HO
GENT

Faculteit Bedrijf en Organisatie

Monitoring-tools voor Docker en Kubernetes

Olivier Troch

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

Promotor:
Bert Van Vreckem

Instelling: HoGent

Academiejaar: 2020-2021

Tweede examenperiode

Faculteit Bedrijf en Organisatie

Monitoring-tools voor Docker en Kubernetes

Olivier Troch

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

Promotor:
Bert Van Vreckem

Instelling: HoGent

Academiejaar: 2020-2021

Tweede examenperiode

Woord vooraf

Deze bachelorproef ”Monitoring-tools voor Docker en Kubernetes” werd geschreven met als doel het voltooien van mijn opleiding toegepaste informatica aan de Hogeschool Gent. Het onderwerp was voor mij ongezien en was daarom zeker een uitdaging, maar wetende dat Kubernetes, Docker en het monitoren ervan in deze tijden relevant zijn maakte mij alleen maar nieuwsgieriger. Dit onderwerp is tot stand gekomen door een docent in Hogeschool Gent om een onderdeel van het lessenpakket aan te vullen.

Graag neem ik ook even de tijd om de mensen te bedanken zonder wiens hulp deze bachelorproef niet tot stand zou gekomen zijn.

Eerst en vooral wil ik mijn promotor, die tegelijk mijn copromotor was, bedanken voor de nodige hulp. Door middel van goede en duidelijke feedback kon ik deze bachelorproef verbeteren waar nodig. Evenals de kans om dit onderwerp te mogen onderzoeken en documenteren.

Ook zou ik ook mijn ouders en medestudenten willen bedanken. In het bijzonder Owen Van Damme, Emiel Van Belle en Rayen Nasra om mij the helpen met morele steun en goede feedback doorheen deze periode.

Ten laatste zou ik ook Cedric Detemmerman willen bedanken voor een goede basis opstelling om op verder te bouwen bij het uitwerken van mijn onderwerp.

Samenvatting

Toegepaste informatica in HoGent kon helaas niet beschikken over alle mogelijke technologieën die in de IT-wereld plaats vonden. Toch waren er bepaalde onderwerpen die de basiskennis van de studenten zeer goed konden versterken. Na het aanleren van Kubernetes en Docker was de monitoring van deze tools de logische volgende stap. Monitoring binnen de Kubernetes en Docker omgeving is om vele reden belangrijk en aan de hand van dit onderzoek werd onder andere een poging gedaan om dit te bewijzen.

De eerste stap in de bachelorproef was een situatie schetsen waarin de IT en HoGent zich toen bevonden. Het onderzoek focuste zich op het ontstaan van de virtualisatie en gecontaineriseerde omgeving en de bestaansreden van monitoring in zo een omgeving.

Via de methodologie werd er onderzocht welke monitoringtools daadwerkelijk geschikt waren om te gebruiken binnen het lessenpakket. De MoSCoW-analyse, wat een requirements analyse was, was hiervoor de beste onderzoeksmethode.

Na het vinden van een geschikte tool, werd hiervoor een gepaste Proof-of-Concept opgesteld. Een Proof-of-Concept die kon worden gerecreëerd door de studenten toepaste informatica, waarmee zij dan aan de slag kunnen om de wereld van monitoring te ontdekken.

Uit het onderzoek voor het vinden van een geschikte monitoringtool waren drie tools overgebleven: Sysdig, Zabbix en Prometheus. Alle drie de tools voldeden op dezelfde manier aan de voorwaarden uit de requirements analyse. Uiteindelijk werd toch beslist om te kiezen voor Prometheus op vertrouwen van de community achter de monitoring tools.

Inhoudsopgave

Lijst van figuren

Lijst van tabellen

1. Inleiding

1.1 Probleemstelling

Studenten binnen de opleiding toegepaste informatica in HoGent zijn niet bekend met technologieën en tools die gebruikt worden binnen het Kubernetes en Docker domein. Het monitoren van deze technologieën is daar een onderdeel van. Ten eerste zijn Kubernetes en Docker geen eenvoudige technologieën om te implementeren en ten tweede is er een groot aanbod van mogelijke tools die hiervoor gebruikt kan worden en weten we niet welke voor deze doelgroep geschikt is. Om dit onderzoek de ondersteunen zal er beroep gedaan worden op de bachelorproef van Detemmerman Cedric waarvan de opstart van Docker en Minikube (lightweight versie van Kubernetes) zal worden gebruikt.

1.2 Onderzoeksvraag

Zoals eerder vermeld is er een groot aanbod aan monitoringtools voor een Docker/Kubernetes environment. Om die reden zal dit onderzoek nagaan welke tools voor de studenten geschikt zijn. Zo bekomen we volgende hoofdonderzoeksvraag:

- Welke monitoringtool voor een Kubernetes/Docker environment is het meest geschikt voor de studenten toegepaste informatica om een goede basiskennis te bekomen.

Eveneens zijn er nog een aantal deelvragen die ook belangrijk zijn in dit onderzoek:

- Wat zijn de belangrijkste verschillen tussen de tools?

- Waar moeten de tools inzicht op geven?
- Welke Proof-of-Concept is voldoende om de tool aan te leren?

1.3 Onderzoeksdoelstelling

Dit onderzoek heeft als doel het vinden van een geschikte monitoringtool die voor de studenten toegepaste informatica een meerwaarde kan betekenen. Door dit onderzoek zal zowel een vergelijking van tools beschikbaar zijn, alsook een reproduceerbare proof-of-concept die studenten aan de hand van documentatie kunnen bekomen. Aan de hand van dit proof-of-concept kunnen de educatieve doeleinden bereikt worden.

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 en worden de gebruikte onderzoekstechnieken besproken om een antwoord te kunnen formuleren op de onderzoeksvragen.

In Hoofdstuk 4, 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 Virtualisatie

2.1.1 De geschiedenis van virtualisatie

Virtualisatie is een term die verwijst naar de abstractie van computerbronnen. Het doel van een virtuele computer omgeving is het gebruik van hulpbronnen/resources te verbeteren door middel van een geïntegreerd besturingsplatform voor gebruikers en applicaties, gebaseerd op aggregatie van heterogene en autonome middelen. Tegenwoordig is virtualisatie op meerdere niveaus belangrijk om systeembeveiliging, betrouwbaarheid en beschikbaarheid te verbeteren en garanderen alsook het verminderen van kosten en het zorgen voor meer flexibiliteit. Virtualisatie is snel uitgegroeid tot een go-to-technologie voor het verhogen van de efficiëntie. Virtualisatie biedt een enorme flexibiliteit, zo kunnen ongelijksoortige architecturen (Linux, MacOS, Windows,...) op één enkele machine tegelijkertijd worden ingezet zonder interferentie (Ameen & Hamo, 2013).

Virtualisatie is een techniek waarin alle fysieke kenmerken van computerbronnen verborgen worden waarvan applicaties, andere systemen en toepassingen gebruik kunnen maken. Het is een softwareabstractielaag tussen de hardware en het besturingssysteem en de toepassingen die erbovenop draaien. Dit wordt een hypervisor of virtual machine monitor(VMM) genoemd. Doordat de hardwarebronnen door de VMM/hypervisor worden gebruikt en niet door de host, is het mogelijk meerdere besturingssystemen te draaien op dezelfde hardware. Al deze verschillende besturingssystemen worden dan virtuele machines (VM's) genoemd. Virtualiteit verschilt alleen van werkelijkheid in de formele wereld, terwijl zij eigenlijk een gelijkaardige essentie of werking bezit. In de computerwereld wordt een virtuele machine gelijkwaardig gesteld aan een gewone machine door

zowel applicaties als andere machines, hoewel de onderliggende mechanismen formeel wel verschillend zijn (Ameen & Hamo, 2013).

Virtuele machines zijn al meer dan 50 jaar in de computergemeenschap, ontwikkeld in 1960 door IBM Corporation. Oorspronkelijk was dit om een grote mainframe computer te distribueren in verschillende instanties. Deze functie werd uitgevonden omdat het onderhoud van de grotere mainframe systemen omslachtig werd. Door het systeem te partitioneren, maakte dit het mogelijk meerdere processen en toepassingen gelijktijdig te kunnen draaien, waardoor de efficiëntie en onderhoudskosten van de omgevingen verminderen. Onder toezicht en leiding van Professor Fernando J. Corbató beter bekend als 'Corby', zijn de ingenieurs en programmeurs begonnen met CTSS, de Compatible Time-Sharing System (Ameen & Hamo, 2013).

De CTSS-supervisor leverde een aantal virtuele machines, allemaal een IBM 7094. Een van deze virtuele machines was de achtergrondmachine en had toegang tot tapedrives(Een tapedrive is een opslag apparaat). De andere virtuele machines waren voorgrondmachines. Deze machines konden normale 7094-machinetaal draaien met een snelheid van 7094 en konden ook een extra instructie uitvoeren, die een groot aantal 'supervisorservices' opriep. Deze 'services' omvatten de terminal-I/O en systeem-I/O. Programma's kunnen worden geschreven voor de voorgrondmachine in elke taal die beschikbaar is voor de 7094. Er werden bibliotheken beschikbaar gesteld zodat compilertalen, zoals MAD, gebruik konden maken van die 'supervisorservices'. CTSS leverde een bestandssysteem dat elke geregistreerde gebruiker een aparte map met bestanden gaf (Van Vleck, 2015).

De belangrijkste kenmerken van CTSS waren de virtuele machines, isolatie van de hardware van gebruikers ten opzichte van de supervisor en van elkaar, en een bestandssysteem per gebruiker. Omdat de virtuele machine dezelfde architectuur en instructieset ondersteunde als de 7094, kon CTSS een groot aantal applicaties ondersteunen die oorspronkelijk waren ontwikkeld voor de FMS-batchomgeving bij MIT en elders (Van Vleck, 2015).

CTSS-software omvatte niet alleen de supervisor, maar ook een reeks voorgrond-opdrachten en subroutine bibliotheken. Veel van deze opdrachten lezen en schrijven bestanden van de gebruikers zijn bestandssysteem. Zo las bijvoorbeeld de MAD-compiler een bestand met de broncode van een programma en schreef hetzelfde bestand met instructies in BSS(block starting symbol) formaat. Het grootste deel van de MAD-opdrachten op de voorgrond waren gelijk aan de MAD-compiler op de achtergrond op de FMS systeemtape, met de toevoeging van een wrapper voor het afhandelen van opdrachtprompt opties en vervangingen voor de invoer- en uitvoerroutines om bestanden te lezen en te schrijven in plaats van tape, en om foutmeldingen naar de terminal van de gebruiker te schrijven (Van Vleck, 2015).

In 1973 laten Srodowa en Bates zien hoe je een virtuele machine kan maken op de IBM OS/360s. Ze tonen het gebruik van de IBM's virtuele machine monitor, de hypervisor om virtuele machines te maken en het efficiënt toewijzen van geheugen, opslag en I/O. Beiden hebben betrekking op onderwerpen die tegenwoordig nog altijd worden bediscussieerd: prestatievermindering, capaciteit, CPU-toewijzing en opslag beveiliging. In 1974 concludeert Goldberg dat het merendeel van de toenmalige computersystemen geen

virtuele machines ondersteunen. De weinige virtuele machines die op dat moment operationeel zijn, bijvoorbeeld CP-67, gebruiken lastige en inadequate technieken vanwege ongeschikte architecturen. Goldberg stelt de Hardware Virtualizer voor, waarin een virtuele machine rechtstreeks zou communiceren met hardware in plaats van door de hostsoftware te gaan. Bijna 30 jaar later zijn brancheanalisten nog altijd enthousiast over de aankondiging van hardware-architecturen die virtuele machines efficiënt kunnen ondersteunen. Zowel AMD als Intel onthulden specificaties voor Pacifica en Vanderpool chiptechnologieën met speciale virtualisatie-ondersteuningsfuncties (Ameen & Hamo, 2013).

De jaren tachtig en begin jaren negentig brachten het 'Distributed computing', wat een techniek is waarbij rekentaken niet door één enkele computer worden uitgevoerd, met zich mee naar de datacenters. Gecentraliseerd computergebruik en virtuele machines werden vervangen door stand-alone servers met specifieke taken: databank, web, applicaties, e-mail. Na aanzienlijke investeringen in gedistribueerde architecturen, is een hernieuwde focus op virtuele machines als aanvullende oplossing voor serverconsolidatieprojecten en datacenter beheerinitiatieven weer opgedoken. Dit betekent dat de virtuele machine op de mainframe gemaakt is. Het is maar onlangs geïntroduceerd op de middenklasse. Technologische vooruitgang in hardware en software maken virtuele machines stabiel, betaalbaar en bieden enorme waarde, mits een juiste implementatie (Daniels, 2009).

Sinds begin 2000 is de opkomst van VMware hun VMware Workstation sterk gestegen. VMware werkt met een Hosted Architecture, dit wil zeggen dat de host een operating system nodig heeft zoals Windows of Linux. Om de complexe mix van prestaties, draagbaarheid, implementatiegemak, enz. te optimaliseren, fungeert het product zowel als een virtuele Machine Monitor(VMM) (rechtstreeks in gesprek met de hardware) als een applicatie die bovenop het hostbesturingssysteem draait. Dit laatste zorgt er voor dat de VMM niet moet omgaan met een groot aantal apparaten beschikbaar op de computers (Singh, 2004).

