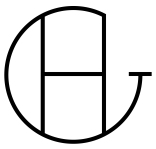


HoGent

Faculteit Bedrijf en Organisatie

Chain logging in Go, onderzoek naar een gepaste framework en databank

Benjamin Van Iseghem

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Chantal Teerlinck
Co-promotor:
Jens De Valck

Instelling: Be-Mobile

Academiejaar: 2018-2019

Tweede examenperiode

Faculteit Bedrijf en Organisatie

Chain logging in Go, onderzoek naar een gepaste framework en databank

Benjamin Van Iseghem

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Chantal Teerlinck
Co-promotor:
Jens De Valck

Instelling: Be-Mobile

Academiejaar: 2018-2019

Tweede examenperiode

Woord vooraf

Samenvatting

Dit onderzoek bespreekt de 4 belangrijkste open source logging oplossingen welke gebruikt kunnen worden bij het ontwikkelen van applicaties gebaseerd op een microservice architectuur. Dit onderzoek neemt de omgeving van Be-Mobile als voorbeeld en referentiepunt. In deze omgeving is sprake van een Kubernetes cluster met 150+ nodes. Na de recente release van Loki door Grafana moet een vergelijkende studie tussen Loki en de al wat meer gevestigde logging oplossingen gevoerd worden.

Het is de bedoeling van dit onderzoek om bedrijven die willen overschakelen naar een microservice omgeving te helpen met de zoektocht naar een geschikte logging oplossing. Ook bedrijven die reeds gebruik maken van een microservice omgeving hebben baat bij het lezen van dit onderzoek. Het kan meer duidelijkheid scheppen over de verschillen tussen elke oplossing. Aangezien drie van de vier onderzochte oplossingen gebruik maken van Elasticsearch kan afgevraagd worden wat het verschil hiertussen deze is.

Het onderzoek focust zich vooral op enkele kernpunten die elke goede logging oplossing moet bevatten. Ook wordt voor elke oplossing getoond hoe de installatie van elke oplossing wordt uitgevoerd. Verder wordt de theorie uitgelegd om deze ook op Kubernetes te installeren en te configureren. Uit dit onderzoek komt naar voren dat er geen duidelijke meest geschikte open source logging oplossing is. Loki kan als meest geschikte aanschouwd worden maar deze oplossing is nog steeds in ontwikkeling en zal pas binnen enkele maanden in productie gebruikt kunnen worden. Verder maken alle andere oplossingen gebruik van Elasticsearch wat voor redelijk wat geheugenverbruik zorgt. De meeste geschikte oplossing hierbij is de EFK stack. De conclusies die uit dit onderzoek getrokken worden leiden tot de vraag naar dieper onderzoek met een grotere testomgeving. Verder zou een extra onderzoek naar Loki binnen enkele maanden aan de orde zijn.

Inhoudsopgave

1 Inleiding 17

1.1 Probleemstelling 17

1.2 Onderzoeksvraag 18

1.3 Onderzoeksdoelstelling 18

1.4 Opzet van deze bachelorproef 18

2 Stand van zaken 21

2.1 Be-Mobile 21

2.2 Golang 22

2.3 Microservices 22

2.4 Kubernetes 23

2.4.1 Wat is Kubernetes? 23

2.4.2 Architectuur 23

2.5 Docker **25**

2.5.1 Situatie voor Docker 25

2.5.2 De oplossing 25

2.6 Logs, traces, en metrics **26**

2.6.1 Wat is een log? 26

2.6.2 Belang van logging 26

2.6.3 Golang logging frameworks 27

2.6.4 Logrus 27

2.6.5 Tracing 28

2.6.6 Metrics 28

2.7 Logging solutions **29**

2.7.1 Wat is een logging solution en hoe werkt het? 29

2.7.2 Belangrijke punten in een logging solution 30

2.7.3 ELK 30

2.7.4 EFK 31

2.7.5 Graylog 32

2.7.6 Grafana Loki 33

3 Methodologie **35**

3.1 Requirementsanalyse **38**

4 ELK **41**

4.1 Installatie en configuratie **41**

4.1.1 Lokale omgeving 41

4.1.2 Kubernetes omgeving 43

4.2 Requirements 43

4.2.1 Must have . . . 43

4.2.2 Should have . . . 44

4.2.3 Could have . . . 45

4.3 Resultaten 45

5 EFK . . . 47

5.1 Installatie en configuratie 47

5.1.1 Lokale omgeving . . . 47

5.1.2 Kubernetes omgeving . . . 49

5.2 Requirements 50

5.2.1 Must have . . . 50

5.2.2 Should have . . . 51

5.2.3 Could have . . . 52

5.3 Resultaten 52

6 Graylog . . . 55

6.1 Installatie en configuratie 55

6.1.1 Lokale omgeving . . . 55

6.1.2 Kubernetes omgeving . . . 57

6.2 Requirements 57

6.2.1 Must have . . . 57

6.2.2 Should have . . . 59

6.2.3 Could have . . . 61

6.3 Resultaten 61

7 Loki 63

7.1 Installatie en configuratie 63

7.1.1 Lokale omgeving 63

7.1.2 Kubernetes omgeving 66

7.2 Requirements 67

7.2.1 Must have 67

7.2.2 Should have 68

7.2.3 Could have 69

7.3 Resultaten 69

8 Conclusie 71

A Onderzoeksvoorstel 73

A.1 Introductie 73

A.2 State-of-the-art 74

A.3 Methodologie 74

A.4 Verwachte resultaten 74

A.5 Verwachte conclusies 74

Bibliografie 75

Lijst van figuren

2.1 Vergelijking tussen een monolith en microservices applicatie 22
2.2 Kubernetes DaemonSet voorbeeld 24
2.3 Kubernetes Architecture 25
2.4 Vergelijking tussen VM's en Docker containers 26
2.5 Voorbeeld van tracing in Jaeger 28
2.6 Voorbeeld van metrics in Prometheus 29
2.7 Het logging proces 29
2.8 Volledige ELK stack 31
2.9 EFK stack 31
2.10 Grafana Loki distributor 33
2.11 Grafana Loki Ingester 34

4.1 ELK stack frontend 42

5.1 EFK stack frontend 49

6.1 Graylog stack frontend 58
6.2 Graylog log detailscherm 59
6.3 Graylog input sherm 60

7.1 Loki query 67

Lijst van tabellen

4.1 ELK resultaten 46

5.1 EFK resultaten 53

6.1 Graylog resultaten 62

7.1 Loki resultaten 70

Listings

>>>>>>>>>>>>>>>>>>>

1. Inleiding

Alle logs op een centrale plaats hebben is een belangrijk aspect bij elke applicatie. Bij monolithische architecturen is dit vanzelfsprekend, maar dankzij de opkomst van de microservice architectuur is dit niet langer mogelijk zonder hulp van tools. Dit onderzoek bestudeert het verschil tussen een monolithische en een microservice architectuur. Daarna wordt op zoek gegaan naar de meest gepaste logging oplossing voor een microservice architectuur in een Kubernetes cluster van 150+ nodes. In dit onderzoek komen enkel de open-source oplossingen aan bod zodat deze volledig zelf kunnen gehost worden en er dus geen extra kosten aan te pas komen voor cloud services.

1.1 Probleemstelling

Go wordt steeds meer gebruikt en wordt door sommigen geprezen als een ideale taal voor microservices. Door de gedecentraliseerde structuur van microservices wordt het moeilijk om het overzicht te bewaren bij het debuggen van applicaties. Elke service heeft een eigen console met logs. Steeds opnieuw wisselen van console en scrollen naar een specifieke log is zeer inefficiënt. Waar naar op zoek moet gegaan worden is een gecentraliseerde logging oplossing die overzicht biedt aan al uw applicaties. De doelgroep voor dit onderzoek is bedrijven die in het proces zijn van overstappen van een monolithische naar een microservice architectuur of reeds overgestapt zijn en nog op zoek zijn naar een oplossing voor gecentralizeerde logging. In dit werk zullen meerdere oplossingen aangeboden worden en zal er aan de hand van een requirement lijst vergeleken welke het meest geschikt is om in productie uit te rollen.

1.2 Onderzoeksvraag

- Wat is de beste open-source logging met tracing oplossing voor een bedrijf als Be-Mobile?

1.3 Onderzoeksdoelstelling

Het is de bedoeling van dit onderzoek om een vergelijking te maken tussen de ondersteunde logging platformen in Go. Er zal een afweging gemaakt worden voor de impact die de oplossing heeft op het systeem, de installatie, het gebruiksgemak, en in welke mate er support is van de community. Een algemene oplossing zal niet geboden worden in dit onderzoek. Wel zal er een opsomming gebeuren van de mogelijkheden en zal aan elke oplossing een score gegeven worden aan de hand van enkele requirements.

- Installatie
- Gebruiksgemak
- Impact op het systeem
- Support door community

1.4 Opzet van deze bachelorproef

De rest van deze bachelorproef is als volgt opgebouwd:

In Hoofdstuk 2 wordt een overzicht gegeven van de stand van zaken binnen het onderzoeksdomein, op basis van een literatuurstudie.

In Hoofdstuk 3 wordt de methodologie toegelicht.

In Hoofdstuk 4 wordt de installatie en configuratie van de ELK of Elastic stack toegelicht. Verder worden alle requirements voor een logging solution vergeleken met de mogelijkheden van de ELK of Elastic stack en een score gegeven voor elke requirement.

In Hoofdstuk 5 wordt de installatie en configuratie van de EFK stack toegelicht. Verder worden alle requirements voor een logging solution vergeleken met de mogelijkheden van de EFK stack en een score gegeven voor elke requirement.

In Hoofdstuk 6 wordt de installatie en configuratie van Graylog toegelicht. Verder worden alle requirements voor een logging solution vergeleken met de mogelijkheden van Graylog en een score gegeven voor elke requirement.

In Hoofdstuk 7 wordt de installatie en configuratie van Loki en Promtail toegelicht. Verder worden alle requirements voor een logging solution vergeleken met de mogelijkheden van Loki en een score gegeven voor elke requirement.

In Hoofdstuk 8, tenslotte, wordt de conclusie gegeven en een antwoord geformuleerd op de onderzoeksvragen. Daarbij wordt ook een aanzet gegeven voor toekomstig onderzoek binnen dit domein.

2. Stand van zaken

2.1 Be-Mobile

Er zijn een groot aantal microservices door Be-Mobile ontwikkeld in Go om verkeersevents op te halen, te verwerken, en door te sturen. Een file met events wordt binnengehaald via een repository en naar JSON omgezet om naar Kafka te sturen. Kafka is een message broker. Dit is in essentie een message queue waarin volgorde kan gegarandeerd worden. Kafka zelf bestaat uit producer- en consumertopics, waar data respectievelijk naar geschreven en uit opgehaald wordt. Daarna worden de messages opgehaald uit kafka en wordt elk individueel event verrijkt met locatie, omschrijving, en andere zaken. Elke stap in de chain haalt de events op van Kafka, communiceert met API's om deze te verrijken en stuurt ze daarna terug naar een andere Kafka topic.

Om deze microservice architectuur te runnen wordt gebruik gemaakt van Kubernetes en Docker. Deze twee technologiën zijn enorm belangrijk in het hele verhaal en worden nog verder uitgelegd in 2.4 Kubernetes, en 2.5 Docker. In het kort is Kubernetes een cluster van nodes die gebruik maken van gecontaineriseerde events. Deze containers worden mogelijk gemaakt dankzij Docker. Elk event wordt niet door dezelfde chain van services gestuurd, er zijn meerdere mogelijke chains die doorlopen kunnen worden. De Kubernetes cluster bevat alle mogelijke chains waar elke service zijn eigen console heeft. Op het moment van het schrijven van dit werk is de enige logging die aanwezig is bij Be-Mobile de verzameling van console logs van Kubernetes. Om te debuggen moet er voortdurend gewisseld worden van console om het probleem te localiseren. Er is nog geen mogelijkheid tot filteren, gericht zoeken, of log metrics (De Valck, 2019).

2.2 Golang

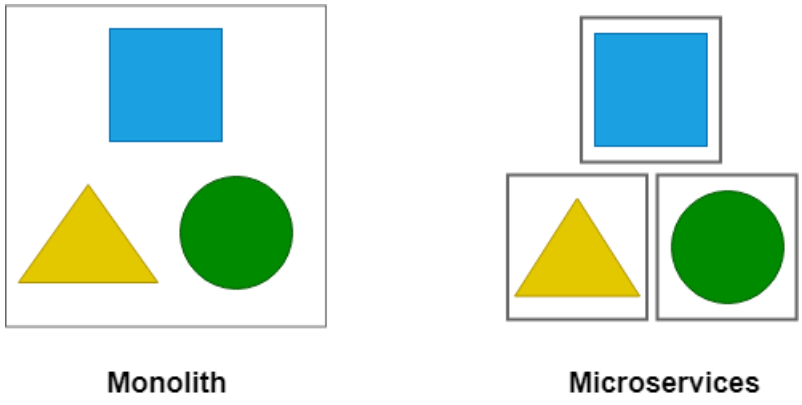
De taal die steeds meer gebruikt wordt in microservice architecturen is Go (Sabic, 2018). De redenen hiervoor worden mooi beschreven door IT Marketplace. Deze stelt dat microservices het best geschreven worden in Go omwille van:

1. “Go is snel en relatief snel te leren” (IT Marketplace, 2016);
2. “Een Go programma in de vorm van een enkel binair bestand is gemakkelijk te deployen” (IT Marketplace, 2016);
3. “Een Go programma is gemakkelijk te omwikkelen in een container” (IT Marketplace, 2016);

Golang is een expressieve, beknopte, en efficiënte taal met een nadruk op concurrency en modulariteit (Golang, 2012).

2.3 Microservices

Een microservice architectuur bestaat uit vele kleine services die elk hun eigen verantwoordelijkheid hebben, maar samen dezelfde functionaliteit bevatten als een monolithische architectuur (zie figuur 2.1). Het grote verschil met deze tweede is dus de structuur waarop deze is gebaseerd. Waar een monolithische architectuur bestaat uit één groot stuk software vol samenhangende componenten, bevat een microservice architectuur tientallen of zelfs honderden kleine services die met elkaar verbonden zijn in een chain. Events worden van de ene service naar de andere doorgestuurd, vaak door middel van een HTTP requests en message brokers (Casey, 2017).



Figuur 2.1: Vergelijking tussen een monolith en microservices applicatie Fuse, 2018

Een microservice architectuur wordt steeds meer gebruikt in organisaties bij het ontwikkelen van hun applicaties. Dit vooral omwille van voordelen zoals simpliciteit en flexibiliteit. Een ander voordeel aan het implementeren van microservices is het gemak waarmee fouten kunnen opgespoord en behandeld worden. Wanneer een service fouten produceert, zal deze sneller te vinden zijn. Aangezien een enkele service vaak slechts een klein stuk software betreft, zal het probleem snel opgelost kunnen worden. Er is dus sprake van een verhoogd

overzicht in de structuur van een applicatie. De schaalbaarheid en het onderhoudsgemak zijn nog twee belangrijke zaken om te vermelden. Er is ook een negatief aspect aan microservices verbonden, namelijk een enorm planning aspect bij de implementatie van deze structuur. Indien op een foute manier toegepast, kan het leiden tot een gedistribueerde monolithische structuur, wat het slechtste van de twee werelden combineert (Carey, 2018).

Logging is een enorm belangrijk aspect van een microservice architectuur. Om een applicatie met microservices te kunnen debuggen moet er een gecentraliseerd logging platform aanwezig zijn. Hoe groter de applicatie, hoe moeilijker om het overzicht van al deze services te bewaren. Elke service heeft een aparte console waarnaartoe gelogd wordt. Dit maakt het moeilijk om alles te doorzoeken in geval van bugs (Fuse, 2018).

Be-Mobile maakt gebruik van een microservice architectuur om hun producten te ontwikkelen. Elk verkeersevent dat binnenkomt, wordt van de ene microservice naar de andere doorgegeven en steeds verder verwerkt. Zo wordt een chain gecreëerd. Er zijn meerdere chains die doorlopen kunnen worden. Dit hangt af van het soort event dat heeft plaatsgevonden. Een belangrijk punt bij het implementeren van een gecentraliseerd logging platform voor Be-Mobile is dus het visualiseren van de chain die gevolgd werd door een bepaald event door middel van tracing (De Valck, 2019).

