Professionele Bachelor Toegepaste Informatica

Spring Boot vs Dropwizard

Brent Willems

Promotoren:

Niels Soeffers — ACA IT-Solutions
Nele Custers — Hogeschool PXL Hasselt

Bachelorpaper Academiejaar 2018-2019

Professionele Bachelor Toegepaste Informatica

Spring Boot vs Dropwizard

Brent Willems

Promotoren:

Niels Soeffers — ACA IT-Solutions
Nele Custers — Hogeschool PXL Hasselt

Bachelorpaper Academiejaar 2018-2019

Dankwoord

Na drie jaar studeren en veel nuttige zaken bij te leren aan de PXL kan ik eindelijk mijn eindwerk voorleggen. In deze drie jaar heb ik niet enkel mijn technische vaardigheden ontwikkeld, maar ook mijn persoonlijke. Ik wil graag stil staan bij iedereen die mij de afgelopen periode enorm heeft gesteund en geholpen.

In eerste plaats wil ik ACA IT-Solutions bedanken voor de stage die ik bij hun heb mogen lopen. Ook wil ik graag mijn stagebegeleiders, Niels Soeffers en Nele Custers bedanken voor de begeleiding, maar ook de rest van het team voor de hulp die ik gekregen heb als ik ergens mee vast zat.

Verder wil ik nog alle lectoren aan de PXL bedanken voor alles, maar ook de lectoren van mijn middelbare school VTI (nu Hast) om me goed voor te bereiden op mijn studies in het hoger. En niet te vergeten de steun van mijn vrienden, familie en mijn ouders die me altijd de keuze hebben gegeven om te doen wat ik graag doe.

Abstract

De klant IDEWE-vzw is een externe dienst voor preventie en bescherming op het werk in België. De projecten die hier voor IDEWE ontwikkeld zijn, maken deel uit van een groot systeem binnen IDEWE. Dit systeem zorgt ervoor dat werknemers van IDEWE hun werk kunnen registreren, afspraken kunnen inplannen en opvolgen.

Mijn project kadert in de Matrix-applicatie binnen dit systeem. Door middel van deze applicatie kan een IDEWE-medewerker nagaan hoeveel en welke dienstverleningen er reeds uitgevoerd zijn en welke nog gepland moeten worden bij een klant. De IDEWE-medewerker krijgt ook een overzicht van de medische onderzoeken die in andere toepassingen worden gepland. De Matrix-applicatie is ontworpen met het Dropwizard-framework. Binnen IDEWE is men voor verschillende applicaties Spring Boot gaan gebruiken. Om mee te zijn met de technologie migreer ik de Matrix-applicatie naar Spring Boot.

Tijdens deze migratie wordt er gewerkt met verschillende technologieën zoals Docker, Maven, Spring, JUnit en Mockito. Docker wordt lokaal gebruikt om de toepassingen op de development pc's op te starten. Maven wordt gebruikt voor het managen van dependencies en het project. JUnit voor het schrijven van unittesten en Mockito voor het mocken van data. Deze onderwerpen worden ook behandeld in het onderzoeksgedeelte van mijn eindwerk. De twee frameworks Spring Boot en Dropwizard worden met elkaar vergeleken. Er worden verschillende zaken behandeld: de populariteit van de frameworks, *dependency injection* en de mogelijkheid om andere technologieën te betrekken zijn hier een deel van. Spring is zeer populair onder *Java developers* terwijl Dropwizard niet zo populair is.

Inhoudsopgave

Inhoudsopgave

Lijst van gebruikte figuren

Lijst van gebruikte afkortingen

HTTP	HyperText Transfer Protocol
REST	Representational State Transfer
API	Application Programming Interfaces
CRUD	Create Read Update Delete
IDE	Integrated Development Environment
ORM	Object-Relational Mapping
JVM	Java Virtual Machine
OS	Operating System

Inleiding

In deze tijd zijn binnen de Java community zeer veel frameworks beschikbaar voor het maken van backend-applicaties. Toch is het belangrijk dat er een correcte keuze gemaakt wordt bij het uitkiezen van een framework. Er zijn verschillende belangrijke onderdelen waaraan gedacht moet worden. Zo is het bijvoorbeeld belangrijk dat een framework over goede documentatie beschikt. Ook is van belang dat een framework te integreren valt met verschillende tools die de dag van vandaag gebruikt worden. Denk aan dashboards om applicaties te monitoren en logs te analyseren.

De stageopdracht bij ACA IT-Solutions bestaat eruit om een applicatie die gebouwd is met het framework Dropwizard om te zetten naar een Spring Boot applicatie. Deze omzetting gebeurt omdat de klant ooit de beslissing heeft genomen Dropwizard te gebruiken, maar voor interne projecten ondertussen is overgeschakeld naar Spring Boot. Het onderzoek hangt hier nauw aan vast. Ik vergelijk deze 2 frameworks aan de hand van hun verschillende eigenschappen en stel een vergelijkingsmatrix op. Ik verwacht met deze opdracht veel bij te leren over Spring Boot, maar ook over de ontwikkeling van grotere applicaties in het algemeen. Verder hoop ik ook een kritische mening te vormen over beide frameworks.

In het eerste hoofdstuk wordt een inleiding over het bedrijf gegeven. Het tweede hoofdstuk bevat een beschrijving van technologieën die gebruikt zijn tijdens de uitwerking van de opdracht. Vervolgens wordt de uitwerking van de stageopdracht beschreven. Tenslotte wordt de onderzoeksvraag verdeeld onder verschillende onderwerpen gevolgd door een conclusie.

I. Stageverslag

1 Bedrijfsvoorstelling

1.1 Inleiding

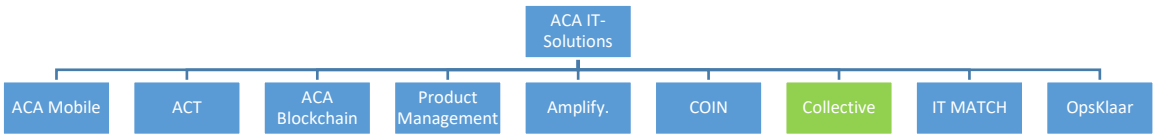
Figuur 1 logo ACA IT-Solutions

ACA IT-Solutions is een middelgroot bedrijf met ongeveer 200 werknemers dat sinds 1989 IT-services en Java-toepassingen op maat aanbiedt aan grote en kleine bedrijven. De omzet evolueerde op 5 jaar tijd van 5 naar 17 miljoen euro per jaar. Het hoofdkantoor is gelegen in Hasselt, verder heeft ACA IT-Solutions ook nog een kantoor in Olen en in Gent. Als podulaire organisatie bestaat ACA IT-Solutions uit verschillende businessunits (pods). Elk van deze units heeft zijn eigen expertise en kan als onafhankelijk bedrijf handelen. De expertise van meerdere services combineren kan echter voor een geweldige (IT) oplossing zorgen.

Bij ACA IT-Solutions wordt er gefocust op Next Generation Agile. Dit is een combinatie van Agile, Lean, Kanban, Scrum en XP. Zo wordt er gebruikgemaakt van kanbanborden om een duidelijk overzicht te hebben van elk project. Er wordt dagelijks een stand-up meeting gehouden aan het kanbanbord. Ook doen de werknemers aan *pair programming*. Dit wil zeggen dat ze samen achter 1 laptop werken om meer van elkaar te kunnen bijleren en minder fouten te maken.

1.2 Afdelingen

Hier licht ik kort de belangrijkste taken van iedere businessunit toe.



Figuur 2 organigram pods

1.2.1 ACA Mobile

ACA Mobile focust zich op de mobiele toepassingen voor bedrijven. Ze doen aan ontwikkeling & implementatie, beveiliging, analyses en ondersteuning. [1]

1.2.2 ACT

ACT houdt zich meer bezig met de grafische kant van de ontwikkeling en gebruikerservaring. Ze maken wireframes, logo's, grafische designs en meer. [2]

1.2.3 ACA Blockchain

ACA Blockchain is een afdeling die zich focust op de blockchain technologie. Zij zoeken uit of er nood is om deze technologie te implementeren bij de klant en helpen hun vervolgens verder door een applicatie uit te bouwen met deze technologie. Verder geven ze ook workshops en trainingen over blockchain. [3]

1.2.4 Product Management

Hoe kan je er zeker van zijn dat je het juiste product bouwt?

Software schrijven is duur, dus het loont om het vanaf de eerste keer juist te doen. De noden van je eindgebruikers ontdekken is niet gemakkelijk. Een goede oplossing bedenken is nog moeilijker. En die oplossing dan ook nog juist schalen is een vaardigheid op zich. Het is vaak een echte uitdaging voor teams om een bruikbaar, waardevol en haalbaar product te bouwen. [4]

1.2.5 Amplify.

Amplifly helpt klanten met allerlei vragen rond Atlassian producten. Verder geven ze ook nog trainingen. [5]

1.2.6 COIN

COIN maakt vooral projecten rond documenten, content en informatie gebaseerd rond producten zoal Alfresco, Liferay en AEM. [6]

1.2.7 Collectiv

Collectiv staat voor prachtige softwareoplossingen op maat en uitstekende consultancy. [7]

1.2.8 IT MATCH

IT MATCH staat voor uitstekende outsourcing & consultancy services aan bedrijven en freelancers. [8]

1.2.9 OpsKlaar

OpsKlaar houdt zich bezig met cloud migratie & optimalisatie, managed cloud, beveiliging en cloud automatisering & DevOps. [9]

1.3 Partners

Figuur 3 partner Alfresco

Alfresco is een documentmanagementsysteem. Bedrijven moeten verschillende belangrijke documenten beheren en daarom is het nodig dat deze documenten gestructureerd opgeslagen worden. Dit maakt het makkelijker deze documenten terug te vinden en te delen wanneer nodig. [10]

Figuur 4 partner Atlassian

Atlassian is het bedrijf achter de bekende tools: Jira, Confluence, Bitbukkit, Trello en nog enkele anderen. Door gebruik te maken van deze tools zorg je ervoor dat je als team beter kan samenwerken. Deze tools kunnen ook zeer goed geïntegreerd worden met elkaar en andere bestaande programma's zoals bijvoorbeeld Slack en VS Code.

Figuur 5 partner Liferay

Liferay portal is een framework om portals te maken voor zowel intern als extern gebruik. Verder biedt liferay ook nog andere services in de categorieën cloud, commerce en analytics.

Figuur 6 partner New Relic

New Relic wordt gebruikt voor het monitoren van applicaties zodat je een overzicht hebt over downtime en andere data. Ook biedt deze software de mogelijkheid om errors bij gebruikers te rapporteren.

Figuur 7 partner Oracle

"Oracle Corporation is vooral bekend vanwege zijn databasesysteem Oracle." [14]

Oracle is een zeer groot Amerikaans bedrijf met als belangrijkste productgroepen: databases, developer tools, E-Business Suite en Oracle Fusion Middleware.

