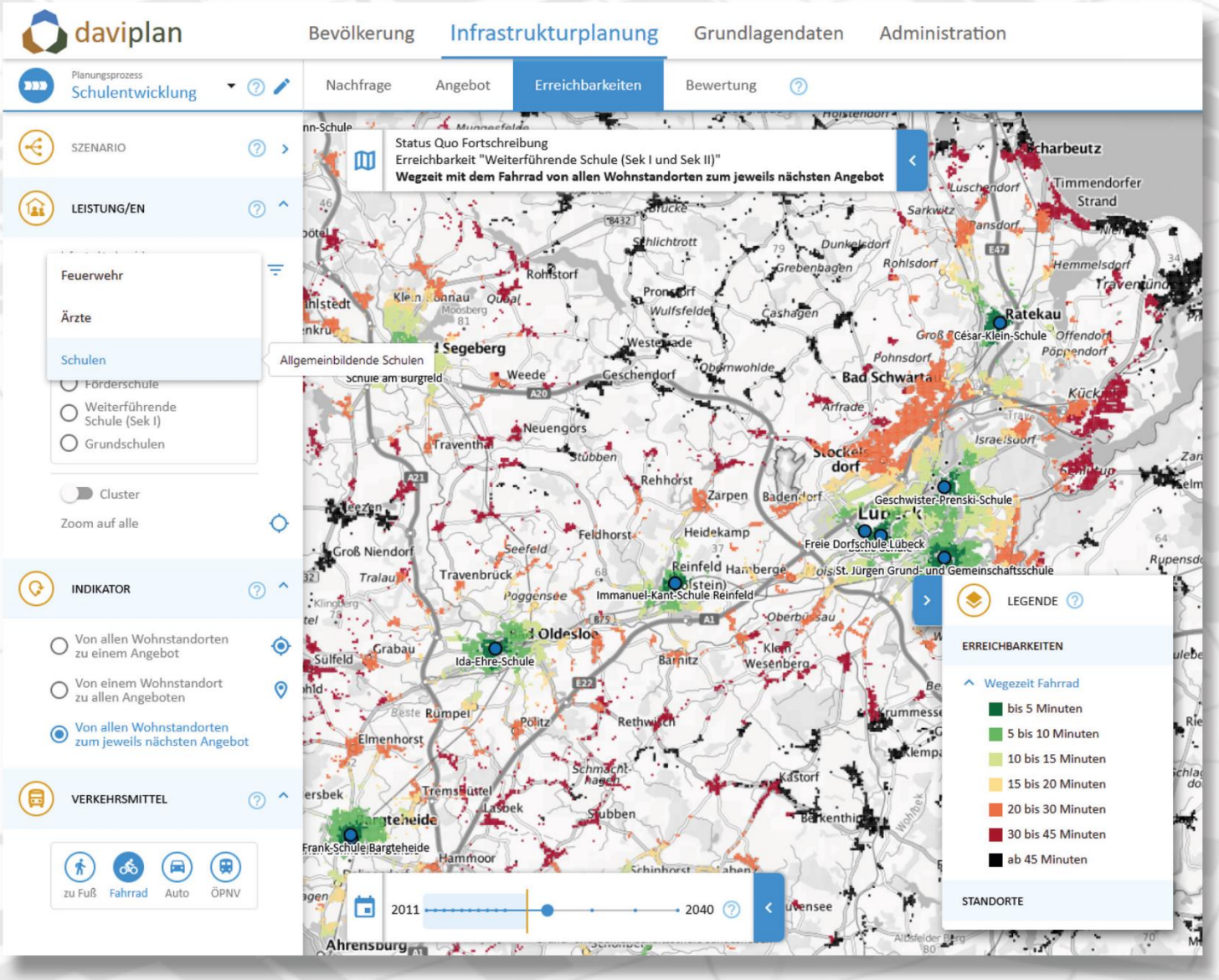
Region gestalten

daviplan

Ein online-Werkzeug für Regionen und Kommunen
zur Planung von Infrastrukturen der Daseinsvorsorge



Installationsanleitung

Inhaltsverzeichnis

1 Systemvoraussetzungen

Das System daviplan ist als Web-Anwendung (Webseite) für den Betrieb auf einzelnen Linux-Servern konzipiert. Die Anleitung bezieht sich daher auf eine Einrichtung der Anwendung auf einem Linux-System (speziell Ubuntu).