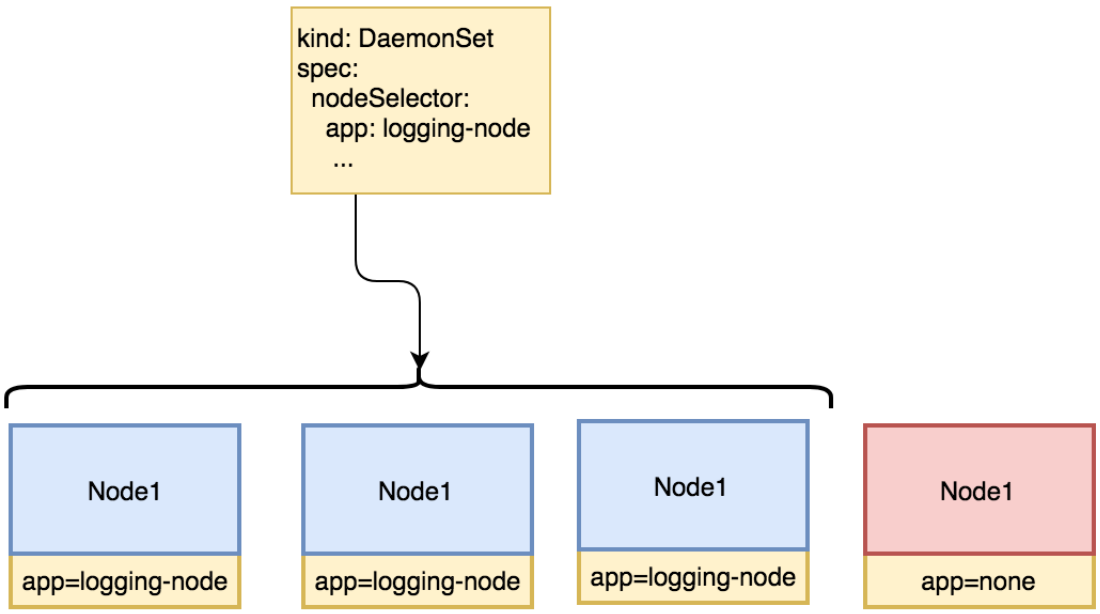
2.4 Kubernetes

2.4.1 Wat is Kubernetes?

De basis van Kubernetes is dat het een open-source systeem is voor het beheren van applicaties die in een container (zie 2.5 Docker) geplaatsd zijn. Containers zijn omhulsels voor services. Alles wat nodig is om de service te kunnen runnen, wordt voorzien door de container. Kubernetes draait als een samenhangende cluster (zie figuur 2.2) zodat men componenten en services doorheen verschillende infrastructuren kan beheren. De manier waarop applicaties gehost worden hangt af van de manier waarop Kubernetes geconfigureerd is. Het is mogelijk om een applicatie te runnen als een DaemonSet, waardoor er in elke node een instantie van de applicatie gerund wordt. Ook zorgt deze configuratie er voor dat er automatisch een nieuwe instantie van de applicatie gecreëerd wordt bij het aanmaken van een nieuwe node. Dit is slechts een voorbeeld van de mogelijkheden van Kubernetes. De optimale configuraties van de onderzochte logging solutions komen later aan bod in de bijlagen van dit werk (Ellingwood, 2018).

2.4.2 Architectuur

Kubernetes bestaat uit een master-minion architectuur (zie figuur 2.3) waar een master server het brein van de cluster voorstelt. Deze is een gateway om verschillende API's bloot te stellen aan de gebruikers. Verder is de master ook verantwoordelijk voor het controleren van de gezondheid van de cluster, scheduling van alle taken, en de communicatie tussen alle componenten (Ellingwood, 2018).



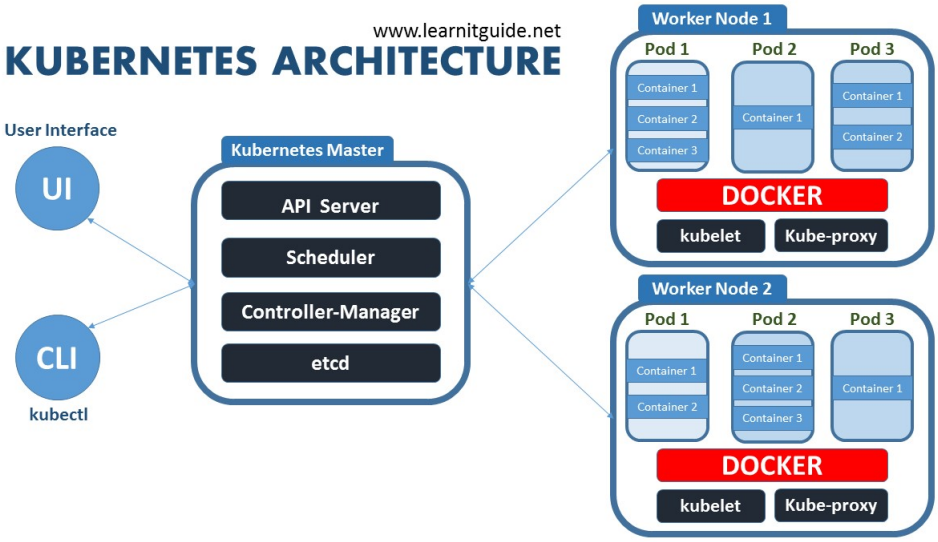
Figuur 2.2: Kubernetes Daemonset voorbeeld Stories, 2017

De ‘minions’ worden nodes genoemd en zijn de servers waarop een of meerdere containers gerund worden. De nodes krijgen instucties van de master over wat er moet gebeuren met verschillende containers. Het creëren van nieuwe containers, het vernietigen van oude containers, en het forwarden van data zijn enkele van de taken van de nodes (Ellingwood, 2018).

Een node bestaat uit een of meerdere pods (zie figuur 2.3). Elke pod is een groep van een of meerdere containers die gedeployed zijn. Deze pods zijn er om nog verder onderscheid te kunnen maken in de configuratie. Elke pod heeft een eigen IP adres binnen de cluster en elke container in een pod deelt een IP adres, hostnaam, en andere zaken (LearnItGuide, 2018).

Om een cluster op te starten wordt gebruik gemaakt van configuratie files in de vorm van YAML files. YAML staat voor ‘YAML Ain’t Markup Language‘ en wordt gebruikt om configuratie mee te maken (Grav, g.d.)) Hierin wordt de gewenste structuur beschreven. De master server leest deze configuraties en delegeert de opdrachten aan zijn nodes om de gewenste structuur te bereiken (Ellingwood, 2018).

Er is ook mogelijkheid tot lokaal testen. Hiervoor wordt Minikube gebruikt. Minikube maakt een lokale cluster aan met één node waarin een aantal pods aangemaakt kunnen worden. Op deze manier is het mogelijk configuraties te testen voordat deze gebruikt worden in productie (Ellingwood, 2018).



Figuur 2.3: Kubernetes Architecture LearnItGuide, 2018

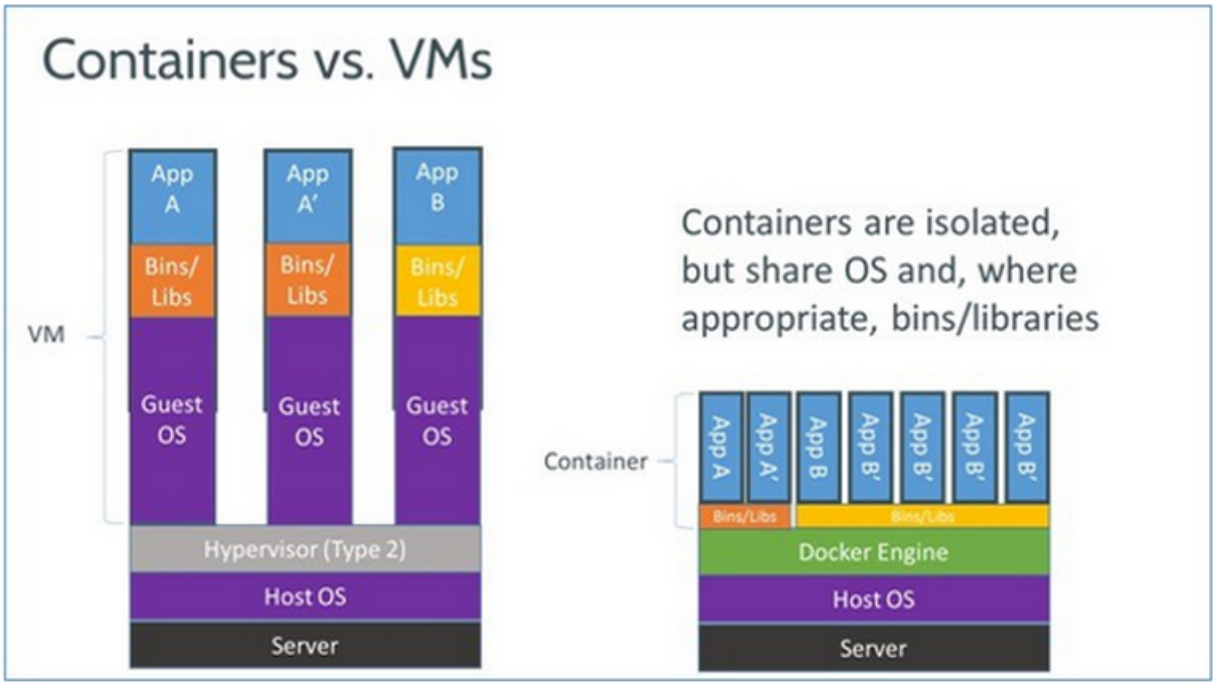
2.5 Docker

2.5.1 Situatie voor Docker

Voor de komst van Docker was de manier waarop software uitgevoerd werd volledig anders. Software laat het niet steeds toe om uitgevoerd te worden in dezelfde omgeving als een ander soort software. Maar hoe slaagde men er in om toch alle programma's werkende te krijgen? Ze maakten gebruik van virtual machines. Deze lieten toe om verscheidene programma's simultaan uit te voeren op dezelfde hardware. Hoewel virtuele machines een oplossing boden voor het probleem, was de kost hoog. Virtuele machines zijn zware software die elk enkele gigabytes in grootte zijn.Verder bracht deze oplossing andere problemen met zich mee zoals software updates, continuous delivery, en continuous integration (Yegulalp, 2018).

2.5.2 De oplossing

Docker containers zijn de oplossing voor dit probleem. Ze zijn licht, draagbaar, en flexibel. De manier waarop ze werken is simpel (zie figuur 2.4). Elk programma of stuk software wordt omhuld in een container die afzonderlijk draaiende wordt gehouden zoals een virtuele machine. Kortom wil dit zeggen dat alles geïsoleerd wordt en er dus geen problemen ontstaan met componenten die niet samen kunnen uitgevoerd worden. Zoals reeds besproken maakt Kubernetes uitstekend gebruik van containers. Het brengt deze met elkaar in contact of houdt ze net gescheiden. Zo kunnen containers die afhangen van elkaar, aan elkaar vastgehaakt worden in een pod. Een andere functionaliteit van Docker container is de herbruikbaarheid. Eens een container aangemaakt is kan deze via Kubernetes meerdere malen herbruikt worden of simultaan uitgevoerd worden (Yegulalp, 2018).



Figuur 2.4: Vergelijking tussen VM's en Docker containers Vaughan-Nichols, 2018

2.6 Logs, traces, en metrics

2.6.1 Wat is een log?

Een log is de documentatie van events die zich voordoen. De meeste systemen en software-applicaties bevatten automatische log generatie. De logs worden dan opgeslagen in een file waar ze bijgehouden worden om later geëvalueerd te worden. Men kan ook opteren om bij het schrijven van een eigen applicatie zelf logging te voorzien om de eigen applicatie beter te kunnen opvolgen en te debuggen. De keuze ligt bij de developer om deze logs dan op te slaan in een file. Standaard worden logs getoond in de console met een timestamp, bericht, en een log niveau (Technopedia, g.d.).

2.6.2 Belang van logging

Logging is een belangrijk aspect van software development. Maar wat is nu het belang van logging? De tijd die een developper investeert in het correct loggen van de belangrijke informatie in een applicatie is hopelijk toch geen verspilde tijd?

Ten eerste is er het belang van duidelijkheid in een applicatie. Logs zorgen er voor dat er precies kan gezien worden wat er gaande is in een applicatie. Bij het debuggen zijn logs van onschatbare waarde om na te gaan wat er fout is gegaan (LogDNA, g.d.).

Ten tweede hebben logs belang voor het hele bedrijf. Wanneer er sprake is van een bedrijf met verschillende teams die elk verantwoordelijk zijn voor hun eigen deel van een

applicatie, zullen er restricties te vinden zijn in de rechten van elk team. Een developper hoort geen rechten te hebben op repositories waar hij zelf niet mee in contact komt. Bij het debuggen van een probleem kan een developper de logs van software waar hij zelf geen rechten op heeft toch bekijken. Zonder logs zouden er rechten toegekend moeten worden om de code te bekijken op zoek naar het probleem (Czanik, 2013).

2.6.3 Golang logging frameworks

Bij Be-Mobile wordt Golang gebruikt om de backend van hun microservices op te bouwen. Deze relatief nieuwe taal heeft reeds enkele ondersteunde opties op vlak van logging frameworks. De twee meest gebruikte zijn Glog en Logrus (Dietrich, 2018). In dit onderzoek zal enkel Logrus ter sprake komen omdat deze het meest actief ondersteund wordt en omdat Logrus het gekozen logging framework is van Be-Mobile (De Valck, 2019).

Glog:

- Uitgebracht door Google;
- Simpele stijl, gebaseerd op die van Golang zelf (Dietrich, 2018);
- Laatste commit 25 januari 2016 (Symonds, 2019);

Logrus:

- Community driven;
- Goede ondersteuning;
- Laatste commit 3 maart 2019 (Eskildsen, 2019);

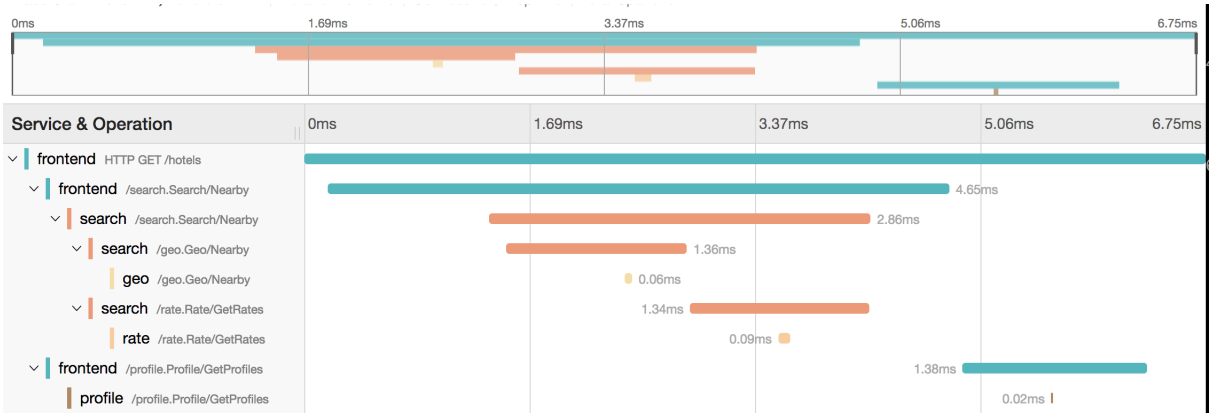
2.6.4 Logrus

“Logrus is een gestructureerde logger voor Go, volledig API compatibel met de standaard library logger” (sirupsen, s.d.). Logrus is een uitbreiding op de standaard library logger. Het biedt een wijd gamma aan functionaliteit bij het loggen van events. De verschillende levels waarin gelogd kan worden zijn: Trace, Debug, Info, Warning, Error, Panic, en Fatal. Deze levels zorgen voor een duidelijke weergave van het gelogde event. Hiermee kan de gebruiker, indien deze gebruik maakt van een log collector, filteren op level en zo vlotter de verschillende events doorlopen tijdens het debuggen. Verder kunnen ook nog andere fields toegevoegd worden die de gebruiker extra mogelijkheden biedt tijdens het filteren. Ten slotte kunnen de logs zowel de console als een wijd gamma andere keuzes als output gebruiken. Logrus kan gebruik maken van hooks om de data door te sturen naar message brokers zoals redis en kafka, alsook naar een file die lokaal opgeslagen kan worden. Hooks zijn zoals de term al doet vermoeden inhakingen op logrus. Hiermee kan er vooraleer de logs in de output verschijnen nog iets met gedaan worden (Eskildsen, 2019).

2.6.5 Tracing

Naast logs is er nog een tweede soort data die belangrijk is in microservices, trace data. De bedoeling van dit soort data is het achterhalen van de volledige chain die doorlopen wordt door een enkel event. Doorheen de chain kan dan gekeken worden hoelang elke microservice gewerkt heeft of hoelang een bepaalde API-call heeft geduurd (zie figuur 2.5). Deze data kan daarna gebruikt worden om metrics mee te maken, bv. De gemiddelde duur van een API-call. Zo kunnen uitschieters onderzocht worden om tot de kern van een probleem te komen (M. Rouse, 2018).

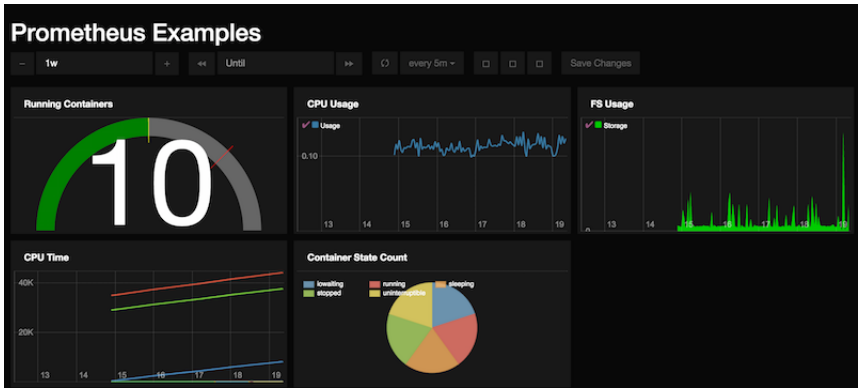
OpenTracing is een specificatie welke gebruikt wordt door de grootste Tracers. Omdat OpenTracing vendor-neutral is kan men dus kiezen welke Tracer men uiteindelijk wil gebruiken bij het verzamelen van traces (Rouse, 2018). De meest gebruikte Tracers voor Go zijn Zipkin en Jaeger (Sabic, 2018).



