



Faculteit Bedrijf en Organisatie

Kubernetes Educatieve Mogelijkheden En Opstellingen

Detemmerman Cedric

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

Promotor:
Pieter-Jan Maenhaut
Co-promotor:
Bert Van Vreckem

Instelling: HoGent

Academiejaar: 2019-2020

Derde examenperiode

Faculteit Bedrijf en Organisatie

Kubernetes Educatieve Mogelijkheden En Opstellingen

Detemmerman Cedric

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

Promotor:
Pieter-Jan Maenhaut
Co-promotor:
Bert Van Vreckem

Instelling: HoGent

Academiejaar: 2019-2020

Derde examenperiode

Woord vooraf

Deze bachelorproef dient als eindwerk voor mijn opleiding Toegepaste Informatica afstudeerrichting Systeembeheer.

Hoewel ik in het begin tijdens de uitwerking van deze bachelorproef twijfels had of dit onderwerp voor mij bestemd was, ben ik doorheen de maanden meer en meer gepassioneerd geraakt door het onderwerp.

Deze bachelorproef was een uitdagende onderneming waar ik veel nieuwe vaardigheden en ervaringen mee heb opgedaan. Dit maakt de afwerking van deze paper een bitter-zoet moment aangezien ik met de kennis die ik nu heb het volledig anders zou aangepakt hebben vanaf het begin.

Ongetwijfeld heb ik deze opdracht niet alleen kunnen afwerken en zou ik zeker enkele personen willen bedanken. Graag zou ik mijn co-promotor Bert van Vreckem en mijn promotor Pieter-Jan Maenhaut willen bedanken om mij een kans aan te bieden om aan deze opdracht te werken en alle steun die ze mij aangeboden hebben. Daarnaast wil ik ook mijn moeder bedanken om mij te blijven ondersteunen doorheen een intussen lang aansleepende professionele bachelor. Voor mij voelt het als een verplichting aan om mezelf te blijven verbeteren om zo hoog mogelijk te mikken ondanks alle moeilijkheden die ik tot nu toe moeten verduren heb en de obstakels die ik in de toekomst nog zal tegenkomen.

Hopelijk betekend deze bachelorproef iets voor u en wens ik u veel leesplezier toe!

Samenvatting

Toegepaste Informatica is een dynamische opleiding die continu moet inspelen op nieuwe technologieën, evoluties en werkmiddelen die gebruikt worden in het IT-werkveld. Wordt dit niet gedaan, dan riskeert de opleiding om verouderde software aan te leren waar een student uiteindelijk weinig aan zal hebben in het werkveld. Kubernetes is zo een evolutie, een belangrijke technologie die de systeembeheer-wereld sterk kan veranderen. Daarom zal Kubernetes eventueel moeten aangeleerd worden. Hoe dat best gebeurt is een vraag op dewelke deze bachelorproef enkele antwoorden zal proberen geven.

Het eerste dat deze bachelorproef zal proberen, is het aan te tonen wat Kubernetes is, hoe Kubernetes tot stand gekomen is, hoe het in elkaar zit en hoe het werkt. Met deze fundamentele informatie gaat de paper even verder kijken naar de plaats van Kubernetes in de systeembeheer-wereld om een beter beeld te krijgen waarom dit geen technologie is die over het hoofd mag gezien worden.

Daarna wordt er gezocht naar verschillende manieren om Kubernetes effectief aan te leren en op te zetten. Deze manieren worden later vergelijkt met elkaar en er wordt gekeken of deze voldoen aan de functionele vereisten waar deze oplossingen moeten aan voldoen, vooraleer ze bruikbaar zijn binnen een leeromgeving.

Wanneer er één of meerdere geschikte oplossingen gevonden zijn die aan alle functional requirements voldoen, werd één hiervan uitgebouwd tot een bruikbare proof of concept. Deze PoC werd daarna op de proef gesteld door middel van een testfase waarin enkele individuen met een kennis-profiel vergelijkbaar met dat van een Informatica student de proof of concept uitproberen.

Deze personen zijn vrijwillige deelnemers die nog geen voorafgaande ervaring hebben over Kubernetes. En kunnen een beter beeld creëren of deze opstellingen in de praktijk

haalbaar is voor studenten informatica.

Uit het onderzoek naar een aanvaardbare oplossing werd er geconcludeerd dat er 3 tools aan alle criteria voldoen: Minikube, K3 en MicroK8s.

Er werd besloten om Minikube verder uit te bouwen tot de PoC waar de tweede test mee werd uitgevoerd.

Wat er daar uit kan besloten worden is dat de opstelling in onder 120 minutes de basis concepten van kubernetes kan uitleggen en voortonen. Daarbij lijkt het er sterk op dat Minikube een beginnervriendelijk platform is om kennis mee te kunnen maken met Kubernetes.

Inhoudsopgave

1	Inleiding	15
1.1	Probleemstelling	15
1.1.1	Het probleem met cloud based solutions	15
1.1.2	alternatieve lokale oefenplatformen voor kubernetes	16
1.2	Onderzoeksvraag	16
1.2.1	Hoofdonderzoeksvraag	16
1.2.2	Deelvragen	16
1.3	Onderzoeksdoelstelling	17
1.4	Hypothese	17
1.5	Opzet van deze bachelorproef	17

2 Stand van zaken 19

2.1 Wat is virtualisatie 19

2.1.1 Geschiedenis van virtualisatie 19

2.1.2 de 3 tijdperken van virtualisatie 21

2.2 Wat is/hoe werkt/waarom kubernetes? 23

2.2.1 Hoe werkt Kubernetes? 24

2.2.2 Waarom wordt Kubernetes gebruikt? 29

2.3 Docker swarm en Kubernetes 30

2.3.1 Docker als de nieuwe marktleider 30

2.3.2 De opkomst van orkestratie met Kubernetes en Docker swarm 30

2.4 Het nut en belang van Kubernetes in het lesprogramma 31

2.4.1 Marktonderzoek door Rancher Labs 31

2.4.2 Marktonderzoek door Cloud native Computing foundation 32

2.4.3 In conclusie 34

2.5 Welke Kubernetes oplossingen zijn er 34

3 Functionele vereisten en methodologie 37

3.1 Must have 38

3.2 Should have 38

3.3 Nice to have 38

3.4 Onderzoeksmethodologie 39

3.4.1 Fase 1: Onderzoeksadvertentie en het aanwerven van geschikte kandidaten 39

3.4.2 Fase 2: Bepalen van de geschikte tools 40

3.4.3 Fase 3: Opstellen proof of concept 40

3.4.4	Fase 4: Uitvoeren test	40
3.4.5	Fase 5: Eind-bevraging van de testers	41
4	Resultaten: Vergelijking van de mogelijke opstellingen	43
4.1	De oplossingen	43
4.2	vergelijking van de 'must haves'	44
4.3	Vergelijking van de 'Should haves' en de 'Nice to haves'	48
4.4	Conclusies en gekozen oplossing voor proof of concept	49
5	Proof Of Concept	51
5.1	Redenering achter de opstelling	51
5.1.1	Vagrant	51
5.1.2	CentOS 7	51
5.1.3	Minikube	52
5.2	Structuur van de Kubernetes omgeving	53
5.2.1	Vagrant configuratie	53
5.2.2	Automatisering van bijkomende installaties	56
6	Resultaten enquête en test	59
6.1	Profiel van de onderzoeksdeelnemers	59
6.2	Opstelling van de test	60
6.3	Test en enquête Resultaten	61
6.3.1	Resultaten: Tijdsmeting	61
6.3.2	Resultaten: Algemene moeilijkheidsgraad	62
6.3.3	Resultaten: Specifieke struikelblokken	63
6.3.4	Resultaten: Heeft de deelnemer iets bijgeleerd over Kubernetes?	64

6.3.5	Extra opmerkingen en observaties	64
7	Conclusies	65
A	Onderzoeksvoorstel	67
A.1	Introductie	67
A.2	State-of-the-art	67
A.3	Methodologie	68
A.4	Verwachte resultaten	68
A.5	Verwachte conclusies	68
B	Proof of concept documentatie	69
B.1	Introductie en Hands-on tutorial.	69
B.2	Opdrachtgedeelte	75
	Bibliografie	79

Lijst van figuren

2.1	Voorstelling van de evolutie van applicatie deployment (Kubernetes, 2020)	23
2.2	Een diagram dat een Kubernetes cluster voorsteld (Kubernetes, 2020)	25
2.3	Voorstelling van een Kubernetes Pod en de inhoud van een pod. (Paris, 2014)	25
2.4	Vereenvoudigde Voorstelling van de weg die een source code applicatie aflegt tot in een Kubernetes pod. (Stapleton, 2016)	27
2.5	Voorbeeld .yaml bestand voor object creatie (Kubernetes, 2020)	28
2.6	Rancher Labs enquête verdeling Container adoptie in professionele productie omgevingen.(Smails, 2019)	31
2.7	Resultaten CNCF enquête rond container gebruik en de uitdagingen rond containers (CNCF, 2019)	33
2.8	Resultaten CNCF enquête rond container gebruik en de uitdagingen rond containers (CNCF, 2019)	33
2.9	Visualisatie van K3s architectuur (K3, 2020)	34
5.1	Vagrant configuratie om box te gebruiken.	52
5.2	De manuele actie nodig om toegang te krijgen tot het dashboard met Minikube.	53
5.3	Overzicht van de gnome desktop interface.	58

B.1	70
B.2	71
B.3	72
B.4	73
B.5	74
B.6	74
B.7	76
B.8	76
B.9	76
B.10	77

Lijst van tabellen

6.1	Resultaten test deelnemers profiel	60
6.2	tijdsbesteding van de deelnemers tijdens de opdracht.	62
6.3	Bevraging moeilijkheidsgraad van de introductie en opdracht. ..	62
6.4	Met welke onderdelen hadden de deelnemers problemen	63
6.5	Algemene mening of de deelnemer iets bijgeleerd heeft.	64

1. Inleiding

1.1 Probleemstelling

Binnen de opleiding Toegepaste Informatica moeten studenten leren werken met technologieën en tools die gebruikt worden in het momentele werkveld, en Kubernetes is daar één van. Helaas is Kubernetes een uitdagende onderneming in installatie en gebruik.

Experts raden daarbovenop ook aan om een cloud-oplossing voor Kubernetes te benutten, zoals Amazon AWS, Google Cloud of Microsoft Azure. Jammer genoeg zijn cloud-oplossingen niet geschikt voor het aanleren van Kubernetes binnen een opleiding.

1.1.1 Het probleem met cloud based solutions

Er zijn vele platformen en tools die gebruikt kunnen worden om een Kubernetes environment op te stellen. De meest gebruikte opties zijn cloud-based oplossingen via Amazon AWS, Microsoft Azure, en Google cloud. Het probleem met Kubernetes cloud services als leerplatform is dat de meesten geen gratis oefenpakket aanbieden. Diegenen die dat wel doen zullen kosten aanrekenen zodra de gebruiker deze gratis limiet overschrijdt. Dit is uiteraard een probleem voor studenten. Wanneer ze per ongeluk hun limiet bereiken, wil dat zeggen dat ze kosten aangerekend zullen krijgen.

1.1.2 alternatieve lokale oefenplatformen voor kubernetes

Om te vermijden dat studenten ongewilde kosten oplopen, is het dus zinvol om naar een alternatieve oplossing te zoeken. Er zijn vele tools en oplossingen beschikbaar die men hiervoor kan gebruiken. Deze oplossing zal dus gratis en lokaal moeten zijn maar moet ook zinvol zijn voor studenten Informatica als leermiddel.

Een lokale oplossing zal ook zeker en vast aan volgende vereisten moeten doen voordat het als een geschikte oplossing kan gezien worden:

- Het systeem moet dus volledig lokaal kunnen draaien op de eigen laptop om kosten bij cloud-platformen te voorkomen.
- De tool moet functioneel zijn met de systeemvereisten die door de opleiding wordt aanbevolen.
- Studenten moeten zo snel mogelijk in staat zijn om de functionaliteit van Kubernetes te kunnen beginnen ontdekken, hiervoor moet de installatie en configuratie zoveel mogelijk geautomatiseerd worden.
- Het systeem moet toelaten de meest gangbare functionaliteiten van Kubernetes uit te proberen: containers opstarten en afsluiten, applicaties uitrollen, netwerkconfiguratie, logging en monitoring, rolling updates, autoscaling, blue/green deployments, omgaan met stateful data (bv. database), enz.

1.2 Onderzoeksvraag

1.2.1 Hoofdonderzoeksvraag

Hoofdstuk 1.1 geeft aan dat er onaanvaardbare risico's en beperkingen zijn met clouds en dus moet er gezocht worden naar een lokaal alternatief om Kubernetes aan te leren. Daarnaast moet deze lokale oplossing ook voldoen aan de functionele vereisten die opgesomd werden.

Wanneer dit als onderzoeksvraag voorgesteld word, kan het de volgende vorm aannemen:

- Welke Kubernetes oplossing is het meest geschikt om als lokaal leerplatform te dienen voor studenten Informatica?

1.2.2 Deelvragen

Als gevolg van het beantwoorden van de onderzoeksvraag en door een stand van zaken op te bouwen rond Kubernetes als lesonderwerp zal er ook naar een antwoord gezocht worden naar enkele deelvragen:

- Wat zijn de relevante verschillen tussen deze tools?
- Hoe ervaren studenten een gesimuleerde kennismaking in een testopstelling?

1.3 Onderzoeksdoelstelling

Het doel van deze bachelorproef is om onderzoek te doen naar de verschillende mogelijkheden die Kubernetes lokaal kunnen opstellen en die als leerplatform kunnen dienen. Daaruit zal besloten dan worden welke tools een meerwaarde kunnen betekenen voor de opleiding Informatica. Dit betekent dat dit onderzoek niet alleen naar de technische mogelijkheden van de verschillende Kubernetes tools zal kijken, maar ook naar hun educatieve mogelijkheden. Hieruit zal er als bijkomend resultaat ook een werkend prototype afgeleverd worden met een handleiding en instructiedocument, afgetoetst voor een student Toegepaste Informatica.

1.4 Hypothese

Deze hypothese wordt opgesteld voordat er diepgaander onderzoek zal gebeuren en zal enkel rekening houden met de gegevens die verkregen waren tijdens het opstellen van het voorstel van deze bachelorproef. Dit voorstel is terug te vinden als bijlage.

- De meest geschikte tool om Kubernetes aan te leren zal voornamelijk afhangen van de gebruiksvriendelijkheid van de tool interface en zal niet afhangen van de verschil in prestatie en de onderdelen van de tool.

1.5 Opzet van deze bachelorproef

Om een antwoord te vinden op onze onderzoeksvragen zal deze bachelorproef als volgt te werk gaan:

Hoofdstuk 2: zal de huidige stand van zaken van Kubernetes als container orchestration tool weergeven gebaseerd op een literatuurstudie.

Hoofdstuk 3: zal de requirements weergeven. Hierin worden ook de onderzoekstechnieken uitgelegd waarmee er een antwoord op de hoofd- en deelvragen zal gevonden worden.

Hoofdstuk 4: zal de long list stapsgewijs onderwerpen aan de verschillende requirements. Dit zorgt ervoor dat de long list ingekort wordt tot een shortlist die aan alle vereisten voldoet.

Hoofdstuk 5: zal de proof of concept voorleggen. Deze PoC zal aangemaakt worden met behulp van de oplossing of één van de oplossingen indien er meerdere zijn.

Hoofdstuk 6: zal de resultaten van een gebruikerstest met de proof of concept aantonen als bijkomende informatie.

Hoofdstuk 7: bevat een conclusie voor dit onderzoek en tracht een antwoord te geven op de hoofd-en deelonderzoeksvragen.

2. Stand van zaken

2.1 Wat is virtualisatie

2.1.1 Geschiedenis van virtualisatie

Virtualisatie is een geraamte (ook gekend als een “framework”) of methodologie om computatie kracht op te delen in meerdere afgezonderde omgevingen. Dit is mogelijk door het toepassen van één of meerdere concepten of technologieën zoals hardware en software partitionering, “time-sharing”, gedeeltelijke of volledige machine simulatie, emulatie, enzovoort. (Singh, 2004)

In het begin was er CTSS, de “compatible Time-Sharing System”. Ontworpen in het Massachusetts Institute of Technology in Cambridge, Massachusetts, afgekort als MIT, onder toezicht en leiding van Professor Fernando Corbato. Één van de CTSS programmeurs was Robert Creasy, die de uiteindelijke leider zal worden van het CP-40 project. Een vroege versie van CTSS was eerst gedemonstreerd in November 1961. (Varian, 1991)

Time-Sharing is het delen van de computatie-kracht van een machine over meerdere gebruikers tegelijkertijd via “multiprogramming” en “multitasking” (Oracle, 2011)

CTSS was ontworpen op IBM processoren, tijdens de ontwikkeling begon Professor Fernando Corbato problemen tegen te komen waar ze overtuigd van waren dat de oplossing voor deze problemen beter opgelost werden door hardware-verandering dan software-verandering. Dit moedigde IBM aan om hun processoren aan te passen. Om die reden, tegen dat CTSS in definitieve productie was in 1963, waren Corbato en zijn team in staat om van CTSS een model technologie te maken voor time-sharing systemen. (Varian, 1991) De nood voor time-sharing was wel niet zonder moeilijkheden. IBMers in het MIT Liaison

kantoor en het Cambridge branch kantoor waren sterke voorstanders van time-sharing en rapporteerden daarom regelmatig vooruitgang naar IBM System/360 van de nieuwe time sharing technologie. In realiteit had dit echter het tegenovergestelde effect van wat ze hoopten te bereiken met deze inlichtingen. In deze tijd was er binnen IBM zelf een grote oppositie voor deze technologie, waren velen ervan overtuigd dat time-sharing nooit een meerwaarde zal betekenen en dat de wereld in plaats daarvan snellere batch processing nodig had. Hierdoor, op april 7, 1964, IBM kondigde hun System/360 mainframe aan zonder “address translation hardware (HAT)”.(Varian, 1991)

address translation hardware layers beheert de hardware dat verantwoordelijk is voor het mappen van virtueel geheugen naar fysiek geheugen. (Varian, 1991)

In de herfst van 1964 vonden de onderzoekers in MIT zich ineens in de positie om op zoek te gaan naar een project om verder mee te gaan. Norm Rasmussen, een werknemer bij IBM, zag het potentieel van een degelijk time-sharing systeem en wist dat IBM één nodig zal hebben voor de S/360 mainframe. Norm besloot om zelf initiatief te nemen om een time-sharing systeem te bouwen samen met Bob Creasy als projectleider, wat uiteindelijk gekend zou worden als het “CP-40 Project”. CP-40 was klaar voor productie in Januari 1967. Verder werd er nog een herimplementatie gedaan van CP-40 genaamd CP-67 versie 1, die uitgebracht was in mei 1968 en verder onderhouden en uitgebracht tot versie 3.2 in 1972. (Varian, 1991)

