HOGESCHOOL ROTTERDAM

BACHELOR SCRIPTIE

Verbetering van schaal- en onderhoudbaarheid in de infrastructuur van Developers.nl

---

Auteur:
Kaj de Munter
0911825

Begeleiders:
Tanja Ubert
Judith Lemmens

28/12/2019

`v0.5`

DEVELOPERS.NL

HOGESCHOOL ROTTERDAM

BACHELOR SCRIPTIE

Verbetering van schaal- en onderhoudbaarheid in de infrastructuur van Developers.nl

Auteur:
Kaj de Munter
0911825@hr.nl
k.demunter@developers.nl
06-81019142

Schoolbegeleiders:
Tanja Ubert
t.ubert@hr.nl
Judith Lemmens
j.h.i.lemmens@hr.nl

Bedrijfsbegeleiders:
Maarten de Boer
m.deboer@developers.nl
Jelle van de Haterd
j.vandehaterd@developers.nl

Een scriptie ingediend ter voldoening aan de vereiste competenties voor de opleiding Informatica

Communicatie, Media, en Informatietechnologie
28/12/2019

`v0.5`

developers.nl
learn. code. share.

HOGESCHOOL ROTTERDAM

Voorwoord

Bedankt aan iedereen die mij koffie heeft gebracht afgelopen periode... Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

HOGESCHOOL ROTTERDAM

Samenvatting

Communicatie, Media, en Informatietechnologie

Informatica

Verbetering van schaal- en onderhoudbaarheid in de infrastructuur van Developers.nl

door Kaj de Munter

Samenvatting van de scriptie die ik als laatst pas ga schrijven... Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

HOGESCHOOL ROTTERDAM

Abstract

Communicatie, Media, en Informatietechnologie

Informatica

Verbetering van schaal- en onderhoudbaarheid in de infrastructuur van Developers.nl

door Kaj de Munter

Samenvatting van de scriptie die ik als laatst pas ga schrijven... Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Inhoud

Figurenlijst

Tabellenlijst

Begrippenlijst

Slack	Het communicatiemiddel waar Developers.nl gebruik van maakt.
Proxmox	Proxmox VE is a complete open-source platform for enterprise virtualization [1]
Service Discovery	Service discovery is the process of automatically detecting devices and services on a network [2]
Kubernetes	Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications [3].
Docker	Docker is a tool designed to make it easier to create, deploy, and run applications by using containers [4].
Ansible	Ansible is a radically simple IT automation engine that automates cloud provisioning, configuration management, application deployment, intra-service orchestration, and many other IT needs [5].
Docker Swarm	Docker Swarm provides native clustering functionality for Docker containers, which lets you turn a group of Docker engines into a single, virtual Docker engine [6].
Serverless Computing	En application defined as a set of event-triggered functions that execute without requiring the user to explicitly manage servers [7].
Infrastructure as Code	Infrastructure as code describes the idea of using a high-level programming language to control IT systems [8].
Ansible Tower	Red Hat Ansible Tower helps you scale IT automation, manage complex deployments and speed productivity. Centralize and control your IT infrastructure with a visual dashboard, role-based access control, job scheduling, integrated notifications and graphical inventory management [9].
Chef	Deploy new code faster and more efficiently. Automate infrastructure and applications [10].
Puppet	Powerful infrastructure automation and delivery [11].
Terraform	Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently. Terraform can manage existing and popular service providers as well as custom in-house solutions [12].

Cloud computing providers	Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [13]. Er bestaan veel cloud computing providers, waaronder bijvoorbeeld: • Amazon web services • DigitalOcean • Microsoft Azure • Google Cloud Platform
12-factor app	Een methodologie voor het bouwen van Software as a Service (SaaS) applications [14].
The Open Group Architecture Framework	A generic framework to build different IT architectures frameworks [15].
4+1 architectural view model	A model for describing the architecture of software-intensive systems, based on the use of multiple, concurrent views [16].

Afkortingenlijst

MI	**M**aintainability **I**ndex
CI	**C**ontinuous **I**ntegration
CD	**C**ontinuous **D**eployment
CM	**C**onfiguration **M**anagement
VU	**V**irtual **U**ser
VM	**V**irtual **M**achine
APM	**A**pplication **P**erformance **M**onitoring
EMS	**E**mployee **M**anagement **S**ystem
QoS	**Q**uality **o**f **S**ervice
AWS	**A**mazon **W**eb **S**ervices
GCP	**G**oogle **C**loud **P**latform
OPA	**O**pen **P**olicy **A**gent
PaC	**P**olicies **a**s **C**ode
IaC	**I**nfrastructure **A**s **C**ode
K8s	**K**ubernete**s**
TCP	**T**ransmission **C**ontrol **P**rotocol
TLS	**T**ransport **L**ayer **S**ecurity
IaaS	**I**nfrastructure **a**s **a** **S**ervice
PaaS	**P**latform **a**s **a** **S**ervice
SaaS	**S**oftware **a**s **a** **S**ervice
TOGAF	**T**he **O**pen **G**roup **A**rchitecture **F**ramework
FastCGI	**F**ast **C**ommon **G**ateway **I**nterface

Hoofdstuk 1

Inleiding

1.1 Aanleiding

"Een visitekaartje voor het bedrijf". Dat is het uitgangspunt van de interne software bij Developers.nl. Niet alleen qua uiterlijk, maar ook van binnen moet de code, de infrastructuur en de werkmethode van hoge kwaliteit zijn. Dit heeft te maken met het feit dat de code uit de website en infrastructuur van Developers.nl open-source wordt gemaakt gedurende deze stage. Het open-source maken van de website betekent dat elke potentiële klant en/of nieuwe medewerker de mogelijkheid heeft om te bekijken wat Developers.nl qua kennis in huis heeft. Het is dus van groot belang dat de kwaliteit gewaarborgd wordt, en dat zo veel mogelijk nieuwe en opkomende technieken worden gebruikt. Dit vereist constant onderhoudswerk. Daarnaast heeft Wheeler [17] geconcludeerd dat open-source software voordelen heeft als:

- Betere beveiliging
- Betere betrouwbaarheid
- Betere prestaties
- Betere schaalbaarheid
- Mindere onderhoudskosten

Developers.nl organiseert maandelijks een "TechNight". Op deze TechNight ontvangt de website van Developers.nl een piek aantal bezoekers, het is belangrijk dat deze pieken goed worden afgehandeld zonder enige downtime. Dit betekent dat onderzoek op de kwaliteit van de huidige website belangrijk is. Bovendien zijn er meerdere interne systemen dan alleen de website, zoals bijvoorbeeld het Employee Management System (EMS). Het onderhouden van deze systemen vereist veel tijd en moeite. Dit wordt voornamelijk door stagiairs of tijdelijke hulpkrachten uitgevoerd. In dit onderzoek wordt voornamelijk gefocussed op de website. Dit is de applicatie die het meest frequent gebruikt wordt, en dus de meeste aandacht verdiend.

1.2 Belang

De interne systemen zijn op eerste gezicht van de buitenkant vrij eenvoudig. Developers.nl wilt – om indruk te wekken op potentiële klanten en nieuwe medewerkers – onder water een applicatie draaien dat "te complex" is. Maar; omdat de ontwikkelaars óf "minder ervaren" stagiairs zijn, óf een tijdelijke hulpkracht zijn kost het onderhouden – vooral met 5 verschillende systemen – erg veel tijd, moeite, en als gevolg hiervan: geld.

1.3 Doelstelling

Dit onderzoek heeft tot doel het verkrijgen van inzicht over de schaal- en onderhoudbaarheid van websites die Developers.nl beheert, om vervolgens deze twee factoren in de praktijk te verbeteren.

1.4 Probleemstelling

Bij Developers.nl werken veel verschillende ontwikkelaars voor een erg variabele tijd aan de interne projecten. Dit heeft te maken met het feit dat de ontwikkelaars die eraan werken vaak tussen twee opdrachten in zitten. Dit betekent dat het van hoog belang is dat een ontwikkelaar de omgeving snel kan opzetten en op korte termijn een kwalitatieve toevoeging kan leveren die in productie staat. De workflow moet verder geoptimaliseerd worden om Developers.nl dit te beloven.

1.5 Hoofd- en Deelvragen

Hoofdvraag

Op welke wijze kan Developers.nl de architectuur van haar websites beter schaal- en onderhoudbaar maken?

Deelvragen

- Wat zijn de wensen en eisen van Developers.nl met betrekking tot de schaal- en onderhoudbaarheid van haar huidige websites?
- Welke standaarden en best-practices voor het waarborgen van schaal- en onderhoudbaarheid zijn relevant voor de eisen van Developers.nl?
- Hoe onderhoud- en schaalbaar zijn de huidige websites van Developers.nl met betrekking tot de relevante kwaliteitsstandaarden?
- Wat voor verbeteringen ten aanzien van schaal- en onderhoudbaarheid kunnen worden toegepast op de huidige websites van Developers.nl?
- Hoe kunnen de gekozen verbeteringen ten aanzien van schaal- en onderhoudbaarheid geïmplementeerd worden?
- Voldoen de verbeteringen aan de vereiste requirements?

1.6 Methodologie

Om antwoord te geven op de hoofdvraag “Op welke wijze kan Developers.nl de architectuur van haar websites beter schaal- en onderhoudbaar maken?” is voornamelijk kwalitatief onderzoek uitgevoerd. Vooronderzoek is uitgevoerd door bestaande literatuur te bestuderen om zo een beter beeld te verkrijgen en om een basis te leggen van de belangrijkste begrippen. Alle bronnen zijn handmatig gecontroleerd op kwaliteit en relevantie. Voor het definiëren van functionele schaalbaarheid is veel overlegd met verschillende software-ontwikkelaars, veel feedback gevraagd aan de community, en zijn zowel formele als informele bronnen samengevoegd om zo tot één concrete definitie te komen.

Om technieken te vinden die behoren bij schaal- en onderhoudbaarheid is deskresearch uitgevoerd door middel van interviews met ontwikkelaars en bestaande onderzoeken te verzamelen. Hierna zijn deze technieken afgebakend tot de meest relevante die bij dit onderzoek horen. Vervolgens is een beschrijvend onderzoek uitgevoerd op de huidige infrastructuur in hoofdstuk 5. Hier zijn bestaande kenmerken en elementen van de huidige infrastructuur tegen de gevonden standaarden en technieken uit hoofdstuk 4 afgewogen. Om verschillende verbeteringen te vinden is net als hoofdstuk 4 deskresearch uitgevoerd. Dit bevat voornamelijk interviews met senior ontwikkelaars die deze technieken in de praktijk gebruiken. Hierna zijn de gevonden methodes kwalitatief onderbouwd.

Voor de daadwerkelijke implementatie is allereerst exploratief onderzoek gedaan naar de bijbehorende technieken en hun best-practices. Het doel hiervan is vooral om ideeën op te doen naar mogelijke implementaties.

1.7 Planning

In tabel 1.1 is de vooraf opgestelde planning te vinden. Het is mogelijk dat hier vanaf is geweken tijdens het daadwerkelijke onderzoek, maar het geeft een ruw beeld van de tijdsverdeling.

TABEL 1.1: Planning

Week	Taak
1, 2	Skelet opzet scriptie, inleiding
3, 4	Theoretisch kader, afbakening
5, 6	Welke standaarden en best-practices voor het waarborgen van schaal- en onderhoudbaarheid zijn relevant voor de eisen van Developers.nl?
7, 8	Hoe onderhoud- en schaalbaar zijn de huidige websites van Developers.nl met betrekking tot de relevante kwaliteitsstandaarden?
9, 10	Wat voor verbeteringen ten aanzien van schaal- en onderhoudbaarheid kunnen worden toegepast op de huidige websites van Developers.nl?
11, 12	Hoe kunnen de gekozen verbeteringen ten aanzien van schaal- en onderhoudbaarheid geïmplementeerd worden?
13 – 17	Praktijk implementatie
18 – 20	Voldoen de verbeteringen aan de vereiste requirements?en conclusie.

1.8 Leeswijzer

Vóór het "echte onderzoek" is eerst in het theoretisch kader (hoofdstuk in 2) een literatuuronderzoek uitgevoerd naar definities van de meest belangrijke begrippen: **Schaalbaarheid**, **onderhoudbaarheid**, en **infrastructuur**. Hierdoor is een concrete basis gelegd voor de opvolgende deelvragen. In hoofdstuk 3 zijn de wensen en eisen van Developers.nl vastgelegd en requirements opgesteld. Daarna is in 4 onderzoek gedaan naar de mogelijke technieken, met de deelvraag: "Welke standaarden en best-practices voor het waarborgen van schaal- en onderhoudbaarheid zijn relevant voor de eisen van Developers.nl?". Hier zijn standaarden en best-practices besproken om te kunnen bewijzen dat de hoofdvraag daadwerkelijk beantwoord is. In het volgende hoofdstuk (5) met deelvraag "Hoe onderhoud- en schaalbaar zijn de

huidige websites van Developers.nl met betrekking tot de relevante kwaliteitsstandaarden?" zijn deze standaarden afgewogen tegen de huidige infrastructuur. Vervolgens wordt in hoofdstuk 6 onderzocht welke verbeteringen hier op toe te passen zijn, hierbij hoort de deelvraag "Wat voor verbeteringen ten aanzien van schaal- en onderhoudbaarheid kunnen worden toegepast op de huidige websites van Developers.nl?". In hoofdstuk 7 wordt de daadwerkelijke implementatie van deze verbeteringen besproken door verschillende opties met elkaar af te wegen. Hier wordt antwoord gegeven op de deelvraag "Wat voor verbeteringen ten aanzien van schaal- en onderhoudbaarheid kunnen worden toegepast op de huidige websites van Developers.nl?". Hierna zijn de geïmplementeerde verbeteringen geëvalueerd in hoofdstuk 8, bijbehorende deelvraag "Voldoen de verbeteringen aan de vereiste requirements?". Nu alle deelvragen zijn beantwoord worden er in hoofdstuk 9 aanbevelingen toegelicht voor toekomstige verbeteringen. Ten slotte wordt in hoofdstuk 10 antwoord gegeven op de hoofdvraag "Op welke wijze kan Developers.nl de architectuur van haar websites beter schaal- en onderhoudbaar maken?" door alle deelconclusies samen te binden tot één hoofdconclusie.