Figuur 2.5: Voorbeeld van tracing in Jaeger Ward, 2015

2.6.6 Metrics

De derde en laatste soort data die gebruikt wordt om een applicatie te monitoren is metrics. Deze houden de systeembeheerders op de hoogte van de status van de applicatie. Dit werk maakt een vergelijkende studie tussen de meest populaire logging solutions, dus buiten deze korte introductie zal er niet verder ingegaan worden op metrics. Metrics maken het mogelijk om in een oogopslag de algemene gezondheid van een applicatie te zien (zie figuur 2.6). Er wordt gebruik gemaakt van grafieken en statistieken om dit weer te geven. Het verschil met tracing is dat metrics de volledige situatie van een applicatie omvatten en tracing een detail observatie is van bepaalde events of services in de applicatie (Reichert, 2018).

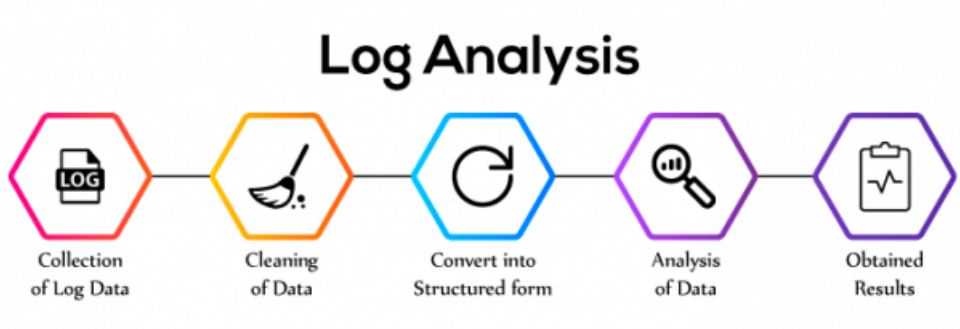


Figuur 2.6: Voorbeeld van metrics in Prometheus Christner, 2015

2.7 Logging solutions

2.7.1 Wat is een logging solution en hoe werkt het?

Voor er meer in detail gegaan wordt over de logging solutions die getest werden in dit werk, zal er een definitie beschreven worden wat beschouwd wordt als een logging solution. Een logging solution is een tool of een combinatie van tools die toelaat om efficiënt aan log management te doen. Het hele proces van logging doorloopt verschillende stappen (figuur 2.7) (LogDNA, g.d.).



Figuur 2.7: Het logging proces Nastel, g.d.

1. Data collectie: Ingestion en log aggregation. Een eerste stap hierbij is data die verzameld wordt vanuit verschillende bronnen. Deze bronnen kunnen de console logs zijn, de opgeslagen log files, syslog, etc. De verantwoordelijk voor dit proces is een collector. De tweede stap is het correct formateren van de verzamelde logs. Zonder deze stap zou de data niet uniform genoeg zijn om in een latere fase de analyse hierop uit te voeren (LogDNA, g.d.).
2. Filtering en analyse: De volgende schakel in het verhaal is de analyse. Eens de logs allemaal hetzelfde formaat hebben, kan er gefilterd worden. Dit kan op verschillende manieren gebeuren. Er kan gefilterd worden op tijd, log level (zie 2.6.4 Logrus), extra toegevoegde tags, service waar de log plaatsvond, en nog veel meer. De analyse gebeurt op basis van grafieken en statistieken. Er is mogelijkheid om bepaalde statistieken bij te houden om gemakkelijk het overzicht te bewaren over situaties die

zich kunnen voordoen (LogDNA, g.d.).
3. Reporting. Nadat de analyse afgerond is en alle grafieken en statistieken aangemaakt zijn, kan er reporting ingesteld worden voor deze. Er kunnen alarmen aangemaakt worden die op verschillende manieren de verantwoordelijke personen op de hoogte stellen dat een bepaalde situatie zich heeft voorgedaan. Bijvoorbeeld wanneer een service een bovengemiddeld aantal error logs produceert op korte termijn (LogDNA, g.d.).
4. Opslag. Hierbij worden logs opgeslagen voor een bepaalde duur zodat deze beschikbaar blijven (LogDNA, g.d.).

2.7.2 Belangrijke punten in een logging solution

Wat is er belangrijk bij het zoeken naar een logging solution? Waar moet rekening mee gehouden worden in een beslissing?

1. Schaalbaarheid is van enorm belang in de groeiende sector van IT. Een bedrijf dat snel groeit kan zich niet veroorloven steeds opnieuw de hele structuur aan te passen om deze te laten meegroeien. Een goede logging solution kan automatisch of met zo weinig mogelijk werk meegroeien met het bedrijf (LogDNA, g.d.). Een voorbeeld hierin is Be-Mobile (zie 2.1 Be-Mobile) (De Valck, 2019).
2. Overhead. Dit is de impact van de gekozen oplossing op het systeem. Zware oplossingen kunnen de applicatie vertraging en minder performant maken (Gifford, 2015).
3. Installatie (LogDNA, g.d.).
4. Gebruiksgemak. Alle developpers in het bedrijf moeten dezelfde tool kunnen gebruiken (LogDNA, g.d.).
5. Snelheid waarmee logs getoond kunnen worden in de gekozen visualisatie tool (LogDNA, g.d.).
6. Automatische parsing van de logs bij het innemen van de logs (LogDNA, g.d.).

2.7.3 ELK

ELK is een acronym voor Elasticsearch, Logstash, Kibana. Dit zijn de drie componenten die een ELK stack bevat. Vaak wordt bovenop deze componenten ook gebruik gemaakt van Beats. Dit is een lichtgewicht data collector die handig kan zijn bij Kubernetes deployments met vele nodes. Bij complexe pipelines zoals Be-Mobile waar een hoge throughput aan data aanwezig is, wordt aangeraden om gebruik te maken van een message broker tussen de data collectie en data processing die voor een betrouwbare volgorde zorgt. Voorbeelden hiervan zijn Kafka, RabbitMQ, Redis (zie figuur 2.8) (Berman, 2018b).

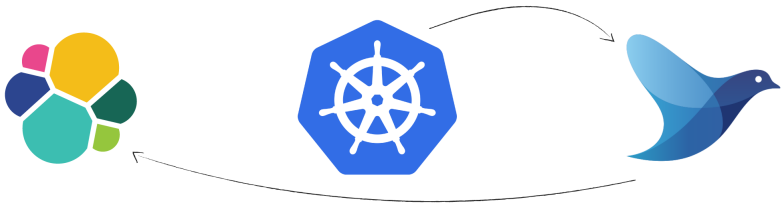
De werking van ELK gaat als volgt. Logs worden geproduceerd en opgeslagen in files. Beats verzamelt deze en stuurt ze meteen door naar een message broker. Daarna volgt Logstash. Deze haalt de logs op van de message broker en verwerkt deze. De logs worden hier gefilterd. Er kunnen via grok nieuwe benamingen gegeven worden aan delen van de log lijn. Ook kunnen dingen zoals datum, IP-adres, en dergelijke toegevoegd worden.

Figuur 2.8: Volledige ELK stack Berman, 2018b

Uiteindelijk worden de logs doorgestuurd naar Elasticsearch waar ze opgeslagen worden. De werking van Logstash wordt volledig bepaald door een config file. De laatste stap is Kibana, dit is de visualitie tool. Kibana gebruikt Elasticsearch als data source om de logs op te halen en toont deze met extra kleur accenten op tags om duidelijkheid te creëren Berman, 2018b; Levy, 2015.

De ELK stack is een goede logging oplossing omdat het een enorm krachtige tool is in combinatie met een gemakkelijk installatie. Naast zijn vele voordelen zijn er ook wat mindere punten aan de ELK stack die genoemd moeten worden. Hoewel de basis installatie vrij simpel uit te voeren is, wordt deze al snel ingewikkelder naargelang de grootte van de cluster waarop deze geïnstalleerd wordt. Er zijn veel factoren waar rekening gehouden mee moet worden en een verkeerde implementatie kan leiden tot inefficiëntie of zelfs crashes op lange termijn. Om deze reden hoort er genoeg aandacht besteed te worden aan resource management bij het opzetten van de ELK stack (Gifford, 2016).

2.7.4 EFK



Figuur 2.9: EFK stack Petrausch, 2017

Soortgelijk aan ELK, is EFK een acronym voor Elasticsearch, Fluentd, Kibana (zie figuur 2.9). Hierbij wordt op een soortgelijke manier gewerkt als bij ELK maar wordt Logstash vervangen door Fluentd. De redenen hiervoor zullen hieronder verder uitgelegd worden. Wanneer gewerkt wordt met Kubernetes moet de afweging gemaakt worden tussen gebruik van Fluentd en Fluentbit. Deze laatste is een nieuwere, kleinere variant van de eerste. Qua functionaliteit zijn ze beiden bijna evenwaardig met als enige grote verschil het aantal

ondersteunde plugins wat veel hoger is bij Fluentd. De grote kracht van Fluentbit is dan weer de grootte ervan. Met slechts 450KB is de voetafdruk van Fluentbit op gedistribueerde omgevingen enorm klein in vergelijking met Fluentd. Fluentbit is dan ook ontwikkeld met gedistribueerde omgevingen als primaire use case (Berman, 2018a).

Deze verschillen tussen Logstash en Fluentd zijn als volgt:

1. Taal. Logstash is geschreven in JRuby welke een Java implementatie is van Ruby en dus een Java Runtime vereist. Fluentd daarentegen is geschreven in CRuby, een C implementatie van Ruby en heeft dus minder vereisten qua omgeving. (Harikumar, 2018).
2. De routing van events. Waar Logstash voor routing zorgt door gebruik te maken van if-then statements, gebruikt Fluentd tags om hetzelfde te doen. Fluentd lijkt daarop iets makkelijk om te configureren (Harikumar, 2018).
3. Plugins. Zoals eerder vermeld heeft Fluentd een enorm grote hoeveelheid plugins met meer dan 350 plugins voor input, output, of verwerking van data. Logstash beschikt ook over een aantal plugins. Het grote verschil zit in de plaats waar deze plugins te vinden zijn. In tegenstelling tot Fluentd is er bij Logstash wel een gecentraliseerde github repository waar alle plugins te vinden zijn (Harikumar, 2018).
4. Message queue. Door gebruik te maken van Fluentd of Fluentbit wordt de nood aan een message broker geëlimineerd. Fluentd heeft namelijk een ingebouwde message queue die alle inkomende messages zelf correct verwerkt. Hierdoor kan Fluentd, in tegenstelling tot Logstash, geïmplementeerd worden in complexe systemen zonder externe message broker (Harikumar, 2018).
5. Geheugenverbruik. Waar Logstash gemiddeld 120MB aan geheugen gebruikt, zal Fluentd hetzelfde kunnen doen met maar 40MB. Beide blijven aan de hoge kant wanneer men een situatie schetst waar gebruik gemaakt wordt van honderden servers. De ontwikkelaars van beide log verzamelaars hebben een oplossing uitgebracht. Voor Logstash is dit Filebeat met een gemiddeld geheugen gebruik van 1,8MB. Voor Fluentd is dit Fluentbit met een gemiddeld geheugen gebruik van 450KB (Peri, 2015).

Om optimaal gebruik te maken van de EFK stack in een gedistribueerde omgeving op een Kubernetes cluster moet gebruik gemaakt worden van zowel Fluentbit als Fluentd. In deze setup bevat elke node in de cluster een Fluentbit pod die de logs van die node verzameld en doorstuurt naar de algemene Fluentd pod op de cluster. Deze fungeert als aggregator en verwerkt de logs om ze daarna door te sturen naar de gekozen output (Berman, 2018a).

2.7.5 Graylog

Graylog is ontstaan in 2009 in een tijd waar logging oplossingen, laat staan open source logging oplossingen quasi onbestaand waren. De marktleider in die tijd had een kostelijke licentie en daarom ontwikkelde de oprichter van Graylog een eigen open source oplossing. Dit wil zeggen dat Graylog vanaf het begin al een logging oplossing hoorde te zijn. Dit in tegenstelling tot ELK op basis van Elasticsearch, welke een full text search engine is. Graylog maakt zelf ook nog steeds gebruik van Elasticsearch als opslagplaats maar

dankzij de Graylog server die voor de databank gepositioneerd is wordt het gebruik ervan geoptimaliseerd voor logs (Graylog, g.d.-b).

De ouderdom van de Graylog betekent dat deze niet is geoptimaliseerd voor het gebruik ervan in een gedistribueerde omgeving zoals een Kubernetes cluster. Er is wel reeds documentatie toegevoegd om Graylog werkende te krijgen op een Kubernetes cluster (LUMIQ.AI, 2017).

Graylog maakt voornamelijk gebruik van een eigen log format dat geoptimaliseerd is voor Elasticsearch, namelijk GELF. Dit staat voor Graylog Extender Log Format. GELF is ondersteund als output bij verschillende log forwarders (Graylog, g.d.-b).

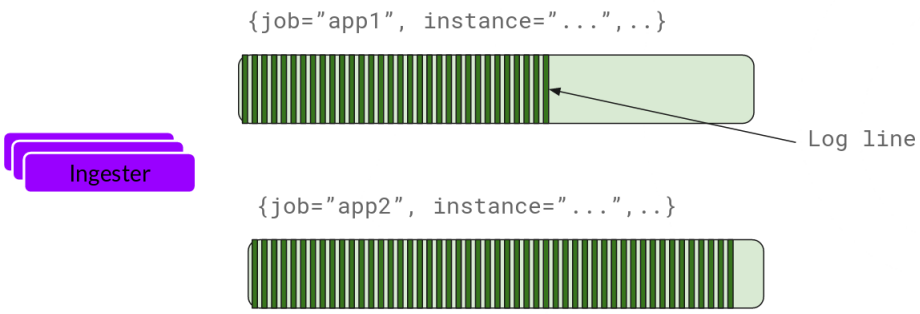
2.7.6 Grafana Loki

De meest recente logging solution, nog steeds in alfa ontwikkeling maar wel reeds beschikbaar. Wat meteen duidelijk wordt, is dat Loki ontwikkeld is voor het gebruik in grote, complexe, gedistribueerde omgevingen. Het is ontwikkeld om zoveel mogelijk logs te kunnen opslaan voor zo weinig mogelijk geld. Tot hiertoe zijn alle databanken gebaseerd op volledige indexering bij het opslaan van logs, dit zorgt voor heel wat opslagruimte die noodzakelijk is om alles draaiende te houden. Loki stapt af van indexering en is gebaseerd op Prometheus, een ander product van Grafana dat gebruikt wordt voor de opslag van metrics. De manier waarop Loki logs opslaat, werkt op basis van labels en enkel de metadata wordt geïndexeerd (oleksii, 2019).

Figuur 2.10: Grafana Loki distributor Grafana, g.d.

Een loki 'stack' bestaat uit 3 belangrijke componenten:

1. Promtail is verantwoordelijk voor het verzamelen van logs. Het is een scraper die op zoek gaat in alle Kubernetes pods naar log files en deze daarna doorstuurt naar Loki. Promtail wordt geïnitialiseerd op basis van een config file waarin beschreven wordt welke log files doorgestuurd worden, welke genegeerd worden, en welke anders



Figuur 2.11: Grafana Loki Ingester Grafana, g.d.

genoemd worden. Soortgelijk aan Fluentbit zal Promtail ook in elke node aanwezig zijn en logs doorsturen naar een algemene loki instantie (Veeramachaneni, 2018).

2. Loki is de datasource van de stack. Het maakt gebruikt van een distributor (zie figuur 2.10) om logs te verspreiden over verschillende ingesters (zie figuur 2.11) vooraleer de logs in een databank terecht komen. De manier waarop Loki logs opslaat is ook anders dan de concurrentie. De ingesters verzamelen soortgelijke logs in chunks. Wanneer een chunck opgevuld is, wordt deze opgeslagen in een Object Storage. Default is dit een lokale Object Storage. De indexen van deze logs worden daarna in een NoSQL database opgeslagen om snel opgehaald te kunnen worden. Deze manier van opslag zorgt voor een snelle ophaling van logs wanneer deze opgevraagd worden in Grafana. Ook zorgt deze manier van opslag voor een grote schaalbaarheid (Veeramachaneni, 2018).
3. Grafana is de frontend die zowel Loki als Prometheus en andere bronnen kan gebruiken als data source. Hier worden de logs gevisualiseerd. Op moment van schrijven van dit werk is er enkel tekstuele filtreerbaarheid beschikbaar op basis van een eigen query taal van Grafana. Ook deze is nog onder constructie en verwacht binnenkort een upgrade. Nummers worden dan ook filtreerbaar met mogelijkheid tot filteren op datum of bepaalde getallen vergelijken (davkal, 2019; Veeramachaneni, 2018).

