Projekt M347

Eine Spring Boot Applikation und MongoDB mit Containern virtualisieren

Von Mademidda und Danijel

Inhaltsverzeichnis

Aufgabe

Unsere Aufgabe war, ein Spring Boot Applikation inklusive Datenbank mit Containern zu Virtualisieren. Wir haben uns dafür entschieden, unser Projekt mit dem Terminplaner aus dem letzten Modul 223 zu verwenden. Als Technologie wurde uns Docker vorgegeben. Da wir auch noch die Datenbank hatten, welche dazugehört, brauchten wir zwei Container. Einen Container für die Spring Applikation und einen für die Datenbank. Um mehrere Container für eine Anwendung zu Erstellen und zu Starten haben wir Docker Compose verwendet, welchen wir mit einem YAML-File konfiguriert haben.

Applikation

Unsere Applikation ist unser letztes Projekt, in welchem wir eine Spring Boot Applikation mit MongoDB als Datenbank gemacht haben. Mit dieser Applikation kann man Termine erstellen, einsehen und bearbeiten.

Wenn man die Seite aufmacht, kommt man auf eine Seite, wo man die drei verschiedenen Aktionen ausführen kann. Wenn man auf den Button «Continue wihtout Key» drückt, wird man auf eine Seite weitergeleitet, bei welcher man einen Termin erstellen kann. Sobald der Termin erstellt wurde, wird eine Bestätigungsseite angezeigt, auf der die Daten nochmals zusammengefasst werden. Zudem werden am Schluss noch zwei generierte Schlüssel angezeigt.

Der eine ist der public-key, mit welchem man die Reservationen nur ansehen kann. Mit dem zweiten Schlüssel, dem private-key, kann man die Termine bearbeiten oder löschen. Die Schlüssel kann man beliebig weitergeben und verwenden. Um die Schlüssel zu verwenden, gibt man sie bei der ersten Seite am entsprechenden Feld ein. Wenn der Schlüssel gefunden wurde, werden die Daten in einem Fenster aufgelistet. Ansonsten kommt eine Meldung, die besagt, dass der Schlüssel nicht gefunden wurde.

Funktionale Anforderungen

ID	Beschreibung	Muss/ Soll/ Kann	Verantwortung	Priorität
N01	**Reservation erstellen** Die Zimmer Reservation muss erstellt werden	Muss	Benutzer	1
N02	**Reservation löschen mit Private-Key** Die Zimmer Reservation muss gelöscht werden, indem man die Private-key eingibt.	Muss	Benutzer	1
N03	**Reservation lesen mit Public-Key** Die Zimmer Reservation muss gelesen werden, indem man die Public-key eingibt.	Muss	Eingeladene Benutzer	1
N04	**Reservation ändern mit Private-Key** Die Zimmer Reservation muss geändert werden, indem man die Private-key eingibt.	Muss	Benutzer	1
N05	**Aktuelle Datum** Reservierung für heute oder in der Zukunft erstellen	Soll	Benutzer	2
N06	**Optimale Zimmerreservierung** Die Zimmer sollten nicht am gleichen Tag und zur gleichen Zeit reserviert werden.	Soll	Benutzer	2
N07	**Default Seite** Wenn der pivate-key und der public-key nicht korrekt sind, kann man auf die Default-Seite wechseln.	Kann	Eingeladene Benutzer / Benutzer	3

Nicht Funktionale Anforderungen

Die Funktionen die der Benutzer braucht, um die Applikation mit Docker auszuführen:

Host

- Java Version 21
- Docker Version 26
- MongoDB Version 7

Endbenutzer

- Chromium basierter Browser

Resource

Die Anwendung „Terminplaner" basiert auf Spring Boot Thymeleaf und MongoDB.

Spring Boot

Das sind die Versionen unserer Anwendung von Spring Boot:

Java Version:	21
Model Version:	4.0.0
Spring Boot:	3.2.5

Dafür benötigen wir folgende Dependencies:

- **Spring Web**
  Behandlung eingehender HTTP-Anfragen und Weiterleitung an die entsprechenden Controller.

- **Spring Data MongoDB**
  Verwaltung der Datenpersistenz, so dass Controller mit MongoDB Repositories interagieren können.