In 1970 IBM kondigde hun volgende mainframe versie aan genaamd S/370, opnieuw zonder address translation hardware. Tijdens een samenkomst verklaarde IBM dat ze hoopten dat de klanten geen virtuele machines nodig zouden hebben. Ondanks deze bewering werd een S/370 versie van CP toch ontwikkeld. Dit project genaamt VM/370 release 1 werd afgewerkt in 1972. Hierdoor werd VM/370 het eerst operating systeem dat virtuele machines ondersteunde op het S/370 mainframe platform. (Varian, 1991)

Rond deze tijd was VM zich snel aan het verspreiden en de opleiding van nieuwe VM systeemprogrammeurs was een probleem aan het worden dat geadresseerd moest worden, samen met de beperkte hulplijnen voor VM support. Een groot deel van de steun voor nieuwe VM installaties moest helaas komen van de gebruikersgemeenschap. Vele IBM afdelingen bleven openlijk vijandig tegenover VM als technologie en velen hiervan ontmoedigden klanten om VM te installeren.(Varian, 1991)

De geboorte van “VMSHARE electronic conference” in 1976 was de belangrijkste stap die de VM gemeenschap genomen had om ondersteuning of support for VM aan te bieden voor gebruikers en ze te verenigen.(Varian, 1991)

De geboorte van VNET, IBMs intern network verenigde en versterkte de VM gemeenschap binnen IBM op dezelfde manier dat VMSHARE de externe gemeenschap samenbracht. In Maart 1979 rapporteerde IBM dat VNET 239 Systemen verbond binnen één netwerk en zorgde ervoor dat IBM zeer vroeg intergeconnecteerd was.(Varian, 1991)

De jaren ‘80 was het punt waar het aantal VM installaties dramatisch begon te groeien en zag de geboorte van bedrijven die zich specialiseerden in de productie van VM software en applicaties hiervoor. Voornamelijk omdat IBM eindelijk VM erkende als een belangrijke

technologie. Doorheen de het merendeel van de jaren 80 had IBM zoveel voor VM gedaan dat de gemeenschap voornamelijk reactief waren in plaats van proactief. (Varian, 1991)

Tijdens de jaren '90 werd de ontwikkeling van VM meer en meer gedecentraliseerd en weg van IBM. In 1991 IBM kondigde OS/2 virtual DOS machine (VDM) met ondersteuning voor x86 virtual 8086 mode, wat in staat was om DOS/Windows te virtualiseren en andere 16-bit operating systemen. In 1997 werd de eerste versie van een virtuele PC voor het Macintosh platform gelanceerd door Connectix en was toen de enige virtuele machine-oplossing voor Macintosh. (Singh, 2004)

In 1999 introduceerde VMware hun VMware Workstation. Dit product had een 'hosted architecture': deze architectuur heeft een host operating system nodig zoals Windows of Linux. Om in staat te zijn om de complexe mix van performantie, draagbaarheid en eenvoudigheid voor implementatie te optimaliseren, diende VMware Workstation als een virtuele machine-monitor en tegelijkertijd als een applicatie die functioneert bovenop de host OS. (Singh, 2004) Als virtualization gedefinieerd is als "het in staat zijn om meerdere operating systemen te draaien op dezelfde host computer", dan is de hypervisor de essentiële component in de virtualisatie stack. Een hypervisor of Virtual Machine Monitor (VMM) creëert een virtueel platform op de host computer waarop meerdere gast operating systemen kunnen draaien en gemonitord worden. (Oracle, 2011)

In het begin van de 21ste eeuw zag de VM sector hun eerste evoluties om de fundering te worden voor wat er vandaag de dag gedefinieerd wordt als virtualisatie. In 2001 VMware bracht hun eerste x86 server virtualisatie product uit. Het jaar 2003 zag de eerste open-source x86 hypervisor Xen. In juli 2006 bracht VMware server uit op de markt, een gratis virtualisatieproduct dat op machine-niveau virtualiseert. Vroeg in 2007 bracht innoTek VirtualBox open source editie uit, het eerste professionele open source PC virtualisatie product. Vanaf 2010 zag virtualisatie een grote verschuiving naar cloud gebaseerde virtualisatie met AWS, Google Cloud en Azure. Data centers gebruiken virtualisatie om abstractie te creëren van de fysieke hardware, maakt grote geaggregeerde poelen aan van computer middelen zoals CPU, geheugen, disks, file opslag, applicaties en netwerk infrastructuur, en bieden deze middelen aan in de vorm van schaalbare virtuele machines. (Oracle, 2011)

Cloud virtualisatie legt meer en meer nadruk op schaalbaarheid, containers met docker en uiteindelijk op container orkestratie platformen zoals Docker Swarm en Kubernetes.

2.1.2 de 3 tijdperken van virtualisatie

Samengevat kan de evolutie van virtualisatie verdeeld worden in 3 tijdperken: het traditionele tijdperk, het virtualisatie tijdperk en het momentele container tijdperk.

De meest traditionele vorm van applicatie-deployment was door applicaties rechtstreeks op een fysieke server te laten draaien, via deze manier was er geen mogelijkheid om beperkingen in te stellen in verband met hoeveel geheugen of processorcapaciteit een applicatie toegelaten was te gebruiken, en dit zorgde voor ongewilde competitie voor deze

middelen tussen verschillende applicaties. De enige oplossing was om iedere applicatie op een verschillende fysieke server te laten draaien, maar die was niet schaalbaar en bovendien zeer kostelijk voor de meeste bedrijven. (Kubernetes, 2020)

De oplossing hiervoor was virtualisatie of virtuele machines. Deze manier van deployment stelde een bedrijf ertoe in staat om meerdere virtuele machines op één enkele fysieke CPU te laten draaien. Virtualisatie isoleert applicaties van elkaar en vermijdt dat een applicatie toegang krijgt tot de gegevens van een andere applicatie. Het virtualiseren van applicaties geeft meer controle over de benodigde geheugen en processorcapaciteit en is schaalbaarder omdat er eenvoudig applicaties kunnen toegevoegd worden, bijgewerkt worden, en hardware kosten verminderd. Elke virtuele machine bevat alle componenten zoals een eigen operating system bovenop de gevirtualiseerde hardware. (Kubernetes, 2020)

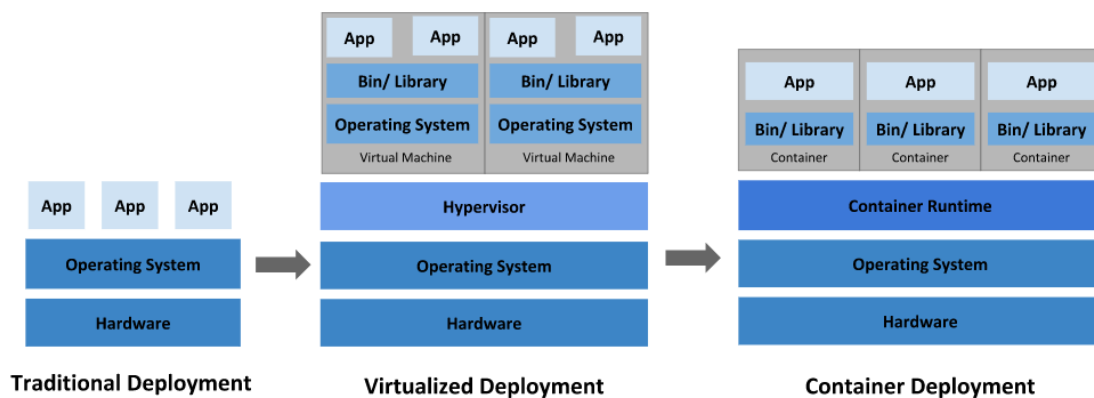
Na de virtuele machines kwamen containers.

Containers zijn zeer gelijkaardig aan virtuele machines. Het verschil tussen deze twee technologieën is dat containers minder strenge isolatie-eigenschappen bevatten in vergelijking met Virtuele machines. Dit zorgt ervoor dat alle containers op een fysieke server hetzelfde operating system delen, waarmee containers als “lightweight” beschreven worden. Net zoals virtuele machines hebben containers hun eigen filesysteem, geheugen en processor ruimte. Doordat ze gescheiden zijn van de onderliggende infrastructuur, zijn ze zeer mobiel en betrouwbaar over verschillende cloud platformen en OS distributies. (Kubernetes, 2020)

Containers mogen zeker niet gezien worden als een ideale verbetering tegenover virtuele machines. Hoewel containers vele voordelen hebben vergeleken met VM's, zijn er toch enkele belangrijke verschillen die moeten blootgelegd worden.

Containers zijn minder geïsoleerd in vergelijking met virtuele machines. Wanneer isolatie verminderd zal dus ook de veiligheid verminderen. Aangezien containers Dezelfde OS gebruiken en kernel wordt het dus makkelijker om toegang te krijgen tot deze containers. (Gupta, 2015)

Een container maakt gebruik van het operating system van de fysieke machine, terwijl een virtuele machine zijn eigen OS bevat. Dit kan in theorie moeilijkheden veroorzaken aangezien een container beperkt is tot de OS van de fysieke machine en dus enkel applicaties kan bevatten die ondersteund worden door deze OS. Vele containers gebruiken VM's als de host operating system in plaats van het rechtstreeks op de hardware te laten uitvoeren, vooral voor containers die in de cloud draaien. (Gerend, 2019)



Figuur 2.1: Voorstelling van de evolutie van applicatie deployment (Kubernetes, 2020)

2.2 Wat is/hoe werkt/waarom kubernetes?

2.2.0.1 Wat is Kubernetes?

Kubernetes is een open-source systeem ontwikkeld door Google dat instaat voor geautomatiseerde inzet of “deployment”, uitbreiding en beheren van gecontaineriseerde applicaties. (Kubernetes, 2020)

Het is een technologie ontworpen om de volledige levenscyclus te beheren van de gecontaineriseerde applicaties en diensten door gebruikt te maken van methodes die voorspelbaarheid, schaalbaarheid en hoge betrouwbaarheid aanbieden. Een gebruiker kan zelf definiëren hoe applicaties moeten draaien en op welke manier dat verschillende applicaties met elkaar of met de buitenwereld kunnen communiceren. Kubernetes staat een gebruiker toe om lopende diensten te vergroten of in te perken, “continuous delivery” van updates, verkeer tussen verschillende applicatie versies laten afwisselen voor testen en het terugdraaien van foutieve deployments. (Ellingwood, 2018)

Kubernetes is geschikt voor horizontaal schaalbare ‘microservices’ applicatie architecturen. Het vergemakkelijkt het gebruik, bereikbaarheid en de ontdekking van containers in een container cluster. (Paris, 2014)

2.2.0.2 Wat is Kubernetes niet

Kubernetes is geen traditioneel PaaS (Platform as a Service) systeem. Aangezien Kubernetes op het container niveau werkt en niet op het hardware niveau, biedt het enkele toepasbare functies aan vergelijkbaar met wat PaaS aanbiedt zoals deployment, scaling, load balancing, user geïntegreerde logging, monitoring en alarm oplossingen. Maar Kubernetes is niet één geheel deze functies zijn optioneel en modulair. Kubernetes biedt de bouwstenen voor een developer platform aan, maar behoudt gebruiker keuze en flexibiliteit waar het belangrijk is. (Kubernetes, 2020)

Kubernetes:

- Beperkt niet de hoeveelheid types van ondersteunde applicaties. Als een applicatie in een container kan draaien zou het ook moeten kunnen draaien op kubernetes.
- Lanceert geen source code en bouwt applicatie niet. Continuous Integration, Delivery, and Deployment (CI/CD) workflows worden bepaald door de organisatie voorkeuren en technische vereisten.
- Biedt geen services op applicatie niveau zoals middleware, data-verwerkende frameworks, databanken of caches als ingebouwde services. Deze componenten kunnen op Kubernetes draaien en/of kan toegankelijk gemaakt worden door applicaties op kubernetes via mechanismen zoals de “Open Service Broker”.
- Kubernetes verplicht geen logging, monitoring en alarm oplossingen. Kubernetes biedt enkele integraties aan als proof of concept en opties om gegevens te verzamelen en exporteren.
- Biedt geen taal/systeem configuratie aan zoals Jsonnet. Kubernetes biedt een declaratieve API aan dat kan aangesproken worden.
- Biedt of adopteert geen enkele uitgebreide systeem configuratie, onderhoud management of zelfherstellende systemen.
- Kubernetes is niet zomaar een orkestratie systeem. In feite elimineert het de nood voor orkestratie. De technische definitie van orkestratie is de uitvoering van een bepaalde “workflow”: doe A dan B en daarna C. in contrast, Kubernetes bestaat uit een set onafhankelijke control processen die continu de huidige staat vooruit duwen naar een gegeven gewenste staat. Voor Kubernetes maakt het niet uit hoe het van A naar C kan gaan. Gecentraliseerd beheer is ook niet noodzakelijk. Dit resulteert in een systeem dat makkelijker is om te gebruiken, krachtiger is, meer robuust, fout bestendig en uitbreidbaar. (Kubernetes, 2020)

2.2.1 Hoe werkt Kubernetes?

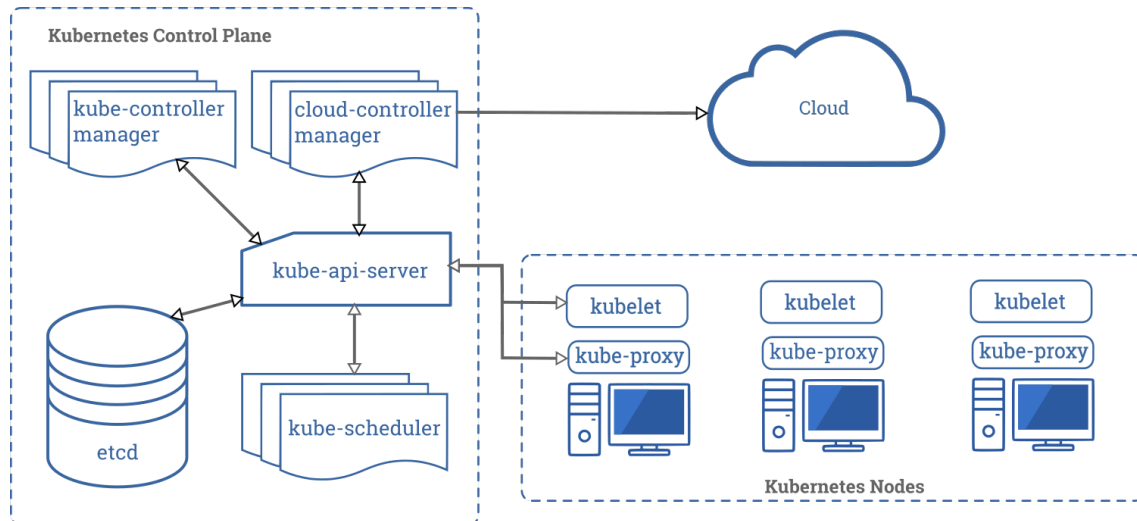
De technologie dat als fundering dient voor Kubernetes is Container virtualisatie.

In tegenstelling tot virtuele machines zijn containers reproduceerbare gestandaardiseerde omgevingen, deze omgevingen hebben het voordeel dat ze slechts éénmalig moeten aangemaakt worden en achteraf vele keren kunnen gebruikt worden. (Moilanen, 2018)

Wanneer Kubernetes geactiveerd is zal er een cluster aangemaakt worden. Een cluster bevat een collectie werkmachines die binnen Kubernetes “nodes” genoemd worden. Deze nodes voeren container applicaties uit. Iedere cluster heeft minimum 1 node. (Kubernetes, 2020)

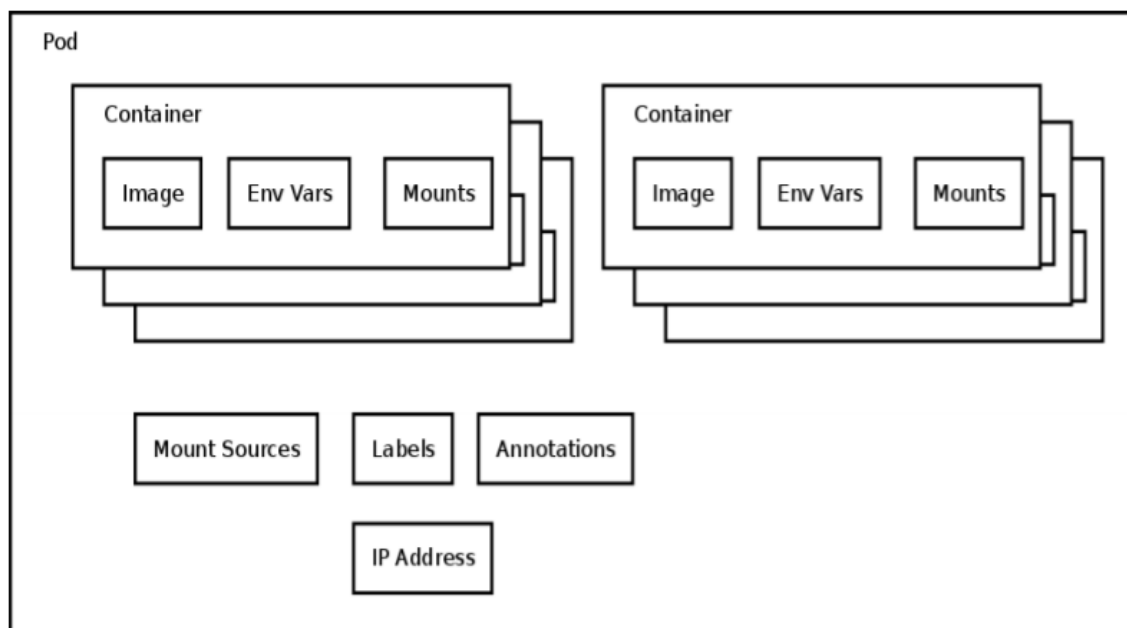
Een node bevat “pods”, een pod is een collectie containers die op dezelfde machine draaien. De nodes en pods binnen de cluster worden beheerd door de “control plane”. In een productieomgeving wordt de control plane functionaliteit vaak verdeeld over meerdere computers en een cluster bevat meestal meerdere nodes om een betere fout-tolerantie en betrouwbaarheid te bekomen. (Kubernetes, 2020)