3. Methodologie

Na de onderzoeksfase van dit werk in de vorm van een literatuurstudie (zie Hoofdstuk 2) volgt hieronder een requirementsanalyse die een opsomming maakt van de belangrijkste requirements. Deze zullen later in de conclusie (zie Hoofdstuk 8) belangrijk zijn in het beoordelen van de meest geschikte oplossing in deze situatie. Wegens het gebrek aan open-source oplossingen zal dit werk slechts 4 oplossingen vergelijken namelijk: ELK, EFK, Graylog, en Loki. Dit zorgt ervoor dat een long en short list niet nodig zijn in dit onderzoek en deze zullen dus achterwege gelaten worden.

De uitwerking van een testomgeving zal als volgt gebeuren. Wegens de manier waarop Minikube is opgebouwd, is het niet mogelijk om meerdere nodes te installeren op een eigen Minikube. Bijgevolg is het dus niet mogelijk om een testomgeving op te zetten waarbij gebruik gemaakt wordt van meerdere nodes. Dit zorgt ervoor dat de complexiteit van de Kubernetes testomgeving drastisch daalt en vervangen kan worden door een lokale testomgeving. Elke oplossing zal lokaal geïnstalleerd met een passende configuratie. Hierbij belangrijk is dat niet elke oplossing is opgebouwd vanuit hetzelfde model (zie onder) dus zullen er verschillen zijn in de implementatie hiervan.

De Elastic stack kan op een soortgelijke wijze opgesteld worden met een dummy data generator in Logstash. De logs worden doorgestuurd naar Elasticsearch en getoond in Kibana. In deze omgeving zal geen message broker opgesteld worden. De theorie hieromtrent wordt wel besproken.

Fluentd maakt gebruik van het push model. Dit betekent dat logs naar Fluentd gepusht moeten worden. Fluentd stelt zich open als http endpoint of luistert naar een TCP socket om deze te forwarden. In de testomgeving zuller er dummy logs geproduceerd worden via een fluentd plugin. Deze worden doorgestuurd naar Elasticsearch en getoond in Kibana.

De configuratie om dummy logs te produceren is te zien in Listing 3.1

```
<source>
    @type dummy
    dummy {"hello":"world"}
</source>
```

Listing 3.1: Fluentd dummy log generator

De Graylog stack maakt ook gebruik van Fluentd als log collector en forwarder. Een soortgelijke stack als de EFK stack zal hiervoor opgesteld worden met als verschil dat Graylog een eigen frontend voorziet.

Promtail maakt gebruik van het pull model, hiervoor moeten logs opgeslagen worden in een file. Deze file wordt automatisch herkend door promtail en doorgestuurd naar Loki. Om deze logs te produceren zullen 2 Go services (zie Listings 3.3 en 3.4) elk in hun eigen Docker container draaien. De services bestaan uit een oneindige loop die steeds enkele logs per second zal produceren en opslaan in een file.

```
input {
    generator {
        lines => [
            "line 1",
            "line 2",
            "line 3"
        ]
        # Emit all lines 3 times.
        count => 3
    }
}
```

Listing 3.2: Logstash dummy log generator

```
import (
    "errors"
    "os"
    "time"

    "github.com/sirupsen/logrus"
)

func main() {
    w, err := os.Create("/Users/benjaminvaniseghem/Documents/logs/
    logfile1.log")
    if err != nil {
        panic(err)
    }
    logger := logrus.New()
    logger.SetOutput(w)
    for {
        logger.Info("Info message")
        logger.Warn("Warning message")
        logger.Error("Error message", errors.New("Error"))
        time.Sleep(1200 * time.Millisecond)
    }
```

```
}
```

Listing 3.3: Log generator main function

```
import (
    "errors"
    "os"
    "time"

    "github.com/sirupsen/logrus"
)

func main() {
    w, err := os.Create("/Users/benjaminvaniseghem/Documents/logs/
    logfile2.log")
    if err != nil {
        panic(err)
    }
    logger := logrus.New()
    logger.SetOutput(w)
    for {
        logger.Info("Info 2 message")
        logger.Warn("Warning 2 message")
        logger.Error("Error message 2", errors.New("Error"))
        time.Sleep(1200 * time.Millisecond)
    }
}
```

Listing 3.4: Log generator 2 main function

Nadat de testomgeving klaar is, zullen een voor een de gekozen logging oplossingen getest worden. Hieronder wordt het hele testproces besproken.

1. Als eerste gebeurt de installatie en het correct configureren van de benodigde software van de oplossing.
2. Na de installatie zal gekeken worden naar de visualisatie van de logs. Hierbij wordt rekening gehouden met de mogelijkheden omtrent het implementeren van grafieken over de gegenereerde logs en alerting in bepaalde situaties.
3. De volgende stap is het ophalen van specifieke logs. Dit kan in veel gevallen door middel van een query taal of bepaalde filters te gebruiken.
4. Tot slot zal de installatie van de oplossing verwijderd worden zodat het proces zich kan herhalen met een andere oplossing.

Om de testen te herhalen op een Kubernetes cluster zijn extra configuraties nodig. In het kader van dit onderzoek is het interessant om deze te bespreken. De extra configuraties van elke oplossing zullen besproken worden in hun eigen respectievelijke stuk.

Kubernetes config file

Een Kubernetes config file bestaat uit veel parameters waardoor een service op veel manieren gedeployed kan worden. Hieronder zijn de belangrijkste algemene parameters

opgesomd. In elk hoofdstuk van de logging oplossingen zullen de gebruikte parameters dieper uitgelegd worden.

- apiVersion : Versie van de Kubernetes API dat gebruikt wordt
- kind: Welk soort object gecreëerd moet worden. Een object kan een van de volgende zijn:
  - Pod
  - Service
  - Volume
  - Namespace
  - ReplicaSet
  - Deployment
  - StatefulSet
  - DaemonSet
  - Job
- metadata: Extra informatie over het object om het makkelijker uniek te identificeren.
- spec: Hier bevindt zich de specificaties omtrent het object. Dit is veranderlijk voor elk soort object.

3.1 Requirementsanalyse

- **Functionele requirements**
  - Moet kunnen scalen naargelang de groeiende Kubernetes cluster
  - Ondersteuning voor verschillende plugins die data extra kunnen verwerken of naar meerdere locaties kunnen doorsturen
  - Moet logs efficiënt opslaan en ophalen
  - Ondersteuning voor cluster omgevingen zoals Kubernetes
  - Ondersteuning voor alerts
  - Ondersteuning voor visualisatie zoals grafieken
- **Niet-functionele requirements**
  - Moet open source zijn
  - Moet (relatief) eenvoudig te configureren zijn
  - Moet (relatief) eenvoudig te gebruiken zijn
  - Moet goed gedocumenteerd zijn
  - Moet een zo klein mogelijke impact hebben op de servers
  - Moet de logs overzichtelijk kunnen tonen

Bovenstaande requirements kunnen als volgt ingedeeld worden met de MoSCoW-techniek (Consortium, g.d.). Deze verdeeld de requirements in ‘Must have‘, ‘Should have‘, en ‘Could have‘ requirements. Deze verdelingen betekenen respectievelijk dat de requirement verplicht is, aanwezig zou moeten zijn maar niet doorslaggevend, of optioneel is.

- **Must have**
  - Moet open source zijn
  - Ondersteuning voor cluster omgevingen zoals Kubernetes
  - Moet een zo klein mogelijke impact hebben op de servers

  - Moet kunnen scalen naargelang de groeiende Kubernetes cluster
- **Should have**
  - Moet (relatief) eenvoudig te configureren zijn
  - Moet (relatief) eenvoudig te gebruiken zijn
  - Moet goed gedocumenteerd zijn
  - Moet de logs overzichtelijk kunnen tonen
  - Ondersteuning voor verschillende plugins die data extra kunnen verwerken of naar meerdere locaties kan doorsturen
  - Ondersteuning voor visualisatie zoals grafieken
- **Could have**
  - Ondersteuning voor alerts

Het hele test proces zal gebruik maken van een lokale testomgeving en zal dus niet het formaat hebben van een cluster zoals bij Be-Mobile of andere bedrijven die tot de doelgroep van dit onderzoek behoren. Bijgevolg zullen de requirements 'Moet een zo klein mogelijke impact hebben op de servers', 'Moet kunnen scalen naargelang de groeiende Kubernetes cluster', en 'Ondersteuning voor cluster omgevingen zoals Kubernetes' niet op deze manier onderzocht kunnen worden. Voor de conclusie van deze requirements zal er gerefereerd worden naar documentatie over de oplossingen in kwestie. Deze gegevens zullen dan vergeleken worden met elkaar.

Om het vergelijken van de oplossingen simpeler voor te stellen zal er voor elke requirement een cijfer van 1 tot 5 gegeven worden aan de oplossing in kwestie. Elk cijfer wordt ook verwerkt met een multiplier aan de hand van de belangrijkheidsgraad van dit requirement. Deze kunnen later vergeleken worden om een conclusie te vormen. Verder zal er een antwoord gegeven worden op de onderzoeksvraag. Zowel de conclusie als het antwoord op de onderzoeksvraag zijn te vinden in Hoofdstuk 8

4. ELK

4.1 Installatie en configuratie

Zoals reeds besproken in Hoofdstuk 2.7.3 ELK, bestaat deze stack uit drie belangrijke elementen die aangevuld kunnen worden met meerdere componenten om deze oplossing efficiënter te maken.

- Logstash
- Elasticsearch
- Kibana
- Filebeat (optioneel, vooral in Kubernetes omgevingen)
- Message broker zoals Kafka, Redis,... (optioneel, vooral in Kubernetes omgevingen)

Logstash is opgebouwd vanuit een pull model. Dit wil zeggen dat Logstash op zoek gaat naar een gedefinieerde input bron waar logs te vinden zijn. Er zijn verschillende inputmogelijkheden zoals file en syslog. Logstash kan ook gebruikt worden met een pushmodel. Het stelt zichzelf open om logs te ontvangen van Filebeats om deze verder te verwerken en door te sturen naar Elasticsearch. Om deze demonstratie eenvoudig voor te stellen wordt gebruik gemaakt van een log generator plugin van Elastic. Deze plugin configuratie is te zien in Listing 4.1. Zo kan gefocust worden op de drie belangrijkste componenten.

4.1.1 Lokale omgeving

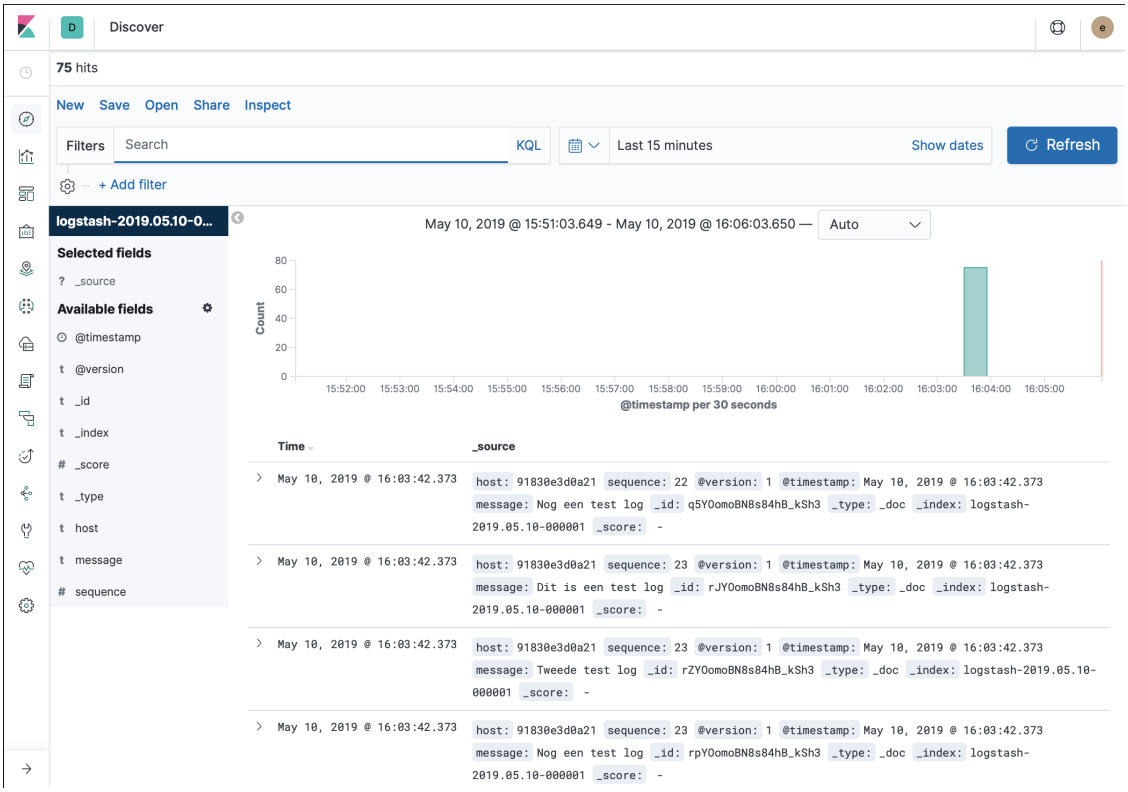
De eenvoudigste manier om een snelle testomgeving op te stellen is door gebruik te maken van een github repositoy met reeds bestaande configuratie en deze naar eigen wens aan

te passen. Met de repository https://github.com/deviantony/docker-elk kan mits enkele aanpassingen een ELK stack uitgerold worden. Hierbij moet de input van de Logstash config in de pipeline folder nog aangepast worden naar de log generator die te zien is in Listing 4.1. Verder wordt gebruik gemaakt van een docker-compose.yaml file om alle services met één commando uit te rollen. Door het gebruik van authenticatie bij deze installatie moet ingelogd worden bij Kibana om verder te gaan. De inloggegevens zijn ook te vinden in de repository.

Eens de setup uitgerold is moet er een index gecreëerd worden in Kibana om de logs te kunnen filteren. Daarna zijn de logs te zien in de 'Discover' pagina zoals te zien is in Figure 4.1.

```
input {
    generator {
        lines => [
            "Dit is een test log",
            "Tweede test log",
            "Nog een test log"
        ]
        # Emit all lines 25 times.
        count => 25
    }
}
```

Listing 4.1: Log generator plugin config Logstash



Figuur 4.1: ELK stack frontend

4.1.2 Kubernetes omgeving

De configuratie van de ELK stack voor Kubernetes is ingewikkelder. Hierbij wordt gebruik gemaakt van een StatefulSet voor de configuratie van Elasticsearch. Dit is lijkt op een gewone Deployment maar heeft enkele verschillen. De belangrijkste is dat elke replica van een Deployment exact hetzelfde is en deze dus uitwisselbaar zijn. In tegenstelling tot alle replicas van een StatefulSet. Deze zijn gebaseerd op eenzelfde configuratie maar zijn niet uitwisselbaar vanwege een unieke, gepersisteerde identifier. StatefulSets worden gebruikt stabiele networken opgebouwd moeten worden met een persistent storage.

De volledige configuratie is buiten de scope van dit onderzoek en zal dus niet besproken worden. De documentatie van Elastic biedt veel ondersteuning bij het opzetten van een eigen Elastic stack.

4.2 Requirements

4.2.1 Must have

Moet open source zijn

Voor elk van de onderzochte oplossingen geldt dezelfde score voor deze requirement, namelijk 5. Elk van de componenten waaruit een ELK of Elastic stack bestaat is vrij te verkrijgen en te gebruiken.

Ondersteuning voor cluster omgevingen zoals Kubernetes

In de documentatie van Elastic (Elastic, g.d.-a) zijn er verschillende links te vinden die een stap voor stap begeleiding geven om deze stack werkende te krijgen. Er zijn ook videos dit nogmaals aantonen dat er een goede ondersteuning is voor Kubernetes.

Om deze redenen zal hiervoor een 4 gegeven voor deze requirement.

Moet een zo klein mogelijke impact hebben op de servers

Bij velen is het al geweten dat Elasticsearch een grote impact heeft op het systeem waar het geïmplementeerd is. Een eerste argument hiervoor is de JVM (Java Virtual Machine) waarmee Elasticearch werkt. Deze heeft volgens Elastic, g.d.-b een standard minimum heap size van 1GB maar wordt in productie snel omhoog gescaled om ervoor te zorgen dat Elasticsearch functioneel blijft. Dit in combinatie met de best practice voor het installeren op Kubernetes gebruikt door Swidler, 2018 waarin minstens 3 nodes van Elasticsearch aangemaakt worden, zorgt meteen al voor een hoog RAM gebruik. Een tweede argument hiervoor is de manier waarop Elasticsearch data, in de geval logs, opslaat. Dit gebeurt door middel van volledige indexering waardoor de benodigde opslagruimte groter zal zijn dan andere databanken die indexering vermijden of slecht enkele delen van de data indexeren.

De combinatie van deze twee argumenten zorgt voor een relatief grote impact op de servers. Daarom krijgt de ELK of Elastic stack een score van 2 voor dit requirement.

Moet kunnen scalen naargelang de groeiende Kubernetes cluster

Indien gebruik gemaakt wordt van een DaemonSet voor de configuratie van FileBeats zal het aantal FileBeat containers automatisch gescaled worden bij het aanmaken van nieuwe pods. Wanneer de Kubernetes cluster een bepaalde grootte heeft, zal ook het aantal Elasticsearch nodes groeien. Deze kunnen automatisch gescaled worden met de correcte configuratie.