1.9 Opdrachtgever

Deze scriptie is geschreven in opdracht van Developers.nl.

1.9.1 Core business

Developers.nl neemt software ontwikkelaars in dienst. De ontwikkelaars die worden aangenomen zullen voornamelijk gespecialiseerd zijn in PHP, Python, Java of front-end. Ze worden uitgezet naar een klant (een extern bedrijf) die naar een ontwikkelaar zoekt. Developers.nl kiest hier voor de beste ontwikkelaar voor de taak en zal deze inzetten bij een klant. De opdrachten van de ontwikkelaars zijn op locatie van de klant en duren voornamelijk langer dan een jaar, maar op uitzondering zijn er ook kortere opdrachten. Zodra de ontwikkelaar klaar is met zijn of haar taak zal Developers.nl zo snel mogelijk een nieuwe opdracht toewijzen [18]. Concreet zegt het positioneringsprofiel [19]: "Detachering van developers die software applicaties bouwen voor verschillende klanten."

1.9.2 Eigen omgeving

Tijdens de stageperiode neemt de stagiair een leidende rol aan in een team van 2 part-time studenten, een derdejaars-stagiair, en de tijdelijke hulpkrachten. Developers.nl heeft rond de 60 software ontwikkelaars. Deze zijn voornamelijk op een externe opdracht bij een klant. Elke vrijdag zullen 5 "kennisambassadeurs" op kantoor zijn. Dit zijn de meest senior ontwikkelaars per team. Deze zijn dan in staat om stagiairs en/of andere medewerkers persoonlijk te helpen. Hoewel ze maar één keer per week op kantoor aanwezig zijn, zijn ze altijd telefonisch bereikbaar of via Slack. Daarnaast kijken de kennisambassadeurs code van de interne systemen inhoudelijk na en geven hier feedback op.

De bedrijfsbegeleider voor deze stage is Maarten de Boer. Dit is de algemene directeur van Developers.nl en is in 2003 afgestudeerd aan de hogeschool Inholland met Strategic marketing. Aangezien Maarten zelf geen technische kennis heeft is er ook een technische begeleider aangewezen: Jelle van de Haterd. Jelle is senior developer, DevOps engineer en kennisambassadeur bij Developers.nl. Hij is in 2006

afgestudeerd op de Hogeschool Rotterdam met als opleiding Grafimediatechnologie [20].

Hoofdstuk 2

Theoretisch Kader

In dit hoofdstuk worden vier belangrijke begrippen uit de onderzoeksvraag behandeld. Er wordt een literatuuronderzoek gedaan naar de bestaande definities van schaalbaarheid, onderhoudbaarheid en architectuur met betrekking tot software. Hierna wordt het begrip afgebakend tot een concrete definitie waar het onderzoek op terug kan vallen.

2.1 Schaalbaarheid

M. D. Hill heeft in 1990 onderzoek gedaan naar een concrete definitie naar schaalbaarheid [21]. In zijn onderzoek concludeert hij het volgende:

> *I examined aspects of scalability, but did not find a useful, rigorous definition of it. Without such a definition, I assert that calling a system 'scalable' is about as useful as calling it 'modern'. I encourage the technical community to either rigorously define scalability or stop using it to describe systems.*

Na Hills conclusie zijn meerdere pogingen gedaan om schaalbaarheid te definiëren. zo zijn L. Duboc, D. S. Rosenblum en T. Wicks op deze conclusie ingegaan en hebben een poging gedaan om een framework te creëren voor karakterisering en analyse van software schaalbaarheid [22]. Dit framework is te complex voor de scope van dit onderzoek, maar zij definiëren schaalbaarheid als: "quality of software systems characterized by the causal impact that scaling aspects of the system environment and design have on certain measured system qualities as these aspects are varied over expected operational ranges".

In een onderzoek over de kenmerken van schaalbaarheid en de impact op prestatie heeft A. B. Bondy [23] schaalbaarheid verdeeld in een aantal verschillende aspecten, waaronder:

- **Structural scalability** (het vermogen van een systeem om uit te breiden in een gekozen dimensie zonder ingrijpende wijzigingen in de architectuur)
- **Load scalability** (het vermogen van een systeem om elegant te presteren naarmate het aangeboden verkeer toeneemt)
- **Space scalability** (het geheugenvereiste groeit niet naar "ondraaglijke niveaus" naarmate het aantal items toeneemt)
- **Space-time scalability** (het systeem blijft naar verwachtingen functioneren naarmate het aantal items dat het omvat toeneemt)

Bondy definieert schaalbaarheid als het vermogen van een systeem om een toenemend aantal elementen, objecten en werk gracieus te verwerken en / of vatbaar te zijn voor uitbreiding.

H. El-Rewini en M. Abd-El-Barr noemen in het boek Advanced computer architecture and parallel processing [24] ook een aantal "onconventionele" definities:

- **Size scalability** (Meet de maximale hoeveelheid processors dat een systeem kan accommoderen)
- **Application scalability** (de mogelijkheid om applicatiesoftware te draaien met verbeterde prestaties op een opgeschaalde versie van het systeem)
- **Generation scalability** (de mogelijkheid om op te schalen door het gebruik van de volgende generatie (snellere) componenten)
- **Heterogeneous scalability** (het vermogen van een systeem om op te schalen met behulp van hardware- en softwarecomponenten die door verschillende leveranciers zijn gemaakt)

C. B. Weinstock en J. B. Goodenough hebben een algemeen onderzoek uitgevoerd naar schaalbaarheid [25]. Zij noemen in hun conclusie dat er voornamelijk twee betekenissen van het woord schaalbaarheid zijn:

1. De mogelijkheid om met verhoogde werkdruk om te gaan (zonder extra resources aan een systeem toe te voegen).
2. De mogelijkheid om met verhoogde werkdruk om te gaan door herhaaldelijk een kosteneffectieve strategie toe te passen om de mogelijkheden van een systeem uit te breiden.

Het valt op dat een concrete definitie van schaalbaarheid alleen duidelijk te definiëren is wanneer het in meerdere verschillende soorten is opgesplitst. Daarom zal in dit onderzoek vanaf dit punt altijd worden gespecificeerd welke soort schaalbaarheid het betreft. In dit onderzoek wordt vooral de focus gelegd op de **structural scalability** en **load scalability** uit [23] omdat deze het meest relevant zijn met betrekking tot de probleemstelling, het beter afhandelen van piekmomenten in de hoeveelheid verkeer. Wel is er wat overlapping tussen deze twee definities, zodra een systeem aan structural scalability voldoet, is een deel van load scalability ook voldaan, aangezien het schalen in een dimensie er voor zorgt dat een systeem een grotere hoeveelheid verkeer aan kan. Het deel dat dan nog mist is de functionele schaalbaarheid uit paragraaf 2.1.1. Application scalability uit [24] heeft veel ook overlapping met load scalability. Omdat load scalability iets generieker is en de twee definities van Weinstock en Goodenough [25] omvat wordt deze geprefereerd boven application scalability. De overgebleven definities zijn minder relevant voor dit onderzoek aangezien ze te maken hebben met hardware, of niet volledig toepasselijk zijn op de architectuur.

Schalen kan op twee verschillende manieren, namelijk horizontaal en verticaal. Horizontaal wilt zeggen dat er geschaald wordt door meerdere machines toe te voegen, terwijl verticaal schalen betekent dat er meer rekenkracht (als bijvoorbeeld een betere CPU of meer RAM) wordt toegevoegd aan een machine. Het gemak waarmee een horizontaal of verticaal kan schalen is **structural scalability**. Ook is bij het schaalbaar maken van systemen van belang dat het zo min mogelijk ten koste gaat van prestaties en niet meer kost dan nodig is.

2.1.1 (Niet-)functionele schaalbaarheid

Functionele schaalbaarheid is een term die in meerdere informele bronnen wordt gebruikt, maar nog nooit concreet gedefinieerd is in de literatuur. De informele

bronnen gebruiken vaak een definitie in de richting van "De mogelijkheid om een systeem te verbeteren door nieuwe functionaliteit toe te voegen zonder bestaande activiteiten te verstoren". Het is echter niet duidelijk waar deze definitie vandaan komt, en is gelijkwaardig aan de definitie van **extensibility** [26], [27]. In dit onderzoek wordt een unieke definitie van functionele schaalbaarheid gedefinieerd: *In welke mate bestaande componenten moeten worden aangepast zodra een nieuwe functionele requirement wordt toegevoegd aan het systeem, en in hoeverre deze goed blijft functioneren naarmate de hoeveelheid gebruik van het systeem toeneemt.*

Omdat functionele schaalbaarheid concreet definieert over wat er precies 'extensible' moet zijn, en in zijn definitie de hoeveelheid gebruik meeneemt, is dit een meeromvattende term dan extensibility. Functionele schaalbaarheid is een *onderdeel* van onderhoudbaarheid (Wijzigbaarheid) en load scalability. Daarnaast past het de definitie van extensibility toe in de context van functionele requirements. Het onderscheidt zich door de complexiteitsgraad van algoritmes en hard-coded limieten mee te nemen in zijn definitie. Meer over de definitie van onderhoudbaarheid is te vinden in paragraaf 2.2.

Ook is het mogelijk om een definitie te creeëren voor **Niet-functionele schaalbaarheid**. Deze definitie is buiten de scope van het onderzoek maar luidt als volgt: "In hoeverre een niet-functionele requirement kan worden verbeterd zonder bestaande componenten te belemmeren, en in welke mate de kwaliteit van die requirement acceptabel blijft naarmate het gebruik van het systeem toeneemt." Om feedback te ontvangen van de gemaakte definities is een blog geschreven. Hier is ook meer informatie te vinden over niet-functionele schaalbaarheid en hoe deze twee definities tot stand zijn gekomen. [1] Feedback is te vinden in Bijlage C.3

2.2 Onderhoudbaarheid

P. Grubb en A. A. Takang definiëren in hun boek "Software Maintenance: Concepts And Practice" onderhoudbaarheid als "The discipline concerned with changes related to a software system after delivery" [28]. In 1993 heeft IEEE een "Standard Glossary of Software Engineering Terminology" opgesteld. Deze begrippenlijst definieert onderhoudbaarheid als "the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment" [29]. Deze twee definities komen uiteindelijk op hetzelfde neer. Grubb en Takang noemen het in de context van een discipline, terwijl IEEE het als een kwaliteitseigenschap definieert. Ook specificeren Grubb en Takang het feit dat het alleen ná het opleveren gebeurt.

Grubb en Takang noemen ook een aantal redenen waarom software moet worden onderhouden:

- Ondersteuning van verplichte upgrades
- Ondersteuning van verzoeken van gebruikers om verbeteringen toe te voegen
- Om toekomstige onderhoudswerkzaamheden te vergemakkelijken

K.K. Aggarwal et al. noemen in hun onderzoek een aantal factoren die van invloed zijn op onderhoudbaarheid van software [30]:

- Leesbaarheid van de broncode

[1] De blog is op de website van Developers.nl geplaatst: https://developers.nl/blog/69/Defining-software-scalability-using-requirements

- Kwaliteit van de documentatie
- Begrijpelijkheid van software

ISO 25010 [31] definieert onderhoudbaarheid als "The degree of effectiveness and efficiency with which a product or system can be modified to improve it, correct it or adapt it to changes in environment, and in requirements" en verdeeld het in een vijftal kwaliteitseigenschappen.

- Modulariteit
- Herbruikbaarheid
- Analyseerbaarheid
- Wijzigbaarheid
- Testbaarheid

Een bekende manier om onderhoudbaarheid te meten is de zogenaamde Maintainability Index (MI). Hier is echter veel kritiek op [32]–[35]. Enerzijds is het een duidelijk cijfer om een indicatie te geven van de onderhoudbaarheid. Anderzijds is het uiteindelijk niet duidelijk welke aspecten precies voor het eindresultaat hebben gezorgd of welke acties er genomen moeten worden om de indicatie te verbeteren. Een andere methode om onderhoudbaarheid te meten is het model van I. Heitlager, T. Kuipers en J. Visser [35]. Dit model gebruikt echter een verouderd ISO standaard, namelijk ISO 9126 – de voorganger van ISO 25010 – en is dus niet relevant meer. Omdat ISO 25010 [31] de meest recente definitie heeft en over het algemeen wordt beschouwd als een effectief framework om software-kwaliteit te waarborgen, word in dit onderzoek deze definitie gebruikt als uitgangspunt.

2.3 Architectuur

P. Kruchten noemt dat software-architectuur zich bezig houdt met het ontwerp en de implementatie van de structuur op hoog niveau [16]. Dit is echter een vrij vage definitie, het is niet duidelijk wat "hoog niveau" precies inhoudt.

S. T. Albin definieert software-architectuur als "De waarneembare eigenschappen van een softwaresysteem" [36]. Ook dit is een onduidelijke definitie, het is namelijk te algemeen.

L. Bass en P. Clements, definiëren de architectuur van software als het volgende [37]: "The architecture of a software-intensive system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them". Gerespecteerde boeken als [38], [39] nemen deze definitie als uitgangspunt. Ook noemen Bass en Clements vier verschillende aspecten die behoren bij software-architectuur:

- **Statische structuur** (interne design-time elementen zoals modules, classes, packages, services, of andere zelfstandige code-eenheden en hun opstelling.)
- **Dynamische structuur** (de runtime-elementen zoals informatie-flows, parallelle of opeenvolgende uitvoering van interne taken, of de invloed die ze hebben op data en hun interacties.)
- **Extern zichtbaar gedrag** (de functionele interacties tussen het systeem en zijn omgeving. Denk aan Informatie-flows in en uit het systeem, of API's.)