Hard- und Software-Voraussetzungen des Servers:

- mindestens 32GB RAM
- Installation von Docker (Installationsanleitung) und docker-compose
- Installation von Python
- Einrichtung eines SSH-Zugriffs auf den Server
- Installation eines Webservers (z.B. Apache2)

Wir empfehlen die Nutzung einer SSD als Speicherort für die Datenbank (standardmäßig im Installationsverzeichnis). Im Vergleich mit einer HDD ist der Zugriff vor allem bei großen Datenmengen deutlicher performanter.

Für einen Betrieb als im Internet verfügbare Web-Anwendung werden außerdem benötigt:

- Eine registrierte Domain
- Ein gültiges SSL-Zertifikat für die registrierte Domain (bei Bedarf einschließlich der Subdomain)
- Eine öffentliche IP des Servers, über die der Server im Internet erreichbar ist
- Weiterleitung von Anfragen an den Namen der (Sub-)Domain an die IP des Servers

2 Installation

Die Installation der Anwendung auf dem Server erfolgt mit einer Reihe von mitgelieferten Skripten (create_environment.py, createsuperuser.sh und init_database.sh) und einer YAML-Spezifikation der Services (*docker-compose.yml* siehe Abbildung 1).

Die eigentliche Anwendung wird als Docker-Container ausgeliefert und als Service auf dem Server betrieben. Der Docker-Container ist verfügbar auf Docker-Hub unter dem Tag *gertzgutscheruemen-app/daviplan.* Die Anwendung benötigt zum vollständigen Betrieb weitere Container, die in der ausgelieferten *docker-compose.yml* spezifiziert sind.

2.1 Inbetriebnahme der Services der Web-Anwendung

Schritt 1: Ablegen der Dateien auf dem Server

Legen Sie ein Verzeichnis auf dem Server an, möglichst in einem geschützten Bereich, auf den nur root-Nutzer Zugriff hat. Kopieren Sie die *docker-compose.yml* und alle Skripte in dieses Verzeichnis. Die Shellskripte (*.sh*) müssen als ausführbar markiert werden (`chmod +x <Dateiname>`).

Alle im laufenden Betrieb erzeugten Daten werden automatisch in diesem Verzeichnis in Unterordnern abgelegt (statische Daten für das Frontend, die Dateien der Datenbank, Routingdaten etc.).

Im Folgenden wird auf dieses Verzeichnis als „Installationsverzeichnis“ Bezug genommen.

Schritt 2: Umgebungsvariablen anlegen

Die Anwendung benötigt für den Betrieb eine Reihe an essentiellen Informationen, die über Umgebungsvariablen in die Container übermittelt werden. Daher muss vor dem Start der Services deren Konfiguration erfolgen.

Führen Sie dazu das Skript **create_environment.py** im Installationsverzeichnis aus (`python create_environment.py`).

Sie werden unter anderem aufgefordert sicherheitsrelevante Schlüssel einzugeben. Da die Schlüssel sehr spezifische Formate benötigen, können Sie sie auch automatisch generieren lassen, indem Sie die Eingabe frei lassen und bestätigen.

Daneben werden Sie aufgefordert einen Port anzugeben, über den die Web-Anwendung lokal erreichbar sein wird. Dieser Port muss frei verfügbar sein, darf also nicht für eine andere Anwendung auf dem Server reserviert sein.

Geben Sie beim Domainnamen die registrierte Domain + Domainendung ein, über die die Webseite später erreicht werden kann (z.B. daviplan-regional.de)

Das Skript erzeugt eine .env-Datei, die beim Start der Services über die docker-compose.yml eingelesen wird.

Beachten Sie, dass das Skript bei Ausführung eine möglicherweise bereits existierende .env-Datei und damit vorherig getätigte Einstellungen überschreibt. Nachträgliche Änderungen sollten daher direkt in der .env-Datei vorgenommen werden.

Schritt 3: Starten des Services

Die beiden nachfolgenden Schritte 4 und 5 benötigen eine laufende Anwendung.

