Onderzoeksverslag Distributed Enterprise Applications

2 APRIL

Klas:	ITA OOSE-A-f DEA
Versie:	1.0 2020/03/06
Autheur:	Robert Boudewijn
Studentnr:	631286
Docent:	Dhr. M. Portier

HAN_UNIVERSITY OF APPLIED SCIENCES

Inhoudsopgave

Inleiding

Voor het vak Distributed Enterprise Applications van het semester OOSE voeren we een onderzoek uit die voortbouwt op de programeeropdacht Spotitube. In deze programmeer opdacht is er een enterprise java applicatie gerealiseerd, die uit componenten bestaat. Het doel van deze onderzoeksopdracht is om een deel van de bestaande applicatie te vervangen door een alternatieve technologie.

Ik heb zelf besloten om onderzoek te doen naar het gebruik van docker. Met als hoofdvraag: **Hoe maak je java intergratie testen met behulp van Docker en Junit 5/jupiter**.

Onderzoeksvragen

Ik heb voor het onderzoek de volgende MoSCoW prioritering besloten:

Must have:

a. Wat is docker en waarvoor is het handig.
b. Hoe kan ik met maven en docker testen

Should have:

a. Welke configuratie heeft mijn docker file nodig voor DAO intergratie testen.

Could have:

a. Zou in Docker ook TomEE kunnen werken.

Wont have:

a. Hoe deploy je docker met TomEE naar een server zoals Azure.

Methodes

Ik wil de hoofdvraag en deelvragen onderzoeken door de ‘ICT research methods pack’ te gebruiken. Voor dit onderzoek ga ik de volgende methodes gebruiken:

Library

Literature study

Voor dit onderzoek wordt een literatuur studie uitgevoerd naar docker. Trefwoord docker zal waarschlijk veel gecombineerd worden met ‘maven’, ‘sql’, ‘mysql’, ‘tomEE’

en ‘Azure’. Er wordt goed gekeken naar de bron van de informatie. Zodat deze altijd betrouwbaar is en daarmee is er ook direct een kwaliteit eis.

Community research

Aangezien intergratie testen met docker vaker voorkomen, zullen forums en andere praat groepen veel informatie kunnen delen. Echter is het begrijpen van het gegeven antwoord hier erg belangrijk. Gewoon dingen kopiëren van stackOverflow zal niet goed zijn. Ook is het belangrijk om goed te kijken naar de opgehaalde informatie. ‘Is dit wel correct?’ ‘Is dit wel efficiënt?’ zijn vragen waar ik me tijdens dit soort onderzoek goed mee bezig moet houden.

Workshop

Prototyping

Het maken van prototypes geeft je direct inzicht in wat er gebeurd en wat er fout kan gaan. Het zorgt er voor dat je diepgang kan maken in je onderzoek. Wanneer je geen prototypes zou gebruiken zou je onderzoek oppervlakkig blijven. Omdat je er ‘vanuit gaat’ dat alles wel werkt. Echter is het belangrijk om te onthouden dat je het gebruikt opzoek naar het juiste product, en dat er dus meerdere prototypes kunnen zijn. Fixeer je niet direct op één uitwerking.

Field

Problem analysis.

Probleem analyse is handig om te gebruiken bij dit soort onderzoeken. Door eerst goed te defineren wat precies het probleem is en hoe je een verbeterde situatie ziet. Zorg je er voor dat je onderzoek in de goede richting blijft en je niet makkelijk afdwaalt.

1. Theoretisch kader

Waarom is het belangrijk om te intergratie testen?

Zelf vind ik het unit testen van een dataAccessObject een beetje vreemd. Het doel van een DAO is namelijk om data(gegevens) op tehalen van een database en dat in maten correct te verwerken. Dit is dus lastig te unit testen.

Doordat je met een unittest alleen één methode test en niet de intergratie met andere methodes moet je veel mocken, of kan je niet eens alle functies testen. Daar moet een andere manier voor zijn. Van dhr. M. Portier kreeg ik te horen dat je dan eigenlijk moet gaan intergratie testen. Dit zou handig zijn om samen met docker te doen, zodat je direct een database kan gebruiken.