Figuur 8 partner Backbase

Backbase is oorspronkelijk een framework om portals te bouwen, maar focust zich nu op de banksector. [15]

Figuur 9 partner Amazon Web Services

Amazon Web Services, ook bekend als AWS is een enorm groot cloud-platform. AWS biedt verschillende services die je als bedrijf betaalt op basis van verbruik. [16]

Figuur 10 partner SAP

SAP is een marktleider in software voor ondernemingstoepassingen en biedt software voor real-time bedrijfsprocessen. [19]

Figuur 11 partner Adobe

Adobe richt zich op het ontwikkelen van grafische software. Denk hierbij aan Photoshop. Verder heeft Adobe een groot assortiment softwarepakketten en services. [20]

2 Voorstelling stageopdracht

2.1 Probleemstelling

Het project situeert zich in het Collective team, deze pod bestaat uit ongeveer 80 werknemers en heeft verschillende projecten lopen. Collective is zelf nog onderverdeeld in verschillende teams. Het team waar ik deel van heb uitgemaakt houdt zich vooral bezig met de klant IDEWE. IDEWE-vzw is een Externe Dienst voor Preventie en Bescherming op het Werk in België. De projecten die hier voor IDEWE ontwikkeld zijn, maken deel uit van een groot systeem binnen IDEWE. Dit systeem zorgt ervoor dat werknemers van IDEWE hun werk kunnen registreren, afspraken kunnen inplannen en opvolgen.

Het project kadert in de Matrix applicatie binnen dit systeem. Door middel van deze applicatie kan een IDEWE-medewerker nagaan hoeveel en welke dienstverleningen er reeds zijn uitgevoerd en welke nog gepland moeten worden bij een klant. De IDEWE-medewerker krijgt ook een overzicht van de dienstverleningen die in andere toepassingen worden gepland. Dit zorgt ervoor dat IDEWE een beter zicht heeft op zijn klanten en makkelijker kan vaststellen welke diensten ze hun klanten nog kunnen aanbieden.

IDEWE heeft enkele jaren geleden de keuze gemaakt het Dropwizard-framework te gebruiken voor de applicaties binnen dit systeem, maar is ondertussen zelf overgeschakeld naar Spring Boot. De Matrix applicatie bestaat uit een Java-backend en een AngularJS-frontend. Er wordt ook gebruikgemaakt van technologieën zoals Hibernate, Maven en Docker. De taak in dit project is het migreren van de Matrix applicatie van Dropwizard naar Spring Boot. Hier hangt dan ook het onderzoek aan vast. In het onderzoek worden deze twee frameworks vergeleken door een vergelijkingsmatrix op te stellen.

2.2 Doelstellingen

Mijn doelstelling is om tijdens deze stage meer kennis op te doen over nieuwe technologieën zoals Docker, maar ook configuratie en werking van het Spring-framework.

2.2.1 Gebruikte technologieën

1.1.1.1 Hibernate

Figuur 12 logo Hibernate

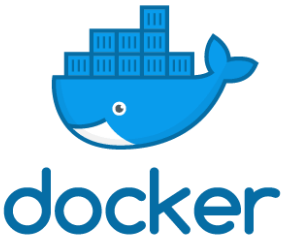
Hibernate ORM is een opensourceframework voor het koppelen van Java-klassen aan databasetabellen en transformeert Java-datatypes naar SQL-datatypes. ORM staat voor *Object-Relational Mapping*, dit is een programmeertechniek voor het converteren van data tussen relationele databases en objectgeoriënteerde programmeertalen. Door gebruik te maken van dit framework hoeft men zelf geen simpele query's meer te schrijven voor het ophalen, opslaan, verwijderen of bewerken van data, maar is er wel nog steeds de mogelijkheid om meer complexe query's te schrijven. [21]

2.1.1.1 Maven

Figuur 13 logo Maven

Maven is een software-projectmanagementtool ontwikkeld door Apache. Deze tool zorgt voor een gemakkelijker *build process* door gebruik te maken van een "Project Object Model" (POM). In deze POM definieer je afhankelijkheden van je project zoals externe modules en componenten. Buiten deze afhankelijkheden kan je ook nog vele andere zaken configureren met Maven, zoals het automatiseren van testen, uitrollen van code, en analyses van code.[22]

3.1.1.1 Docker



Figuur 14 logo Docker

Docker is een opensourceframework waarmee het mogelijk wordt een applicatie in een compacte, verplaatsbare container te verpakken. Zo wordt het installeren van een applicatie op een server even eenvoudig als het installeren van een mobiele app op je tablet of smartphone.

Docker voorkomt de overhead die ontstaat bij gebruik van meerdere virtuele systemen die elk een geheel eigen OS nodig hebben, wat niet alleen resources bespaart, maar applicaties sneller laat werken omdat de noodzaak van een hypervisor (een programma dat ervoor zorgt dat meerdere besturingssystemen en applicaties hardware van 1 computer kunnen delen) wegvalt. De technologie werkt prima lokaal, binnen (virtuele) datacenteromgevingen en binnen de cloud. Binnen mijn project wordt er automatisch een Docker-image gemaakt bij het builden van de code via Maven. [22]

4.1.1.1 Jersey

Figuur 15 logo Jersey

Jersey is een framework voor het ontwikkelen van RESTful Web Services in Java met een eigen API die gebouwd is op de JAX-RS specificatie maar verder nog extra functionaliteit voorziet. JAX-RS voorziet een deel gedefinieerde interface die Jersey implementeert. Dit zorgt ervoor dat er een standaard is in het ontwikkelen van RESTful Web Services.

5.1.1.1 Spring Boot

Figuur 16 logo Spring Boot

Spring Boot is een project dat gebouwd is op het Spring-framework en zorgt ervoor dat je zelf heel snel een Java-backend kan ontwikkelen.

6.1.1.1 AngularJS

Figuur 17 logo AngularJS

AngularJS is een front-end-framework dat wordt onderhouden door Google, dit is de voorganger van hetgeen we nu kennen als Angular. Hierin wordt het visuele van de applicatie ontwikkeld, de website dus. Zelf kom ik hier niet zo veel mee in contact omdat ik vooral aan de backend van de applicatie ga werken.

1.1.1.1 Junit

Figuur 18 logo JUnit

JUnit is een *unit testing framework* voor Java. Spring Boot maakt gebruik van dit framework. JUnit wordt gebruikt om testen te schrijven die de kwaliteit van de software verzekeren door bepaalde requirements af te checken. Unit-testen spelen een belangrijke rol in het ontwikkelen en onderhouden van software. Het doel is om te valideren dat elk stukje van de software werkt zoals verwacht. Unit-testen verhogen het vertrouwen in het wijzigen/onderhouden van code. Als er goede unit-testen geschreven worden en deze telkens als er code wordt gewijzigd, worden uitgevoerd, kunnen we eventuele gebreken die door wijzigingen zijn aangebracht, onmiddellijk opsporen.

2.1.1.1 Mockito

Figuur 19 logo Mockito

Mockito is een java-test-framework dat toelaat mock-objecten aan te maken. Door gebruik te maken van deze objecten kan je checken of er na het uitvoeren van een bepaalde methode een specifieke methode is uitgevoerd. Ook kan je met zo een mock-object ervoor zorgen dat een bepaalde service dummy data retourneert in plaats van echte data, dit laat toe dat je makkelijker rand-scenario's kan testen. Door gebruik te maken van Mockito hoef je geen tijd meer te spenderen om zelf mock-objecten te schrijven. Een ander voordeel is dat dit framework je toelaat heel makkelijk aan Test Driven Development te doen, Test Driven Development is een manier van software schrijven waar er eerst testen geschreven worden en dan aan de hand van deze testen de implementatie wordt geschreven.
Dit gebeurt door eerst de test te schrijven aan de hand van de requirements, hierna te kijken of de test faalt, dan code te schrijven en kijken of de test slaagt. Indien dit niet het geval is wordt de code bijgewerkt tot dat deze correct geschreven is en de test dus slaagt.

3.1.1.1 Thymeleaf

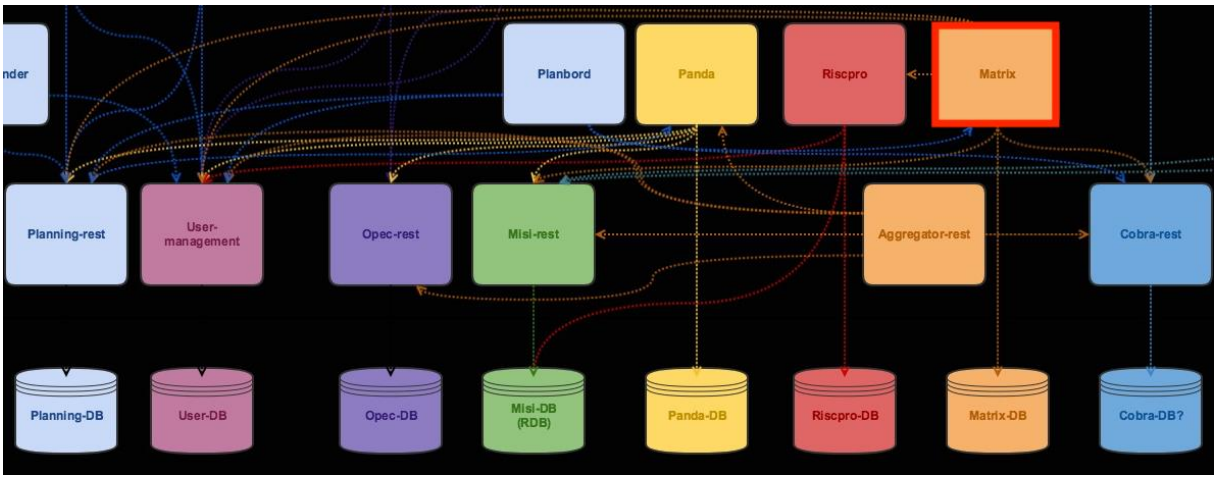
Figuur 20 logo Thymeleaf

Thymeleaf is een *templating engine*, een *templating engine* genereerd html-bestanden door parameters te verwerken in een template. In deze templates kan gebruikgemaakt worden van for-loops en if-statements om html-bestanden te genereren. Deze *templating engines* worden veel gebruikt in een MVC-structuur. In het project wordt deze technologie niet gebruikt voor het genereren van html-bestanden in de web-app, maar wel voor e-mails. Dit omdat er in de web-app gebruikgemaakt wordt van AngularJS. AngularJs is geen optie voor in emails, omdat er dan teveel javascript geladen moet worden.

3 Uitwerking stageopdracht

3.1 Initiële fase

Bij het uitwerken van de stageopdracht is eerst de onderzoeksvraag behandeld, omdat er op deze manier eerst wat opzoekwerk kon gebeuren over de frameworks die behandeld moesten worden. Ook heb ik meer info gekregen over het project en de applicatie eens doorlopen met mijn stagebegeleider om wat meer inzicht te krijgen in het project dat ik moet gaan omzetten.
Het eerste wat ik heb gedaan toen ik aan mijn opdracht begon was het opzetten van de ontwikkelomgeving. Om deze omgeving te kunnen opzetten heb ik eerst Docker moeten installeren en een *private git-repository* moeten ophalen. Een *git-repository* is een plaats waar code beheerd wordt. In mijn geval is deze *git-repository* enkel toegankelijk voor de werknemers die aan dit project moeten werken, als versioning systeem wordt er Git gebruikt en de repositories bevinden zich op een Bitbucket-server. Bitbucket kan net als GitHub in de cloud gehost worden of op een eigen server. Na de code binnen te halen heb ik ook eerst verschillende *Docker containers* moeten starten door een script te runnen dat alle nodige *images* gaat downloaden en op basis van deze images de *containers* gaat starten.
Tenslotte heb ik nog de nodige databanken moeten binnenhalen om hierna het project te kunnen opstarten. De databanken draaien ook in de Docker omgeving. Deze containers zijn aan elkaar gelinkt zodat de applicaties de nodige databanken en andere applicaties waarmee ze communiceren kunnen aanspreken.