Objecten binnen Kubernetes waarvan vaak pods kunnen worden gegroepeerd door middel



Figuur 2.2: Een diagram dat een Kubernetes cluster voorstelt (Kubernetes, 2020)

van een label. Een label is een sleutel-waarde paar. Elk object kan met meerdere labels verbonden zijn en elke Label kan aan meerdere objecten vastgemaakt worden. Labels zijn belangrijk voor bepaalde concepten zoals onder andere: replication controller, replica sets en Label selectors. (Kubernetes, 2020) Label selectors worden gebruikt om objecten te selecteren op basis van hun labels. Hiermee kan gespecificeerd worden naar beide de key van een label en de waarde daarvan. Bijvoorbeeld: Role= WebServer Dit zal alle objecten selecteren die die Role key bezitten waarvan de waarde “webserver” is. (Sayfan, 2017)



Figuur 2.3: Voorstelling van een Kubernetes Pod en de inhoud van een pod. (Paris, 2014)

2.2.1.1 Control plane componenten

De componenten van het control plane ook gekend als master bepalen de globale beslissingen voor de cluster(bijvoorbeeld scheduling). Het staat ook in voor het detecteren van cluster events en het reageren op deze events. Een voorbeeld van een cluster event is bijvoorbeeld het opstarten van een nieuwe pod wanneer de vereisten hiervoor niet voldaan zijn. (Kubernetes, 2020)

Zoals er geschreven staat binnen de officiële documentatie van (Kubernetes, 2020) zijn dit de verschillende componenten van de control plane:

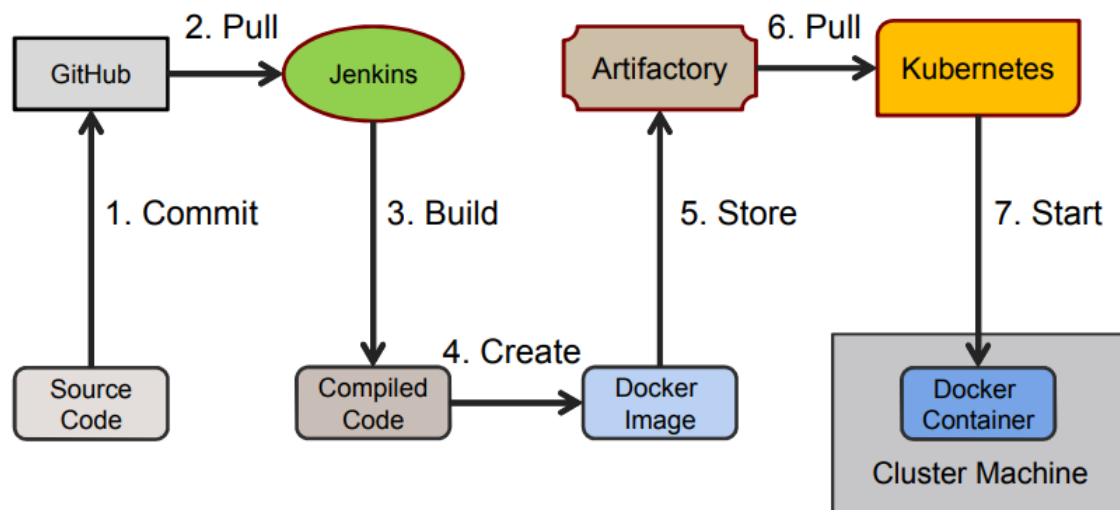
- Kube-apiserver: de API server weergeeft de Kubernetes API. Dit dient als front-end voor het Kubernetes control plane.
- etcd: een consistente en bereikbare sleutel waarde opslag dat door Kubernetes gebruikt wordt als een opslag voor configuratie data voor de cluster.
- kube-scheduler: control plane component dat uitkijkt voor nieuwe gecreëerde pods met nog geen toegewezen node. Het selecteert een node die dan kan draaien op één van deze pods. Enkele factoren waar rekening mee gehouden worden voor scheduling keuzes zijn: individuele en collectieve vereisten, hardware/software/policy beperkingen, deadlines en werkonderbrekingen.
- kube-controller-manager: voert controller processen uit. Controller processen overzien de gedeelde staat van de cluster door de API server en voert aanpassingen uit om de momentele staat van de cluster naar de verwachte staat te veranderen. De verschillende controllers zijn:
 - Node controller: verantwoordelijk voor het opmerken en reageren wanneer een node offline gaat.
 - Replication controller: is samen met replica sets verantwoordelijk voor het onderhouden van het correcte aantal pods voor elke replication controller object in het systeem. Kubernetes zal proberen garanderen dat er altijd evenveel pods draaien gelijk aan de hoeveelheid gespecificeerd in een replication controller of een replica set. Wanneer het aantal onder de verwachte hoeveelheid zakt zullen er nieuwe instanties opgezet worden.
 - Endpoint controller: vult de endpoint objects. (joins Services en Pods)
 - Service Account en Token controllers: creëert standaard account en API toegang tokens voor nieuwe namespaces.
- cloud controller manager: Voegt cloud-specifieke control logic in. Deze controller stelt de cluster ertoe in staat om zichzelf te verbinden met een cloud provider hun eigen API. De cloud controller manager gebruikt enkel de controllers van een toegewezen cloud provider. Clusters die lokaal of als leeromgeving draaien hebben geen cloud controller manage

2.2.1.2 Node componenten

Node componenten zijn de onderdelen van iedere node en zijn verantwoordelijk voor het onderhoud van draaiende pods en de “Kubernetes runtime environment”.

Zoals er geschreven staat binnen de officiële documentatie van (Kubernetes, 2020) zijn dit de verschillende componenten van de control plane:

- Kubelet: Draait op elke node in de cluster en zorgt ervoor dat alle containers binnen een pod aan het draaien zijn.
- Kube-proxy: Is een netwerk proxy dat op elke node draait in de cluster, en implementeert een deel van de Kubernetes Service concept. Het beheert de netwerk regels binnen elke node. Deze netwerk regels laten netwerk communicatie toe naar de pods van de netwerk sessies binnen de cluster.
- Container runtime: Is de software dat verantwoordelijk is voor de containers actief te laten draaien. Kubernetes kan verschillende container runtimes gebruiken: Docker, Containerd, CRI-O en elke implementatie van de Kubernetes CRI(Container runtime interface).



Figuur 2.4: Vereenvoudigde Voorstelling van de weg die een source code applicatie aflegt tot in een Kubernetes pod. (Stapleton, 2016)

2.2.1.3 Kubernetes objecten

Kubernetes objects zijn persistente entiteiten binnen het Kubernetes systeem, deze entiteiten beschrijven de staat van een bepaalde cluster. Ze beschrijven:

- Welke gecontaineriseerde applicaties er draaien en op welke node.
- De hoeveelheid CPU en RAM beschikbaar voor deze applicaties
- De policies die bepalen hoe een applicatie zich moet gedragen op vlak van heropstartingen, aanpassingen en fout bestendigheden.

Bijna alle Kubernetes objecten bevatten twee object velden: de spec en de status. De spec is een beschrijving van de verwachte karakteristieken van het object, met andere woorden hoe het object zich zou moeten gedragen, dit wordt de “desired state” genoemd. Het status veld beschrijft de momentele staat van het object ook gekend als “current state”, het control plane zal continu elk object beheren zodat de current state overeenkomt met de desired state. (Kubernetes, 2020)

Om een object creëren in Kubernetes is er een object spec dat de desired state beschrijft en een object naam nodig. Indien de Kubernetes API gebruikt wordt om een object aan te maken dan zal de API request deze informatie in JSON formaat bijvoegen. Vaak wordt deze informatie aan kubectl als .yaml meegegeven. Kubectl vertaalt deze informatie naar JSON wanneer er een API request gemaakt wordt. (Kubernetes, 2020)

Onderstaande afbeelding toont een voorbeeld .yaml bestand dat de vereiste velden en het object spec aantoont.

```
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.2
          ports:
            - containerPort: 80
```

Figuur 2.5: Voorbeeld .yaml bestand voor object creatie (Kubernetes, 2020)

2.2.2 Waarom wordt Kubernetes gebruikt?

Kubernetes is een goede manier op applicaties als bundel te laten draaien. In een productieomgeving moeten containers beheert worden zodat de applicaties ononderbroken beschikbaar zijn voor de gebruiker. Een voorbeeld hiervan is wanneer er een container stilvalt, moet er een andere container opgestart worden. Het zou eenvoudiger zijn moest deze taak uitgevoerd worden door een systeem. (Kubernetes, 2020)

Dit is één van de krachten van Kubernetes, het biedt een framework aan om systemen op een fouttolerante manier te laten draaien. Het zorgt ervoor dat de applicaties zich op een gepaste manier kunnen uitbreiden en maakt gebruik van bepaalde deployment patronen. (Kubernetes, 2020)

Zoals er geschreven staat binnen de officiële documentatie van (Kubernetes, 2020), Kubernetes kan de volgende services aanbieden:

- Service discovery and load balancing: Kubernetes kan een container blootleggen via de DNS naam of via het eigen IP adres. Wanneer gebruikers verkeer te hoog wordt kan Kubernetes dit netwerk verkeer verspreiden zodat de deployment stabiel blijft.
- Storage orchestration: Kubernetes laat een beheerder toe om automatisch een opslagsysteem naar keuze te monteren (to mount).
- Automated rollouts and rollbacks: Een beheerder kan een “gewenste staat” beschrijven voor Kubernetes deployed containers. Hiermee kan het de actuele staat van de opstelling telkens aangepast worden naar de gewenste staat wanneer deze staten niet overeen komen.
- Automatic bin packing: Een beheerder kan een cluster dat uit nodes bestaat die gecontaineriseerde tasks uitvoeren aan Kubernetes aanbieden, de beheerder kiest zelf hoeveel CPU en RAM elke containers nodig heeft. Kubernetes kan daarna zelf beslissen hoe het deze containers in de verschillende nodes zal opdelen om zo efficiënt mogelijk te zijn met CPU en RAM.
- Self-healing: Kubernetes herstart zelf gefaalde containers, vervangt containers en beëindigd containers die niet reageren op de “user-defined health check”.
- Secrets and configuration management: Kubernetes laat een beheerder toe om gevoelige gegevens zoals wachtwoorden en SSH sleutels op te slaan. Deze sleutels kunnen kunnen toegepast en aangepast worden zonder dat container images moeten herbouwd worden en zonder ze bloot te moeten leggen in de stapel configuratie.

Hoewel Kubernetes ontworpen werd door google mag het zeker niet bekeken worden als een product dat enkel geschikt is voor enorme internet service bedrijven. Kubernetes is uitgegroeid tot een groeiende open source gemeenschap en dit zorgt ervoor dat de technologie meer toegankelijk is voor bedrijven ongeacht hun grote. (Hightower, 2019)

2.3 Docker swarm en Kubernetes

Docker is in 2020 één van de marktleiders in containerisering en is daarmee één van de meest populaire container technologieën. Docker Swarm is een container orchestration platform gelijkaardig aan Kubernetes. Deze sectie zal meer inzicht geven over de gelijkenissen en verschillen tussen deze twee platformen. Het nut hiervan is om een beter beeld te creëren waarom er specifiek voor Kubernetes gekozen werd en niet docker swarm.

2.3.1 Docker als de nieuwe marktleider

De cloud wereld in 2013 was een zeer andere omgeving vergeleken met 2020. Bash scripts waren nog maar net plaats aan het maken voor declaratieve configuratie van IaaS met systemen. Het idee van orkestratie en zeker container orkestratie bestond wel, maar was kleinschalig en bestond nog niet in de cloud en zeker niet in enterprise. (Burns, 2018)

Docker veranderde alles hieraan. Door het populariseren van lichte container runtime en door hoe eenvoudig docker het maakt om applicaties in te pakken, distribueren en in te zetten. Zonder de verschuiving van het cloud development perspective dat docker veroorzaakte zou Kubernetes niet bestaan. (Burns, 2018)

Docker is een tool dat gemakzucht belooft bij het distribueren van een applicatie, het inzetten van deze applicatie op schaal in eender welke omgeving en het stroomlijnen van het werkproces van agile software organisaties. (Matthias, 2015)

2.3.2 De opkomst van orkestratie met Kubernetes en Docker swarm

Werknemers bij Google wouden met behulp van docker een cloud native applicatie oplossing meer toegankelijk maken voor een groter publiek. De conclusie was dat Docker die de nadruk legt op “single machines” niet de volledige oplossing was. Google had nood aan een open-source orchestrator that containers kon inzetten en onderhouden in grote hoeveelheden over een vloot van machines. (Burns, 2018)

Het waren die realisaties dat leidde tot de ontwikkeling van een prototype systeem dat later bekend zou worden als Kubernetes. (Burns, 2018)

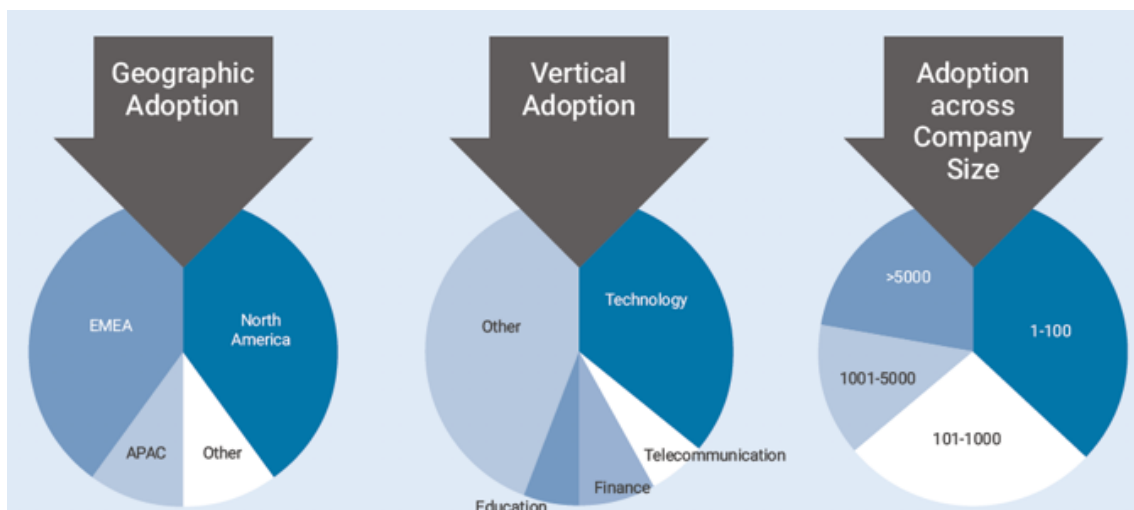
In 2015 bracht Docker de eerste beta van hun Swarm tool uit. Het idee achter Swarm is om een enkele interface naar de docker client tool to hebben dat ondersteunt wordt door een hele cluster in plaats van één enkele Docker daemon. Swarm is gefixeerd op het clusteren van computing resources vergelijkbaar in functie met Kubernetes. (Matthias, 2015)

2.4 Het nut en belang van Kubernetes in het lesprogramma

Deze sectie is bedoeld om meer duidelijkheid scheppen waarom het belangrijk is voor de studenten Informatica van vandaag om Kubernetes te begrijpen voor ze de jobmarkt betreden. Vele van deze argumenten kunnen toegepast worden op container orchestration tools in het algemeen maar om binnen de scope van dit onderzoek te blijven zal de nadruk op Kubernetes gelegd worden.

2.4.1 Marktonderzoek door Rancher Labs

Om een beter beeld te krijgen over Kubernetes binnen de professionele wereld heeft Rancher Labs een onderzoek afgelegd dat bestond uit 1,106 respondenten van grote en kleine bedrijven. Dit onderzoek werd afgelegd in 25 verschillende sectoren zoals technologie, financiën, telecommunicatie, onderwijs, overheids administratie en de gezondheidszorg. De respondenten waren bijna gelijk verdeeld tussen Noord-Amerika en de EMEA(Europa, het Midden-Oosten, en Afrika) + APAC(Regio Azië-Stille Oceaan). (Smails, 2019)



Figuur 2.6: Rancher Labs enquête verdeling Container adoptie in professionele productie omgevingen.(Smails, 2019)

Het onderzoek gedaan door Rancher Labs stelt volgende resultaten vast die relevant zijn voor deze sectie:

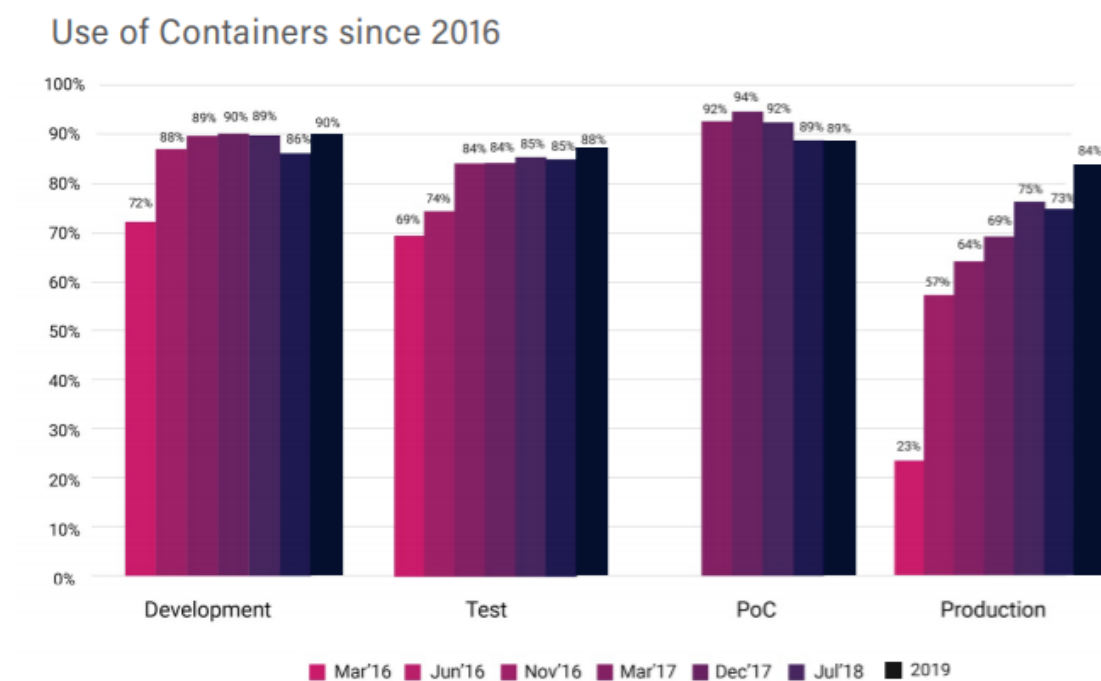
- 85% van de respondenten gebruikten containers in hun productie omgeving
- 90% van de bedrijven die container management systemen gebruiken doen beroep op Kubernetes voor container orchestratie.
- Multi-cluster omgevingen worden steeds vaker gebruikt, 91% van de respondenten draaien meer dan 1 cluster terwijl 45% van diegenen die niet meerdere clusters gebruiken dat wel van zin zijn binnen de komende 6 maanden van wanneer het onderzoek is afgelegd.

