# Opzetten van een Spring Boot voorbeeldapplicatie

In dit document licht ik kort toe hoe een Spring Boot REST service gebouwd kan worden op basis van een PostGreSQL database waarin vragen en antwoorden zijn opgeslagen.

PostgreSQL draait binnen een docker image.

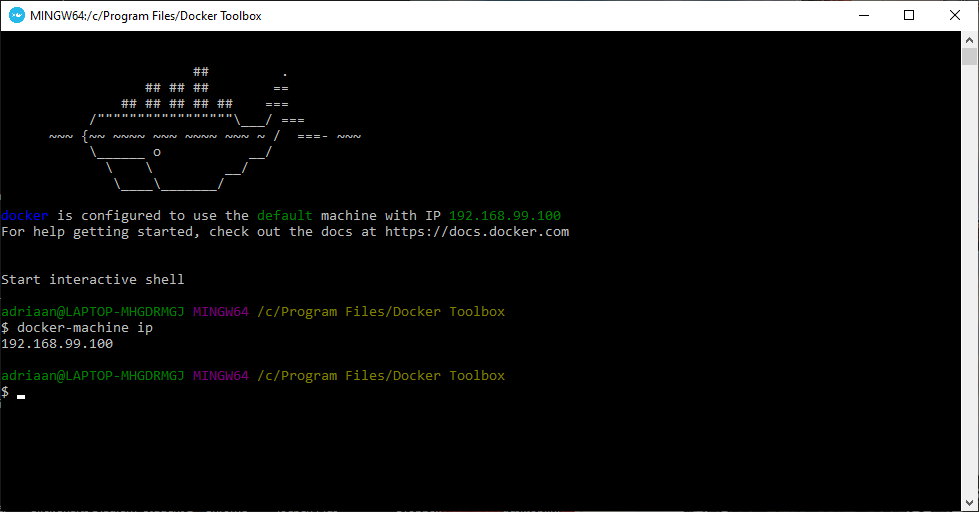
Opstarten van de Docker container met PostGreSQL:

1. **Opstarten van de docker engine en het achterhalen van het ip adres**

Vanwege de niet-pro versie van Windows 10 gebruik ik de docker toolbox in plaats van de “gewone” docker installatie. Gevolg is wel dat docker op een iets andere manier moet worden gebruikt. In plaats van “localhost” in het URL adres gebruik je een ander IP adres.

Om docker te starten gebruik je start.sh vanuit de docker toolbox home folder "C:\Program Files\Docker Toolbox". Dit venster gebruik je tevens om dit ip adres te achterhalen met het commando

docker-machine ip

Dit venster is overigens een linux venster.  
Verder mogen de docker commando’s worden gegeven vanuit een windows cmd venster.  


1. **Opstarten van de PostGreSQL container, connecten en testen van de connectie**

Het kan zijn dat er al een docker container draait maar gestopt is, vind dit uit met het commando vanuit een windows command venster:

docker container ls

* Is er geen container te vinden met als image ‘postgres’ : start de postgres container met het commando

docker run --name mypostgres --net host -e POSTGRES\_PASSWORD=mysecretpassword -d -p 5432:5432 postgres

* Is er wel zo’n container te vinden maar is deze gestopt:  
    
   CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

0805d5f596ff postgres "docker-entrypoint.s…" 44 hours ago Up 27 hours mypostgres01

De container heeft een container id, in dit geval 0805d5f596ff   
Herstart dan de container via

docker container start xxxx

waarbij xxxx het container id is, in dit geval 0805d5f596ff, of

docker container start *naam*

waarbij *naam* de naam is van de container, in dit geval mypostgres01

* Via een front end tool, in mijn geval TablePlus, kan een connectie gemaakt worden met een PostGreSQL database.  
  Gebruiker is postgres, password is mysecretpassword  
    
   Connecten kan ook vanuit de container zelf, op de volgende manier – in dit geval is geen password nodig.  
    