Figuur 21 Schema applicatie

Het applicatie-landschap is gebaseerd op een microservice-architectuur. De matrix applicatie is hier één blokje van. Ook zien we in deze afbeelding dat elke service over een eigen databank beschikt. Dit omdat elke service dan onafhankelijk is en het belangrijk is om de services apart te kunnen releasen zonder invloed te hebben op de andere services. Dit geeft als voordeel dat de complexiteit beperkt wordt.

3.2 Maven

De eerste stap die ik gezet heb in het omzetten van de applicatie was het toevoegen en vervangen van de dependencies met Maven. Dit was zeker geen simpele taak omdat er veel dependencies waren die van elkaar afhankelijk zijn en conflicten geven als ze van versie afwijken. In het project werd er ook gebruikgemaakt van een *parent pom* file. Op deze manier kan ervoor gezorgd worden dat de versies van verschillende dependencies op een centrale plaats beheerd kunnen worden. Ook kan er in Maven gebruikgemaakt worden van *properties* die je declareert in de pom.xml file. Deze kan je zien als variabelen die je in Maven kan gebruiken. Maven zorgt ervoor dat tijdens het verpakken van de software de properties die gedefinieerd zijn in de pom.xml file worden ingevuld op de plaatsen waar je deze gebruikt. Wat wel belangrijk is om te weten is dat de syntax van deze properties wijzigt als je Spring Boot gebruikt. Dit is een van de problemen waar ik zelf wel even naar heb zitten zoeken. Bij het gebruik van Spring Boot vervang je de ${property} door @property@. Dit omdat Spring Boot zelf gebruikmaakt van deze notatie als je waardes wil ophalen die in de application.properties file gedefinieerd staan.

Tijdens het vervangen van de dependencies heb ik natuurlijk ook de nodige Spring-Boot dependencies toegevoegd. Om gebruik te maken van Spring Boot heb ik de nodige annotaties in de code toegevoegd.
Dropwizard maakt gebruik van een configuratiebestand in yaml-formaat, maar Idewe heeft ervoor gekozen om met property-files te werken.

3.3 Spring Configuratie

Nu dat dit alles gebeurd is kunnen we de applicatie al opstarten maar krijgen we errors omdat er verschillende beans niet gevonden worden. Dit heb ik opgelost door de beans te configureren, omdat er in de applicatie al gebruikgemaakt werd van Spring was er al configuratie voorzien, maar deze werd nog niet gebruikt door Spring Boot. Om gebruik te maken van deze configuratie heb ik verschillende configuratieklassen aangemaakt zoals in volgende figuur.

```
1  package be.idewe.matrix;
2
3  import ...
5
6  @Configuration
7  @ImportResource({"classpath:spring/domain-cobra-context.xml"})
8  public class CobraRestConfig {
9  }
```

Figuur 22 Configuratieklasse

De @Configuration annotatie zorgt er voor dat Spring Boot de nodige beans gaat aanmaken. In dit geval staan de beans gedefinieerd in de xml-file die we importeren. Ik heb op deze manier verschillende gelijkaardige klassen aangemaakt. Nadat alle configuratie klassen waren aangemaakt bleken er toch nog enkele zaken niet juist geconfigureerd te zijn, dus heb ik zelf de nodige beans gedefinieerd in de configuratie klasse door gebruik te maken van de @Bean annotatie. Achteraf heb ik nog verschillende xml-bestanden omgezet naar Java-configuratie.

Toen deze configuratie in orde was, ben ik begonnen met het omzetten van de Jersey *resources*. *Resources* in Jersey zijn gewoon RestControllers in Spring Boot. Bij deze omzetting heb ik de @RestController annotatie moeten toevoegen bij elke *resource*, ook heb ik verschillende andere annotaties moeten vervangen zoals @Path naar @RequestMapping om ervoor te zorgen dat de methodes worden afgehandeld als er een *request* komt op een bepaalde URL. De @QueryParam annotatie moest ik vervangen door de @RequestParam annotatie, deze wordt gebruikt voor het ophalen van parameters die worden meegegeven in de URL. Dit kan bijvoorbeeld zijn: https://website.be/home?country=BE&language=nl
De parameters in deze URL zijn dus *country* en *language*.
Verder gaan we ook gebruikmaken van de @Autowired annotatie. Deze zorgt er voor dat Spring Boot beheert welk object teruggegeven wordt als het in de code opgevraagd wordt. Sommige klassen zouden we in principe maar één keer moeten instantiëren, vaak zijn dit klassen die we gebruiken om objecten aan te maken of om data op te halen. Door gebruik te maken van deze annotatie gaat Spring Boot zorgen dat deze dan de juiste bean injecteert in het object.

Naast het omzetten van de *resources* naar *RestControllers* heb ik ook deze klassen *gerenamed* om de conventies te volgen. De xml-file met de configuratie is later nog omgezet naar Java-configuratie.

3.4 Testen

Voor deze controllers zijn uiteraard ook testen geschreven. Omdat we nu gebruikmaken van Spring Boot en niet meer van Jersey voor de controllers, gaan deze testen niet meer werken. Daarom heb ik ervoor gezorgd dat deze testen ook omgezet werden. Om de controllers te kunnen testen heb ik gebruikgemaakt van MockMvc en JUnit en Mockito. Met MockMvc kunnen we ervoor zorgen dat we de controllers kunnen testen zonder dat hiervoor de http-server moet worden opgestart. Ook gebruiken we voor de testklasse de annotatie @WebMvcTest(klassenaamController.class). Deze annotatie zorgt ervoor dat we binnen de klasse de juiste controller kunnen testen. Vervolgens kunnen we op ons MockMvc object de methode “perform” aanspreken om zo een *request* te sturen en op te lijsten wat we als antwoord terug willen krijgen.

Na deze omschakeling waren er al een deel van de testen in orde, maar voor de meeste testen heb ik nog verschillende fouten moeten oplossen zoal bijvoorbeeld een klasse die niet meer wordt gebruikt in de huidige versie van Spring vervangen, of configuratie van JSON-serialisatie. Nadat alle testen opgelost waren heb ik de applicatie kunnen opstarten, maar waren er nog problemen in verband met het tonen van de frontend-applicatie. Dit had te maken met het *serven* van de statische files. In Dropwizard moest dit pad expliciet geconfigureerd worden. We hebben ervoor gekozen om het default pad “/static” van Spring Boot te gebruiken. Hiervoor moesten we bestaande configuratie in Maven aanpassen.

3.5 Spring Security

Verder heb ik ook Spring Security geconfigureerd. Binnen de applicatie wordt er gebruikgemaakt van *http-basic* security. Dit wil zeggen dat er bij elke *request* naar de server de *credentials* van de gebruiker gecodeerd mee worden gestuurd. De applicatie is zo opgebouwd dat er in de frontend automatisch wordt uitgelogd als het een 401-response ontvangt. Als er uitgelogd moet worden doen we een *request* naar de server om uit te loggen die dan een 401-response terugstuurt. Authenticatie is echter niet het enige aspect dat te pas komt bij Spring Security. Naast de authenticatie hebben we ook de autorisatie. De autorisatie is het opleggen van restricties aan gebruikers. Binnen de applicatie kan een gebruiker verschillende rollen hebben. Deze rollen worden gebruikt om te bepalen welke acties deze gebruiker wel of niet mag uitvoeren.

```
1  @Override
2  protected void configure(final HttpSecurity http) throws Exception {
3      ...
4      .antMatchers("/auth/admin/*").hasRole("ADMIN")
5      .antMatchers("/auth/*").hasAnyRole("ADMIN","USER")
6      ...
7  }
```

Figuur 23 voorbeeld Spring Security

In deze figuur zie je de matcher “/auth/admin/*”. De * in deze URL betekent dat deze toegepast is op alle mappings op /auth/admin/ en alles wat hier nog achter zou komen. De *.hasRole(“ADMIN”)* zorgt ervoor dat deze *endpoints* enkel toegankelijk zijn voor gebruikers die ingelogd zijn en de rol admin hebben. Wel belangrijk om te weten is dat bij het gebruik van de hasRole-functie wordt gekeken of de rol het prefix “ROLE_” bevat. Als je al bestaande gebruikers hebt met rollen waar er niet elke rol begint met “ROLE_” gaat dit dus niet werken. Hier heb ik zelf even op vast gezeten omdat ik dacht dat er andere zaken niet in orde waren met mijn configuratie. De beste manier om dit op te lossen is door geen gebruik te maken van de hasRole-functie, maar de hasAuthority-functie. Deze functie doet exact hetzelfde, maar dan zonder te kijken naar deze prefix. Het is wel belangrijk dat de volgorde van deze matchers correct is. Zo begin je net als in het voorbeeld eerst met de meest specifieke mappings om vervolgens de security in te stellen op telkens meer algemene mappings. Spring Security maakt standaard ook gebruik van sessies. Bij de authenticatie tussen client en server krijg je een cookie van de server waar de ID van de sessie in staat. Zo hoef je niet telkens je *credentials* mee te sturen, maar stuur je deze cookie mee.

4 Besluit

Tijdens mijn stageperiode bij ACA IT-Solutions heb ik kunnen vaststellen dat er nog heel veel valt te leren als je pas in het werkveld terecht komt. De stageopdracht was zeker en vast een uitdaging. Eerder heb ik nooit in zo een groot project moeten werken, daarom was het in het begin best moeilijk om van start te geraken. Niet enkel de grootte van het project speelt hier een rol, maar ook het feit dat er gebruikgemaakt wordt van verschillende modules die door de klant geschreven zijn en de microservices-architectuur van de applicatie. Ook de hoeveelheid van configuratie kon soms wat overweldigend zijn. Tijdens de opleiding aan school hebben we enkel relatief kleine projecten moeten maken. Bij zulke kleine projecten is alle configuratie en structuur vaak zeer duidelijk, maar als je aan projecten moet werken die al jaren in ontwikkeling zijn wordt het moeilijker om mee te zijn. De aanwezigheid van testen heeft me ook laten inzien dat deze zeer belangrijk zijn. Door deze testen zijn er tijdens het omzetten van de applicatie verschillende dingen opgemerkt die anders waarschijnlijk later pas aan het licht zouden komen.

Er waren veel problemen die ik zelf heb kunnen oplossen, maar als ik echt te lang vast zat op een probleem heb ik wel hulp gevraagd aan mijn stagebegeleider of aan iemand anders in het team. Ik vind zeker en vast dat ik altijd goed ben geholpen. Als ik een probleem tegen kwam probeer ik altijd eerst te begrijpen wat het probleem juist is, misschien dat ik dan al weet waar het aan kan liggen. Als dit niet voldoende is zoek ik op internet naar het probleem en vind ik meestal wel een oplossing.
Dit is echter wel iets waar ik nog aan moet werken, het vinden van een goede balans tussen een vraag stellen en zelf iets opzoeken. Ik heb veel dingen zelf opgezocht, maar ik ben van mening dat ik meer vragen had moeten stellen dan was ik misschien sneller geholpen met bepaalde problemen.