Het meest essentiële onderdeel van die virtualisatie is de Hypervisor, en wordt ook wel de Virtual Machine Monitor genoemd. De Hypervisor creëert een virtueel platform op de host om daarop meerdere gast besturingssystemen te draaien. Op deze manier kunnen meerdere dezelfde of verschillende exemplaren van een besturingssysteem draaien, en de hardwarebronnen delen die door de host worden aangeboden. Datacenters gebruiken tegenwoordig virtualisatie om abstractie te maken van de fysieke hardware, en deze bronnen aan gebruikers of klanten aan te bieden in de vorm van agile, schaalbare, geconsolideerde virtuele machines. Hoewel de technologie en de gebruiksredenen geëvolueerd zijn, blijft de kernbetekenis van virtualisatie hetzelfde: één host besturingssysteem in staat stellen om meerdere onafhankelijke besturingssystemen tegelijkertijd te laten draaien (Oracle, 2012).

2.1.2 Verschillende typen virtualisatie

Virtualisatie maakt een scheiding van onderliggende hardware mogelijk, wat ervoor zorgt dat je op een hoger niveau bewerkingen kan uitvoeren. Virtualisatietechnologie kan worden onderverdeeld in verschillende soorten virtualisatie:

2.1.2.1 Full virtualisatie

In full virtualisatie is de hardware-interface die wordt aangeboden aan de hypervisor bijna dezelfde als de interface die wordt geboden door het fysieke toestel. Dit betekent dat er voor virtualisatie weinig tot geen wijziging nodig is aan de besturingssystemen en applicaties als deze compatibel zijn met de hardware van het fysieke toestel. Deze virtualisatie kan worden onderverdeeld in zowel Bare-metal virtualisatie als Hosted virtualisatie. Bij bare-metal virtualisatie is er geen hostbesturingssysteem, dit wil zeggen dat de hypervisor rechtstreeks draait op de onderliggende hardware. Hosted virtualisatie wil zeggen dat het hostbesturingssysteem elk algemeen besturingssysteem kan zijn, zoals Windows, zoals Linux, etc. waarop de hypervisor draait. Deze virtualisatie maakt gebruik van een extra softwarelaag om het delen van bestanden tussen de host en het gast besturingssysteem alsnog mogelijk te maken (Kedia e.a., 2013).

2.1.2.2 Paravirtualisatie

Paravirtualisatie is een virtualisatietechniek die een software-interface presenteert aan de virtuele machines, die vergelijkbaar is, maar niet identiek aan de onderliggende hardware/software interface. Het voordeel van deze techniek is dat het de overheadkosten voor virtualisatie vermindert en betere prestaties levert (Kedia e.a., 2013).

2.1.2.3 Operating system virtualisatie

De kernel staat meerdere geïsoleerde besturingssystemen toe. Deze instanties draaien op een bestaand besturingssysteem en geven een reeks bibliotheken weer waarmee applicaties communiceren. Hierdoor krijgen de instanties de illusie dat ze worden uitgevoerd op een machine die speciaal voor het gebruik ervan bedoeld is. Deze virtualisatie staat beter bekend als container virtualisatie, wat later nog aan bod zal komen (Kedia e.a., 2013).

2.1.2.4 Desktop virtualisatie

Desktop virtualisatie is een mechanisme om een PC omgeving met centrale applicaties vanaf een centrale server weer te geven. Het stelt gebruikers in staat applicaties voor verschillende besturingssystemen op een enkele host uit te voeren. Het biedt flexibiliteit om de applicaties en clients te verplaatsen wanneer nodig (Kedia e.a., 2013).

2.1.2.5 Server virtualisatie

Bij server virtualisatie wordt basishardware gevirtualiseerd, waardoor gast besturingssystemen die er op draaien zonder de nood van de gehele operating system kunnen opereren. Servervirtualisatie is het proces waarbij een fysieke server wordt opgedeeld in meerdere unieke en geïsoleerde virtuele servers (Kedia e.a., 2013).

2.1.2.6 Applicatie virtualisatie

Bij deze virtualisatie kan een gebruiker een servertoepassing lokaal draaien met behulp van lokale bronnen zonder de applicatie volledig lokaal te installeren. Het biedt elke gebruiker een geïsoleerde virtuele applicatieomgeving die fungeert als een laag tussen de host en het besturingssysteem. Een voorbeeld van deze soort virtualisatie is een Java Virtual Machine(JVM), aangezien het staat tussen het besturingssysteem en de java-applicatiecode (Kedia e.a., 2013).

2.1.2.7 Storage virtualisatie

Opslag virtualisatie is een techniek waarbij de opslag van gegevens uit een fysieke opslag wordt opgehaald en over het netwerk wordt verspreid. Dit is een vorm van resource-virtualisatie. Dit wordt vaak gebruikt in een Storage Area Network(SAN) (Kedia e.a., 2013).

2.1.2.8 Netwerkvirtualisatie

Netwerkvirtualisatie is het proces waarbij hard- en softwarebronnen gecombineerd worden tot een virtueel netwerk als één enkele verzameling bronnen. Het zorgt voor beter infrastructuurgebruik, zoals het hergebruik van poorten en fysieke bronnen, voor meerdere andere netwerkbronnen zoals hosts, virtuele machines, routers, switches etc. Zo helpt het ook bij verlagen van de kosten door netwerkbronnen te delen (Kedia e.a., 2013).

2.1.2.9 Resource virtualisatie

Resource virtualisatie wordt beschouwd als opslagvolumes, namespaces en de netwerkbronnen in een gevirtualiseerd systeem. Alle onderdelen kunnen samen een grotere pool van bronnen worden en zo kan één enkele bron, zoals schrijfruimte, ook worden opgedeeld in een aantal kleinere en gemakkelijk toegankelijke bronnen (Kedia e.a., 2013).

2.1.3 Container virtualisatie

Containergebaseerde virtualisatie is een lichtere benadering van virtualisatie om met behulp van de host-kernel meerdere virtuele omgevingen op te zetten. Deze virtuele omgevingen worden meestal containers genoemd. Containergebaseerde virtualisatie virtualiseert op het besturingssysteem niveau, en laat toe om meerdere applicaties op te zetten

zonder redundante host-kernels van andere machines. De containers zien eruit als normale processen van de buitenkant, die bovenop de kernel draaien die gedeeld worden met de host. Ze bieden een geïsoleerde omgeving met de nodige middelen om applicaties uit te voeren. Deze middelen worden gedeeld met de host of afzonderlijk per container ter beschikking gesteld (Bui, 2015).

2.1.3.1 Verschil tussen containers en virtuele machines

Containergebaseerde virtualisatie zorgt voor sommige voordelen tegenover een hypervisor-gebaseerde virtualisatie. Ten eerst zorgt een containergebaseerde virtualisatie voor een grotere dichtheid van virtuele omgevingen. Dit komt omdat een container geen volledig besturingssysteem heeft, de grootte en nodige middelen zijn ook minder om een applicatie in een container te draaien. Met als resultaat dat meerdere containergebaseerde virtuele machines opgezet kunnen worden op dezelfde machines t.o.v. traditionele virtuele machines. Ten tweede zorgen containers voor een betere performance (Ruiz e.a., 2015).

De performance van containergebaseerde virtualisatie is in de meeste gevallen beter dan met een hypervisor-gebaseerde virtualisatie. Desondanks het genoemde voordeel is containergebaseerde virtualisatie niet in staat om een verschil in virtuele omgevingen te ondersteunen, zoals een hypervisor dat wel kan. De oorzaak hiervan is, dat containers hetzelfde type moeten zijn als de host. Zo kan bijvoorbeeld een Windows host niet draaien op een Linux host. Aan de hand van een hypervisor-gebaseerde virtuele machine kan dat wel, waardoor je zowel Windows als Linux containers op een Linux host kan draaien (Ruiz e.a., 2015).

2.1.3.2 Docker

Docker is een source-platform waarop applicaties worden uitgevoerd en het proces gemakkelijker te ontwikkelen en distribueren is. De applicaties die in Docker gebouwd zijn, worden met alle ondersteunende afhankelijkheden verpakt in een zogenaamde container. De containers blijven op een geïsoleerde manier draaien bovenop de kernel van het besturingssysteem. De extra abstractielaag kan invloed hebben op de prestaties. Hoewel containervirtualisatie al meer dan 10 jaar bestaat, is Docker, een relatief nieuwe technologie, zonder twijfel één van de beste containerisatietools omdat het capaciteiten heeft dat voorgaande technologieën niet hadden (Babak Bashari Rad, 2017).

In eerste instantie biedt Docker de mogelijkheid om containers te maken en te controleren. Applicaties kunnen dan makkelijk in deze containers gestoken worden. Daarnaast kunnen deze gevirtualiseerde applicaties makkelijk overal worden gebruikt zonder enige wijziging. Tenslotte kan Docker gemakkelijk samenwerken met externe programma's die helpen bij het opstellen en beheren van Docker containers (Babak Bashari Rad, 2017).

2.1.3.3 Alternatief voor Docker

Hoewel Docker één van de populairste containertechnologieën is, zijn er andere technologieën die Docker voorafgegaan zijn, die ontstaan zijn op hetzelfde moment of meer recentelijk zijn geïntroduceerd. Het basisconcept van de programma's zijn in grote lijnen gelijk en dan komt het vaak aan op technische verschillen. Voorbeelden van andere goede containertechnologieën zijn RKT(Rocket, CoreOS), LXD(Lexdi, Canoncical Ltd.), Linux VServer en Windows Containers. Daar dit onderzoek gaat over monitoring zal hier verder geen vergelijking plaatsvinden.

2.1.3.4 Kubernetes

Kubernetes is een draagbaar, uitbreidbaar, opensource platform voor het beheren van gecontaineriseerde workloads en services, dat zowel manuele configuratie als automatisering mogelijk maakt (Kubernetes, 2021).

De kracht van Kubernetes is het uitvoeren van workloads op meerdere servers. Het is één van de beste tools als het gaat om containerclusterbeheer. Het voert de juiste containers op het juiste moment uit, schaalt ze op en neer op basis van belasting, lost hardware- of containerstoringen op en beheert netwerken en opslag (Rancher, 2020).

Origineel ontwikkeld door Google, geïnspireerd door een decennia van kennis door het uitrollen van schaalbare en betrouwbare systemen in containers door applicatie georiënteerde APIs. Sinds de introductie in 2014 is Kubernetes één van de grootste en meest populaire open source projecten in de wereld. Het is de standaard API geworden voor het bouwen van cloud-native applicaties, present in bijna elke publieke cloud. Kubernetes is een bewezen infrastructuur voor gedistribueerde systemen dat geschikt is voor cloud-native ontwikkelaars van alle schalen, zo kan het gaan over een cluster van een aantal Raspberry Pi's tot een warenhuis met de nieuwste computers. Het zorgt voor de nodige software om succesvol betrouwbare en schaalbare systemen op te zetten (Brendan Burns, 2019).

2.1.3.5 Alternatief voor Kubernetes

Hoewel er veel voordelen zijn voor het gebruik van Kubernetes, denk maar aan documentatie en support van de community, is het niet mis om de alternatieven te bekijken. Amazon Elastic Container Service, Mirantis Kubernetes Engine (Docker Enterprise) zijn hier goede voorbeelden van. Daar dit onderzoek gaat over monitoring zal hier verder geen vergelijking plaatsvinden. Maar zoals eerder vermeld (2.1.3.3) gaat dit onderzoek niet verder gaan in het vergelijken van deze applicaties.

2.1.3.6 Samenwerking tussen Docker en Kubernetes

In een Docker-cluster omgeving moeten enkele taken worden uitgevoerd, zoals plannen van deployments, communicatie en schaalbaarheid. Deze taken worden daarom uitgevoerd door orkestratietools, met als groot voorbeeld, Kubernetes. Kubernetes is één van de meest populaire projecten in de open source-geschiedenis (Journal, 2020).

Docker helpt bij het maken van containers, en Kubernetes laat toe dat de gebruiker ze tijdens runtime kan beheren. Docker wordt gebruikt voor het verpakken en verzenden van de applicatie en Kubernetes om de applicatie te implementeren en te schalen. Wanneer er gebruik gemaakt wordt van een klein aantal containers is het gebruik van Kubernetes nog niet nodig, maar vanaf het aantal containers begint op te schalen is het wel aan te raden. Dit is ook waar Kubernetes in uitblinkt (Journal, 2020).

Wanneer ze samen worden gebruikt, dienen Docker en Kubernetes voor het inschakelen van digitale transformatie en als tool voor moderne cloudarchitectuur. Het gebruik van beide was de nieuwe norm in de branche en zorgde voor een snellere implementatie en release van applicaties (Journal, 2020).

Ongelukkigerwijze, is sinds december 2020 aangekondigd dat de samenwerking tussen Docker en Kubernetes zal stopgezet worden. Docker is nog steeds een handig hulpmiddel voor het bouwen van containers, en de docker images kunnen nog steeds in een Kubernetes-cluster worden uitgevoerd. Dockershim, een tijdelijke oplossing om ondersteuning van Docker toe te voegen aan Kubernetes, zal in Kubelet, versie 1.23, verwijderd worden en hierdoor ook niet meer ondersteund worden door Kubernetes. Docker is niet comptabiliteit met het gebruik van de ”Container Runtime Interface” en was daarom afhankelijk van Dockershim. De oplossing voor dit probleem is simpelweg overschakelen van Docker naar een andere ondersteunde containerruntime (Jorge Castro, 2020).