Indien gesteld wordt dat de oorspronkelijke configuratie correct is gebeurd, kan gesteld worden dat de ELK of Elastic stack automatisch scaled naargelang de groei van de cluster. Daarom krijgt de ELK of Elastic stack een score van 5 voor deze requirement.

4.2.2 Should have

Moet (relatief) eenvoudig te configureren zijn

De best practices om Elasticsearch te configureren op Kubernetes zijn dat er minsten 3 nodes aangemaakt moeten worden. 1 master en meerdere data nodes. Er zijn talloze manieren waarop Elasticsearch kan geïnstalleerd worden en heel wat zaken die kunnen mislopen.

FileBeat, Logstash, en Kibana zijn eenvoudig te configureren en vormen geen probleem, zowel lokaal als op Kubernetes.

Om deze redenen krijgt de ELK of Elastic stack een 2 voor dit requirement

Moet (relatief) eenvoudig te gebruiken zijn

De leercurve om gebruik te maken van Elasticsearch is vrij hoog, terwijl de leercurve om gebruik te maken van Kibana relatief laag is.

Voor de requirement krijgt de ELK of Elastic stack een score van 4.

Moet goed gedocumenteerd zijn

Dankzij de maturiteit van Elastic is er reeds veel documentatie aanwezig, zowel in de vorm van eigen documentatie en documentatie vanuit de user community. De documentatie van Elastic zelf is zeer uitgebreid voor algemeen gebruik maar voor speciale use cases zal vaak eerder documentatie van de community gebruikt moeten worden.

Voor de requirement krijgt de ELK of Elastic stack een score van 4.

Moet de logs overzichtelijk kunnen tonen

De frontend van ELK of Elastic stack is Kibana. Deze zorgt voor een overzichtelijke visualisatie van alle logs. Dankzij de volledige indexering van de data binnen Elasticsearch, is er een uitgebreid filtersysteem waar Kibana gebruik van kan maken. Ook kunnen er deep text queries uitgevoerd worden.

Om deze redenen krijgt de ELK of Elastic stack een 5 voor deze requirement.

Ondersteuning voor verschillende plugins die data extra kunnen verwerken of naar meerdere locaties kunnen doorsturen

Filebeat is de logcollector die de logs doorstuurt naar Logstash. Logstash zelf heeft een aantal ondersteunde plugins die gebruikt kunnen worden. Zo kan het een groot aantal inputs verwerken en de data doorsturen naar een groot aantal outputs. Dataverwerking plugins zijn ook beschikbaar maar in mindere mate.

Om deze redenen krijgt de ELK of Elastic stack een 3 voor deze requirement

Ondersteuning voor visualisatie zoals grafieken

Kibana voorziet functionaliteit voor het produceren van grafieken. Verschillende soorten grafieken en tellers zijn ondersteund.

Voor de requirement krijgt de ELK of Elastic stack een score van 5.

4.2.3 Could have

Ondersteuning voor alerts

Kibana voorziet functionaliteit voor alerts. Deze alerts kunnen via verschillende kanalen verzonden worden.

Voor de requirement krijgt de ELK of Elastic stack een score van 5.

4.3 Resultaten

Zoals te zien is in tabel 4.1 scoort de ELK stack een 2 op een Must Have requirement, namelijk ‘Moet een zo klein mogelijke impact hebben op de servers‘. Dit wijst erop dat ELK niet geschikt is voor grote clusters. Hoe groter de cluster, hoe groter de impact. Het scoort goed op overzichtelijkheid en functionaliteit.

Een uitgebreide conclusie is te vinden in Hoofdstuk 8 Conclusie.

Requirement	**Score (op 5)**	**Multiplier**	**Score met multiplier**
Moet open source zijn	5	10	50
Ondersteuning voor cluster omgevingen zoals Kubernetes	4	10	40
Moet een zo klein mogelijke impact hebben op de servers	2	10	20
Moet kunnen scalen naargelang de groeiende Kubernetes cluster	5	10	50
Moet (relatief) eenvoudig te configureren zijn	2	5	10
Moet (relatief) eenvoudig te gebruiken zijn	4	5	20
Moet goed gedocumenteerd zijn	4	5	20
Moet de logs overzichtelijk kunnen tonen	5	5	25
Ondersteuning voor verschillende plugins die data extra kunnen verwerken of naar meerdere locaties kan doorsturen	3	5	15
Ondersteuning voor visualisatie zoals grafieken	5	5	25
Ondersteuning voor alerts	5	1	5
Totale score	39		280

Tabel 4.1: ELK resultaten

5. EFK

5.1 Installatie en configuratie

Zoals reeds besproken in Hoofdstuk 2.7.4 EFK, bestaat deze stack uit drie belangrijke componenten.

- Fluentd
- Elasticsearch
- Kibana

Fluentd is opgebouwd vanuit een push model. Hierbij pusht de service zelf zijn logs naar Fluentd waar ze verwerkt worden. Om deze demonstratie eenvoudiger te tonen wordt hier gebruik gemaakt van dummy logs als input voor Fluentd. Dit zijn standaard logs die geproduceerd worden door een plugin.

5.1.1 Lokale omgeving

Om de EFK stack lokaal op te zetten wordt vaak gebruik gemaak van compose up. Compose up laat toe om meerdere docker containers tegelijk op te starten. Alle configuratie om de componenten draaiende te krijgen wordt hierin verwerkt. Voor Fluentd wordt binnenin deze compose-up.yaml (zie Listing 5.1) verwezen naar de fluentd.conf file (zie Listing 5.2). In deze .conf file worden de input en output gedefinieerd. In deze configuratie haalt Fluentd logs binnen van verschillende dummy plugins en stuurt deze automatisch door naar een Elasticsearch instantie alsook naar de stdOut. Belangrijk hierbij is de poort van Elasticsearch welke reeds gedefinieëerd werd in de compose-up.yaml (Listing 5.1).

Door simpelweg deze file uit te voeren zullen alle componenten live draaien in hun eigen Docker container. Wanneer dan naar localhost:5601 gekeken wordt kan men de Kibana frontend van deze stack zien. Na het definiëren van een index zullen op de 'Discover' pagina alle logs van deze index terug te vinden zijn. In Figuur 5.1 is het voorbeeld te zien van hoe Kibana eruit ziet met de gebruikte configuratie.

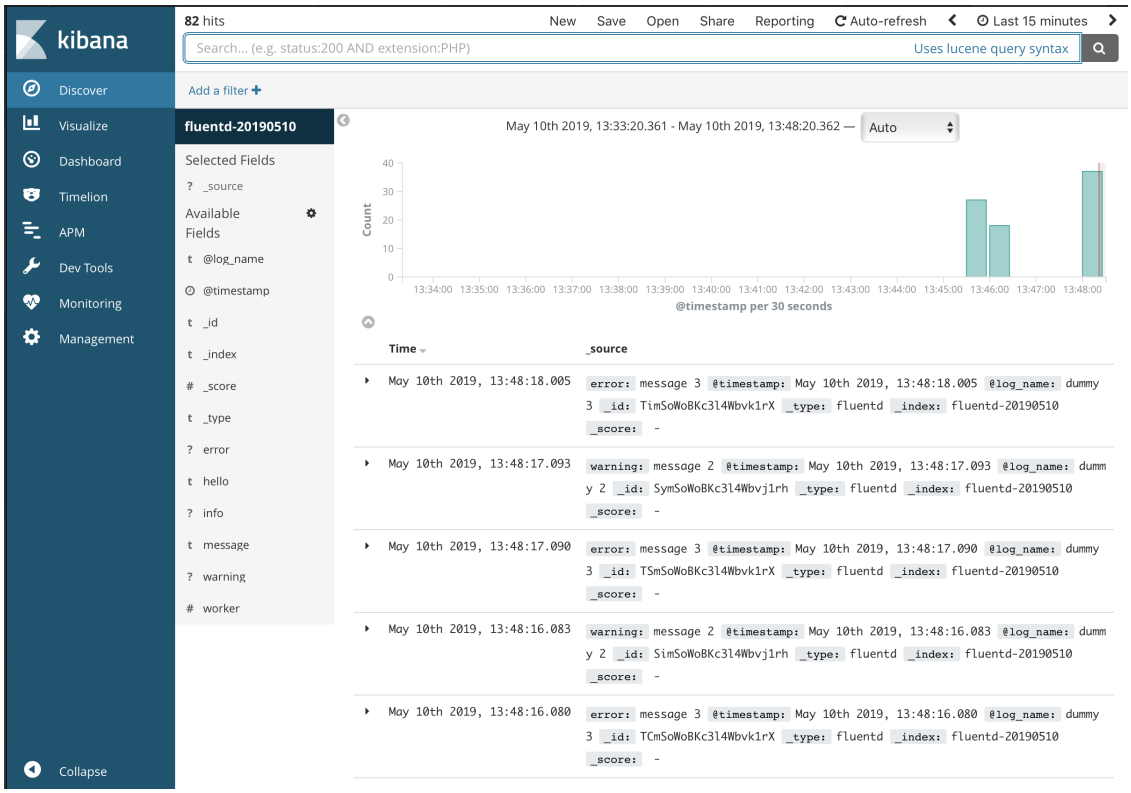
```
version: '2'
services:
    fluentd:
        build: ./fluentd
        volumes:
            - ./fluentd/conf:/fluentd/etc
        links:
            - "elasticsearch"
        ports:
            - "8080:8080"

    elasticsearch:
        image: docker.elastic.co/elasticsearch/elasticsearch:6.2.4
        environment:
            discovery.type: single-node
        expose:
            - 9200
        ports:
            - "9200:9200"

    kibana:
        image: docker.elastic.co/kibana/kibana:6.2.4
        links:
            - "elasticsearch"
        ports:
            - "5601:5601"
```

Listing 5.1: Compose up yaml file for EFK

```
<source>
    @type dummy
    dummy {"info":"message 1"}
    tag dummy 1
</source>

<source>
    @type dummy
    dummy {"warning":"message 2"}
    tag dummy 2
</source>

<source>
    @type dummy
    dummy {"error":"message 3"}
    tag dummy 3
</source>

<match **>
    @type copy
    <store>
```

```
22          @type elasticsearch
23          host elasticsearch
24          port 9200
25          index_name fluentd
26          type_name fluentd
27          logstash_format true
28          logstash_prefix fluentd
29          logstash_dateformat %Y%m%d
30          include_tag_key true
31          tag_key @log_name
32          flush_interval 1s
33      </store>
34      <store>
35          @type stdout
36      </store>
37  </match>
```

Listing 5.2: Fluentd config file



Figuur 5.1: EFK stack frontend

5.1.2 Kubernetes omgeving

De configuratie van de EFK stack voor Kubernetes is ingewikkelder. Hierbij wordt, net zoals bij de ELK stack, gebruik gemaakt van een StatefulSet voor de configuratie van Elasticsearch.

In het artikel van Jetha, 2018 wordt uitgebreid besproken hoe de EFK stack kan opgezet worden.

5.2 Requirements

5.2.1 Must have

Moet open source zijn

Voor elk van de onderzochte oplossingen geldt dezelfde score voor deze requirement, namelijk 5. Elk van de componenten waaruit een EFK stack bestaat, namelijk Elastischsearch, Fluentd, en Kibana, is vrij te verkrijgen en te gebruiken.

Ondersteuning voor cluster omgevingen zoals Kubernetes

Bij de EFK stack geldt een soortgelijke uitleg als bij de ELK of Elastic stack. Dezelfde documentatie welke gebruikt kan worden voor de ELK of elastic stack kan gebruikt worden voor de EFK stack. Met bijkomende documentatie over het configureren van zowel Fluentd als Fluentbit.

Deze argumenten leiden tot een score van 4 voor de EFK stack.

Moet een zo klein mogelijke impact hebben op de servers

Zoals eerder besproken in de sectie 2.7 heeft deze oplossing een minimale impact op de servers wanneer in acht wordt genomen dat er gebruik wordt gemaakt van Elastiscsearch als databank. Het enige verschilpunt met ELK is dus het gebruik van Fluentd en Fluentbit. Wanneer deze op een correct manier geconfigureerd zijn, zal deze zuiniger zijn op vlak van geheugen gebruik (zie 2.7.4, vergelijking tussen Fluentd en Logstash).

Er blijft wel gebruik gemaakt worden van Elasticsearch waardoor de score voor EFK niet hoger dan 3 kan zijn.

Moet kunnen scalen naargelang de groeiende Kubernetes cluster

Ook hier moet gebruik gemaakt worden van een DaemonSet voor de configuratie van Fluentbit om een automatische scaling te garanderen.

Indien gesteld wordt dat de oorspronkelijke configuratie correct is gebeurd, kan gesteld worden dat de EFK stack automatisch kan scalen naargelang de groei van de cluster. Daarom krijgt EFK een score van 5 voor dit requirement.

5.2.2 Should have

Moet (relatief) eenvoudig te configureren zijn

Voor EFK geldt eenzelfde uitleg als de ELK of Elastic stack voor dit requirement. De configuratie van Elastischsearch is ingewikkeld en neemt veel tijd in beslag om correct uit te voeren.

Fluentd en Fluentbit zijn relatief eenvoudig te configureren en vormen geen probleem voor deze requirement. Wel moet vermeld worden dat de configuratie van Fluentd en Fluentbit moeilijker is dan deze van Logstash en Filebeat (Harikumar, 2018).

Om deze redenen krijgt EFK, net als de ELK of Elastic stack, een 2 voor dit requirement.

Moet (relatief) eenvoudig te gebruiken zijn

Voor EFK geldt eenzelfde uitleg als de ELK of Elastic stack voor dit requirement. Er geldt een hoge leercurve voor Elasticsearch maar wanneer er sprake is van basis use cases zoals simpele log visualatie in de vorm van simpele grafieken, zal het gebruik van Kibana vanzelfsprekend zijn.

Om deze redenen krijgt EFK, net als de ELK of Elastic stack, een 4 voor dit requirement.

Moet goed gedocumenteerd zijn

Voor EFK geldt eenzelfde uitleg als de ELK of Elastic stack voor dit requirement. De Elasticsearch documentatie is zeer uitgebreid. De documentatie van Fluentd en Fluentbit is ook talrijk aanwezig in verschillende officiele en community bronnen.

Voor dit requirement krijgt EFK net als de EFK of Elastic stack een score van 4.

Moet de logs overzichtelijk kunnen tonen

Voor EFK geldt eenzelfde uitleg als de ELK of Elastic stack voor dit requirement. De frontend voor de EFK stack is Kibana. De EFK stack beschikt over dezelfde voordelen als de ELK of Elasticstack.

Om deze reden krijgt EFK dezelfde score als de ELK of Elastic stack, namelijk 5.

Ondersteuning voor verschillende plugins die data extra kunnen verwerken of naar meerdere locaties kunnen doorsturen

Bij de EFK stack wordt gebruik gemaakt van Fluentbit om de logs te verzamelen en door te sturen naar Fluentd. Deze verwerkt de logs en stuurt deze door naar Elasticsearch. Zowel Fluentbit als Fluentd beschikken over een groot aantal plugins om data te verwerken en door te sturen naar meerdere locaties.

Voor de requirement krijgt de EFK stack een score van 5.

Ondersteuning voor visualisatie zoals grafieken

Voor EFK geldt eenzelfde uitleg als de ELK of Elastic stack voor dit requirement. Door het gebruik van Kibana kan gesteld worden dat EFK een uitstekende visualisatie heeft voor alle logs.

Voor de requirement krijgt de EFK stack een score van 5.

5.2.3 Could have

Ondersteuning voor alerts

Voor EFK geldt eenzelfde uitleg als de ELK of Elastic stack voor dit requirement. Dankzij Kibana voorziet de EFK stack ondersteuning voor alerts. Deze kunnen via verschillende kanalen verzonden worden.

Voor de requirement krijgt de EFK stack een score van 5.

5.3 Resultaten

Zoals te zien is in tabel 5.1, scoort EFK een hoge score. Het scoort goed bij de Must Haves en kan dus als een geschikte oplossing beschouwd worden. Het laagste cijfer is een 2, wat gegeven is voor de requirement ‘Moet (relatief) eenvoudig te configureren zijn‘, dit is te wijten aan de complexiteit die gepaard gaat met Elasticsearch. Het scoort de maximale score op verschillende requirements, dit is vooral te wijten aan het gebruik van Kibana.

Een uitgebreide conclusie is te vinden in Hoofdstuk 8 Conclusie.