Wat ik ergens wel jammer vind aan de opdracht is dat het geen opdracht is waar ik zelf iets kon creëren. Ik ben van mening dat als je aan iets werkt waar je meer resultaat ziet van wat je doet, dit je ook meer voldoening brengt. Tijdens deze opdracht heb ik echter wat meer onder de motorkap gewerkt, maar wel veel kunnen leren, vooral in verband met Maven, Spring en Docker. Buiten het werken met de technologieën heb ik ook verschillende dingen geleerd in verband met het werken in IntelliJ. Denk bijvoorbeeld aan het gebruik maken van verschillende shortcuts om bepaalde dingen veel sneller te kunnen opzoeken. Buiten deze shortcuts heb ik ook geleerd dat je remote kan debuggen. Dit is iets wat ik voordien niet wist, maar wel zeer handig is als je je applicaties in een Docker-omgeving draait.

II. Onderzoekstopic

1 Wat zijn de verschillen tussen Spring Boot en Dropwizard

1.1 Spring Boot

Figuur 24 Spring Boot logo

> "Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".
>
> We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need very little Spring configuration." [23]

Spring Boot is een project dat gebouwd is op het Spring-framework en zorgt ervoor dat je zelf heel snel een backend kan ontwikkelen. Oorspronkelijk moest je in het Spring-framework alles zelf configureren, dit zorgde ervoor dat je redelijk wat configuratiebestanden had. Zeker in grote projecten is dit niet altijd even evident. Spring Boot lost dit op door zelf alles te configureren aan de hand van de dependencies die je gebruikt.

1.2 Dropwizard

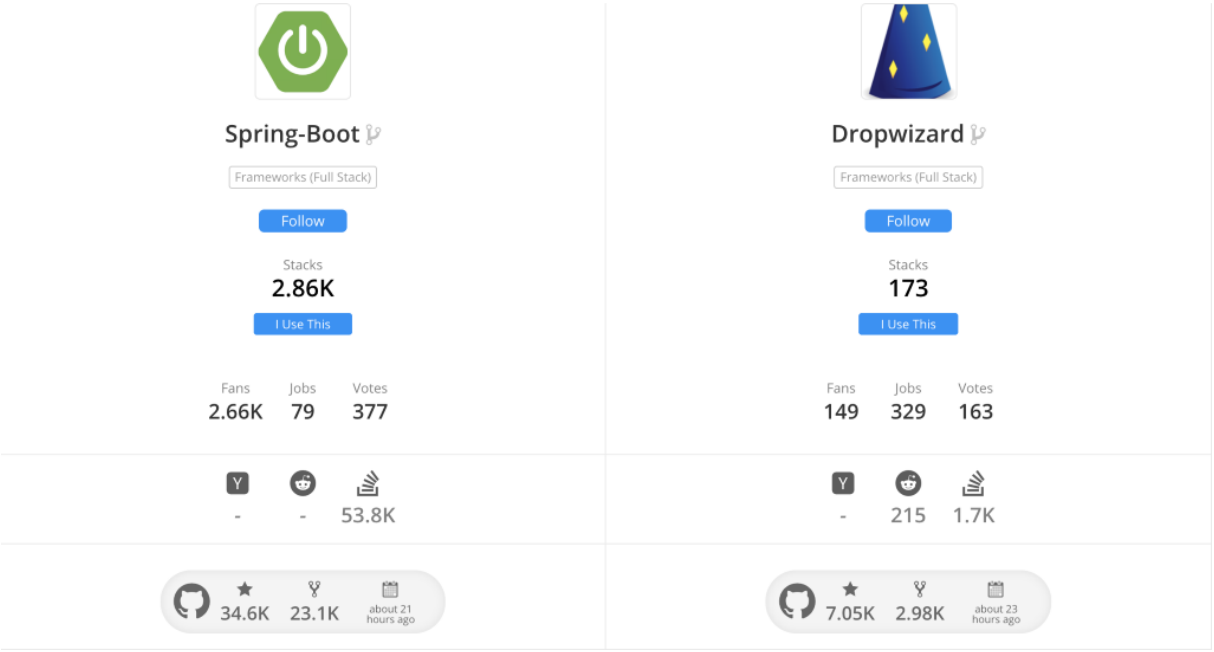
Figuur 25 Dropwizard logo

> "Dropwizard is a Java framework for developing ops-friendly, high-performance, RESTful web services." [24]

Dropwizard is een framework dat verschillende andere frameworks en *libraries* bevat om op deze manier snel een nieuw project op te kunnen zetten geschikt voor productie. Dropwizard levert standaard al geavanceerde configuratie, metrieken, *logging* en meer.

1.3 Vergelijking

1.3.1 Populariteit



Figuur 26 Stackshare vergelijking Spring Boot Dropwizard

In deze vergelijking zien we duidelijk dat Spring Boot populairder is dan Dropwizard. Spring Boot heeft een veel grotere community, dit kunnen we bijvoorbeeld al zien aan het aantal vragen die gesteld zijn op het platform Stack Overflow. Zo zien we dat Spring Boot een 54 000 tal vragen op Stack Overflow heeft en Dropwizard maar 1700. [25]
Dit maakt Spring Boot al direct een stuk interessanter, als je een probleem hebt met de implementatie of configuratie van een onderdeel van de applicatie is de kans zeer groot dat je snel een antwoord op je vraag zal terugvinden. Dit in tegenstelling tot het minder populaire framework Dropwizard, waar je waarschijnlijk nog zal moeten wachten tot er iemand is die je een antwoord kan geven terwijl je dan misschien nog steeds niet verder geholpen bent. Ook is de kans groot dat nieuw personeel al kennis heeft van Spring Boot terwijl dit bij dropwizard niet het geval is. De populariteit van een framework lijkt mij een zeer belangrijke eigenschap. Dit omdat het los van de vele bronnen informatie die je online terugvindt, ook op een of andere manier bevestigt dat het de eigenschappen bevat die nodig zijn om succesvol te zijn. Los van de populariteit staat er op de site van Spring ook zeer duidelijke documentatie, nuttige handleidingen en tutorials.

1.3.2 HTTP Server

Spring Boot start je applicatie op in een Tomcat container, maar Spring Boot biedt ook de mogelijkheid om een Jetty container of Undertow container te gebruiken. Deze containers bevatten allen een HTTP-server zodat er via het HTTP-protocol gecommuniceerd kan worden met de applicatie. De container zorgt ervoor dat inkomende HTTP-aanvragen automatisch verwerkt zodat de geïmplementeerde code is uitgevoerd wordt. Dat Spring Boot deze server zelf start met je applicatie is een zeer groot voordeel, omdat je anders zelf eerst een keuze moest maken welke server te gebruiken, deze downloaden, installeren, configureren, je applicatie *packagen* en deze op de server zetten.

Dropwizard gebruikt Jetty als container, maar Jetty is dan ook de enige container die ondersteund wordt door Dropwizard.

Voor de REST API gebruikt Spring Boot standaard zijn eigen Spring implementatie, maar er kan ook gekozen worden om met Jersey te werken terwijl er bij Dropwizard enkel Jersey ondersteund wordt.

1.3.3 Dependency Injection

Een ander groot verschil tussen deze frameworks is *dependency injection*. Dit is een geavanceerd *design pattern* dat het mogelijk maakt om op een modulaire manier toegang te hebben tot afhankelijkheden (andere delen van de code, bijvoorbeeld andere klassen) zonder dat deze hard gecodeerd zijn.
In Spring Boot zit *dependency injection* standaard al ingebakken in het framework, maar bij Dropwizard is dit niet het geval. Dit zorgt ervoor dat je een externe *library* moet importeren. Dat is een nadeel omdat ik persoonlijk vind dat een framework ervoor zou moeten zorgen dat je kwalitatieve code schrijft door te forceren om bepaalde belangrijke concepten toe te passen zonder hiervoor meer te moeten doen.

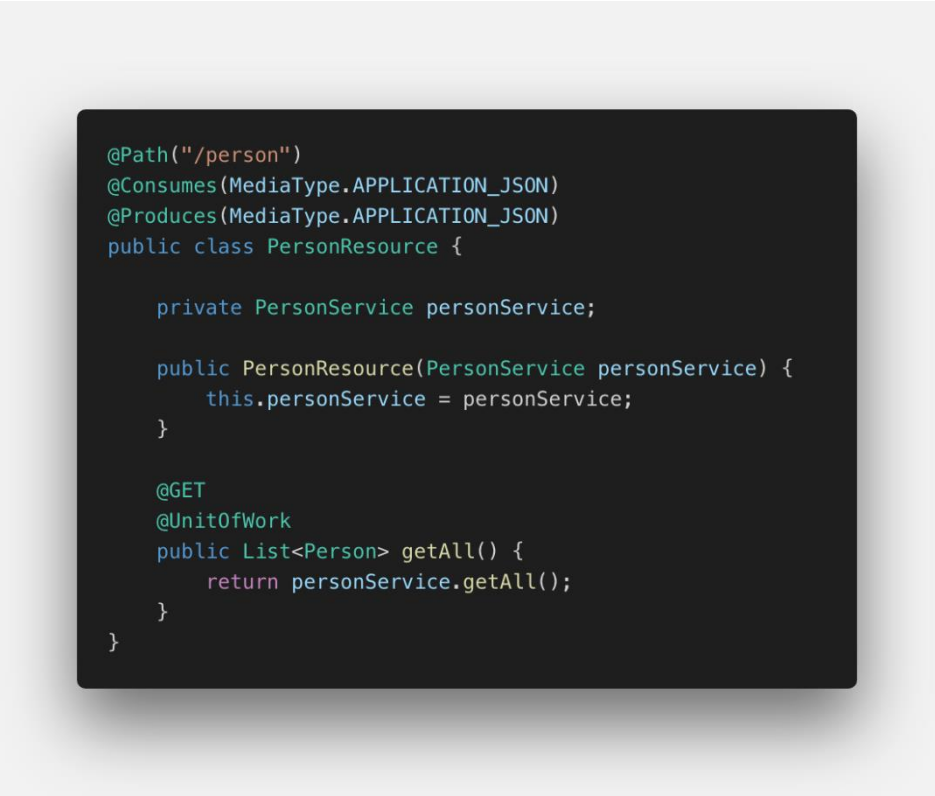
In het Spring Boot-framework is het heel gemakkelijk om aan *dependency injection* te doen, je hoeft alleen de bean te definiëren en de annotatie @Autowired te gebruiken om Spring Boot de rest te laten doen.

```java
@RestController
@RequestMapping("/person")
public class ExampleController {

    @Autowired
    private PersonService personService;

    @GetMapping
    @Transactional
    public ResponseEntity getAllPersons(){
            return ResponseEntity.ok().body(personService.getAll());
    }

}
```

Figuur 27 voorbeeld controller Spring Boot

Hier zien we een controller klasse die dient om de *requests* te beantwoorden met de juiste data. We zien hier dat de @Autowired gebruikt wordt om een implementatie van een personService te injecteren, deze service is een aparte klasse voor de businesslogica en de *repository* die in staat voor de CRUD-operaties, CRUD staat voor *Create Read Update Delete.* Deze service is dus aanspreekbaar voor het aanmaken, opvragen, bewerken en verwijderen van de persoon-objecten.