2.2 Monitoring

Monitoring is het proces om metrische gegevens of logs te verzamelen over de werking van hardware en software van een IT-infrastructuur. Aan de hand van deze gegevensverzameling wordt er voor gezorgd dat alles functioneert zoals verwacht zodat applicaties en services kunnen blijven ondersteunen. Basis monitoring wordt uitgevoerd door controle uit te voeren over de werking van een apparaat, terwijl meer geavanceerde monitoring gedetailleerde overzichten geven over de operationele statussen, inclusief gemiddelde reactie tijden, aantal applicatie-instanties, foutboodschappen, CPU-gebruik en applicatie beschikbaarheid (TechTarget, 2020).

IT monitoring verspreidt zich over 3 secties:
Basis. De infrastructuur is de onderste laag van een softwarestack en omvat zowel fysieke als virtuele apparaten, zoals servers, CPU’s en VM’s.
Software. Dit deel wordt ook wel de bewakingssectie genoemd en analyseert wat er op de apparaten in de basis gebeurd, inclusief CPU-gebruik, geheugen gebruik, load op de

servers enzovoort.
Interpretatie. Verzameld statistieken die worden gepresenteerd via grafieken of diagrammen, vaak op een GUI-dashboard. Dit wordt vaak bereikt door integratie met tools die specifiek gericht zijn op datavisualisatie.

Monitoring kan met of zonder 'agents'. Een agent is een onafhankelijk programma die op een apparaat wordt geïnstalleerd om gegevens over hardware of software, prestatiegegevens te verzameld en rapporteert aan een monitoringserver. Wanneer er geen agent is, zal de monitoringserver gebruik maken van communicatieprotocollen om een agent te emuleren met ongeveer dezelfde functionaliteiten om zo gegevens te verzamelen (TechTarget, 2020).

Er zijn twee soorten monitoring wanneer het over de frequentie gaat. Enerzijds is er 'Real Time Monitoring' wat een techniek is waarbij systemen continu gegevens verzamelen en toegankelijk zijn voor IT-teams om de actieve en lopende status te bepalen. Metingen van real-time monitoringssoftware geven gegevens weer uit de huidige IT-omgeving, evenals het recente verleden, waardoor IT-personeel snel kunnen reageren op actuele gebeurtenissen. Door data op deze manier te monitoren, kan het IT-personeel de infrastructuur verbeteren of mogelijke complicaties identificeren voordat ze zich voordoen, omdat ze een patroon of trend in de gegevens herkennen.

Anderzijds is er ook trend monitoring, een analyse van één specifieke gebeurtenis op een bepaald moment. Dit kan gebruikt worden om een probleem te identificeren dat onmiddellijk moet worden verholpen, zoals een 100% volle schijf of overmatig gebruik van de CPU (TechTarget, 2020).

2.2.1 Monitoring binnen een container environment

De verandering van monitoring binnen een containeromgeving ten opzichte van een hypervisor omgeving kan uitgelegd worden in 5 punten:

- De korte duur van containers
- De snelle groei van objecten, services en statistieken om te loggen
- Services zijn het nieuwe focuspunt van monitoring
- Een meer diverse groep van monitoring-eindgebruikers
- Nieuwe gedachten resulteren in nieuwe methoden

2.2.1.1 Korte duur en schaal van containers

De tijdelijke aard van containers en instanties van virtuele machines zorgt voor uitdagingen op het gebied van tracking. Omdat een container zeer dynamisch is, vereist dit nieuwe monitoringmethoden om observaties te doen over de status van die container. Door de korte duur van containers en de snelgroeiende schaal heeft het weinig zin om container per container te bekijken, in plaats daarvan wordt aangeraden om clusters (meerdere containers) te volgen/monitoren. Traditionele benaderingen van monitoring zijn gebaseerd op het introduceren van datacollectors, genaamd 'agents' of 'remote access hooks' in de

systemen zelf. Deze kunnen de snelheid van de provisioning en het dynamisch opschalen echter niet aan. Je kan op twee manieren data opvangen van een server, enerzijds door te 'pollen' en anderzijds door te 'pushen'. De tweede aanpak heeft een voorkeur voor een containeromgeving, daar deze containers van korte duur zijn en omdat het te lang kan duren om deze containers in de eerste plaats te vinden en dan te pollen. Om te kunnen pushen wordt een agent geïnstalleerd, wat het 'pol'-proces elimineert (Williams, 2016).

2.2.1.2 Snelle groei van objecten, services en statistieken

De explosie van gegenereerde gegevens is een bekend fenomeen. Vroeger vroegen mensen zich af hoe ze al die gegevens moeten opslaan, tegenwoordig ligt de nadruk meer op hoe die gegevens gebruikt kunnen worden zonder ze op te slaan. Door de opkomst van Internet of Things(IoT)-sensoren en de introductie van containers, zijn er nu meer objecten dan ooit om te monitoren. De overvloed aan datapunten, metrieken en objecten die moeten worden gevolgd, is een ernstig probleem. Het streamen van gegevens biedt veel mogelijkheden voor realtime analyses, maar het moet wel nog worden verwerkt en opgeslagen. Er zijn technische oplossingen die de schaal aankunnen, maar tegen aanzienlijke kosten voor zowel financiën als prestaties. Hoewel er wel databases van de volgende generatie zijn(NoSql), zijn ze niet geoptimaliseerd voor deze use case. Loggegevens kunnen echter niet voor onbepaalde tijd bewaard worden. Sommige oudere bestanden worden nooit gebruikt, wat gebruikers motiveert om eerder te focussen op metrische gegevens in plaats van logs. Metrische gegevens zijn gegevens die in geaggregeerde vorm of op regelmatige tijdstippen worden verzameld (Williams, 2016).

2.2.1.3 Services zijn het nieuwe aandachtspunt

Met een hernieuwde focus op wat er daadwerkelijk gemonitord moet worden, zijn er drie aandachtsgebieden: de gezondheid van containerclusters, microservices en toepassingen. Het beoordelen van clusters van containers - in plaats van afzonderlijke containers - is een betere manier voor infrastructuurbeheerders om de impact van services te begrijpen. Hoewel het waar is dat applicatiebeheerders individuele containers kunnen uitschakelen en herstarten, zijn ze meer geïnteresseerd in het begrijpen welke clusters gezond zijn. Met deze informatie kunnen ze het cluster inzetten op een andere infrastructuur of extra bronnen toevoegen om de optimale werking ervan te ondersteunen. Containerorkestratieoplossingen helpen door een efficiënte planning van containers op clusters van hosts mogelijk te maken. Volgens Dynatrace Cloud Technology Lead Alois Mayr, in een interview met The New Stack: "als je kijkt naar applicatiebewaking, ben je vooral geïnteresseerd in de services die in containers worden uitgevoerd, in plaats van in de containers zelf" (Williams, 2016).

2.2.1.4 Een meer diverse groep van monitoring-eindgebruikers

De focus op het monitoren van applicaties in plaats van alleen op infrastructuur gebeurt om twee redenen. Allereerst is er een nieuwe groep mensen bij de monitoring betrokken. Ten tweede zijn applicaties relevanter voor de algehele bedrijfsprestaties.

Ondanks de vooruitgang van de afgelopen jaren is de monitoring over het algemeen nog steeds reactief. Het is gericht op de doelstellingen van het IT-team dat de feitelijke infrastructuur beheert. Deze mentaliteit bewijst ontwikkelaars een slechte dienst, omdat ze over het algemeen gegevens uit de tweede hand ontvangen. Ontwikkelaars worden steeds vaker aangesproken op applicaties nadat ze in productie zijn genomen. Zoals de ”Effective Performance Engineering” van Todd DeCapua en Shane Evans opmerkt, wordt ontwikkelaars gevraagd om ”producten van de hoogste kwaliteit te leveren en continue feedback en optimalisatieaanbevelingen te geven, zodat andere teams snel en volledig geautomatiseerd kunnen opleveren.” De DevOps-beweging is gestegen, althans gedeeltelijk, als reactie op het verlangen van ontwikkelaars naar meer zichtbaarheid gedurende de volledige levenscyclus van een applicatie. Nu zijn DevOps-rollen vaak de volledige stackbeheerders en operators van applicaties.

Samen met DevOps zal de praktijk van ”site reliability engineering (SRE)” van invloed zijn op het gebruik van monitoringtools. Vanuit dit perspectief zal de monitoring nog steeds grotendeels worden beheerd door een operationeel team, maar de verantwoordelijkheid om ervoor te zorgen dat nieuwe applicaties en services worden gecontroleerd, kan worden gedelegeerd aan applicatieontwikkelaars. Shariq Rizvi, mede-oprichter van Netsil, zei in een interview met The New Stack dat SRE’s en DevOps-ingenieurs anders zijn dan software-ingenieurs. Hij vindt dat SRE-teams het beheer van diensten moeten opsplitsen, waardoor er meer specialisatie ontstaat. Dan Turchin, mede-oprichter en ”chief product officer” van Neva, zei in een interview met The New Stack dat hij gelooft dat DevOps-functies de technici van het ”Network Operations Center (NOC)” vervangen, die traditioneel vanuit het perspectief van een datacenter naar de dingen keken. Als de ouderwetse netwerkstatistieken worden verdrongen door statistieken van de cloudinfrastructuur, kan dit waar zijn (Williams, 2016).

2.2.1.5 Nieuwe gedachten resulteren in nieuwe methoden

Hoewel monitoring verandert om aan de behoeften van verschillende functies te voldoen, gaat het ook over een meer alomvattende benadering. Charity (2016) vertelt dat in plaats van te vertrouwen op een vaste reeks vragen en controles, mensen zouden moeten evolueren naar de observeerbaarheid van systemen. Dit moet gebeuren omdat die vaste datapunten, alleen, niet de benodigde inzichten opleveren. Er zijn nieuwe tools nodig om gelijke tred te houden en de mogelijkheid te bieden om te voorspellen wat er gaat breken. Veel van deze tools maken gebruik van artificiële intelligentie-technieken.

Observeerbaarheid wil niet zeggen dat testen het probleem altijd identificeert. Charity is dan ook van mening dat instrumentatie net zo belangrijk is als testen. Als je complexe systemen gebruikt, kun je niet alles in je hoofd modelleren. Naast het veranderen van

de instrumentatie, stelt ze ook voor om te focussen op het begrijpbaar maken van de monitoringssystemen. Dit betekent feitelijk definiëren wat de gegevens vertegenwoordigen en dezelfde definities gebruiken als uw collega's, zowel binnen als buiten de organisatie. Verder is er ook frustratie over de noodzaak om door meerdere, statische dashboards te scrollen. Als reactie hierop maken ontwikkelaars meer intuïtieve, interactieve dashboards en maken de ontwikkelaars zelf gebruik van artificiële intelligentie om voor elke server te bepalen welke informatie wanneer wordt weergegeven (Williams, 2016).

2.2.2 Monitoring binnen een containeromgeving

Om ongelijksoortige metrische bronnen samen te brengen, is een robuust platform voor infrastructuurmonitoring vereist dat een enkel dashboard kan bieden om gegevens uit verschillende bronnen te bekijken. Het vereist ook veel aandacht en planning om ervoor te zorgen dat gegevens kunnen correleren om eenvoudige end-to-end foutopsporing mogelijk te maken.

Conceptueel worden gecontaineriseerde applicaties op dezelfde manier gemonitord als traditionele applicaties. Het vereist gegevens op verschillende lagen in de stacks voor verschillende doeleinden. Het meet en verzamelt metrische gegevens zowel op de container zelf als op de infrastructuurlaag voor een beter resourcebeheer. Er zijn ook toepassingsspecifieke gegevens, prestatiebeheer van toepassingen en traceringsinformatie nodig voor het oplossen van problemen met toepassingen.

Robuuste observatie en eenvoudige foutopsporing zijn de sleutel voor het bouwen van een microservices-architectuur. Naarmate uw systeem groeit, neemt ook de complexiteit van het bewaken van de gehele applicatie toe. Traditioneel wordt een onafhankelijke dienst gebouwd die maar één doel dient, en die communiceert met andere diensten om een grotere taak uit te voeren en dit allemaal naast bedrijfslogica. Deze taken worden echter erg complex als je honderden containers hebt die verschillende programmeertalen in verschillende teams gebruiken. Dit is waar service-mesh in beeld komt.

Een servicemesh beheert de communicatielaag van uw microservicesysteem door een softwarecomponent als een sidecar proxy (een ontwerp dat bepaalde functies zoals communicatie tussen services, monitoring en beveiliging abstraheert om het volgen en onderhouden van de applicatie als geheel vergemakkelijkt (Gilles, 2019)) te implementeren. Dit softwarecomponent voert netwerkfuncties, verkeerscontrole, tracering en logboekregistratie uit. Elke service heeft zijn eigen proxyservice en alle proxyservices samen vormen de mesh. Elke service communiceert niet rechtstreeks met andere services, maar alleen met zijn eigen proxy. De service-to-service-communicatie vindt plaats tussen proxy's.

Met een servicemesh kan een netwerk van microservices gebouwd worden en het grootste deel van de netwerklaag gescheiden worden om het beheer en de waarneembaarheid te stroomlijnen. Als er zich problemen voordoen, kan de oorzaak van het probleem snel achterhaald worden (AWS, 2021).

2.3 Monitoring in het lessenpakket

Dit onderdeel van de literatuurstudie moet duidelijkheid scheppen waarom monitoring moet worden geïntroduceerd voor de studenten van toegepaste informatica. Hoewel dit onderzoek vooral focus legt op monitoring in een containeromgeving, zal hier toch de nadruk gelegd worden op monitoring in het geheel. Daar dit eigenlijk zo goed als overeen komt.

2.3.1 Het belang van monitoring

Quality of service (QoS) is zo goed als de infrastructuur waaronder service wordt geleverd. Helaas besteden veel IT-afdelingen meer tijd aan het reageren op onverwachte problemen dan aan het voorkomen ervan. Het bewaken van de gehele IT-omgeving is een bedrijfskritische taak die zorgt voor optimale prestaties en bedrijfscontinuïteit (Matrix, 2016).

