

Web Engineering 3

Vorlesung 5

Hochschule Zittau/Görlitz

Christopher-Manuel Hilgner

Agenda

Vorlesung

- Spring Data JPA
- Konfiguration von Spring Anwendungen
- Einführung in Docker

Seminar

- Relationen zwischen Entities
- Konfiguration der Spring Anwendung
- Einfaches Docker
- PostgreSQL

Spring Data JPA

Spring Data JPA

Konfiguration einer Spring Anwendung

Konfiguration einer Spring Anwendung

YAML File

```
1  server:
2    port: 8080
3
4  spring:
5    datasource:
6      url: jdbc:postgresql://localhost:5432/db
7      username: db_user
8      password: db_password
9    jpa:
10     hibernate:
11       ddl-auto: create-drop
12     show-sql: true
13     properties:
14       hibernate:
15         format_sql: true
```



YAML File

```
1 server:
2   port: 8080
```

YAML

- Konfiguration für den Spring server
- port bestimmt den Port der Spring Anwendung

YAML File

application.yml

YAML

```
1 spring:
2   datasource:
3     url: jdbc:postgresql://localhost:5432/db
4     username: db_user
5     password: db_password
```

- Konfiguration des Datenbankzugriffs
- url enthält URL zur gewünschten Datenbank.
- username: Username der ausgewählten Datenbank
- password: Passwort der ausgewählten Datenbank

YAML File

application.yml		YAML
1	jpa:	
2	hibernate:	
3	ddl-auto: create-drop	
4	show-sql: true	
5	properties:	
6	hibernate:	
7	format_sql: true	

- ddl-auto: Bestimmt, welche Operationen automatisch auf der Datenbank ausgeführt werden sollen
 - create-drop erschafft immer eine neue Datenbank, wenn die Spring Anwendung gestartet wird. Die Alte wird dabei gelöscht.
- **Wichtig:** Wenn keine embedded Datenbank genutzt wird, werden Datenbanken nicht automatisch erstellt.

YAML File

```
application.yml
1  jpa:
2    hibernate:
3      ddl-auto: create-drop
4      show-sql: true
5    properties:
6      hibernate:
7        format_sql: true
```

- show-sql: Ausgeführte SQL Befehle werden in der Konsole ausgegeben. Macht Debugging einfacher.

YAML File

application.yml		YAML
1	jpa:	
2	hibernate:	
3	ddl-auto:	create-drop
4	show-sql:	true
5	properties:	
6	hibernate:	
7	format_sql:	true

- format_sql: Formatiert den SQL Output in eine lesbarere Form

Properties File

application.properties		properties
1	server.port=8080	
2	spring.datasource.url=jdbc:postgresql://localhost:5432/db	
3	spring.datasource.username=db_user	
4	spring.datasource.password=db_password	
5	spring.jpa.hibernate.ddl-auto=create-drop	
6	spring.jpa.show-sql=true	
7	spring.jpa.properties.hibernate.format_sql=true	

Einführung in Docker

Dockerfile

Compose

```
1  services:
2    db:
3      image: postgres
4      container_name: postgres_db_container
5      environment:
6        - POSTGRES_USER=db_user
7        - POSTGRES_PASSWORD=db_password
8        - POSTGRES_DB=db
9      networks:
10       - application_network
11    ports:
12      - "8001:5432"
13  networks:
14    application_network:
15      driver: bridge
```

YAML

Compose

Services

```
1 services:
2   db:
3     # Service Definition
4     # Weitere Services
```

YAML

- In einer Map, unter dem Key `services`, werden alle Services definiert, die von der Compose gestartet werden sollen
- Name wird durch Key bestimmt
- Hier wäre der Name des Services: `db`

Compose

Services

```
1  db:
2    image: postgres
3    container_name: postgres_db_container
4    environment:
5      - POSTGRES_USER=db_user
6      - POSTGRES_PASSWORD=db_password
7      - POSTGRES_DB=db
8    networks:
9      - application_network
10   ports:
11     - "8001:5432"
```