In Dropwizard is dit jammer genoeg niet ingebakken in het framework.



```java
@Path("/person")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public class PersonResource {

    private PersonService personService;

    public PersonResource(PersonService personService) {
        this.personService = personService;
    }

    @GET
    @UnitOfWork
    public List<Person> getAll() {
        return personService.getAll();
    }
}
```

Figuur 28 voorbeeld controller in Dropwizard

Deze *resource* klasse heeft hetzelfde doel als de controller klasse in het vorige voorbeeld, maar deze werkt nog niet zonder extra configuratie binnen Dropwizard.

```java
@Override
private void run(final TestConfiguration testConfiguration, final Environment environment) {
    final PersonSerivce personSerivce = new PersonService();
    final PersonResource personResource = new PersonResource();
    environment.jersey().register(personResource);
}
```

Figuur 29 voorbeeld configuratie Dropwizard

Je zal hier in de run methode van de applicatie klasse zelf nog de objecten moeten aanmaken, aan elkaar linken en registreren. Er zullen vaak meerdere zaken moeten doorgegeven worden aan verschillende klassen, dit kan bij grote projecten zeer onoverzichtelijk worden. Dit is een groot nadeel aan Dropwizard, omdat er bij Spring Boot voor deze simpele gevallen geen configuratie nodig is. Voor meer complexe gevallen kan er binnen Spring Boot wel nog extra configuratie plaatsvinden. Het enige nadeel van Spring Boot is dat er van alles automatisch gedaan wordt wat ervoor kan zorgen dat nieuwkomers binnen het framework niet weten wat

er aan het gebeuren is. Tegelijkertijd is dit dan weer een groot voordeel omdat er juist minder configuratie dient te gebeuren en er enkel voor speciale gevallen iets dient geconfigureerd te worden.

Om in Dropwizard *dependencies* te kunnen injecteren bestaan er *dependency injection frameworks*, zo kwam ik tijdens mijn onderzoek vaak het framework Guice [26] tegen, dit is een framework dat gemaakt is door Google.

Verder is het ook mogelijk om Spring te gebruiken in je Dropwizard applicatie. [27]
Op die manier kan je in Dropwizard ook gebruikmaken van de @Autowired annotatie en los je het probleem van *dependency injection* ook op.

1.3.4 Logging

Spring Boot ondersteunt een groot aantal verschillende logging frameworks. Spring Boot gebruikt standaard Logback, maar biedt ook ondersteuning voor Log4j, Log4j2, SLF4J en Apache Commons Logging terwijl Dropwizard enkel Logback en Slf4j ondersteunt.

Ook kan er bij Spring Boot gebruikgemaakt worden van de *sping-cloud-starter-sleuth* dependency. Eens deze dependency is toegevoegd dan is Sleuth geactiveerd in de applicatie. Sleuth gaat ervoor zorgen dat alle binnenkomende requesten en alle uitgaande responses een id krijgen zodat de logs getraceerd kunnen worden. Dit is zeer handig in applicaties die gebaseerd zijn op een microservice-architectuur, omdat er in deze architectuur operaties uitgevoerd worden tussen verschillende systemen. Via deze techniek kan er een correlatie gebeuren tussen de verschillende services in de logbestanden.

1.3.5 Metrics

Metrics zijn metingen van de applicatie. Deze metingen hebben betrekking tot het geheugenverbruik van de applicatie, CPU-gebruik, reactietijden, errors, enz. Voor een applicatie die in productie staat is het belangrijk om deze data beschikbaar te hebben. Moesten we deze data niet ter beschikking hebben zouden we het moeilijker hebben om te achterhalen wat er fout is gegaan, terwijl je dit zo snel mogelijk zou willen weten als je applicatie in productie staat.

Dropwizard gebruikt zijn eigen *library* genaamd Metrics [28] om deze metingen beschikbaar te stellen. Deze *library* is op zijn eigen al bijna even bekend als dropwizard zelf. Deze *library* wordt ook verschillende keren vermeld in de (iets oudere) documentatie van Spring Boot. [29]
Door middel van de Metrics-*library* kun je via (onder andere) een REST-API statistieken opvragen over de gezondheid van je applicatie: hoe staat het met het geheugengebruik van de JVM, welke HTTP statuscodes worden er zoal door je REST-*endpoints* teruggegeven, hoeveel ERROR-events worden per seconde naar de logs geschreven, en noem maar op. Je wordt ook aangemoedigd om zelf health checks toe te voegen. Health checks zijn kleine testen die laten zien of je applicatie goed werkt in de productieomgeving. Deze testen controleren bijvoorbeeld of er een databaseconnectie beschikbaar is.

Spring Boot heeft hier Actuator voor, dit is een sub-project van Spring Boot dat ervoor zorgt dat je tools beschikbaar hebt voor productieapplicaties. Eens je de Actuator *dependency* hebt toegevoegd, dan worden er automatisch gegevens verzameld in verband met verbruik,

performance, ... Deze gegevens zijn dan te vinden op het /actuator *endpoint* van je applicatie, maar je kan in je properties-file nog configureren om over meer of minder gegevens te beschikken. Verder kan je best ook nog deze *endpoints* beveiligen, zodat niet iedereen aan deze data kan. Zelf ga je nauwelijks naar deze *endpoints* surfen, maar ga je eerder een externe monitoring tool gebruiken die gemaakt is om de *endpoints* uit te lezen en deze data overzichtelijk te maken door bijvoorbeeld bepaalde data in grafieken te plotten.

Spring Boot voorziet in de documentatie integratie met maar liefst 17 verschillende monitoringsystemen [30], terwijl er in de documentatie van Dropwizard niet zo veel is terug te vinden over deze integraties. Zo volstaat het één enkele *dependency* toe te voegen aan de pom.xml file, de url van de monitoring server en eventuele API-key toe te voegen aan de properties-file om al deze data beschikbaar te hebben in een monitoringsplatform naar keuze. Het gebruik van deze monitoringsystemen is zeer nuttig, zeker als er meerdere applicaties onderhouden worden. Zo heb je als team altijd een mooi overzichtelijk dashboard van alle applicaties waar je logs, statistieken, *threads* kan bekijken, maar ook omgevingsvariabelen kan aanpassen.

Dropwizard voorziet in de documentatie integratie met maar 2 monitoringsystemen. Deze worden op ongeveer dezelfde manier gekoppeld dan bij Spring Boot. Als je zelf opzoekwerk gaat doen ga je waarschijnlijk wel nog andere monitoringsystemen vinden die je kan gebruiken voor je dropwizard applicatie. Zo heb je ook nog DataDog, een bekend monitoringsplatform die wel in de documentatie van Spring Boot terug te vinden is. Dit platform blijkt ook vrij simpel te gebruiken binnen dropwizard [31], maar hiervan vinden we niets terug in de documentatie van dropwizard.

1.3.6 Testing

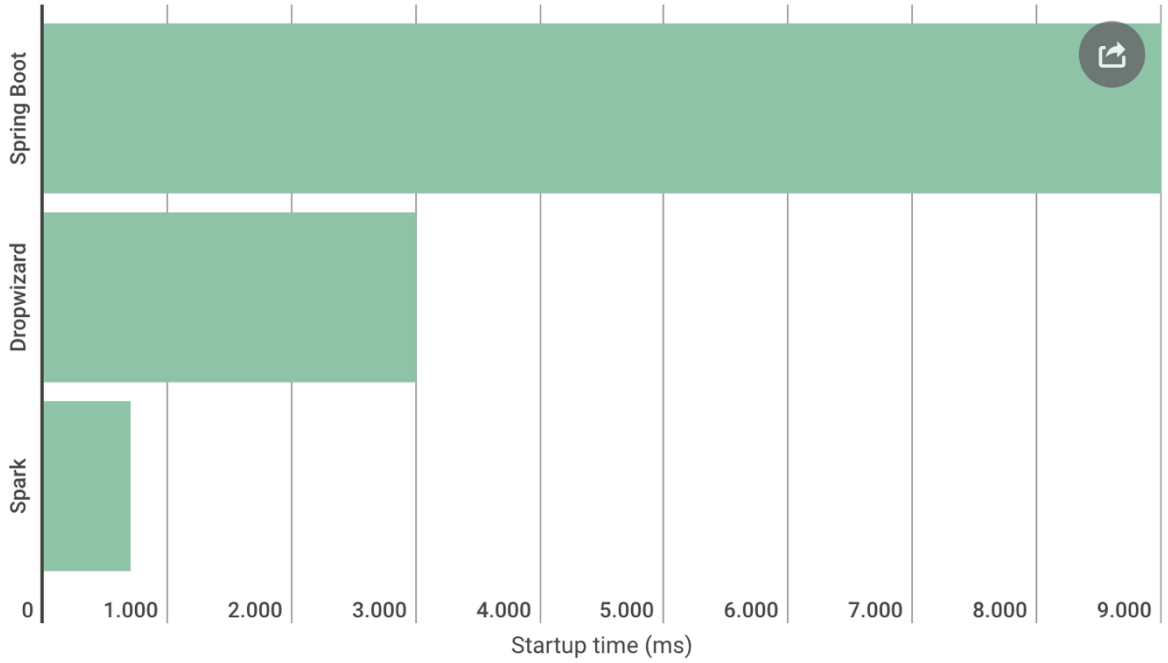
Door het toevoegen van de *dependency "spring-boot-starter-test"* zijn volgende *libraries* beschikbaar:

- JUnit: de standaard voor het testen van Java-applicaties
- Spring Test & Spring Boot Test: *library* voor test integratie met Spring
- AssertJ: een *library* voor het schrijven van leesbare *assertions*
- Hamcrest: een *library* voor het schrijven van *assertions*
- Mockito: een framework voor het schrijven van mocks in Java
- JSONassert: een *library* voor het vergelijken van JSON-objecten
- JsonPath: een *query language* die ervoor zorgt dat je makkelijk de info kunt terugvinden die je nodig hebt in JSON-data.

Omdat sommige testcases soms complex kunnen worden is het belangrijk dat er voldoende documentatie is en niet al te veel configuratie.

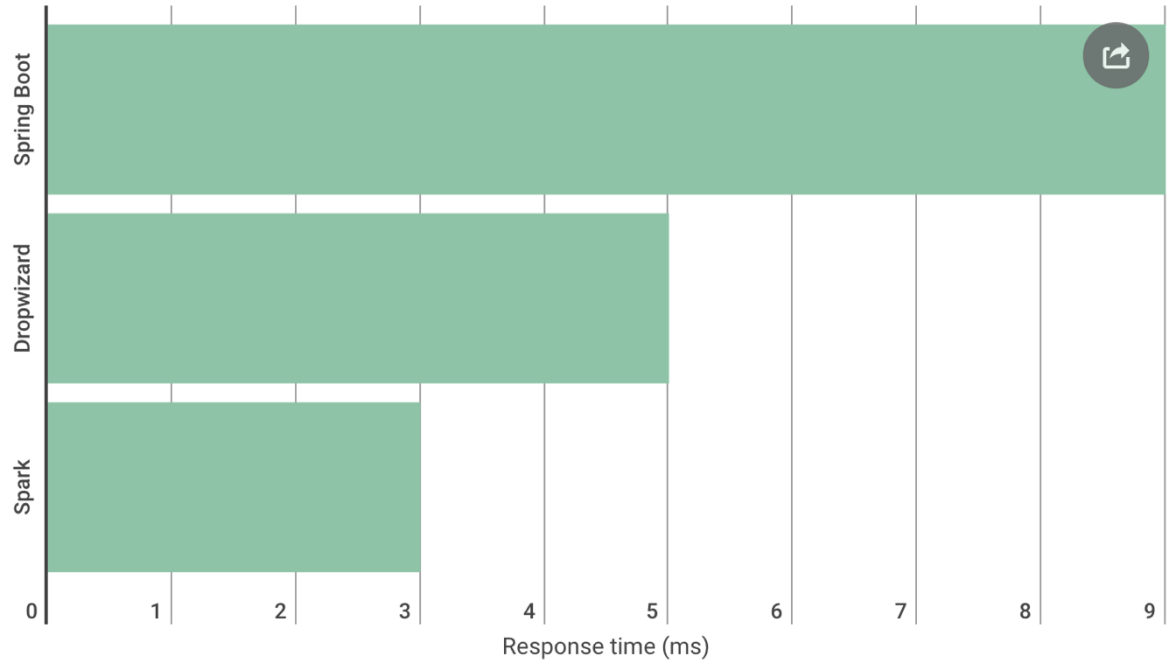
Een groot voordeel aan Spring Boot is dat je gebruik kan maken van MockMvc. Als je je controllers test door hier gebruik van te maken zullen je testen veel sneller uitgevoerd worden. Dit komt omdat MockMvc toelaat om de controllers te testen zonder de nood om een volledige http-server te starten. Terwijl er bij Dropwizard een in-memory Jersey server wordt gestart. Als je verschillende controllers hebt, voor elke controller een test klasse hebt, zorgt dit voor een enorme vertraging bij het uitvoeren van je testen.