  Connecten van binnenuit de container met PostGreSQL via de command line SQL van PostGres.

|  |
| --- |
| Start nu van binnen deze container de postgres sql op. In ons geval is het container id gelijk aan 309a6a850721:  docker exec -it 309a6a850721 bash  Vanuit de bash prompt de postgres sql starten als gebruiker postgres  psql -U postgres  Dan connecten met de juiste database  \c mytest  Hier kun je de tabel infotable bekijken:  select \* from infotable;  Database aanmaken  create database mytest;  Table aanmaken  create table infotable(naam varchar(30));  insert into infotable(naam) values ('adriaan');  commit hoeft niet, autocommit. |

Connecten vanuit TablePlus: Maak een connectie aan met het eerder verkregen IP adres van de docker engine. In mijn geval is dat 192.168.99.100. NB vanuit het startvenster van TablePlus kun je connecties aanmaken, wijzigen door rechtermuisklik. NB. Bij het aanmaken van het docker image is als een poort aangegeven 5432 met port mapping 5432, voor PostGreSQL is dit de default.

|  |
| --- |
|  |

Via Test button kun je vaststellen of de verbinding werkt. Via Connect krijg je een browser van de tabellen. Via het SQL icoontje krijg je een sql venstertje op de geconnecte database. Je kunt via het database icoontje een andere database kiezen.

Opzetten van het Spring-Boot project  
Opzetten van het spring-boot project: hiervoor wordt gebruikgemaakt van een PostGreSQL database met questions en answers.

<https://www.callicoder.com/spring-boot-jpa-hibernate-postgresql-restful-crud-api-example/>

1. Maven en Spring Boot klaar maken van Eclipse  
     
   Eclipse zelf moet Maven – klaar gemaakt worden via een update (geloof ik). Misschien, maar dat kan ik ook niet meer precies herinneren, moet je Eclipse ook via een plugin Spring Boot klaar maken. Ik heb dat allemaal gedaan maar ben vergeten wat eraan te pas kwam. Het was niet zo moeilijk.
2. Bootstrapping van het project  
   vanuit de command line met

spring init --name=postgres-demo --dependencies=web,jpa,postgresql postgres-demo

of via de web initializer van spring boot: <https://start.spring.io/>

NB.

Het pom.xml bestand zorgt ervoor dat de applicatie door Maven wordt voorzien van de juiste libraries bij het voorbereiden, compileren en bouwen van het project. Bij gebruik van Spring Boot is het vitaal dat spring-boot deel uitmaakt van het pom.xml bestand.

1. Opzetten van een eerste simpel Spring Boot project in Eclipse.  
     
   Als je een maven project met spring-boot opzet in Eclipse zijn er een flink aantal punten waarop Spring Boot via “autoconfiguratie” moet zorgen voor de juiste referenties, het aanmaken van bepaalde beans en dergelijke – er wordt in de achtergrond heel wat voor je gedaan. Om te zorgen dat initiëel de code via Spring Boot kan worden voorbereid en worden gecompileerd moet allereerst de pom.xml file door Eclipse worden gebruikt om libraries op te halen via het internet. Klik op het project met de rechtermuis, zoek “Maven” op in de lijst op en klik hier met rechts op en selecteer “Update Project”. Dat voorbereiden en compileren gaat ook wel eens mis om verschillende redenen.

|  |
| --- |
| Greetingcontroller  Je kunt vanuit Eclipse zorgen dat er eerst een maven voorbeeldproject tot stand komt met “zo weinig mogelijk” en daarmee verder bouwen. Gebruik hiervoor bijvoorbeeld de greeting applicatie met spring-boot, deze gebruikt nog geen database:  Zie greetingcontroller.docx  <https://spring.io/guides/gs/rest-service/>  Bouw deze bij voorkeur met Maven. Zie het voorbeeld in de folder “greetingcontroller” en het bestand greetingcontroller.doc.  Initiëel zijn er door Eclipse allerlei fouten getoond in de code die “verdwijnen” door de spring-boot initialisatie van het project in Eclipse. Ik ben er nog steeds niet helemaal uit hoe / wanneer door Eclipse dit nu gebeurt via het voorbereiden van het spring-boot gebeuren op de achtergrond bij compilatie. Uiteindelijk, zo is de ervaring, komt dit goed nadat je hebt gecompileerd vanuit de configuratie van Maven in Eclipse via het goal “spring-boot:run”. Dit kan overigens ook via de command line via het tool mvnw via het commando mvnw spring-boot:run vanuit de hoofdfolder van het project. |

1. De applicatie hoort een *main* te heben waarin de SpringApplication wordt gestart

|  |
| --- |
| package hello;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  public class Application  {  public static void main(String[] args) {  SpringApplication.run(Application.class, args);  }  } |

1. Configureer de datasource en de database connectie in het application.properties bestand  
   Uiteraard moet je nu wel met spring-init gezorgd hebben voor de juiste opzet met de PostGreSQL database.