Sie starten die Anwendung inklusive aller benötigten Services, indem Sie im Installationsverzeichnis den Befehl `docker-compose up -d` ausführen. Die Anwendung ist daraufhin über den von Ihnen angegebenen Port lokal erreichbar. Die Container erhalten automatische Namen, die dem Installationsverzeichnis entsprechen. Bitte benennen Sie die Container nicht selbst, sonst funktionieren die Shell-Skripte in Schritt 4 und 5 nicht.

Die Anwendung ist in der *docker-compose.yml* so konfiguriert, dass sie sowohl bei einem Crash als auch bei einem Neustart des Servers automatisch (neu) gestartet wird. Eine Ausnahme ist, wenn Sie sie willentlich stoppen (mit `docker-compose down`)

Schritt 4: Datenbank initialisieren

Die Datenbank enthält zu Beginn (fast) keine Objekte. Die Anwendung benötigt aber im Betrieb einige vordefinierte Daten wie beispielsweise vordefinierte Gebietseinteilungen.

Mit Ausführung des Skripts **init_database.sh** werden im laufenden Container die Datenbankobjekte angelegt.

Alternativ können Sie den Befehl, der in dem Skript ausgeführt wird, auch manuell eingeben:

```
docker exec -it <container> python manage.py initproject
```

wobei „<container>“ mit dem durch Docker vergebenen bzw. festgelegten Namen des Containers ersetzt werden muss.

Beachten Sie, dass die Ausführung dieses Befehls Teile bereits bestehender Daten löscht! Führen Sie Ihn also nicht aus, wenn Sie Datenverlust in einem Systems mit bereits eingespielten Daten vermeiden wollen.

Schritt 5: Superuser anlegen

Zu Beginn sind noch keine Nutzer definiert. Damit Sie sich initial auf der Webseite einloggen können, muss ein Superuser (Administrator) erstellt werden. Dieser kann sich dann auf die Webseite einloggen und weitere Nutzer:innen anlegen.

Superuser haben vollen Zugriff auf alle Daten. Vergeben Sie also ein entsprechend sicheres Passwort und schützen Sie es vor unbefugtem Zugriff.

Zudem haben Superuser direkten Zugriff auf die Daten im Django-Adminbereich der Web-Anwendung über *https://<Domain+Domainendung >/django-admin/.*

Mit Ausführung des Skripts ***createsuperuser.sh*** wird im laufenden Container ein neuer Superuser angelegt. Sie werden aufgefordert einen Nutzernamen, eine optionale E-Mail-Adresse und ein sicheres Passwort einzugeben. Die eingegebene Nutzer-Passwort-Kombination gilt dann auch für den Login auf der Webseite.

Alternativ können Sie den Befehl, der in dem Skript ausgeführt wird, auch manuell eingeben:

```
docker exec -it <container> python manage.py createsuperuser
```

wobei „<container>“ mit dem durch Docker vergebenen bzw. festgelegten Namen des Containers ersetzt werden muss.

Sie können später die Passwörter der Superuser nicht in der Oberfläche der Anwendung ändern. Möchten Sie die Passwörter dennoch ändern, können Sie dies nur im Django-Adminbereich oder über folgenden Befehl:

```
docker exec -it <container> python manage.py changepassword <Nutzername>
```

2.2 Weitere Konfigurationsmöglichkeiten

Im Folgenden werden Einstellungen beschrieben, die von den im Auslieferungszustand angedachten Einstellungen abweichen. Dazu müssen Anpassungen an der *docker-compose.yml* vorgenommen, die ansonsten unverändert bleiben sollte.

Datenbank

In der *docker-compose.yml* ist eine PostGIS-Datenbank voreingestellt, die nicht passwortgeschützt ist, deren Kommunikation nicht verschlüsselt wird und die alle Verbindungen annimmt.

Da sie mit den vorgeschlagenen Einstellungen nicht von außen erreichbar ist und die Daten im gleichen Verzeichnis abgelegt werden, in dem sich auch die *docker-compose.yml* und die *.env*-Datei befinden, stellt dies dennoch kein Sicherheitsrisiko dar. Sollten Sie allerdings die Datenbank nach außen über Ports verfügbar machen wollen, sollten Sie sie mit einem starken Passwort schützen und eine SSL-Verschlüsselung erzwingen.