Monitoring is cruciaal omdat problemen niet altijd direct duidelijk zijn, maar de neiging hebben om zich in de loop van de tijd op te bouwen. De rol van IT als een belangrijke motor van operationele prestaties en innovatie maakt het een verplichting van netwerkbeheerders om een zeer responsief controlesysteem te implementeren om alle netwerkactiviteiten bij te houden (Matrix, 2016).

Er zijn een aantal belangrijke redenen waarom het hebben van een volledig zicht op de IT-infrastructuur belangrijk is:

2.3.1.1 Volledige controle over IT-middelen

Het IT-netwerk is het hart van moderne bedrijven. Om de prestaties van hardware- en softwarebronnen bij te houden, moet de IT uit de eerste hand kennis hebben van de gezondheid van het netwerk en volledige controle hebben over trends in het gebruik van bronnen, servercapaciteiten en kwaadwillende activiteiten om problemen te voorkomen voordat ze zich voordoen (Matrix, 2016).

2.3.1.2 Geïnformeerde besluiten

IT-monitoring is meer dan alleen het bijhouden van infrastructuuractiviteiten. Het is een essentieel onderdeel van het creëren van een algemene IT-strategie voor bedrijven. Van CIO's (chief information officer) wordt verwacht dat ze aantonen dat ze de controle hebben over de IT-omgeving en dat ze toegang hebben tot kritieke gegevens voor belangrijke en onbelangrijke besluiten. Efficiënte monitoring wordt geleverd met effectieve rapportage om ervoor te zorgen dat alle belanghebbenden worden geïnformeerd en dat aan de belangrijkste prestatie-indicatoren wordt voldaan (Matrix, 2016).

2.3.1.3 Verhoogde productiviteit en verbeterde servicebeschikbaarheid

Wanneer de prestaties worden verbeterd, wordt de productiviteit ook verhoogd. Het is belangrijk voor IT-teams om de interacties van de verschillende systemen binnen het netwerk nauwlettend in de gaten te houden om gebieden te identificeren die verbetering behoeven. Kwetsbare gebieden hebben speciale aandacht nodig om te anticiperen op mogelijke problemen en om preventieve oplossingen te bieden.

Serviceonderbreking als gevolg van downtime is een vloek voor eindgebruikers. Een netwerk vertraging kan bijvoorbeeld veroorzaakt worden door eindgebruikers die te veel bronnen tegelijk willen gebruiken. Regelmatig en willekeurig controleren op de gebruikspatronen kan ervoor zorgen dat een evenwichtige werklastverdeling kan worden gecreëerd om een continue beschikbaarheid van de service te garanderen (Matrix, 2016).

2.3.1.4 Probleemdetectie voordat zich een ramp voordoet

Onvoorspelbare catastrofes worden veroorzaakt door de natuur en zijn vaak niet te voorkomen, maar door de mens veroorzaakte rampen, of ze per ongeluk of opzettelijk zijn is bijzaak, kunnen worden voorkomen door een robuust monitoringsysteem. Storingspatronen, frequentie van downtime, ontbrekende, verwijderde of toegevoegde bestanden, evenals bestanden en programma's die niet openen met legitieme wachtwoorden, zijn waarschuwingssignalen die moeten worden gecontroleerd en opgelost om een vermijdbare ramp te voorkomen(Matrix, 2016).

2.3.1.5 Besparingen en hogere winst

Kosten besparen en winst verhogen zijn legitieme zakelijke doelstellingen. Preventie door monitoring betekent dat er minder geld wordt uitgegeven in vergelijking met de kosten van oplossingen voor problemen die zich plotseling kunnen voordoen. Proactieve monitoring van infrastructuur helpt financiële verliezen te voorkomen als gevolg van niet-gedetecteerde uitval van systemen als gevolg van slechte (of een gebrek aan) monitoring (Matrix, 2016).

2.3.2 Het nut van container monitoring

Hoewel containers vluchtig zijn, is inzicht in metrische gegevens, logboeken en traceringen alleen maar voordelig voor operatoren van deze gecontaineriseerde platformen. Door te begrijpen wat er gebeurt, en niet alleen op cluster- of hostniveau, maar ook binnen de containerruntime en -toepassing, kunnen organisaties beter geïnformeerde beslissingen nemen, zoals wanneer instanties, taken of pods moeten worden geschaald in grootte. DevOps of systeemingenieurs kunnen ook de efficiëntie en snelheidsresolutie verbeteren door automatisering toe te voegen, zoals het gebruik van automatische waarschuwingen om het schalen van containers dynamisch te activeren. Door bijvoorbeeld het geheugengebruik actief te bewaken, kan er een drempelwaarde geïdentificeerd worden en operators op de hoogte gesteld worden wanneer het resourceverbruik de resourcelimiet bereikt heeft, of

door automatisch extra nodes toe te voegen vóór dat de beschikbare CPU of RAM uitgeput is (AWS, 2021).

2.3.3 Conclusie

Zoals beschreven in de bovenstaande alinea's is het monitoren van een containeromgeving en een gehele it omgeving alleen maar voordelig. Door middel van monitoring kunnen veel fouten of problemen voorkomen worden waardoor kosten en moeite gespaard blijven. Echter wordt monitoring vaak over het hoofd gezien omdat het voor bedrijven een kost is zonder opbrengst, wat ongetwijfeld niet waar is. Daarom is het belangrijk om het concept van monitoring vroeg aan te leren om monitoring in een IT-omgeving te standaardiseren. Met de opkomst van containers is dit onderzoek zeker op zijn plaats.

2.4 Mogelijke tools om te monitoren in een containeromgeving

Hoewel deze studie focus legt op de metrische gegevens van containers, zullen er in deze opsomming ook tools staan die worden gebruikt voor het verzamelen en analyseren van logs. In het volgende hoofdstuk zal dan de vergelijking van de tools plaatsvinden om zo de, in dit onderzoek, minder interessante tools te filteren.

2.4.1 Docker Stats

Op een basis level kan je aan de hand van een ingebouwd commando, Docker containers monitoren. Er kan gedetailleerde, realtime informatie verkregen worden over het gebruik van de containers. Deze informatie bestaat uit, CPU gebruik, geheugen gebruik, schijf -en netwerk-I/O en het aantal actieve processen. "Docker Stats" bewaart geen geschiedenis en kan maar 1 host per keer monitoren. Aan de hand van scripts kunnen wel gegevens verzameld worden van meerdere host en zo bewaard worden. Op zich is de informatie die verzameld wordt minimaal maar door een combinatie met "Docker logs" en "Docker events" kan de monitoring naar een hoger niveau getild worden (Sissons, 2021).

2.4.2 cAdvisor

cAdvisor is een open-source project, oorspronkelijk ontwikkeld door Google, dat net als "Docker Stats" de gebruikers informatie biedt over gebruik van bronnen tijdens het uitvoeren van containers. Net zoals "Docker Stats" is het ook geïmplementeerd als een daemon-proces. Het verzamelt, aggregeert, verwerkt en exporteert informatie over de actieve containers.

Deze tool heeft een webinterface en kan meerdere grafieken genereren. Maar net zoals "Docker Stats" kan cAdvisor maar één host monitoren. Indien de verzamelde informatie moet worden opgeslagen, moet ook een externe database geconfigureerd worden zo-

als Prometheus en InfluxDB. cAdvisor is op zich geen complete monitoringoplossing en wordt vaak in combinatie gebruikt met andere monitoringtools (Sissons, 2021).

2.4.3 Scout

Scout is een monitoringapplicatie die een monitoringservice biedt op cloudgebasseerde applicaties en cloudgebasseerde databanken. Scout biedt een uitgebreide functionaliteit voor het verzamelen, filteren en monitoren van gegevens met flexibele alerts en integraties met alerting-services van derden. Het team van Scout geeft documentatie voor het schrijven van scripts om gebruik te maken van de Docker Stats API (zoals hierboven vermeld) en geeft metrische gegevens door aan Scout voor monitoring. Er is een Docker-Scout-container beschikbaar op Docker Hub, die het installeren en configureren een stuk eenvoudiger maakt (Sissons, 2021).

2.4.4 Pingdom

Net zoals Scout, is Pingdom ook een cloudgebasseerde applicatie, beheerd door SolarWinds, een bedrijf dat zich richt op het monitoren van IT-infrastructuur. Bij Pingdom is belangrijkste domein het monitoren van websites, maar deze tool kan ook gebruikt worden voor servermonitoring met maar liefst 90 plugins. Pingdom onderhoudt docker-scout, dit is dezelfde StatsD-agent die door Scout gebruikt wordt (Sissons, 2021).

2.4.5 Datadog

Datadog is net zoals de twee vorige tools een cloudmonitoring service. Datadog biedt ook een container-agent voor installatie op een Docker-host. In plaats van generieke StatsD te gebruiken zoals de eerder genoemde cloudbewakingsoplossingen, heeft Datadog een verbeterde StatsD ontwikkeld, DogStatsD genaamd.

De Datadog-agent verzamelt de volledige set metrische gegevens die beschikbaar zijn via de Docker-API en geeft deze door, waardoor een zeer gedetailleerde monitoring wordt geboden. Datadog biedt een goede toegang tot metrische gegevens en meer mogelijkheden bij het definiëren van waarschuwingsvoorwaarden dan de eerder genoemde cloudservices. Net als de andere services kan Datadog ook worden gebruikt om andere services en applicaties te monitoren, en beschikt het over een bibliotheek met meer dan 200 plugins. Datadog bewaart ook gegevens gedurende 18 maanden, wat langer is dan de bovenstaande cloudservices (Sissons, 2021).

2.4.6 Sysdig

Sysdig is voornamelijk een cloudgebasseerde monitoringoplossing, gericht op containers. Een deel van zijn functionaliteiten zijn beschikbaar in open-source projecten, en Sysdig biedt ook om de applicatie lokaal te installeren (Sissons, 2021).

2.4.7 Prometheus

Prometheus is een populaire, open-source tool voor monitoring en alerting, oorspronkelijk gebouwd door de developers bij SoundCloud. Waar Prometheus verschilt tegenover de cloudgebasseerde tools, is dat de tool modulair is en zelf wordt gehost, wat betekent dat gebruikers de tool op hun clusters inzetten. Prometheus installeert automatisch op elke Docker-host een agent en aan de hand van een 'pull' protocol haalt het zo alle gegevens binnen via HTTP bij een reeks exporteurs. Sommige van deze exporteurs worden officieel onderhouden door Prometheus zelf, terwijl de overschot externe bijdragen zijn. Prometheus biedt functies zoals service-detectie, een aparte push-gateway voor specifieke typen metrieken, en heeft een ingebouwde querytaal (PromQL) die dient voor het opvragen van multidimensionale gegevens. Het heeft ook een uitgebreide ingebouwde webinterface en API. Deze GUI is wel sterk afhankelijk van de gebruikers die kennis bezitten van PromQL. Daarom ondersteunt Prometheus ook tools van derden om als dashboard te gebruiken, zoals Grafana. (Sissons, 2021).

2.4.8 ELK/EFK Stack

De ELK/EFK-stack is een open-source software stack die het monitoren van containers, virtuele machine, alsook klassieke servers op zich neemt. Deze stack bestaat uit drie open-source projecten die zijn samengevoegd om de ELK/EFK stack te bekomen. De stack is veelzijdig en wordt veel gebruikt voor een verscheidenheid aan analytische toepassingen, waarbij het monitoren van logbestanden een belangrijke rol speelt. ELK/EFK is genoemd naar drie componenten.

- Elasticsearch: wordt gebruikt om de logs op te slaan
- Logstash/Fluentd: wordt gebruikt voor zowel het transporteren als het verwerken en opslaan van logs, die dan doorgestuurd worden naar Elasticsearch
- Kibana: visualisatietool gehost door Nginx of Apache die de data van Elasticsearch gebruikt

Alhoewel de gelijkenissen tussen Logstash en Fluentd groter is dan de verschillen, gebruiken de meeste developers Fluentd. Maar zelfs gebruikers van Logstash staan voor op de curve als het aankomt op log management (Sissons, 2021).

2.4.9 Sensu

Sensu is een self-hosted monitoringtool voor algemene doeleinden die een verscheidenheid aan monitoringstoepassingen ondersteunt. Sensu maakt gebruikt van Agents die de nodige containers monitoren. Het aantal Sensu-plug-ins blijft groeien en er zijn tientallen door Sensu en community ondersteunde plug-ins waarmee metrische gegevens uit verschillende bronnen kunnen worden opgehaald. Sensu heeft vele features, maar een nadeel voor de gebruikers is dat het framework moeilijk te installeren, configureren en onderhouden is omdat de componenten zelf niet "Dockerized" zijn (Sissons, 2021).

2.4.10 Zabbix

Zabbix is software die talloze parameters van een netwerk, de gezondheid en integriteit van servers, virtuele machines, applicaties, services, databases, websites, de cloud en meer bewaakt. Zabbix maakt gebruik van een flexibel meldingsmechanisme waarmee gebruikers op e-mail gebaseerde waarschuwingen voor vrijwel elk evenement kunnen configureren. Hierdoor kan snel worden gereageerd op serverproblemen. Zabbix biedt uitstekende rapportage- en gegevensvisualisatiefuncties op basis van de opgeslagen gegevens (Zabbix, g.d.-b).