2.4.2 Marktonderzoek door Cloud native Computing foundation

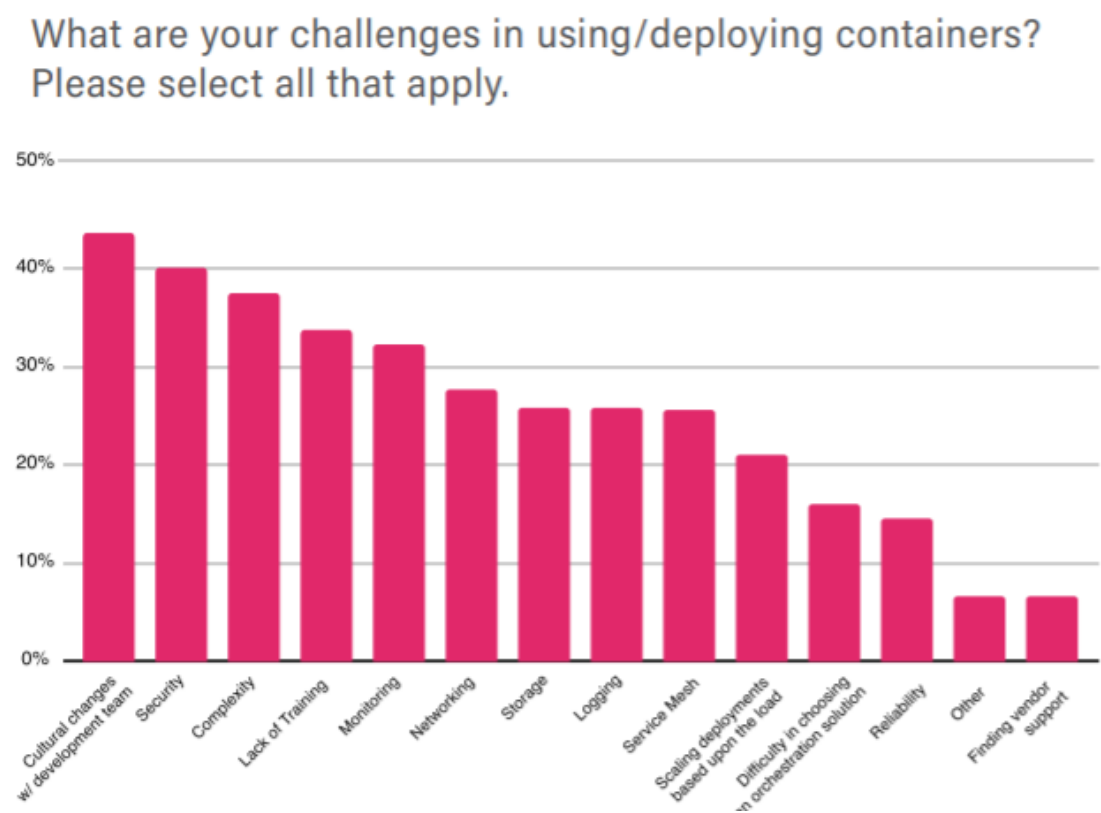
Een ander onderzoek gedaan door Cloud native Computing foundation geeft gelijkaardige resultaten. CNCF deed onderzoek om beter te begrijpen waar en hoe cloud native technologieën werden toegepast. Het onderzoek onderzocht verschillende zaken en vergelijkt deze met onderzoeksresultaten van voorgaande jaren. Verschillende onderwerpen worden ondervraagd zoals release cycles, infrastructuur grote, cloud en het gebruik van containers. Hier zal er enkel naar de resultaten rond containers gekeken worden.

In totaal werden er 1337 respondenten werden voor dit onderzoek ondervraagd tussen september en oktober 2019. De respondenten waren bijna gelijk verdeeld tussen Europa (37%) en Noord-Amerika (38%), gevolgd door Azië (18%). 71% van de respondenten zijn afkomstig van een grote onderneming (100+ werknemers) en 30% van de respondenten zijn afkomstig van zeer grote ondernemingen (5000+ werknemers).(CNCF, 2019) Het onderzoek gedaan door CNCF stelt volgende resultaten vast die relevant zijn voor deze sectie:

- In totaal gebruikten 84% van de respondenten containers in hun productie omgeving.
- Ondanks de vele container management tools gebruikt 89% een vorm van Kubernetes in 2019. In 2018 was dit slechts 83%
- Één van de grootste uitdagingen rond het gebruik van containers is het gebrek aan training of opleiding. In totaal gaven 34% van alle respondenten toe dat dit een hindernis is.



Figuur 2.7: Resultaten CNCF enquête rond container gebruik en de uitdagingen rond containers (CNCF, 2019)



Figuur 2.8: Resultaten CNCF enquête rond container gebruik en de uitdagingen rond containers (CNCF, 2019)

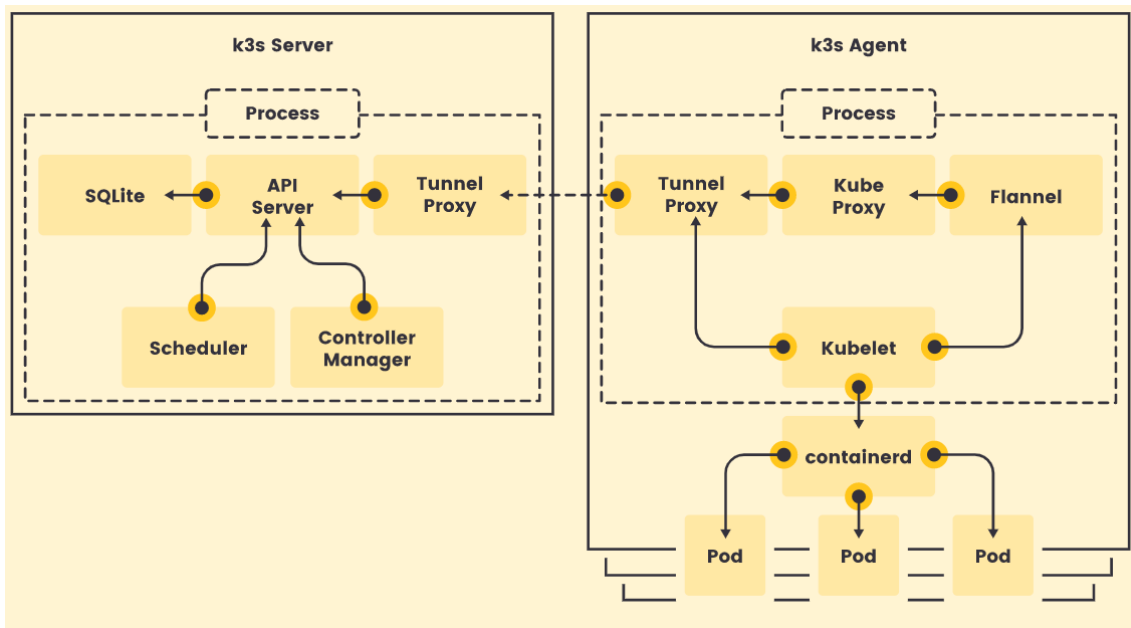
2.4.3 In conclusie

Deze marktonderzoeken tonen aan dat een grote hoeveelheid bedrijven niet alleen containers toepassen, maar ook dat er zich een globale trend aan het plaatsvinden is waarin meer en meer bedrijven Kubernetes in hun infrastructuur gebruiken. Tot slot is één van de grootste uitdagingen om efficiënt met containers te werken het gebrek aan training bij de werknemers.

2.5 Welke Kubernetes oplossingen zijn er

Minikube: Minikube is een open source tool om lokaal een cluster op te zetten met één enkele node binnen een virtuele machine. Het is ontworpen om eenvoudig en snel de functionaliteit van Kubernetes uit te testen binnen een 'lightweight' omgeving. Minikube is beschikbaar op alle OS distributies.

K3: K3 is een gecertificeerde Kubernetes distributie ontworpen voor productie omgevingen zonder toezicht. K3 is ingepakt als een <40MB binary dat het aantal benodigheden verminderd en het aantal stappen beperkt voor een installatie, het opstarten en het auto-updaten van een cluster. (K3, 2020)



Figuur 2.9: Visualisatie van K3s architectuur (K3, 2020)

Kind:

Kind is een tool ontworpen om Kubernetes clusters lokaal te laten draaien op verschillende OS distributies. Het start een Kubernetes cluster binnen een docker container in plaats van een virtuele machine.

MicroK8s:

Microk8 is een tool binnen het Kubernetes ecosysteem voornamelijk bestemd voor Ubuntu servers. Het beschrijft zich als een lightweight, snelle en volledige conforme upstream Kubernetes tool dat clusteren vergemakkelijkt .

Kube-spawn:

Kube-spawn is een tool voor het opzetten van een multi-node Kubernetes cluster op één enkele Linux machine ontworpen voor lokale opstellingen te laten draaien en testen.

Paulczar: container.training:

Container Training is een github opstelling bedoeld als leermiddel aangevuld met slides, scripts, een demo applicatie en code samples. Deze opstelling is ontworpen voor gebruikers die onervaren zijn met docker, containers of een bepaalde programmeertaal. Deze opstelling is ook geschikt voor ofwel een klas omgeving of zelfstudie.

Kelsey Hightower: Kubernetes the hard way :

Kubernetes the hard way is een opstelling die volledig ontworpen is met de intentie om Kubernetes aan te leren. Het is dus geen “lightweight” tool, maar eerder een leermiddel. Deze opstelling is bedoeld voor een publiek dat achteraf een Kubernetes productieomgeving zou moeten ondersteunen. Dit zorgt ervoor dat deze opstelling zelden geautomatiseerde commandos gebruikt die een opstelling vergemakkelijken en dus een hogere complexiteit heeft dan andere opties om te verzekeren dat een gebruiker elk proces begrijpt dat nodig is om een Cluster te laten draaien. hogere complexiteit heeft dan andere opties om te verzekeren dat een gebruiker elk proces begrijpt dat nodig is om een Cluster te laten draaien.

3. Functionele vereisten en methodologie

In dit hoofdstuk wordt er in detail bekeken welke eigenschappen vereist zijn voor de uiteindelijke proof of concept en welke, hoewel optioneel, ook een meerwaarde betekenen. De eigenschappen worden verdeeld in een “must have” lijst waarin elke eigenschap absoluut noodzakelijk is en een “should have” lijst die eigenschappen bevat die niet verplicht zijn, maar die wel de tool of platform in kwestie aantrekkelijker maken dan oplossingen die mogelijks niet over deze eigenschappen bezitten.

Vervolgens wordt er toegelicht hoe het onderzoek is opgesteld. Hierbij worden de gebruikte onderzoekstechnieken besproken en de fasen van het onderzoek toegelicht. Door middel van dit onderzoek kunnen bepaalde observaties waarnemen die een antwoord geven op onze onderzoeksvraag.

3.1 Must have

- De gebruikte oplossing moet volledig gratis zijn.
- De gebruikte oplossing moet lokaal beschikbaar zijn. Dit houdt in dat de opstelling of proof of concept alle nodige functionaliteiten bevat terwijl deze enkel en alleen draait op de laptop van de student door middel van bv.Virtualbox.
- De oplossing moet aan alle functionele verwachtingen voldoen op een laptop die niet meer dan de minimum specificaties bevat opgelegd door de hogeschool.
- De oplossing en bijkomende instructies/documentaties moeten toegankelijk zijn voor het niveau van een 2de of 3de jaars student. Dit houdt in dat de studenten over de volgende basiskennis bezitten:
 - basiskennis Linux en VirtualBox.
- De oplossing is zodanig opgesteld dat het de volgende functies en features kan aantonen:
 - Pods, nodes of containers.
- monitoring van de containers zodanig om inzicht te krijgen in de werking van het systeem.

3.2 Should have

- De proof of concept deployment wordt liefst opgezet via Vagrant en/of Ansible. Een .ova file met gedetailleerde en reproduceerbare installatieprocedure is ook geschikt.
- De mogelijkheid om volgende functies aanwezig te hebben wordt sterk aanbevolen:
 - load balancing en auto scaling

3.3 Nice to have

- Rolling updates functionaliteit
- Blue green deployments functionaliteit

3.4 Onderzoeksmethodologie

Zoals in Hoofdstuk 1 beschreven is, stelt dit onderzoek zich de vraag:

- Welke Kubernetes oplossing is het meest geschikt om als lokaal leerplatform te dienen voor studenten Informatica?

Daarnaast zijn de volgende deelvragen ook van toepassing:

- Hoe ervaren studenten een gesimuleerde kennismaking in een testopstelling?
- Wat zijn de relevante verschillen tussen deze tools?

Om voor deze vragen een objectief antwoord te vinden werd dit onderzoek op een gestructureerde manier verdeeld in verschillende fasen.

1. Het onderzoek werd geadverteerd op verschillende platformen om deelnemers te vinden die als respondenten kunnen dienen.
2. De verschillende opgesomde tools worden met elkaar vergeleken om één of meerdere geschikte oplossingen te vinden.
3. Een proof of concept wordt opgesteld op basis van één van de geschikte oplossingen die voldoet aan alle vereisten en zoveel mogelijk 'should-have' kwaliteiten bevat.
4. De geschikte respondenten zullen de proof of concept krijgen dat vergelijkbaar is met wat studenten Toegepaste Informatica zouden krijgen als kennismaking met Kubernetes. Iedere deelnemer zal opgevolgd worden terwijl ze bepaalde taken uitvoeren binnen de proof of concept.
5. De respondenten krijgen na de testperiode een enquête waarin hun ervaring bevraagd wordt.

3.4.1 Fase 1: Onderzoeksadvertentie en het aanwerven van geschikte kandidaten

Het onderzoek werd geadverteerd op verschillende platformen om deelnemers te vinden die als respondenten kunnen dienen. Gezien we op zoek zijn naar een bepaalde doelgroep van personen, namelijk een individu die overeenkomt of dicht aansluit met het profiel van een 2de of 3de jaars student Toegepaste Informatica, werd de enquête opgesteld om na te gaan of deelnemers de nodige kennis bevatten vergelijkbaar met een student van deze opleiding.

De enquête keek niet strikt na of de respondenten studenten Informatica zijn, maar of dat ze tenminste de minimum basiskennis bevatte. Zo werd er bijvoorbeeld gevraagd naar hun kennis met Virtualbox, Vagrant en basiskennis Linux. Als bijkomende informatie werd er ook om hun leeftijd gevraagd.

3.4.2 Fase 2: Bepalen van de geschikte tools

Op basis van de opgelegde vereisten werden de verschillende oplossing vergeleken om te bekijken welke tools een goede keuze zouden zijn om een proof of concept uit op te stellen.

De long list werd eerst vergeleken met de vereisten in de 'must have' list van hoofdstuk. Dit zijn de verplichte requirements die nodig zijn, dit betekende dat in theorie iedere tool die aan deze requirements voldoet een geschikte tool was. Nadien werden de oplossingen vergeleken met de "should have" lijst. De tools die aan al deze vereisten voldoen bevatte dus ook de optionele vereisten en waren daarom nog aantrekkelijker om te gebruiken.

3.4.3 Fase 3: Opstellen proof of concept

Een proof of concept werd opgesteld. Deze opstelling werd gebouwd op de meest geschikte tool of één van de meest geschikte tools die we uit het vergelijkend onderzoek van fase 2 halen. Deze opstelling voldeed aan alle vereisten en daarnaast ook aan zoveel mogelijke 'should-have' kwaliteiten. De PoC werd zodanig opgesteld om zo snel mogelijk en eenvoudig de belangrijke functies aan te tonen. Om deze opstelling herbruikbaar en gebruiksvriendelijk te maken, werd deze opgesteld binnen virtualbox en kan eenvoudig ingezet worden met behulp van Vagrant, een software die gekend is voor studenten Toegepaste Informatica Systeembeheer.

Tot slot bevatte de bijkomende instructies een korte opdracht die gebruikt werd om een deel van de eind-enquête te kunnen beantwoorden.

3.4.4 Fase 4: Uitvoeren test

De geschikte respondenten kregen de proof of concept dat vergelijkbaar was met wat studenten toegepaste Informatica zouden krijgen als kennismaking met Kubernetes. Iedere deelnemer werd opgevolgd terwijl hij of zij bepaalde taken volbracht met Kubernetes. De volledige test bestaat uit 3 delen:

1. Een introductie voor Kubernetes en een basis uitleg rond bepaalde begrippen en functies.
2. Een "hands-on" labo gedeelte waar de gebruiker eenvoudig en snel de belangrijke functies kan uitproberen en hun effecten kan waarnemen.
3. Een opdrachten-gedeelte waarin ze de verworven kennis van deel 2 moeten toepassen om bepaalde resultaten te bekomen.

Doorheen de test werden eventuele knelpunten, moeilijkheden en opdrachten onderdelen waar er vaak om hulp gevraagd werd werden genoteerd om later aan te kunnen tonen.

3.4.5 Fase 5: Eind-bevraging van de testers

De respondenten kregen na de testperiode een enquête die hun ervaring bevroeg. Met deze enquête werd niet alleen de algemene stemming van de testpersonen bevraagd, maar ook hun eigen mening over de verschillende onderdelen van de test.

4. Resultaten: Vergelijking van de mogelijke oplossingen

Dit hoofdstuk vergelijkt verschillende Kubernetes tools om te bepalen of ze aan de functionele vereisten voldoen. Daarnaast zal er ook gekeken worden naar opmerkelijke eigenschappen die slechts gedeeltelijk te maken hebben met de opgelegde vereisten maar wel een voordeel of nadeel kunnen betekenen voor de tool.

Dit hoofdstuk zal op een afbouwende wijze de selectie oplossingen overlopen. Dit betekent dat opties die niet voldoen aan de belangrijkste vereisten, niet meer verder zullen vergeleken worden.