1.3.7 Performance



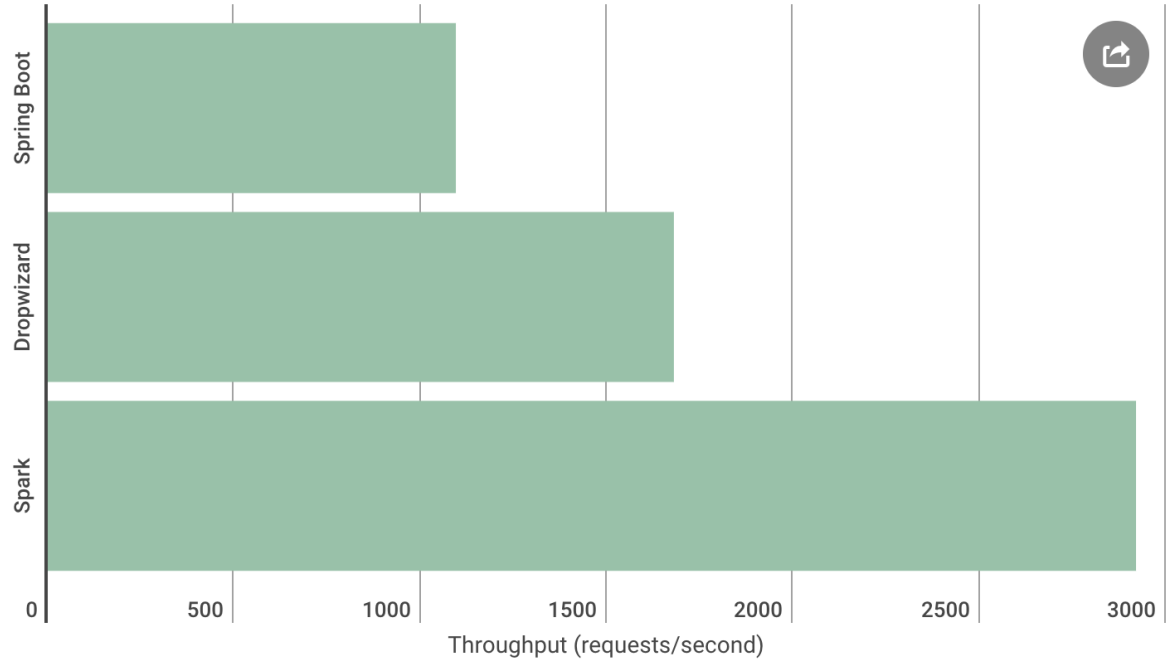
Figuur 30 opstarttijden frameworks

We zien hier dat Spring Boot met 9 seconden het langste duurt om op te starten. Dropwizard is hier 3x sneller. Ook zien we dat het framework Spark een zeer snelle opstarttijd heeft. [32] Er zijn verschillende factoren die meespelen in verband met deze opstarttijd, zo kunnen we bij Spring Boot bijvoorbeeld de *dependency* spring-boot-devtools gebruiken om tijdens het ontwikkelen de applicatie sneller te laten opstarten. Deze *dependency* zorgt ervoor dat de applicatie automatisch vernieuwd wordt wanneer er *gebuild* wordt. Zo kan je bijvoorbeeld in de IDE Intellij een macro maken die automatisch *build* wanneer je je project opslaat. Dit versnelt het ontwikkelingsproces, maar wil niet noodzakelijk zeggen dat een Spring Boot applicatie sneller zou opstarten dan een Dropwizard applicatie. Dropwizard heeft geen alternatief voor deze *dependency* die we in Spring Boot kunnen gebruiken. Wat ervoor zorgt dat we bij elke verandering in de code de server volledig moeten herstarten. Terwijl er in Spring Boot maar een deel van de applicatie opnieuw wordt ingeladen in het beste geval.



Figuur 31 response time frameworks

In de benchmarks van de post op de website Medium, zien we dat de reactietijd van dropwizard sneller is dan die van Spring Boot. [32] Zelf heb ik via de applicatie Postman een hello-world applicatie lokaal getest door 500 *requests* te sturen en hiervan de gemiddelde reactietijd op te berekenen. Via Postman kan men een collectie maken van *requests* en deze automatisch x aantal keren laten verzenden naar je server. Hierna kan je een JSON-file exporteren waarin de totale duur staat van de verzonden *requests*. Als je dit getal dan deelt door het aantal verzonden *requests* die je hebt verzonden dan bekom je het gemiddelde. Bij de Spring Boot hello-world applicatie was de reactietijd gemiddeld
7,5 ms. Bij Dropwizard was dit 9,5 ms. Deze tijden zijn echter moeilijk te vergelijken omdat we dit maar met een kleine hello-world applicatie lokaal hebben getest. Uit de post op Medium weten we enkel dat de server in een Docker container met 1GB RAM-geheugen draait en 1 CPU heeft, verder weten we ook dat de data uit het geheugen gehaald wordt en niet uit een database, en via Jackson naar JSON-formaat wordt omgezet. Bij de test die ik heb uitgevoerd maken we gebruik van een mysql database.



[32]

Figuur 32 requests per seconde

Als gevolg op de snellere reactietijd van Dropwizard zien we hier dat Dropwizard ook meer *requests* per seconde kan behandelen.

1.3.8 Vergelijkingsmatrix

	DROPWIZARD	SPRING BOOT
WAT IS HET	Dit framework zorgt ervoor dat je een aantal nuttige *libraries* ter beschikking hebt voor het bouwen van webapplicaties die gebruikt kunnen worden in een productieomgeving.	Spring Boot is een project dat gebouwd is op het Spring-framework. Spring Boot zorgt voor een makkelijke en snelle manier om een webapplicatie op te zetten.
POPULARITEIT	Dropwizard is uitgebracht in 2011 en wordt nog steeds gebruikt door sommige grote bedrijven als Uber, Airbnb en Openstack [32]	Spring Boot is uitgebracht in 2014 en op het moment zeer populair, dit betekent ook dat je sneller antwoorden gaat vinden als je ergens mee vast zit.
HTTP	Jetty	Tomcat (default), Jetty, Undertow of Reactor Netty
REST	Jersey	Spring (default), JAX-RS
JSON	Jackson	Jackson, GSON, JSON-B
DEPENDENCY INJECTION	Er is *library* beschikbaar voor *dependency injection*, je zal zelf een *third-party dependency injection framework* moeten toevoegen aan je project.	Ingebakken in Spring Boot
UNIT TESTING	Dropwizard-testing module	Spring-boot-starter-test
MONITORING	Metrics	Actuator
OPSTARTTIJD*	3s	9s

1.4 Conclusie

Uit vorige vergelijkingen kunnen we besluiten dat Spring Boot ten eerste zeer populair is en zelf zeer veel configuratie automatisch voorziet. Dit voordeel van configuratie voorzien heeft echter wel een keerzijde waar niet altijd over nagedacht wordt, als nieuwe gebruiker van dit framework is het soms moeilijk om te begrijpen wat er zich achter de schermen afspeelt.
Bij het Spring Boot-framework is het ook zo dat er veel mogelijkheden zijn in verband met de gebruikte technologieën. Zo heb je bijvoorbeeld de optie om als HTTP-server Tomcat, Jetty of Undertow te gebruiken. Bij Dropwizard heb je hier minder aan te zeggen en dat is niet per se een minpunt. Dropwizard houdt zich een beetje extremer aan het *Convention over configuration* principe ook wel bekend als *coding by convention* door zich bijvoorbeeld maar aan 1 logger, HTTP-server, … te houden. Dit is een concept van programmeren waarbij conventies worden gevolgd en het de bedoeling is dat door deze conventies minder beslissingen gemaakt moeten worden, waardoor meer eenvoud ontstaat.

De metingen laten zien dat Dropwizard sneller is, maar deze snelheid is afhankelijk van verschillende factoren. We weten namelijk niet in hoeverre deze metingen nog relevant zijn als we kijken naar applicaties met veel configuratie, *dependencies* of zeer complexe datastructuren. Ook tonen deze metingen enkel resultaat dat representatief is voor Dropwizard zelf, stel dat we gebruikmaken van verschillende *third-party libraries* in een Dropwizard applicatie dan zou het kunnen dat de performantie van de applicatie sterk achteruitgaat.

2 Onderzoek Docker

2.1 Probleemstelling

Tijdens het ontwikkelen van software is het belangrijk dat deze software op een eenvoudige manier gedeployed kan worden. In het verleden moest men vaak verschillende stappen doorlopen om de applicatie te deployen. Door gebruik te maken van Docker, kunnen er verschillende van deze stappen in dit proces geautomatiseerd worden. Dit zorgt ervoor dat er meer gefocust kan worden op het ontwikkelen van de software.

In dit onderzoek, zoek ik uit hoe deze tool gebruikt kan worden om een applicatie te deployen. Aan de hand van een PoC beschrijf ik hoe je een applicatie door gebruik te maken van Docker eenvoudiger kan deployen.

2.2 Uitvoering

De applicatie in deze PoC is een kleine Spring Boot-applicatie die bij het opstarten een database seed. Door een http-request te doen naar de server, gaat de applicatie deze data in JSON-formaat terugsturen.

Om deze applicatie via Docker te kunnen deployen heb ik een Dockerfile gemaakt. Deze Dockerfile ziet er als volgt uit.

```
1  FROM  openjdk:8-jdk-alpine
2  VOLUME /tmp
3  ARG JAR_FILE="target/onderzoek-docker-0.0.1-SNAPSHOT.jar"
4  COPY ${JAR_FILE} app.jar
5  ENTRYPOINT ["java","-jar","/app.jar"]
```

Figuur 33 voorbeeld Dockerfile

1. We vertrekken van een bestaande image waar we de openjdk beschikbaar hebben. Deze image is gebaseerd op Alpine Linux, die maar 5 MB groot is. Dit om de image zo klein mogelijk te houden.
2. Bij het afsluiten of herstarten van een container zijn we alles van deze container kwijt. De volume verwijst naar /tmp hier creëert een Spring Boot-applicatie zijn folders voor Tomcat. (In deze applicatie zou dit niet nodig zijn, maar als je applicatie naar het filesysteem iets moet gaan wegschrijven is dit wel nodig. Anders zou dit bij het herstarten van de applicatie weer weg zijn.)
3. We definiëren verder de locatie van de jar-file die we willen deployen.
4. We zorgen ervoor dat de jar file wordt gekopieerd naar de container als app.jar
5. We zorgen ervoor de applicatie wordt gestart via het java commando met de nodige parameters.