Probleemstelling: Unittesten zijn niet dekkend genoeg voor DAO's
Doelstelling: De intergratie tussen het systeem en de database testen.
Hoofdvraag: 'Hoe maak je java intergratie testen met behulp van Docker en Junit 5/jupiter?'
Deelvragen:

1. 'Wat is docker en waarvoor is het handig?'
2. 'Hoe kan ik met maven en docker testen?'
3. 'Welke configuratie heeft mijn docker file nodig voor DAO intergratie testen?'

2. Docker

In dit hoofdstuk behandel ik de resultaten die ik heb gevonden bij de eerste deelvraag. Wat is docker en waarvoor is het handig.

2.1 Wat is docker

Docker is een open source platvorm voor het ontwikkelen en uitvoeren van applicaties (“Docker overview”, 2021). Docker is een soort virtual machine. Voordat je docker start maak je een docker file. Hierin beschrijf je wat je virtuele omgeving gaat doen en welk besturingssysteem het draait.

Een virtuele omgeving wordt ook wel een container genoemd. Op een container draait een image, dat kan je het beste vergelijken met een besturingssysteem.

Het grote voordeel van docker is dat je van je configuratie die beschreven staat in je dockerfile een image kan maken. En die image kunnen andere mensen dan weer uitvoeren. Hierdoor zorg je er voor dat je applicatie op elke computer het zelfde draait omdat je sowieso de juiste dependencys hebt, het niet uit maakt welk bestuuringssysteem je hebt, of je java geinstalleerd hebt of hoe je alles in gesteld hebt. Zolang je maar een computer hebt met de docker software (FireShip, 2020).

2.2 Hoe gebruik je docker precies

Je kan docker configureren door – zoals in hoofdstuk “2.1 Wat is docker” aangegeven – een dockerfile te maken. Een dockerfile is een bestand zonder extentie(‘.zip’,’.docx’) met de naam ‘docker file’.

In je docker file gebruik je de speciale docker syntax. Hierna zal ik stap voor stap toelichten hoe je een simpele dockerfile in elkaar kan zetten, deze info heb ik behaalt van FireShip (2020).

FROM: Hiermee kan je een image aangeven waar op de container zal werken. Bij image moet je denken aan een soort besturingssysteem.
WORKDIR: Met workdir geef je aan wat de working directory is van je app. Je kan het beschouwen als windows command **cd**.

Docker werkt in layers, elk command is een layer en layers worden gecashed. Hoe hoger je command staat hoe langer hij gecasched blijft zolang er maar boven die layer niets veranderd. Daarom kan het handig zijn om je dependency's zo hoog mogelijk in je docker file te zetten. Zodat deze bij code updates niet opnieuw gecashed hoeven te worden.

COPY: Met copy kan je bestanden uit je eigen directory kopiëren naar je docker directory. COPY heeft twee argumenten nodig, het eerste argument is het bestand op jou computer en het tweede argument is de locatie in de container.

RUN: Met run kan je commando's uitvoeren zoals je dat normaal ook over de terminal doet. Neem bijvoorbeeld npm install. Dit gaat overigens in een SHELL FORM.

Er is ook een Docker ignore file. Dit is net zoals een gitignore. Alle directory's die in de dockerignore staan zullen bij een copy worden overgeslagen. Handig! Vooral met je dependency's die je in een andere layer wilt hebben.

ENV: Met ENV can je environment variables maken voor in de docker container.
EXPOSE: In EXPOSE geef je de poorten op die de container mag doorlaten/ op mag luisteren. Hierdoor wordt je webserver bereikbaar.
CMD: CMD kan maar 1x worden gebruikt in een container. CMD verteld docker wat er precies moet worden uitgevoerd in de container. Dit commando wordt geschreven in EXEC FORM (["NPM", "START"]).

Dit is een voorbeeld dockerfile die ik heb geschreven om maven tests uit te laten voeren:

```
1. FROM maven:3.6.0-jdk-11-slim AS build
2. WORKDIR /
3. COPY pom.xml /
4. RUN mvn install
5.
6. COPY /src /src
7.
8. CMD ["mvn", "test"]
9.
```

In dit code voorbeeld pakt die een image van maven jdk 11. Wordt de pom gekopieerd, worden de dependency's geïnstalleerd en daarna pas de source code gekopieerd.