YAML

- Einzelne Services werden durch Map unter dem Service-Key definiert

Compose

Services

```
1 services:
2   db:
3     image: postgres
```

YAML

- Key `image` bestimmt das Container-Image, dass hier genutzt werden soll
- Können vom System kommen oder zum Beispiel aus der Docker Registry
- Hier wird ein Image genutzt, welches für die PostgreSQL Datenbank zugeschnitten ist

https://hub.docker.com/_/postgres

Compose

Services

```
1 services:
2   db:
3     container_name: postgres_db_container
```

YAML

- Key `container_name` bestimmt den Namen des Containers

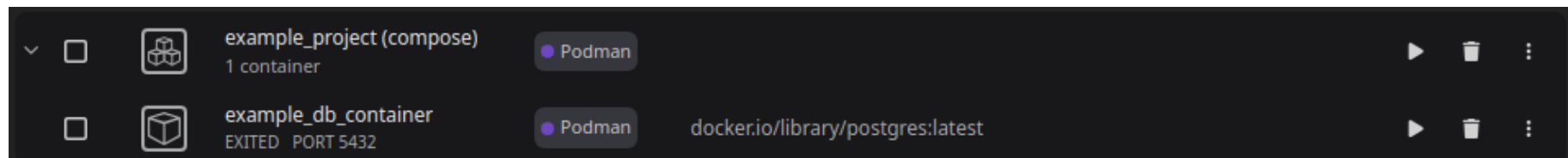


Figure 1: Aktuell nicht laufender Container in der Podman Desktop UI.
Der Container hat den Namen: `example_db_container`

Compose

Services - Environment Variablen

compose.yml

YAML

```
1 environment:
2   - POSTGRES_USER=db_user
3   - POSTGRES_PASSWORD=db_password
4   - POSTGRES_DB=db
```

application.yml

YAML

```
1 spring:
2   datasource:
3     url: jdbc:postgresql://localhost:5432/db
4     username: db_user
5     password: db_password
```

Compose

Services

```
1 services:
2   db:
3     networks:
4       - application_network
```

YAML

- Key networks bestimmt, zu welchen Netzwerken der Container zugeordnet werden soll
- Es können dabei mehrere Netzwerke genutzt werden
- Netzwerke werden in einer Map angegeben
- Hier wird nur das application_network genutzt

Compose

Services - Ports

```
1 services:
2   db:
3     ports:
4       - "8001:5432"
```

YAML

- Key ports bestimmt, welche Ports von dem Container exposed werden sollen
- Es können mehrere Ports in einer Map angegeben werden
- Jede Angabe besteht aus zwei Ports

Compose

Services - Ports

```
1 - "8001:5432"
```

YAML

Der linke Port ist der Port, der von außerhalb des Containers erreicht werden kann

Der rechte Port ist der Port, der innerhalb des Containers genutzt werden soll. Er wird auf den linked Port gemapped.

In diesem Beispiel hier, wird der default Port einer PostgreSQL Datenbank, **5432**, der innerhalb des Container genutzt wird, auf den Port **8001** gemapped. Die Datenbank kann somit außerhalb des Containers auf dem Port **8001** erreicht werden.

Compose

Services - Depends On

```
1 services:
2   db:
3     depends_on:
4       - other_container
```

YAML

- Key `depends_on` bestimmt, welche Container bereits laufen müssen, damit der aktuelle Container gestartet werden kann
- Es können mehrere Container in einer Map angegeben werden
- Der Name des Containers ist hier der Service Name

Compose

Netzwerke

```
1 networks:
2   application_network:
3     driver: bridge
```

YAML

- Definierung von Netzwerken für die Docker Container
- Container, die auf einem gemeinsamen Netzwerk sind, können untereinander kommunizieren
- bridge Netzwerk wird am Häufigsten genutzt
- Netzwerke werden als Map unter dem Key `networks` definiert
- Netzwerk-Name ist Key des jeweiligen Netzwerks
- `driver` Key bestimmt, welche Art von Netzwerk genutzt wird

Prüfung

Prüfung

MENSA IST VORBEI