Op dit moment kunnen we, nadat we via maven onze jar file hebben laten genereren, deze dockerfile al uitvoeren om een image te maken. Dit kunnen we doen door het commando “docker build .” in de root folder die onze dockerfile bevat uit te voeren.

Via de dockerfile-maven-plugin van Spotify kunnen we de image automatisch laten genereren nadat men via maven gepackaged heeft.

```xml
<plugin>
    <groupId>com.spotify</groupId>
    <artifactId>dockerfile-maven-plugin</artifactId>
    <version>1.4.10</version>
    <executions>
        <execution>
            <id>default</id>
            <goals>
                <goal>build</goal>
            </goals>
        </execution>
    </executions>
    <configuration>
        <repository>brentwillems/demo</repository>
    </configuration>
</plugin>
```

Figuur 34 Configuratie dockerfile-maven-plugin

We kunnen de applicatie op dit moment ook al in een Docker container laten draaien via het commando “Docker run {image id}”, maar we gaan dit doen via Docker Compose. Docker Compose is een tool voor het runnen van Docker applicaties. Deze wordt voornamelijk gebruikt als men in een applicatie met meerdere containers werkt.

In deze PoC heb ik een docker-compose.yml gemaakt die ervoor zorgt dat we de applicatie kunnen starten in een container en deze linken aan een database-container.

docker-compose.yml | werknemers@172.21.0.3 [console] | werknemers.werknemer [werknemers@172.21.0.3]

```yaml
version: '3.7'
services:
  werknemer-service:
    image: brentwillems/demo:latest
    ports:
      - 8080:8080
    depends_on:
      - werknemer-db
    environment:
      SPRING_DATASOURCE_URL: "jdbc:mysql://werknemer-db/werknemers?useSSL=false&allowPublicKeyRetrieval=true"
      SPRING_DATASOURCE_USERNAME: "dbuser"
      SPRING_DATASOURCE_PASSWORD: "dbpass"
  werknemer-db:
    image: mysql:8.0.14
    environment:
      MYSQL_ROOT_PASSWORD: "password"
      MYSQL_USER: "dbuser"
      MYSQL_PASSWORD: "dbpass"
      MYSQL_DATABASE: "werknemers"
      MYSQL_ONETIME_PASSWORD: "true"
```

Figuur 35 docker-compose.yml bestand

version: dit is de versie van Docker Compose die we gebruiken. Versie 3.7 is momenteel de meest recente versie.

services: hier definiëren we de containers die we willen hebben. We definiëren een service met als naam *werknemer-service*. Dit wordt de container voor de Spring Boot applicatie. We gebruiken de image die we zojuist aangemaakt hebben.

We mappen poort 8080 van de host op poort 8080 van de container.

depends_on:
De container is afhankelijk van een andere container. In dit geval de database-container. Dit zorgt ervoor dat de containers in de juiste volgorde worden opgestart.

environment:
We geven hier de variabelen mee om een connectie te kunnen maken met de database. Ook kan je hier zien dat de url bestaat uit de naam van de service. De services kunnen elkaar aanspreken aan de hand van hun naam. Dit kan zeer handig zijn in een microservice-architectuur, want je kan deze ook gebruiken om tussen de applicaties te communiceren.

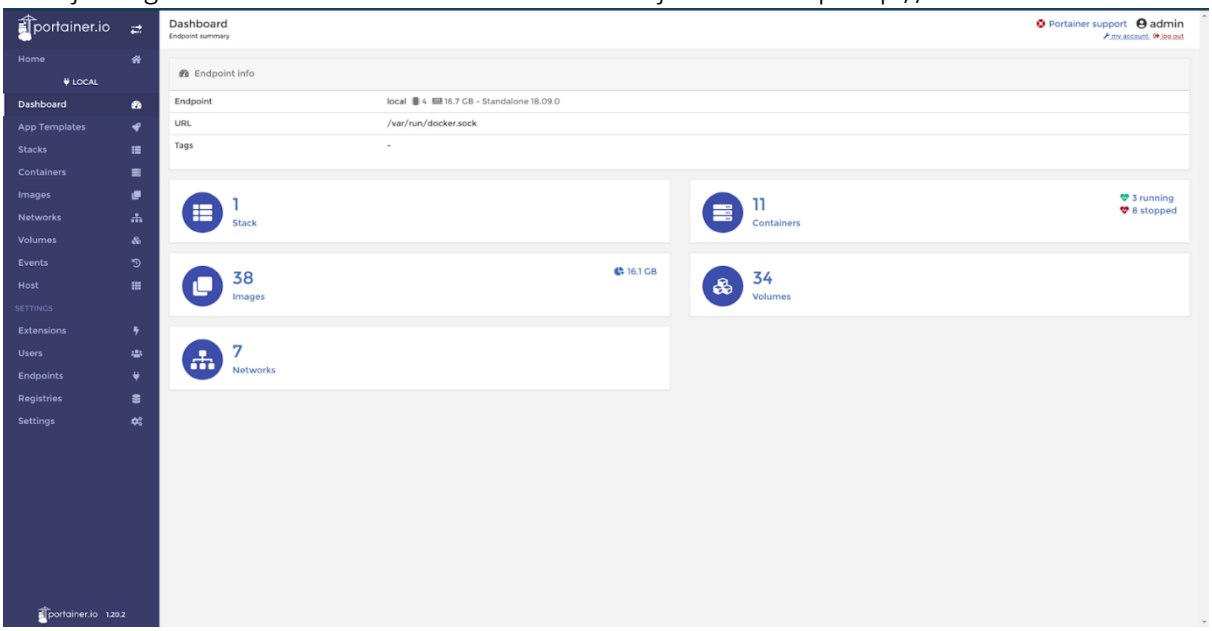
Verder wordt de database-container gedefinieerd. We gebruiken de mysql image en geven parameters mee om de database te configureren.

Als we nu de docker-compose file uitvoeren door het commando “docker-compose up” te gebruiken zien we dat de containers zijn opgestart en de applicatie werkt.

In deze PoC heb ik ook gebruikgemaakt van Portainer. Portainer is een GUI om het gebruik van docker overzichtelijk te maken.
Je kan Portainer heel makkelijk installeren omdat ook hier een Docker-image voor gemaakt is. Dit doe je door de volgende commando’s uit te voeren:

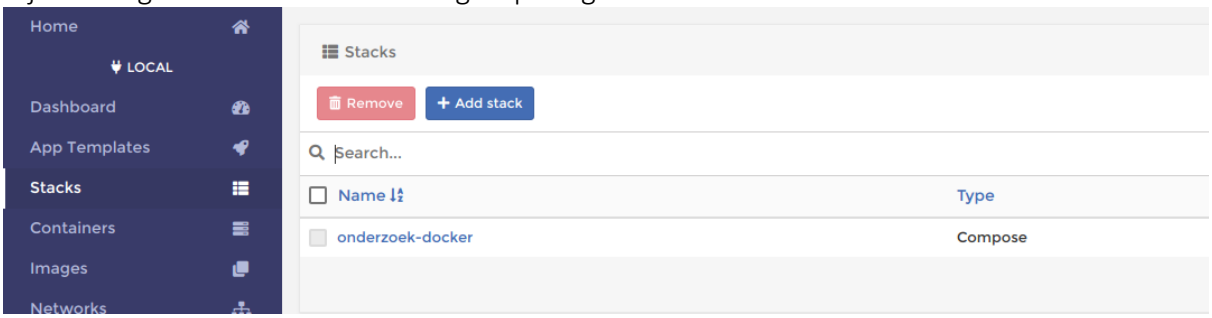
- docker volume create portainer_data
- docker run -d -p 9000:9000 --name portainer --restart always -v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer

Eens je dit gedaan hebt is Portainer beschikbaar in je browser op http://localhost:9000.



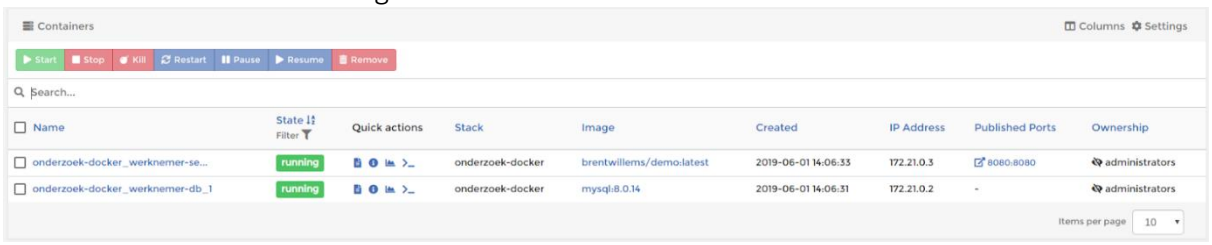
Figuur 36 Dashboard Portainer

Bij de categorie stacks zien we een groepering van docker-containers die samenhoren.



Figuur 37 Screenshot Portainer

Hier zien we de stack die we gemaakt hebben.



Figuur 38 Screenshot Portainer

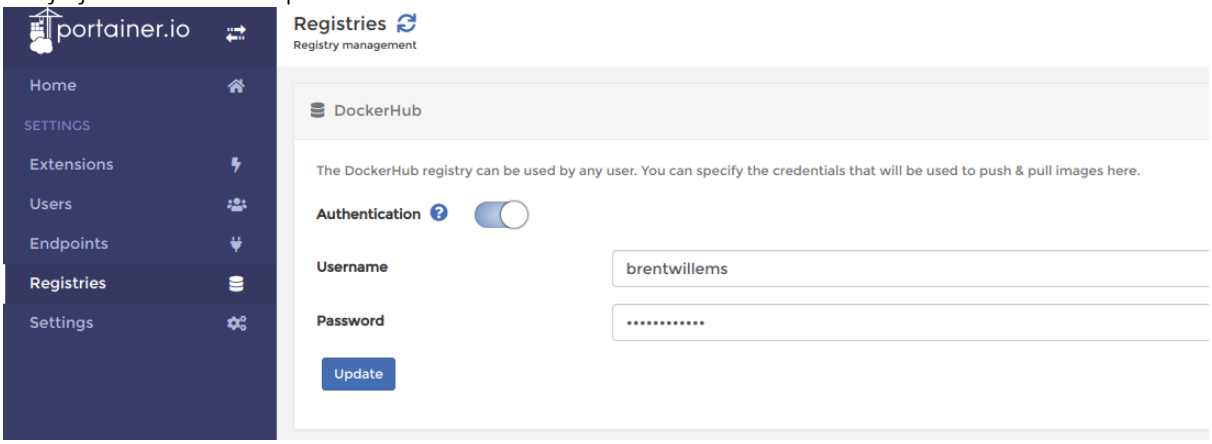
In deze stack zitten de 2 containers die we hebben aangemaakt.

De applicatie is nu gedeployed aan de hand van docker-containers.
Docker maakt het ook zeer gemakkelijk om deze applicatie op andere systemen te deployen.
Om dit te kunnen doen heb ik een account aangemaakt op DockerHub.

Met dit account kan je je images pushen naar de publieke registry. Dit kunnen we doen via de dockerfile-maven-plugin die we eerder hebben gebruikt voor het bouwen van de image.