Requirement	**Score (op 5)**	**Multiplier**	**Score met multiplier**
Moet open source zijn	5	10	50
Ondersteuning voor cluster omgevingen zoals Kubernetes	4	10	40
Moet een zo klein mogelijke impact hebben op de servers	3	10	30
Moet kunnen scalen naargelang de groeiende Kubernetes cluster	5	10	50
Moet (relatief) eenvoudig te configureren zijn	2	5	10
Moet (relatief) eenvoudig te gebruiken zijn	4	5	20
Moet goed gedocumenteerd zijn	4	5	20
Moet de logs overzichtelijk kunnen tonen	5	5	25
Ondersteuning voor verschillende plugins die data extra kunnen verwerken of naar meerdere locaties kan doorsturen	5	5	25
Ondersteuning voor visualisatie zoals grafieken	5	5	25
Ondersteuning voor alerts	5	1	5
Totale score	47		300

Tabel 5.1: EFK resultaten

6. Graylog

6.1 Installatie en configuratie

6.1.1 Lokale omgeving

De opzet van Graylog in een lokale omgeving gebeurt relatief eenvoudig. De uitstekende documentatie van Graylog zorgt ervoor dat er meer dan een manier is om dit te doen. De manier waarop hier tewerk is gegaan, gaat als volgt. Er werd een docker-compose.yaml file aangemaakt met de configuratie die te zien is in Listing 6.1. Deze docker-compose definieert drie services, MongoDB, Elasticsearch, en Graylog. MongoDB is verantwoordelijk voor het opslaan van de metadata. Elasticsearch is de database waar alle logs naartoe worden gestuurd. Graylog is de frontend waar de logs zullen getoond worden.

Met het commando 'docker-compose up' wordt deze configuratie uitgevoerd en zal de Graylog server samen met Elasticsearch en MongoDb lokaal draaien. Daarna kunnen er logs naar Elasticsearch gestuurd worden. In dit voorbeeld is gekozen voor Fluentd omdat deze een plugin bezit dit GELF als output kan definiëren. GELF werd reeds uitgelegd in 2.7.5 Graylog. Dit configuratie van Fluentd is te zien in Listing 6.2. Deze configuratie wordt gebruikt bij het uitvoeren van Fluentd in het commando dat te zien is in Listing 6.3.

```
1  version: '2'
2  services:
3      # MongoDB: https://hub.docker.com/_/mongo/
4      mongodb:
5          image: mongo:3
6              # Elasticsearch: https://www.elastic.co/guide/en/
   elasticsearch/reference/6.6/docker.html
7          elasticsearch:
8              image: docker.elastic.co/elasticsearch/elasticsearch-oss
```

```
    :6.6.1
9           environment:
10              - http.host=0.0.0.0
11              - transport.host=localhost
12              - network.host=0.0.0.0
13              - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
14          ulimits:
15          memlock:
16          soft: -1
17          hard: -1
18          mem_limit: 1g
19          # Graylog: https://hub.docker.com/r/graylog/graylog/
20      graylog:
21          image: graylog/graylog:3.0
22          environment:
23              # CHANGE ME (must be at least 16 characters)!
24              - GRAYLOG_PASSWORD_SECRET=somepasswordpepper
25              # Password: admin
26              - GRAYLOG_ROOT_PASSWORD_SHA2=8
    c6976e5b5410415bde908bd4dee15dfb167a9c873fc4bb8a81f6f2ab448a918
27              - GRAYLOG_HTTP_EXTERNAL_URI=http://127.0.0.1:9000/
28          links:
29              - mongodb:mongo
30              - elasticsearch
31          depends_on:
32              - mongodb
33              - elasticsearch
34          ports:
35              # Graylog web interface and REST API
36              - 9000:9000
37              # Syslog TCP
38              - 1514:1514
39              # Syslog UDP
40              - 1514:1514/udp
41              # GELF TCP
42              - 12201:12201
43              # GELF UDP
44              - 12201:12201/udp
```

Listing 6.1: docker-compose.yaml

```
1  <source>
2      @type dummy
3      dummy {"info":"message 1"}
4      tag dummy 1
5  </source>
6
7  <source>
8      @type dummy
9      dummy {"warning":"message 2"}
10     tag dummy 2
11 </source>
12
13 <source>
14     @type dummy
15     dummy {"error":"message 3"}
```

```
    tag dummy 3
</source>

<match **>
    type gelf
    host 0.0.0.0
    port 12201
    <buffer>
        flush_interval 5s
    </buffer>
</match>
```

Listing 6.2: Fluentd configuratie

```
$ fluentd -c ./fluent/fluent.conf
```

Listing 6.3: Uitvoeren van Fluentd met configuratie

Nadat Graylog en Fluentd volledig zijn uitgerold, kan naar localhost:9000 gesurft worden om de Web Interface te zien. Hier moet eerst nog ingelogd worden met de inloggevens:

- username: admin
- password: admin

Na het inloggen kan bij systeem->inputs een databank ingegeven worden waar op zoek wordt gegaan naar logs. Dit scherm is te zien in Figuur 6.3. Hier moet dezelfde poort gebruikt worden als deze gedefinieerd in de docker-compos configuratie van Graylog. Na het toevoegen van een input zijn de verstuurde logs te zien op de startpagina, zie Figuur 6.1. Wanneer extra details van een log gewenst zijn, kan doorgeklikt worden hierop om een detailscherm te verkijgen, zie Figuur 6.2.

6.1.2 Kubernetes omgeving

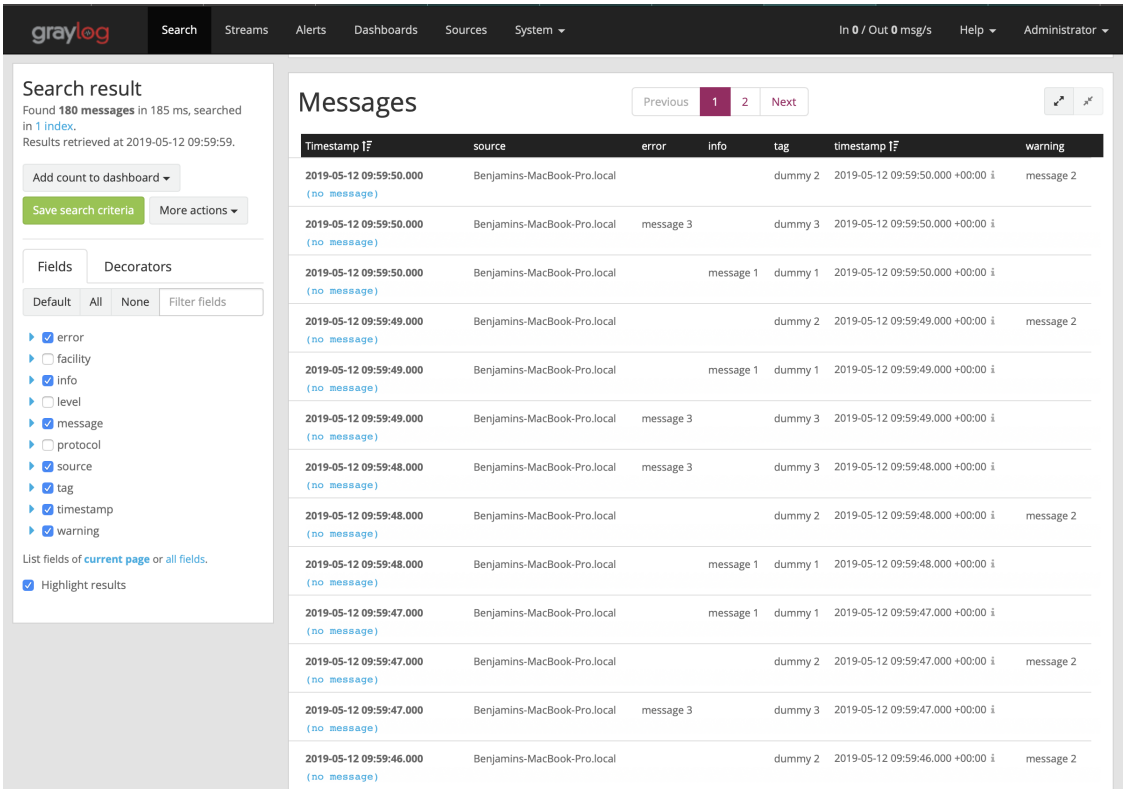
Graylog configureren voor een Kubernetes omgeving is geen eenvoudige taak. Er wordt best gebruik gemaakt van github repositories om deze aan te passen naar eigen use case. Een dieper inzicht in deze complexe configuratie is overbodig in dit onderzoek en zal dus overgeslagen worden.

6.2 Requirements

6.2.1 Must have

Moet open source zijn

Voor elk van de onderzochte oplossingen geldt dezelfde score voor deze requirement, namelijk 5. Graylog is open source en kan vrij gebruikt worden op eigen servers.



Figuur 6.1: Graylog stack frontend

Ondersteuning voor cluster omgevingen zoals Kubernetes

Zoals reeds vermeld in 2.7.5 zorgt de ouderdom van deze oplossing ervoor dat deze niet geoptimaliseerd is voor een gedistribueerde omgeving. Graylog heeft wel al een oplossing aangeboden waardoor het mogelijk wordt om de oplossing in een Kubernetes cluster te plaatsen.

Om deze reden krijgt Graylog een score van 2 op dit requirement.

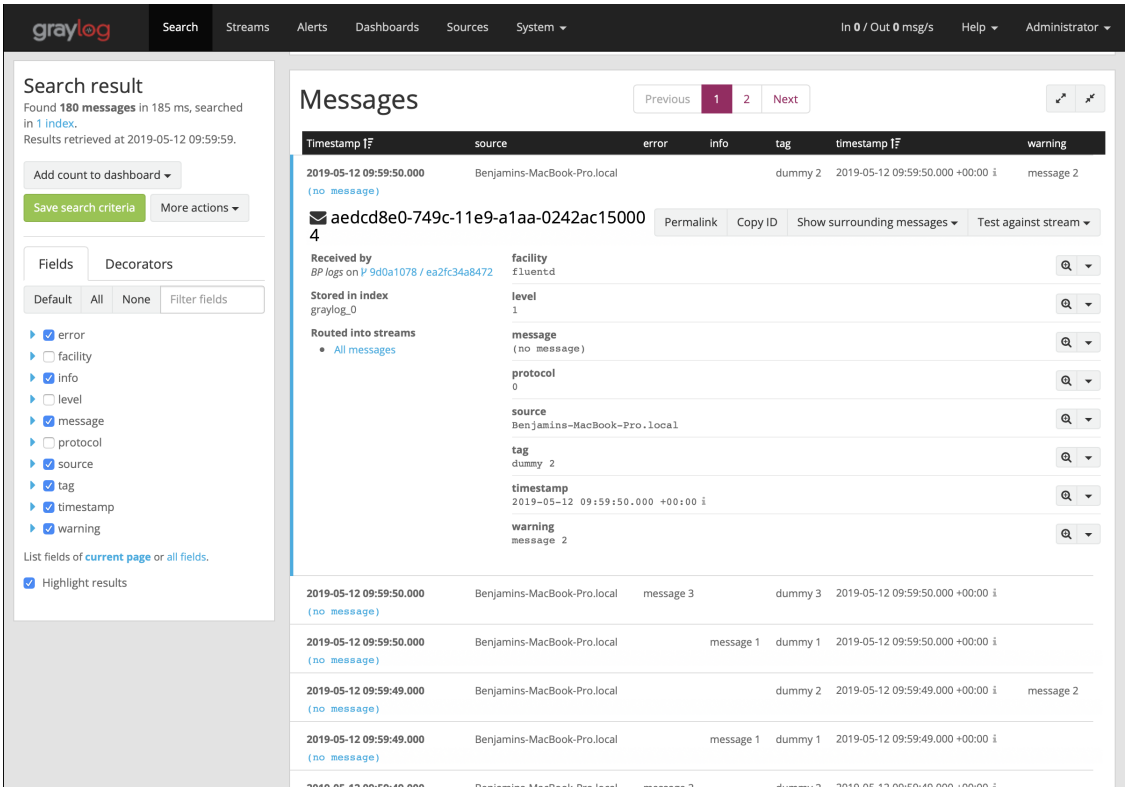
Moet een zo klein mogelijke impact hebben op de servers

Voor Graylog geldt een zelfde uitleg als de ELK of Elastic stack voor dit requirement. Het maakt gebruik van Elasticsearch voor de opslag van logs. Het verschil hierin is dat Graylog gebruik maakt van een eigen log format GELF waardoor de opslag ervan geoptimaliseerd wordt (Graylog, g.d.-a).

Om deze reden krijgt Graylog een score van 3 op dit requirement.

Moet kunnen scalen naargelang de groeiende Kubernetes cluster

Het is mogelijk om Graylog mee te scalen met een groeiende cluster. Dit vergt echter wel de toevoeging van Fluentbit of Fluentd als een DaemonSet. Dit zorgt voor extra



Figuur 6.2: Graylog log detail

configuratie waar in een latere requirement rekening mee gehouden moet worden.

Om deze reden krijgt Graylog een score van 3 op dit requirement.

6.2.2 Should have

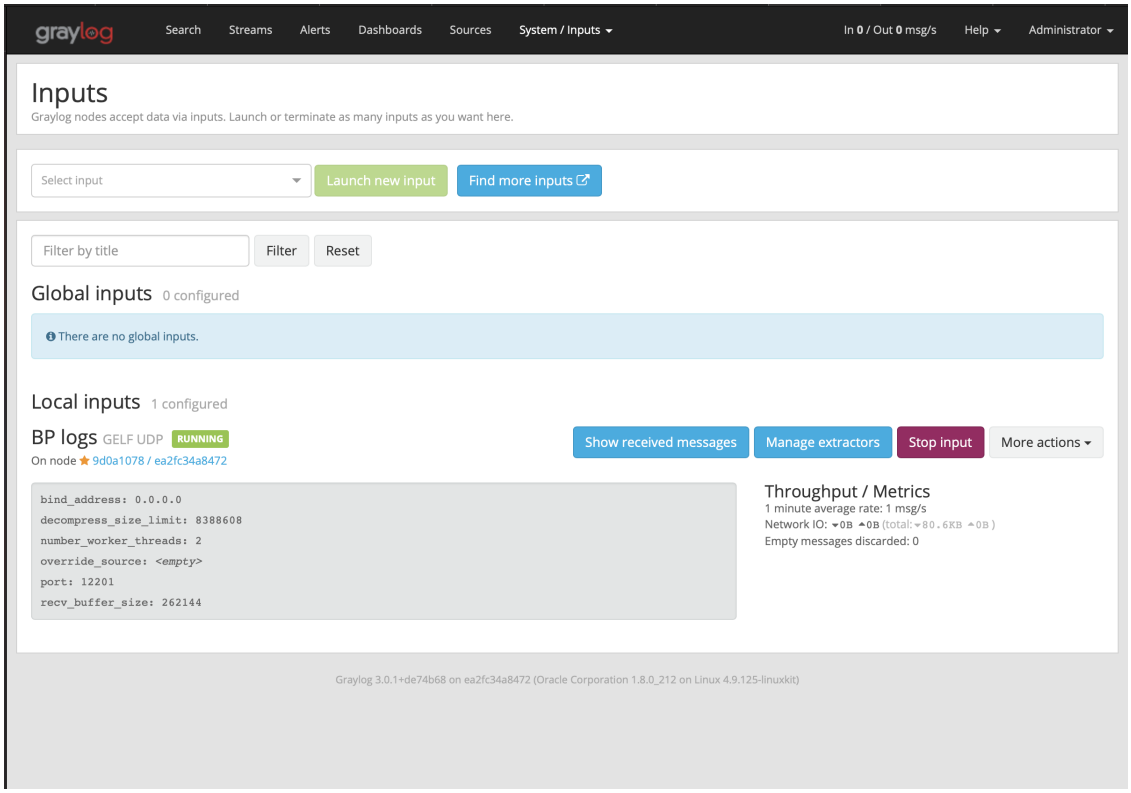
Moet (relatief) eenvoudig te configureren zijn

Zoals hierboven bij de requirement ‘Moet kunnen scalen naargelang de groeiende Kubernetes cluster‘reeds vermeld werd, zorgt de toevoeging van Fluentbit of Fluentd voor extra configuratie. Dit komt bovenop de configuratie van Elasticsearch, Graylog zelf, en MongoDB voor het opslaan van de metadata. De configuratie van Elasticsearch is hetzelfde als reeds besproken in Hoofdstuk 5 EFK.

Om deze redenen krijgt Graylog een score van 2 op dit requirement.

Moet (relatief) eenvoudig te gebruiken zijn

Na de installatie is het moeilijkste achter de rug. De Graylog front end webservice beschikt over een soortgelijke functionaliteit als Kibana bij de ELK en EFK stacks. Bijgevolg kan een soortgelijk score gegeven worden aan Graylog, namelijk 4.



Figuur 6.3: Graylog input scherm

Moet goed gedocumenteerd zijn

De maturiteit van Graylog zorgt voor een overvloed aan documentatie voor het opstellen van Graylog in bijna elk soort omgeving, behalve Kubernetes. Aangezien de ondersteuning hiervoor nog steeds minimaal is, kan hetzelfde gezegd worden over de documentatie omtrent een Kubenernetes omgeving.

Omdat in het kader van dit onderzoek enkel gekeken wordt naar de documentatie omtrent Kubernetes deployment krijgt Graylog een score van 2 op dit requirement.

Moet de logs overzichtelijk kunnen tonen

De Graylog front end webservice beschikt over een soortgelijke functionaliteit als Kibana bij de ELK en EFK stacks. Hoewel de interface anders is opgebouwd dan die van Kibana kan dezelfde score gegeven worden aan elk van deze drie oplossingen, namelijk 5.

Ondersteuning voor verschillende plugins die data extra kunnen verwerken of naar meerdere locaties kunnen doorsturen

Dankzij de maturiteit van Graylog is er een grote hoeveelheid diverse plugins beschikbaar. Er is ook veel documentatie beschikbaar voor elke plugin en de manier waarop plugins in het algemeen gebruikt worden.