Wenn Sie eine bestimmte Datenbank verwenden oder die vorkonfigurierte Datenbank mit Passwort und SSL schützen möchten, müssen Sie entsprechende Änderungen in der *docker-compose.yml* vornehmen und in der *.env*-Datei die Datenbank-Umgebungsvariablen (DB_NAME, DB_PASS etc.) anpassen.

Versionierung

In der *docker-compose.yml* ist die Web-Anwendung so spezifiziert, dass die neueste verfügbare Version benutzt wird (Tag „latest“). Sollten Sie eine bestimmte Version festlegen wollen, tauschen Sie den „latest“-Tag unter „image“ des Services „web“ mit dem entsprechenden Tag der Version aus. Die verfügbaren Tags sind im Docker-Hub einsehbar.

Beachten Sie, dass nur bei erstmaligem Aufruf eines Containers der aktuelle Stand von Docker-Hub abgerufen wird. Das gilt sowohl für den „latest“-Tag, als auch für alle anderen Tags. In Abschnitt 2.4 wird beschrieben, wie sie den aktuellen Stand abrufen können.

Sie können auch die Versionen der anderen genutzten Container (Redis und PostGIS) verändern. Allerdings wurde die Anwendung auf die voreingestellten Versionen angepasst und mit diesen getestet. Ein reibungsloser Betrieb ist bei Änderung der Versionsnummern nicht garantiert.

Interner Betrieb im lokalen Netzwerk als Staging-Server

Wenn daviplan nur intern in einem Netzwerk betrieben werden soll ohne Domain und ohne Zugriff von außen, kann die Anwendung auch als Staging-Server betrieben werden. Der Zugriff auf die Website erfolgt dann im Browser über die IP des Servers und den Port der Anwendung. Der Staging-Modus ist allerdings unsicherer und etwas weniger stabil als der produktive Modus.

Um die Anwendung als Staging-Server zu betreiben, muss in der *docker-compose.yml* die Umgebungsvariable ENVIRONMENT auf „staging“ (ohne Anführungszeichen) gesetzt werden.

In der *.env*-Datei muss die IP des Servers in der Umgebungsvariable ALLOWED_HOSTS eingetragen werden. Ist die IP des Servers nicht statisch bzw. treten Probleme beim Aufruf der Seite (zum Beispiel beim Login) auf, können Sie stattdessen auch ein „*“ (ohne Anführungszeichen) eintragen.

Der ALLOWED_HOSTS-Eintrag ist eine Sicherheitsmaßnahme gegen Host-header-Angriffe auf die API und prüft, dass nur die Website auf diese zugreifen darf. Ein „*“ erlaubt Zugriffe ungeachtet der anfragenden IP bzw. des Domainnamens. Wenn die Seite nicht von außerhalb des Netzwerks erreichbar ist, ist das aber nicht sicherheitsrelevant.

Beim Staging-Server entfällt das Mapping der statischen Dateien über den Webserver (siehe Abschnitt 2.6), da im Backend das Django-Framework die Auslieferung aller benötigten Dateien übernimmt.

Logs während der Berechnungen in den Grundlagendaten werden im Frontend im DEBUG-Modus angezeigt. Das bedeutet, dass neben den normalen Informationen deutlich mehr Nachrichten in der „Daten-Historie“ angezeigt werden.

Im Gegensatz zum produktiven Betrieb kann im Staging-Modus die Rest-API über den Browser ohne die Verwendung von Webtokens aufgerufen werden (über die Route */api/*), vorausgesetzt es existiert eine Session (z.B. Einloggen über */django-admin/*).

Sie können den Staging-Server theoretisch auch über eine Domain von außen erreichbar machen, wenn Sie den Webserver entsprechend konfigurieren. Das sollte aus Sicherheitsgründen aber nur zu Testzwecken erfolgen. Der produktive Modus ist für einen öffentlichen Betrieb über eine Domain besser geeignet.

Ausschließlicher lokaler Betrieb (ohne Netzwerkzugriff)

Bei einem ausschließlich lokalen Betrieb ohne Zugriff über eine Domain entfällt die Einrichtung des Webservers ganz. Ansonsten verhält sich die Anwendung wie im Betrieb als Staging-Server.

Derzeit wird für diesen Betrieb nur der Port 8000 unterstützt (der in Abschnitt 2.1, Schritt 2 angegebene Port - Variable EXT_PORT). Die gestartete Anwendung ist ausschließlich lokal über diesen Port erreichbar.