- **Kwaliteitseigenschappen** (externe zichtbare, niet-functionele eigenschappen van een systeem zoals prestaties, beveiliging of schaalbaarheid.)

ISO/IEC/IEEE 42010:2011 definieert software-architectuur als "Fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution" [40]. The Open Group Architecture Framework (TOGAF) voegt nog een tweede definitie toe aan deze context [41]: "The structure of components, their inter-relationships, and the principles and guidelines governing their design and evolution over time". TOGAF is gebaseerd op een viertal architectuur-domeinen: business, data, applicatie en technische / infrastructuur architectuur. In dit onderzoek wordt alleen de technische / infrastructuur architectuur gebruikt. Dit domein omvat de IT infrastructuur, middleware, netwerken, communicaties en standaarden. Onder deze definitie passen ook de vier aspecten uit [37].

Hoofdstuk 3

Verwachtingen

Dit hoofdstuk gaat over de deelvraag "Wat zijn de wensen en eisen van Developers.nl met betrekking tot de schaal- en onderhoudbaarheid van haar huidige websites?". Om de verwachtingen duidelijk in kaart te brengen zijn discussies gevoerd met Jelle van de Haterd; stagebegeleider en kennisambassadeur DevOps. Notulen van deze discussie zijn te vinden in bijlage C.1.

3.1 Hoe ziet Developers.nl onderhoudbaarheid?

Om de workflow sneller te laten verlopen moeten zo veel mogelijk processen geautomatiseerd worden. Hieronder valt voornamelijk het automatiseren van de kwaliteitswaarborging. Het is belangrijk dat een nieuwe toevoeging of applicatie aan een aantal standaarden voldoet, zodat ontwikkelaars snel aan de slag kunnen. Jelle vind ten opzichte van daadwerkelijke code voornamelijk dat de infrastructuur aan verbetering toe is.

Developers.nl als product owner wil een nieuwe toevoeging of eventueel een volledig nieuwe applicatie kunnen bedenken en deze zo snel mogelijk geïmplementeerd zien. De workflow qua integratie en deployment moet verder geoptimaliseerd worden om Developers.nl deze snelheid te beloven.

3.2 Hoe ziet Developers.nl schaalbaarheid?

Developers.nl als product owner wil de voortgang van nieuwe toevoegingen of applicaties nauw kunnen volgen, zodat de kwaliteit hiervan op tijd gevalideerd kan worden. Dit heeft te maken met de agile werkmethodiek die Developers.nl gebruikt voor het ontwikkelen van haar interne applicaties.

Om de snelheid te verhogen waarop ontwikkelaars hun toevoegingen kunnen laten zien, ziet Developers.nl graag het concept van "Feature environments". Dit wil zeggen dat er per aangemaakte git branch gelijk gedeployed wordt naar een aparte omgeving, die dan te bekijken is door de product owner. Dit betekent dat er dus meerdere (verschillende) instanties van de applicatie naast elkaar moeten kunnen draaien. Daarnaast ziet Developers.nl graag de mogelijkheid om rekenkracht van de server te verdelen over deze instanties, waardoor er dus horizontaal geschaald moet kunnen worden. Developers.nl prioriteert de feature environments boven het verdelen van de rekenkracht, omdat het implementeren van feature environments "twee vliegen in één klap" is. Dit zorgt namelijk voor onderhoudbaarheid én het betekent dat de oplossing schaalbaar is.

3.3 Waar wilt Developers.nl meer over te weten komen?

Developers.nl wil weten of de best-practices van de 12-Factor App toepasselijk zijn op de huidige infrastructuur. Ook wil Developers.nl meer te weten komen over verschillende standaarden die zij kunnen opvolgen om de kwaliteit van hun infrastructuur te waarborgen.

3.4 Wat zijn de concrete requirements waar de oplossing aan moet voldoen?

Om schaalbaarheid te realiseren moet de onderhoudbaarheid van de infrastructuur ook op een voldoende niveau zijn, hier moeten kwaliteitsstandaarden voor onderzocht worden. Daarnaast moet de oplossing zo generiek mogelijk zijn, en dus voor meerdere applicaties toe te passen zijn. Dit betekend dus ook dat de oplossing geschikt moet zijn voor applicaties met veel verkeer.

Om requirements op te stellen wordt gebruik gemaakt van de MoSCoW-methode. Deze afkorting staat voor: **M**ust haves, **S**hould haves, **C**ould haves, en **W**on't haves [42].

Must haves

- De oplossing moet méér dan twee unieke instanties van de website naast elkaar kunnen draaien.
- De oplossing moet moet unieke instanties van de website automatisch kunnen aanmaken.
- De oplossing moet een methode bevatten om kwaliteit van nieuwe toevoegingen aan de infrastructuur automatisch te waarborgen.
- Logging en monitoring tools.
- De oplossing moet voldoen aan één of meerdere kwaliteitsstandaarden.

Should haves

- De oplossing moet generiek genoeg zijn zodat meerdere applicaties er gebruik van kunnen maken.
- De oplossing moet de website horizontaal kunnen laten schalen bij een toe- of afnemende hoeveelheid verkeer. De efficiëntie moet 1:1 zijn. Bijvoorbeeld: Één extra instantie moet 50% van het verkeer opvangen.

Could haves

- De oplossing moet de website *automatisch* laten schalen.
- De website moet op een cloud provider draaien.
- De website moet serverless draaien.

Won't haves

- Een "boilerplate" voor het opzetten van nieuwe (schaal- en onderhoudbare) projecten.

3.5 Conclusie

Developers.nl wilt onderhoudbaarheid bereiken door kwaliteitsstandaarden af te dwingen. Ook wilt Developers.nl twee soorten schaalbaarheid. Eén in de vorm van het deployen van verschillende branches in aparte omgevingen, en één in de vorm van horizontaal schalen om meer verkeer aan te kunnen. Deze twee soorten kunnen niet gerealiseerd worden zonder de onderhoudbaarheid te waarborgen. Er zijn vier concrete requirements:

- Méér dan twee unieke instanties van een applicatie naast elkaar kunnen draaien, en deze automatisch kunnen aanmaken.
- Generiek genoeg zijn zodat meerdere applicaties er gebruik van kunnen maken.
- Een manier bevatten om kwaliteit van nieuwe toevoegingen aan de infrastructuur automatisch te waarborgen.
- Horizontaal kunnen schalen bij een toe- of afnemende hoeveelheid verkeer.

Hoofdstuk 4

Technieken

Dit hoofdstuk gaat over de deelvraag "Welke standaarden en best-practices voor het waarborgen van schaal- en onderhoudbaarheid zijn relevant voor de eisen van Developers.nl?".

4.1 ISO 25010

ISO normen zijn wereldwijde standaarden, daarom is dit een ideale manier om kwaliteit te waarborgen. ISO-norm 25010 [31] is de opvolger van ISO-9126 en beschrijft software kwaliteitskenmerken in acht categorieën. Een categorie hiervan is onderhoudbaarheid. Elke categorie heeft een aantal subcategorieën, bij onderhoudbaarheid zijn dat [1]:

Modulariteit: De mate waarin een systeem of computerprogramma opgebouwd is in losstaande componenten zodat wijzigingen van een component minimale impact hebben op andere componenten.

Herbruikbaarheid: De mate waarin een bestaand onderdeel gebruikt kan worden in meer dan één systeem of bij het bouwen van een nieuw onderdeel.

Analyseerbaarheid: De mate waarin het mogelijk is om effectief en efficiënt de impact, van een geplande verandering van één of meer onderdelen, op een product of systeem te beoordelen, om afwijkingen en/of foutoorzaken van een product vast te stellen of om onderdelen te identificeren die gewijzigd moeten worden.

Wijzigbaarheid: De mate waarin een product of systeem effectief en efficiënt gewijzigd kan worden zonder fouten of kwaliteitsvermindering tot gevolg.

Testbaarheid: De mate waarin effectief en efficiënt testcriteria vastgesteld kunnen worden voor een systeem, product of component en waarin tests uitgevoerd kunnen worden om vast te stellen of aan die criteria is voldaan.

Om de applicatie "onderhoudbaar" te noemen moeten alle subcategorieën voldoende worden vervuld. Er moet een analyse worden uitgevoerd per subcategorie over eventuele tekortkomingen.

4.2 De Twelve-Factor App

A. Wiggins [14] heeft een methodologie opgezet om moderne, schaalbare en onderhoudbare web-applicaties te bouwen. De methodologie past goed bij de probleemstelling, het minimaliseren van de kosten en tijd die het kost om nieuwe ontwikkelaars aan het project te laten werken en de gemakkelijkheid van het schalen. Daarnaast wilde Developers.nl graag weten of de 12-Factor app relevant kan zijn voor de huidige infrastructuur. Ook zorgt het voor structural scalability, draagbaarheid

[1] Vertaling van ISO-norm uit wikipedia: https://nl.wikipedia.org/wiki/ISO_25010

tussen uitvoeringsomgevingen, de mogelijkheid om te deployen op moderne cloud platformen en een minimale divergentie tussen development en productie waardoor CD gemakkelijk wordt om te implementeren.

De methodologie heeft 12 factoren (best-practices) die voor deze eigenschappen zorgen. Deze methodologie wordt vaak aangeraden door professionals, en is ook een methodologie die Developers.nl graag terug ziet in haar applicaties. Kritiek op de 12-factor app gaat voornamelijk over het feit dat het gelimiteerd is tot Heroku [43]. Vijf jaar nadat Wiggins de 12-Factor App heeft opgesteld is K. Hoffman aan de slag gegaan met een boek genaamd "Beyond the 12-Factor App" [44]. Dit boek heeft als doel om de 12 factoren concreter te definiëren en voegt daarnaast nog 3 extra factoren toe om applicaties in de cloud niet alleen te laten functioneren maar ook te laten gedijen. Hierdoor is de methodologie ook niet meer persé gericht op Heroku. Deze factoren zijn telemetry, security, en het concept "API first". Eigenlijk is een betere benaming voor dit onderzoek dus de "Fifteen-Factor App". In bijlage B.1 is een gevolg samen met een uitleg bij elke factor geplaatst. Zodra een factor te maken heeft met onderhoudbaarheid is er een subcategorie van ISO 25010 onderhoudbaarheid bij geplaatst. Alle 15 factoren zullen worden meegenomen bij het behandelen van de deelvragen.

4.3 Schaalbaarheids-controle

In het onderzoek van Weinstock en Goodenough [25] noemen zij dat het niet echt mogelijk is om te testen of een systeem schaalbaar is. Wel zijn er methoden om de schaalbaarheid te waarborgen:

- Onderzoek de "performance curves" en karakteriseer deze met een Big O notatie. Hoe veranderen deze curves bij het aanpassen van een schaalstrategie?
- Identificeer mechanismen om knelpunten aan het licht te brengen of waar aannames van het schaalbaarheids- ontwerp beginnen te worden geschonden. Deze knelpunten hebben vooral te maken met de eerste betekenis van Weinstock en Goodenough. Er moet gecontroleerd worden op de toenemende administratieve werkdruk, de "hard-coded" limieten op capaciteit, de user-interface en de complexiteitsgraad van algoritmen. De schaalbaarheids-aannames gaan over het onderzoeken hoe de uitbreiding van een systeem nieuwe problemen kan onthullen. Zodra een systeem zich uitbreidt is er een grotere kans op errors in de systeemconfiguratie, "zeldzame" errors komen vaker voor, is het belangrijk dat een probleem in het systeem gelokaliseerd blijft, en kan het een stuk complexer en lastiger worden om het systeem te begrijpen.
- Voer een SWOT analyse uit op de schaalbaarheids-strategie.
  - **S**trengths (de soorten groei waar de strategie voor ontworpen is)
  - **W**eaknesses (de soorten groei waar de strategie niet voor ontworpen is)
  - **O**pportunities (mogelijke veranderingen in werklast of technologie die de strategie goed zou kunnen benutten)
  - **T**hreats (mogelijke veranderingen in de werklast of technologie die de strategie in twijfel zouden kunnen trekken)

Door het karakteriseren van de performance curves met een Big O notatie wordt voornamelijk de load scalability gewaarborgd. Het identificeren van de knelpunten en aannames samen met het uitvoeren van een SWOT analyse zorgt voor het waarborgen van de functionele schaalbaarheid.

4.4 Conclusie

De 12-Factor App is een methodologie die twaalf best-practices samenvoegt om moderne, schaal- en onderhoudbare web-applicaties te bouwen. Het boek “Beyond the 12-factor app” [44] is hierop verder gegaan door nog een drietal factoren toe te voegen. Door een applicatie te evalueren op deze vijftien factoren, samen met de definitie van ISO-25010 [31] is te beoordelen of deze schaal- en onderhoudbaar is. Om de schaalbaarheid van een systeem te waarborgen zijn van Weinstock en Good-enough [25] een geschikte manier.

Hoofdstuk 5

Huidige situatie

Dit hoofdstuk gaat over de deelvraag "Hoe onderhoud- en schaalbaar zijn de huidige websites van Developers.nl met betrekking tot de relevante kwaliteitsstandaarden?".

5.1 Huidige Architectuur

In dit onderzoek wordt voornamelijk gefocussed op de website. Dit is de applicatie die het meest frequent gebruikt wordt, en dus de meeste aandacht verdiend. De huidige website is een combinatie van een PHP & Symfony back-end API en Content Management Systeem, samen met een React + next.js front-end. De infrastructuur is momenteel gebouwd op Docker(-compose) + Ansible. Bitbucket pipeline wordt gebruikt voor het Continuous Integration / Deployment.