2.2.1 Building

Om je docker image te bouwen gebruik je in je terminal het commando docker build. Door -t toe te voegen kan je je image een naam geven (normaal is username/titel:versienummer). Na de naam geef je je directory op waar je container gemaakt moet worden. Door een punt (.) te gebruiken doe je dat in je huidige directory. (FireShip, 2020)

2.2.2 Docker uitvoeren

Gebruik Docker Run imageID/tagname om je image op te laten starten.
Wanneer je iets als een webserver wilt runnen in je docker moet je het port forwarden in je run command. Dat doe je met **docker run -p LocalPoort:ContainerPoort**.
Docker containers stoppen niet wanneer je de terminal sluit. Daarvoor moet je in de UI het stoppen. **PAS OP**! De data die is opgeslagen in de container wordt dan verwijderd. Met Volumes kan je er voor zorgen dat je bestanden wel bewaard blijven. Je maakt dan eigenlijk een gedeelde map met je container en je computer. (FireShip, 2020)

In de Docker applicatie kan je van elk van je containers gemakkelijk de commandline inzien en gebruiken alsof je in die bijvoorbeeld linux distro bent.

2.2.3 Docker compose

Met docker is het de bedoeling dat je maar 1 service draait per container. Daarom moet je database in een apparte container staan van je webserver. Hiervoor kan je gemakkelijk Docker compose gebruiken. Met docker comopse kan je defineren welke docker files op welke manier moeten worden uit gevoerd.

Je maakt een docker compose in een *docker-compose.yml*. Hierin zet je:

a. versie
b. services
c. volumes

In elke service kan je een appart component zetten. In het onderstaande voorbeeld bijvoorbeeld 'web' en 'db'. in deze componenten kan je de build locatie aangeven of de image die je wilt gebruiken, environment variables, poorten waarop ze moeten zijn geforward of gedeelde volumes.

docker-compose.yml

```
1. version: '3'
2. services:
3.   web:
4.     build: . # huidige directory
5.     ports:
6.       - "8080:8080" # LocalPoort:ContainerPoort
7.   db:
8.     image: "mysql"
9.     environment:
10.        MYSQL_ROOT_PASSWORD: root
11.     volumes:
12.       - db-data:/foo
13.
14.  volumes:
15.    db-data:
```

Met het commando **docker-compose up** kan je nu je alle containers tegerlijkertijd openen. en met het commando **docker-compose down** sluit je direct weer alle containers af. (FireShip, 2020)

3. Automatische maven tests

We kunnen natuurlijk een maven dockerfile maken en die tests laten uitvoeren en tegelijkertijd een mysql database laten opstarten. Maar dat is niet heel efficient. Mooier zou zijn wanneer je maven tests start, je mysql docker image automatisch start.

Dit zou je kunnen doen door een maven plugin. Na wat zoek werk heb ik de plugin van fabric8io (fabric8io/docker-maven-plugin) gevonden. Ik heb verschillende fora te horen gekregen dat dit goed werkt echter krijg ik het zelf niet werkende. Daarom heb ik mijn eigen script geschreven om de database aan te zetten.

Ik heb twee batch bestanden gemaakt. In deze bestanden wordt van de dockerfile een image gebouwd en automatisch die container opgestart. Hierna wacht het bestand 30 seconden tot dat de database is opgestart (zie figuur 1). In het andere bestand wordt de container gestopt en verwijderd, hiervoor heb ik gebruik gemaakt van (“Docker rm”, 2021).