Beachten Sie, dass der Betrieb der Anwendung im Development-Modus noch unsicherer ist als im Betrieb als Staging-Server. Veröffentlichen Sie daher die Anwendung in diesem Modus nicht im Internet. Ihnen steht allerdings frei, den Port der Anwendung im internen Netzwerk freizugeben.

Um die Anwendung in diesem Development-Modus lokal zu betreiben, setzen Sie in der *docker-compose.yml* die Umgebungsvariable ENVIRONMENT auf „development“ (ohne Anführungszeichen).

2.3 Backup einrichten

Für ein Backup der in der *docker-compose.yml* im Auslieferungszustand voreingestellten Datenbank, kann das ausgelieferte Shellskript **backup_rotated.sh** verwendet werden. Das Skript muss dazu zusammen mit der zugehörigen Konfigurationsdatei **backup.config** im Installationsverzeichnis der *docker-compose.yml* abgelegt werden.

Der Wert der in der Konfigurationsdatei angegebenen Variable BACKUP_DIR sollte mit einem Verzeichnis auf dem Server ersetzt werden. Die Sicherungen der Datenbank werden bei Ausführung in das angegebene Verzeichnis geschrieben. Schützen Sie dementsprechend das Verzeichnis vor unbefugtem Zugriff. In dem Verzeichnis werden bei Ausführung Unterordner je Datum des Backups angelegt. Innerhalb dieser Ordner werden wiederum Ordner für die Backups der Systemdatenbank und der eigentlichen Datenbank mit den Daten der Anwendung als SQL-Dumps abgelegt.

Ältere Backups werden bei Ausführung des Skripts automatisch gelöscht. Die Zeiträume lassen sich in der Konfigurationsdatei festlegen.

Die anderen Variablen der Konfigurationsdatei müssen nur verändert werden, wenn Sie spezielle Anpassungen benötigen.

Für eine einmalige Sicherung des momentanen Zustands der Datenbank geben Sie im Installationsverzeichnis folgenden Befehl ein (backup_rotated_pg.sh muss als ausführbar markiert sein):

```
backup_rotated.sh -c <Konfigurationsdatei>
```

wobei <Konfigurationsdatei> mit dem Pfad zur tatsächlichen Konfigurationsdatei ersetzt werden muss. Wenn Sie den Parameter „-c backup.config“ weglassen, wird die Datei *backup.config* verwendet, vorausgesetzt sie liegt in demselben Verzeichnis.

Um regelmäßige Updates durchzuführen, können Sie den Aufruf des Skript zum Beispiel als Cronjob festlegen.

2.4 Updates einspielen

In der ausgelieferten *docker-compose.yml* ist für die Web-Anwendung der „latest“-Tag als Version voreingestellt. Im Docker-Hub wird fortlaufend die aktuelle Version als „latest“ getaggt.

Um zu prüfen, ob eine aktuellere Version vorliegt und diese dann herunterzuladen, muss folgender Befehl im Installationsverzeichnis ausgeführt werden:

```
docker pull gertzgutscheruemenapp/daviplan:latest
```

Um die Änderungen bei laufendem System wirksam zu machen, starten Sie bitte den Container mit dem Befehl `docker-compose restart` im Installationsverzeichnis neu.

2.5 Neustart der Anwendung

Für den Fall, dass die Anwendung über einen längeren Zeitraum nicht mehr auf Anfragen reagiert oder sich vollständig aufgehängt hat und nicht von selbst wieder startet, können Sie die Services neu starten, indem sie `docker-compose restart` im Installationsverzeichnis ausführen.

Alternativ können Sie die Anwendung mit `docker-compose down` stoppen und mit `docker-compose up -d` wieder starten.