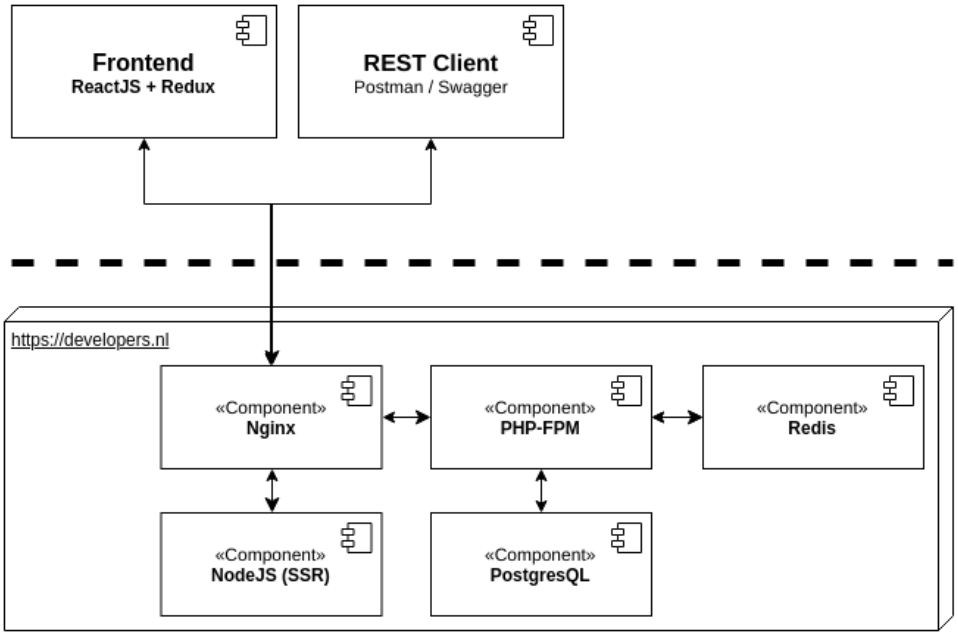
In figuur 7.2 is een component diagram te vinden van de huidige website structuur. De front-en backend structuur bevat 5 docker containers:

- **PHP-FPM** (back-end)
- **Nginx** (front-en backend)
- **Redis** (back-end)
- **NodeJS** (front-end)
- **PostgreSQL** (back-end)

PHP-FPM is een FastCGI Process Manager, deze Container serveert de Symfony "FosREST" API en het Content Management Systeem. De NodeJS container serveert een statische Next.js React applicatie en maakt gebruik van Server Side Rendering. Er zit een Nginx reverse proxy in die kiest om een request naar de back-end of de front-end te laten gaan. Redis is een Key-Value Database die gebruikt wordt voor het cachen, en een PostgreSQL container als database. De Bitbucket Pipeline gebruikt Ansible om op de servers de geüpdatete containers te pullen en te starten.

Voor zowel de front- als backend is één monitoring tool genaamd "Sentry" geïmplementeerd. Sentry creëert een duidelijk overzicht voor alle errors die opkomen in productie.

Ook heeft Developers.nl een "Employee Management Systeem" (EMS) gebouwd. Deze heeft een soortgelijke structuur aan de website. Het EMS bevat zeer veel gevoelige informatie en het is dus van hoog belang dat dit goed beveiligd is.



FIGUUR 5.1: Infrastructuur website front-en backend [45]

5.2 Metingen

Nu de infrastructuur in kaart is gebracht luidt de vraag; hoe schaalbaar is deze infrastructuur eigenlijk? Om dit te beantwoorden worden de verschillende definities van schaalbaarheid individueel behandeld.

5.2.1 Structural scalability

Definitie: Het vermogen van een systeem om uit te breiden in een gekozen dimensie zonder ingrijpende wijzigingen in de architectuur. Bij structural scalability horen factor 2 **(API First)** en 5 **(Configuration, credentials, and code)** van de 15-factor app.

API First

De website van Developers.nl is momenteel in 2 delen gesplitst: de React Front-end en de PHP API als back-end. Deze worden apart ontwikkeld, waardoor dus het principe altijd wordt toegepast. Daarnaast heeft het EMS geen API, en is dus out-of-scope voor deze factor.

Configuration

Een test om te bewijzen dat alle configuratie correct uit de code is verwerkt, is of de applicatie op elk gewenst moment open-source kan worden gemaakt zonder geclassificeerde informatie vrij te geven.

Voor de website wordt er gebruik gemaakt van docker-secrets en ansible-vault. Deze combinatie zorgt ervoor dat er nooit wachtwoorden, API sleutels en dergelijke plain-text in versiebeheer komt te staan. Deze secrets worden uiteindelijk in de containers als environment variabelen opgeslagen en uitgelezen door Symfony. In

het EMS is deze techniek nog niet gebruikt en staan credentials wél plaintext in de repository.

Om aan factor 5 te voldoen moet de configuratiefiles niet per specifieke omgeving (productie, test, staging) gegroepeerd worden maar moeten juist individueel per deployment geregeld worden. Dit gebeurt in zowel het EMS als de website, de bitbucket pipeline heeft zijn eigen specifieke environment variabelen om te gebruiken en de variabelen in de docker containers worden meegegeven in de algemene docker-compose file die in elke deployment hetzelfde zal zijn.

5.2.2 Load scalability

Definitie: Het vermogen van een systeem om elegant te presteren naarmate het aangeboden verkeer toeneemt. Bij load scalability horen factor 12 **(stateless processes)**, 13 **(concurrency)** en 7 **(disposability)** van de 15-factor app methodologie.

Stateless processes

Factor 12 vereist dat de applicatie als één of meerdere "stateless processes" moet uitgevoerd worden. Bij de PHP containers worden geüploade bestanden weggeschreven naar een volume, dit zorgt ervoor dat de container niet volledig stateless meer is. Ook zijn databases in docker containers geplaatst, dit is een stateful process aangezien het van belang is dat niet alle data verloren gaat zodra de container stopt.

Concurrency

Voor factor 13 is het van belang dat een applicatie horizontaal uit te schalen is. Zolang de applicatie aan factor 7 (Disposability) en 12 (Stateless processes) voldoet, zit deze factor goed [44]. Er is alleen nog geen manier geïmplementeerd om daadwerkelijk meerdere Docker containers naast elkaar te draaien of te managen.

Disposability

Voor factor 7 moet een applicatie opstarttijd minimaliseren. Zodra de docker images de initiële buildtime voorbij zijn kan de applicatie snel uit en aan worden gezet.

Ook vereist factor 7 dat processen netjes worden afgesloten zodra ze een `SIGTERM` ontvangen. Zodra een docker container met `docker stop <container>` gestopt wordt zal er een SIGTERM worden gestuurd naar de draaiende processen. De vier containers met processen zijn PostgreSQL, PHP-FPM, Nginx en Redis. Deze sluiten allemaal netjes af, de outputs zijn te zien in Bijlage A.2.

Ook moeten de processen bestendig zijn tegen "sudden death". Om dit te simuleren kan `docker kill <container>` gebruikt worden om een `SIGKILL` te sturen naar de hoofdprocessen. In bijlage A.3 is te zien dat alle containers na een `docker kill` zonder problemen weer kunnen opstarten.

5.2.3 Weinstock & Goodenough controle

Om de functionele schaalbaarheid te waarborgen zullen de 3 methoden van Weinstock en Goodenough [25] uitgevoerd worden. Performance curves zullen worden gevisualiseerd, knelpunten zullen worden uitgelicht en een SWOT analyse op de schaalbaarheid zal worden uitgevoerd.

Om de performance curves te visualiseren zal een load-test worden uitgevoerd. Er zijn hier meerdere tools voor vergeleken, waaronder:

- https://loader.io/
- https://gatling.io/
- https://k6.io/
- http://tsung.erlang-projects.org/

De gratis versie van loader.io is niet genoeg voor de wensen van de test, voor gatling.io is Ruby kennis nodig, en voor Tsung worden de tests in XML geschreven wat het lastig maakt om de load op te schalen. Uiteindelijk is gekozen voor K6 omdat zo goed als elke ontwikkelaar genoeg Javascript kennis heeft om deze tool te gebruiken. Ook heeft k6 een eenvoudige manier om de hoeveelheid Virtual Users (VU) geleidelijk te verhogen. Om de uitkomsten te visualiseren is InfluxDB samen met Grafana gebruikt. In bijlage A.1 is de implementatie hiervan te vinden.

TODO: PERFORMANCE CURVES

Één limiterende factor bij het schalen van de website is de hoeveelheid opslag. Voornamelijk omdat het CMS dubbel functioneert als "file-server". Daarnaast bevat de content van de website een grote hoeveelheid foto's en video's, waardoor het opslaggebruik snel kan oplopen. Door het commando `$ df -h` is te zien dat 27G – oftewel 57% – van de totale 49G wordt gebruikt.

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        49G   27G   21G  57% /
```

Bij nader onderzoek is te zien dat de directory die gebruikt wordt voor statische bestanden (waar ook de geüploade bestanden in zitten) maar `278M` in beslag neemt, dus er is nog veel ruimte (21G) voor uitbreiding in dit aspect en zal voor een redelijk lange tijd geen probleem vormen:

```
root@developers:/etc/developers.nl# du -shc ./static/
278M        ./static/
```

Omdat 27G nogal veel leek voor wat er op de server draait is er onderzoek uitgevoerd naar de oorzaak. Het blijkt dat er veel ongebruikte oude Docker volumes en images op de server staan. Na een `$ docker system prune --volumes` en een `$ docker image prune -a` is er 14G vrijgemaakt:

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        49G   13G   34G  28% /
```

Qua schaalbaarheid is dit dus een verbeterpunt. De images en volumes moeten automatisch worden opgeschoond.

Een andere factor zou kunnen zijn dat de rate-limits van externe API's wordt bereikt. De twee externe API's die nu worden gebruikt zijn `meetup.com` voor de TechNights en `bullhorn.com` voor de vacatures. Voor Bullhorn heeft developers.nl de "Enterprise Edition", dit betekend dat Developers.nl 50 API sessies tegelijk kan hebben, en maximaal 2.000.000 calls per dag heeft. Aangezien de website niet dichtbij deze getallen komt, en hoogstwaarschijnlijk op lange termijn niet gaat halen zit dit goed. Toch worden de responses gecached in Redis waardoor de calls nog minder zullen zijn. Voor de API van Meetup zijn er maximaal 200 requests per uur en maximaal 200 results per request. Ook dit verkeer heeft de website voorlopig

nog niet, maar om toch zeker te zijn dat dit niet wordt bereikt worden ook deze responses in Redis opgeslagen. De cachemethode is generiek genoeg opgezet waardoor het voor toekomstige API's ook kan worden toegepast.

Om de schaalbaarheid nog verder te analyseren is een SWOT analyse uitgevoerd op basis van de aanbeveling van Weinstock en Goodenough.

Strengths

- Stateless processes - Concurrency - Disposability

Weaknesses

- De hoeveelheid opslag - Hoge hoeveelheid verkeer - Nog geen manier van automatisch schalen geïmplementeerd

Opportunities

- Het daadwerkelijk schalen door middel van container orchestration

Threats

- De hoeveelheid onderhoud dat een nieuwe schaalstrategie met zich mee brengt.

5.2.4 Onderhoudbaarheid

De definitie van onderhoudbaarheid waarvan wordt uitgegaan in dit onderzoek luidt als volgt: "The degree of effectiveness and efficiency with which a product or system can be modified to improve it, correct it or adapt it to changes in environment, and in requirements". Om de onderhoudbaarheid van de huidige infrastructuur te analyseren worden de vijf subcategorieën van ISO-25010 [31]; Modulariteit, Herbruikbaarheid, Analyseerbaarheid, Wijzigbaarheid en Testbaarheid individueel behandeld.

Modulariteit

De mate waarin een systeem of computerprogramma opgebouwd is in losstaande componenten zodat wijzigingen van een component minimale impact heeft op andere componenten.

De website van Developers.nl is opgebouwd in twee applicaties, de Node & React frontend samen met de PHP & Symfony backend. Een wijziging in één van de API endpoints van de backend zou kunnen betekenen dat de frontend breekt. Om dit te voorkomen is een versioning systeem [1] geïmplementeerd waardoor er zonder problemen individueel de front-of backend gedeployed kan worden.

Voor de 15-Factor App geldt Factor 1 **(One codebase, one application)**, 3 **(Dependency management)** en 8 **(Backing services)**. Factor 1 vereist dat er per applicatie één enkele codebase is. De huidige situatie is dat er één repository is voor de back-end van de website, één voor de front-end van de website, en één voor het EMS. De regel wordt gebroken omdat het opbouwen van de infrastructuur met Ansible op elke repository voorkomt. Om aan Factor 3 te voldoen wordt Composer gebruikt voor het managen van de dependencies. Daarnaast worden waardes als

[1] Voor de API wordt FOSRestBundle gebruikt, deze heeft een eigen implementatie van versioning, zie: https://symfony.com/doc/master/bundles/FOSRestBundle/versioning.html

de database host of het Redis adres in environment variabelen opgeslagen, waardoor factor 8 wordt voldaan. Om dit te implementeren worden backing services gedefinieerd als een handle, deze ziet er voor de database als volgt uit:
`postgresql://username:password@developers.nl/database`

Herbruikbaarheid

De mate waarin een bestaand onderdeel gebruikt kan worden in meer dan één systeem of bij het bouwen van een nieuw onderdeel.

Hoewel de Docker containers en Ansible infrastructuur generiek zijn opgesteld zijn er toch specifieke aanpassingen voor de website en voor het EMS. Dit betekent dat bijvoorbeeld de PHP Docker image van de website niet dezelfde is als de PHP Docker image voor het EMS. Ook is de door-ansible-opgebouwde infrastructuur nog niet herbruikbaar voor meerdere projecten, aangezien het in dezelfde codebase zit als de applicatie.

Analyseerbaarheid

De mate waarin het mogelijk is om effectief en efficiënt de impact, van een geplande verandering van één of meer onderdelen, op een product of systeem te beoordelen, om afwijkingen en/of foutoorzaken van een product vast te stellen of om onderdelen te identificeren die gewijzigd moeten worden.

