



HoGent

Faculteit Bedrijf en Organisatie

Docker for Windows

Stephan Heirbaut

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

Promotor:
Steven Vermeulen
Co-promotor:
Gert Schepens

Instelling: —

Academiejaar: 2017-2018

Tweede examenperiode

Faculteit Bedrijf en Organisatie

Docker for Windows

Stephan Heirbaut

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

Promotor:
Steven Vermeulen
Co-promotor:
Gert Schepens

Instelling: —

Academiejaar: 2017-2018

Tweede examenperiode

Woord vooraf

Deze bachelorproef voert een vergelijkende studie uit tussen Docker for Windows en Docker for Linux. Specifiek wordt er gebruik gemaakt van een Windows Server 2016 met GUI, waarop Docker for Windows wordt geïnstalleerd. Deze Linux-omgeving bestaat uit een CentOS 7.4 waarop Docker for Linux wordt geïnstalleerd. Op beide platformen worden daarna twee containers uitgerold. De eerste container is een Microsoft SQL server en de tweede container bevat een .NET webapplicatie.

Ten eerste werd de documentatie vergeleken tussen beide omgevingen op vlak van volledigheid en gebruiksgemak. Vervolgens werden beide systemen getest op de uitvoeringstijd bij een volledige installatie en bij de installatie van de containers. Voor de installatie van beide systemen werd gebruik gemaakt van Vagrant, zodat de installatie van het besturings-systeem, de benodigde applicaties en de containers zoveel mogelijk geautomatiseerd werd. Dit maakte het mogelijk om op een methodologische manier de performantie te testen van beide systemen. Nadien werd er een vergelijkende studie uitgevoerd tussen de veiligheid van de CentOS-omgeving en van de Windows-omgeving. Van Linux is geweten dat de veiligheid goed is als men alle functionaliteiten ervan gebruikt. De vraag was dus: welke mogelijkheden heeft Windows om dit te evenaren? Ten slotte volgt er een conclusie over welk systeem het beste past binnen DevOps-teams en hoe men het beste met beide omgaat.

De redenen waarom ik dit onderwerp koos zijn meervoudig. Eerst en vooral vind ik de Container-technologie uitermate fascinerend en ga ik hier zeker verder mee experimenteren na deze bachelorproef. Daarnaast was ik aangenaam verrast toen Docker besloot samen te werken met Microsoft om hun technologie beschikbaar te stellen voor het Windows OS. Dit past natuurlijk wel binnen het manifest van Microsoft CEO Satya Nadella om het bedrijf een nieuwe richting te doen inslaan. Ten slotte leek dit mij een uitermate geschikt onderwerp voor een bachelorproef. Er zijn namelijk al verschillende Blog-posts en artikels

hierover verschenen, maar de meeste raken alleen maar de oppervlakte van de zaak of erger: leveren geen concreet cijfermateriaal, maar slechts suggestieve meningen.

Het uitwerken van de bachelorproef ging vrij vlot, buiten een paar problemen met de webapplicatie en het automatiseren van de Windows-omgeving. Het schrijven verliep wat moeilijker, omdat ik nooit sterk ben geweest met taal. Echter, ondanks het moeilijke schrijven vond ik dit wel een aangename ervaring. Ik had namelijk nog nooit eerder een werk van zo'n kaliber mogen opleveren en hoewel het zwaar werken was, ben ik wel tevreden over het eindresultaat.

Ten slotte zou ik nog graag enkele mensen willen bedanken:

Als eerste zou ik mijn promotor Steven Vermeulen willen bedanken. Hij heeft me vooral op literair vlak een stevige duw in de goede richting gegeven met zijn feedback. Daarnaast zou ik ook graag mijn co-promotor willen bedanken: Gert Schepens. Zelfs tijdens drukke momenten heeft hij tijd kunnen vrijmaken voor mij en mijn bachelorproef. Vervolgens wil ik ook even mijn vriendin, Gwynn Brewée, in de bloemetjes zetten. Ze heeft mij meerdere malen geholpen bij het verbeteren van mijn tekst. Ook wil ik mijn mama bedanken voor haar onvoorwaardelijke steun doorheen mijn woelige schoolperiode. Ten slotte wil ik ook Bert Van Vreckem bedanken voor zijn hulp bij dit onderwerp. Zonder zijn hulp zou de webapplicatie misschien nog steeds niet werken. Via hem ben ik ook op het idee gekomen om dit onderwerp te behandelen.

Stephan Heirbaut 25 mei 2018, Lokeren

Samenvatting

Docker for Windows Server 2016 is uit beta gekomen op 22 februari 2017. Maar, ondanks het feit dat dit platform nu al een ruime tijd beschikbaar is, heeft het nog steeds geen tractie gevonden bij DevOps-teams. Dit ondanks het feit dat dit een krachtige tool kan zijn voor organisaties die ook Microsoft Certified Partners willen zijn. In een notendop zal er gekeken worden of Docker for Windows Server 2016 een goed alternatief is voor het draaien van Docker in een Linux-omgeving. Om dit op een methodologische manier te testen wordt er vertrokken vanuit een standaard Windows-Server 2016 en CentOS 7.4-installatie waarop Docker geïnstalleerd wordt, waarna beide omgevingen getest worden door middel van prestatie- en beveiligingstesten, om zo een vergelijking te maken van beide omgevingen. Daarbovenop zal ook de documentatie voor beide platformen bekeken worden op vlak van compleetheid.

De verwachting is dat het verschil tussen beide systemen minimaal zal zijn, maar dat de prestaties van de CentOS-server beter zullen zijn. De documentatie voor CentOS zal ook completer zijn, maar zowel Windows als Docker zijn vermoedelijk een grote inhaalbeweging aan het maken.

Verdere vragen die men hierna nog zou kunnen stellen zijn:

- Hoe goed scoren beide op vlak van user-friendliness?
- Hoe goed ondersteunen beide Cloud platforms?
- Hoe stabiel draaien beide omgevingen?

Inhoudsopgave

| | | |
|----------|------------------------------|-----------|
| 1 | Inleiding | 15 |
| 1.1 | Probleemstelling | 15 |
| 1.2 | Onderzoeksvraag | 15 |
| 1.3 | Onderzoeksdoelstelling | 16 |
| 1.4 | Opzet van deze bachelorproef | 16 |
| 2 | Stand van zaken | 19 |
| 2.1 | Stand van zaken | 19 |
| 2.1.1 | DevOps | 19 |
| 2.1.2 | Docker | 21 |
| 2.1.3 | Linux Containers | 25 |
| 2.1.4 | Hyper-V | 26 |
| 2.1.5 | Windows Job objects | 28 |

| | | |
|--------|------------------------|----|
| 2.1.6 | Silos | 28 |
| 2.1.7 | Virtual Box | 28 |
| 2.1.8 | Vagrant | 29 |
| 2.1.9 | PowerShell | 30 |
| 2.1.10 | Bash | 30 |
| 2.1.11 | Software testing | 31 |

3 Methodologie 33

| | | |
|-----|--------------------------------------|----|
| 3.1 | Literatuurstudie | 33 |
| 3.2 | Bekijken van documentatie | 33 |
| 3.3 | Opzetten van de servers | 33 |
| 3.4 | Uitvoeren van de verschillende tests | 34 |
| 3.5 | Vergelijkende studie uitvoeren | 34 |

4 Documentatie 35

| | | |
|-----|-----------------------------|----|
| 4.1 | Requirements | 35 |
| 4.2 | Installatie | 36 |
| 4.3 | Container tot stand brengen | 36 |
| 4.4 | Automatisatie | 37 |
| 4.5 | Conclusie | 37 |

5 Opstelling 39

| | | |
|-------|---------------------|----|
| 5.1 | Windows Server 2016 | 39 |
| 5.1.1 | Vagrant | 39 |
| 5.1.2 | PowerShell | 42 |

| | | |
|------------|--------------------------------|-----------|
| 5.2 | CentOS 7.4 | 45 |
| 5.2.1 | Vagrant | 45 |
| 5.2.2 | Bash | 46 |
| 6 | Performantie test | 49 |
| 6.1 | CentOS 7.4 | 50 |
| 6.1.1 | Performantie installatie | 50 |
| 6.1.2 | Performantie containers | 51 |
| 6.2 | Windows Server 2016 | 53 |
| 6.2.1 | Performantie installatie | 53 |
| 6.2.2 | Performantie applicatie | 54 |
| 6.3 | Conclusie | 56 |
| 7 | Security test | 57 |
| 7.1 | CentOS | 58 |
| 7.1.1 | Linux containers | 58 |
| 7.1.2 | SELinux | 58 |
| 7.1.3 | Firewalld | 59 |
| 7.1.4 | Seccomp | 59 |
| 7.2 | Windows | 60 |
| 7.3 | Hyper-V containers | 60 |
| 7.4 | Integrity levels | 60 |
| 7.5 | Windows Defender | 61 |
| 7.5.1 | ACL | 61 |
| 7.6 | Conclusie | 61 |

| | | |
|----------|---------------------------|-----------|
| 8 | Conclusie | 63 |
| A | Onderzoeksvoorstel | 65 |
| A.1 | Introductie | 65 |
| A.2 | State-of-the-art | 65 |
| A.3 | Methodologie | 66 |
| A.4 | Verwachte resultaten | 66 |
| A.5 | Verwachte conclusies | 66 |
| | Bibliografie | 67 |

Lijst van figuren

| | | |
|-----|--|----|
| 2.1 | Links ziet men hoe de communicatie in klassieke ontwikkelomgevingen gebeurt, rechts in DevOps-teams. | 20 |
| 2.2 | De DevOps manier van werken. | 21 |
| 2.3 | In onderstaande afbeeldingen ziet u het verschil tussen containers en VM's | 22 |
| 2.4 | De modules van Docker CE | 26 |
| 2.5 | Hoe men te werk gaat bij software testing | 31 |
| 5.1 | Het eerste deel van de configuratie van het Vagrantfile-bestand voor de Windows-opstelling | 40 |
| 5.2 | Het tweede deel van de configuratie van het Vagrantfile-bestand voor de Windows-opstelling | 41 |
| 5.3 | De index-pagina van het voorbeeldproject wanneer de containers opgezet zijn. | 41 |
| 5.4 | De inhoud van het Chocolatey script | 42 |
| 5.5 | De inhoud van het eerste Docker script. Deze installeert Docker. . | 42 |
| 5.6 | De inhoud van het tweede Docker script. Deze installeert Python en Pip, creëert een test-container en voegt de Firewall-regels toe | 43 |
| 5.7 | Pip wordt aangeroepen om docker-compose te installeren. | 43 |
| 5.8 | Hiermee worden de Docker Images opgehaald van de Cloud ... | 43 |

| | | |
|------|---|----|
| 5.9 | Het creëren van het Dockerfile-bestand en docker-compose-bestand. Alsook het opvullen van het Dockerfile-bestand. | 44 |
| 5.10 | Het opvullen van het docker-compose-bestand. Alsook het aanroepen van 'docker-compose build' en 'docker-compose up -d'. | 44 |
| 5.11 | Het eerste deel van de configuratie van het Vagrantfile-bestand voor de CentOS-opstelling. | 45 |
| 5.12 | Het tweede deel van de configuratie van het Vagrantfile-bestand voor de CentOS-opstelling. | 45 |
| 5.13 | Het eerste deel van de vereiste functies voor de CentOS-opstelling. | 46 |
| 5.14 | Het tweede deel van de vereiste functies voor de CentOS-opstelling. | 47 |
| 5.15 | Hier worden de Docker Images opgehaald. | 47 |
| 5.16 | Hier wordt het Dockerfile- en docker-compose-bestand opgevuld. Waarna 'docker-compose build' en 'docker-compose up -d' wordt uitgevoerd. | 48 |
| 6.1 | De scatterplot voor de resultaten van 'time vagrant up' voor de CentOS-opstelling | 50 |
| 6.2 | De boxplot voor de resultaten van 'time vagrant up' voor de CentOS-opstelling | 51 |
| 6.3 | De scatterplot voor de resultaten van 'time vagrant provision' voor de CentOS-opstelling | 52 |
| 6.4 | De boxplot voor de resultaten van 'time vagrant provision' voor de CentOS-opstelling | 52 |
| 6.5 | De scatterplot voor de resultaten van 'time vagrant up' voor de Windows-opstelling | 53 |
| 6.6 | De boxplot voor de resultaten van 'time vagrant up' voor de Windows-opstelling | 54 |
| 6.7 | De scatterplot voor de resultaten van 'time vagrant provision' voor de Windows-opstelling | 55 |
| 6.8 | De boxplot voor de resultaten van 'time vagrant provision' voor de Windows-opstelling | 55 |