4.1 De oplossingen

Hier is een lijst terug te vinden van de verschillende oplossingen of leeromgevingen die zullen onderworpen worden aan de requirements. Gezien de hoofdonderzoeksvraag een ruime uitkijk heeft op wat allemaal in aanmerking kan komen voor een geschikte Kubernetes oplossing, wordt er ook gekeken naar cloud oplossingen en diverse Kubernetes leermiddelen om met duidelijkheid aan te tonen waarom deze opties ongeschikt zijn.

1. Cloud-based platform (google cloud, aws, microsoft azure cloud)
2. Minikube
3. K3
4. Kind
5. MicroK8s
6. Paulczar: container.training
7. Kelsey hightower: Kubernetes the hard way.

4.2 vergelijking van de 'must haves'

Must haves zijn vereiste requirements. Oplossingen die niet aan deze verwachtingen voldoen zijn dus ongeschikt als om Kubernetes aan te leren binnen een leeromgeving toegepaste informatica. Dit betekent dat een oplossing niet verder zal bekeken worden zodra één eigenschap gevonden werd waar de oplossing niet aan kan voldoen.

- De gebruikte oplossing moet volledig gratis zijn.
- De gebruikte oplossing moet lokaal beschikbaar zijn. Dit houdt in dat de opstelling of proof of concept alle nodige functionaliteiten bevat terwijl deze enkel en alleen draait op de laptop van de student door middel van bv. Virtualbox.
- De oplossing moet aan alle functionele verwachtingen voldoen op een laptop die niet meer dan de minimumspecificaties bevat, opgelegd door de hogeschool.
- De oplossing en bijkomende instructies/documentaties moeten toegankelijk zijn voor het niveau van een 2de of 3de jaars student. Dit houdt in dat de studenten over de volgende basiskennis bezitten:
 - basiskennis Linux en VirtualBox.
- De oplossing is zodanig opgesteld dat het de volgende functies en features kan aantonen:
 - Pods, nodes of containers.
- monitoring van de containers zodanig om inzicht te krijgen in de werking van het systeem.

De gebruikte oplossing moet volledig gratis zijn.

Cloud-based platformen

Hoewel deze oplossing in feite verschillende opties omvat, hebben ze allemaal de gemeenschappelijke eigenschap dat deze oplossingen niet volledig gratis zijn. Elke cloud-based oplossing is ofwel onmiddellijk betalend, of biedt een proefversie aan die, hoewel initieel gratis, kosten kan aanrekenen zodra een gebruiker de gratis limiet overschrijdt.

Als gevolg betekent dit dat cloud platformen niet aan deze vereiste voldoen en dus ook niet verder zullen bekeken worden.

Minikube:

Minikube is volledig gratis.

K3:

K3 is volledig gratis.

Kind:

Kind is is volledig gratis.

MicroK8s:

Microk8s is volledig gratis.

Paulczar: container.training:

Paulczar container training is volledig gratis.

Kelsey hightower: Kubernetes the hard way:

Deze oplossing is eerder een leeromgeving dat gebruik maakt van andere tools of oplossingen als fundering. In het geval van 'Kubernetes the hard way' gaat het er vanuit dat google cloud SDK gebruikt wordt wat een cloud oplossing is waar er potentieel betaald moet worden.

Dit betekent dus dat dit geen ideale oplossing is en dus ook niet verder zal bekeken worden.

De gebruikte oplossing moet lokaal beschikbaar zijn. De opstelling of proof of concept moet alle nodige functionaliteiten bevatten terwijl deze enkel en alleen draait op de laptop van de student door middel van bv.Virtualbox.

Minikube:

Minikube is een lokale tool die virtualbox ondersteunt en zich kan beperken tot één laptop.

K3:

K3 is een lokale tool die virtualbox ondersteunt en zich kan beperken tot één laptop.

Kind:

MicroK8 is een lokale tool die virtualbox ondersteunt en zich kan beperken tot één laptop.

MicroK8s:

Kind is een lokale tool die virtualbox ondersteunt en zich kan beperken tot één laptop.

Paulczar: container.training:

Vergelijkbaar met 'Kubernetes the hard way' is deze oplossing ook een leermiddel dat gebruik maakt van andere tools of oplossingen en dus niet zelfstandig kan werken. Hoewel dit leermiddel nuttig kan zijn met het gebruik van andere tools of oplossingen aangezien deze tools zijn dat lokaal kunnen opgesteld worden, valt deze meerwaarde op de onderliggende oplossingen die hiervoor gebruikt kunnen worden. Daarom zal deze oplossing niet verder bekeken worden.

De oplossing moet aan alle functionele verwachtingen voldoen op een laptop die niet meer dan de minimum specificaties bevat opgelegd door de hogeschool

Minikube:

Vereist Geheugen: 2GB Supporting Operating Systems: MacOS/Windows/Linux

K3:

Vereist Geheugen: 512MB Supporting Operating Systems: Linux

Kind:

Vereist Geheugen: 6-8GB Supporting Operating Systems: MacOS/Windows/Linux

De officiële documentatie van Kind raadt een minimum geheugen aan van 6-8 gigabyte aan. De minimum specificaties voor een laptop die gebruikt zal worden aan de hogeschool verwacht 8 gigabyte, deze minimum vereiste is dus niet praktisch gezien er best ook nog andere applicaties draaien.

MicroK8s:

Vereist Geheugen: 4GB Supporting Operating Systems: MacOS/Windows/Linux (nadruk op ubuntu distributie)

De oplossing en bijkomende instructies/documentaties moeten toegankelijk zijn voor het niveau van een 2de of 3de jaar student.

Minikube:

Minikube heeft een zeer uitgebreide documentatie en gemeenschap en kan eenvoudig opgezet worden met behulp van Vagrant. Minikube vergemakkelijkt het toevoegen van functionaliteiten met behulp van 'add ons' en maakt het eenvoudig om toegang te krijgen tot het Kubernetes dashboard.

K3:

K3 heeft een uitgebreide documentatie en gemeenschap en kan eenvoudig opgezet worden met behulp van een voorgebouwd installatiescript.

MicroK8s:

MicroK8 heeft een uitgebreide documentatie dat als hulpmiddel kan dienen. De installatie van MicroK8 vereist wel Snapd als extra benodigdheid. Snapd kan eventueel een opstelling met behulp van Vagrant verstoren maar dit zou geen groot probleem mogen zijn voor een student 3de jaar.

De proof of concept is zodanig opgesteld dat de werking van het pods, nodes en containers kan aantonen.

Minikube:

Minikube is beperkt in functionaliteit omdat het slechts een single-node cluster kan opzetten. Dit heeft als effect dat Minikube niet meer dan 1 node binnen een cluster kan opzetten om de tool meer beginnervriendelijk te maken. De werking van deze enkele node kan wel nog steeds aangetoont worden.

K3:

K3 is in staat om multi-node clusters op te zetten en kan hiermee extra functies aantonen rond het onderhouden van meerdere nodes tegelijkertijd.

MicroK8s:

Net zoals Minikube is MicroK8 beperkt in functionaliteit omdat het slechts een single-node cluster kan opzetten.

Een monitoring functie van de containers zodanig om inzicht te krijgen in de werking van het systeem is vereist.

Minikube:

Minikube vergemakkelijkt het toegangsproces tot een dashboard enorm. Alle veranderingen binnen de opstelling kunnen eenvoudig waargenomen worden op het dashboard.

K3:

K3 biedt de mogelijkheid aan tot een dashboard, maar lijkt, in tegenstelling tot Minikube, geen vereenvoudigd proces te hebben om toegang te verkrijgen.

MicroK8s:

MicroK8 biedt de mogelijkheid aan tot een dashboard maar lijkt geen vereenvoudigd proces te hebben zoals Minikube om toegang te verkrijgen.

4.3 Vergelijking van de ‘Should have’s’ en de ‘Nice to have’s’

- De proof of concept deployment wordt liefst op gezet via Vagrant en/of Ansible. Een .ova file met gedetailleerde en reproduceerbare installatieprocedure is ook geschikt.
- De mogelijkheid om volgende functies aanwezig te hebben wordt sterk aanbevolen:
 - load balancing en auto scaling
- Rolling updates functionaliteit
- Blue green deployments functionaliteit

De proof of concept deployment kan worden op gezet via Vagrant, Ansible en/of een .ova file met gedetailleerde en reproduceerbare installatieprocedure.

Minikube:

Minikube kan eenvoudig via Vagrant opgezet worden en heeft meerdere beschikbare ansible rollen om voor de installatie te gebruiken.

K3:

K3 kan eenvoudig met Vagrant opgesteld worden en heeft ook (beperkte) mogelijkheden om via ansible de installatie te voltooien.

MicroK8s:

Voor MicroK8s zijn er meerdere ansible rollen beschikbaar. Ook kan het opgezet worden via Vagrant. Er werd wel opgemerkt dat een Snapd installatie nodig is en deze kan eventueel problemen veroorzaken tijdens de ‘Vagrant up’.

De mogelijkheid om load balancing en auto scaling aanwezig te hebben.

Minikube, K3 en MicroK8s:

Alle overblijvende oplossingen stellen beide functionaliteiten beschikbaar.

De mogelijkheid om rolling updates en blue green deployments aanwezig te hebben.

Minikube, K3 en MicroK8s:

Alle oplossingen stellen beide functionaliteiten beschikbaar.

4.4 Conclusies en gekozen oplossing voor proof of concept

De voorgaande vergelijkingen toonden aan dat 3 tools geschikt waren om gebruikt te worden binnen een leeromgeving:

- Minikube
- K3
- MicroK8's

De grootste verschillen tussen deze 3 keuzes liggen onder andere bij hun minimumeisen waarbij K3 een groot voordeel heeft op vlak van benodigd RAM geheugen, terwijl MicroK8s volgens de documentatievereiste meer RAM nodig heeft, wat voor sommige laptops een uitdaging kan zijn. K3 wordt dan wel benadeeld doordat het enkel installeerbaar is op een Linux distributie, terwijl Minikube en MicroK8's ook Windows en MacOS ondersteunen. Een ander verschil is dat Minikube enkele belangrijke zaken vereenvoudigt zoals het gebruik van add ons. Als laatste groot verschil is dat K3 multi-node clusters ondersteund terwijl de andere opties dit niet doen.

De uiteindelijke tool die gebruikt werd om de Proof of concept op te zetten is Minikube, vanwege de eenvoudigheid om sommige functies gebruiksklaar te maken, de meer dan aanvaardbare minimum vereisten, de grote beginnersdocumentatie en handleidingen en de mogelijkheid om Minikube op alle operating systems te kunnen opzetten.

5. Proof Of Concept

Dit hoofdstuk zal de werking van de proof of concept of PoC beschrijven, de stappen om de op het op te bouwen en de werking ervan aan te tonen. De reden waarom dit hoofdstuk voor de enquête en testresultaten valt is omdat deze resultaten gebaseerd zijn op de PoC. Het begrijpen van de Proof of Concept is daarvoor dus een meerwaarde.

5.1 Redenering achter de opstelling

5.1.1 Vagrant

De leeromgeving kan eenvoudiger en sneller ingezet worden als de vereiste kennis en nodige technische capaciteiten zo nauw mogelijk aansluiten met wat een 2de jaar en/of 3de student op dat moment toe in staat is. Om die reden leek het logisch om de technologieën te gebruiken waar dit type student al bekend mee was. Hierdoor werd de keuze gemaakt om gebruik te maken van Vagrant, zolang dat het installatie en configuratieproces gelijkaardig was aan de manier waarop studenten met Vagrant leren werken hebben.

Het gebruik van Vagrant garandeert ook een eenvoudige manier om de opstelling te hergebruiken en om ze opnieuw te installeren, moesten er problemen optreden of moest een student de opstelling per ongeluk aanpassen of zelfs onbruikbaar maken.

5.1.2 CentOS 7

Het operating system van de box die volgens de standaardconfiguratie zal gedownload worden, is een aangepaste versie van de Linux distributie CentOS 7 OS. De box heeft als

aanpassing dat het voorverpakt komt met een Minikube installatie op de machine waardoor er minder fout kan lopen tijdens de ‘Vagrant up’ installatie. CentOS is ook een operating system waar studenten na hun Projecten 2 vak al vrij goed mee overweg kunnen of alleszins in aanraking mee gekomen zijn.

```
Vagrant.configure("2") do |config|  
  config.vm.box = "mrvtage/centos7-minikube"  
end
```

Figuur 5.1: Vagrant configuratie om box te gebruiken.

5.1.3 Minikube

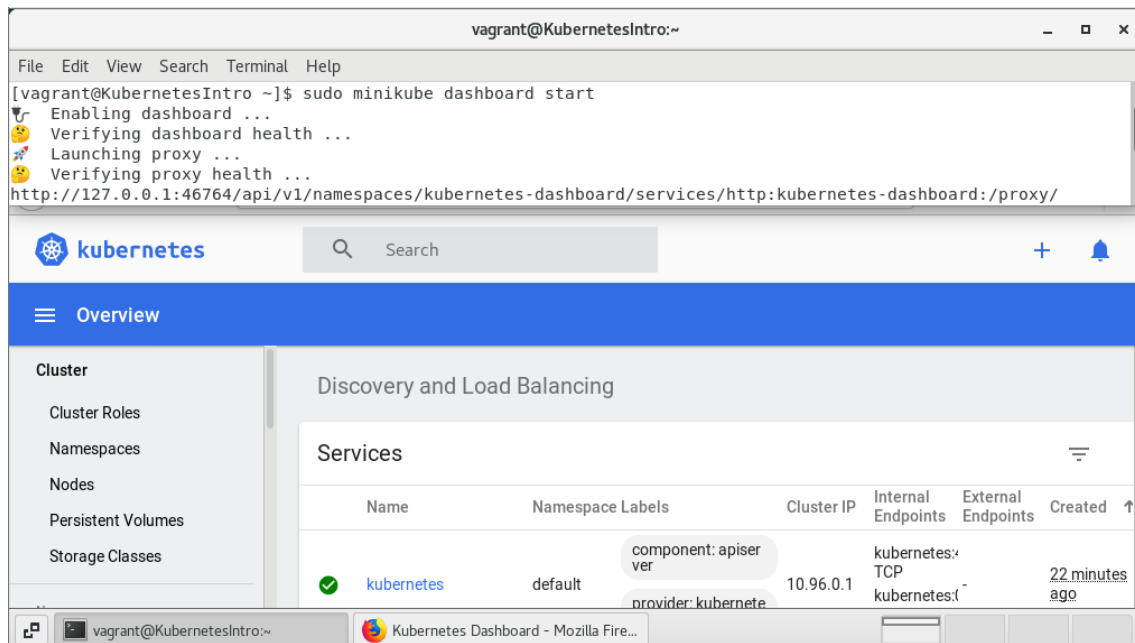
Zoals er al eerder naar verwezen is, zal er gebruik gemaakt worden van Minikube om Kubernetes toegankelijk te maken voor studenten. Het is wel voor herhaling vatbaar dat Minikube zeker niet de enige aantrekkelijke oplossing was en is dus geen strikte noodzaak is om te gebruiken. De redenen waarom er uiteindelijk voor deze oplossing gekozen werd, zijn voornamelijk terug te vinden in bepaalde taak vereenvoudigingen die Minikube geïmplementeerd heeft, waardoor de oplossingen aanzienlijk meer beginnervriendelijk worden.

De 2 voornaamste praktische voordelen die Minikube aanbiedt zijn:

- Add ons: dienen om bepaalde functies te importeren die niet standaard in Kubernetes zitten die noodzakelijk zijn voor sommige functionaliteiten die moesten aangetoond worden binnen de opstelling. De normale manier om deze toevoegingen te installeren kan voor problemen zorgen, door gebruik te maken van de add ons kan dit gedaan worden met slechts 1 commando.
- Vereenvoudigde dashboard toegang: Om normaal toegang te krijgen tot het Kubernetes browser dashboard moet er eerst een token gegenereerd worden, deze token moet verbonden zijn met een bepaalde gebruiker binnen Kubernetes die liefst administrator rechten heeft. Hoewel sommige van deze stappen kunnen geautomatiseerd worden kan er nog altijd veel fout lopen. Minikube voorziet een commando dat niet alleen het dashboard onmiddellijk toegankelijk maakt zonder token, maar ook meteen een browservenster opent voor de gebruiker en rechtstreeks surft naar het dashboard ip.

Verder waren de minimum specificaties voor Minikube een RAM geheugen van 2 gigabyte. Hoewel dit aanzienlijk meer is dan K3, een andere mogelijke oplossing die slechts 512 megabyte vereist, is 2 gigabyte zeker geen hoge eis om te stellen.

Tot slot is Minikube al het langste beschikbaar uit de 3 aantrekkelijke oplossingen. Dit heeft als gevolg dat voor Minikube een ruime hoeveelheid troubleshooting hulp beschikbaar is.



Figuur 5.2: De manuele actie nodig om toegang te krijgen tot het dashboard met Minikube.

5.2 Structuur van de Kubernetes omgeving

5.2.1 Vagrant configuratie

De Vagrant configuratie is zeer gelijkaardig aan de manier waarop studenten leren werken met Vagrant. De Vagrantfile is een minimaal aangepaste versie van 'one Vagrant file to rule them all'¹ geschreven door Bert Van Vreckem. Een belangrijke toevoeging is een extra configuratie die een grafische interface toestaat.