Voor de 15-Factor App geldt Factor 6 **(Logs)**, 10 **(Administrative processes)** en 14 **(Telemetry)**. Voor factor 6 is het belangrijk dat alle relevante logs naar de `stdout` worden gestuurd, dit gebeurt voor alle containers waardoor de logs gemakkelijk te zien zijn via `$ docker logs <container>` . Voor factor 10 moeten alle administrative processes als individuele processen gedraaid worden. Deze processen als bijvoorbeeld database migrations zijn een simpel commando. In het geval van database migrations is dit `$ bin/console doctrine:migrations:migrate` [2]. Deze commando's worden meegenomen bij het bouwen van de docker image waardoor het automatisch wordt uitgevoerd, maar wél als een apart proces. Factor 14 vertelt dat de applicatie voldoende moet gemonitord worden. Er zijn voor zowel het EMS als de website geen monitoring tools in gebruik voor performance. Wel is Sentry geïmplementeerd, een monitoring tool die gericht is op errors.

Wijzigbaarheid

De mate waarin een product of systeem effectief en efficiënt gewijzigd kan worden zonder fouten of kwaliteitsvermindering tot gevolg.

Voor de 15-Factor app geldt Factor 4 **(Design, build, release and run)**, 9 **(Environment parity)** en 11 **(Port binding)**. Factor 4 is volledig van kracht, allereerst is de design fase het beslissen wat voor features er in de volgende release komen. De build stage is het bouwen van de Docker images om die klaar te maken voor de volgende fase – release – waar de Docker images in de environment worden geplaatst en uiteindelijk voor de laatste fase gerund worden. Factor 9 vereist dat verschillende environments als development, test en productie zo gelijk mogelijk aan elkaar zijn. Docker maakt dit een stuk gemakkelijker, en is geïmplementeerd in zowel het EMS als de website. De verschillen tussen environments zijn minimaal. Factor 11 vereist

[2] Dit is onderdeel van de DoctrineMigrationsBundle, zie:
https://symfony.com/doc/master/bundles/DoctrineMigrationsBundle/index.html
voor meer informatie.

dat applicaties services via port binding exporteren. Voor PHP is dit niet de best ondersteunde manier van werken [46]–[48], aangezien PHP ontworpen is om een webserver te gebruiken. Er is wel een library voor beschikbaar genaamd ReactPHP, maar dit is redelijk onbekend en heeft dus als gevolg niet voldoende ondersteuning om PHP-FPM samen met Nginx weg te concurreren. Hierdoor zou het gebruik van port binding met PHP de onderhoudbaarheid juist verlagen.

Testbaarheid

De mate waarin effectief en efficiënt testcriteria vastgesteld kunnen worden voor een systeem, product of component en waarin tests uitgevoerd kunnen worden om vast te stellen of aan die criteria is voldaan.

Er is een implementatie om test-coverage te visualiseren, maar er is niks in de ontwikkelstraat dat ervoor zorgt dat de coverage minimaal hetzelfde blijft.

5.3 Conclusie

Om de deelvraag "Hoe onderhoud- en schaalbaar zijn de huidige websites van Developers.nl met betrekking tot de relevante kwaliteitsstandaarden?" te beantwoorden zijn door dit onderzoek meerdere punten van verbetering gevonden. Een verbeterpunt in de schaalbaarheid is de hoeveelheid opslag van de server. Deze kan snel vol raken door ongebruikte Docker volumes en images die ontstaan bij een deployment.

Ook wordt factor 1 **(One codebase, one application)** van de 15-Factor App niet volledig opgevolgd. De infrastructuur wordt op meerdere plekken opgebouwd en zou netter staan in een aparte codebase. De applicaties voldoen aan factor 13 (Concurrency) maar er wordt nog geen gebruik van gemaakt.

Factor 11 **(Port binding)** is voor PHP geen goed idee aangezien PHP juist ontworpen is om een webserver ervoor te hebben, dit verlaagt dus juist de onderhoudbaarheid. Er zijn voor zowel het EMS als de website geen monitoring tools in gebruik voor performance, dat betekend dat factor 14 **(Telemetry)** beter kan. Er is geen concrete manier om tests uit te voeren of aan testcriteria is voldaan. Hierdoor is de Testbaarheid van de systemen minimaal.

Hoofdstuk 6

Verbeteringen

Dit hoofdstuk gaat over de deelvraag "Wat voor verbeteringen ten aanzien van schaal- en onderhoudbaarheid kunnen worden toegepast op de huidige websites van Developers.nl?". Gebaseerd op de conclusie uit hoofdstuk 5 en de requirements van Developers.nl zijn verschillende technieken onderzocht om de gevonden verbeterpunten te verbeteren. Daarna zijn deze gevonden technieken op prioriteit geordend door middel van de requirements.

6.1 Feature-environments

Met feature-environments is het mogelijk om een staging-omgeving te creëren voor elke individuele feature branch. Dit betekent dat de product owners en developers elke feature afzonderlijk van andere features kunnen testen, om deze daarna naar productie te deployen. De feature-environment workflow is niet erg conventioneel of populair, aannemelijk omdat het niet gemakkelijk is om te implementeren. De meest relevante informatie voor dit onderzoek is een blog van Christian Lüdemann [49], hoewel de implementatie niet past bij de huidige situatie van Developers.nl. Uit onderzoek over verschillende technieken zijn de volgende opties gekomen:

- Een Kubernetes cluster met individuele namespaces voor elke environment.
- Tools als `https://platform.sh/` of `https://continuouspipe.io/` .
- Een dynamische reverse proxy die requests naar verschillende docker netwerken stuurt.
- Een dynamische reverse proxy die requests naar verschillende aparte VM's stuurt die gemanaged worden met Proxmox, of een cloud provider.

Een kubernetes cluster net namespaces valt af, dit heeft te maken met het onderhoudswerk van een cluster. Meer hierover is te lezen in paragraaf 6.4. Ook Docker Swarm is geen optie, Swarm heeft geen equivalent van kubernetes namespaces en er kunnen niet meerdere nodes op één machine staan. Tools als Platform.sh of Continuouspipe.io hebben een geïntegreerde oplossing voor feature environments, maar voegen veel overhead toe. Bovendien kost Platform.sh geld, waar Developers.nl het niet voor over heeft.

Er zijn twee opties voor het maken van feature environments met een reverse proxy. Het gebruik maken van aparte VM's versus het aanmaken van docker netwerken. Het zusterbedrijf van Developers.nl – gemvision – maakt gebruik van Proxmox om hun Virtual Machines te beheren. Hoewel Proxmox een goede tool is om de VM's te beheren, is het echt gebaseerd op enterprise settings. Dit maakt het lastiger voor andere ontwikkelaars om goed gebruik te maken van deze tool. Aangezien Docker

netwerken al geïntegreerd zijn in Docker voegt deze oplossing veel minder overhead toe. Dit is dus de beste optie.

Er zijn verschillende reverse proxies. Relevante voor dit project zijn:

- Traefik
- jwilder/Nginx-Proxy
- HAProxy

Elke reverse proxy heeft zijn voor- en nadelen. De Website en het EMS maken al gebruik van Nginx als FastCGI webserver voor PHP-FPM. Daarom is nginx-proxy een interessante optie[1]. Nginx-proxy is een abstractielaag boven Nginx die gebruik maakt van de Docker socket om op die manier requests naar de juiste plek te sturen. Het nadeel is dat Nginx hier niet expliciet voor bedoeld is, waardoor een aantal tekortkomingen aanwezig zijn, zoals ondersteuning voor Docker Swarm[2]. Traefik werkt goed samen met Docker (swarm), aangezien Traefik ingebouwde service-discovery heeft voor docker containers en Let's Encrypt. Bovendien heeft Traefik een minder steile learning curve door de aanwezige documentatie. Het nadeel is dat het meer overhead creëert doordat het een compleet aparte en nieuwe tool is die moet worden toegevoegd. Dit nadeel is ook aanwezig bij HAProxy. HAProxy is snel en capabel voor load balancing, maar is complex om op te zetten zodra het op feature environments toekomt, omdat er geen ingebouwde service-discovery aanwezig is.

In het geval van Developers.nl is Traefik de beste optie, omdat het de minste complexiteit bevat en goed samen met Docker en Docker Swarm werkt. Een nadeel van de service-discovery oplossingen is dat de Docker Socket moet worden geëxposeerd. Dit is een groot beveiligingslek[3]. Zodra een hacker met kwaadaardige intenties Traefik weet te kapen betekent dat dat de hacker root toegang heeft op de host machine. Een oplossing hiervoor is het exposen van de Docker Socket door het Transmission Control Protocol (TCP) en deze te beveiligen door middel van Transport Layer Security (TLS).

In bijlage C.2 zijn Slack conversaties met Jelle te vinden die te maken hebben met de feature environments, en over het beveiligen van de Docker socket.

6.2 Één generieke infrastructuur

One codebase, One application is Factor 1 van de 15-Factor App en zorgt voor herbruikbaarheid van ISO 25010. De huidige infrastructuur van de Website en het EMS wordt opgebouwd met Ansible. "Ansible is a radically simple IT automation engine that automates cloud provisioning, configuration management, application deployment, intra-service orchestration, and many other IT needs" [5]. Kort gezegd is Ansible een Configuration Management (CM) tool.

Zoals de structuur nu is opgebouwd word er voor elke applicatie een apart stuk IaC geschreven. Om modulariteit en herbruikbaarheid van ISO 25010 [31] te verbeteren is het mogelijk om één centrale, algemene infrastructuur repository te maken waar installaties (stukken code dus) als Docker of databases en user-management kunnen worden hergebruikt voor meerdere applicaties.

Andere IaC tools als Chef of Puppet, kunnen ook voor dit doeleinde worden gebruikt, maar aangezien Ansible al gebruikt wordt is het niet efficiënt om dit om te

[1] https://github.com/jwilder/nginx-proxy
[2] https://github.com/jwilder/nginx-proxy/issues/927
[3] https://github.com/containous/traefik/issues/4174

herschrijven naar iets anders. Het is mogelijk om nog een hoger level van abstractie toe te voegen door een IaC tool als Terraform te gebruiken om een machine in te richten.

6.3 Policy-as-Code

Developers.nl heeft in haar website al een aantal manieren om kwaliteit van code te waarborgen. Automatische unit, integration, functional en end-to-end tests. Er mist een manier om de infrastructuur te waarborgen op kwaliteit. Dit is een verplichtte requirement: "De oplossing moet een methode bevatten om kwaliteit van nieuwe toevoegingen aan de infrastructuur automatisch te waarborgen". Door policies zijn er duidelijke standaarden waaraan moet worden voldaan. Policies kunnen in meerdere vormen voorkomen [50], waaronder:

- **Compliance Policies.** Deze policies verzekeren dat nieuwe toevoegingen voldoen aan standaarden als bijvoorbeeld AVG of SOC.
- **Security Policies.** Deze policies verdedigen de integriteit van de infrastructuur door bijvoorbeeld te verzekeren dat bepaalde poorten niet open staan.
- **Operational Excellence.** Deze policies voorkomen uitval of verslechtering van geleverde services, bijvoorbeeld door nieuwe configuratie te valideren.

Vóór Policy-as-Code (PaC) werden deze policies opgeschreven door iemand en handmatig gecontroleerd. De – nog vrij nieuwe – techniek PaC richt zich op het automatiseren van dit proces door policies te kunnen definiëren in de vorm van code.

De techniek zorgt ervoor dat configuratie getest kan worden op kwaliteit. Daarnaast komt het voordeel dat de policies opgeslagen kunnen worden in versiebeheer. Hierdoor kunnen de policies ook worden hergebruikt. Dit sluit goed aan met de huidige Infrastructure-as-Code (IaC) oplossing die Developers.nl gebruikt voor het inrichten van haar servers.

Het automatiseren van deze kwaliteitscontroles verhoogt de onderhoudbaarheid aanzienlijk. Als we uitgaan van de ISO-25010 definitie van onderhoudbaarheid [31] zorgt het omzetten van een handmatige naar een geautomatiseerde controle voor betere **analyseerbaarheid** op veranderingen van de systemen, omdat afwijkingen en/of fouten gemakkelijker worden vastgesteld. Daarnaast zijn policies in principe testcriteria, waardoor als gevolg ook de **testbaarheid** van de systemen stijgt bij het implementeren van policies as code. Ook zorgen policies voor **wijzigbaarheid**, aangezien het systeem gewijzigd kan worden zonder fouten of kwaliteitsverminderingen als gevolg omdat het simpelweg niet geïmplementeerd mag worden zodra een wijziging niet aan een policy voldoet. Bovendien zijn security policies handig voor het beschermen van de gevoelige data die het EMS bevat.

Er zijn twee technieken om PaC te implementeren. HashiCorps "Sentinel" en "Open Policy Agent" (OPA). In verband met de reden dat Sentinel closed-source is, is OPA de betere keuze. Dit past beter bij de bedrijfscultuur, slogan en budgetwensen van Developers.nl. Ook is Sentinel alleen toepasbaar op hashiCorp producten, waardoor de techniek een stuk beperkter is.

6.4 Container Orchestration

Om gebruik te maken van Factor 13 **(Concurrency)** moet de applicatie horizontaal en verticaal kunnen schalen. Dit heeft ook te maken met de should-requirement "De oplossing moet de website horizontaal kunnen laten schalen bij een toe- of afnemende hoeveelheid". Omdat de systemen bij Developers.nl op Docker containers draaien moet er een manier zijn om deze te beheren. "Container orchestration" platforms helpen bij het automatiseren van alle aspecten behorend bij het beheren van containers. Dit doen zij door meerdere containers als één entiteit te beschouwen – voor doeleinden van beschikbaarheid, schaalbaarheid, netwerken en de initiële containerimplementatie [51].