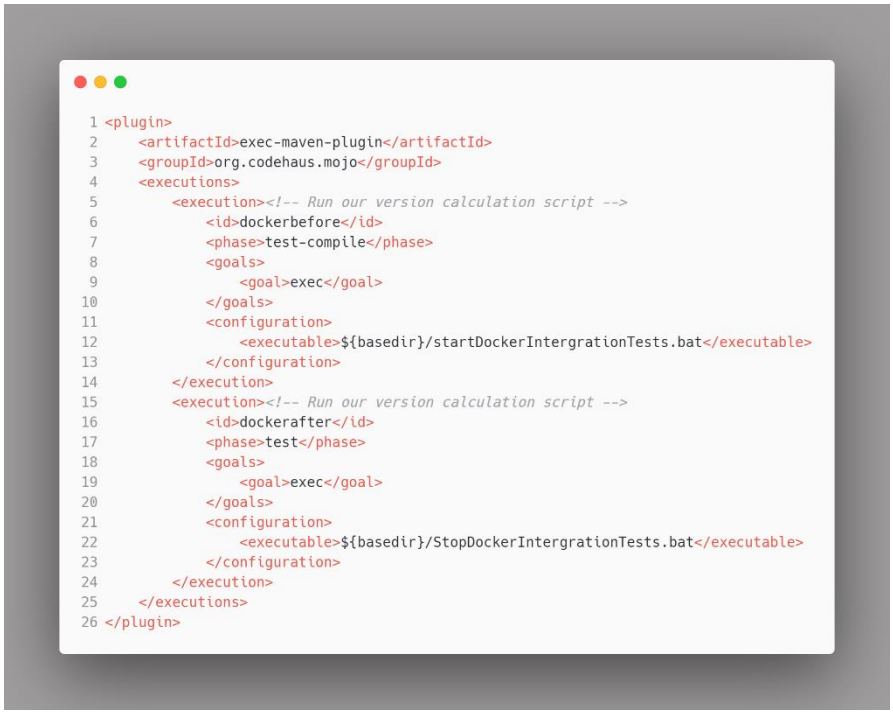
```
@echo off
docker build -t spotitube_intergrationtestdb .
docker run --name spotitube_intergrationtestdb -dp 3306:3306 spotitube_intergrationtestdb
Echo Waiting for mysql to start.
PING -n 30 127.0.0.1>nul
Echo Now we are ready

```

Figuur 1; start docker voor intergratietesten

Dit werkt goed. Maar jammer genoeg niet efficiënt.
Om deze bestanden te openen heb ik ‘org.codehaus.mojo, exec-maven-plugin plugin’ gebruikt. Hiermee kan je een fase aangeven en een executable invoeren. Voor spotitube zijn de volgende instellingen gebruikt:

Fase: test-compile, executable: ${basedir}/startDockerIntergrationTests.bat
Fase: test, executable: ${basedir}/StopDockerIntergrationTests.bat



```xml
<plugin>
    <artifactId>exec-maven-plugin</artifactId>
    <groupId>org.codehaus.mojo</groupId>
    <executions>
        <execution><!-- Run our version calculation script -->
            <id>dockerbefore</id>
            <phase>test-compile</phase>
            <goals>
                <goal>exec</goal>
            </goals>
            <configuration>
                <executable>${basedir}/startDockerIntergrationTests.bat</executable>
            </configuration>
        </execution>
        <execution><!-- Run our version calculation script -->
            <id>dockerafter</id>
            <phase>test</phase>
            <goals>
                <goal>exec</goal>
            </goals>
            <configuration>
                <executable>${basedir}/StopDockerIntergrationTests.bat</executable>
            </configuration>
        </execution>
    </executions>
</plugin>
```

Figuur 2; maven pulgin

3.1 integratie testen

Het maken van een integratietest is ook heel anders dan een gewone unittest. In dit hoofdstuk lopen we door het proces en alles dat verschillend is.

Een belangrijk punt van integratietest tussen software en database is dat de database voor elke nieuwe test weer schoon moet worden gemaakt. Dit kost erg veel tijd en is daarom ook één van de nadelen van integratietesten. Voor spotitube is er een database object file aangemaakt. In dit bestand staan alle test records voor de database. Hieruit kan je de objecten gebruiken om de database te vullen en te vergelijken met wat je terug krijgt van de methodes.

Ook is er een aparte TestConnection class. In de TestConnection kan je een nieuwe connectie maken met de database en is er een methode om de database weer terug te brengen naar begin staat.