2.6 Konfiguration des Webservers

Die Konfiguration des Webservers hängt vom System und vom benutzten Webserver ab. Generell müssen folgende Punkte konfiguriert werden:

- Die Weiterleitung von HTTP/HTTPS-Anfragen (*https://<domain>)* an die Domain auf den lokalen Port der Anwendung
- Die Weiterleitung von Websocket-Anfragen (*ws://<domain>*) auf den lokalen Port der Anwendung
- Die Einbindung des SSL-Zertifikats
- Im produktiven Betrieb: Mappen der Routen */static* und */media* auf die entsprechenden Verzeichnisse im Ordner *public* im Installationsverzeichnis (inklusive allgemeine Zugriffsberechtigung)
- Die Weiterleitung des Headers zum Service der Web-Anwendung

Im Folgenden wird beispielhaft eine Konfiguration eines Apache2-Webservers skizziert. In Abbildung 2 ist ein Template für eine Konfigurationsdatei von Apache gezeigt, die die oben genannten Punkte erfüllt. Zur Anpassung an Ihre Umgebung, müssen Sie die Werte der zu Beginn definierten Variablen ersetzen. Außerdem müssen die Pfade zu den Dateien des SSL-Zertifikats angepasst werden.

Die Konfigurationsdatei muss im Verzeichnis mit den Website-Konfigurationsdateien von Apache abgelegt, üblicherweise unter */etc/apache2/sites-available/.* Sie kann einen beliebigen Namen erhalten, muss aber auf *.conf* enden.

Zum Ausführen der Konfiguration werden folgende Apache-Mods benötigt, die sich jeweils mit `a2enmod <Modname>` aktivieren lassen:

- headers
- proxy
- ssl
- proxy_http

Wenn in der Konfigurationsdatei ein Errorlog angegeben ist, muss dessen Verzeichnis manuell angelegt werden, bevor die Konfiguration ausgeführt werden kann.

Um die Konfiguration und damit die Seite zu aktivieren, geben Sie `a2ensite <NameDerKonfiguration>` ein. Beachten Sie, dass der Name der Konfiguration dem Namen der Konfigurationsdatei ohne deren Endung *.conf* entspricht. Der Befehl a2ensite legt einen Softlink der Konfiguration unter */etc/apache2/sites-enabled/* ab. Analog entfernt `a2dissite <NameDerKonfiguration>` den Softlink wieder.

Wenn alle Einstellungen korrekt durchgeführt wurden, die Services der Web-Anwendung laufen, sollte nach Aktivierung der Apache-Konfiguration und einem Neustart von Apache (`systemctl reload apache2`) die Webseite unter dem eingestellten Domainnamen erreichbar sein.

```yaml
version: '3.1'
services:
  # PostGIS-Datenbank (ungeschützt, bitte nicht nach außen freigeben)
  postgis:
    image: postgis/postgis:13-3.1-alpine
    restart: unless-stopped
    environment:
      POSTGRES_HOST_AUTH_METHOD: trust
      POSTGRES_DB: datentool
    volumes:
      - postgres_data:/var/lib/postgresql/data/
  # Redis wird für das Logging in der Oberfläche und für Background-Tasks der API benötigt
  redis:
    image: redis:7.0.4
    restart: unless-stopped
    expose:
      - "6379"
    volumes:
      - ./redis_data:/data
  # OSRM-Router je Modus laufen in separaten Containern
  router-car:
    image: gertzgutscheruemenapp/osrm-flask
    restart: unless-stopped
    volumes:
      - ./routing_data:/app/data
  router-bike:
    image: gertzgutscheruemenapp/osrm-flask
    restart: unless-stopped
    volumes:
      - ./routing_data:/app/data
  router-walk:
    image: gertzgutscheruemenapp/osrm-flask
    restart: unless-stopped
    volumes:
      - ./routing_data:/app/data
  # die eigentliche Web-Anwendung (Auslieferung des Frontends + Rest-API)
  web:
    # Einbindung der Umgebungsvariablen aus der .env-Datei
    env_file: .env
    image: gertzgutscheruemenapp/daviplan:latest
    # weitere Umgebungsvariablen
    environment:
      # auf „staging" setzen für einen Staging-Server oder „development" für lokalen Test
      - ENVIRONMENT=productive
      - DJANGO_SITENAME=datentool
      - MODE_CAR_HOST=router-car
      - MODE_BIKE_HOST=router-bike
      - MODE_WALK_HOST=router-walk
      - MODE_CAR_SERVICE_PORT=8001
      - MODE_BIKE_SERVICE_PORT=8001
      - MODE_WALK_SERVICE_PORT=8001
      - REDIS_HOST=redis
    links:
      - postgis:db
      - router-car:router-car
      - router-bike:router-bike
      - router-walk:router-walk
      - redis:redis
    # Befehle bei Start: Ausführung eines Skripts im Container, das die Migration der
    # Datenbank durchführt und den Server und Hintergrundprozesse startet
    # (in Abhängigkeit der Umgebungsvariable "")
    command: /bin/sh -c "sleep 5s && /run.sh"
    # der Port, über den die Anwendung von außen erreichbar ist
    ports:
      - "${EXT_PORT}:8000"
    # Neustart der Anwendung bei Crash oder Server-Neustart bis sie absichtlich gestoppt wird
    restart: unless-stopped
    volumes:
      # Verzeichnis mit den static und media files
      - ./public:/datentool/public
volumes:
  postgres_data:
```