- **Lombok**
  Es vereinfacht den Java-Code. Entities und andere Klassen werden kleiner und einfacher zu warten.

- **Thymeleaf**
  Das Tool rendert die Ansichten und generiert dynamisch HTML-Seiten auf Basis der von den Controllern bereitgestellten Daten.

MongoDB

Die Anwendung wird auf MongoDB Kompass zugreifen (mongodb://localhost:27017) und in der Dantebank zugreifen.

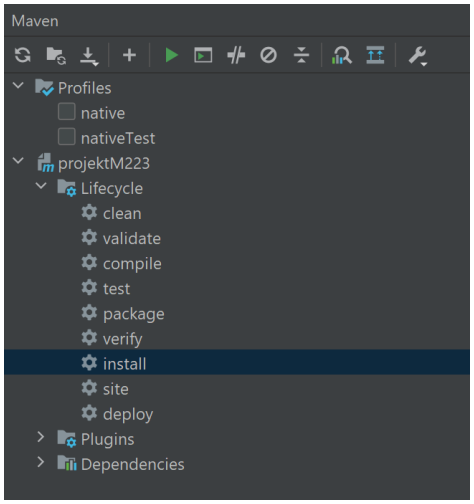
Der Terminplan benötigt nur eine Collection, nämlich *reservations*, da wir nur die Daten für die erstellten Reservationszimmer haben.

Datenbank	reservationManagementDb
Collection	reservations

Anleitung

Maven

Als aller erstes müssen wir das Maven in unserem Projekt installieren, um dessen Befehle ausführen zu können. Dabei müssen wir darauf achten, dass die Version vom SDK übereinstimmt. Wir haben die Version 21 von Eclipse Temurin verwendet.



Um Maven zu installieren, gehen wir in IntelliJ unter Maven und wählen «install» aus. Wenn alles richtig gemacht wurde und die Versionen kompatibel sind, sollte Maven ohne Fehler installiert sein. Hier haben wir auch den «clean» Befehl, den wir ausführen können, um den Maven Ordener zu «Reinigen». Dabei wird der Target Ordner mit dem app.jar File gelöscht, welches wir benötigen, um unsere Applikation in einen Docker Container zu packen. Wenn wir unsere Applikation dann wieder normal ausführen, werden die Dateien nochmals generiert.

Docker konfigurieren

Damit wir unsere Applikation in einem Docker Container ausführen können, müssen wir diesen zuerst konfigurieren und dann erstellen. Da wir hier zwei Container brauchen, erstellen wir noch ein YAML-File, um mehrere Container mit Docker Compose zu erstellen und zu Starten

Dockerfile

```
# Start with a base image containing Java runtime
FROM openjdk:21-jdk-slim

# Add a volume pointing to /tmp
VOLUME /tmp

# Copy the application's jar file into the container
COPY target/projektM223a.jar app.jar

# Run the jar file
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

Als erstes müssen wir ein Image auswählen. Dies ist der Wichtigste Teil, um unsere Applikation zu virtualisieren. Das Image kann man mit einem .iso File von einem Betriebssystem vergleichen. Es erstellt die Vorlage und Grundlage für den Container, auf dem die Applikation läuft. Wir haben uns für ein dünnes Image von OpenJDK in der Version 21 entschieden, weil es wenig Speicherplatz braucht und dadurch sehr effizient ist.

Im zweiten Schritt erstellen wir ein Volumen für unsere Applikation bzw. für unsere Datenbank, weil wir unsere Daten persistent im Docker speichern wollen. Ohne dem Volumen verschwinden unsere Daten beim Stoppen des Containers.

Als nächstes fügen wir unsere Applikation in den Container ein. Die gebaute und ausführbare Applikation ist in Java das .jar File. Es ist vergleichbar mit einem .exe file bei beispielsweise .NET Applikationen. Wir kopieren dieses File aus unserem Lokalen «Target» Ordner in den Container hinein.

Um sicherzustellen, dass unsere .jar Datei die aktuelle ist, führen wir den Maven clear Befehl aus um den Target Ordner, welcher die .jar Datei beinhaltet, löschen. Danach führen wir die Applikation nochmals normal aus, um den Ordner nochmals zu generieren.

Zum Schluss sagen wir dem Container, dass die app.jar Datei beim Starten ausgeführt werden soll.

YAML-File

Da wir mehrere Container brauchen, verwenden wir Docker Compose. Dafür brauchen wir ein YAML- File, um die Container zu konfigurieren.