3. Methodologie

In dit hoofdstuk werden alle tools uit hoofdstuk 2.4 vergeleken met elkaar om zo een best passende tool voor het onderzoek te vinden. Aan de hand van een ”MoSCoW”-analyse werd bepaald wat de vereisten waren. Een deel van de vereisten zijn bepaald aan de hand van voorgaand onderzoek (Cedric, 2019). Het andere deel werd bepaald aan de hand van de bachelorproef opdracht en communicatie met de (co)promotor.

- Must have: alle noodzakelijke eigenschappen
- Should have: eigenschappen die niet verplicht zijn, maar wel een meerwaarde zijn
- Could have: eigenschappen die een kleine meerwaarde hebben
- Would have: alles wat een meerwaarde kan hebben maar niet nodig is voor dit onderzoek

Tenslotte werd de opstelling van het onderzoek uitgelegd. Aan de hand van die opstelling werd een checklist opgemaakt en in de conclusie een antwoord geformuleerd op de onderzoeksvraag.

3.1 Requirements-analyse

3.1.1 Must have

- De tool moet gratis zijn of de trial moet lang genoeg duren.
- De tool moet voor een containeromgeving zijn.
- De volledige PoC moet lokaal geïnstalleerd zijn.
- De PoC moet mogelijk zijn op de laptop van de studenten die voldoet aan de opgelegde specificaties van HoGent.
- Via de Proof-Of-Concept moeten de functies van de monitoringtool goed genoeg zijn om de I/O metrics te analyseren.

3.1.2 Should have

- Aan de hand van één bestand kan de Proof-of-Concept opgezet worden zodat de studenten direct aan de slag kunnen met de monitoringtool.
- Installatie en configuratie moet van een gemiddeld niveau zijn.

3.1.3 Could have

- Opzetten van alerts.

3.1.4 Would have

- Het verzamelen en analyseren van logs.

3.2 Onderzoeksmethodologie

Eerder vermeld in dit onderzoek, luidt de volgende onderzoeksvraag:

- Welke monitoringtool voor een Kubernetes/Docker environment is het meest geschikt voor de studenten toegepaste informatica om zo een goede basiskennis te bekomen.

Eveneens zijn er nog een aantal deelvragen die ook belangrijk zijn:

- Wat zijn de belangrijkste verschillen tussen de tools?
- Waar moeten de tools inzicht op geven?
- Welk proof-of-concept is voldoende om de tool aan te leren?

Om een antwoord te formuleren op de vragen wordt een onderzoek gevoerd in verschillende stappen.

- Stap 1: Vergelijken van de tools op basis van de requirements-analyse.
- Stap 2: Opstellen van een Proof-of-Concept.
- Stap 3: Checklist overlopen met de vereisten.

3.3 Stap 1: Vergelijken van de tools

Aan de hand van een aantal vooropgestelde vereisten werden de tools vergeleken en zo werd beslist welke tool de beste was voor de PoC. Het eerste deel van de vergelijking werd gedaan aan de hand van de vereisten die opgesteld waren in de ”must have” sectie van de MoSCoW-strategie. Tenslotte werden de tools ook vergeleken met de andere secties van de strategie.

3.3.1 Vergelijking op basis van de must haves

Zoals de naam al aangeeft, bestaat deze sectie uit initiatieven die een ”must” zijn. Indien een tool niet aan de verwachtingen voldeed die in de analyse aangegeven waren, was de tool niet geschikt voor de Proof-of-Concept en werd deze in het verloop van de analyse niet verder onderzocht.

De volgende vereisten werden voordien opgesteld:

- De tool moet gratis zijn of de trial moet lang genoeg duren.
- De tool moet voor een containeromgeving zijn.
- De volledige PoC moet lokaal geïnstalleerd zijn.
- De PoC moet mogelijk zijn op de laptop van de studenten die voldoet aan de opgelegde specificaties van HoGent.
- Via de Proof-Of-Concept moeten de functies van de monitoringtool goed genoeg zijn om de I/O metrics te analyseren.

3.3.1.1 De tool moet gratis zijn of de trial moet lang genoeg duren

Deze ”must have” werd eerst bekeken omdat de grootste splitsing hier zou gebeuren. De tool moest gratis zijn of de trial moest lang genoeg duren en 30 dagen was hiervoor voldoende. Op Scout en Pingdom na waren alle tools gratis. Doordat Pingdom een trial beschikbaar stelde die lang genoeg duurde, werd deze ook opgenomen naar de volgende vereiste. Scout had ook een trial maar deze duurde maar 14 dagen.

3.3.1.2 De tool moet voor een containeromgeving zijn

In theorie kan elke tool die lokaal op Linux draait, ook in een containeromgeving worden opgesteld. Maar voor deze ”must have” werd gekeken naar tools die ontwikkeld zijn voor een containeromgeving. Uit onderzoek is gebleken dat alle tools geschikt waren voor een containeromgeving. Hoewel Datadog een cloud tool was, was deze toch bruikbaar voor een containeromgeving door het gebruik van een agent. Sensu’s componenten waren niet dockerized maar waren toch bruikbaar in een containeromgeving.

3.3.1.3 De volledige PoC moet lokaal geïnstalleerd zijn

De volledige PoC moest lokaal geïnstalleerd kunnen worden. Zodat alles binnen 1 enkele Virtuele Machine op de laptop van de studenten kon geconfigureerd worden. Op Datadog na konden alle tools lokaal geïnstalleerd worden. Datadog was een tool die, buiten de agents, enkel op de Cloud werkte.

3.3.1.4 De PoC moet mogelijk zijn op de laptop van de studenten die voldoet aan de opgelegde specificaties van HoGent.

De tool moest voldoen aan de minimum specificaties die werden opgelegd door HoGent wanneer zij adviseerden bij de aankoop van een laptop. De laptop moest beschikken over een geïntegreerde webcam, min. processor i7 high performance graphics met minimum-cache van 6 MB, min. 16 GB RAM-geheugen, min. 512 GB SSD, scherm van min. 15 inch(resolutie 1920 x 1080), dedicated (aparte) grafische kaart met minimaal 4 GB videogeheugen (HoGent, g.d.).

Omdat ELK/EFK stack een server nodig had met minstens 16 GB ram was dit de enige tool die niet geschikt was voor de laptops van de studenten (Elastic, g.d.). Docker Stats en cAdvisor waren lightweight applicaties, Sysdig had een 64-bits Linux-distributie met een minimale kernelversie van 3.10 nodig en ondersteuning van de Docker Engine 1.7.1 of hoger was ook vereist voor elke serverinstantie (Sysdig, 2020a). Per agent werd beschreven dat ongeveer 500MB tot 2GB aan opslag wordt gebruikt en 2% van de totale CPU en 512MiB geheugen genoeg is (Sysdig, 2020b). Voor Prometheus werd verwacht dat de server minstens 2 CPU Cores heeft, minstens 4 GB ram geheugen en minstens 20 GB Disk space (oreilly, g.d.). Voor Sensu werd minimaal verwacht dat de server een 64-bit Intel of AMD CPU heeft, 4GB ram, 4GB Disk space (Sensu, g.d.). Zabbix vereiste minimaal 128 MB fysiek geheugen en 256 MB vrije schijfruimte. De hoeveelheid benodigde

schijfgeheugen hing natuurlijk af van het aantal hosts en parameters dat werd gemonitord (Zabbix, g.d.-a).

3.3.1.5 Via de Proof-Of-Concept moeten de functies van de monitoringtool goed genoeg zijn om de I/O metrics te configureren en analyseren.

Onder goed genoeg werd verstaan dat de PoC een webinterface nodig had met een mogelijkheid tot configuratie en analysatie van I/O metrics. Hieruit werd geconcludeerd dat alle tools op Docker Stats na geschikt waren.

3.3.2 Vergelijking op basis van de should haves

Om de keuze van de tools nog te verkleinen werd een tweede vergelijking uitgevoerd.

De volgende vereisten werden voordien opgesteld:

- Aan de hand van één enkel commando kan de Proof-of-Concept opgezet worden zodat de studenten direct aan de slag kunnen met de monitoringtool.
- Installatie en configuratie moet van een gemiddeld niveau zijn.

3.3.2.1 Installatie en configuratie moet van een gemiddeld niveau zijn.

Voor de installatie en configuratie werd een gemiddeld niveau van vaardigheden verwacht en algemene kennis. De studenten die deze tool zullen gebruiken waren op de hoogte van de volgende vaardigheden:

- Linux
- Virtual Box
- Docker
- Kubernetes

De moeilijkheidsgraad werd gebaseerd op een onderzoek door Rancher (Sissons, 2021). Omdat een te makkelijk niveau ook geen meerwaarde is voor de studenten waren cAdvisor en Pingdom hier ook uit de lijst verdwenen, samen met Sensu omdat de configuratie hiervan te moeilijk was. Alle andere tools waren hiervoor wel geschikt.

3.3.2.2 Aan de hand van één bestand kan de Proof-of-Concept opgezet worden zodat de studenten direct aan de slag kunnen met de monitoringtool.

Dit was voor alle tools hetzelfde, een bijhorend .ova bestand zal ter beschikking staan om zo gemakkelijk de Proof-of-Concept te kunnen recreëren.

3.3.3 vergelijking op basis van de could en would haves

Dit zijn de eigenschappen die een kleine meerwaarde hadden voor de Proof-of-Concept.

- Opzetten van alerts.
- Het verzamelen en analyseren van logs.

3.3.3.1 Opzetten van alerts

Alle tools hadden enige vorm van alerting aanwezig.

3.3.3.2 Het verzamelen en analyseren van logs

In alle drie de tools was het mogelijk om logs te verzamelen en te analyseren.

3.3.4 Conclusie

Uit de voorgaande vergelijking zijn 3 tools geschikt verklaard om gebruikt te worden in het lessenpakket: Zie tabel 3.1:

- Sysdig
- Prometheus
- Zabbix

Alle tools deden praktisch hetzelfde met een paar verschillen onderling. Zo waren Prometheus en Zabbix volledig gratis en moet je voor Sysdig betalen als je teveel hosts had. Zabbix en Prometheus hadden beiden hun eigen sterke en zwakke punten. Beiden hadden ook een laag gebruik van RAM en ROM en hadden beiden een bovengemiddeld niveau.

De tool die uiteindelijk gebruikt werd om de Proof-of-Concept op te zetten is Prometheus, omdat de community rond Prometheus kennelijk fanatieker was. De tool was eenvoudig genoeg, de minimum requirements voldeden zeker aan de vereisten van de laptops van de studenten, de tool was ook gratis, kan lokaal geïnstalleerd worden met een handige webinterface en werkt perfect in een container omgeving omdat Prometheus ”container-aware” was, dit wou zeggen dat de tool zelf de containers kon ontdekken.

Tabel 3.1: Tools vergelijking

	Docker Stats	cAdvisor	Scout	Pingdom	Datadog	Sysdig	Prometheus	ELK/EFK Stack	Sensu	Zabbix
Gratis of trial van minstens 30 dagen	Ja	Ja	Nee, trial van 14 dagen	Nee, trial van 30 dagen	Ja	Ja	Ja	Ja	Ja	Ja
Geschikt voor een containeromgeving	Ja	Ja	/	Ja	Ja, met een agent	Ja, met een agent	Ja	Ja	Ja	Ja
Volledig lokale installatie	Ja	Ja	/	Ja	Nee	Ja	Ja	Ja	Ja	Ja
Geschikt voor de laptop van de studenten	Ja	Ja	/	Ja	/	Ja	Ja	Nee, te veel ram geheugen	Ja	Ja
I/O metrics	Ja	Ja	/	Ja	/	Ja	Ja	/	Ja	Ja
Webinterface	Nee	Ja	/	Ja	/	Ja	Ja	/	Ja	Ja
Moeilijkheidsgraad * = Makkelijk ** = Gemiddeld *** = Boven gemiddeld **** = Moeilijk	/	*	/	*	/	**	**	/	****	***
Opzetten van alerts	/	/	/	/	/	Ja	Ja	/	/	Ja
Verzamelen en analyseren van logs	/	/	/	/	/	Ja	Ja	/	/	Ja

3.4 Stap 2: Opstellen van de Proof-of-Concept

In deze stap werd een PoC opgesteld aan de hand van de meest geschikte tool die het resultaat was uit de eerste stap. Door die eerste stap was een tool gekozen die voldeed aan alle vereisten die opgesteld waren in een MoSCoW-analyse en mogelijke extra's. De PoC was aan de hand van documentatie reproduceerbaar of eventueel een ".ova" file waarmee je de volledige opstelling ook kon importeren indien er zich problemen voor deden tijdens de installatie. De opstelling werd opgezet met tools die voor de studenten alreeds bekend waren, waardoor de focus vooral op monitoring werd gelegd.

3.4.1 Stap 2.1: Overnemen Proof-of-concept Cedric Detemmerman

De eerste stap in het opzetten van de Proof-of-Concept was het overnemen van de opstelling die door Cedric (2019) voorzien was.

3.4.1.1 Vagrant

Het gebruik van Vagrant was voor de studenten reeds gekende leerstof. Vagrant zorgde ervoor dat de initiële opstelling gemakkelijk kon worden opgezet. Dit zorgde ervoor dat wanneer de student de machine zodanig aanpaste deze terug makkelijk opgezet kon worden (Cedric, 2019).

3.4.1.2 CentOS7

CentOS7 is het operating system dat werd gebruikt. Deze CentOS7 was echter enkel de basebox, hierop was door "mrvantage"[1] een aanpassing gedaan die de installatie van Minikube afgehandeld had. Doordat studenten alreeds gebruik hadden gemaakt van dit operating system, zorgde dit ervoor dat installatie en gebruik vlot verliep (Cedric, 2019).