See <https://github.com/bertvv/vagrant-shell-skeleton/> for details

```
require 'rbconfig'
require 'yaml'
```

```
# set default LC_ALL for all BOXES
ENV["LC_ALL"] = "en_US.UTF-8"
```

```
# Set your default base box here
DEFAULT_BASE_BOX = "mrvantage/centos7-minikube"
```

```
VAGRANTFILE_API_VERSION = '2'
PROJECT_NAME = '/' + File.basename(Dir.getwd)
```

¹<https://bertvv.github.io/notes-to-self/2015/10/05/one-vagrantfile-to-rule-them-all/>

```

hosts = YAML.load_file('vagrant-hosts.yml')

# {{{ Helper functions

def windows_host?
  Vagrant::Util::Platform.windows?
end

# Set options for the network interface configuration. All values are
# optional, and can include:
# - ip (default = DHCP)
# - netmask (default value = 255.255.255.0)
# - mac
# - auto_config (if false, Vagrant will not configure this network interface)
# - intnet (if true, an internal network adapter will be created instead of a
#   host-only adapter)
def network_options(host)
  options = {}

  if host.key?('ip')
    options[:ip] = host['ip']
    options[:netmask] = host['netmask'] || '255.255.255.0'
  else
    options[:type] = 'dhcp'
  end

  options[:mac] = host['mac'].gsub(/[-:]/, '') if host.key?('mac')
  options[:auto_config] = host['auto_config'] if host.key?('auto_config')
  options[:virtualbox__intnet] = true if host.key?('intnet') && host['intnet']
  options
end

def custom_synced_folders(vm, host)
  return unless host.key?('synced_folders')
  folders = host['synced_folders']

  folders.each do |folder|
    vm.synced_folder folder['src'], folder['dest'], folder['options']
  end
end

# Adds forwarded ports to your Vagrant machine
#
# example:
# forwarded_ports:

```



```
# - guest: 88
# host: 8080
def forwarded_ports(vm, host)
  if host.has_key?('forwarded_ports')
    ports = host['forwarded_ports']

    ports.each do |port|
      vm.network "forwarded_port", guest: port['guest'], host: port['host']
    end
  end
end

# }}}
Vagrant.configure("2") do |config|
  # ... other configuration

  end

  Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
    hosts.each do |host|
      config.vm.define host['name'] do |node|
        node.vm.box = host['box'] ||= DEFAULT_BASE_BOX
        node.vm.box_url = host['box_url'] if host.key? 'box_url'
        node.ssh.insert_key = false
        node.vm.hostname = host['name']
        node.vm.network :private_network, network_options(host)
        custom_synced_folders(node.vm, host)
        forwarded_ports(node.vm, host)

        # Add VM to a VirtualBox group
        node.vm.provider :virtualbox do |vb|
          vb.memory = host['memory'] if host.key? 'memory'
          vb.cpus = host['cpus'] if host.key? 'cpus'

          # WARNING: if the name of the current directory is the same as the
          # host name, this will fail.
          vb.customize ['modifyvm', :id, '--groups', PROJECT_NAME]
        end
        # Add VM to a VirtualBox group
        node.vm.provider :virtualbox do |v|
          v.gui = true
        end
        node.vm.boot_timeout = 1200
        # Run configuration script for the VM
        node.vm.provision 'shell', path: 'provisioning/' + host['name'] + '.ps1'
      end
    end
  end
end
```

```
end
```

```
# -*- mode: ruby -*-
# vi: ft=ruby :
```

Deze vagrant file voorziet een .yml file om op geordende wijze verschillende parameters te definiëren genaamd vagrant-hosts. Hier werden er enkele belangrijke instellingen voor de machine geconfigureerd.

De vagrant-hosts file is vrij beperkt. Hierin werd de naam van de virtuele machine geconfigureerd, het IP adres van de machine, het aantal cpu's dat zal toegewezen worden aan de machine, de naam van de box die gebruikt zal worden en de hoeveelheid geheugen in megabyte die zal toegewezen worden aan de machine.

```
- name: KubernetesIntro
ip: 10.0.0.11
box: "mrvantage/centos7-minikube"
memory: 3072
cpus: 2
```

De box is afkomstig van de Vagrant cloud en is een CentOS7 machine voorverpakt met Minikube. In deze opstelling werd er 3 gigabyte toegewezen, deze hoeveelheid kan variëren maar moet minimum 2 gigabyte zijn volgens de minikube documentatie. Diezelfde raadt ook minstens 2 cpus aan. Het IP adres kan naar eigen voorkeur ingesteld worden.

5.2.2 Automatisering van bijkomende installaties

Om zo weinig mogelijk tijd te verliezen aan installaties en voorbereidingen, werden velen van deze extra benodigdheden geautomatiseerd via bash script. Dit wordt automatisch uitgevoerd terwijl vagrant de machine configureert, vlak na de algemene instellingen afgerond zijn.

Opmerkelijke onderdelen van dit script zijn de gnome desktop installatie om de toegang tot het dashboard te vereenvoudigen, de installatie van de cluster role binding en het instellen van de correct driver voor Minikube.

```
echo "start running some premade installation scripts"
sudo dnf install epel-release -y
sudo dnf upgrade -y
echo "hopefully epel-release installed"
echo "installing gnome desktop for the kubernetes dashboard"
sudo yum groupinstall -y 'gnome desktop'
sudo yum install -y 'xorg*'
sudo yum remove -y initial-setup initial-setup-gui
sudo systemctl isolate graphical.target
sudo systemctl set-default graphical.target
```

```
echo "preparing minikube and correcting driver"
sudo minikube config set driver docker
minikube delete
minikube start

echo "installing dashboard"
sudo kubectl apply -f https://raw.githubusercontent.com/kubernetes/dash

echo "setting up the service account admin and binding the ClusterRole
sudo kubectl delete clusterrolebinding kubernetes-dashboard

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

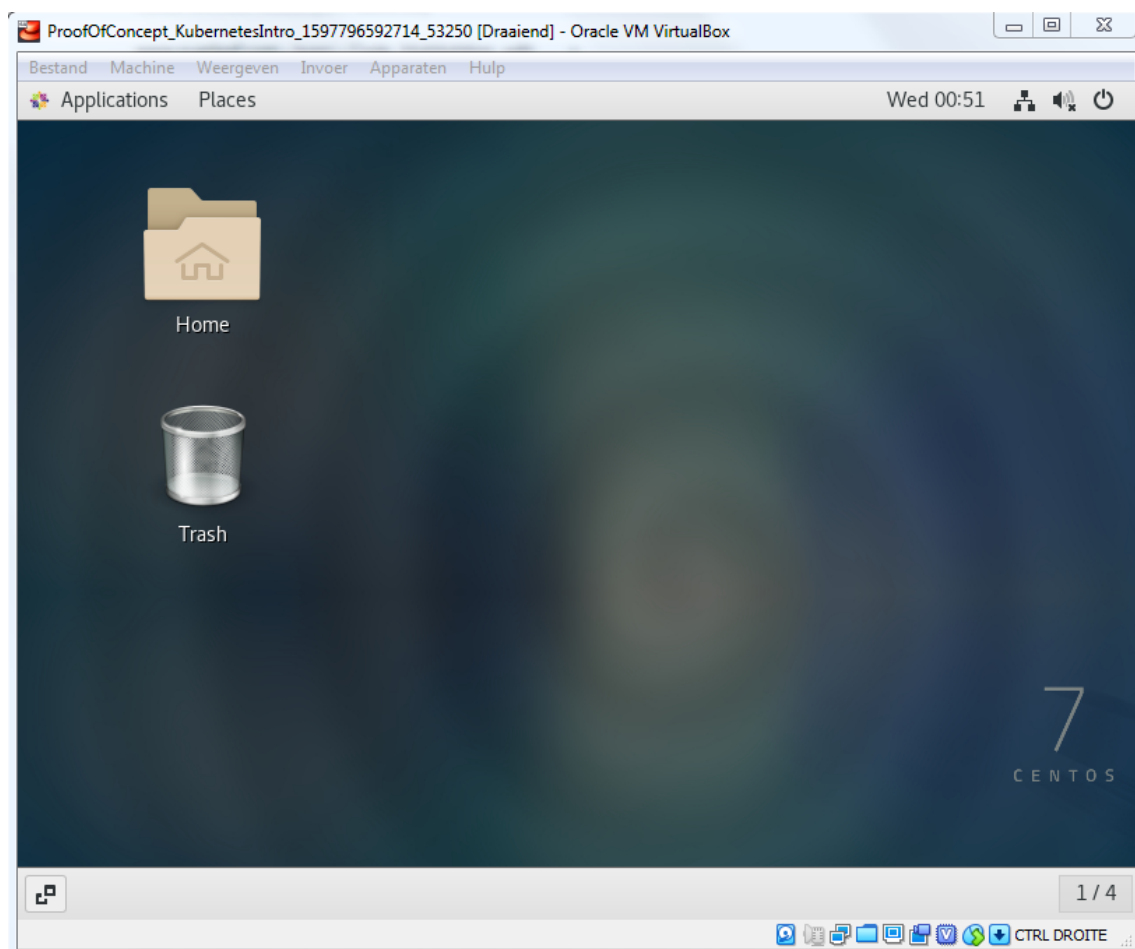
kubectl apply -f dashboard-adminuser.yml

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

kubectl apply -f admin-role-binding.yml

#sudo kubectl -n kube-system describe secret $(kubectl -n kube-system g
#kubectl proxy
#http://10.0.0.11:8001/api/v1/namespaces/kubernetes-dashboard/services/
```

Door de vele vereenvoudigingen die Minikube aanbiedt, zoals de eenvoudige dashboard toegang, waren enkele onderdelen van dit script achteraf als overbodig verklaart, sommige van deze configuraties zijn als commentaar toegevoegd.



Figuur 5.3: Overzicht van de gnome desktop interface.

6. Resultaten enquête en test

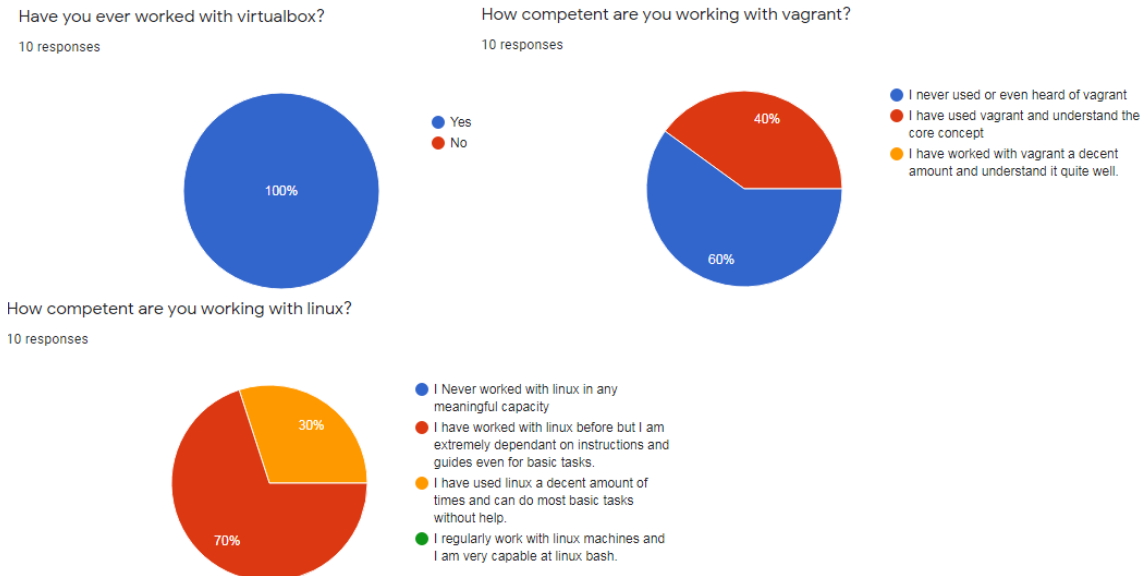
In dit hoofdstuk worden de resultaten van de enquêtes en test besproken. Dit is om een beter beeld te verkrijgen hoe testpersonen de gekozen Kubernetes tool ervaren en heeft ook het bijkomend effect dat er informatie kan worden opgedaan en kan verwerkt rond de uitvoering van de opstelling, bijvoorbeeld knelpunten en de gemiddelde tijd die nodig was om alle voorgelegde taken af te werken.

6.1 Profiel van de onderzoeksdeelnemers

Een selecte steekproef werd opgezet waarbij er personen gevraagd werden om de proof of concept uit te proberen. Om zo goed mogelijk de praktijk na te bootsen, werden enkel personen toegelaten die ten minste een basiskennis hadden van een deelvak van de Informatica. De test werd geconcludeerd nadat 10 deelnemers de test hadden afgelegd. Om een ruwe inschatting te kunnen maken over de relevante kennis van de deelnemers, werd hen de volgende vragen gesteld in de eerste enquête:

- Have you ever worked with virtualbox before?
- How competent are you working with Vagrant?
- How competent are you working with Linux

Uit deze vragen werd er het volgende vastgesteld:



Tabel 6.1: Resultaten test deelnemers profiel

Dit resultaat stelt het volgende vast:

- 100% van de deelnemers hadden zeker en vast al met virtualbox gewerkt.
- 60% Had nog nooit van Vagrant gehoord en hadden er dus nog niet mee gewerkt, 40% had wel al een basiskennis van Vagrant.
- 70% van de deelnemers hadden een basiskennis Linux, 30% van de deelnemers gaven aan dat ze al meer gevorderd waren in het gebruik van Linux.

6.2 Opstelling van de test

Alle deelnemers kregen een folder waarin alle Vagrant configuraties aanwezig waren. Hierbij werd er ook een instructiedocument¹ meegegeven dat bestond uit 3 delen:

- Een theoretische introductie naar kubernetes waarin kernbegrippen werden verklaard.
- Een voorbeeld labo dat eenvoudig uitgetprobeerd kon worden met aanvullende uitleg en links naar officiële documentatie.
- Een iets meer uitdagende opdrachtenreeks waarin de testers relatief zelfstandig bepaalde acties correct moesten uitvoeren.

Tijdens de activiteiten konden de deelnemers wel om hulp en extra uitleg vragen. Dit werd tijdens de opdrachten beperkt om een beter beeld te krijgen of de deelnemers de basistechnieken onder de knie had.

¹De Instructies zijn terug te vinden als bijlage in de appendix

De test werd uitgevoerd over het internet met behulp van voice calls en het delen van elkaars scherm.

Wanneer er deelnemers waren die nog niet alle benodigde software hadden geïnstalleerd zoals Vagrant, werd dit eerst ingehaald. Ook hadden enkele deelnemers problemen met virtualisatie los van de opstellingen zoals incorrecte Hyper-V instellingen; deze werden uiteindelijk allemaal opgelost.

Aangezien er verwacht wordt van echte studenten dat deze software al op voorhand geïnstalleerd is en onder normale omstandigheden werkt, maakten deze tijdsbestedingen geen deel uit van de tijdmetingen. De tijdsopname begon zodra Vagrant succesvol met de download fase begonnen was. Tijdens deze wachttijd was de Kubernetes introductie al begonnen.

6.3 Test en enquête Resultaten

Nadat de volledige test tot een einde kwam, werden alle deelnemers gevraagd een enquête in te vullen over hun ervaringen van de opstelling en de test.

Deze enquête heeft de volgende relevante vragen gesteld:

- How difficult was it to follow the tutorial and then complete the assignment?
- Which aspect(s) did you find challenging?
- Do you understand the overall basics of Kubernetes after finishing the course?

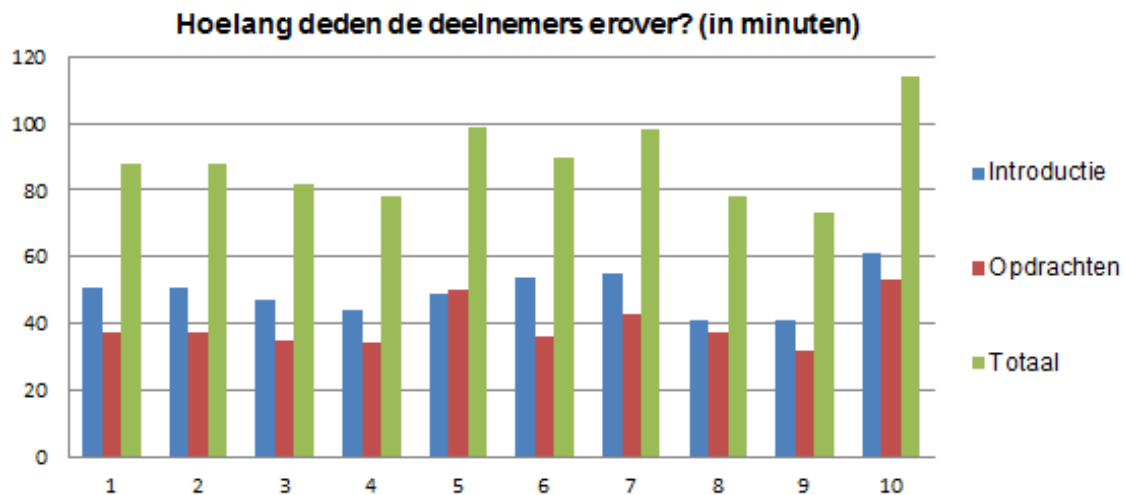
Daarnaast werd ook gemeten hoelang deelnemers erover deden om de test te voltooien. Deze meting is verdeeld in twee delen: Introductie + het voorbeeldslabo en het opdrachtengedeelte. Deze 2 tijdsmetingen bij elkaar gaven de totale tijd die nodig was om de test te voltooien.

6.3.1 Resultaten: Tijdsmeting

In deze figuur staat de tijdsduur voorgesteld van hoelang de deelnemers erover gedaan hebben om de introductie + het voorbeeldslabo en de opdracht te voltooien.

De snelste deelnemer heeft de opdracht voltooid in 73 minuten. De traagste deelnemer deed er 114 minuten over. De gemiddelde duur tussen alle 10 deelnemers lag op 89 minuten.

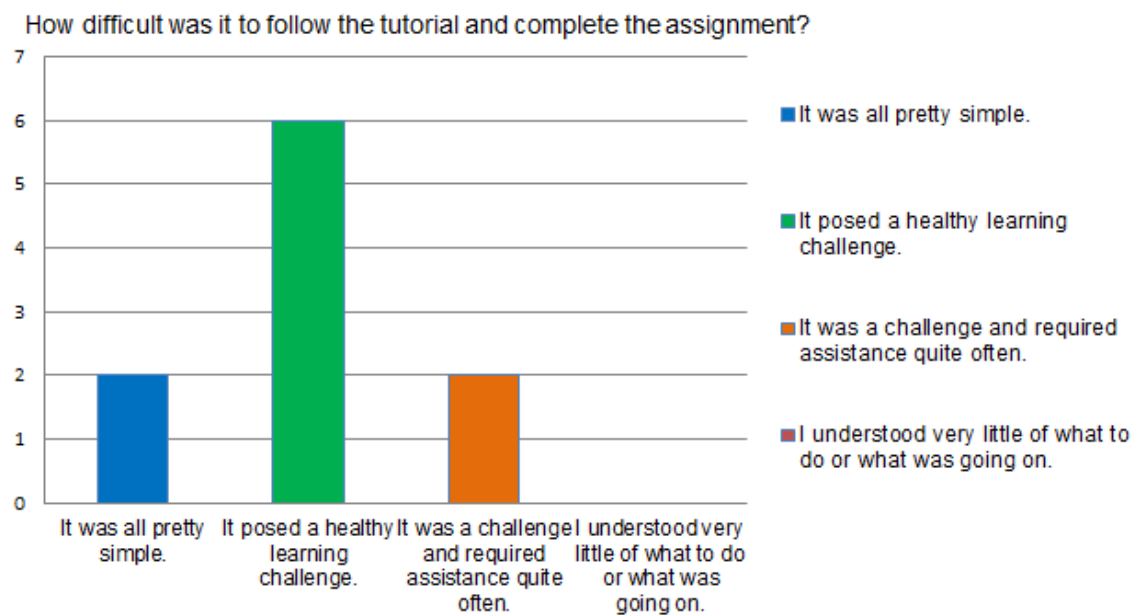
Deze resultaten suggereren dat de opdracht binnen een redelijke tijdsduur van 120 minuten kan voltooid worden.