3.1.1 Testen.

Voor elke test wordt de database eerst weer terug gebracht naar zijn oorspronkelijke staat. Dit wordt gedaan doormiddel van een @BeforeEach. Voor het testen wordt er gebruik gemaakt van de arrange, mock, act, assert methode. Hiermee worden er eerst

data neer gezet om mee te testen, daarna een mock gemaakt, daarna wordt de methode uitgevoerd en naderhand gebeurt echt het testen.

Het mocken is iets wat nu anders gaat gebeuren. Want we mocken nu de getConnection. Waarbij we dan onze eigen database connectie mee sturen. Hierbij moet je er wel aandenken dat maven een database dependency heeft.

3.2 Docker file voor integratietesten

In hoofdstuk 1 is aangegeven hoe je precies een dockerfile aanmaakt. Maar één van de onderzoeksvragen is ook wat is een goede dockerfile voor intergratie testen. Dat bespreken we in dit hoofdstuk.

Het lijkt natuurlijk het makkelijkste om gewoon een mysql image te pakken en die direct te gebruiken voor een test. Maar het is net niet zo gemakkelijk maar komt wel dicht bij. We moeten namelijk voor mysql een aantal dingen instellen. Denk hierbij aan een standaard database en een wachtwoord voor de root user.

Dit is gemakkelijk gedaan met het ENV keyword. De Dockerfile ziet er daarna als volgt uit:

```
1. FROM mysql
2. ENV MYSQL_ROOT_PASSWORD=root
3. ENV MYSQL_DATABASE=spotitube
```

Conclusie

Het maken van docker intergratie testen is heel makkelijk. Echter komt er meer bij kijken dan eerst bedacht.

Discussie

Bibliografie

Docker overview. (2021, 29 maart). Geraadpleegd op 30 maart 2021, van https://docs.docker.com/get-started/overview/#:%7E:text=Docker%20uses%20a%20client%2Dserver,and%20distributing%20your%20Docker%20containers.&text=The%20Docker%20client%20and%20daemon%20communicate%20using%20a%20REST%20API,sockets%20or%20a%20network%20interface.

FireShip. (2020, 24 augustus). *Learn Docker in 7 Easy Steps - Full Beginner's Tutorial* [Videobestand]. Geraadpleegd van https://www.youtube.com/watch?v=gAkwW2tuIqE&ab_channel=Fireship

Docker rm. (2021, 1 april). Geraadpleegd op 2 april 2021, van https://docs.docker.com/engine/reference/commandline/rm/

Gebruikte figuren

Bijlages:

Bijlage 1 bevindingen :

Dit zijn groffe aantekeningen van het uitvoeren van mijn onderzoek.

Om maven met Docker te gebruiken heb ik een bestaande image gebruikt. Jammer genoeg is er geen image voor de JDK 12. dus
heb ik in mijn POM mijn JDK veranderd naar 11. // outdated

Het eerste waar ik tegen aan liep was dat maven eerder klaar was dan de db. Daarvoor heb ik wait-for-it.sh gebruikt. Dit

moest ik nog wel een beetje tweaken. Omdat de standaard tijd van 15 seconde niet genoeg was voor de db om op te
starten.

Daarna liep ik tegen het probleem van de datasource aan. De datasource was iets dat altijd door tomee geinject werdt.
Echter kan dit nu niet gebeuren. Omdat DataSource zelf een interface is moest ik op zoek gaan naar een andere manier om
een data source te maken. Ik zit nu te denken aan een maven dependency.

Na een gesprek met Michel heeft hij mij geholpen dat ik de connection moet mocken. Of te wel een nieuwe connection aanmaken en deze bij de get connection mocken.

Ook hoeven mijn tests niet te draaien in docker. Hiervoor kan ik een maven dependency gebruiken die automatisch mijn container start. https://github.com/fabric8io/docker-maven-plugin

Om een database connectie te kunnen maken in je tests heb ik deze video gebruikt: https://www.youtube.com/watch?v=2i4t-SL1VsU&ab_channel=luv2code , uit eindelijk heb ik dit gebruit voor maven https://mvnrepository.com/artifact/mysql/mysql-connector-java/8.0.23

HAN_UNIVERSITY
OF APPLIED SCIENCES