Lijst van tabellen

| | | |
|-----|--|----|
| 2.1 | De voornaamste Docker commando's voor het beheren van Docker Images en Boxes | 24 |
| 6.1 | De resultaten van 'time vagrant up' voor de CentOS-opstelling. . | 50 |
| 6.2 | De resultaten van 'time vagrant provision' voor de CentOS-opstelling. | 51 |
| 6.3 | De resultaten van 'time vagrant up' voor de Windows-opstelling. | 53 |
| 6.4 | De resultaten van 'time vagrant provision' voor de Windows-opstelling. | 54 |

1. Inleiding

Ten eerst zal in deze inleiding het onderwerp van deze bachelorproef worden afgebakend. Vervolgens wordt de context en nood van de bachelorproef uitlegt. Daarnaast zullen de probleemstelling, onderzoeksvragen en onderzoeksdoelstelling besproken worden. Ten slotte wordt ook de opzet van deze bachelorproef worden toegelicht.

1.1 Probleemstelling

Het grootste probleem bij Docker is dat het op zich nog een relatief jonge technologie is, die bovendien een eigen syntaxis heeft. Daarnaast is er een andere denk- en werkwijze vereist om Docker te beheersen. Doorheen de tijd zijn Linux-administrators reeds vertrouwd geraakt met deze technologie, maar voor Windows-administrators is deze technologie nog gloednieuw.

Dit onderzoek zal vooral een meerwaarde bieden voor DevOps-teams of bedrijven die op zoek zijn naar Windows-oplossingen voor hun problemen in verband met automatisatie en continue oplevering. Zoals bijvoorbeeld mijn stagebedrijf Orbid. (Steven, 2018)

1.2 Onderzoeksvraag

De onderzoeksvragen die uit deze probleemstelling voortvloeien zijn de volgende:

- Hoe vlot kan men een Docker-opstelling maken op een Windows-besturingssysteem tegenover een Linux-besturingssysteem?

- Hoe is het gesteld met de documentatie voor Docker for Windows tegenover de bestaande documentatie voor Linux?
- Hoe groot is de snelheidswinst bij Linux tegenover Windows?
- Hoe is het gesteld met de veiligheid? Hoe pakt men dit aan vanuit een Windows-administrator perspectief?

Op al deze vragen kon tot op heden geen afdoend antwoord worden gegeven. Men heeft in het verleden wel al Docker for Windows uitgetest en vergeleken met Linux, maar nooit op een concrete en methodische manier. Vaak benaderde men deze technologie ook vanuit een bestaande mening, en niet vanuit een neutraal perspectief.

1.3 Onderzoeksdoelstelling

Het beoogde resultaat van deze bachelorproef is een concrete vergelijkende studie tussen Docker for Linux en Docker for Windows. Waarbij er vooral gekeken wordt naar de volgende aspecten:

- Documentatie
- Performatie
- Veiligheid

1.4 Opzet van deze bachelorproef

De rest van deze bachelorproef is als volgt opgebouwd:

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

In Hoofdstuk 3 wordt de methodologie toegelicht en worden de gebruikte onderzoekstechnieken besproken om een antwoord te kunnen formuleren op de onderzoeksvragen.

In Hoofdstuk 4 wordt de documentatie van beide platformen besproken. Er wordt gekeken naar volledigheid, interne en externe bronnen, en hoeveel ondersteuning er is vanuit de hoofdorganisatie.

In Hoofdstuk 5 worden beide opstellingen bekeken en besproken, meer specifiek hoe het is om beide op te bouwen en hoe het zit met gelijkenissen en verschillen.

In Hoofdstuk 6 wordt gekeken naar het resultaat van de werklading die beide systeem te verduren hebben gekregen en hoe beide gepresteerd hebben.

In Hoofdstuk 7 worden de beschikbare veiligheidsmaatregelen voor Docker besproken binnen beide systemen en hoe effectief deze zijn.

In Hoofdstuk 8, tenslotte, wordt de conclusie gegeven en een antwoord geformuleerd op

de onderzoeksvragen. Daarbij wordt ook een aanzet gegeven voor toekomstig onderzoek binnen dit domein.

2. Stand van zaken

Voor 22 februari 2017 kon men enkel Docker installeren op Linux-besturingssystemen. Ondernemingen zonder Linux-kennis die wilden evolueren naar DevOps werden op die manier gedwongen om iemand in dienst te nemen die deze kennis wel had, ofwel om zelf een spoedcursus te volgen. Echter met de komst van Windows 10 en Windows Server 2016, en de daarmee gepaard gaande groei van PowerShell en Hyper-V, is Docker nu ook toegankelijk voor organisaties die meer Windows-gezind willen zijn. (T. Brown, 2017)

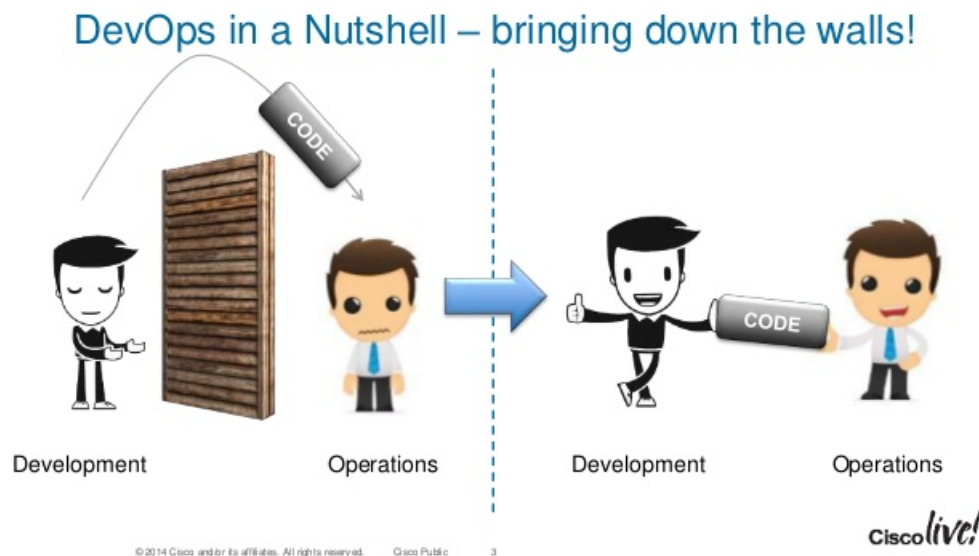
Het zou namelijk een vergissing zijn om Docker zomaar te negeren als men de DevOps-richting wil uitgaan. Het biedt immers verschillende tools aan voor zowel developers als system administrators. Bijvoorbeeld: Containers as a Service (CaaS) en role-based access control voor Operations (system administrators), en een zelfbedieningsmanier van werken voor Developers waarbij ze services kunnen opvragen wanneer ze die nodig hebben. Momenteel wordt Docker gebruikt in 44 procent van de ondernemingen die van plan zijn om de DevOps-richting uit te gaan. (Christopher, 2017)

2.1 Stand van zaken

2.1.1 DevOps

DevOps is een samenstelling van de woorden 'Development' en 'Operations': ontwikkeling en beheer. Voorheen werkten deze IT-groeperingen strikt gescheiden, waardoor men ofwel bij de ene groep zat, ofwel bij de andere. Dit zorgde er voor dat, als men een applicatie wou maken, de developers deze dan eerst ontwikkelden, waarna de systeembeheerders de applicatie uitrolden. Als de systeembeheerders hierna nog problemen

Figuur 2.1: Links ziet men hoe de communicatie in klassieke ontwikkelomgevingen gebeurt, rechts in DevOps-teams.



hadden met het uitrollen, verliep de communicatie veelal stroef. De applicatie werkte immers op de computers van de developers, dus waarom zou dat niet het geval zijn bij de systeembeheerders? Echter, aangezien beide groeperingen onvoldoende kennis hadden over elkaars werkveld, was er veel onbegrip.

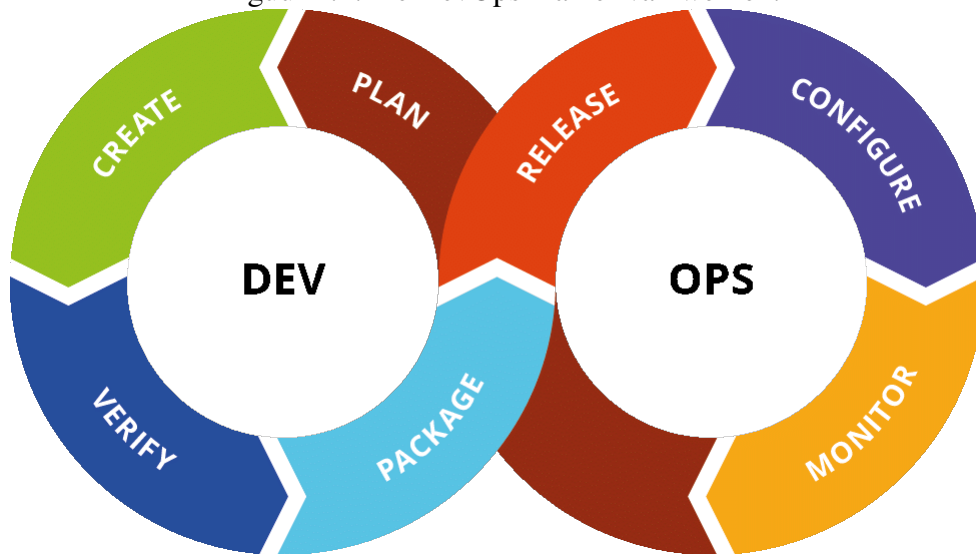
Als éénduidige definitie voor DevOps gebruikt met het acroniem 'CALMS'.

- Culture (Cultuur)
- Automatisation (Automatisatie)
- Learning (Leren, voorheen Lean)
- Measure (Meten)
- Sharing (Delen)

Bij DevOps worden deze groepen gedwongen om samen te werken in één team, zodat het geheel groter wordt dan de som. In praktijk vertaalt dit zich naar een cultuur die gestimuleerd dient te worden, waarin het automatiseren van zoveel mogelijk zaken centraal staat met een focus op continue ontwikkeling en oplevering (cfr. CI/CD). Hierbij is het leren, delen van informatie, en communicatie in het algemeen, heel belangrijk. Er worden verschillende manieren voorzien om informatie te verzamelen uit de applicatie, door methodes te implementeren die men gebruikt om een doorzichtiger systeem te creëren. Het is dus eerder een groep van concepten en ideeën die uitgegroeid zijn tot een beweging die steeds meer veld wint.