Er zijn twee technieken voor container orchestration leidend in de context van Docker, namelijk Docker Swarm of Kubernetes (K8s). Over het algemeen is Swarm een stuk gemakkelijker en minder complex dan K8s. Dit zou betekenen dat als er rekening word gehouden met onderhoudbaarheid, swarm de beste keuze is om te gebruiken. Maar om de grootste hoeveelheid controle over de containers te hebben is K8s de juiste tool. Ook wordt K8s beter ondersteund door cloud providers doordat AWS, GCP, en Azure een speciale service bieden om K8s toe te passen. Dit heeft te maken met het feit dat de community van K8s ook een stuk groter is vergeleken met Swarm. Het is wel mogelijk om Swarm te gebruiken met de cloud services maar het er is geen out-of-the-box service zoals er bij K8s wel is. Als er gekeken wordt naar de wensen van Developers.nl en de scope van dit onderzoek is Swarm de meest passende keuze. De meeste prioriteit qua schalen gaat niet specifiek naar het automatiseren en managen van hoge hoeveelheden verkeer maar naar het draaien van meerderen omgevingen naast elkaar, om zo verschillende features apart te deployen of A/B te testen.

6.5 Opschonen Docker volumes en images

De images en volumes moeten automatisch worden opgeschoond.

6.6 Logging & Monitoring

Logging en monitoring is een verplichtte requirement vanuit Developers.nl. Uit onderzoek in hoofdstuk 5 blijkt dat Telemetry, Factor 14 van de 15-Factor app nog aan verbetering toe is. [44] noemt drie verschillende categorieën van data om te monitoren in een applicatie:

- Application performance monitoring (APM)
- Domain-specific telemetry
- Health and system logs

Om de juiste keuze van monitoring tool te maken is het belangrijk om te specificeren wat Developers.nl graag gemonitord ziet worden. Na een overleg met Jelle is besloten om te beginnen met APM als de hoeveelheid CPU/geheugen dat wordt gebruikt. Om dit te realiseren kan een tool als Prometheus samen met Grafana worden gebuikt, waardoor het erg simpel is om in de toekomst de hoeveelheid en soort data dat wordt gemonitord uit te breiden.

6.7 Codecov

Om de testbaarheid te verbeteren moet een tool worden gebruikt om de testcoverage van nieuwe toevoegingen te waarborgen. De meest prominente tool hiervoor is codecov. Het zorgt voor een duidelijk overzicht van de coverage tools en kan direct bij pull-requests nakijken of de nieuwe features wel voldoende zijn getest.

6.8 Prioriteiten

Feature Environments Generieke infrastructuur Policy-as-code Container orchestration Opschonen docker volumes en iages Logging & Monitoring Codecov

6.9 Conclusie

Om de kenmerken modulariteit en herbruikbaarheid van ISO 25010 te verbeteren kan er een centrale infrastructuur IaC repository gemaakt worden met Ansible. Om de kenmerken analyseerbaarheid, testbaarheid en wijzigbaarheid te verbeteren kunnen policies worden afgedwongen door middel van PaC. Om de testbaarheid te waarborgen kan een tool als Codecov worden gebruikt.

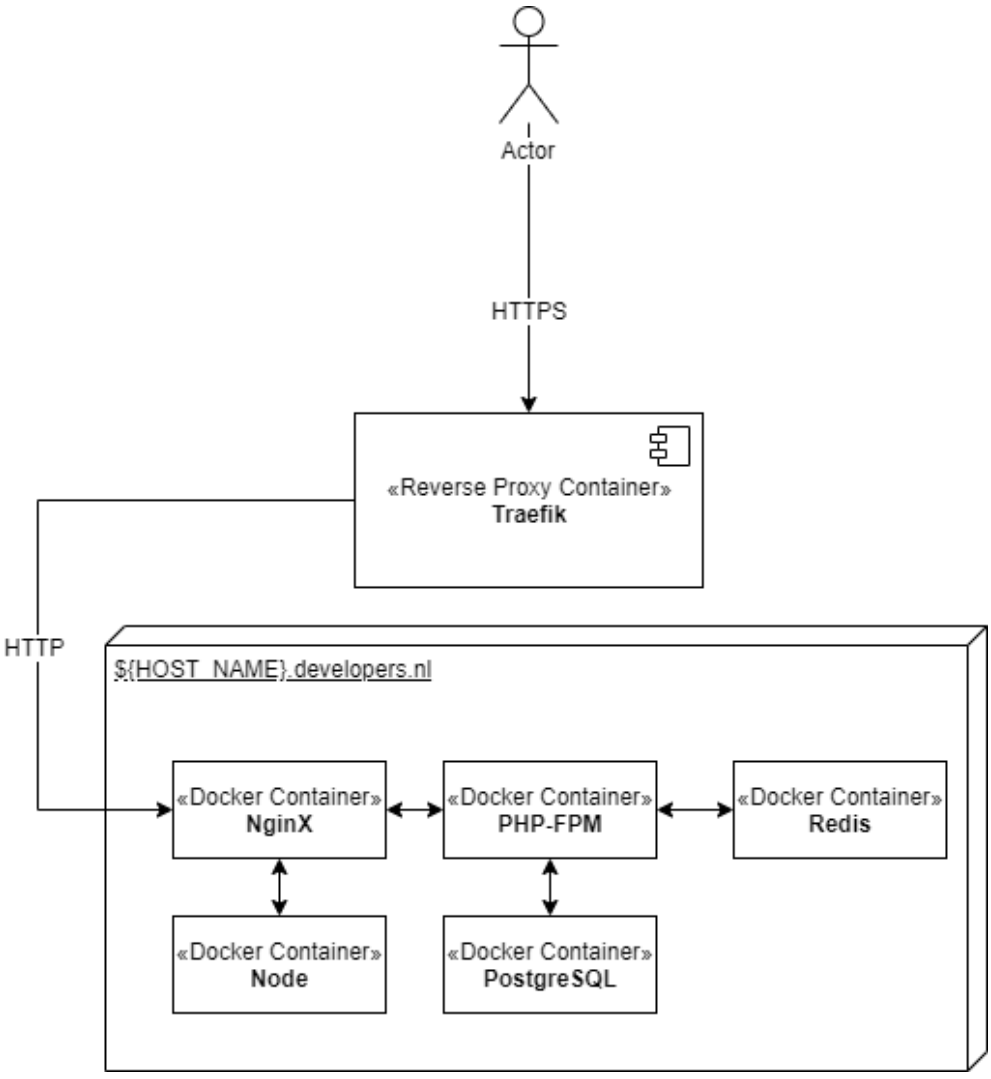
Hoofdstuk 7

Implementatie

Dit hoofdstuk gaat over de deelvraag “Hoe kunnen de gekozen verbeteringen ten aanzien van schaal- en onderhoudbaarheid geïmplementeerd worden?”

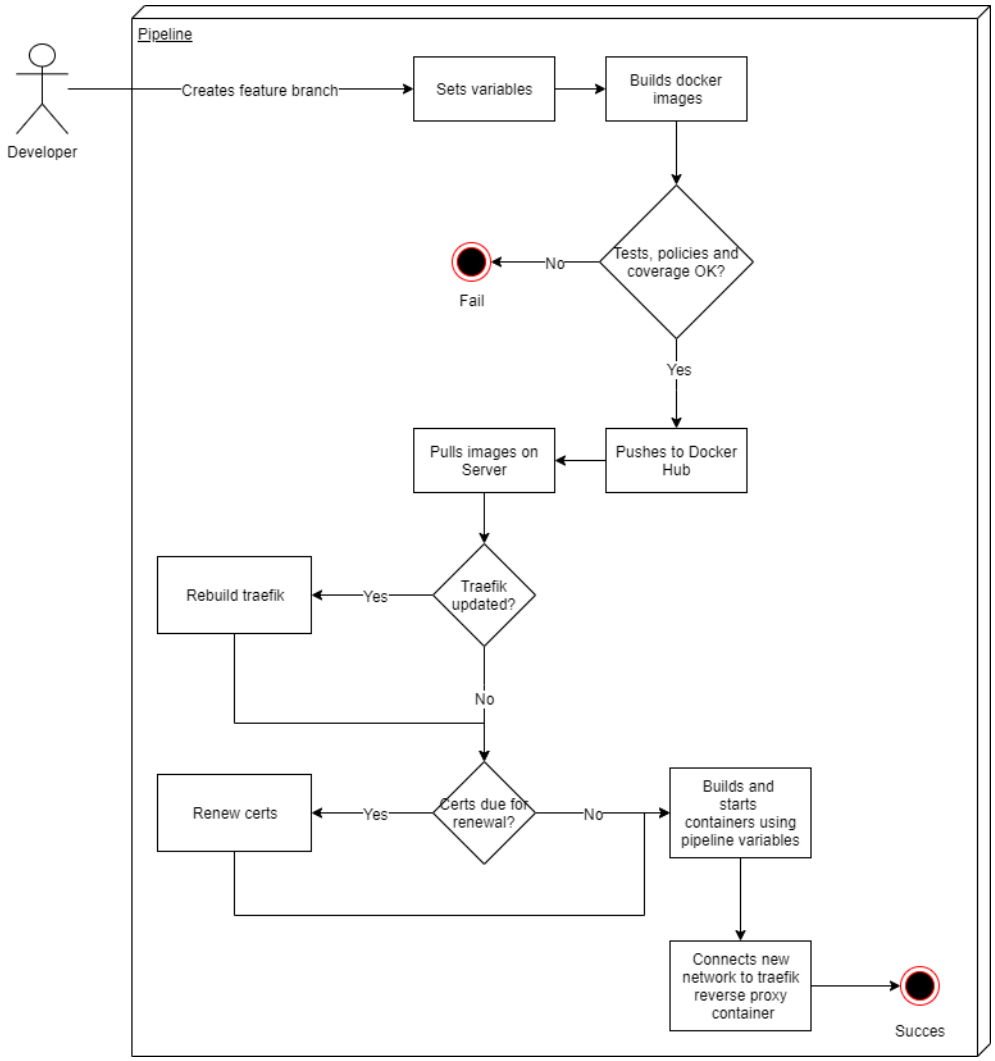
7.1 Feature-environments

Nieuwe infrastructuur:



FIGUUR 7.1: Nieuwe infrastructuur met Traefik reverse proxy

Activity Diagram:



FIGUUR 7.2: Activity Diagram voor de pipeline

7.2 Codecov

Code voor het implementeren is te zien in bijlage A.4

Hoofdstuk 8

Requirements

Deze paragraaf gaat over de deelvraag “Voldoen de verbeteringen aan de vereiste requirements?”

8.1 Conclusie

Boter, kaas en eieren.

Hoofdstuk 9

Aanbevelingen

9.1 Deployment targets

9.1.1 Cloud service providers

Momenteel worden de applicaties binnen Developers.nl gehost op een simpele, traditionele server van TransIP. Een overweging om te maken is of dit niet beter naar een cloud service provider kan worden verplaatst, aangezien dit mogelijk de onderhoudbaarheidslast verminderd. L. Wang *et al.* [52] definiëren cloud computing als "A computing Cloud is a set of network enabled services, providing scalable, QoS guaranteed, normally personalized, inexpensive computing infrastructures on demand, which could be accessed in a simple and pervasive way". Volgens [53] zijn er drie verschillende categorieën van Cloud computing:

- Infrastructure as a Service (IaaS): Een virtueel aangeboden infrastructuur van rekenkracht en/of geheugen [54].
- Platform as a Service (PaaS): Een aangeboden platform voor ondersteuning van deployment, ontwikkelen en testen van applicaties [55].
- Software as a Service (SaaS): Een aangeboden (web)applicatie dat direct gebruikt kan worden [54].

Om te overwegen of een cloud provider bij de wensen van Developers.nl past worden de voor-en nadelen op een rijtje gezet:

Voordelen cloud: De kosten van cloud hosting zijn flexibel, er wordt alleen betaalt voor wat er daadwerkelijk gebruikt wordt, zolang er maar verstandig gebruik van wordt gemaakt. Dit betekend dat het mogelijk erg kostenefficiënt kan zijn voor Developers.nl aangezien er tijdens de maandelijkse "Tech Nights" piekmomenten zijn op de website, en er heel weinig verkeer is op het EMS. Een ander voordeel is dat er bijna een ongelimiteerde hoeveelheid opslagruimte beschikbaar is. Aangezien het CMS van de website dubbel functioneert als "file-server" en er veel foto's worden geüpload is het fijn dat er geen rekening hoeft worden gehouden met de hoeveelheid opslag. Bovendien worden software updates automatisch uitgevoerd, software als K8s kunnen al inbegrepen zijn bij de infrastructuur en het maakt schalen gemakkelijker. Aangezien cloud providers meer middelen voor beveiliging hebben wordt de veiligheid ook een stuk verbeterd.

Voordelen traditioneel: Ook al is cloud hosting meer kostenefficiënt is het toch mogelijk dat een web host goedkoper uitkomt. Zolang er maar geen hoge piekmomenten zijn in het verkeer. Dit is dus niet van toepassing op Developers.nl aangezien de "Tech Nights" of andere evenementen voor piekmomenten zorgen.

Nadelen cloud: Cloud providers hebben de mogelijkheid voor technische problemen buiten de controle van klanten, waardoor het mogelijk is dat er downtime

ontstaat. Ook kan het duurder uitpakken zodra er niet goed word omgegaan met de schaalstrategie of benodigde rekenkracht.

Nadelen traditioneel: De mogelijkheid bestaat dat er meer kosten worden gemaakt dan nodig is. Ook is shared-hosting een risico omdat zodra een andere klant veel rekenkracht opeist de kans ontstaat dat de prestaties dalen.

Er zijn veel verschillende cloud providers, waarvan de grootste Amazon Web Services (AWS), Microsoft Azure en Google Cloud Platform (GCP) zijn. TODO: kiezen welke.. kosten nagaan? Veel werk?

9.2 Serverless computing

In verband met de lage hoeveelheid verkeer op het EMS is “serverless computing” een goede oplossing om kosten te besparen.

