praxis.md

Praxis: docker-compose

Vorbereitung

- Erstelle ein neues Verzeichnis und öffne deinen Texteditor

```
mkdir docker-compose && code docker-compose
```

- Erstelle eine neue `docker-compose.yml` Datei im neuen Verzeichnis

Erste Schritte

- Als erstes müssen wir die verwendete `docker-compose` Version definieren, wir nutzen v3
- Danach definieren wir die Services also Container, die erstellt werden sollen:
- Für den ersten Test nutzen wir `nginx:latest`
- Beachte, dass der `:` ein spezielles Symbol in Yaml ist und deshalb das image quotiert werden muss `"`

```yaml
version: '3'

services:
  web:
    image: "nginx:latest"
```

- Um die Anwendung zu starten gebe nun folgenden Befehl in die Console ein:
- **WICHTIG** Die Console muss sich im gleichen Ordner befinden, wie die docker-compose.yml Datei!**

```
docker-compose up
```

- Du kannst den Prozess mit `Ctrl-C` wieder beenden

Port-Forwarding

- Um auch auf das Frontend zugreifen zu können, müssen wir den nginx Port noch vom Container weiterleiten
- Das funktioniert nach folgendem Prinzip: `HOST:CONTAINER`
- Also erst der Port den wir im Browser eingeben, und dann der Port im Container

```yaml
version: '3'

services:
  web:
    image: "nginx:latest"
    ports:
      - "8080:80"
```

- Jetzt kannst du im Browser folgende URL eingeben: `http://localhost:8080`

Volumes

- Volumes werden hauptsächlich für zwei Zwecke genutzt:

    i. Um Daten über Container Restarts hinweg zu persistieren
    ii. Um Dateien im Container zu überschreiben

- In diesem Beispiel wollen wir die `Welcome to nginx` Seite überschreiben
- Erstelle einen neuen Ordner: `src`
- Erzeuge eine neue Datei `src/index.html` mit folgendem Inhalt

```
<h1>Hi from Docker</h1>
```

- Nun müssen wir nur noch den `/usr/share/nginx/html` Ordner im Container mit unserem lokalen `src` Ordner überschreiben

```
version: '3'

services:
  web:
    image: "nginx:latest"
    ports:
      - "8080:80"
    volumes:
      - "./src:/usr/share/nginx/html"
```

Wordpress

- Im folgenden setzen wir eine Wordpress Installation mit docker-compose auf
- Hierfür orchestrieren wir zwei verschiedene Container:
    - Eine `mariadb` Datenbank (mysql fork)
    - Den `wordpress` Container
- Lösche den Inhalt der `docker-compose.yml` Datei und füge folgendes ein:
- Über das `command` Feld können weitere Commandline Argumente angegeben werden, die beim Start des Containers aufgerufen werden
- In diesem Fall setzen wir die Authentication-Methode zu `mysql_native_password` was für Wordpress benötigt wird

```
services:
  db:
    image: mariadb:11.1.2
    command: '--default-authentication-plugin=mysql_native_password'

  wordpress:
    image: wordpress:latest
```

- Viele Docker images können mittels Environment Variable konfiguriert werden.
- In unserem Fall müssen wir für die Datenbank User, Passwort und Datenbankname festlegen und diese auch der Wordpress installation mitteilen:

```
services:
  db:
    image: mariadb:11.1.2
    command: '--default-authentication-plugin=mysql_native_password'
    environment:
      - MYSQL_ROOT_PASSWORD=somewordpress
      - MYSQL_DATABASE=wordpress
      - MYSQL_USER=wordpress
      - MYSQL_PASSWORD=wordpress
```

```
  wordpress:
    image: wordpress:latest
    environment:
      - WORDPRESS_DB_HOST=db
      - WORDPRESS_DB_USER=wordpress
      - WORDPRESS_DB_PASSWORD=wordpress
      - WORDPRESS_DB_NAME=wordpress
```

- Ähnlich wie zuvor geben wir auch hier ein Port Mapping an:

```
  wordpress:
    image: wordpress:latest
    ports:
      - 8000:80
```

- In diesem Schritt nutzen wir **Volumes** um Daten der DB auch über Restarts hinweg zu sichern
- Wir definieren zuerst ein Top-level Volume am Ende der Datei:

```
volumes:
  db_data:
```

- Dieses können wir dann im Volume-Mapping des `db` Service verwenden:

```
services:
  db:
    image: mariadb:11.1.2
    command: '--default-authentication-plugin=mysql_native_password'
    volumes:
      - db_data:/var/lib/mysql
```

- Die gesamte Datei sieht als jetzt wie folgt aus:

```
services:
  db:
    image: mariadb:11.1.2
    command: '--default-authentication-plugin=mysql_native_password'
    volumes:
      - db_data:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=somewordpress
      - MYSQL_DATABASE=wordpress
      - MYSQL_USER=wordpress
      - MYSQL_PASSWORD=wordpress

  wordpress:
    image: wordpress:latest
    ports:
      - 8000:80
    environment:
      - WORDPRESS_DB_HOST=db
      - WORDPRESS_DB_USER=wordpress
      - WORDPRESS_DB_PASSWORD=wordpress
      - WORDPRESS_DB_NAME=wordpress
volumes:
  db_data:
```

- Jetzt erstellen wir noch ein Network um die Kommunikation zwischen den Services zu definieren (vorallem für mögliche weitere Services wichtig)

```
networks:
  wp-net: {}
```

- Dieses Network muss jetzt in den Services verwendet werden:

```
# Verkürzte Darstellung
services:
  db:
    image: mariadb:11.1.2
    # ...
    networks:
      - wp-net

  wordpress:
    image: wordpress:latest
    # ...
    networks:
      - wp-net
```

- Eine weiteres Feature ist die Restart-Policy, in unserem Fall wollen wir, dass bei einem Fehler der Service automatisch neu gestartet wird:

```
restart: always
```

Finale Wordpress docker-compose Datei

```
services:
  db:
    image: mariadb:11.1.2
    command: '--default-authentication-plugin=mysql_native_password'
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      - MYSQL_ROOT_PASSWORD=somewordpress
      - MYSQL_DATABASE=wordpress
      - MYSQL_USER=wordpress
      - MYSQL_PASSWORD=wordpress
    networks:
      - wp-net

  wordpress:
    image: wordpress:latest
    ports:
      - 8000:80
    restart: always
    environment:
      - WORDPRESS_DB_HOST=db
      - WORDPRESS_DB_USER=wordpress
      - WORDPRESS_DB_PASSWORD=wordpress
      - WORDPRESS_DB_NAME=wordpress
    networks:
      - wp-net
volumes:
  db_data:

networks:
  wp-net: {}
```