Tabel 6.2: tijdsbesteding van de deelnemers tijdens de opdracht.

6.3.2 Resultaten: Algemene moeilijkheidsgraad

In deze figuur staat de verdeling van de antwoorden op de vraag: "Hoe moeilijk was het om de handleiding te volgen en om daarna de opdrachten te voltooien."



Tabel 6.3: Bevraging moeilijkheidsgraad van de introductie en opdracht.

Zoals te zien vond 60% van de deelnemers het een gezonde uitdaging die zeker haalbaar was voor individuen die nog geen voorkennis over Kubernetes bezaten.

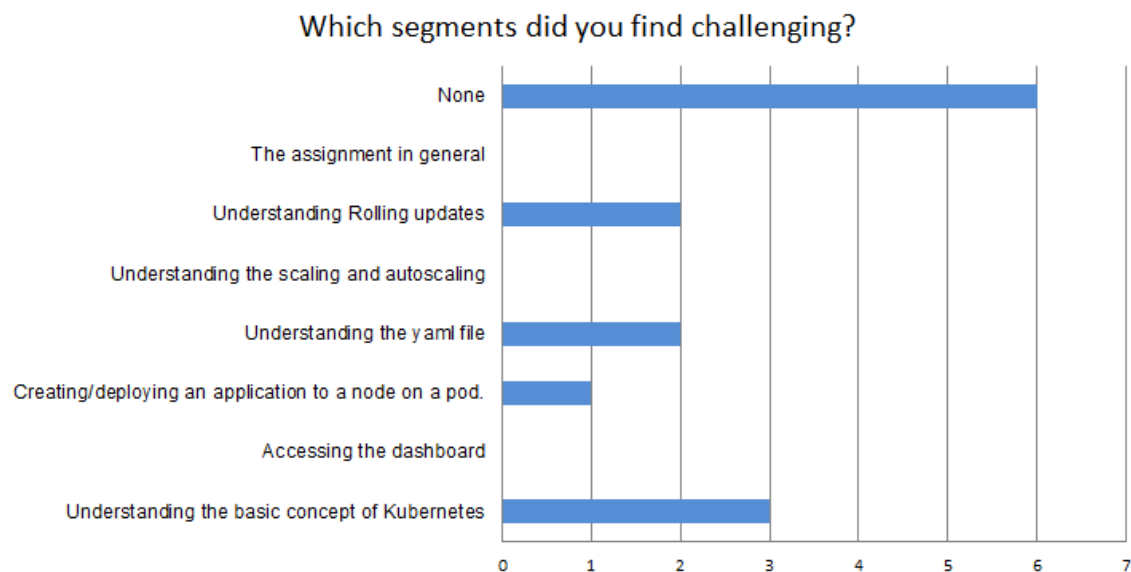
20% van de deelnemers gaf aan dat het zelfs vrij eenvoudig was.

Maar ook vond 20% de opdracht uitdagend en hadden frequent hulp nodig om concepten

te begrijpen.

6.3.3 Resultaten: Specifieke struikelblokken

Hier werd er gevraagd of deelnemers met specifieke onderdelen van de ervaring moeilijkheden hadden. Dit was om na te gaan welk onderdeel ofwel lastiger was om te begrijpen ofwel niet duidelijk genoeg was tijdens de test. Men kon meerdere antwoorden aanvinken bij deze vraag en dat gaf het volgende resultaat:



Tabel 6.4: Met welke onderdelen hadden de deelnemers problemen

Eerst en vooral: 60% van de deelnemers heeft hier aangegeven dat geen enkel onderdeel opmerkelijk zwaarder was dan de anderen.

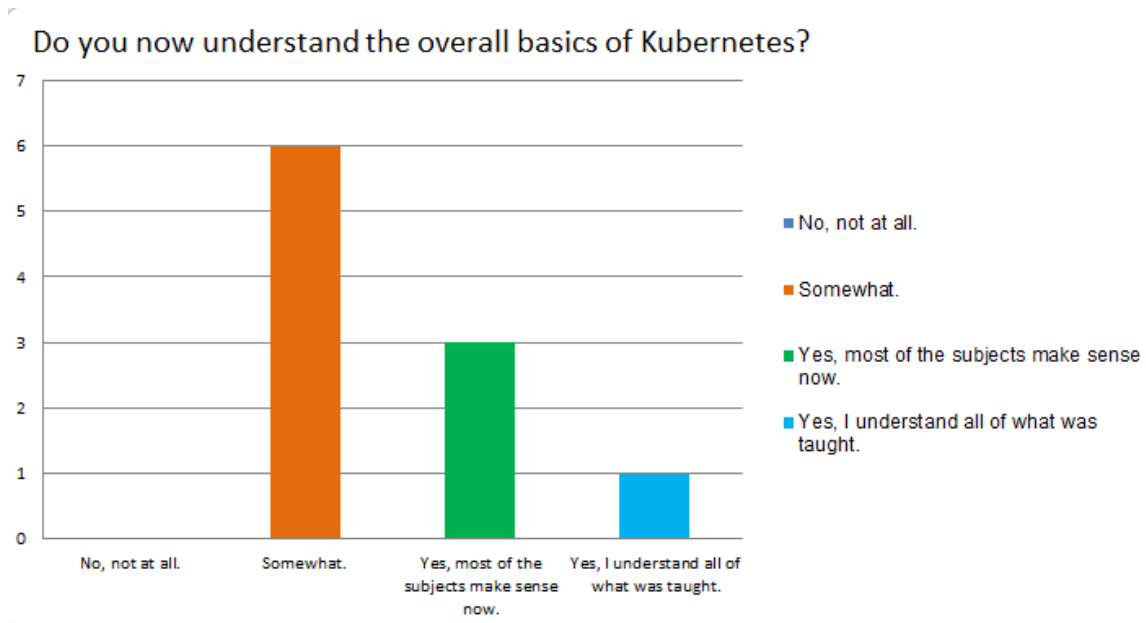
Verder kan er gezien worden dat 4 onderdelen voor één of meerdere deelnemers een probleem vormde. Er waren 2 deelnemers die aangaven dat rolling updates een probleem was.

Nog eens 2 deelnemers hadden het moeilijk om met .yaml files te werken.

Uiteindelijk hadden 3 deelnemers aangegeven dat het basisconcept van Kubernetes een uitdaging was om te begrijpen.

6.3.4 Resultaten: Heeft de deelnemer iets bijgeleerd over Kubernetes?

In deze figuur staat de verdeling van de antwoorden op de vraag: "Hoe goed begrijpt u nu de basisfuncties en werking van Kubernetes?"



Tabel 6.5: Algemene mening of de deelnemer iets bijgeleerd heeft.

30% van de deelnemers gaven aan dat hij of zij er iets uit geleerd had, maar nog niet alle eigenschappen begreep.

60% gaf aan dat hij of zij de basis concepten en functies grotendeels begreep.

Opmerkelijk gaf één deelnemer aan dat alles binnen de test goed begrepen was.

6.3.5 Extra opmerkingen en observaties

Een opmerkelijke observatie van de test is dat er zeer weinig Linux kennis nodig was om de test tot een goed einde te brengen. De enige uitzondering is het gebruik van de vi teksteditor, wat op zich optioneel is door gebruik te maken van alternatieve opties om bepaalde taken te voltooien binnen Kubernetes of door het installeren van een eenvoudigere text editor.

7. Conclusies

In hoofdstuk 1.1 werd de doelstelling van dit onderzoek gedeclareerd, deze was: Het doel van deze bachelorproef is om onderzoek te doen naar de verschillende mogelijkheden die Kubernetes lokaal kunnen opstellen en als leerplatform kunnen dienen, waaruit vervolgens besloten zal worden welke tools een meerwaarde kunnen betekenen voor de opleiding Informatica.

Dit betekent dat dit onderzoek niet alleen naar de technische mogelijkheden van de verschillende Kubernetes tools zal kijken, maar ook naar hun educatieve mogelijkheden. Hieruit zal er als bijkomend resultaat ook een werkend prototype afleveren op een github repository met een handleiding en instructies document afgetoetst voor een student Toegepaste Informatica.

Volgende onderzoeksvraag werd voor deze taak geformuleerd:

- Welke Kubernetes oplossing is het meest geschikt om als lokaal leerplatform te dienen voor studenten Informatica?

Uit dit onderzoek om een geschikte Kubernetes oplossing te zoeken, werd vastgesteld dat van alle oplossingen die bekeken werden, er 3 geschikt waren om te gebruiken.

- Minikube
- K3
- MicroK8's

Verdere vergelijking tussen deze 3 keuzes gaf aan dat K3 een voordeel had in hoeveel RAM geheugen vereist was. MicroK8s was hierin de verliezer en vereiste de meeste

hoeveelheid RAM geheugen uit de 3. K3 was wel enkel beschikbaar op Linux distributies, terwijl Minikube en MicroK8's ook Windows en MacOS ondersteunen. Alledrie de oplossingen waren ontworpen om te dienen als een vereenvoudigde Kubernetes opstelling en hadden allemaal bepaalde zaken die de werking meer beginnervriendelijk maakten. Voor MicroK8s was Snapd nodig om te functioneren. Dit was een extra installatie die potentieel wel problemen kon veroorzaken tijdens de Vagrant configuratie in het geval dat deze installatie geautomatiseerd zou worden.

Voor de proof of concept werd Minikube gekozen. Om deze opstelling degelijk op de proef te stellen, werd er een test opgezet om het gebruik van deze opstelling die gebouwd werd met Minikube. De test toonde aan dat de ervaring door de deelnemers algemeen als positief kan beschouwd worden.

De deelnemers waren algemeen vrij tevreden over hun deelname en ervaring met de Kubernetes Minikube opstelling en gaven aan dat ze de basis concepten die voorgelegd werden begrepen op het einde van de test.

De deelnemers hadden allemaal een profiel vergelijkbaar met dat van typische studenten Toegepaste Informatica en waren dus een vrij goede groep om deze opstelling op te testen.

Wat uiteindelijk opmerkelijk bleek te zijn, was de weinige hoeveelheid Linux kennis die nodig was om de opdracht te voltooien. Met uitzondering van één enkel onderdeel, namelijk vi, voelde geen enkele deelnemer aan dat hij of zij te weinig kennis had over Linux om de technologie te kunnen begrijpen.

Hopelijk is deze bachelorproef een meerwaarde voor het curriculum van de opleiding Toegepaste Informatica zodat er een nieuwe technologie aan het lespakket kan toegevoegd worden waar studenten Systeembeheer later hoogstwaarschijnlijk mee in aanraking zullen komen.

A. Onderzoeksvoorstel

A.1 Introductie

Deze opdracht zal de verschillende bestaande vereenvoudigde Kubernetes implementaties onderzoeken om een opstelling te bepalen die tegelijk informatief een beeld kan geven van Kubernetes en eenvoudig kan opgezet worden door studenten van het tweede of derde traject jaar die slechts over een basiskennis systeembeheer beheersen. HoGent wil over een lokale Kubernetes opstelling beschikken om als gratis leeromgeving te gebruiken. Dit is belangrijk omdat eenvoudige cloud oplossingen onmogelijk gratis uitgeprobeerd kunnen worden, om extra kosten voor de studenten te vermijden moet er een kosteloze opstelling afgeleverd worden. Het doel van dit onderzoek is dus om uit verschillende reproduceerbare mogelijkheden een geschikte Kubernetes architectuur op te bouwen die op kosteloze wijze informatief de werking en opbouw van Kubernetes kan uitleggen aan onervaren studenten.

A.2 State-of-the-art

Steeds meer projecten gebruiken container solutions met Kubernetes voor het hosten van producten, het aantal containers vermenigvuldigd zich snel (Vizard, 2019). Het opzetten, onderhouden en managen van deze containers kan al snel een probleem vormen. Kubernetes is momenteel een belangrijke "container orchestration software" voor het automatiseren van het opstellen, onderhouden en verwijderen van containers in de cloud. (Velez, 2019) Deze technologie is een recente ontwikkeling en heeft een grote kans om de koploper te worden voor het onderhouden van gevirtualiseerde omgevingen. Een Kubernetes cluster is een collectie van hosts, opslag en network resources dat Kubernetes gebruikt om verschillende processen uit te voeren voor een applicatie. (Sayfan, 2017)

A.3 Methodologie

Dit onderzoek zal beginnen met het nagaan of er al onderwijs instellingen Kubernetes in hun curriculum verwerkt hebben en dit aan studenten aanleren. Vanuit dit deelonderzoek zal er daarna een lijst opgebouwd worden van alle mogelijke en reproduceerbare methodes om Kubernetes op te bouwen via Vagrant en Ansible. Elke tool binnen deze lijst zal met elkaar vergeleken worden om te bepalen of het gebruik van deze tool voldoende nuttig is voor het aanleren van Kubernetes. En of dit voor studenten informatica relatief haalbaar is. De mogelijkheden die hieruit voortkomen zullen verder met elkaar vergeleken worden via eenvoudige basis opstellingen en eventueel steekproeven. Via het opbouwen van een basis opstelling en het aanvullen van deze omgeving met leerrijke scenario's die de werking en functies van Kubernetes transparant maken kan er bekeken worden of deze opstelling een meerwaarde kan betekenen voor het HoGent curriculum.

A.4 Verwachte resultaten

Het verwachte resultaat is een eenvoudig reproduceerbare opstelling die met basisinstructies door onervaren studenten kan opgezet worden op een lokale computer via Vagrant en/of Ansible. Deze opstelling bevat scenario's die de werking van Kubernetes verder uitleggen en bevat verschillende opties die kunnen ingesteld worden op een Kubernetes implementatie om de functie van deze opties duidelijk te maken.

A.5 Verwachte conclusies

Er wordt verwacht om een geschikte tool te vinden waarmee Kubernetes kan aangeleerd worden aan studenten die een basis kennis bevatten over virtualisatie en Vagrant.

B. Proof of concept documentatie

B.1 Introductie en Hands-on tutorial.

Introduction to Kubernetes

<https://www.youtube.com/watch?v=PH-2FfFD2PU>

Kubernetes, at its basic level, is a system for running and coordinating containerized applications across a cluster of machines. It is a platform designed to completely manage the life cycle of containerized applications and services using methods that provide predictability, scalability, and high availability.

One server functions as the master server. This server acts as a gateway and brain for the cluster by exposing an API for users and clients, health checking other servers and deciding how best to split up and assign work.

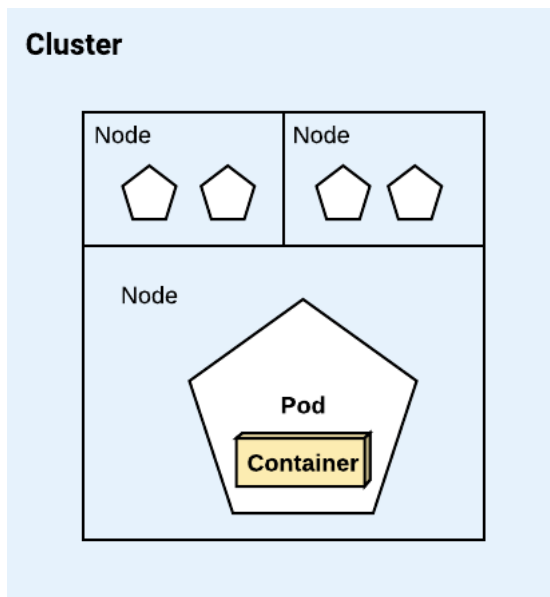
The other machines in the cluster are designated as nodes: servers responsible for accepting and running workloads using local and external resources. To help with isolation, management, and flexibility, Kubernetes runs applications and services in containers, so each node needs to be equipped with a container runtime (like Docker or rkt). The node receives work instructions from the master server and creates or destroys containers accordingly, adjusting networking rules to route and forward traffic appropriately.

As mentioned above, the applications and services themselves are run on the cluster within containers. The underlying components make sure that the desired state of the applications matches the actual state of the cluster.

To start up an application or service, a declarative plan is submitted in JSON or YAML defining what to create and how it should be managed. The master server then takes the plan and figures out how to run it on the infrastructure by examining the requirements and the current state of the system.

Basic components

Containers run in pods, which are the atomic object type within a Kubernetes cluster. Here's how they relate:



Figuur B.1:

pod: A pod defines the logical unit of the application; it can contain one or more containers and each pod is deployed onto a node.

node: This is the virtual machine serving as the worker in the cluster; pods run on the nodes. For the purpose of this tutorial, we will be using a single-node environment, everything will be done within 1 node.

Replication Sets: Often, when working with Kubernetes, rather than working with single pods, you will instead be managing groups of identical, replicated pods. These are created from pod templates and can be horizontally scaled by controllers known as replication controllers and replication sets.

Services: A service is a component that acts as a basic internal load balancer and ambassador for pods. A service groups together logical collections of pods that perform the same function to present them as a single entity.

Hands-on practise

This section will showcase the step-by-step basics of deploying applications using Kuber-

netes and trying out several utilities. The external links in this section lead to the official documentation for more in-depth documentation.

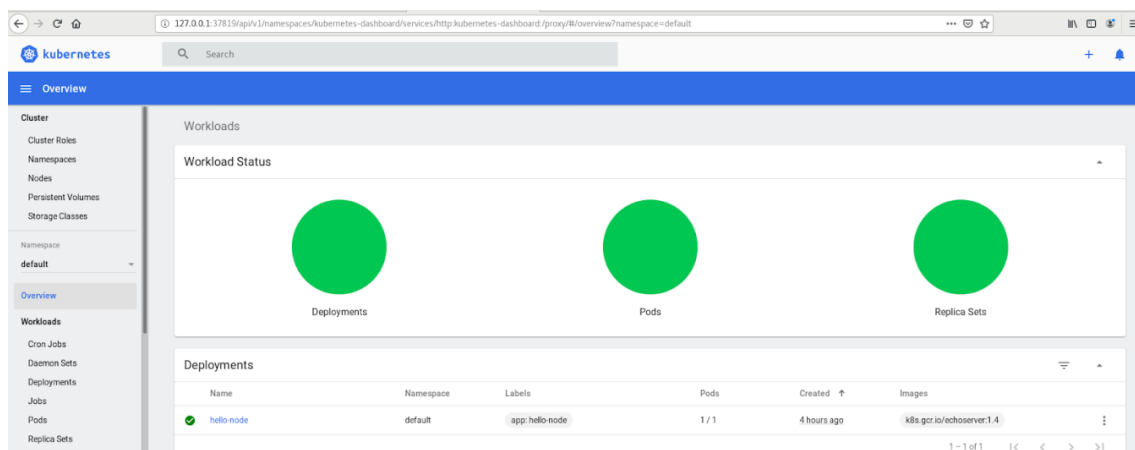
This tutorial uses Minikube, which is designed to be a fast and beginners-friendly way to learn Kubernetes. <https://kubernetes.io/docs/setup/learning-environment/minikube/>

Note: Several prerequisites such as setting up Minikube and properly setting up admin permission have already been taken care of using the vagrant provisioning.