Abbildung 1 *Die ausgelieferte docker-compose.yml zur Spezifikation der zum Betrieb der Web-Anwendung benötigten Services*

```
# VARIABLEN, Bitte die Einträge mit den tatsächlichen Werten der Umgebung ersetzen!
# registrierte Domain (+ optionale Subdomain) über die die Webseite erreichbar sein wird
Define DOMAIN subdomain.domain.domainendung
# der locale Port des Services der Web-Anwendung (siehe EXT_PORT in der .env im
# Installationsverzeichnis
Define DATENTOOL_PORT 8092
# das Installationsverzeichnis mit der docker-compose.yml
Define DATENTOOL_DIR /path/to/dockercompose/

# Weiterleitung von HTTP-Anfragen auf das sichere HTTPS
<VirtualHost *:80>
    ServerName ${DOMAIN}
    Redirect 301 / https://${DOMAIN}/
</VirtualHost>

# HTTPS
<VirtualHost *:443>
    ServerName ${DOMAIN}
    # Ihre Kontakt-E-Mail
    ServerAdmin Admin@admin.org

    # Ort des Error-Logs (MUSS MANUELL ANGELEGT WERDEN)
    ErrorLog /var/log/apache2/datentool/error.log

    # Weiterleitung der Header
    RequestHeader set Host "${DOMAIN}"
    RequestHeader add X-Forwarded-Ssl on
    RequestHeader set X-Forwarded-For %{REMOTE_IP}e
    RequestHeader set X-Forwarded-Proto "https"
    RequestHeader set X-Forwarded-Port 443

    ProxyRequests Off
    ProxyPreserveHost On

    # Auschluss der static- and media-Routen von der Weiterleitung
    # (warden weiter unten auf die Verzeichnisse gemappt)
    ProxyPassMatch ^/static !
    ProxyPassMatch ^/media !
    # Weiterleitung der HTTP-Anfragen and den Port der Anwendung
    ProxyPass / http://localhost:${DATENTOOL_PORT}/

    # Weiterleitung der Websocket-Anfragen
    RewriteEngine On
    RewriteCond %{HTTP:UPGRADE} ^WebSocket$ [NC,OR]
    RewriteCond %{HTTP:CONNECTION} ^Upgrade$ [NC]
    RewriteRule ^/?(.*) "ws://localhost:${DATENTOOL_PORT}/$1" [P,L]

    # Mapping der static- und media-Routen auf die Ordner im Installationsverzeichnis
    Alias "/static" "${DATENTOOL_DIR}/public/static"
    Alias "/media" "${DATENTOOL_DIR}/public/media"
    <Directory "${DATENTOOL_DIR}/public">
        Options FollowSymLinks
        Require all granted
    </Directory>

    SSLEngine On
    SSLProxyEngine On
    # SSL-Zertifikate
    # ERSETZEN mit den tatsächlichen Zertifikatdateien!
    SSLCertificateFile  /etc/ssl/certs/certificate.de.crt
    SSLCertificateKeyFile /etc/ssl/private/certificate.de.key
    # falls das Zertifkat intermediate ist und immer wieder verlängert wird
    SSLCACertificateFile /etc/ssl/certs/intermediate.crt

    # erlaubte Protokolle und Verschlüsselungen der SSL-Verbindung
    SSLProtocol all -SSLv2 -SSLv3 -TLSv1
    SSLHonorCipherOrder on
    SSLCipherSuite ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-
CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-
GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-
RSA-AES128-SHA256

</VirtualHost>
```

Abbildung 2 Template einer Apache2-Konfigurationsdatei für den Zugriff auf die Web-Anwendung über den Domainnamen im Internet