(Ernest, 2017)

Figuur 2.2: De DevOps manier van werken.



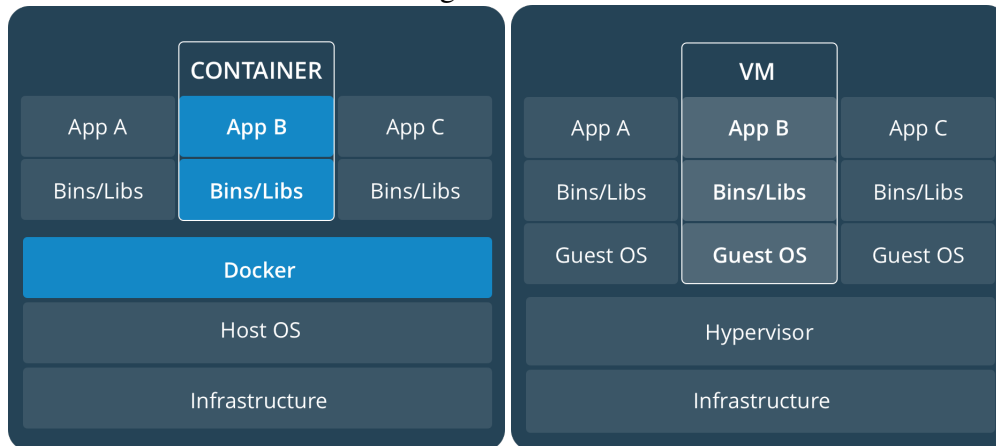
2.1.2 Docker

De beste manier om de verschillen in werking te beschrijven tussen fysieke servers, virtuele machines (VM's) en containers, is door ernaar te kijken als een woning. Fysieke servers en virtuele machines zijn zoals huizen. Ze werken volledig onafhankelijk en voorzien zelf in al hun noden qua infrastructuur. Een fysieke server wordt bovendien volledig onafhankelijk opgebouwd vanaf nul. Een virtuele machine echter wordt naast een bestaand host-besturingssysteem gedraaid, waarbij het een bepaalde hoeveelheid infrastructuur opeist. Een virtuele machine is dus eigenlijk een computer in een computer. Beide systemen hebben hetzelfde grote nadeel, namelijk dat er veel meer infrastructuur nodig is en dat deze niet optimaal wordt gebruikt. Containers zijn eerder te vergelijken met appartementen, in die zin dat de infrastructuur wordt opgedeeld. Alle middelen van de host-machine waar de Docker Daemon op geïnstalleerd wordt, worden verdeeld onder de verschillende containers. Elke container vraagt hierbij alleen de nodige grondstoffen om te voldoen aan de noden van zijn bewoner, net zoals bij een appartement. Containers lijken op het eerste zicht erg op virtuele machines, maar het is belangrijk om te onthouden dat de onderliggende architectuur sterk verschilt tussen beide.

Doordat een Container alleen de infrastructuur opeist die hij nodig heeft, bouwt die een lichter en onafhankelijk uitvoerbaar softwarepakket op, dat consistentie kan garanderen omdat het telkens dezelfde achterliggende Docker Images en Daemon gebruikt. Een van de meest voorkomende problemen bij het uitrollen van een applicatie is het verschil tussen de test- en productieomgeving, en de hiermee gepaarde infrastructuurlast. (Uhrig, 2014)

Docker is een tool om deze Containers te bouwen. Hiermee kan men het uitrollen van applicaties sneller en makkelijker maken, doordat het de applicatie en al haar behoeften eerst in één geautomatiseerd pakket verzamelt. Dit pakket noemt men een image. De uitgerolde versie van een image noemt men een Container en deze zal los van het host-besturingssysteem werken.

Figuur 2.3: In onderstaande afbeeldingen ziet u het verschil tussen containers en VM's



Om een Container op te bouwen van begin tot einde maakt men gebruik van de Docker Client. Via de Client kan men communiceren met de Docker Daemon. De Docker Daemon is een proces dat op de achtergrond draait tot het aangeropen wordt door de gebruiker via de syntaxis van Docker. De Docker Daemon is uiteindelijk degene die de Docker commando's interpreteert en uitvoert, zoals het bouwen, draaien en onderhouden van containers.

Bij het bouwen van een container via Docker zijn er 3 werkmethodes te onderscheiden, namelijk een standaard-, een minimalistische- en een geautomatiseerde methode. De minimalistische methode wordt gebruikt bij installaties die geen customisation vereisen, bijvoorbeeld een SQL-server. De standaardmethode wordt gebruikt bij alle andere installaties, bijvoorbeeld een CentOS-server. De standaardmethode bestaat uit 4 stappen, dewelke hieronder meer uitgebreid besproken worden. Bij de minimalistische methode worden enkel de eerste en de laatste stap uitgevoerd. (Steven, 2018) (Jean-François, 2017)

- docker pull

De eerste stap is docker pull. Hiermee kan men een Docker Image ophalen van de Docker Hub. Deze images zijn momentopnames van bijvoorbeeld containers of besturingssystemen. Deze kunnen, afhankelijk van de gebruiker, uitgebreid worden via een Dockerfile. Het is met deze parent image dat men een container kan bouwen.

- Dockerfile

De tweede stap is Dockerfile, waarin men aan de hand van de syntaxis een gepersonaliseerde Docker Image kan definiëren. Men kan aangeven vanaf welke image men wil beginnen via het 'FROM'-commando. Men hoeft hier echter niet per se een parent image mee te geven, bijvoorbeeld via 'FROM scratch'. Vervolgens kan men environment variables meegeven via 'ENV', 'ADD' of 'EXPOSE'. Daarnaast kan men via 'RUN' ook orders meegeven om specifieke commando's uit te voeren die de container zou herkennen, zoals Bash of PowerShell. Ten slotte kan men via 'ENTRYPOINT' aangeven of de container uitvoerbaar moet zijn of niet.

- docker build

De derde stap is docker build, waarmee men opnieuw de Docker Daemon aanspreekt. Op die manier kan die de syntaxis interpreteren en een gepersonaliseerde container bouwen.

- docker run <options> <image>

Ten slotte is er docker run, waarmee men het bevel geeft aan de Docker Daemon om de image uit te rollen tot een container. Hoe de image zich uitrolt, hangt af van de bevelen in het commando en van de image die gebruikt wordt. (TutorialsPoint, 2018)

Bij de geautomatiseerde methode maakt men ook gebruik van de 4 hierboven beschreven stappen, maar voegt men een extra component toe onder de vorm van docker-compose.yml. In dit bestand kan men meerdere services beschrijven waarvan Docker containers moet maken. Daarnaast dient dit bestand geplaatst te worden in dezelfde folder als het 'dockerfile'-bestand. Ten slotte worden alle services die hierin beschreven staan in orde gebracht door de uitvoering van het 'docker-compose up'-commando.

Docker-compose is daarom vooral handig wanneer men meerdere services in orde wil brengen die afhankelijk zijn van elkaar, zoals een webapplicatie en de bijhorende SQL-server. Hieronder ziet u een voorbeeld van een docker-compose.yml-bestand. (Janetakis, 2017)

backend:

image: redis:3

restart: always

frontend:

build: commander

links:

- backend:redis

ports:

- 8081:8081

environment:

- VAR1=value

restart: always

Docker voorziet ook commando's om de Docker Images en Boxes te beheren. De voorname staan hieronder beschreven.

| Commando | Uitleg |
|---|---|
| <code>docker images</code> | toont alle Docker Images beschikbaar op dit systeem |
| <code>docker ps</code> | toont alle draaiende containers |
| <code>docker inspect <container></code> | toont de specificaties van een specifieke container |
| <code>docker logs <container></code> | toont alle logs voor een specifieke container |
| <code>docker start <container></code> | start een specifieke container op |
| <code>docker stop <container></code> | stopt een specifieke container |
| <code>docker rm <container></code> | verwijdert containers |

Tabel 2.1: De voornaamste Docker commando's voor het beheren van Docker Images en Boxes

(Grubor, 2017)

Doordat elk deel van een applicatie in een 'appartementje' zit, is het leveren van continue ontwikkeling en oplevering voor de applicatie heel gemakkelijk. Men moet immers alleen die specifieke Container updaten/graden. Hierdoor is het de perfecte technologie om te dienen als test- en productieomgeving. Daarnaast versoepelt het ook de communicatie binnen het DevOps-team, doordat iedereen continu in dezelfde omgeving werkt. Er zijn immers geen grote veranderingen nodig bij de verschillende stappen binnen de leveringsketen, omdat men in staat is om verschillende frameworks te draaien op één platform. Containers zijn namelijk vrij agnostisch op vlak van programmeertaal of platform. Ten slotte kunnen Containers ook makkelijk van host-besturingssysteem verplaatst worden. Al deze kenmerken van Docker Containers vergemakkelijken het werk van DevOps-teams aanzienlijk. (Mike, 2016)

Door al deze redenen blijft Docker ook jaar na jaar groeien. Tijdens de recentste DockerCon heeft de CEO van Docker dit ook aangetoond met volgend cijfermateriaal:

- Meer dan 14 miljoen Docker-hosts;
- Meer dan 900.000 applicaties die draaien op Docker;
- Een verhoging van 40 procent in het toepassen ervan binnen 1 jaar.

(Cloud, 2017)

Docker heeft deze groei ook verdiend door constant in contact te blijven met zijn gebruikers. In het verleden kregen ze immers de klacht dat ze te snel of te traag updates toevoegden aan het systeem. Om dit te remediëren hebben ze naast Docker Community Edition, waarmee alles is begonnen, nu ook Docker Enterprise Edition ontworpen. Daarbovenop is er een verschuiving naar time-based updates. Hierbij kan men kiezen tussen een maandelijks update, maar zonder garantie op stabiliteit van de runtime (Edge), ofwel voor een viermaandelijks update voor gebruikers die meer stabiliteit prefereren, zowel op vlak van onderhoud als op vlak van beheer (Stable). Daarbovenop heeft de Enterprise Edition ook een meer uitgebreide ondersteuning met verschillende gradaties. (Ramandeep, 2018)

Docker CE

Zoals eerder aangehaald is Docker Community Edition het originele platform waarmee Docker begonnen is. Het is ideaal voor kleinere teams of onafhankelijke developers die willen experimenteren met Container-technologie. Deze technologie is beschikbaar voor zowel Linux als Mac, waardoor het perfect is als kleine en snelle installatie waarmee men direct aan de slag kan gaan. Ten slotte biedt het ook ondersteuning voor het uitrollen naar Cloud-omgevingen, zoals Amazone Web Services of Azure. (Nick, 2017)

Docker EE

Docker Enterprise Edition is het volledige pakket voor zij die op een professionele manier met een Containers-as-a-Service-platform willen werken. Docker EE is een geïntegreerd en getest platform voor Linux- of Windows Enterprise en Cloud-providers, met door Docker gecertificeerde componenten en ondersteuning. In het bijzonder is het voor datacenters een handig dashboard waar men de zogenaamde multi-architecture orchestration, secure software supply chain en infrastructure independence aangeboden krijgt. Vooral aan dat laatste heeft Docker extra veel aandacht besteed. (Vivek, 2018)

Docker for Windows

Daarnaast heeft Docker ook een versie ontwikkeld die specifiek gericht is op Developers die gebruik willen maken van Windows als ontwikkelplatform. Docker for Windows maakt gebruik van dezelfde Daemon en Client als Docker for Linux, maar met een paar verschillen. Docker for Windows maakt bijvoorbeeld gebruik van de Windows-native Hyper-V Virtualisatie, wat betekent dat de gebruiker een Windows 8+ Pro edition of een Windows Server 2008+ nodig heeft, aangezien alleen deze versies van Windows over Hyper-V beschikken. Daarnaast is de enige manier om Docker CE for Windows te installeren door gebruik te maken van een GUI-installatie. Men kan gebruik maken van PowerShell-commando's zoals Invoke-WebRequest, maar er is geen native PowerShell-commando voor het installeren van Docker for Windows. Docker EE for Windows kan dan wel weer via de commandline geïnstalleerd worden. Ten slotte wordt docker-compose ook automatisch geïnstalleerd als men Docker CE for Windows installeert. Het maakt namelijk deel uit van het totaalpakket. Dit in tegenstelling tot Docker EE for Windows, waarbij docker-compose wél nog apart geïnstalleerd moet worden.