Het application.properties bestand bevat globale configuratie instellingen voor de applicatie. Het bevindt zich in de hoofdfolder van de applicatie. Initialiseer de connectie met de datasource (Postgres) en zet Hibernate als JPA engine properties voor Postgres SQL in application.properties. Voor het creëeren en bijwerken van de database moet de ddl-auto property van hibernate op “update” staan.

application.properties:

|  |
| --- |
| ## Spring DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)  spring.datasource.url=jdbc:postgresql://localhost:5432/postgres\_demo  spring.datasource.username= rajeevkumarsingh  spring.datasource.password=  # The SQL dialect makes Hibernate generate better SQL for the chosen database  spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect  # Hibernate ddl auto (create, create-drop, validate, update)  spring.jpa.hibernate.ddl-auto = update |

1. Breid de applicatie uit met een controller  
   De controller bevat de endpoints voor de webservices. Voorbeeld de controller van de Greeting demo:

|  |
| --- |
| package hello;  import java.util.concurrent.atomic.AtomicLong;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestParam;  import org.springframework.web.bind.annotation.RestController;  @RestController  public class GreetingController {  private static final String template = "Hello, %s!";  private final AtomicLong counter = new AtomicLong();  @RequestMapping("/greeting")  public Greeting greeting(@RequestParam(value="name", defaultValue="World") String name) {  return new Greeting(counter.incrementAndGet(),  String.format(template, name));  }  } |

1. Breid de applicatie uit met “entities”   
   Deze “java” entiteiten worden omgezet in tabellen en kolommen. Zie spring boot voor de annotaties:

|  |
| --- |
| @Entity  @Table(name = "questions")  public class Question extends AuditModel {  @Id  @GeneratedValue(generator = "question\_generator")  @SequenceGenerator(  name = "question\_generator",  sequenceName = "question\_sequence",  initialValue = 1000  )  private Long id;  @NotBlank  @Size(min = 3, max = 100)  private String title;  @Column(columnDefinition = "text")  private String description;  … |

1. Test de applicatie “droog” en kijk of de tabellen worden aangemaakt
2. Maak lagen aan via packages voor controller, services, model, repository en excepties

Scheid de applicatie in lagen. De bovenste laag is de controller laag, vervolgens services, vervolgens model, vervolgens repository en exceptions. “Gebruikt door” bepaalt de volgorde van boven naar beneden.

|  |
| --- |
| Controller |
| Services |
| Model |
| Repository |
| Exceptions |

Het repository package bevat de JPA repository met JPA operaties op de data. Zolang alleen van de “standaard JPA operaties” gebruik wordt gemaakt is de code sectie leeg. Voorbeeld:

|  |
| --- |
| **package** com.example.postgresdemo.repository;  **import** com.example.postgresdemo.model.Answer;  **import** org.springframework.data.jpa.repository.JpaRepository;  **import** org.springframework.stereotype.Repository;  **import** java.util.List;  @Repository  **public** **interface** AnswerRepository **extends** JpaRepository<Answer, Long> {  List<Answer> findByQuestionId(Long questionId);  } |

Het model package bevat de entiteiten klassen zoals Answer en Question bij de postgres-demo.  
Het controller package bevat de controller die zorgt voor het verwerken van de requests vanuit http.

Het exception package bevat exceptie klassen die fouten verwerken.  
Mogelijk is er nog een service laag met services. Er kunnen meerdere microservices zijn, ieder hebben ze hun eigen service klasse.

|  |
| --- |
|  |

1. Bouw en test de applicatie verder
2. Zet als laatste de application.properties file de property voor het wijzigen van de database uit