3.4.1.3 Configuratie Vagrant

De Vagrant configuratie was de eerste logische stap. Hiervoor werd gebruik gemaakt van de "Ansible Skeleton" Github repository van Bert Van Vreckem[2]. Studenten hadden hier al gebruik van gemaakt in vorige projecten waardoor het gebruik hiervan voor de hand liggend was. In de vagrant file werd de DEFAULT_BASE_BOX aangepast naar de box van "mrvantage":

Listing 3.1: vagrantfile

```
DEFAULT_BASE_BOX = "mrvantage/centos7minikube"
```

[1] https://app.vagrantup.com/mrvantage/boxes/centos7-minikube
[2] https://github.com/bertvv/ansible-skeleton

In de ”vagrant-hosts.yml” file werd de volgende entry toegevoegd en de alreeds bestaande entry werd verwijderd:

Listing 3.2: vagrant-hosts.yml

```
---
- name: KubernetesMonitoringIntro
  ip: 10.0.0.11
  box: "mrvantage/centos7-minikube"
  memory: 8192
  cpus: 6
---
```

In deze opstelling werd 8gb ram voorzien, wat ruim voldoende was. Indien het toestel het niet toeliet kon dit ook 4gb ram zijn. Het IP adres kon naar eigen keuze ingesteld worden (Cedric, 2019).

3.4.1.4 Bijkomende configuratie

De bijkomende configuratie zorgde voor de juiste instellingen van Minikube, een GUI, vereenvoudigen van de toegang to het dashboard, installatie van de ”cluster role binding” en het instellen van de correcte driver voor Minikube.

Listing 3.3: Extra configuratie

```
"start␣running␣some␣premade␣installationscripts␣"

sudo yum install epelrelease y
sudo yum upgrade y

"installing␣gnome␣desktop␣for␣the␣kubernetesdashboard"
sudo yum groupinstall y gnome desktop
sudo yum install y xorg*
sudo yum remove y initialsetup initialsetupgui
sudo systemctl isolate graphical.target
sudo systecmtl setdefault graphical.target

"preparing␣minikube␣and␣correcting␣driver"
sudo minikube config set driver docker
minikube delete
minikube start --cpus 4 --memory 4096


"installing␣dashboard"
kubectl apply -f
https://raw.githubusercontent.com/kubernetes/dashboard/v2.2.0/
aio/deploy/recommended.yaml
```

```
"setting up the serviceaccount admin and binding the Cluster Role"
kubectl delete clusterrolebinding kubernetesdashboard

echo "
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kube-system
" > dashboardadminuser.yml

kubectl apply f dashboardadminuser.yml

echo "
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin-user
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: admin-user
  namespace: kube-system
" > adminrolebinding.yml

kubectl apply f adminrolebinding.yml
```

3.4.1.5 Opzetten containers

Alle opstellingen die hier genoteerd werden, zijn er om later genoeg pods te kunnen monitoren op het dashboard van Prometheus. Allereerst werd een Nginx server opgezet aan de hand het eerste commando, waarop daarna een replica optie werd toegevoegd. Dit zorgde ervoor dat de Nginx pods van 2 naar 1 gingen:

Listing 3.4: nginx installatie

```
kubectl apply -f https://k8s.io/examples/application/deployment.yaml
kubectl scale deployment nginx-deployment --replicas=1
```

Hierna werd een Apache server opgezet met het volgende commando, waarop daarna een autoscale optie werd toegevoegd. Dit zorgde ervoor dat na extra gebruik van CPU extra pods werden gecreëerd:

Listing 3.5: apache installatie

```
kubectl apply -f https://k8s.io/examples/application/php-apache.yaml
kubectl autoscale deployment php-apache
        --min=3 --max=10 --cpu-percent=70
```

3.4.2 Stap 2.1: Installeren van Prometheus/Grafana

De tweede stap was de opstelling van Prometheus. De opstelling begon aan de hand van een aantal simpele commando's die na mekaar werden uitgevoerd. Hiervoor gebruikten we de 'Helm' tool, zonder Helm zouden alle yaml files van Prometheus en Grafana één voor één aangemaakt en geconfigureerd moeten worden om de samenwerking te initialiseren.

Listing 3.6: helm installatie

```
curl -fsSL -o get_helm.sh
https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3
chmod 700 get_helm.sh
./get_helm.sh
```

De volgende stap was om Helm up te date te brengen zodat voor de download van Prometheus de meest recente repository gebruikt werd.

Listing 3.7: helm repository update

```
helm repo add stable "https://charts.helm.sh/stable"
helm repo update
```

Nadien werden aan de hand van het 'Helm install' commando de Prometheus en Grafana pods gedownload.

Listing 3.8: helm prometheus installatie

```
helm install prometheus stable/prometheus-operator
```

Na enige tijd was de monitoringstack opgezet en klaar voor gebruik. Standaard was alles "out-of-the-box" geconfigureerd, dit hield in dat zowel de workers nodes als Minikube componenten in orde waren. De volgende stap was het bereiken van de Grafana GUI, aan de hand van het ip en port forwarding (bij productiegebruik zou het beter zijn om de ingress rules aan te passen) kon de web interface bereikt worden. Het eerste commando diende om de juiste naam te vinden van de pod. Het tweede commando werd gebruikt om het poortnummer te vinden.

Listing 3.9: prometheus-grafana naam/poort

```
kubectl get pod
kubectl logs prometheus-grafana-... -c grafana
```

Het poort nummer was te lezen op de voorlaatste lijn en was in dit geval, ”3000”. Op de 6de laatste lijn was de username te vinden, dit was in dit geval ”admin”. Met het volgende commando werd de grafana webinterface gestart.

Listing 3.10: port forward

```
kubectl port-forward deployment/prometheus-grafana 3000
```

Na het uitvoeren van het commando kon de Grafana webinterface bereikt worden via ”http://localhost:3000” en kon er ingelogd worden met username ”admin” en het standaard password ”prom-operator”, dit wachtwoord was te vinden in de charts van de helm Github-pagina[3].

3.4.3 Stap 2.2: Installatie WordPress/MySql

De derde stap was het installeren van WordPress met een achterliggende MySql databank. Voor het installeren werd de gids van Kubernetes gebruikt[4].

3.4.3.1 secrets.yml aanmaken

Omdat WordPress zijn data in een MySql databank bewaarde was het nodig om een wachtwoord voor Kubernetes te configureren. Het eerste commando was om een base64 wachtwoord aan te maken. Het tweede commando was het maken van het secrets.yml bestand met het touch commando.

Listing 3.11: aanmaken wachtwoord en secrets.yml

```
echo -n 'yourPassword' | base64
touch secrets.yml
```

Hierna werd het volgende toegevoegd aan de yml file. De output van het vorige commando was je wachtwoord voor het password veld in het yml bestand.

Listing 3.12: secrets.yml

```
apiVersion: v1
kind: Secret
metadata:
  name: mysql-pass
type: Opaque
data:
  password: base64_encoded_password_van_vorige_stap
```

De volgende stap was het creëren van het wachtwoord. Hierdoor was het wachtwoord te vinden op het dashboard van Minikube onder ”secrets”.

```
kubectl apply -f secrets.yml
```

[3] https://github.com/helm/charts/tree/master/stable/prometheus-operator

[4] https://kubernetes.io/docs/tutorials/stateful-application/mysql-wordpress-persistent-volume/

3.4.3.2 Downloaden resources voor MySQL en WordPress

Aan de hand van de volgende commando's werden de nodige yaml files gedownload om de pods op te zetten.

Listing 3.13: bestanden MySql

```
curl -LO
https://k8s.io/examples/application/wordpress/mysql-deployment.yaml
```

Listing 3.14: bestanden WordPress

```
curl -LO
https://k8s.io/examples/application/wordpress/wordpress-deployment.
```

3.4.3.3 Implementatie resources voor MySql en WordPress

De volgende commando's werden uitgevoerd om de implementaties van de Kubernetes-resources te maken.

Listing 3.15: implementatie

```
kubectl apply -f mysql-deployment.yaml
kubectl apply -f wordpress-deployment.yaml
```

De volgende stap was het opvragen van de URL om WordPress te bereiken.

Listing 3.16: URL WordPress

```
minikube service wordpress --url
```

Aan de hand van de URL die de output was van het vorige commando, werd naar de WordPress site gegaan, hier is de wizard volledig gevolgd tot dat de landing page van WordPress zichtbaar was.

3.4.4 Stap 2.3: Verkennen van Grafana webinterface

De volgende stap was het verkennen van Grafana en het configureren van nieuwe dashboards.

3.4.4.1 Standaard dashboards

Grafana voorzag standaard een heleboel dashboards, die te bereiken waren via "Dashboard" > "Manage". Het eerste interessante dashboard was het "Compute Resources / Node(Pods)" dashboard. Hierop was het volledige CPU gebruik makkelijk af te lezen en was het duidelijk welke server het meeste CPU nodig had. Hierop was ook een handig overzicht van het werk geheugen, genaamd Memory Quota.

Het dashboard om alle pods en hun resources metrics te bekijken was te bereiken via ”Compute Resources / Pod”. Op dit dashboard kon je elke pod apart bekijken en hiervan de CPU Usage, CPU Throttling, geheugen gebruik, netwerkverkeer enzovoort bekijken. Om de pod aan te passen volgens namespace in Minikube, kon je vanboven zowel de namespace als pod veranderen via een dropdownmenu.

Het dashboard om de pods met hun netwerk metrics te bekijken was te bereiken via ”Kubernetes / Networking / Pod”. Dit dashboard was handig om te gebruiken indien je de netwerk status van je pod wou bekijken. Hierop kon je bijvoorbeeld zien wat de bandbreedte was, het aantal pakketten etc.

3.4.4.2 Groei van containers

De volgende commando’s werden één voor één uitgevoerd voor een load-generator te genereren om live pods te zien verschijnen op het dashboard dat te bereiken was via ”Kubernetes / Compute Resources / Namespace (Pods)”. Via deze manier kon gezien worden dat bij het creëren van nieuwe pods het dashboard aangevuld werd met de nieuwe pods.

Listing 3.17: load-generator

```
kubectl run -it rm load-generator image=busybox /bin/sh
while true; do wget -q -O- http://php-apache; done
```

3.4.4.3 Nieuwe dashboards importeren

Een tweede manier om dashboards te gebruiken was ze importeren via ID codes die te vinden waren op de Dashboard repository van Grafana[5]. Als voorbeeld werd een dashboard toegevoegd met code 2115. Dit werd gedaan via ”Dashboard” > ”Manage” > ”Import”. De databank van user-created dashboards was immens groot en daarom is gebruik gemaakt van een basis dashboard om de werking van het importeren alleszins goed te tonen. Op dit dashboard was alsnog heel veel te zien, zoals:

- Total usage
- Pods CPU usage
- Pods memory usage
- Pods netwerk I/O
- CPU usage van alle processen
- ...

[5] https://grafana.com/grafana/dashboards/

3.4.5 Stap 2.4: Maken van eigen dashboards

Naast het gebruiken van standaard dashboards of het importeren van vooraf gemaakte dashboards, was er ook nog een andere manier om dashboard toe te voegen. Via deze weg werden zelfgemaakte dashboards toegevoegd: ”Dashboard” > ”New Dashboard”.

Het dashboard dat gemaakt werd was voor het controleren van metrics op de WordPress/-MySql pod. Dit dashboard kon eventueel ook voor andere pods gebruikt worden.

3.4.5.1 Dashboard instellingen

De eerste stap was om de algemene instellingen van het dashboard te veranderen. Bij de instellingen, te bereiken via het tandwiel, werd allereerst het volgende aangepast onder general:

- Name: Dashboard PoC
- Auto-Refresh: 10s,30s,1m,5m,15m,30m,1h,2h,1d

Hierna werden 4 variables toegevoegd onder de tab ”Variables”.