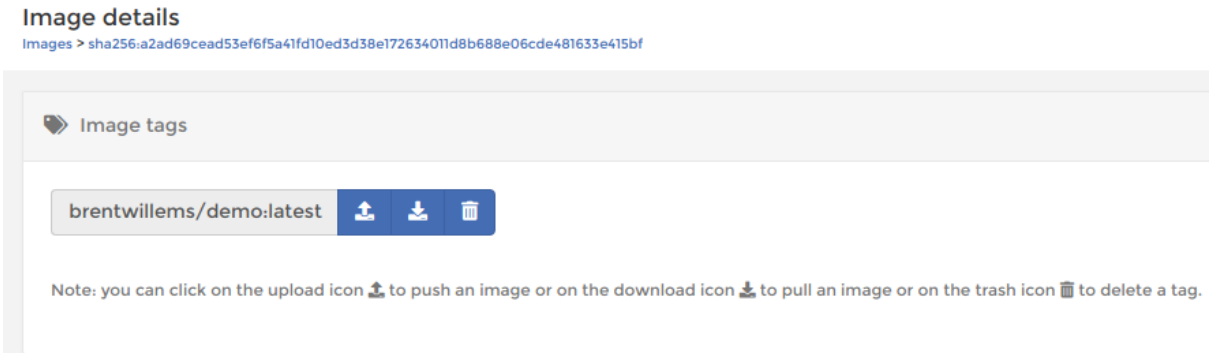
In de configuratie van deze plugin stellen we een repository in. Als we deze instellen moeten we ook onze username van DockerHub en ons password instellen. Als dit is gebeurd kan je via het commando “mvn dockerfile:push” je image naar deze registry pushen.

Het pushen van de image kan ook via de Portainer applicatie. Hiervoor ga je naar registries en vul je je username en password in.



Figuur 39 Screenshot Portainer

Vervolgens kan je naar je image navigeren.



Figuur 40 Screenshot Portainer

Hier kan je nu je image pushen.

De image kan nu door iedereen gebruikt worden, omdat je repository publiek is.
Je kan ook gebruik maken van een private repository of je registry zelf hosten.
Het enige wat we nu moeten doen als we deze applicatie willen deployen op een ander device, is ervoor zorgen dat Docker geïnstalleerd is en je de docker-compose.yml file uitvoert op dit device.

```
PS D:\Bureaublad> mv .\docker-compose.yml.txt .\docker-compose.yml
PS D:\Bureaublad> docker-compose up
Creating network "bureaublad_default" with the default driver
Pulling werknemer-db (mysql:8.0.14)...
8.0.14: Pulling from library/mysql
5e6ec7f28fb7: Downloading [============================>                      ]   12.9MB/22.5MB
4140e62498e1: Download complete
e7bc612618a0: Download complete
1af808cf1124: Download complete
ff72a74ebb66: Download complete
3a28cb03e3dc: Download complete
2b52dda3bd7d: Download complete
dc89e81122ad: Download complete
ecba98b7a588: Downloading [=======>                                           ]  14.49MB/95.64MB
109b011a27be: Download complete
5f380f98ab52: Waiting
cdda841f5c5c: Waiting
```

Figuur 41 Terminal docker-compose up

Figuur 42 Terminal docker-compose up

Docker gaat dan naar je image zoeken, en deze lokaal niet vinden. Omdat hij deze lokaal niet kan vinden gaat hij kijken in het register en zal Docker deze image vinden. De database image komt al uit het register, dus deze wordt ook binnengehaald en vervolgens is je applicatie gedeployed op je device.

2.3 Conclusie

Het gebruik van Docker geeft je verschillende voordelen, zo vond ik het zeer opmerkelijk dat je op deze manier heel snel en eenvoudig je database kan opzetten. Tijdens het opzetten van deze PoC ben ik vooral problemen tegengekomen omdat er verkeerde drivers en versies gebruikt werden. Ik ben zelf al eerder in contact gekomen met Docker door er een beetje mee te experimenteren in een groepsproject voor school, maar op dat moment wist ik er nog niet zo veel van als nu en was het niet echt een meerwaarde. Door het gebruik van de beschikbare plug-ins en docker-compose ben ik zeker overtuigd dat Docker een meerwaarde is voor een project, omdat dit de opzet een stuk gemakkelijker maakt. Het was heel gemakkelijk om via Docker de applicatie op een ander systeem (met zelfs een ander OS) te deployen. Ook het gebruik van een Docker registry lijkt me zeer handig om software te bouwen. Zo kan je bijvoorbeeld in een microservice-applicatie in team aan bepaalde services werken, zonder al te veel werk te hebben met het up-to-date houden van de andere services waar andere teams aan hebben gewerkt.

Bibliografie

[1] „ACA IT-Solutions,” [Online]. Available: https://mobile.aca-it.be/.

[2] „ACA IT-Solutions,” [Online]. Available: https://www.aca-it.be/en/services/#pod--act.

[3] „ACA IT-Solutions,” [Online]. Available: https://www.aca-it.be/en/services/blockchain/.

[4] „ACA IT-Solutions,” [Online]. Available: https://www.aca-it.be/en/services/po-offering/.

[5] „ACA IT-Solutions,” [Online]. Available: https://www.aca-it.be/nl/onze-services/#pod--amplifly-2.

[6] „ACA IT-Solutions,” [Online]. Available: https://coin.aca-it.be/nl.

[7] „ACA IT-Solutions,” [Online]. Available: https://collectiv.aca-it.be/.

[8] „IT MATCH,” [Online]. Available: https://www.itmatch.be/nl.

[9] „ACA IT-Solutions,” [Online]. Available: https://opsklaar.aca-it.be/.

[10] [Online]. Available: https://ecm2.nl/producten/alfresco-document-management/waarom-alfresco.

[11] „Atlassian,” [Online]. Available: https://www.atlassian.com/.

[12] „Wikipedia,” [Online]. Available: https://en.wikipedia.org/wiki/Liferay.

[13] „Wikipedia,” [Online]. Available: https://en.wikipedia.org/wiki/New_Relic.

[14] „Wikipedia,” [Online]. Available: https://nl.wikipedia.org/wiki/Oracle_Corporation.

[15] „Wikipedia,” [Online]. Available: https://nl.wikipedia.org/wiki/Backbase. [Geopend maart 2019].

[16] D. d. Man. [Online]. Available: https://webwereld.nl/development/89736-wat-betekent-amazon-web-services-voor-ontwikkelaars.

[17] „Wikipedia,” [Online]. Available: https://nl.wikipedia.org/wiki/Proximus_Groep. [Geopend maart 2019].

[18] „Wikipedia,” [Online]. Available: https://en.wikipedia.org/wiki/Axway_Software. [Geopend maart 2019].

[19] „SAP,” [Online]. Available: https://www.sap.com/belgie/about.html. [Geopend maart 2019].

[20] „Wikipedia,” [Online]. Available: https://nl.wikipedia.org/wiki/Adobe_Systems.

[21] „Wikipedia,” [Online]. Available: https://nl.wikipedia.org/wiki/Hibernate. [Geopend maart 2019].

[22] [Online]. Available: https://computerworld.nl/cloud/84263-wat-is-docker. [Geopend maart 2019].

[23] „Spring,” [Online]. Available: https://spring.io/projects/spring-boot. [Geopend februari 2019].

[24] „Dropwizard,” [Online]. Available: https://www.dropwizard.io. [Geopend februari 2019].

[25] „Stackshare,” [Online]. Available: https://stackshare.io/stackups/dropwizard-vs-spring-boot. [Geopend 1 maart 2019].

[26] „Google Guice,” [Online]. Available: https://github.com/google/guice. [Geopend maart 2019].

[27] „GitHub,” [Online]. Available: https://github.com/hmsonline/dropwizard-spring.

[28] „Dropwizard Metrics,” [Online]. Available: https://metrics.dropwizard.io/4.0.0/. [Geopend maart 2019].

[29] [Online]. Available: https://docs.spring.io/spring-boot/docs/1.3.1.RELEASE/reference/html/production-ready-metrics.html.

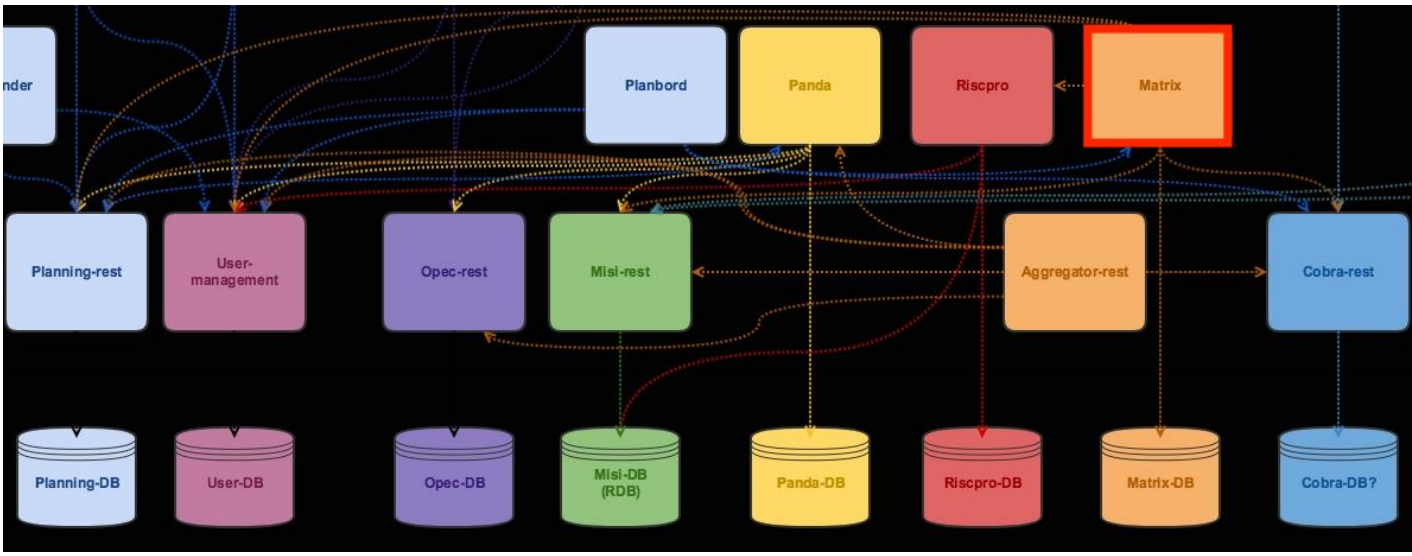
[30] „Spring Boot voorziet 17 verschillende monitoringssystemen,” [Online]. Available: https://docs.spring.io/spring-boot/docs/current/reference/html/production-ready-metrics.html. [Geopend 7 maart 2019].

[31] „GitHub Dropwizard - DataDog,” [Online]. Available: https://github.com/coursera/metrics-datadog.

[32] „Benchmarks Spring Boot, Dropwizard, Spark,” [Online]. Available: https://medium.com/@pablopallocchi/fastest-microservices-in-java-509a34f2ee7c.

[33] „Spring I/O,” [Online]. Available: https://springio.net.

Bijlage


nder
Planbord
Panda
Riscpro
Matrix
Planning-rest
User-management
Opec-rest
Misi-rest
Aggregator-rest
Cobra-rest
Planning-DB
User-DB
Opec-DB
Misi-DB (RDB)
Panda-DB
Riscpro-DB
Matrix-DB
Cobra-DB?