```yaml
version: '3.8'

services:
  mongo:
    image: mongo:latest
    container_name: mongoreservationdb
    restart: always
    ports:
      - 27017:27017
    volumes:
      - mongo-data:/data/db

  app:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: reservation-app
    restart: always
    ports:
      - "8080:8080"
    depends_on:
      - mongo
    environment:
      SPRING_DATA_MONGODB_URI: mongodb://mongo:27017/reservationManagementDb

volumes:
  mongo-data:
```

Zuerst müssen wir die Version angeben. Danach geben wir die Informationen für die Datenbank als Service ein. Hier haben wir nur eine Datenbank und somit nur einen Service. Als Image haben wir einfach das neuste Image vom Offiziellen MongoDB. Wir benennen danach den Container und befehlen ihm, dass er sich neu startet, falls er von selber abstürzt. Als nächstes geben wir den Port an, welcher geöffnet sein soll. Auf diesem Port hört der Datenbank Server und arbeitet damit. Zum Schluss erstellen wir noch ein Volumen, in dem die Daten von der Datenbank persistent gespeichert werden sollen.

Als nächstes müssen wir noch die eigentliche Applikation konfigurieren. Vieles ist ähnlich wie bei den Services. Wir sagen dem container, dass er die Applikation mit allem builden soll. Der Punkt «.» bei «context» steht für alles. Danach geben wir das Dockerfile an welches wir bereits haben. Wir benennen den Container und befehlen ihm sich neu zu starten, falls er abstürzt, genau gleich wie beim Datenbank Container. Auch den Port geben wir an. Diesmal ist der Port der HTTP-Port 8080, weil unsere Applikation eine Web-Anwendung ist. Das besondere hierbei ist noch, dass wir noch die Dependencies angeben müssen. In diesem Fall «mongo» für unsere MongoDB. Wir geben am Schluss noch den Connection String von der Datenbank ein auf welchen unsere Applikation zugreifen soll. Der Connection String ist der gleiche wie unser Lokaler, jedoch ist «localhost» durch «mongo» ersetzt. Auch die Datenbank muss hier nach dem «/» angegeben werden.

Zum Schluss geben wir noch an, welche Volumen schlussendlich erstellt werden sollen.

Starten der Container

Wenn alles richtig gemacht wurde, sollten wir die Container erfolgreich builden und starten können. Als aller erstes führen wir den Maven Befehl «clear» aus um unser .jar File zu löschen. Danach starten wir unsere Applikation normal um das .jar File nochmals neu zu generieren.

```
PS C:\Users\danij\OneDrive - Kt. SG BLD\Dokumente\IMS\Modul 223\projektM223\projektM223> docker-compose build
```

Wir benutzen hier den Befehl «**docker-compose build**» um die Container erst mal zu bauen. Wichtig ist hierbei, dass wir uns auch wirklich im end-ordner befinden, in welchem unser Projekt liegt. Das häufigste Problem wird vom YAML-File verursacht. Es muss darauf geachtet werden, dass die Leerzeichen immer in zweierschritten stehen und nirgends unnötig stehen, da YAML sehr sensitiv ist.

```
PS C:\Users\danij\OneDrive - Kt. SG BLD\Dokumente\IMS\Modul 223\projektM223\projektM223> docker-compose up
```

Wenn das builden erfolgreich ausgeführt wurde, sollte man die Container mit dem Befehl «**docker-compose up**» in der Regel auch erfolgreich starten können.

Stoppen der Container

Um die Container zu stoppen, können wir den Befehl «**docker-compose down**» verwenden. Wenn wir die Container über das Terminal im IntelliJ gestartet wurden, müssen wir nur die Tastenkombination Controll + C drücken, um die Container zu stoppen. Auch das Stoppen im Docker Desktop ist eine Möglichkeit