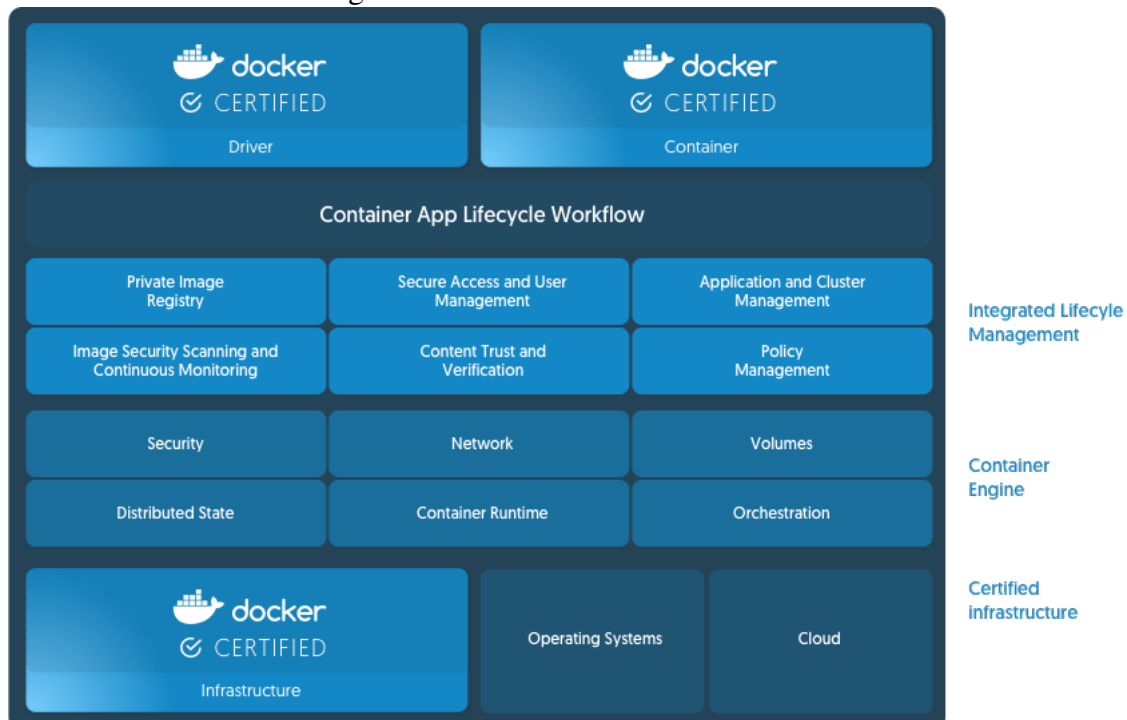
(Fulton, 2017)

2.1.3 Linux Containers

Docker CE for Linux maakt gebruik van de technologie achter Linux Container voor het maken van zijn Docker Containers.

Linux containers hebben hun eigen netwerk resources, geheugen, geïsoleerde CPU en I/O block, maar ze gebruiken wel dezelfde kernel als het host-besturingssysteem. Hierdoor

Figuur 2.4: De modules van Docker CE



kunnen ze lichte en gestroomlijnde containers bouwen. Cgroups en Namespaces maken dit mogelijk. (Wang, 2017)

Cgroups

Om de isolatie van system resource, zoals CPU en geheugen, te controleren heeft Linux cgroups. Cgroups kan processen groeperen zodanig dat deze maar toegang krijgen tot een beperkte hoeveelheid resources. (Weissig, 2013)

Namespaces

Als men resource isolation wil voor één proces, kan men gebruik maken van Namespaces. Er zijn hiervoor verschillende opties zoals: PID (process id), networking namespace of mount namespace. (Evans, 2016)

2.1.4 Hyper-V

Hyper-V is één van de technologieën die gekozen werd om de twee servers, CentOS en Windows Server 2016, op te laten draaien. Dit is het Microsoft-platform voor het virtualiseren van hardware. Het maakt gebruik van hypervisor-gebaseerde virtualisatietechnologie om de interactie tussen de virtuele machine en de hardware te voorzien. Hierdoor is men in staat om een andere computer te laten werken bovenop het host-besturingssysteem. De

voornaamste componenten en hun functies van dit platform zijn: (Cooley, 2018)

- Windows Hypercall

Hypercall voorziet communicatie met de Hypervisor.

- Windows hypervisor

Dit is een softwarelaag wiens primaire verantwoordelijkheid is om geïsoleerde execution environments te voorzien. Deze execution environments noemen partitions. Hypervisor is gelegen tussen de hardware en één of meer besturingssystemen, zodat het hardwaregebruik kan optimaliseren voor al de partitions.

- Hyper-V Virtual Machine Management Service

VMMS is verantwoordelijk voor het beheren van de staat van alle virtuele machines in zogenaamde child partitions.

- Virtual Machine Bus

Dit channel-based communicatiemechanisme voorziet de communicatie tussen verschillende partitions op systemen met meer dan één virtual partition.

- Virtualization Service Provider

VSP bevindt zich in de root partition en voorziet synthetische ondersteuning aan child partitions over de VMBus.

- Virtual Infrastructure Driver

De VID voorziet partition-, processor- en memory services.

Daarnaast bevat Hyper-V ook nog andere low level componenten, zoals:

- APIC - Advanced Programmable Interrupt Controller
- IC - Integration component
- MSR - Memory Service Routine
- VMWP - Virtual Machine Worker Process
- VSC - Virtualization Service Client
- WinHv - Windows Hypervisor Interface Library
- WMI - Virtual Machine Management Service

Ten slotte heeft Hyper-V ook enkele tools voor het beheren van en verbinden met zijn virtuele machines. Deze kunnen geïnstalleerd worden op zowel computers waar Hyper-V op staat, als op externe computers. Deze zijn meer specifiek:

- Hyper-V Manager
- Hyper-V module for Windows PowerShell
- Virtual Machine Connection

- Windows PowerShell Direct

Om deze servers te verbinden met het internet wordt ook gebruik gemaakt van de Hyper-V Virtual Switch. Deze laag 2 Ethernet netwerkswitch voorziet programmeer- en uitbreidbare mogelijkheden om het beheren van virtuele en fysieke netwerken te vergemakkelijken. Het voorziet veiligheid, isolatie en service levels om de nodige veiligheid af te dwingen. (Juarez, 2018)

Hyper-V is een type 1 Hypervisor. Dit betekent dat het opstarten van een virtuele machine als volgt gaat:

- Eerst wordt de fysieke machine opgestart.
- Vervolgens wordt de controle van het boot-systeem van de fysieke machine overgedragen aan Hyper-V, bijvoorbeeld BIOS of EUFI.
- Daarna start Hyper-V het management operating system op.
- Ten slotte maakt Hyper-V partitions aan voor de virtuele machines, afhankelijk van de boot- of gebruikersinstructies.

Daarnaast draait een type 1 Hypervisor ook direct op de hardware. Daarom worden ze ook wel 'native', 'bare metal' of 'embedded' Hypervisors genoemd.

2.1.5 Windows Job objects

Windows Job Objects zijn het equivalent van cgroups. Net zoals cgroups groeperen ze processen en voeren ze daarna resource controls uit op deze groep, zodat deze bijvoorbeeld de CPU niet te veel in beslag nemen voor andere processen.

2.1.6 Silos

Windows Silos zijn dan weer het equivalent van namespaces. Deze zorgen voor een isolation execution environment voor file system, registry en object namespaces. (Fulton, 2017)

2.1.7 Virtual Box

Virtual Box is de tweede Hypervisor die ondersteund wordt in deze bachelorproef. Deze wordt ontwikkeld door Oracle Corporation, maar is wel een type 2 Hypervisor. Type 2 Hypervisors gedragen zich meer als een gewone software-applicatie en worden daarom 'host' Hypervisors genoemd. Ze maken een abstractie van de verbinding met de hardware en sturen hiervoor requests naar het host OS, net zoals gewone applicaties.

De opstartvolgorde is hierdoor ook anders:

- Eerst wordt de fysieke machine gestart.
- Vervolgens wordt de controle van het boot-systeem van de fysieke machine overge-

dragen aan Virtual Box, bijvoorbeeld BIOS of EUFI.

- Daarna start een gebruiker de Virtual Box-applicatie op.
- Ten slotte zal de gebruiker kiezen welke virtuele machines worden opgestart, waarbij Virtual Box de benodigde hosting processes aanmaakt.

Ten slotte bestaat Virtual Box uit een soortgelijke lijst van componenten, waarbij er toch kleine wijzigingen zijn omdat het een type 2 is.

- IPRT (a portable runtime library)
- Virtual Machine Monitor
- Execution Manager
- Recompiled Execution Monitor
- Trap Manager
- Hardware Acceleration Manager
- Guest Interface Manager
- Pluggable Device Manager
- Page Manager
- Patch Manager
- Time Manager
- Configuration Manager
- Saved State Manager
- Virtual USB
- Debug Facility

(Siron, 2017)

2.1.8 Vagrant

Vagrant is een tool voor het bouwen en onderhouden van virtuele machines. Het maakt gebruik van een enkele workflow voor het automatiseren van ontwikkelomgevingen. Doordat het zo makkelijk te gebruiken is, verlaagt het de installatietijd en verhoogt het de productie. Dit is waarom het een geliefde tool is binnen DevOps-teams.

Vagrant werkt aan de hand van een configuratiebestand dat uitgelezen moet worden. Het configuratiebestand dat Vagrant nodig heeft, heet een 'Vagrantfile'. Hierin wordt alles gedefinieerd dat nodig is, meer bepaald: welke box, scripts en andere bestanden er nog nodig zijn voor de installatie.

Een 'Vagrant Box' is een verpakte base image die een clone is van een virtuele machine. Een base image is het minimum van een besturingssysteem waarop er nog verschillende andere functies gebouwd kunnen worden, zoals bijvoorbeeld een GUI. Via deze box wordt een besturingssysteem en bijbehorende software geïnstalleerd. Deze boxes worden opgehaald door middel van 'config.vm.box = boxnaam'. Als de box nog niet eerder gebruikt werd, wordt deze eerst gedownload vanop Vagrant Cloud, waarna de box wordt opgeslagen zodat deze steeds beschikbaar is voor later gebruik.

Als de box goed is, maar er moet nog verdere installatie of configuratie van de software

gebeuren, kan men deze ook meegeven met het 'provision'-gedeelte. Met provisioning voegt men verdere configuratiemogelijkheden toe, zoals het uitvoeren van scripts, het uploaden van bestanden of folders tijdens de installatie, of het specificeren van een netwerkadres. Door provisioning kan men ook makkelijk wijzigingen aanbrengen aan het systeem tijdens de run-time. Er wordt ook gebruik gemaakt van een eigen syntax die aangeleerd dient te worden. Deze bestaat uit 'vagrant' en de functie die uitgevoerd dient te worden, bijvoorbeeld: met 'vagrant up' wordt geprobeerd om het systeem online te brengen. De configuratie van de 'Vagrantfile' heeft ook zijn eigen syntax, bestaande uit het woord 'config' en de functie bepalend wat ermee moet gebeuren.