1 Accessing the dashboard

<https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/> For a more approachable overview of our cluster we will be using the dashboard.

The Dashboard UI is not enabled by default: Sudo minikube dashboard The dashboard will become available on the master's localhost IP: this means it cannot be accessed via the host computer and the VM browser must be used. Browse to the dashboard using the ip address it provided This should be the expected result when accessing the IP:



Figuur B.2:

The information visible on the dashboard can also be accessed using commands, a few examples of these are: `kubectl get deployments` `kubectl get pods` `kubectl get all --all-namespaces` (shows you everything within the namespace we will be in)

Minikube comes with several add-ons, such as Kubernetes dashboard, Kubernetes DNS, and so on. certain addons will be required later, this command is useful to ensure they are enabled. We can list the available addons by executing the following command: `minikube addons list`

Optional command used to generate an access token for the dashboard for the admin user that was created beforehand: `kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep admin | awk '{print $1}')`

2 create a deployment

A Kubernetes Pod is a group of one or more Containers, the pod in this tutorial has only one Container.

Use the `kubectl create` command to create a Deployment that manages a Pod. The Pod runs a Container based on the Docker image which in this case will be pulled from the internet.

```
kubectl create deployment hello-node --image=k8s.gcr.io/echoserver:1.4
```

3 Create a load balancer service

By default, the Pod is only accessible by its internal IP address within the Kubernetes cluster. To make the hello-node Container accessible from outside the Kubernetes virtual network, you have to expose the Pod as a Kubernetes Service. Expose the Pod to the public internet using the `kubectl expose` command: `kubectl expose deployment hello-node --type=LoadBalancer --port=8080` The `--type=LoadBalancer` flag indicates that you want to expose your Service outside of the cluster. On cloud providers that support load balancers, an external IP address would be provisioned to access the Service. On Minikube. The LoadBalancer type can make the service accessible through the minikube service command: `minikube service hello-node` Using this command inside the vm on the master will immediately open up a browser that will show the deployed app.

View the Pod and Service you just created: `kubectl get pod,svc -n kube-system`

4 creating an nginx webserver deployment based on a YAML file

<https://kubernetes.io/docs/concepts/containers/images/> <https://kubernetes.io/docs/tasks/run-application/run-stateless-application-deployment/> <https://kubernetes.io/docs/concepts/workloads/controllers/>

Like the hello-minikube will we be pulling the yaml file from the internet: `kubectl apply -f https://k8s.io/examples/application/deployment.yaml` Running `kubectl get all --all-namespaces` afterwards shows our deployment was added.

NAMESPACE	NAME	READY	UP-TO-DATE	AVAILABLE	AGE
default	deployment.apps/hello-node	1/1	1	1	5h40m
default	deployment.apps/nginx-deployment	0/2	2	0	5s

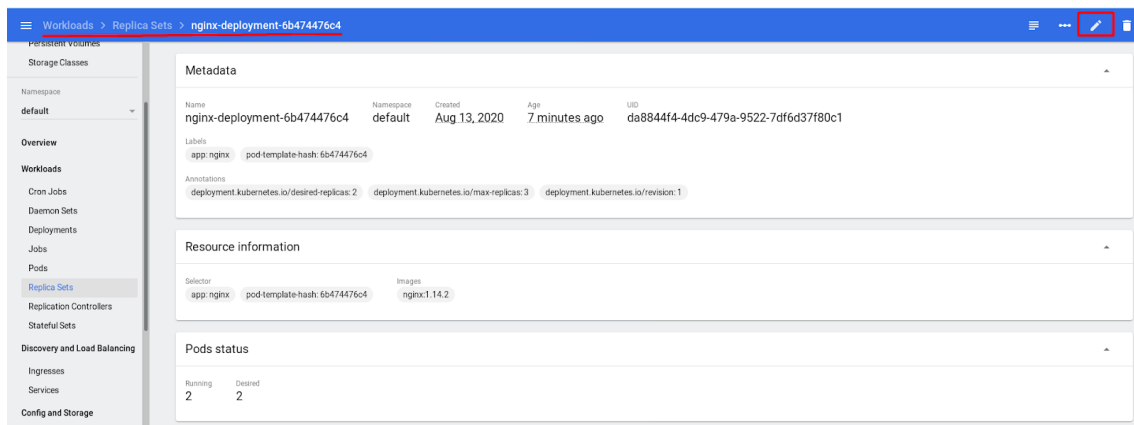
Figuur B.3:

The following command will give you an overview of the deployment: `kubectl describe deployment nginx-deployment` The follow command will show you the specific pods that are being used to deploy the nginx deployment: `kubectl get pods -l app=nginx` The “app=nginx” refers to the label given to the pods as seen in the below yaml configuration. A label can be any key and value and is not limited to just “app”

This is what the Yaml file looks like: — application/deployment-update.yaml
 Copy application/deployment-update.yaml to clipboard
 apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
 kind: Deployment
 metadata: name: nginx-deployment
 spec: selector: matchLabels: app: nginx replicas: 2
 template: metadata: labels: app: nginx spec: containers: - name: nginx image: nginx: 1.14.2 ports: - containerPort: 80 —

- A Deployment named nginx-deployment is created, indicated by the `.metadata.name` field.
- The Deployment creates 2 replicated Pods, indicated by the `.spec.replicas` field. In the yaml
- The `.spec.selector` field defines how the Deployment finds which Pods to manage. In this case, you simply select a label that is defined in the Pod template (`app: nginx`).

The deployment and it's corresponding yaml configuration can also be viewed on the dashboard:



Figuur B.4:

5 Scaling and autoscaling deployments

For autoscaling we will need the following addon: `sudo minikube addons enable metrics-server`

Since Deployments use ReplicaSets internally to manage pods, they support scaling up or down. Let's scale our nginx deployment to run 5 pods instead of 2.

You can increase the number of pods in your Deployment by applying a new YAML file. This command changes the YAML file and sets replicas to 5, which specifies that the Deployment should have five pods, this can be done via the following command: `kubectl scale deployment.v1.apps/nginx-deployment --replicas=5`







An alternative way to achieve this is by editing the deployment configuration YAML directly with `vi` using the command: `kubectl edit deployment.v1.apps/nginx-deployment`

Verify that the Deployment has 5 pods: `kubectl get pods -l app=nginx`

The dashboard will also display the 3 new pods being created.

You can also use Horizontal Pod Autoscaling to automatically increase or decrease the number of pods in a deployment based on the CPU load on the node.

`kubectl autoscale deployment nginx-deployment --min=3 --max=10 --cpu-percent=70` This

Pods					
Name	Namespace	Labels	Node	Status	Restarts
 nginx-deployment-6b474476c4-5rf2v	default	app: nginx pod-template-hash: 6b474476c4	localhost.localdomain	Waiting: ContainerCreating	0
 nginx-deployment-6b474476c4-lx5sq	default	app: nginx pod-template-hash: 6b474476c4	localhost.localdomain	Waiting: ContainerCreating	0
 nginx-deployment-6b474476c4-nfwfb	default	app: nginx pod-template-hash: 6b474476c4	localhost.localdomain	Waiting: ContainerCreating	0
 nginx-deployment-6b474476c4-5qhgb	default	app: nginx pod-template-hash: 6b474476c4	localhost.localdomain	Running	0
 nginx-deployment-6b474476c4-pg7jr	default	app: nginx pod-template-hash: 6b474476c4	localhost.localdomain	Running	0
 hello-node-7bf657c596-q5qbf	default	app: hello-node pod-template-hash: 7bf657c596	localhost.localdomain	Running	1

Figuur B.5:

command adds or kills pods from the Deployment according to the amount of CPU load on the node. It ensures that pods have an average CPU load of 70%. When the load is less, the Deployment kills extra pods as long as their number is not less than three.

To get a list of the HorizontalPodAutoscaler objects in your cluster, run: `kubectl get hpa`



To delete this specific HorizontalPodAutoscaler object: `kubectl delete hpa nginx-deployment`

6 Rolling updates

The real power of a Deployment lies in its ability to update the pod templates without causing application outage. The following command changes the image tag of the containers named nginx to use the image nginx version tagged 1.16.1 instead of 1.14.2: `kubectl set image deployment/nginx-deployment nginx=nginx:1.16.1 --record`

An alternative way to achieve this is by editing the deployment configuration YAML directly using: `kubectl edit deployment.v1.apps/nginx-deployment`. You can most clearly see the updating in progress on the dashboard:

Replica Sets

Name	Namespace	Labels	Pods	Created ↑	Images
 nginx-deployment-7b45d69949	default	app: nginx pod-template-hash: 7b45d69949	5 / 5	a minute ago	nginx:1.16.1
 nginx-deployment-6b474476c4	default	app: nginx pod-template-hash: 6b474476c4	0 / 0	39 minutes ago	nginx:1.14.2

Figuur B.6:

The algorithm that Kubernetes Deployments use when deciding how to roll updates is to keep at least 25% of the pods running. Accordingly, it doesn't kill old pods unless a sufficient number of new ones are up. In the same sense, it does not create new pods until

enough pods are no longer running. Through this algorithm, the application is always available during updates

You can use the following command to determine the update strategy that the Deployment is using:

```
kubectl describe deployments | grep Strategy
```

Kubernetes automatically decides that it needs to keep at least 75% of the pods available. That is, only 25% of the pods (one out of four, for example can be down during the update process). If you want to override this behavior, you can edit the nginx yaml file using: `kubectl edit deployment.v1.apps/nginx-deployment` And change `spec.strategy.type` as follows: `"maxUnavailable: 50%"` By setting the `maxUnavailable` parameter to 50%, Kubernetes can bring down as much as half the running Pods during the update. This number can either be a percentage or a whole number.

Blue green deployments

An alternative to rolling updates where you have two different versions of your application, the latest (green) and the running (blue). Once the green deployment is ready, you can configure the Service to select the new pods (through Pod Selector). If everything goes without issues, you can also update the blue version to be the latest and use it as a staging environment.

B.2 Opdrachtgedeelte

Assignment

For the Assignment you will be given all the tools necessary to set up a certain application and will be tasked to configure it in a specific way. Refer to the above examples and online documentation to finish the assignment. The following cheatsheet can be used to help you: <https://kubernetes.io/docs/reference/kubectl/cheatsheet/#viewing-finding-resources>

The online pullable configuration link: <https://k8s.io/examples/application/php-apache.yaml>

This is the corresponding yaml configuration: =====
 apiVersion: apps/v1 kind: Deployment metadata: name: php-apache spec: selector: matchLabels: run: php-apache replicas: 1 template: metadata: labels: run: php-apache spec: containers: - name: php-apache image: k8s.gcr.io/hpa-example ports: - containerPort: 80 resources: limits: cpu: 500m requests: cpu: 200m — apiVersion: v1 kind: Service metadata: name: php-apache labels: run: php-apache spec: ports: - port: 80 selector: run: php-apache =====
 Can you guess based on the configuration what type of application is being created here?

1) Successfully creating this deployment should leave you with 1 pod. Use the correct command to verify that this pod exists and is running whilst filtering the other deployment pods out. The output should look like this:

NAME	READY	STATUS	RESTARTS	AGE
php-apache-5c4f475bf5-f24kd	1/1	Running	1	23m

```
[vagrant@KubernetesIntro ~]$
```

Figuur B.7:

You can check all created deployments using : `kubectl get deployments` to also see the php deployment.

2) this application currently only uses 1 pod with a 500mb cpu limit, let's change this deployment so it's desired state contains 2 pods instead of 1. You can verify the change like this:

```
[vagrant@KubernetesIntro ~]$ kubectl get deployments
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
hello-node	1/1	1	1	20h
nginx-deployment	5/5	5	5	15h
php-apache	2/2	2	2	42m

```
[vagrant@KubernetesIntro ~]$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
hello-node-7bf657c596-q5qbf	1/1	Running	4	20h
nginx-deployment-7b45d69949-4wxw7	1/1	Running	3	14h
nginx-deployment-7b45d69949-l8mcv	1/1	Running	3	14h
nginx-deployment-7b45d69949-rxvr4	1/1	Running	3	14h
nginx-deployment-7b45d69949-t2dwm	1/1	Running	3	14h
nginx-deployment-7b45d69949-vs6l5	1/1	Running	3	14h
php-apache-5c4f475bf5-98ssw	1/1	Running	0	97s
php-apache-5c4f475bf5-f24kd	1/1	Running	1	43m

Figuur B.8:

3) This application will be using up a lot of cpu power soon, to make sure our deployment can handle it, we will be setting up autoscaling so the app can easily assign new pods if needed.

Set the minimum amount of pods to 1 and the maximum to 10 and set it so that the app maintains a 20% cpu usage per node.

To verify this, use the correct command to see this output:

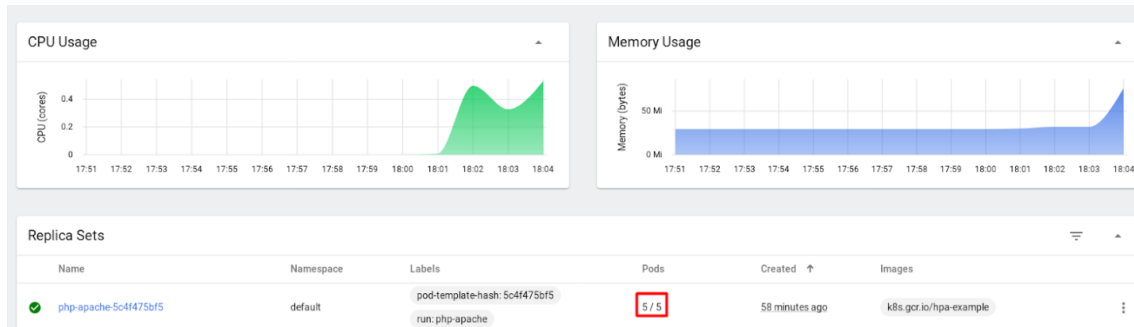
NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
nginx-deployment	Deployment/nginx-deployment	<unknown>/70%	3	10	5	14h
php-apache	Deployment/php-apache	0%/50%	1	10	1	47m

```
[vagrant@KubernetesIntro ~]$
```

Figuur B.9:

Now we will be manually making the application do some cpu-intensive process. Open up a new command prompt in the vm and run this: `kubectl run -it --rm load-generator --image=busybox /bin/sh` Then press enter, Then enter the following code and press enter: `while true; do wget -q -O- http://php-apache; done` After waiting about a minute, you should check the dashboard to see the replica set of php-apache.

In this example, the replicas have increased to 5 to deal with the cpu-intensive task.



Figuur B.10:

4) Next up we will be lowering the possible cpu of each node as well as change its rolling update strategy. Set the maxUnavailable to 90% and set the cpu limit to 300m.

We decided we don't want php-apache, and we will be replacing the image on the pods with nginx 1.16 from before. Use the following command: `kubectl set image deployment/php-apache php-apache=nginx:1.16.1 --record`

Soon after changing the image, you will be able to view on the dashboard that 9 out of 10 nodes will be terminated to be replaced by the new version one as we dictated by changing the maxUnavailable for rolling updates. The last old version pod will be terminated once 10% of the new version is online and running.

Stop the cpu-intensive task by doing control-C in the command prompt or by closing it.

Bibliografie

- Burns, B. (2018, juni 20). The History of Kubernetes and the Community Behind It. <https://kubernetes.io/blog/2018/07/20/the-history-of-kubernetes-the-community-behind-it/>
- CNCF. (2019). *Deployments are getting larger as cloud native adoption becomes mainstream*. https://www.cncf.io/wp-content/uploads/2020/03/CNCF_Survey_Report.pdf
- Ellingwood, J. (2018, mei 2). *An Introduction to Kubernetes* (DigitalOcean, Red.). <https://www.digitalocean.com/community/tutorials/an-introduction-to-kubernetes>
- Gerend, J. (2019). Containers vs. virtual machines. *Microsoft Docs*. <https://docs.microsoft.com/en-us/virtualization/windowscontainers/about/containers-vs-vm>
- Gupta, U. (2015). Comparison between security majors in virtual machine and linux containers. *Information Networking Institute*. <https://arxiv.org/abs/1507.07816>
- Hightower, K. (2019, juli 9). *Kubernetes: Up and Running*. O'Reilly Media, Inc.
- K3. (2020). *lightweight Kubernetes* (K3, Red.). <https://k3s.io/>
- Kubernetes. (2020, juli 27). *Understanding Kubernetes Objects*. <https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/>
- Matthias. (2015). *Docker Up And Running*.
- Moilanen, M. (2018). *Deploying an application using Docker and Kubernet* (masterscriptie). Oulu University of Applied Sciences. https://www.theseus.fi/bitstream/handle/10024/146845/Moilanen_Miika_Opinnaytetyo.pdf
- Oracle. (2011). *Brief History of Virtualization*. https://docs.oracle.com/cd/E26996_01/E18549/html/VMUSG1010.html
- Paris, E. (2014). *Kubernetes: What's it do?* <http://people.redhat.com/~eparis/kubernetes/kube.pdf>
- Sayfan, G. (2017, mei). *Mastering Kubernetes*.
- Singh, A. (2004). An introduction to virtualization.

- Smails, P. (2019). Rancher Labs Industry Survey Shows Rapid Adoption of Containers and Kubernetes, But Challenges Remain. <https://rancher.com/blog/2019/container-industry-survey-results/>
- Stapleton, C. (2016, april 1). High-Performance Containerization Challenges. In *High-Performance Containerization Challenges*. <https://www.osti.gov/biblio/1365107>
- Varian, M. (1991, april). VM AND THE VM COMMUNITY: Past, Present, and Future. <http://www.leeandmelindavarian.com/Melinda/neuvm.pdf>
- Velez, G. (2019). Kubernetes vs. Docker: A Primer. *container journal*.
- Vizard, M. (2019). Survey Finds Kubernetes Clusters Are Multiplying. *container journal*. <https://containerjournal.com/topics/container-ecosystems/survey-finds-kubernetes-clusters-are-multiplying/>