Hoofdstuk 10

Conclusie

Kaas

Literatuurlijst

[1] P. S. S. GmbH. (2019). Proxmox - powerful open-source server solutions, [Online]. Available: https://www.proxmox.com/.

[2] AVINetworks. (2019). Service discovery definition, [Online]. Available: https://avinetworks.com/glossary/service-discovery/.

[3] The Linux Foundation. (2019). Production-grade container orchestration, [Online]. Available: https://kubernetes.io.

[4] Docker Inc. (2019). Enterprise container platform for high-velocity innovation, [Online]. Available: https://docker.com.

[5] Red Hat inc. (2019). How ansible works, [Online]. Available: https://www.ansible.com/overview/how-ansible-works.

[6] Amber Ankerholz. (Apr. 2016). 8 container orchestration tools to know, [Online]. Available: https://www.linux.com/news/8-open-source-container-orchestration-tools-know/.

[7] A. Pérez, G. Moltó, M. Caballer, and A. Calatrava, "Serverless computing for container-based architectures", Feb. 2018. DOI: https://doi.org/10.1016/j.future.2018.01.022.

[8] A. M. Andreas Wittig, "Amazon web services in action", 2016. [Online]. Available: https://s3-ap-southeast-1.amazonaws.com/tv-prod/documents%2Fnull-Amazon+Web+Services+in+Action.pdf.

[9] Red hat inc. (2019). Ansible tower, [Online]. Available: https://www.ansible.com/products/tower.

[10] Chef. (2019). Chef, [Online]. Available: https://chef.io.

[11] Puppet. (2019). Unparalleled infrastructure automation and delivery, [Online]. Available: https://puppet.com.

[12] HashiCorp. (2019). Introduction to terraform, [Online]. Available: https://www.terraform.io/intro/index.html.

[13] T. G. Peter Mell, "The nist definition of cloud computing", p. 1, Oct. 2011. DOI: https://doi.org/10.6028/NIST.SP.800-145. [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-145/final.

[14] A. Wiggins, "The twelve-factor app", 2017. [Online]. Available: https://12factor.net.

[15] D. D. Gereld Weber, *Trends in Enterprise Application Architecture*. Feb. 2006, vol. 4437.

[16] P. Kruchten, "Architectural blueprints—the "4+1" view model of software architecture", pp. 42–50, Nov. 1995.

[17] D. A. Wheeler, "Why open source software/free software (oss/fs)? look at the numbers", Nov. 2004. [Online]. Available: http://www.robotcub.org/index.php/robotcub/content/download/290/1049/file/Why%20Open%20Source%20Software.pdf.

[18] K. de Munter, "Stageplan en oriëntatie developers.nl", 2017.

[19] Developers.nl, "Positioneringsprofiel developers.nl", 2018.

[20] K. de Munter, "Afstudeervoorstel", 2019.

[21] M. D. Hill, "What is scalability?", vol. 18, pp. 18–21, 4 Dec. 1990. [Online]. Available: https://dl.acm.org/citation.cfm?id=121975.

[22] T. W. Leticia Duboc David S. Rosenblum, "A framework for modelling and analysis of software systems scalability", May 2006. [Online]. Available: http://discovery.ucl.ac.uk/4990/1/4990.pdf.

[23] A. B. Bondi, "Characteristics of scalability and their impact on performance", Sep. 2000. [Online]. Available: https://www.win.tue.nl/~johanl/educ/2II45/2010/Lit/Scalability-bondi%202000.pdf.

[24] M. A.-E.-B. Hesham El-Rewini, *Advanced computer architecture and parallel processing*. 2005.

[25] J. B. G. Charles B. Weinstock, "On system scalability", 2006. [Online]. Available: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=7887.

[26] Magento. (2019). Extensibility and modularity, [Online]. Available: https://devdocs.magento.com/guides/v2.3/architecture/extensibility.html.

[27] A. L. Niklas Johansson, "Designing for extensibility: An action research study of maximizing extensibility by means of design principles", Jun. 2009. [Online]. Available: http://hdl.handle.net/2077/20561.

[28] A. A. T. Penny Grubb, *Software Maintenance: Concepts And Practice (Second Edition)*. River Edge, N.J.: World Scientific, 2003, vol. 2.

[29] IEEE, "Standard glossary of software engineering terminology", IEEE 610.12, 1990.

[30] J. K. C. Krishan K. Aggarwal Yogesh Singh, "An integrated measure of software maintainability", 2002. DOI: https://doi.org/10.1109/RAMS.2002.981648.

[31] ISO, "Software product quality", International Organization for Standardization, Geneva, CH, ISO 25010, 2011.

[32] O. T. Berna Seref, "Software code maintainability: A literature review", vol. 7, 3 May 2016. [Online]. Available: http://aircconline.com/ijsea/V7N3/7316ijsea05.pdf.

[33] R. Niedermayr, "Why we don't use the software maintainability index", Mar. 2016. [Online]. Available: https://www.cqse.eu/en/blog/maintainability-index.

[34] A. van Deursen, "Think twice before using the "maintainability index"", Aug. 2014. [Online]. Available: https://avandeursen.com/2014/08/29/think-twice-before-using-the-maintainability-index/.

[35] J. V. Ilja Heitlager Tobias Kuipers, "A practical model for measuring maintainability", 2007. [Online]. Available: https://www.softwareimprovementgroup.com/wp-content/uploads/2016/10/APracticalModelForMeasuringMaintainability.pdf.

[36] S. T. Albin, *The art of software architecture: design methods and techniques*. Wiley Publishing, Inc., Indianapolis, Indiana, Mar. 2003, vol. 9, ISBN: 9780471468295.

[37] P. C. Len Bass, *Software Architecture in Practice*. Pearson Education (US), Sep. 2012, ISBN: 9780321815736.

[38] E. W. Nick Rozanski, *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives*. Nov. 2011, ISBN: 9780132906074.

[39] R. K. Humberto Cervantes, *Designing Software Architectures: A Practical Approach*. Addison-Wesley Professional, 2016, ISBN: 9780134390789.

[40] IEEE, "Standard glossary of software engineering terminology", ISO/IEC/IEEE std. 42010, 2011.

[41] a. s. o. T. O. G. TOGAF® Standard Version 9.2, "Core concepts", 2018. [Online]. Available: `https://pubs.opengroup.org/architecture/togaf9-doc/arch/chap02.html`.

[42] H. van Vliet, *Software Engineering: Principles and Practice*. May 1993, vol. 3, p. 63, ISBN: 978-0470031469.

[43] B. Horowitz, "Mra, part 5: Adapting the twelve-factor app for microservices", *Nginx, Inc.*, Jul. 2016.

[44] K. Hoffman, *Beyond the Twelve-Factor App – Exploring the DNA of Highly Scalable, Resilient Cloud Applications*. O'Reilly Media, Inc., Apr. 2016, ISBN: 9781491944011.

[45] Developers.nl. (2019). Documentatie interne systemen developers.nl.

[46] C. McMahon, "The 12 factor php app", Nov. 2014. [Online]. Available: `http://slashnode.com/the-12-factor-php-app-part-2/`.

[47] V. Tardia, "The 12 factors of php", Oct. 2016. [Online]. Available: `https://vito.tardia.me/blog/the-12-factors-of-php`.

[48] B. Holt, "The twelve-factor app applied to php", Nov. 2011. [Online]. Available: `https://www.bradley-holt.com/2011/11/the-twelve-factor-app-applied-to-php/`.

[49] C. Lüdemann, "Feature environments in all environments – a guide to faster delivery", Nov. 2018. [Online]. Available: `https://christianlydemann.com/feature-branches-in-all-environments-a-guide-to-test-once-and-deploy/`.

[50] A. DadGar, "Why policy as code?", *HashiCorp, Inc.*, Jan. 2018. [Online]. Available: `https://www.hashicorp.com/blog/why-policy-as-code/`.

[51] A. Khan, "Key characteristics of a container orchestration platform to enable a modern application", vol. 4, pp. 42–48, 5 Dec. 2017. DOI: `10.1109/MCC.2017.4250933`.

[52] L. Wang, G. von Laszewski, M. Kunze, and J. Tao, "Cloud computing: A perspective study", pp. 1–11, 2010. DOI: `https://doi.org/10.1007/s00354-008-0081-5`.

[53] D. Chappell, "A short introduction to cloud platforms – an enterprise-oriented view", Aug. 2008.

[54] A. Apostu, F. C. Puican, G. Ularu, G. Suciu, and G. Todoran, "Study on advantages and disadvantages of cloud computing – the advantages of telemetry applications in the cloud", 2013.

[55] J. F. S. William Y. Chang Hosame Abu-Amara, *Transforming Enterprise Cloud Services*. 2010, ISBN: 9789048198450.

Bijlage A

Code

A.1 Docker-compose opstelling voor k6, InfluxDB & Grafana

Om de loadtest met k6, influxDB en grafana op te stellen heeft Loadimpact een docker-compose opstelling gemaakt. Na wat onderzoek is het opgevallen dat deze opstelling erg verouderd is. Daarom is ervoor gekozen om een eigen opstelling te maken:

```
version: '3.4'

networks:
  k6:
  grafana:

services:
  influxdb:
    image: influxdb:1.5.4
    networks:
    - k6
    - grafana
    ports:
      - "8086:8086"
    environment:
      - INFLUXDB_DB=k6

  grafana:
    image: grafana/grafana:6.4.1
    networks:
      - grafana
    ports:
      - "3000:3000"
    environment:
      - GF_AUTH_ANONYMOUS_ORG_ROLE=Admin
      - GF_AUTH_ANONYMOUS_ENABLED=true
      - GF_AUTH_BASIC_ENABLED=false
    volumes:
      - ./grafana/datasource.yml:/etc/grafana/provisioning/datasources/datasource.yml

  k6:
    image: loadimpact/k6:0.25.1
    networks:
```

```
      - k6
    ports:
      - "6565:6565"
    environment:
      - K6_OUT=influxdb=http://influxdb:8086/k6
    volumes:
      - ../k6:/k6
```

Hiervoor is een Pull-Request gemaakt naar loadimpact/k6 om dit te verbeteren. `https://github.com/loadimpact/k6/pull/1183` samen met de issue `https://github.com/loadimpact/k6/issues/1182` . Hierin is te lezen wat precies de veranderingen waren. De maintainers van k6 waren blij met de verandering en hebben deze geaccepteerd en gemerged naar master. De loadtest is geschreven in javascript met de volgende code:

```
import http from "k6/http";
import { sleep, check } from "k6";

export let options = {
  stages: [
    { duration: "10s", target: 20 },
    { duration: "10s", target: 40 },
    { duration: "10s", target: 60 },
  ]
};

export default function() {
  check(http.get("https://test.developers.nl/"), {
    "is status 200": (r) => r.status === 200
  });
  sleep(1);
};
```

A.2 Docker container exits

PostgreSQL

```
LOG: received smart shutdown request
LOG: background worker "logical replication launcher" (PID 43) exited
 ↪  with exit code 1
LOG: shutting down
LOG: database system is shut down
```

PHP-FPM

```
NOTICE: Terminating ...
NOTICE: exiting, bye-bye!
```

Redis

```
1:signal-handler (1570781278) Received SIGTERM scheduling shutdown...
# User requested shutdown...
* Saving the final RDB snapshot before exiting.
* DB saved on disk
* Removing the pid file.
# Redis is now ready to exit, bye bye...
```

Nginx

```
[notice] 1#1: signal 15 (SIGTERM) received from 56, exiting
[notice] 48#48: exiting
[notice] 47#47: exiting
[notice] 47#47: exit
[notice] 1#1: signal 14 (SIGALRM) received
[notice] 1#1: signal 17 (SIGCHLD) received from 48
[notice] 1#1: cache manager process 48 exited with code 0
[notice] 1#1: worker process 47 exited with code 0
[notice] 1#1: exit
```

A.3 Docker container kill & restarts

```
$ docker ps -q
d0829783af18
f72e9967771b
01dd48ff5a59
fab794731d47
ca510c065d11
3ee85578efb5

$ docker kill $(docker ps -q)

$ docker ps
d0829783af18
f72e9967771b
01dd48ff5a59
fab794731d47
ca510c065d11
3ee85578efb5

$ docker ps -q

$ docker start $(docker ps -aq)
d0829783af18
f72e9967771b
```

```
01dd48ff5a59
fab794731d47
d68d7ab9809c
ca510c065d11
3ee85578efb5
e1866ab6c1af

$ docker ps -q
d0829783af18
f72e9967771b
01dd48ff5a59
fab794731d47
ca510c065d11
3ee85578efb5
```

A.4 Codecov implementatie

Om codecov te implementeren in de website is het volgende gebeurd:
De README.md is bijgewerkt:

```
## Tests

We enforce that code coverage stays acceptable using codecov:

[![codecov](https://codecov.io/bb/developers_nl/developers.nl/branch/ma
ster/graph/badge.svg?token=DzAv79t9Gd)](https://codecov.io/bb/developer
s_nl/developers.nl)
```

Bitbucket en codecoverage environment variabelen moesten worden doorgegeven door build arguments. Het builden van de Docker images gebeurd met Ansible:

```
docker_images:
  - dockerfile: docker/php7-fpm/Dockerfile
  path: ../
  name: developersnl/website-php-fpm
  buildargs:
    GROUP_ID: 9000
    USER_ID: 9000
    BITBUCKET_BRANCH: "{{ lookup('env','BITBUCKET_BRANCH') }}"
    BITBUCKET_COMMIT: "{{ lookup('env', 'BITBUCKET_COMMIT') }}"
    BITBUCKET_BUILD_NUMBER: "{{ lookup('env','BITBUCKET_BUILD_NUMBER')
     ↪  }}"
    BITBUCKET_REPO_OWNER: "{{ lookup('env','BITBUCKET_REPO_OWNER') }}"
    BITBUCKET_REPO_SLUG: "{{ lookup('env','BITBUCKET_REPO_SLUG') }}"
    BITBUCKET_PR_ID: "{{ lookup('env','BITBUCKET_PR_ID') }}"
    CODECOV_TOKEN: "{{ lookup('env','CODECOV_TOKEN') }}"
    CI: "{{ lookup('env','CI') }}"
```