Vagrant ondersteunt bovendien verschillende platformen om zijn installatie op uit te voeren, gaande van VirtualBox en Hyper-V tot Docker en AWS. Het grote aantal beschikbare platformen om op te releasen is ook een sterkte van Vagrant.

Daarbovenop zijn er mogelijkheden om de functionaliteit van Vagrant zelf uit te breiden door gebruik te maken van plug-ins die geschreven worden door andere gebruikers, of door gebruik te maken van andere Configuration Management tools, zoals Ansible. Plug-ins worden meestal geschreven in de programmeertaal Ruby. Voor de gebruikte Windows Server-opstelling in deze bachelorproef is deze functionaliteit heel handig, omdat de Windows Server herstart dient te worden na de installatie van Docker. Hiervoor kan een plug-in gebruikt worden.

Omwille van al deze redenen werd geopteerd om gebruik te maken van deze technologie. Er zijn gelijkaardige zero-touch installatiemogelijkheden, zoals Chef, Puppet of Terraform. Echter geen van hen biedt hetzelfde gebruiksgemak of dezelfde mogelijkheden. (Sturgeon, 2012)

2.1.9 PowerShell

De configuratie van de Windows Server gebeurt grotendeels aan de hand van deze command-line interface. Deze CLI, die bovendien gebruik maakt van het .NET framework om objecten terug te geven, is speciaal ontwikkeld voor system administrators. Een groot voordeel hierbij is dat objecten veel makkelijker te verwerken zijn, in tegenstelling tot andere CLI's die vaak tekst gebruiken voor het uitvoeren van commando's. Daarnaast is de structuur van de command-lets (cmdlet) simpel en makkelijk aan te leren. De commando's bestaan namelijk altijd uit dezelfde samenstelling, bestaande uit 'werkwoord'-'te behandelen onderwerp', bijvoorbeeld: Get-Service. (Pichardo, 2014)

2.1.10 Bash

Als tegenpool is er Bash voor Linux-besturingssystemen. Deze CLI dateert al van 1989, met als volledige naam 'Bourne Again shell'. In essentie werkt Bash op dezelfde wijze als PowerShell. Net zoals PowerShell herkent Bash namelijk een aantal woorden, waardoor men ermee kan communiceren en het acties kan laten uitvoeren. Het grote verschil ligt in de uitvoering: Bash heeft namelijk enkel tekst als in- en uitvoer, wat het verwerken van

Figuur 2.5: Hoe men te werk gaat bij software testing



opdrachten kan bemoeilijken. Echter, omdat Bash reeds geruime tijd bestaat, kan deze Unix shell wel genieten van een enorme gebruiks- en ondersteuningsbasis. Bash is de werkmethode bij uitstek voor Linux-administrators. (Billemont, 2018) (Baker, 2018)

2.1.11 Software testing

Bij software testing ligt de focus op kwaliteit in alle delen van de organisatie. Dit is belangrijk omdat het ontwikkelingsteam verantwoordelijk is voor de kwaliteit. QA-teams kunnen op tijd aan de alarmbel trekken, maar de verantwoordelijkheid van kwaliteit ligt nog steeds bij het ontwikkelingsproces. Daarom eist software testing dat iedereen, inclusief de gebruiker, betrokken is bij het ontwikkelingsproces. Software testing heeft een methodologische manier ontwikkeld om de kwaliteit te waarborgen, waarbij men laag per laag en stap per stap gaat kijken of het systeem voldoet aan de vereisten.

Verder zegt !!!BRON!!! over software testing dat het een empirische en technische manier is om software of services te testen, doordat men op een objectieve manier kijkt naar de kwaliteit van een software of service. Dit is de voornaamste reden waarom voor deze methode gekozen is in deze bachelorproef. Daarnaast sluit het heel goed aan bij de principes van het DevOps manifeste.

Meer specifiek wordt er gebruik gemaakt worden van performantie- en veiligheidstesten.

Performantietesten zijn testen waarbij men gaat kijken hoe het systeem presteert wanneer het een bepaalde werkdruk moet verwerken, met de focus op snelheid, schaalbaarheid en stabiliteit. (Noga, 2016)

Bij veiligheidstesten wordt er gekeken naar alle verschillende veiligheidsmaatregelen, en in hoeverre ze in staat zijn om de kwetsbaarheden van het systeem weg te werken. (Mark, 2012) (ThinkSys, 2017)

(Maurice, 2018) (Kaner, 2006)

3. Methodologie

In dit hoofdstuk wordt uitgelegd hoe er te werk is gegaan. Elke titel stelt een grote fase voor in deze bachelorproef. Er wordt kort toegelicht welke stappen er ondernomen zijn in deze fasen en vooral waarom.

3.1 Literatuurstudie

Het startpunt van deze bachelorproef was een literatuurstudie over Docker for Windows, waarvan het resultaat zichtbaar is in de inleiding. Specifiek werd opgezocht hoe Docker werkt en welke requirements nodig zijn om het succesvol te installeren op een Windows Server 2016 en CentOS 7.4.

3.2 Bekijken van documentatie

Vervolgens werd de verzamelde documentatie voor beide omgevingen bekeken en beoordeeld op volledigheid. Op basis van deze gegevens werd een conclusie getrokken.

3.3 Opzetten van de servers

Hierna werd een Windows Server 2016 en CentOS 7.4 gemaakt met op beide Docker geïnstalleerd, door middel van Vagrant en PowerShell voor Windows Server, en Bash voor CentOS. Op deze twee omgevingen werden uiteindelijk ook de software tests uitgevoerd.

Deze omgevingen werden daarna geüpload op GitHub zodat de code vrij beschikbaar was voor iedereen.

3.4 Uitvoeren van de verschillende tests

Nadat het opstellen van beide omgevingen was afgerond en de systemen volledig automatisch vanaf nul naar draaiend konden worden gebracht aan de hand van een werkende applicatie, werden op beide 2 manieren van software testing uitgevoerd, namelijk:

- Performance testing
- Security testing

Deze manieren werden verschillende malen uitgevoerd, zodat het populatiegemiddelde groot genoeg was om een kleinere invloed te ondervinden van outliers.

3.5 Vergelijkende studie uitvoeren

Ten slotte werden alle resultaten van de testresultaten verzameld en verwerkt, om het gemiddelde, de variantie en de standaardafwijking te verkrijgen. Deze waarden werden nadien gebruikt om een grafische weergave te creëren van de resultaten.

4. Documentatie

Docker is al enkele jaren beschikbaar voor Linux en in die tijdspanne hebben de verschillende gebruikers van het systeem al gretig onderzocht hoe ze deze nieuwe technologie het beste kunnen gebruiken. Echter, nog belangrijker is dat de gebruikers ook documenteren wat ze doen en hoe ze bepaalde resultaten behalen, bijvoorbeeld via blogs of issues op GitHub. Hierdoor wordt er niet alleen een indrukwekkende hoeveelheid documentatie gecreëerd over het Docker systeem, maar verlaagt ook de instapdrempel voor nieuwe gebruikers. Tezamen met de officiële documentatie die Docker zelf voorziet, kan men vorm geven aan een reeks van best practices om optimaal gebruik te maken van Docker. In dit hoofdstuk wordt er gekeken in hoeverre Microsoft, Docker en hun gebruikers al in staat zijn om de literaire kloof te dichten in vergelijking met Linux.

4.1 Requirements

Er is weinig info te vinden over de vereisten die nodig zijn om Docker met succes te installeren en uit te voeren.

Voor CentOS komt dit neer op drie lijnen tekst die eisen dat men gebruik maakt van een maintained version, dat 'centos-extras' repository enabled is en dat men gebruik maakt van de 'overlay2' storage driver. (Docker, 2018a)

Windows is op dit vlak een klein beetje beter. Het heeft een hele pagina speciaal toegewijd aan de installatie van Docker, waarin één hoofdstuk beschrijft wat er specifiek nodig is om Docker te installeren: Hyper-V enabled, virtualisatie in de BIOS aanzetten, 64bit Windows 10 Pro, Enterprise en Education of Windows Server 2016 en een beschrijving van hoe de container zou werken met andere gebruikers en parallel instances. (Docker, 2018b)

4.2 Installatie

Docker installeren is ook heel verschillend op beide systemen, wat op zich niet zo verbazend is gezien het verleden van beide apparaten.

Voor CentOS focust men puur op de command-line interface. Dit is ook logisch, aangezien er bij de basisinstallatie van CentOS geen GUI inbegrepen is. Daarom gaat Docker er meteen van uit dat men gebruik zal maken van Bash, wat een CLI is. De gids die Docker hiervoor aanbiedt is ook vrij compleet. Men vertrekt zelfs vanuit het idee dat de gebruiker al een oudere versie kan hebben staan, met als gevolg dat het ook Bash-commando's installeert die eerst de oude versie verwijderen en vervolgens vanaf nul alle nodige stappen ondernemen. (Mays, 2015)

Ook hierin verschilt Windows sterk. Docker gaat er bij het Windows-platform van uit dat men gebruik wil maken van de GUI en biedt bijkomend de mogelijkheid om de installatie via PowerShell te laten gebeuren, wat ook een CLI is. Normaal gesproken zou het een bonus moeten zijn dat er twee verschillende benaderingsmethodes zijn, maar in dit geval is het eerder een nadeel omdat de focus op de GUI ligt. Dit is een minpunt omdat de meeste gebruikers zich liever focussen op automatisatie, wat bij de GUI geen optie is. Verder is dit ook een rare keuze, aangezien PowerShell een moderne en flexibele taal is omdat het met objecten werkt. Dit zou het beheren van de Docker-applicatie moeten vergemakkelijken, wat niet het geval is. (Rickard, 2018)

4.3 Container tot stand brengen

De installatie van een container vereist bij de beide platformen grotendeels dezelfde logica, namelijk via Docker-commando's. Het enige verschil is het aanmaken, invullen en uitlezen van de Dockerfile.

Zoals eerder aangehaald maakt CentOS gebruik van Bash om Docker te installeren. Deze trend zet zich ook voort bij het aanmaken van de Dockerfile. De in- en uitvoer van Bash is tekstgebaseerd, wat uitstekend past bij de vereiste voor het succesvol uitvoeren van een Dockerfile, namelijk Docker-commando's. (Cezar, 2016)

Voor Windows maakt Docker gebruik van PowerShell om de Dockerfile aan te maken. Zodoende kan men in de CLI-omgeving blijven en daarin verder werken nadat het bestand is aangemaakt. Zoals eerder aangehaald maakt PowerShell gebruik van objecten, C# om precies te zijn. Om een tekstbestand aan te maken, zoals een Dockerfile, is dit geen probleem, maar het is wel spijtig dat Docker hier niet beter gebruik van maakt. (Rickard, 2018)

4.4 Automatisatie

Rekening houdend met de doelgroep van deze bachelorproef is automatisatie zeker geen punt dat genegeerd mag worden. Automatisatie is immers één van de pijlers van DevOps.