- General
  name = datasource
  type = Datasource
- Data Source Options
  type = Prometheus

- General
  name = cluster
  type = query
  hide = variable
- Data source options
  type = $datasource
  refresh = On Dashboard Load
  query = label_values(kube_pod_info, cluster)
  sort = Alphabetical (ASC)

- General
  name = namespace
  type = query
  hide = variable
- Data source options
  type = $datasource
  refresh = On Dashboard Load
  query = label_values(kube_pod_infocluster=”$cluster”, namespace)
  sort = Alphabetical (ASC)

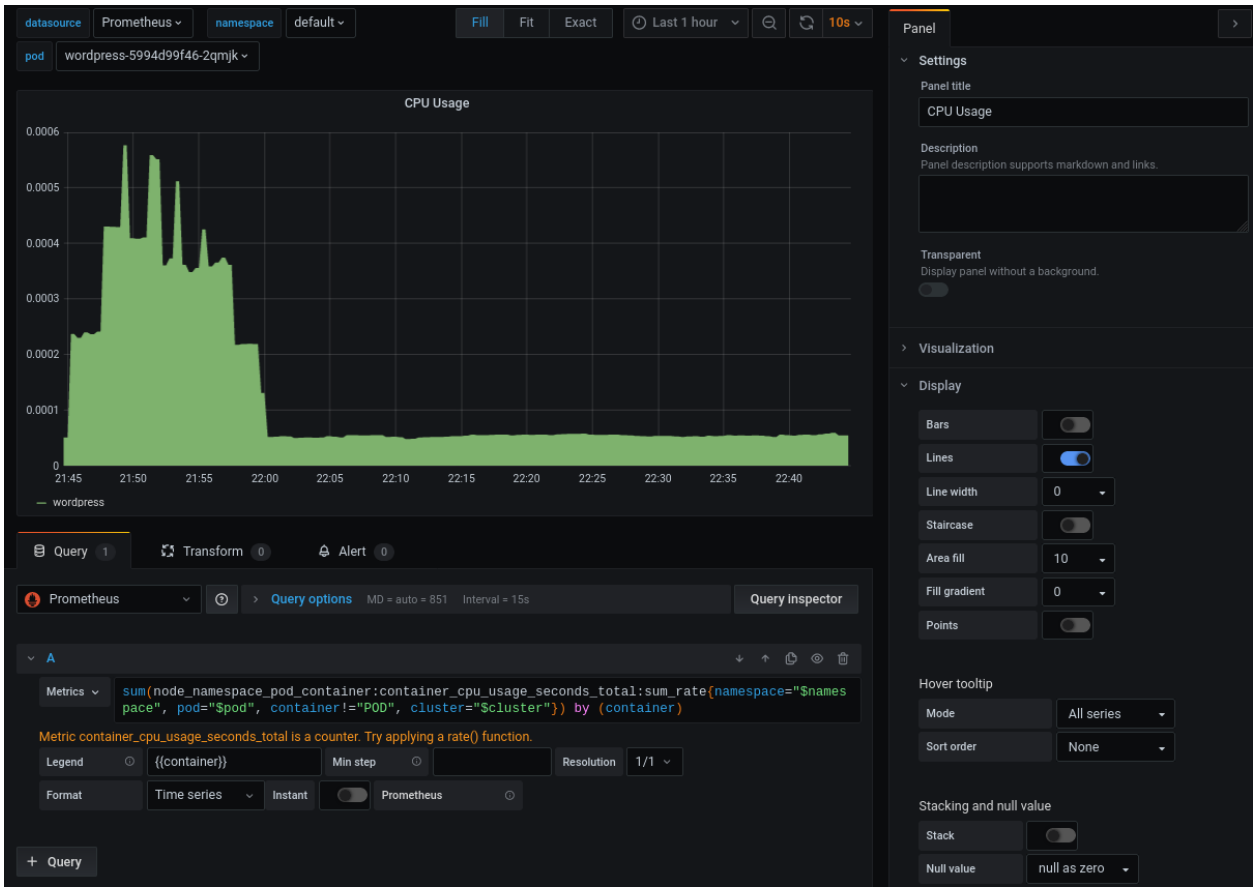
- General
  name = pod
  type = query
  hide = variable
- Data source options
  type = $datasource
  refresh = On Time Range Chan
  query = label_values(kube_pod_infocluster=”$cluster”, namespace=”$namespace”, pod)
  sort = Alphabetical (ASC)

Door deze instellingen was het mogelijk om op het dashboard via ”datasource”, ”namespace” en ”pod” de juiste pod te bereiken door deze in de dropdownmenu’s te selecteren. Voor dit voorbeeld was dit: datasource = prometheus, namespace = default en pod = WordPress. Ten laatste werd de time range veranderd naar ”Last 1 hour” en de refresh tijd naar ”10s”.

3.4.5.2 Creëren panelen

Aan de hand van het maken van panelen kon er, naar keuze, op metrics gemonitord worden. Panelen creëren is niet evident en vereiste enige kennis voor het schrijven van de nodige query's. De community achter deze tool verzorgde gelukkig genoeg documentatie om te helpen bij het schrijven van deze query's. [6]

Het eerste paneel dat gemaakt werd, was het paneel om het CPU gebruik te bekijken. Door een actie (installeren van een plugin/thema) uit te voeren op de WordPress site kon een verhoging van het CPU gebruik gezien worden in de grafiek. Bij "settings" kon de naam aangepast worden naar "CPU Usage". De volgende afbeelding werd gebruikt om de nodige velden in te vullen.



Figuur 3.1: Instellingen voor paneel 1

Het tweede paneel dat gemaakt werd, was het paneel om het werkgeheugen gebruik te bekijken. Door een actie (installeren van een plugin/thema) uit te voeren op de WordPress site kon een verhoging van het werkgeheugen gebruik gezien worden in de meter. Bij "settings" kon de naam aangepast worden naar "Memory Usage". De afbeelding werd gebruikt om de nodige velden in te vullen.

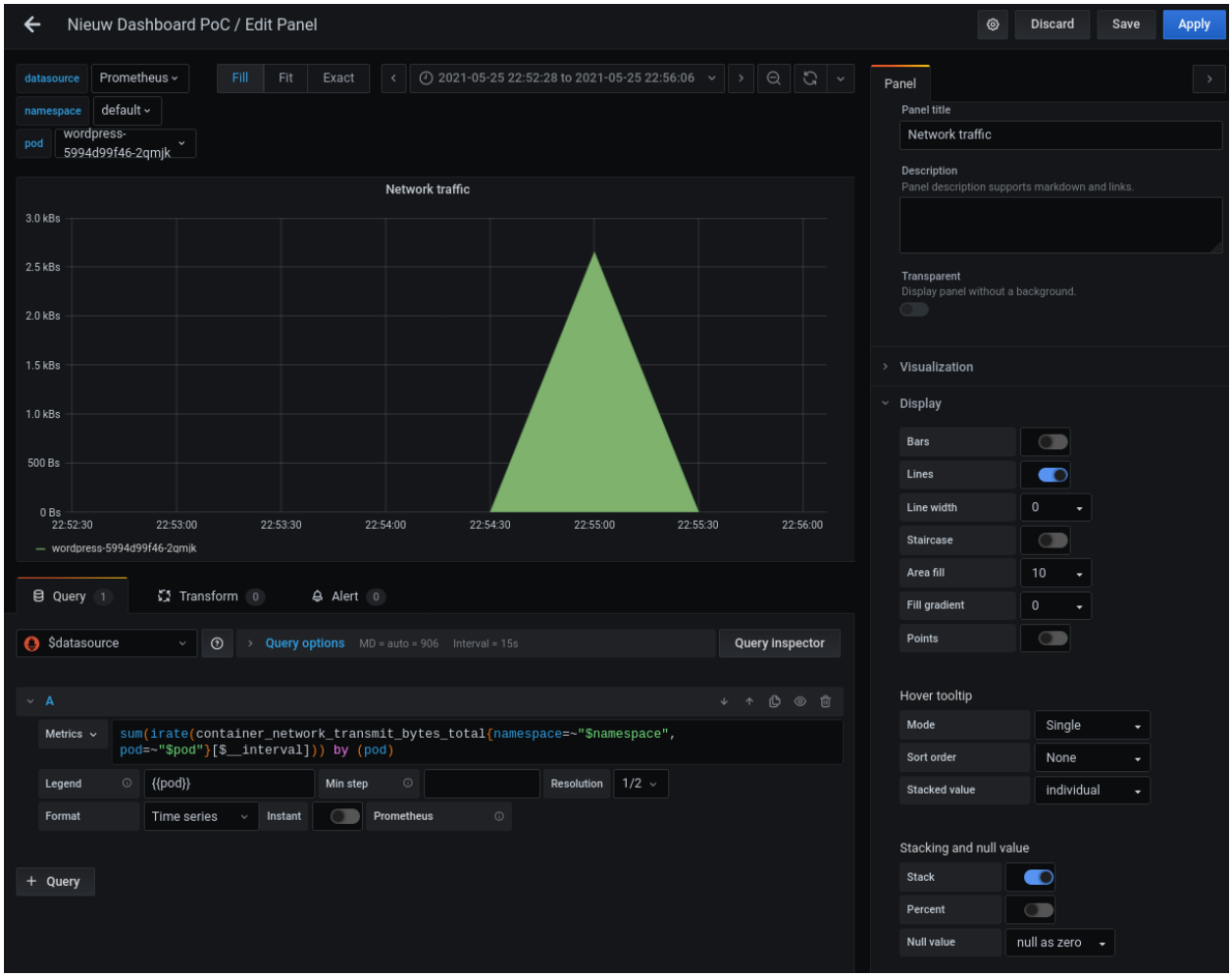
[6] https://grafana.com/docs/grafana/latest/panels/queries/

(a)

(b)

Figuur 3.2: Instellingen voor paneel 2

Het derde paneel dat gemaakt werd, was het paneel om het netwerkverkeer te bekijken. Door een actie (downloaden van een plugin/thema) uit te voeren op de WordPress site kon een verhoging van het netwerkverkeer gezien worden in de grafiek. Bij ”settings” kon de naam aangepast worden naar ”Network Usage”. De afbeelding werd gebruikt om de nodige velden in te vullen. š



Figuur 3.3: Instellingen voor paneel 3

3.4.6 Stap 2.5: Grafana dashboard in werking

Na het instellen van deze drie panelen, was er een dashboard beschikbaar waarbij de pod kon aangepast worden. Om te zien dat de grafieken en meters responsief waren op een mogelijke bewerking die in een pod plaatsvond, kon er gebruik gemaakt worden van 3.4.4.2 ”Groei van containers”. Hiervoor werd verwacht dat de pod op het dashboard ingesteld werd op een ”apache”pod waarop de groei van de containers zichtbaar was.

3.4.7 Stap 2.6: Bereiken van de Prometheus webinterface

Hoewel de webinterface van Grafana voorkeur had door de gebruiksvriendelijke manier, werd deze webinterface van Prometheus toch opgezet om de interface te kunnen bereiken. De interface was te bereiken via ”http://localhost:9090” na het uitvoeren van het volgende commando.

Listing 3.18: port forward prometheus

```
kubectl port-forward
prometheus-prometheus-prometheus-oper-prometheus-0 9090
```

3.4.8 Samenvatting

Nu waren er drie webinterfaces die te bereiken waren voor de monitoring van de container omgeving, waarvan twee interfaces bijna volledig opgezet waren met een simpel commando door middel van ”Helm” pakketten.

- Minikube Dashboard te bereiken via het ”MiniKube Dashboard” commando
- Grafana Dashboard te bereiken via http://localhost:3000
- Prometheus Dashboard te bereiken via http://localhost:9090

3.5 Stap 3: Overlopen checklist

Na het opstellen van de Proof-of-Concept werd een checklist overlopen met daarin de vooropgestelde vereisten voor een succesvolle Proof-of-Concept. Aan de hand van de checklist werd bepaald of de PoC goed genoeg was voor gebruik in het lessenpakket.

De checklist bestond uit de volgende punten:

- De tool moet gratis zijn of de trial moet lang genoeg duren.
- De tool moet voor een containeromgeving zijn.
- De volledige PoC moet lokaal geïnstalleerd zijn.
- De PoC moet mogelijk zijn op de laptop van de studenten die voldoet aan de opgelegde specificaties van HoGent.
- Via de Proof-Of-Concept moeten de functies van de monitoringtool goed genoeg zijn om de I/O metrics te analyseren.
- Aan de hand van één bestand kan de Proof-of-Concept opgezet worden zodat de studenten direct aan de slag kunnen met de monitoringtool.
- Installatie en configuratie moet van een gemiddeld niveau zijn.
- Opzetten van alerts.
- Het verzamelen en analyseren van logs.

Uit de vorige checklijst kon het volgende geconcludeerd worden:

- De tool was gratis.
- De tool was geschikt voor een containeromgeving.
- De volledige PoC was lokaal geïnstalleerd.
- De PoC was mogelijk op de laptop van de studenten.
- Via de Proof-Of-Concept waren de functies van de monitoringtool goed genoeg om de I/O metrics te analyseren.
- Een OVA bestand was niet aanwezig om het makkelijk reproduceerbaar te maken maar werd wel door dit onderzoek ter beschikking gesteld.
- Installatie en configuratie waren van een gemiddeld niveau zijn.
- Opzetten van alerts was mogelijk maar niet uitgevoerd in dit bachelorproef.
- Het verzamelen en analyseren van logs was mogelijk maar niet uitgevoerd in dit bachelorproef.

4. Conclusie

Op het begin van het onderzoek werden een aantal onderzoeksvragen opgesteld waaronder één hoofdonderzoeksvraag en meerdere deelvragen:

- Welke monitoringtools zijn geschikt?
- Wat zijn de belangrijkste verschillen tussen de gekozen tools?
- Waar moet de gekozen tool inzicht op geven?
- Welk proof-of-concept is genoeg om de leerstof te verstaan?

De hoofdonderzoeksvraag van dit onderzoek was het vinden van een tool die geschikt was om de studenten toegepaste informatie genoeg bij te leren over de mogelijkheden in een monitoringtool voor een Docker/Kubernetes omgeving. Dit betekende dat de tool zowel technische mogelijkheden nodig had, alsook educatieve.

Door een requirements-analyse uit te voeren en hierin de belangrijkste verschillen bloot te leggen werden hieruit drie tools geschikt verklaard:

- Sysdig
- Zabbix
- Prometheus

De drie tools waren praktisch hetzelfde op basis van de requirements-analyse waardoor de keuze werd gemaakt door de monitoringcommunity, hierbij was Prometheus de tool die met voorsprong uitblonk. Voor de Proof-of-Concept werd dan uiteindelijk ook voor die tool gekozen.

Er werd een aantal dashboards overlopen die standaard geïnstalleerd werden waarin verschillende metrics zichtbaar werden gesteld. Het importeren van json files voor com-

munity created dashboards werd ook uitgelegd omdat daar uit snel een dashboard kan worden opgezet. Daarna werd ook een eigen dashboard gecreëerd om aan te tonen dat het maken van dashboards vrij eenvoudig en interactief was. Op deze dashboards werden dan een aantal technieken toegepast om aan te tonen dat bij gebruik van een pod, deze zeker aanpassen.