In de php7-fpm dockerfile zijn de build args omgezet naar environment variablen, een aantal apk packages toegevoegd en is het codecov script toegevoegd:

```
FROM application AS test

ENV SYMFONY_PHPUNIT_VERSION 8.0.0

ARG BITBUCKET_BRANCH
ARG BITBUCKET_BUILD_NUMBER
ARG BITBUCKET_REPO_OWNER
ARG BITBUCKET_REPO_SLUG
ARG BITBUCKET_PR_ID
ARG CODECOV_TOKEN
ARG CI
ARG BITBUCKET_COMMIT

ENV BITBUCKET_BRANCH=$BITBUCKET_BRANCH
ENV BITBUCKET_BUILD_NUMBER=$BITBUCKET_BUILD_NUMBER
ENV BITBUCKET_REPO_OWNER=$BITBUCKET_REPO_OWNER
ENV BITBUCKET_REPO_SLUG=$BITBUCKET_REPO_SLUG
ENV BITBUCKET_PR_ID=$BITBUCKET_PR_ID
ENV CODECOV_TOKEN=$CODECOV_TOKEN
ENV CI=$CI

# TODO: Cange VCS_COMMIT_ID to BITBUCKET_COMMIT when
 ↪  https://github.com/codecov/codecov-bash/pull/225 is deployed
ENV VCS_COMMIT_ID=$BITBUCKET_COMMIT

COPY --from=composer:1.9.0 /usr/bin/composer /usr/bin/composer

RUN apk add \
    php7-pdo_sqlite \
    php7-sqlite3 \
    php7-phar \
    php7-pear \
    php7-dev \
    redis \
    curl \
    bash \
    git \
    mercurial \
    findutils \
    g++ \
    make \
 && . /bin/pcov.sh \
 && redis-server --daemonize yes --requirepass test \
 && composer install -d /app/src --optimize-autoloader
 ↪  --no-interaction --no-suggest --no-scripts \
 && chmod u+x,g+x /app/src/bin/phpunit \
 && /app/src/bin/phpunit --configuration /app/src/phpunit.xml
 ↪  --coverage-clover=coverage.xml \
 && curl -s https://codecov.io/bash | bash -s - -X coveragepy
```

Er is een script geschreven om pcov te installeren zodat dit kan hergebruikt worden zowel in de 'develop.sh' entrypoint als in de test-stage van de dockerfile.

```
#!/bin/sh

# Add PHP Coverage ini configuration
echo "- Enabling pcov"
cat <<-EOF > /etc/php7/conf.d/pcov.ini
extension=pcov
pcov.enable=1
EOF

echo "- Installing pcov"
if ! pecl list | grep pcov >/dev/null 2>&1;
then
    pecl install pcov ||
    {
        echo "Could not pecl install pcov" >&2;
        exit 1;
    }
fi
```

Om BitBucket een betere ondersteuning te geven met codecov is hier ook een Pull-Request naar codecov-bash gemaakt. Deze is te zien op: `https://github.com/codecov/codecov-bash/pull/225` . De maintainers van codecov waren tevreden met deze verbeteringen en hebben de Pull-Request geaccepteerd en gemerged.

Bijlage B

Tabellen

B.1 Beyond the 12-factor app

In deze tabel worden de 15 factoren behandeld.

Factor	Naam	Gevolg	Waarom?
1	One codebase, one application	Onderhoudbaarheid (Modulariteit)	Een applicatie is een losstaand component waardoor wijzigingen minimale impact hebben op andere componenten.
2	API first	Structural scalability	Door de API op de eerste rang te zetten van het development proces wordt de mogelijkheid gecreëerd om met elkaars contracten te communiceren zonder interne ontwikkelingsprocessen te verstoren. Zo kunnen veel nieuwe services gemakkelijker worden toegevoegd.
3	Dependency management	Onderhoudbaarheid (modulariteit)	Gemakkelijk opzetten van project voor nieuwe ontwikkelaars.
4	Design, build, release, and run	Onderhoudbaarheid (wijzigbaarheid)	Door duidelijke stadia te definiëren worden wijzigingen aan de applicatie sneller in productie geplaatst.
5	Configuration, credentials, and code	Structural scalability	Environment variabelen zijn niet in omgevingen maar per deployment opgezet, zo maakt de hoeveelheid omgevingen niet uit.
6	Logs	Onderhoudbaarheid (analyseerbaarheid)	Door logs naar de `stdout` te sturen is het gemakkelijker om specifieke fouten te vinden, overzicht te creëren en actief meldingen te versturen naar ontwikkelaars.
7	Disposability	Load scalability	Door processen gemakkelijk te laten stoppen en starten gaat het schalen een stuk sneller.

8	Backing services	Onderhoudbaarheid (modulariteit)	Door backing services als "attached resources" te behandelen maakt het niet uit welke techniek er wordt gebruikt en zijn deze dus los gekoppeld.
9	Environment parity	Onderhoudbaarheid (wijzigbaarheid)	Er kan een stuk vaker gedeployed worden naar een specifieke omgeving, doordat alle omgevingen zo goed als gelijk aan elkaar zijn.
10	Administrative processes	Onderhoudbaarheid (analyseerbaarheid)	Door commands in versiebeheer op te slaan is er een duidelijk overzicht en een geschiedenis van alle "one-off processes" die gebeuren.
11	Port binding	Onderhoudbaarheid (wijzigbaarheid)	Door HTTP als een service te beschouwen ontstaat er meer controle over lagere levels van de infrastructuur (HTTP & TCP).
12	Stateless processes	Load scalability	Mede door de shared-nothing architectuur kan het systeem gemakkelijker schalen.
13	Concurrency	Load scalability	Door het horizontaal of verticaal schalen kan de applicatie een groeiende hoeveelheid verkeer beter aan.
14	Telemetry	Onderhoudbaarheid (Analyseerbaarheid)	Door gegevens van de applicatie in productie goed te kunnen monitoren is op te maken hoe de applicatie zich gedraagt. Zodra er iets fout is kan er meteen op worden gereageerd.
15	Authentication and authorization	Security	Een cloud-native applicatie moet veilig zijn, aangezien de code over meerdere data centers wordt getransporteerd en door veel verschillende cliënten wordt benaderd.

Bijlage C

Gesprekken

C.1 Requirements

Dit zijn de gemaakte aantekeningen tijdens discussies over de requirements met Jelle:

- schaalbaarheid: meerdere omgevingen (feature branches)
- Merge train -> automatische merges en deploys
- Pulumi
- Generieke boilerplate voor een CI/CD Pipeline
- segregation of duties
- docker swarm voor performance curves te laten zien
- Advies over deployment targets
- testbaarheid -> static code analysis -> integratie tests ->
- Hoeveelheid coverage -> pipeline
- Probleemstelling & requirements
- Functional scalability
- Wat te monitoren?
- Product owner validatie & automatisch testen apart
- Validaties zo veel mogelijk automatisch (policies, segregation of duties)
- Kwaliteit waarborgen -> concreter
- Functional scalability -> Extensibility
- Extensibility functioneel? niet functioneel? Vragen stellen feedback online!
- Monitoring: Metrics als ruimte -> cpu -> memory uiteindelijk verkeer, etc. promethius
- terraform (firewalls, netwerken) voor alles tot aan de VM en ansible om de vm af te configureren
- TransIP Terraform API
- Nexpertise Terraform

C.2 Feature environments

Slack conversatie met Jelle:

```
Hey ik heb denk ik iets gevonden wat mij wel een leuke oplossing
lijkt:
https://github.com/jwilder/nginx-proxy
Het luistert naar je docker run commands om daaruit environment
variabelen te halen; waaronder VIRTUAL_HOST , waardoor het dus iets
als VIRTUAL_HOST=${BITBUCKET_BRANCH}.${HOSTNAME} kan worden. Wat vind
jij hiervan? Een mogelijke oplossing? Of tenminste een deel
hiervan.... zodat er niet een volledige abstractielaag op zit
```

Reactie van Jelle:

```
Zou idd een oplossing zijn, kijk anders ook ffe naar traefik.io
```

Conversatie over het beveiligen van de Docker Socket:

```
Kaj:
Oke dit lijkt mij wel een probleem:
https://github.com/containous/traefik/issues/4174
Zowel de nginx-proxy als traefik hebben dit.. is dit:
https://github.com/Tecnativa/docker-socket-proxy écht de beste
oplossing hiervoor, of heb jij toevallig nog een geniale ingeving?

Jelle:
Zou ik me even in moeten verdiepen, met de oplossing die ik met nginx
aan het rommelen was gebruikte ik docker labels en docker inspect

Kaj:
Hmm oke oke
Misschien is dat ook wel een goeie, want nginx moet er toch inblijven
aangezien traefik geen fastCGI support heeft

Jelle:
Ja maar das dan achter traefik als load balancer zegmaar

Kaj:
Oh.. dus dan heb je traefik statisch geconfigureerd?

Jelle:
Bekijk de docker-compose file:
https://medium.com/@luiscoutinh/reverse-proxy-com-docker-traefik-nginx-php-mysql-mosqu

Kaj:
Ja precies, dat is ongeveer hoe het wordt aangeraden. Maar ook die
oplossing exposed de docker socket
```

```
Jelle:
ja idd dat is een probleem, iig voor productie
Expose the Docker socket over TCP, instead of the default Unix socket
file

Kaj:
Dat is eigenlijk het enige waar ik tegenaan loop, want een oplossing
met traefik lijkt mij bijna precies wat ik zoek

Jelle:
dat kan wel, via certificaten beveiligen:
https://docs.traefik.io/providers/docker/#docker-api-access
```

C.3 (niet-)functionele schaalbaarheid

Na het promoten van de geschreven blog[1] is dit de meest populaire blog van Developers.nl in 2019 geworden. Dit heeft het volgende feedbackpuntje opgeleverd: "Scalability is a two way thing, so adding and removing should be in the definition (and thought lines)". Waar ik het volledig mee eens ben, en zal verbeteren in de toekomst. Ook is er een leuke discussie uit gekomen. Marlon Etheredge, MSc vroeg:

```
Hi Kaj,

Mijn vraag behoeft enige introductie.

Ik ben werkzaam in een deelgebied van de
informatica/software-engineering waarbij performantie zeer belangrijk
is, computer graphics. Onze implementaties dienen zo snel als mogelijk
antwoorden te geven op soms complexe berekeningen, doorgaans in
(minder dan) millisecondes.

Schaalbaarheid in mijn context staat dan ook voor twee dingen; ten
eerste gaat het om het niet schenden van tijdsgrenzen waar wij mee te
maken hebben (e.g. één frame dient in 1/50 seconde klaar te zijn)
onafhankelijk van de hoeveelheid data die verwerkt moet worden. Ten
tweede staat schaalbaarheid voor implementaties die rekbaar zijn op
basis van veranderende eisen die aan een systeem worden gesteld.

In deze context gaat het dan niet zo zeer om een veranderend systeem,
waarbij bijvoorbeeld functionaliteit wordt toegevoegd
("... must be modified as soon ..." in je eerste definitie), of het
systeem verbeterd wordt ("... is able to be improved ..." in je tweede
definitie), maar eerder om een kwaliteitskenmerk van een systeem in
ogenschouw nemend welke eisen mogelijk in de toekomst aan dit systeem
gesteld zullen worden en de hoeveelheid energie die het zal kosten om
het systeem te laten aansluiten op deze eisen. In die zin denk ik
overigens ook dat dit een interessant onderwerp is, aangezien het
ontwerpen van dergelijke systemen fundamenteel is aan de informatica.
```

[1] https://developers.nl/blog/69/Defining-software-scalability-using-requirements

```
Mijn concrete vraag aan jou is als volgt; je schrijft:

"Instead of adding more of the same requirement, non-functional
requirements like security or usability are always able to be
improved. Therefore, scaling a non-functional requirement is the same
as improving it. Setting clear requirements helps proving your
solution is scalable non-functionally."

Is verandering (bijvoorbeeld in de vorm van verbetering) noodzakelijk
voor schaalbaarheid, of is het mogelijk schaalbaarheid te toetsen los
van verandering?
```

Mijn antwoord:

```
Schaalbaarheid in jouw context sluit goed aan op mijn twee definities:
Je noemt "het niet schenden van tijdsgrenzen ... onafhankelijk van de
hoeveelheid data"; in dit geval zijn de berekeningen een functionele
requirement, en deze moeten voldoende blijven functioneren naarmate
het hoeveelheid gebruik toe neemt. In deze context gaat het
functioneren dus over het niet schenden van tijdsgrenzen, en de
hoeveelheid gebruik over de hoeveelheid data.

De tweede definitie die je noemt (schaalbaarheid voor implementaties
die rekbaar zijn op basis van veranderende eisen) omvat in dit geval
zowel functionele als niet functionele schaalbaarheid. In mijn
definities heb ik het vooral over opschalen, dit is nog een
verbeterpunt. Het concept van "veranderende eisen" vind ik een mooie.

Om antwoord te geven op je vraag: Schaalbaarheid is een
kwaliteitskenmerk, het daadwerkelijk schalen is een uitoefening van
dit kenmerk. Dus, ja, het is mogelijk om schaalbaarheid te toetsen los
van daadwerkelijke verandering. Een kwaliteitsanalyse op kenmerken als
complexiteit van algoritmes bijvoorbeeld. In de context van
functionele schaalbaarheid is dit "to what extent it continues to
function properly as the amount of use of the system increases", en
van niet-functionele schaalbaarheid "to what extent the quality of
that requirement remains acceptable as the use of the system
increases".

Ik hoop dat ik hiermee je vraag voldoende heb beantwoord, zo niet hoor
ik het graag uiteraard.
```

Marlon was tevreden met dit antwoord:

```
Duidelijk, dank voor je antwoord, erg interessant.
```