CentOS heeft hierbij een streepje voor. Aangezien Docker meteen vertrekt vanuit Bash, dient men alleen deze commando's nog in een script te steken met de extensie `.sh` of `.bash` en dit vervolgens aan te roepen. (Mays, 2015)

Ook hier is het wat moeilijker voor Windows. Een groot struikelblok is namelijk dat de nadruk minder op automatisatie ligt en meer op GUI, waardoor automatiseren moeilijker gaat. Echter, het grootste struikelblok is dat het Windows-platform een heropstart vereist na de installatie van Docker. Via plug-ins en scripting kan men hier nog rond werken, maar het maakt de automatisatie wel wat moeilijker. (Rickard, 2018)

4.5 Conclusie

De voornaamste informatiebron, zowel voor Docker for Windows als voor CentOS, is de documentatie op de officiële websites van Docker, RedHat en Microsoft. Docker en Microsoft doen hun best om de literaire kloof te sluiten, in tegenstelling tot CentOS. Zowel Docker als Microsoft proberen hierbij gretig gebruik te maken van hun communities om artikels te publiceren die door gebruikers worden geschreven of aangepast. Op deze manier hopen ze om het platform te promoten, zodat toekomstige gebruikers in staat zijn om hun eigen Docker-omgeving op te starten.

Ondanks deze inhaalbeweging gaat de meeste documentatie er nog steeds van uit dat men Docker wil draaien op Linux. Aangezien Docker voor Linux al beschikbaar is sinds 13 maart 2013 en pas recent voor Windows (22 februari 2017) is dit ook geen verrassing.

Het voordeel hiervan is wel dat de documentatie voor Docker for Windows vaker up-to-date is, omdat deze pas recent geschreven is en dan nog vaak door mensen die doorheen de jaren meegegroeid zijn met de technologie. Hoewel Container-technologie al lang bestaat, is het gebruik ervan tot recent vrij onbestaand geweest. Hierdoor heeft Docker for Windows de eerste moeilijke jaren van het uitzoeken naar een gepaste werkwijze vermeden.

Ten slotte is er wel één groot voordeel betreffende de documentatie voor Docker for Linux, namelijk troubleshooting. Doordat er al meer mee geëxperimenteerd is, zijn er al oplossingen gevonden voor verschillende problemen. Dit is van onschatbare waarde voor de CentOS-opstelling en is tegelijkertijd een hekelpunt voor de Windows-opstelling.

5. Opstelling

De set-up van beide machines wordt geautomatiseerd door middel van Hyper-V, Vagrant en scripts. Hiermee kan men garanderen dat, telkens er gewerkt wordt aan de machines, deze op dezelfde manier worden geïnstalleerd en aangepast. Een uitdaging hierbij is het vinden van gepaste Vagrant Boxes voor beide platformen, want hoewel CentOS wel een officiële Vagrant Image voorziet op Vagrant Cloud, wat de opslagplek is van alle publieke images, doet Microsoft dit niet. Hierdoor wordt men gedwongen om gebruik te maken van onofficiële bronnen. Ten slotte is het belangrijk dat de middelen en de werklading van beide machines evenredig zijn waar mogelijk. Natuurlijk zijn er subtiele verschillen, maar deze worden maximaal weggewerkt indien mogelijk.

5.1 Windows Server 2016

5.1.1 Vagrant

Voor de Windows-opstelling wordt er gekozen om de 'w16s'-image te gebruiken van de gebruiker 'gusztavvargadr'. Deze Vagrant Image bevat een Windows Server 2016 Standard 1607 (14393.2155). Verder bevat deze image ook al Chocolatey, wat maakt dat er alleen nog maar gekeken moet worden of Chocolatey up to date is.

Chocolatey is een community-driven packet manager voor Windows, zoals APT er bijvoorbeeld een is voor Debian. Met een packet manager kan men het installeren en beheren van applicaties automatiseren. Hiermee kan men de installatie dus vereenvoudigen.

Vervolgens wordt er ingesteld dat deze Vagrant Box 4GB RAM en 2 cores krijgt bij het uitrollen, zodat de installatie van Docker en uiteindelijk van de containers vlot en snel kan

Figuur 5.1: Het eerste deel van de configuratie van het Vagrantfile-bestand voor de Windows-opstelling

```

1  # -*- mode: ruby -*-
2
3  # vi: set ft=ruby :
4
5  VAGRANTFILE_API_VERSION = "2"
6
7  ENV['VAGRANT_DEFAULT_PROVIDER'] = 'virtualbox'
8
9  Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
10
11     config.vm.network "private_network", ip: "192.168.56.16"
12
13     # full version WS2016
14     config.vm.box = "gusztavvargadr/w16s"
15
16     # alternative:
17     # config.vm.box = "mwrock/Windows2016"
18
19     config.vm.network "public_network", bridge: "Intel(R) Dual Band Wireless-AC 3160"
20
21     # configuration for virtualbox
22     config.vm.provider "virtualbox" do |v|
23         v.name = "DockerForWinVM"
24         v.memory = 4096
25         v.cpus = 2
26     end
27
28     # configuration for hyperv
29     config.vm.provider "hyperv" do |v|
30         v.vmname = "DockerForWinVM"
31         v.memory = 4096
32         v.cpus = 2
33     end
34
35     config.vm.boot_timeout = 420
36
37     config.vm.hostname = "DockerForWinVM"
38
39     config.vm.provision :shell, path: "Provisioning/Chocolatey.ps1"
40
41     config.vm.provision :shell, path: "Provisioning/Docker-P1.ps1"
42

```

verlopen. Daarnaast stelt men ook een naam in voor de VM, evenals een host-naam. Dit maakt het makkelijker om de VM later aan te spreken.

Uitzonderlijk voor de Windows Server wordt er ook gebruik gemaakt van een plug-in voor Vagrant, namelijk, de 'Vagrant Reload'-plug-in van Aidan Nagorcka-Smith. Hiermee kan men het 'Vagrant Reload'-commando uitvoeren tijdens het uitrollen van de Vagrant Image naar een Box. Dit is een belangrijke voorwaarde, want de Windows Server dient heropgestart te worden na de installatie van Docker.

Daarnaast dient er ook gebruik gemaakt te worden van een klein .BAT-script om docker-compose te installeren via pip.

Ten slotte wordt er ook gebruik gemaakt van een reeks provision-commando's om het systeem te voorzien van de nodige scripts en een voorbeeld-project. Het voorbeeld-project bevat een .NET-webapplicatie, die de onderstaande startpagina zou moeten tonen:

Figuur 5.2: Het tweede deel van de configuratie van het Vagrantfile-bestand voor de Windows-opstelling

```
42
43     # reload plugin for restart during provision
44     # download from aidanns/vagrant-reload with 'vagrant plugin install vagrant-reload'
45     config.vm.provision :reload
46
47     config.vm.provision :shell, path: "Provisioning/Docker-P2.ps1"
48
49     config.vm.provision :shell, path: "Provisioning/docker-compose.bat"
50
51     # config.vm.provision :shell, path: "Provisioning/Restart-Network.ps1"
52
53     # Verschil in connectiestring = User ID != User
54     config.vm.provision "file", source: "../VoorbeeldProjectWindows", destination: "~/VoorbeeldProject"
55
56     # config.vm.provision "file", source: "D:/Documenten/School/HoGent/StageEnBachelorproef/bachelorproef,
57
58     config.vm.provision :shell, path: "Provisioning/images.ps1"
59
60     config.vm.provision :shell, path: "Provisioning/applicaties.ps1"
61
62 end
63
```

Figuur 5.3: De index-pagina van het voorbeeldproject wanneer de containers opgezet zijn.

DienstenCheques


Home




About

Contact

Register

Log in

 Visual Studio



<

>

There are powerful new features in Visual Studio for building modern web apps. [Learn More](#)

○ ● ○ ○ ○

Application uses

- Sample pages using ASP.NET Core MVC
- Bower for managing client-side libraries
- Theming using [Bootstrap](#)

How to

- [Add a Controller and View](#)
- [Manage User Secrets using Secret Manager](#).
- [Use logging to log a message](#).
- [Add packages using NuGet](#).
- [Add client packages using Bower](#).
- [Target development, staging or production environment](#).

Overview

- [Conceptual overview of what is ASP.NET Core](#)
- [Fundamentals of ASP.NET Core](#) such as Startup and middleware.
- [Working with Data](#)
- [Security](#)
- [Client side development](#)
- [Develop on different platforms](#)
- [Read more on the documentation site](#)

Run & Deploy

- [Run your app](#)
- [Run tools such as EF migrations and more](#)
- [Publish to Microsoft Azure Web Apps](#)

© 2017 - DienstenCheques

Figuur 5.4: De inhoud van het Chocolatey script

```
# Controlen en aanpassen Execution-Policy Windows PowerShell naar BYPASS
$CurrentPolicy = get-ExecutionPolicy

Write-Host "#CURRENT POLICY IS $CurrentPolicy#"

if($CurrentPolicy -ne "Bypass")
{
    Write-Host "SETTING CURRENT POLICY TO Bypass"
    Set-ExecutionPolicy Bypass
}

# Installeren Chocolatey
Write-Host "#INSTALLING CHOCOLATEY#"
iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))
choco upgrade chocolatey
Write-Host "#CHOCOLATEY INSTALLED#"

# Chocolatey globale uitvoerrechten geven
Write-Host "#SET CHOCOLATEY EXECUTIONRIGHTS#"
chocolatey feature enable -n=allowGlobalConfirmation
Write-Host "#EXECUTIONRIGHTS SET#"
```

Figuur 5.5: De inhoud van het eerste Docker script. Deze installeert Docker.

```
# Controlen en aanpassen Execution-Policy Windows PowerShell naar BYPASS
$CurrentPolicy = get-ExecutionPolicy

Write-Host "#CURRENT POLICY IS $CurrentPolicy #"

if($CurrentPolicy -ne "Bypass")
{
    Write-Host "SETTING CURRENT POLICY TO Bypass"
    Set-ExecutionPolicy Bypass
}

# Installeren Docker
Write-Host "#INSTALLING DOCKER#"
Install-Module -Name DockerMsftProvider -Repository PSGallery -Force
Install-Package -Name docker -ProviderName DockerMsftProvider -Force
Write-Host "#DOCKER INSTALLED#"

Write-Host "#REBOOTING#"
```

5.1.2 PowerShell

De 6 scripts die voorzien worden, configureren elk een component dat nodig is om de omgeving af te werken.

- Chocolatey

In dit script wordt er gekeken of Chocolatey al geïnstalleerd is. Zoja, wordt er gecontroleerd of Chocolatey reeds up to date is, en anders installeert het de packet manager.

- Docker #1

Hierna wordt het eerste deel van de Docker installatiescripts uitgevoerd. Dit installeert Docker en geeft daarna een melding dat er een reboot aan komt.

Nadat Docker geïnstalleerd is, vereist het systeem dat het opnieuw wordt opgestart. Deze plug-in is de makkelijkst gevonden manier, zonder in te grijpen in het automatisatie-proces.