Om deze reden krijgt Graylog een score van 4 op dit requirement.

Ondersteuning voor visualisatie zoals grafieken

Voor Graylog geldt een zelfde uitleg als de ELK of Elastic stack voor dit requirement. Aangezien de de front end web service van Graylog en Kibana zo soortgelijk zijn, krijgen deze beiden dezelfde score, namelijk 5.

6.2.3 Could have

Ondersteuning voor alerts

Voor Graylog geldt een zelfde uitleg als de ELK of Elastic stack voor dit requirement. Aangezien de de front end web service van Graylog en Kibana zo soortgelijk zijn, krijgen deze beiden dezelfde score, namelijk 5.

6.3 Resultaten

Wanneer enkel gekeken wordt naar de eerste kolom met de scores op 5 is te zien dat Graylog slecht scoort op belangrijke requirements zoals ondersteuning voor cluster omgevingen, configuratie, en documentatie. De frontend van Graylog scoort dan weer goed met de overzichtelijkheid ervan. Wanneer de belangrijkheidsgraad in rekening wordt gebracht valt meteen op dat Graylog veel minder geschikt is voor de besproken use case dan de andere oplossingen. Een uitgebreide conclusie is te vinden in Hoofdstuk 8 Conclusie.

Requirement	**Score (op 5)**	**Multiplier**	**Score met multiplier**
Moet open source zijn	5	10	50
Ondersteuning voor cluster omgevingen zoals Kubernetes	2	10	20
Moet een zo klein mogelijke impact hebben op de servers	3	10	30
Moet kunnen scalen naargelang de groeiende Kubernetes cluster	3	10	30
Moet (relatief) eenvoudig te configureren zijn	2	5	10
Moet (relatief) eenvoudig te gebruiken zijn	4	5	20
Moet goed gedocumenteerd zijn	2	5	10
Moet de logs overzichtelijk kunnen tonen	5	5	25
Ondersteuning voor verschillende plugins die data extra kunnen verwerken of naar meerdere locaties kan doorsturen	4	5	20
Ondersteuning voor visualisatie zoals grafieken	5	5	25
Ondersteuning voor alerts	5	1	5
Totale score	40		245

Tabel 6.1: Graylog resultaten

7. Loki

7.1 Installatie en configuratie

7.1.1 Lokale omgeving

De lokale installatie van de Loki stack kan volledig overgenomen worden van de officiële documentatie van Loki die te vinden is op github.com/grafana/loki. Er zijn 3 belangrijke stappen bij installeren van deze stack.

1. Promtail zoekt de log files in de locaties die gespecifieerd worden in de configuratie (zie listing 7.1). Daarna stuurt deze ze door naar Loki waar ze opgeslagen worden.
2. Loki slaat de logs die via de gespecifieerde poort binnenkomen via Promtail.
3. Grafana is de frontend van deze stack, hier worden de eerste 1000 logs van Loki binnengehaald en overzichtelijk gepresenteerd.

Grafana biedt ook een gratis cloud hosted oplossing aan voor test doeleinden. Indien gebruik gemaakt wordt van deze oplossing moet enkel promtail geconfigureerd worden. Documentatie hiervoor is te vinden op de documentatie pagina van Grafana.

Promtail

De belangrijkste configuratie punten van de configuratie in Listing 7.1 worden hier besproken.

Een eerste belangrijk configuratie punt is de client url. Deze definieert waar de gevonden logs naartoe worden gestuurd. Bij de lokale configuratie zal dit altijd localhost zijn. De poort 3100 die in Listing 7.1 gedefineerd staat, is terug te vinden in Listing 7.2 als de poort

van Loki waarop geluisterd wordt naar binnenkomende calls.

Een tweede belangrijk configuratie punt is de scrapeconfigs. Deze definieert alles van het scrapen logs. In zijn simpelste vorm, zoals te zien is in Listing 7.1, gaat het slechts om 1 job die op zoek gaat naar logs in slechts 1 locatie. Deze configuratie kan snel ingewikkeld worden wanneer sprake is van meerdere jobs die elk meerdere locaties in de gaten houden. Elke locatie heeft zijn eigen label. Zo worden de logs in Grafana makkelijker gefilterd en zullen deze overzichtelijker te tonen zijn. Een andere manier om de configuratie ingewikkelder te maken is het blacklisten van sommige locaties. Dit wil zeggen dat sommige locaties genegeerd worden terwijl andere wel gescrapet worden. Extra informatie over de volledige configuratiemogelijkheden zijn te vinden op github.com/grafana/loki.

Deze configuratie wordt op zijn beurt gebruikt binnen docker container die te vinden is in de officiële docker hub repositories van Grafana, namelijk hub.docker.com/u/grafana. Bij het aanmaken van deze container moet ook een volume gespecifieerd worden. Dit volume is een locatie in de lokale omgeving die gekoppeld wordt met de container. Het commando in dit voorbeeld is te zien op Listing 7.2.

```
server:
    http_listen_port: 9080
    grpc_listen_port: 0

positions:
    filename: /tmp/positions.yaml

client:
    url: http://localhost:3100/api/prom/push

scrape_configs:
    - job_name: log-test
        entry_parser: raw
        static_configs:
        - targets:
            - localhost
        labels:
            job: bp
            __path__: /etc/promtail/logs/*.log
```

Listing 7.1: promtail-config yaml file

```
$ docker run --name promtail --volume "$PWD:/etc/promtail" grafana/
    promtail:master -config.file=/etc/promtail/config.yaml
```

Listing 7.2: Aanmaken van de promtail container

Loki

De belangrijkste configuratie punten van de configuratie in Listing 7.3 worden hier besproken.

Een eerste belangrijk configuratiepunt is de http listen port. Zoals reeds vermeld in de subsectie Promtail hierboven is dit de poort waar Loki naar luistert en Promtail logs naar

stuurt.

Een tweede belangrijk configuratiepunt is de ingester. De functie hiervan werd reeds besproken in de literatuurstudie van dit werk.

Een derde belangrijk configuratiepunt is de schema config. Deze definieert de databanken waar de chunks en indexen opgeslagen worden. De chunks worden in een objectstore opgeslagen. In een basis lokale configuratie zal dit het filesystem zijn zoals in Listing 7.3 maar er zijn reeds andere object store databanken ondersteund door Grafana. Voorbeelden hiervan zijn Google Cloud en AWS S3. De indexen worden opgeslagen in een NoSQL databank. Standaard staat deze ingesteld op boltdb, wat een lokale NoSQL databank is. Ook deze databank kan veranderd worden door een gewenste databank.

Een laatste belangrijk punt in de configuratie van Loki is de storage config. Zoals hierboven reeds vermeld worden de indexen en chunks opgeslagen in respectievelijk een NoSQL databank en een Object Store. De locatie hiervan wordt in deze configuratie gespecifieerd.

Deze configuratie wordt op zijn beurt gebruikt binnen docker container die te vinden is in de officiële docker hub repositories van Grafana, namelijk hub.docker.com/u/grafana. Het commando in dit voorbeeld is te zien op Listing 7.2.

```
auth_enabled: false

server:
    http_listen_port: 3100

ingester:
    lifecycler:
        address: 127.0.0.1
        ring:
            store: inmemory
            replication_factor: 1
        chunk_idle_period: 15m

schema_config:
    configs:
        - from: 0
            store: boltdb
            object_store: filesystem
            schema: v9
            index:
                prefix: index_
                period: 168h

storage_config:
    boltdb:
        directory: /tmp/loki/index

    filesystem:
        directory: /tmp/loki/chunks

limits_config:
```

```
32     enforce_metric_name: false
```

Listing 7.3: Loki config yaml file

```
1 $ docker run --name loki grafana/loki:latest -config.file=/etc/loki/
    local-config.yaml
```

Listing 7.4: Aanmaken van Loki container

Grafana

Om Grafana lokaal te hosten wordt gebruik gemaakt van Homebrew voor Mac OS. Met twee simpele commando's, Listing 7.5 en Listing 7.6 wordt Grafana opgestart op de standaard poort, namelijk 3000.

De volgende stap is Loki linken met Grafana. Dit gebeurt door Loki als datasource te selecteren in de Grafana UI. Daarna zijn de logs te zien in het Explore tab van de UI. Nu is het mogelijk om logs te filteren en te queriën op label of job zoals te zien is in Afbeelding 7.1.

```
1 $ brew install grafana
```

Listing 7.5: install grafana

```
1 $ brew services start grafana
```

Listing 7.6: start grafana

7.1.2 Kubernetes omgeving

De Kubernetes deployment van deze oplossing is relatief eenvoudig. Grafana ondersteund nog geen dashboards en alerts voor een Loki datasource, daarom is het niet nodig om services hiervoor te deployen. Grafana kan simpelweg deployed worden op Kubernetes met de out-of-the-box docker image die te vinden is op de docker hub repository van Grafana. In Listing 7.7 wordt het commando hiervoor getoond. Om de Grafana UI te bereiken moet de Kubernetes poort hiervan nog geforward worden naar een eigen poort naar keuze. Dit gebeurt door gebruik te maken van het commando in Listing 7.8

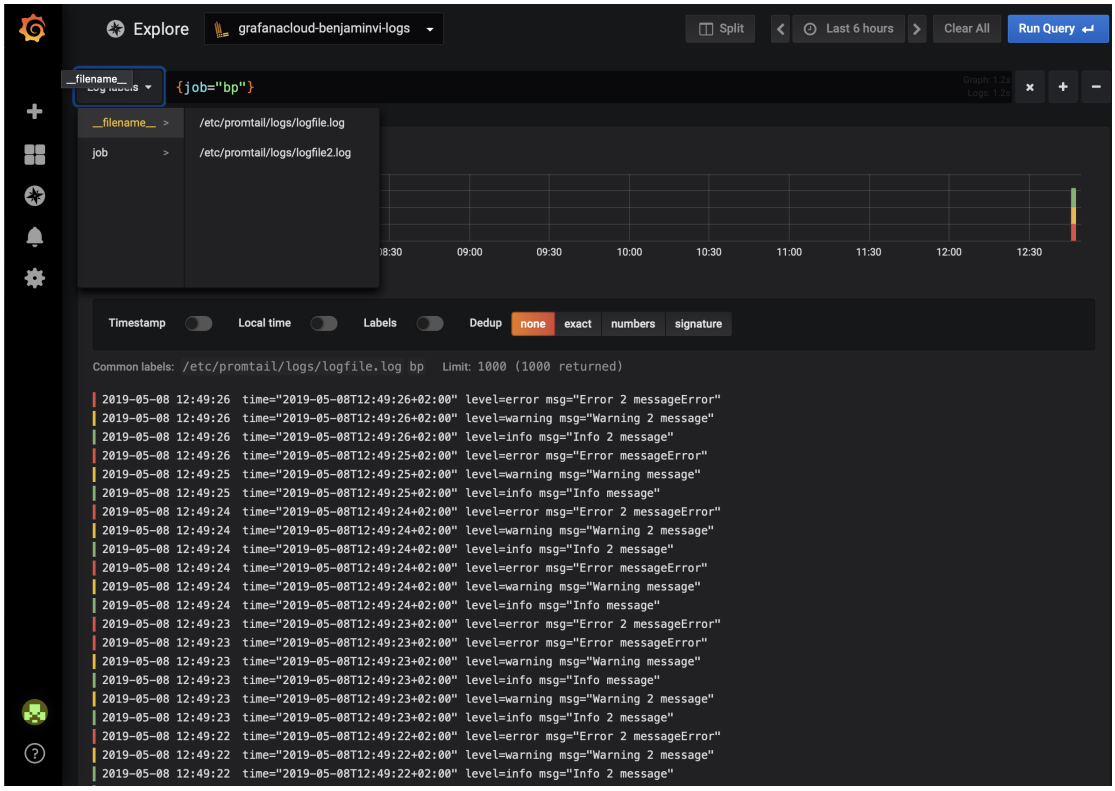
```
1 $kubectl create deployment grafana --image=docker.io/grafana/grafana:
    latest
```

Listing 7.7: Grafana Kubernetes deployment

```
1 $kubectl port-forward service/loki-grafana 3000:80
```

Listing 7.8: Port-forwarding van Grafana

De andere componenten Promtail en Loki zijn te installeren via Helm charts die beschikbaar zijn in de github pagina van Grafana, namelijk github.com/grafana/loki/tree/master/production. Door gebrek aan documentatie valt hier geen extra uitleg bij te geven over de exacte configuratie die gebruikt wordt in deze Helm charts.



Figuur 7.1: Loki query

7.2 Requirements

7.2.1 Must have

Moet open source zijn

Voor elk van de onderzochte oplossingen geldt dezelfde score voor deze requirement, namelijk 5. Elk van de componenten waaruit een Loki ‘stack‘ bestaat, namelijk Promtail, Loki, en Grafana, is vrij te verkrijgen en te gebruiken.

Ondersteuning voor cluster omgevingen zoals Kubernetes

Loki is speciaal voor Kubernetes ontwikkeld. Deze oplossing kan ook buiten Kubernetes gebruikt worden maar werkt het best in een Kubernetes cluster.

Om deze reden krijgt Loki een score van 5 op dit requirement.

Moet een zo klein mogelijke impact hebben op de servers

Loki heeft een minimale impact op de servers. Waar Elasticsearch gebruik maakt van volledige indexering bij het opslaan van logs, indexeert Loki enkel de metadata van logs waardoor het opslagverbruik lager ligt. Verder is ook nagedacht over de manier van het

opslaan. De indexen worden opgeslagen in een object store. De logs worden via een distributor ondergebracht in ingestors. Deze worden gevuld met soortgelijke logs en worden opgeslagen in een NoSQL databank eens deze opgevuld zijn.

Om deze redenen krijgt Loki een score van 5 op dit requirement.

Moet kunnen scalen naargelang de groeiende Kubernetes cluster

Promtail, de log scraper van Loki, kan ook geconfigureerd worden als een DaemonSet waardoor automatische scaling gegarandeerd kan worden.

Om deze reden krijgt Loki een score van 5 op dit requirement.

7.2.2 Should have

Moet (relatief) eenvoudig te configureren zijn

In het requirement 'Moet goed gedocumenteerd zijn' hieronder zal nog verder uitgelegd worden waarom er weinig documentatie beschikbaar is. Het gebrek aan documentatie leidt tot een moeilijke configuratie voor een nieuwe gebruiker. Elk van de component zijn relatief eenvoudig te configureren eens wat meer ervaring is opgedaan.

Om deze redenen krijgt Loki een score van 3 op dit requirement.

Moet (relatief) eenvoudig te gebruiken zijn

Het gebruiksgemak van Loki wordt meteen duidelijk na de installatie en configuratie. Het overzicht van de logs en de manier waarop logs worden opgeslagen met labels zorgen voor een aangename gebruikerservaring.

Om deze reden krijgt Loki een score van 5 op dit requirement.

Moet goed gedocumenteerd zijn

De documentatie van Loki is op het moment van schrijven quasi onbestaand. Buiten de officiële documentatie van Grafana zijn er slechts enkele bronnen beschikbaar. Dit is voornamelijk te wijten aan het feit dat Loki nog in een beta fase zit. Deze oplossing is nog niet zo lang geleden uitgekomen en wordt nog volop ontwikkeld.

Om deze reden krijgt Loki een score van 2 op dit requirement.

Moet de logs overzichtelijk kunnen tonen

Grafana maakt gebruik van een eigen interface om de logs te tonen. Deze worden in een lijst van 1000 onder elkaar geplaatsd. Wanneer specifiek op zoek moet worden gegaan is

er de mogelijk op te zoeken op labels en zo de getoonde lijst te verfijen. De query taal om logs te filteren is nog onder constructie waardoor de functionaliteit op dit moment vrij beperkt is. Dit is een belangrijke factor in de score van dit requirement

Om deze reden krijgt Loki een score van 2 op dit requirement.

Ondersteuning voor verschillende plugins die data extra kunnen verwerken of naar meerdere locaties kunnen doorsturen

Er zijn niet veel plugins beschikbaar die een soortgelijke functionaliteit hebben als beschreven in de requirement. Maar dit probleem is makkelijk op te lossen met het implementeren van Fluentd als log collector. Met deze configuratie scrapet Promtail al pod logs naar Fluentd. Deze verwerkt de logs en kan gebruik maken van een grote hoeveelheid plugins hiervoor. Na de verwerking worden de logs dan doorgestuurd naar Loki waar ze opgeslagen worden.

Om deze reden krijgt Loki een score van 4 op dit requirement.

Ondersteuning voor visualisatie zoals grafieken

Grafana heeft reeds ondersteuning voor grafieken en dergelijke voor het visualiseren van verschillende databronnen. Momenteel is Grafana ondersteuning voor een Loki databron aan het toevoegen voor dit onderdeel en zal ook aan deze requirement voldaan worden. Aangezien dit nog niet het geval is krijgt Loki een score van 1 voor deze requirement.

7.2.3 Could have

Ondersteuning voor alerts

Grafana heeft reeds ondersteuning voor alerts voor verschillende databronnen. Momenteel is Grafana ondersteuning voor een Loki databron aan het toevoegen voor dit onderdeel en zal ook aan deze requirement voldaan worden. Aangezien dit nog niet het geval is krijgt Loki een score van 1 voor deze requirement.