De community van talloze experts die het opensource project ondersteunen zorgden ervoor dat de installatie van de tool vrij eenvoudig op te zetten was. Zonder deze tools was de moeilijkheidsgraad voor het opzetten van Prometheus eventueel al een reden genoeg om een andere tool te kiezen. Doordat Prometheus een befaamde tool was die in vele bedrijven gebruikt werd, was er al een vermoeden dat deze tool in de laatste vergelijking zou terecht komen. Door het gebruik van een containeromgeving kon er gemakkelijk een andere pod toegevoegd worden om die dan alsook te monitoren in Prometheus. Dit wil zeggen dat de tool genoeg inzicht gaf op alles wat nodig was om de onderzoeksvragen te beantwoorden.

Hopelijk werd dit onderzoek gezien als een meerwaarde voor het lessenpakket bij de studenten toegepaste informatica zodat monitoring voor gecontaineriseerde omgevingen basiskennis werd. Aangezien de populariteit van deze technologie alleen maar steeg kon dit zeker een meerwaarde zijn, al was het enkel voor de lezer van dit onderzoek.

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

Containertechnologie vindt in recordtempo zijn weg naar de data-omgeving van de ondernemingen (Cole, 2016). Het gemak waarmee containerplatformen zoals Docker kunnen worden ingezet, suggereert dat ze de dominantste architectuur zijn en zullen blijven voor deze en de volgende generatie services en microservices. De uitdaging om een goede monitoring van containers op te stellen is belangrijk. Zoals te verwachten, zullen traditionele monitoringsplatforms die vooral gebaseerd zijn op virtualisatie, niet voldoende zijn. Containers zijn zeer vluchtig in ontstaan en net zo snel in het verdwijnen. Die snelheid is vaak te danken aan een geautomatiseerd proces waarbij een minimale input vereist is. Deze containers bevinden zich tussen host -en applicatielaag, wat het moeilijk maakt om in detail te zien hoe ze zich gedragen en of ze efficiënt gebruik maken van hulpbronnen (e.g. CPU, RAM, etc.) en goede systeem prestaties leveren. De valkuil voor vele bedrijven is het overschatten van een eenvoudige hostmonitoring voor een container. Dit idee wordt echter al snel uit de weg geruimd, omdat het aantal containers snel begint te vergoten wat een traditioneel hostmonitoring niet kan bijhouden. Binnen HoGent is er voorlopig geen opleidingsonderdeel omtrent monitoring, specifiek op een Docker omgeving in een Kubernetes orkestratie. Daar dit een zeer interessant en relevant onderwerp is, is het een goede zaak om deze leerstof bij te leren aan de toekomstige studenten van HoGent. De onderzoeksvraag komt in dit geval uit van een docent tewerkgesteld in HoGent. Omdat er nog geen intern onderzoek heeft plaatsgevonden, is deze onderzoeksvraag dus ontstaan.

De doelstelling van dit onderzoek is bepalen welke monitoringtool hiervoor geschikt is en welke haalbaar is voor de studenten.
Zo bekomen we volgende onderzoeksvragen:

- Welke monitoringtools zijn geschikt?
- Wat zijn de belangrijkste verschillen tussen de gekozen tools?
- Waar moet de gekozen tool inzicht op geven?
- Welk proof-of-concept is genoeg om de leerstof te verstaan?

A.2 State-of-the-art

Hoewel er vele onderzoeken zijn naar goede monitoringtools voor Docker, zijn het vooral onderzoeken waarbij ook de betalende tools vergeleken worden (Ribenzaft, 2020) en (Cirelly, 2020). Deze onderzoeken specificeren zich ook niet op één specifieke 'proof-of-concept' maar leggen vooral de voor- en nadelen van elke tool uit. Hoewel het boek van Alex Williams (Cole, 2016) ook niet echt een 'proof-of-concept' heeft, komt het onderzoek toch dichter in de buurt van wat het verwachte resultaat is van deze bachelorproef. Hierin wordt ook over een aantal andere onderwerpen gesproken omtrent monitoring die zeker interessant zijn. Daar een deel van dit boek ook een aantal tools beschrijft en vergelijkt is dit zeker relevant aan de onderzoeksvraag. De bedoeling van deze bachelorproef is het onderzoeken van zowel zelf-gehoste open source-oplossingen als commerciële cloud-gebaseerde services te bekijken, dit boek kaart ook deze opties aan. Volgens het boek van Alex Williams (Cole, 2016) is de conclusie dat de keuze sterk afhangt van de resultaten die je wenst te bereiken die bij uw werklast passen, eventueel met een combinatie van extra tools. Door opzoekingswerk en verzameling van interne informatie worden de vereisten bepaald,zo kan er gekozen worden voor de juiste monitoringtools. Tot heden is er geen exacte kopie van de onderzoeksvraag, wat deze bachelorproef uniek maakt en interessanter.

A.3 Methodologie

Allereerst zal een MoSCoW requirements-analyse opgesteld worden aan de hand van een interview met mogelijke belanghebbende. Voor dit onderzoek zal vooral de mening van docenten die dit onderwerp zullen voordragen aan de studenten belangrijk zijn. Hierna zal een literatuurstudie opgesteld worden waarin volgende onderwerpen zullen afgehandeld worden:

- Concept monitoring, soorten monitoring, taken van monitoring
- Architectuur monitoringsystemen in de context van containers/Kubernetes
- Overzicht aanbod producten/tools in dit marktsegment

Vervolgens zullen de gevonden producten/tools afgetoetst worden aan de requirements die

voordien zijn opgesteld door de belanghebbende. Nadien wordt een bestaande 'proof-of-concept' gebruikt (Cedric, 2019) waarop de meest belovende producten toegepast worden. Indien er in deze 'proof-of-concept' tekortkomingen zijn voor het bekomen van de vooraf bepaalde requirements, zal deze verwerkt worden tot het punt waar er deze tekortkomingen miniem zijn.

A.4 Verwachte resultaten

Uit deze studie, naar een gepaste monitoringtool, wordt verwacht dat aan de hand van de experimenten, het duidelijk wordt welke tool een basis kan zijn voor een deel van het opleidingsonderdeel waarop beslist wordt wat toekomstige studenten kunnen gebruiken. Het opstellen van een goede proof-of-concept is uiterst belangrijk voor het bereiken van goede resultaten.

A.5 Verwachte conclusies

Het experiment zou een goede basis moeten hebben om zo de essentiële onderdelen van monitoring aan te leren aan studenten toegepaste informatica. Met een basis wordt vooral bedoeld dat de 'proof-of-concept' makkelijk reproduceerbaar is en dat de 'beste' monitoringtool en zijn functionaliteiten allemaal uit te voeren zijn. Bovendien is het de bedoeling dat de 'proof-of-concept' alle deelaspecten van monitoring bevat met inzicht op, in geval van dit onderzoek, zicht op het voortdurend meten van allerlei performantiemetrieken. Indien het experiment aan deze eisen voldoet, mogen we concluderen dat dit een geslaagd onderzoek is.

Bibliografie

Ameen, R. Y. & Hamo, A. Y. (2013). Survey of server virtualization. *arXiv preprint arXiv:1304.3557*.

AWS. (2021, januari 1). *Container Monitoring - Why, how, and what to look out for* (AWS, Red.). Verkregen 7 mei 2021, van https://aws.amazon.com/cloudwatch/container-monitoring/

Babak Bashari Rad, M. A., Harrison John Bhatti. (2017, maart 1). *An Introduction to Docker and Analysis of its Performance* (tech. rap.). Asia Pacific University of Technology en Innovation.

Brendan Burns, K. H., Joe Beda. (2019, april 10). *Kubernetes Up Running - Dive into the Future of Infrastructure* (O'Reilly, Red.). Verkregen 27 april 2021, van https://books.google.be/books?hl=en&lr=&id=-5izDwAAQBAJ&oi=fnd&pg=PP1&dq=kubernetes&ots=IsB4lxkeeI&sig=RBWC7ndt9aEs0bUHZ1usfNSGgQw&redir_esc=y#v=onepage&q&f=false

Bui, T. (2015). Analysis of docker security. *arXiv preprint arXiv:1501.02967*.

Cedric, D. (2019). *Kubernetes Educatieve Mogelijkheden En Opstellingen* (masterscriptie). HoGent.

Charity, M. (2016, april 10). *Why Honeycomb? Black Swans, Unknown-Unknowns, and the Glorious Future of Doom* (C. Majors, Red.). Verkregen 1 mei 2021, van https://www.honeycomb.io/blog/why-honeycomb-black-swans-unknown-unknowns-and-the-glorious-future-of-doom/

Cirelly, J. (2020, september 20). *7 Best Docker Monitoring Tools*. https://www.itprc.com/docker-monitoring-tools/

Cole, A. (2016, september 29). *The Importance of Container Monitoring*. https://www.stratoscale.com/blog/importance-container-monitoring/

Daniels, J. (2009). Server virtualization architecture and implementation. *XRDS: Crossroads, The ACM Magazine for Students*, *16*(1), 8–12.

Elastic. (g.d.). *Elasticsearch: The Definitive Guide* (Elastic, Red.). Verkregen 13 mei 2021, van https://www.elastic.co/guide/en/elasticsearch/guide/current/hardware.html

Gilles, A. (2019, januari 1). *sidecar proxy* (A. Gilles, Red.). https://searchitoperations.techtarget.com/definition/sidecar-proxy

HoGent. (g.d.). *Op zoek naar een nieuwe laptop voor je opleiding?* (HoGent, Red.). Verkregen 13 mei 2021, van https://www.hogent.be/student/een-vlotte-start/laptopaanbod/

Jorge Castro, K. C., Duffie Cooley. (2020, december 2). *Don't Panic: Kubernetes and Docker* (Kubernetes, Red.). Verkregen 23 mei 2021, van https://kubernetes.io/blog/2020/12/02/dont-panic-kubernetes-and-docker/

Journal, C. (2020, december 29). *How Docker and Kubernetes Work Together* (P. Belagatti, Red.). Verkregen 27 april 2021, van https://containerjournal.com/topics/container-ecosystems/how-docker-and-kubernetes-work-together/

Kedia, P., Nagpal, R. & Singh, T. P. (2013). A survey on virtualization service providers, security issues, tools and future trends. *International Journal of Computer Applications*, *69*(24).

Kubernetes. (2021, februari 1). *What is Kubernetes?* (K. Bradshaw, Red.). Verkregen 21 april 2021, van https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Matrix. (2016, september 28). *The Importance of Infrastructure Monitoring* (Matrix, Red.). Verkregen 7 mei 2021, van https://matrix-ibs.com/2016/09/28/importance-infrastructure-monitoring/

Oracle. (2012, januari 1). *Introduction to Virtualization* (Oracle, Red.). Verkregen 19 april 2021, van https://docs.oracle.com/cd/E26996_01/E18549/html/VMUSG1011.html

oreilly. (g.d.). *Hands-On Infrastructure Monitoring with Prometheus by Joel Bastos, Pedro Araujo* (A. Joel Bastos Pedro, Red.). Verkregen 13 mei 2021, van https://www.oreilly.com/library/view/hands-on-infrastructure-monitoring/9781789612349/bc69aa2f-7482-421f-b070-9116ca39cae7.xhtml

Rancher. (2020, januari 1). *A guide to Kubernetes with Rancher* (tech. rap.). Rancher.

Ribenzaft, R. (2020, maart 9). *Top Six Open Source Tools for Monitoring Kubernetes and Docker*. https://devops.com/top-six-open-source-tools-for-monitoring-kubernetes-and-docker/

Ruiz, C., Jeanvoine, E. & Nussbaum, L. (2015). Performance evaluation of containers for HPC. *European Conference on Parallel Processing*, 813–824.

Sensu. (g.d.). *Hardware requirements* (Sensu, Red.). Verkregen 13 mei 2021, van https://docs.sensu.io/sensu-go/6.3/operations/deploy-sensu/hardware-requirements/#backend-minimum-requirements

Singh, A. (2004, januari 1). *An Introduction to Virtualization* (A. Singh, Red.). Verkregen 17 april 2021, van https://webx.ubi.pt/~hgil/utils/Virtualization_Introduction.html

Sissons, G. (2021, januari 26). *Comparing 10 Docker Container Monitoring Solutions for Rancher* (G. Sissons, Red.). Verkregen 8 mei 2021, van https://rancher.com/comparing-10-container-monitoring-solutions-rancher/

Sysdig. (2020a, augustus 13). *Host Requirements for Agent Installation* (Sysdig, Red.). Verkregen 13 mei 2021, van https://docs.sysdig.com/en/system-requirements.html

Sysdig. (2020b, augustus 13). *Host Requirements for Agent Installation* (Sysdig, Red.). Verkregen 13 mei 2021, van https://docs.sysdig.com/en/host-requirements-for-agent-installation.html

TechTarget. (2020, mei 1). *IT monitoring* (A. Gilles, Red.). Verkregen 29 april 2021, van https://searchitoperations.techtarget.com/definition/IT-monitoring

Van Vleck, T. (2015, maart 16). *The IMB 7094 and CTSS* (T. Van Vleck, Red.). Verkregen 17 april 2021, van https://multicians.org/thvv/7094.html

Williams, A. (2016, september 19). *Monitoring Management With Docker Containers*.

Zabbix. (g.d.-a). *Requirements* (Zabbix, Red.). Verkregen 13 mei 2021, van https://www.zabbix.com/documentation/current/manual/installation/requirements

Zabbix. (g.d.-b). *What is Zabbix* (Zabbix, Red.). Verkregen 11 mei 2021, van https://www.zabbix.com/documentation/4.0/manual/introduction/about