- Docker #2

Vervolgens wordt Posh-Docker geïnstalleerd. Deze tool voorziet de installatie van Python en Pip door gebruik te maken van Chocolatey. Daarnaast wordt er in dit script ook getest

Figuur 5.6: De inhoud van het tweede Docker script. Deze installeert Python en Pip, creëert een test-container en voegt de Firewall-regels toe

```
net start com.docker.service

# Installeren Pip
Write-Host "#INSTALLING PYTHON/PIP#"
choco install -y python2
setx PATH "$env:path;C:\tools\python2;C:\tools\python2\Scripts" -m
$env:path += ";C:\tools\python2;C:\tools\python2\Scripts"
choco install pip

# Testen installatie
Write-Host "#RUN TEST CONTAINER#"
docker container run hello-world:nanoserver

# Toevoegen poorten voor Docker
Write-Host "#ADDING FIREWALL RULES#"
netsh advfirewall firewall add rule name=mssqlin dir=in action=allow protocol=TCP localport=1433
netsh advfirewall firewall add rule name=mssqlout dir=out action=allow protocol=TCP localport=1433

netsh advfirewall firewall add rule name=dockerappin dir=in action=allow protocol=TCP localport=8080
netsh advfirewall firewall add rule name=dockerappout dir=out action=allow protocol=TCP localport=8080

netsh advfirewall firewall add rule name=httpin dir=in action=allow protocol=TCP localport=80
netsh advfirewall firewall add rule name=httpout dir=out action=allow protocol=TCP localport=80

netsh advfirewall firewall add rule name="docker engine" dir=in action=allow protocol=TCP localport=2375
```

Figuur 5.7: Pip wordt aangeroepen om docker-compose te installeren.

```
echo '#INSTALLING DOCKER-COMPOSE#' |
pip install docker-compose
```

of Docker correct is geïnstalleerd en worden de firewall-rules ingesteld.

- docker-compose.bat

Hierna wordt het docker-compose.bat-script uitgevoerd. Dit script maakt gebruik van Pip om docker-compose te installeren.

- Images

Vervolgens wordt het Images.ps1-script uitgevoerd. Dit script is verantwoordelijk voor het installeren van de container waarin de databank wordt gehost die de applicatie nodig heeft. Dit gebeurt aan de hand van een Microsoft SQL server Docker Image die van het internet wordt gehaald en waar vervolgens een container mee wordt gebouwd.

- Application

Ten slotte haalt het laatste script die specifieke .NET-sdk Docker Image op die nodig is om de container te bouwen die de webapplicatie vereist. Daarna maakt het script de Dockerfile aan en vult deze op met de vereiste tekst om de container te bouwen, alsook de commando's om daarna de Docker Image en Container te bouwen.

Figuur 5.8: Hiermee worden de Docker Images opgehaald van de Cloud

```
Write-Host "#DOWNLOADING IMAGES#"
# download mssql image
docker pull microsoft/mssql-server-windows-developer

# download dotnet image
docker pull microsoft/dotnet:2.0.5-sdk-2.1.4
Write-Host "#IMAGES DOWNLOADED#"
```

Figuur 5.9: Het creëren van het Dockerfile-bestand en docker-compose-bestand. Alsook het opvullen van het Dockerfile-bestand.

```
# Naar directory gaan waarin het .csproj-bestand zit
sl ../../Users/vagrant/VoorbeeldProject/DienstenCheques

# Aanmaken app-folder, dockerfile en docker-compose.yml
Write-Host "#CREATING APP, DOCKERFILE EN DOCKER-COMPOSE.YML#"
New-Item "dockerfile" -Force
New-Item "docker-compose.yml" -Force
New-Item "app" -ItemType Directory -Force

# Opvullen dockerfile
Write-Host "#FILLING DOCKERFILE#"
$DockerCommands = 'FROM microsoft/dotnet:2.0.5-sdk-2.1.4'
$DockerCommands | Add-Content "Dockerfile"
$DockerCommands = 'WORKDIR /app'
$DockerCommands | Add-Content "Dockerfile"
$DockerCommands = 'COPY . /app'
$DockerCommands | Add-Content "Dockerfile"
$DockerCommands = 'RUN ["dotnet", "restore"]'
$DockerCommands | Add-Content "Dockerfile"
$DockerCommands = 'RUN ["dotnet", "build"]'
$DockerCommands | Add-Content "Dockerfile"
$DockerCommands = 'EXPOSE 5000/tcp'
$DockerCommands | Add-Content "Dockerfile"
$DockerCommands = 'ENV ASPNETCORE_URLS http://*:5000'
$DockerCommands | Add-Content "Dockerfile"
$DockerCommands = 'ENTRYPOINT ["dotnet", "run"]'
$DockerCommands | Add-Content "Dockerfile"
```

Figuur 5.10: Het opvullen van het docker-compose-bestand. Alsook het aanroepen van 'docker-compose build' en 'docker-compose up -d'.

```
# Opvullen docker-compose.yml
# indentatie belangrijk!
Write-Host "#FILLING DOCKER-COMPOSE.YML#"
$ComposeCommand = 'version: "3"'
$ComposeCommand | Add-Content "docker-compose.yml"
$ComposeCommand = 'services:'
$ComposeCommand | Add-Content "docker-compose.yml"
$ComposeCommand = '  web:'
$ComposeCommand | Add-Content "docker-compose.yml"
$ComposeCommand = '    build: .'
$ComposeCommand | Add-Content "docker-compose.yml"
$ComposeCommand = '    ports:'
$ComposeCommand | Add-Content "docker-compose.yml"
$ComposeCommand = '      - "8080:5000"'
$ComposeCommand | Add-Content "docker-compose.yml"
$ComposeCommand = '    depends_on:'
$ComposeCommand | Add-Content "docker-compose.yml"
$ComposeCommand = '      - db'
$ComposeCommand | Add-Content "docker-compose.yml"
$ComposeCommand = '    db:'
$ComposeCommand | Add-Content "docker-compose.yml"
$ComposeCommand = '      image: "microsoft/mssql-server-windows-developer"'
$ComposeCommand | Add-Content "docker-compose.yml"
$ComposeCommand = '      environment:'
$ComposeCommand | Add-Content "docker-compose.yml"
$ComposeCommand = '        SA_PASSWORD: "Vagrant123"'
$ComposeCommand | Add-Content "docker-compose.yml"
$ComposeCommand = '        ACCEPT_EULA: "Y"'
$ComposeCommand | Add-Content "docker-compose.yml"
$ComposeCommand = 'networks:'
$ComposeCommand | Add-Content "docker-compose.yml"
$ComposeCommand = '  default:'
$ComposeCommand | Add-Content "docker-compose.yml"
$ComposeCommand = '    external:'
$ComposeCommand | Add-Content "docker-compose.yml"
$ComposeCommand = '      name: nat'
$ComposeCommand | Add-Content "docker-compose.yml"

# Bouwt de image
Write-Host "#BUILDING IMAGE#"
docker-compose build

# Bouwt de container
Write-Host "#BUILDING CONTAINER#"
docker-compose up -d
```

Figuur 5.11: Het eerste deel van de configuratie van het Vagrantfile-bestand voor de CentOS-opstelling.

```

1  # -*- mode: ruby -*-
2
3  # vi: set ft=ruby :
4
5  VAGRANTFILE_API_VERSION = "2"
6  # Staat standaard op virtualbox maar kan op hyperv gezet worden
7  # ALS JE HYPERV GEBRUIKT MOET JE config.vm.network IN COMMENTAAR ZETTEN
8  ENV['VAGRANT_DEFAULT_PROVIDER'] = 'virtualbox'
9
10 ▾ Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
11
12     # IP Adres toevoegen aan virtuele machine
13     # IN COMMENTAAR ZETTEN ALS JE HYPERV GEBRUIKT, NOG NIET ONDERSTEUND DOOR VAGRANT BIJ HYPERV
14     config.vm.network "private_network", ip: "192.168.56.15"
15
16     config.vm.box = "centos/7"
17

```

Figuur 5.12: Het tweede deel van de configuratie van het Vagrantfile-bestand voor de CentOS-opstelling.

```

20 ▾ config.vm.provider "virtualbox" do |v|
21     v.name = "CentOSDocker"
22     v.memory = 4096
23     v.cpus = 2
24 end
25
26 # configuration for hyperv
27 ▾ config.vm.provider "hyperv" do |v|
28     v.vminame = "DockerForWinVM"
29     v.memory = 4096
30     v.cpus = 2
31 end
32
33 config.vm.hostname = "CentOSDocker"
34
35 config.vm.provision :shell, path: "scripts/prereq.sh"
36
37 config.vm.provision :shell, path: "scripts/images.sh"
38
39 config.vm.provision "file", source: "D:/Documenten/School/HoGent/StageEnBachelorproef/bachelorproef/BachelorproefDocker/CentOSDocker/VoorbeeldProject/centos7.4.iso"
40
41 # config.vm.provision "file", source: "D:/Documenten/School/HoGent/StageEnBachelorproef/bachelorproef/BachelorproefDocker/CentOSDocker/09solSportsScripts"
42
43 config.vm.provision :shell, path: "scripts/applicaties.sh"
44 end
45

```

5.2 CentOS 7.4

5.2.1 Vagrant

Om de CentOS-server te installeren wordt er gebruik gemaakt van de officiële Vagrant Image die beschikbaar wordt gesteld door de organisatie achter CentOS, namelijk Red Hat Enterprise Linux (RHEL). Deze voorziet alleen een basisinstallatie van een CentOS 7.4 server, maar voorziet wel de mogelijkheid om uitgerold te worden op alle populaire platformen.

Ook hier krijgt het systeem 4GB RAM en 2 cores, zodat er op een eerlijke manier naar de performantie van beide systemen kan worden gekeken. Verder krijgt het systeem ook een hostname en VMName.

Ten slotte worden er 3 scripts voorzien bij het provision-gedeelte, zodat alle benodigde componenten kunnen worden geïnstalleerd. Hieronder behoort onder andere een script om

Figuur 5.13: Het eerste deel van de vereiste functies voor de CentOS-opstelling.

```

1  #!/usr/bin/bash
2
3  # Provisioning script
4
5  # Settings for Bash
6  # abort on nonzero exitstatus
7  set -o errexit
8  # abort on unbound variable
9  set -o nounset
10 # don't mask errors in piped commands
11 set -o pipefail
12
13 # The name of the user that is going to manage the Docker service
14 readonly docker_admin=vagrant
15
16 # Install utilities
17 echo "Installing utilities"
18
19 yum -y install epel-release \
20             git \
21             nano \
22             patch
23
24 yum -y install python-pip
25
26 # Install Docker CE
27 echo "Installing Docker"
28
29 # Docker requires yum-utils and yum-config-manager
30 yum install -y yum-utils \
31             device-mapper-persistent-data \
32             lvm2
33
34 yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
35
36 # Installing Docker CE stable
37 yum install -y docker-ce
38
39 # Optional: Docker CE edge and test repository
40 # yum-config-manager --enable docker-ce-edge
41 # yum-config-manager --enable docker-ce-test
42

```

dezelfde webapplicatie te voorzien als voor Windows.

5.2.2 Bash

- prereq

In dit gedeelte van het script worden alle functionaliteiten voorzien die nodig zijn voordat men kan beginnen met het uitrollen van de applicatie, waaronder de installatie van Docker, docker-compose, Epel-Release, Git, Nano en anderen. Daarnaast worden ook regels toegevoegd aan de firewall, zodat de container beschikbaar is nadat deze volledig is opgesteld.

- images

Bij het volgende script worden alle Docker Images van Docker Hub gehaald, wat de officiële online repository van Docker is waar alle beschikbare Docker Images op staan.

- applicaties

Ten slotte worden in het laatste script de benodigde databank en applicatie omhoog gebracht aan de hand van een dockerfile, docker run- en docker build-commando's. De dockerfile- en dockercompose.yml-bestand worden opgevuld met een 'EOF' identifier. Hierdoor blijft Bash de invoer pipen in de dockerfile tot hij 'EOF' tegenkomt, waarna hij stopt zonder 'EOF' mee te nemen.