7.3 Resultaten

Zoals te zien is in tabel 7.1, scoort Loki in eerste instantie niet zo hoog bij een standaard score op 5. Wanneer deze score wordt verwerkt aan de hand van belangrijkheidsgraad, is te zien dat Loki wel degelijk veel potentieel biedt. De score is gebaseerd op hoe Loki functioneert op het moment van schrijven van dit werk. Het product heeft nog een lange weg te gaan maar heeft reeds een perfecte score op de vier Must Have requirements. Mits wat verbeteringen op vlak van documentatie en visualisatie zal de score van Loki nog verbeteren.

Requirement	**Score (op 5)**	**Multiplier**	**Score met multiplier**
Moet open source zijn	5	10	50
Ondersteuning voor cluster omgevingen zoals Kubernetes	5	10	50
Moet een zo klein mogelijke impact hebben op de servers	5	10	50
Moet kunnen scalen naargelang de groeiende Kubernetes cluster	5	10	50
Moet (relatief) eenvoudig te configureren zijn	3	5	15
Moet (relatief) eenvoudig te gebruiken zijn	5	5	25
Moet goed gedocumenteerd zijn	2	5	10
Moet de logs overzichtelijk kunnen tonen	2	5	10
Ondersteuning voor verschillende plugins die data extra kunnen verwerken of naar meerdere locaties kan doorsturen	5	5	25
Ondersteuning voor visualisatie zoals grafieken	1	5	5
Ondersteuning voor alerts	1	1	1
Totale score	39		291

Tabel 7.1: Loki resultaten

Een uitgebreide conclusie is te vinden in Hoofdstuk 8 Conclusie.

8. Conclusie

In dit onderzoek wordt een antwoord gegeven op de onderzoeksvraag ‘Wat is de beste open-source logging met tracing oplossing voor een bedrijf als Be-Mobile¿. Om dit te onderzoeken is een vergelijkende studie uitgevoerd tussen de voornaamste vier logging oplossingen, namelijk de Elastic stack, de ELK stack, Graylog, en Grafana Loki. Er werd gekeken naar enkele kernelementen van een goede logging oplossing en elk van de onderzochte oplossingen werd hieraan getoetst.

Als eerste conclusie kan gesteld worden dat logging en tracing verschillende onderwerpen zijn. Tracing werd ook onderzocht in dit werk en hoewel dit onderzoek zeer nuttig is geweest voor Be-Mobile is er besloten om de uitwerking hiervan niet op te nemen in dit werk.

Een tweede conclusie is dat er geen enkele oplossing voldoet aan de eisen die in het begin van dit onderzoek gesteld werden. EFK en ELK scoren uitstekend op visualisatie en documentatie, maar door de overhead die gepaard gaat met Elasticsearch zijn deze minder geschikte oplossingen. De complexiteit van de manier waarop Elasticsearch geconfigureerd moet worden in een Kubernetes cluster speelt hier een grote rol. Loki daarentegen zorgt voor een minimale overhead dankzij de architectuur ervan en biedt veel mogelijkheden. Het nadeel aan Loki is dat ontwikkeling ervan zich nog steeds in een beta fase bevindt. Dit leidt tot een gebrek aan documentatie en slechts een minimale ondersteuning in Grafana. Ondanks de jeugdigheid van Loki, kan het nu al aanschouwt worden als de meest geschikte oplossing in een microservice omgeving met een hoog aantal nodes. Waar EFK, ELK, en Graylog reeds volledig ontwikkeld zijn en dus vastzitten aan de manier waarop ze zijn opgebouwd, is Loki nog volop in ontwikkeling en bezit het dus veel potentieel om uit te groeien tot de ideale open source logging oplossing op de markt.

Een derde conclusie die kan getrokken worden uit dit onderzoek is dat elk van de onderzochte oplossingen beter gehost wordt door de service die de deze ontwikkeld heeft. De kost die gepaard gaat met de uitbreiding van een Kubernetes cluster zal op een gegeven moment de kost van hosting voorbij gaan. Het nadeel aan deze aanpak is het verlies van controle over de cluster. Hoewel dit zeker geen oplossing is in het geval van Be-Mobile, kan dit zeker een optie zijn voor een ander bedrijf die dit werkt leest.

Concreet kan gesteld worden dat de EFK stack de voordeligste optie is wanneer Elasticsearch wordt gebruikt. Loki is de oplossing met het meeste potentieel. De keuze hiertussen moet gemaakt worden op basis van budget. Wanneer beschikt wordt over een groot budget, zal EFK de meest geschikte oplossing zijn. Indien hier absoluut geen budget voor beschikbaar is, moet geïnvesteerd worden in de toekomst met Loki als meest geschikte oplosssing.

Deze conclusies waren volledig onverwacht. Zoals te zien is in het onderzoeksvoorstel van dit werk, is te zien dat oorspronkelijk tracing verwerkt was in dit werk. Ook werd verwacht dat op zoek zou gegaan worden naar een databank en logging framework, niet naar een volledige oplossing. Bij aanvang van het onderzoek zelf werd snel duidelijk dat het werk zou gaan over de verschillende logging oplossingen. Zelfs op dat moment werden zulke conclusies niet verwacht.

Wegens de nogal opiniegerichte wijze van vergelijken in dit werk, wordt de nood aan een nieuw onderzoek duidelijk. Dit zou dieper ingaan op de impact op het systeem van elke oplossing. Een simulatie van een soortgelijke omgeving als bij Be-Mobile is hiervoor vereist. Deze vergelijking zou in een latere fase nogmaals uitgevoerd kunnen worden, dit wanneer Loki al wat verder ontwikkeld werd. De verwachte conclusie voor die vergelijking zou dan zeker in het verdeel van Loki zijn.

Dit werk kan als nuttig beschouwd worden door bedrijven of startups die op zoek zijn naar een logging backend voor hun microservices. De requirement ‘Moet kunnen scalen naargelang de groeiende Kubernetes cluster‘ is een belangrijke requirement voor een zulke doelgroep. Ook personen die interesse in microservices of logging in het algemeen kunnen dit werk gebruiken als basis om een diepere kennis te vergaren in het onderwerp.

A. Onderzoeksvoorstel

Het onderwerp van deze bachelorproef is gebaseerd op een onderzoeksvoorstel dat vooraf werd beoordeeld door de promotor. Dat voorstel is opgenomen in deze bijlage.

A.1 Introductie

Be-Mobile houdt zich bezig met het verwerken van traffic data. Hierbij wordt gebruik gemaakt van een pipeline van verschillende microservices en API's die alle binnenkomende data verwerken. Om het proces dat de data doorloopt duidelijker te maken en om eventuele problemen in de toekomst makkelijker op te sporen, zal er overal gepaste logging aan toegevoegd worden. Real time worden er dus veel logs per seconde gegenereerd. Daarom is het belangrijk om de juiste logging framework en databank te implementeren. Het werk beschrijft dus een databankonderzoek. Meer bepaald het op zoek gaan naar de meest geschikte combinatie van een logging framework en databank om een grote, constante flow van logs te centraliseren en op te slaan. Welke databank kan het best dit soort data opslaan. Welk logging framework is het meest geschikt in deze omgeving?

Elke situatie in verband met data is anders. Het is daarom belangrijk eerst eigen onderzoek te verrichten in plaats van zomaar een databank te implementeren omdat deze in een andere situatie optimaal was.

A.2 State-of-the-art

Een eerste reden voor dit onderzoek is de populariteit van deze taal. J. Rouse (2017) stelt dat de groei hiervan te wijten is aan het feit dat Go een lichtgewicht, open source taal is die past in de microservices architecturen van vandaag. De programmeertaal Go is gereleased in 2012 en werd vooral gebruikt door Google (Golang, 2018). De taal zelf heeft een grote community achter zich gekregen en blijft in populariteit groeien. Een andere reden voor dit onderzoek is het gebrek aan soortgelijke onderzoeken. Ongetwijfeld zijn er al onderzoeken zoals deze uitgevoerd, maar niet met de bedoeling om deze te publiceren, binnen een bedrijf bijvoorbeeld. Vandaar dat deze dus niet beschikbaar zijn online. Garcin (2017) stelt dat Go reeds veel packages en interfaces heeft die gebruikt kunnen worden in database koppeling en logging. Aan opties dus geen gebrek. Daarom is het interessant om dit onderzoek uit te voeren.

A.3 Methodologie

Het onderzoek start met het onderzoeken welk soort databank het meest geschikt is voor het opslaan van log bestanden, een relationele of NoSQL databank. Eens een beslissing hierover gemaakt is, kan dieper worden gezocht naar welke van de gekozen soort databanken het beste passen bij deze soort data. Nadien zal ook een lijst van mogelijke frameworks gemaakt worden die instaan voor het produceren van logs. De volgende stap in het proces is het maken van een webservice aan de hand van een vue.js frontend en Go backend dat op hoog tempo logs produceert. Dit om de echte omgeving te simuleren. Daarna zal elke databank om beurt getest worden in deze webservice om uit te maken welke het meest performant is in deze omgeving. De logging frameworks kunnen in deze webservice ook getest worden. De resultaten zullen dan statistisch verwerkt worden om een conclusie te maken.

A.4 Verwachte resultaten

Over resultaten kunnen er nog geen voorspellingen gemaakt worden, maar na een eerste literatuurstudie kan er verwacht worden dat een NoSQL databank het best zal fungeren in een omgeving zoals deze. De resultaten die geproduceerd zullen worden door de simulatie zullen over het algemeen niet erg verschillen tussen de verschillende databanken.

A.5 Verwachte conclusies

De combinatie van databank en logging framework is belangrijk en zal een grote rol spelen doorheen dit onderzoek. Het zal niet voldoende zijn om de meest performante databank uit te zoeken. Deze zal ook goed moeten kunnen samenwerken met de gekozen framework. Dit kan een extra moeilijkheid vormen doorheen het onderzoek.

Bibliografie

Berman, D. (2018a, juni 28). Fluentd vs. Fluent Bit: Side by Side Comparison. Verkregen van https://logz.io/blog/fluentd-vs-fluent-bit/

Berman, D. (2018b, december 20). The Complete Guide to the ELK Stack. Verkregen van https://logz.io/learn/complete-guide-elk-stack/

Carey, S. (2018, augustus 10). What are microservices? Verkregen van https://www.computerworlduk.com/applications/microservices-explained-is-this-just-tweaked-soa-or-something-much-bigger-3638372/

Casey, K. (2017, augustus 17). How to explain microservices in plain English. Verkregen van https://enterprisersproject.com/article/2017/8/how-explain-microservices-plain-english

Christner, B. (2015). How to setup Prometheus Docker Monitoring. Verkregen van https://www.brianchristner.io/how-to-setup-prometheus-docker-monitoring/

Consortium, A. B. (g.d.). MoSCoW Prioritisation. Verkregen van https://www.agilebusiness.org/content/moscow-prioritisation

Czanik, P. (2013, april 16). Why logging is important? Verkregen van https://www.syslog-ng.com/community/b/blog/posts/why-logging-is-important

davkal. (2019). Using Grafana to Query your logs. Verkregen van https://github.com/grafana/loki/blob/master/docs/usage.md

De Valck, J. (2019, februari 18). private communication.

Dietrich, E. (2018, juni 5). Getting started quickly with go logging. Verkregen van https://www.scalyr.com/blog/go-logging/

Elastic. (g.d.-a). Running the Elastic Stack on Kubernetes. Verkregen van https://www.elastic.co/guide/en/elastic-stack-get-started/current/get-started-kubernetes.html

Elastic. (g.d.-b). Setting the heap size. Verkregen van https://www.elastic.co/guide/en/elasticsearch/reference/current/heap-size.html

Ellingwood, J. (2018, mei 2). An Introduction to Kubernetes. Verkregen van https://www.digitalocean.com/community/tutorials/an-introduction-to-kubernetes

Eskildsen, S. (2019, februari 20). Logrus. Verkregen van https://github.com/sirupsen/logrus

Fuse, S. (2018, juli 30). Microservices, Explained. Verkregen van https://medium.com/@SourceFuse/microservices-explained-b3b052b6f05d

Garcin, P. (2017). Surveying The Go Database Landscape. Verkregen van https://www.activestate.com/blog/surveying-go-database-landscape/

Gifford, J. (2015, februari 19). Data is King: Measuring the Impact of Logging on Your Application. Verkregen van https://www.loggly.com/blog/measuring-the-impact-of-logging-on-your-application/

Gifford, J. (2016, mei 12). Logging: To ELK or not to ELK? That is the question. Verkregen van https://www.loggly.com/blog/to-elk-or-not-to-elk-that-is-the-question/

Golang. (2012). Golang Documentation. Verkregen van https://golang.org

Golang. (2018). The Go Blog. Verkregen van https://blog.golang.org

Grafana. (g.d.). Verkregen van https://grafana.com/loki

Grav. (g.d.). YAML Syntax. Verkregen van https://learn.getgrav.org/15/advanced/yaml

Graylog. (g.d.-a). Graylog Documentation. Verkregen van https://docs.graylog.org/en/3.0/

Graylog. (g.d.-b). The thinking behind the Graylog architecture and why it matters to you. Verkregen van http://docs.graylog.org/en/3.0/pages/ideas_explained.html

Harikumar, S. (2018, april 18). The LOG Battle: Logstash and Fluentd. Verkregen van https://medium.com/tensult/the-log-battle-logstash-and-fluentd-c65f2f7c24b4

IT Marketplace, i. (2016, juni 22). Why Use Go for Microservices. Verkregen van https://medium.com/@itmarketplace.net/why-use-go-for-microservices-831ee754bf20

Jetha, H. (2018, november 26). How To Set Up an Elasticsearch, Fluentd and Kibana (EFK) Logging Stack on Kubernetes. Verkregen van https://www.digitalocean.com/community/tutorials/how-to-set-up-an-elasticsearch-fluentd-and-kibana-efk-logging-stack-on-kubernetes

LearnItGuide. (2018, november 19). What is Kubernetes - Learn Kubernetes from Basics. Verkregen van https://www.learnitguide.net/2018/08/what-is-kubernetes-learn-kubernetes.html

Levy, T. (2015, mei 5). How to Deploy the ELK Stack in Production. Verkregen van https://logz.io/blog/deploy-elk-production/

LogDNA. (g.d.). What is Log Management? The Complete Logging Guide. Verkregen van https://logdna.com/what-is-log-management/

LUMIQ.AI. (2017, februari 14). Running Graylog on Kubernetes. Verkregen van https://medium.com/lumiq-tech/running-graylog-on-kubernetes-63022fc35da2

Nastel. (g.d.). Understanding Log Analytics, Log Mining & Anomaly Detection. Verkregen van https://www.nastel.com/blog/understanding-log-analytics-log-mining-anomaly-detection-2/

oleksii. (2019, maart 11). Collect and View Logs with Grafana Loki. Verkregen van https://medium.com/pharos-production/collect-and-view-logs-with-grafana-loki-33d9155ac581

Peri, N. (2015, november 8). Fluentd vs. Logstash: A Comparison of Log Collectors. Verkregen van https://logz.io/blog/fluentd-logstash/

Petrausch, C. (2017, januari 18). Kubernetes Logging with Fluentd and the Elastic Stack. Verkregen van https://www.inovex.de/blog/kubernetes-logging-with-fluentd-and-the-elastic-stack/

Reichert, D. (2018, januari 5). Logs and Metrics: What are they, and how do they help me? Verkregen van https://www.sumologic.com/blog/logs-metrics-overview/

Rouse, J. (2017). Why Go us skyrocketing in popularity. Verkregen van https://opensource.com/article/17/11/why-go-grows

Rouse, M. (2018, april 30). Distributed tracing. Verkregen van https://searchitoperations.techtarget.com/definition/distributed-tracing

Sabic, N. (2018, mei 23). Jaeger vs Zipkin – OpenTracing Distributed Tracers. Verkregen van https://sematext.com/blog/jaeger-vs-zipkin-opentracing-distributed-tracers/

Stories, S. T. (2017, juni 13). Kubernetes 101 DaemonSets. Verkregen van https://hackernoon.com/kubernetes-101-daemonsets-5-c5bbfcbb1579

Swidler, J. (2018, augustus 7). High Performance ELK with Kubernetes: Part 1. Verkregen van https://engineering.udacity.com/high-performance-elk-with-kubernetes-part-1-1d09f41a4ce2

Symonds, D. (2019, februari 20). Glog. Verkregen van https://github.com/golang/glog

Technopedia. (g.d.). Log Files. Verkregen van https://www.techopedia.com/definition/5445/log-file

Vaughan-Nichols, S. J. (2018, mei 21). What is Docker and why is it so darn popular? Verkregen van https://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular/

Veeramachaneni, G. (2018, december 12). Loki: Prometheus-inspired, open source logging for cloud natives. Verkregen van https://grafana.com/blog/2018/12/12/loki-prometheus-inspired-open-source-logging-for-cloud-natives/

Ward, H. (2015, november 22). go-micro-services. Verkregen van https://github.com/harlow/go-micro-services

Yegulalp, S. (2018, september 6). What is Docker? Docker containers explained. Verkregen van https://www.infoworld.com/article/3204171/what-is-docker-docker-containers-explained.html