Figuur 5.14: Het tweede deel van de vereiste functies voor de CentOS-opstelling.

```
43 # Set Dockeradmin
44 echo "Adding user vagrant to group docker"
45 usermod -aG docker ${docker_admin}
46
47 # Service management
48 echo "Enabling services"
49 systemctl start docker.service
50 systemctl start firewalld.service
51
52 echo "Starting services"
53 systemctl enable docker.service
54 systemctl enable firewalld.service
55
56 # Test docker
57 echo "Testing Docker installation"
58
59 docker run hello-world
60
61 # Add ports to firewall
62 echo "Configuring firewall"
63 firewall-cmd --add-port=8080/tcp --permanent
64 firewall-cmd --add-port=80/tcp --permanent
65 firewall-cmd --add-port=5000/tcp --permanent
66 firewall-cmd --add-port=1433/tcp --permanent
67 firewall-cmd --reload
68
69 # Restarting Docker service due to firewall
70 echo "Restarting docker.service"
71 systemctl restart docker.service
72
73 # Install Docker-Compose
74 # sudo curl -L https://github.com/docker/compose/releases/download/1.21.0/docker-compose-
75
76 # chmod +x /usr/local/bin/docker-compose
77 echo "Installing docker-compose"
78 pip install docker-compose
79
80 echo "Testing Docker-compose"
81 docker-compose --version
```

Figuur 5.15: Hier worden de Docker Images opgehaald.

```
1 #!/usr/bin/bash
2 # abort on nonzero exitstatus
3 set -o errexit
4 # abort on unbound variable
5 set -o nounset
6 # don't mask errors in piped commands
7 set -o pipefail
8
9 # Required images
10 echo "Pulling images"
11
12 # MSSQL image
13 docker pull microsoft/mssql-server-linux
14
15 # .NET image
16 docker pull microsoft/dotnet:2.0.5-sdk-2.1.4
```

Figuur 5.16: Hier wordt het Dockerfile- en docker-compose-bestand opgevuld. Waarna 'docker-compose build' en 'docker-compose up -d' wordt uitgevoerd.

```

1  #!/usr/bin/bash
2  # abort on nonzero exitstatus
3  set -o errexit
4  # abort on unbound variable
5  set -o nounset
6  # don't mask errors in piped commands
7  set -o pipefail
8
9  # Going to the directory which contains the .csproj-file
10 cd /home/vagrant/VoorbeeldProject/DienstenCheques
11
12 # cd /home/vagrant/09solSportsStore-master/SportsStore
13
14 # Creates dockerfile
15 echo "Creating dockerfile"
16 cat > Dockerfile << EOF
17 FROM microsoft/dotnet:2.0.5-sdk-2.1.4
18
19 RUN mkdir -p /app
20
21 WORKDIR /app
22
23 COPY . /app
24
25 RUN dotnet restore
26
27 RUN dotnet build
28
29 EXPOSE 5000/tcp
30
31 ENV ASPNETCORE_URLS http://*:5000
32
33 ENTRYPOINT ["dotnet", "run"]
34 EOF
35
36 # Creates docker-compose.yml
37 echo "Creating docker-compose.yml"
38 cat > docker-compose.yml << EOF
39 version: "3"
40
41 services:
42   web:
43     build: .
44     ports:
45       - "8080:5000"
46     depends_on:
47       - db
48   db:
49     image: "microsoft/mssql-server-linux"
50     environment:
51       SA_PASSWORD: "Vagrant123"
52       ACCEPT_EULA: "Y"
53 EOF
54
55 # Building image
56 echo "Building docker image"
57 docker-compose build
58
59 # Running container
60 echo "Building container"

```

6. Performantie test

In dit hoofdstuk worden de resultaten besproken van de uitgevoerde performantietests.

De performantie van beide systemen werd op twee manieren getest: Eerst werd de benodigde tijd gemeten voor het uitvoeren van het 'vagrant up'-commando. Dit commando installeerde en configureerde het besturingssysteem, de applicaties en de containers vanaf nul.

Vervolgens werd de benodigde tijd gemeten bij het uitvoeren van het 'vagrant provision'-commando. Hierbij waren het besturingssysteem en de applicaties al geïnstalleerd en geconfigureerd, en moesten alleen de containers nog tot stand gebracht worden. Om de internetsnelheid als factor te schrappen, waren de Docker Images ook al gedownload.

Daarnaast werd 'time' als prefix aan het commando toegevoegd. Hierdoor werd de benodigde uitvoeringstijd op het einde van de commando's getoond.

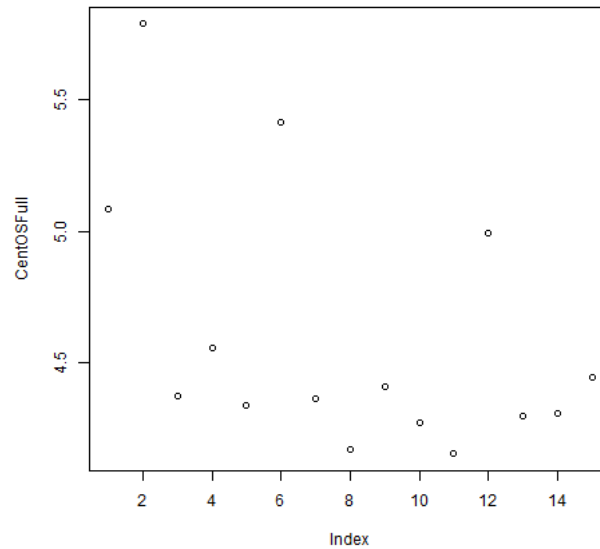
Vervolgens werden beide commando's vijftien keer uitgevoerd op beide opstellingen. Hierdoor werd de invloed van outliers verminderd. Outliers zijn observatiepunten die ver van andere punten verwijderd liggen.

Ten slotte werden deze resultaten genoteerd in een tabel in Rstudio, zodat deze resultaten konden worden gebruikt om het gemiddelde, de variantie en standaarddeviatie te berekenen, alsook voor het genereren van een scatterplot en boxplot.

| Meting 1 | Meting 2 | Meting 3 | Meting 4 | Meting 5 | Meting 6 | Meting 7 | Meting 8 | Meting 9 |
|----------|----------|----------|----------|----------|----------|-----------|-----------|----------|
| 5m0.840s | 5m7.846s | Meting 3 | Meting 4 | Meting 5 | 5m4.108s | 4m36.605s | 4m17.360s | 4m40.9 |

Tabel 6.1: De resultaten van 'time vagrant up' voor de CentOS-opstelling.

Figuur 6.1: De scatterplot voor de resultaten van 'time vagrant up' voor de CentOS-opstelling



6.1 CentOS 7.4

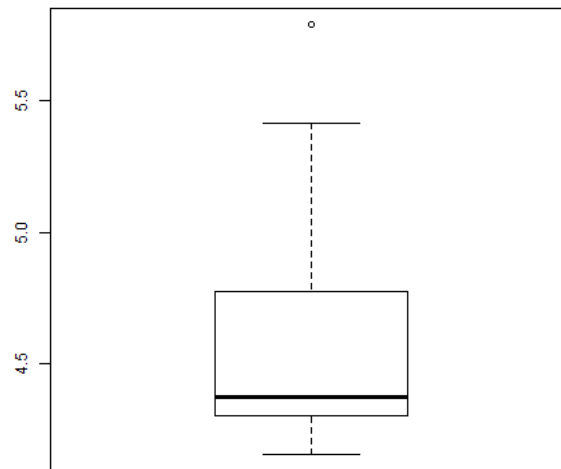
6.1.1 Performantie installatie

Hieronder kan men de eerste resultaten zien van het 'time vagrant up'-commando voor de CentOS-opstelling.

$$\mu = 4.59897 \sigma^2 = 0.2386372 \sigma = 0.488505$$

De gemiddelde benodigde tijd voor het commando bij de CentOS-opstelling was laag en vrijwel stabiel, met slechts enkele outliers. De reden voor deze outliers was omdat het ophalen van een Docker Image gevoelig is aan slechtere internetverbindingen. Dit was de grootste bottleneck voor deze opstellingen. De impact hiervan op de installatie was in dit geval groot, zoals te zien is aan de hoge variatie en de standaarddeviatie. Maar, over het algemeen is de performantie van de CentOS-opstelling wel goed te noemen.

Figuur 6.2: De boxplot voor de resultaten van 'time vagrant up' voor de CentOS-opstelling



| Meting 1 | Meting 2 | Meting 3 | Meting 4 | Meting 5 | Meting 6 | Meting 7 | Meting 8 | Meting 9 | Meting 10 |
|----------|----------|----------|----------|----------|----------|----------|----------|----------|-----------|
| 7.062s | 5.839s | 5.804s | 6.000s | 5.733s | 6.415s | 5.911s | 5.844s | 5.856s | 5.972s |

Tabel 6.2: De resultaten van 'time vagrant provision' voor de CentOS-opstelling.

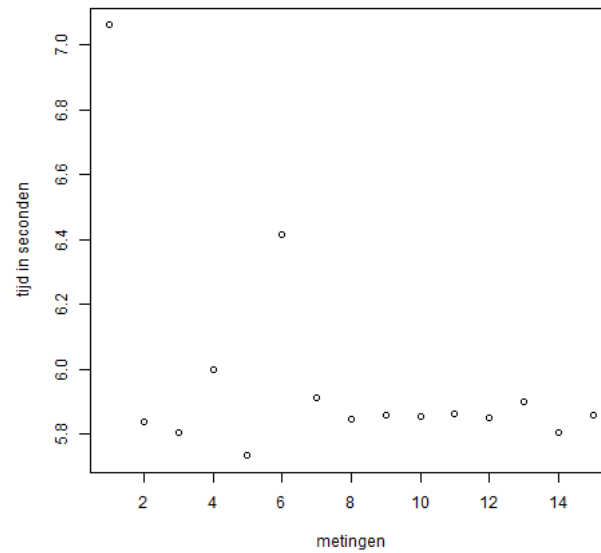
6.1.2 Performantie containers

Hieronder kan men de resultaten zien voor elke keer dat het 'time vagrant provision'-commando werd uitgevoerd op de CentOS-opstelling, maar dan deze keer uitgedrukt in seconden. Bij deze resultaten moest het Operating System niet meer geïnstalleerd worden en ook de Docker Image moest niet meer gedownload worden. Er werd alleen gekeken hoeveel tijd de Docker Daemon nodig had om de orders te interpreteren en de container op te stellen.

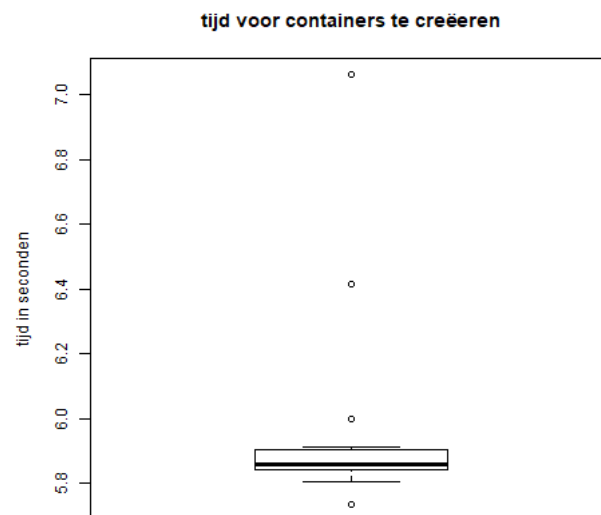
$$\mu = 5.972333 \sigma^2 = 0.1150491 \sigma = 0.3391889$$

De installatie van enkel de container verliep vlot, met een stabielere gemiddelde tijd en een kleinere variantie en standaarddeviatie. Dit kwam doordat de applicaties en Docker Images niet meer moesten worden gedownload. Hierdoor lagen de tijden veel lager en was er geen invloed meer van de bottleneck.

Figuur 6.3: De scatterplot voor de resultaten van 'time vagrant provision' voor de CentOS-opstelling



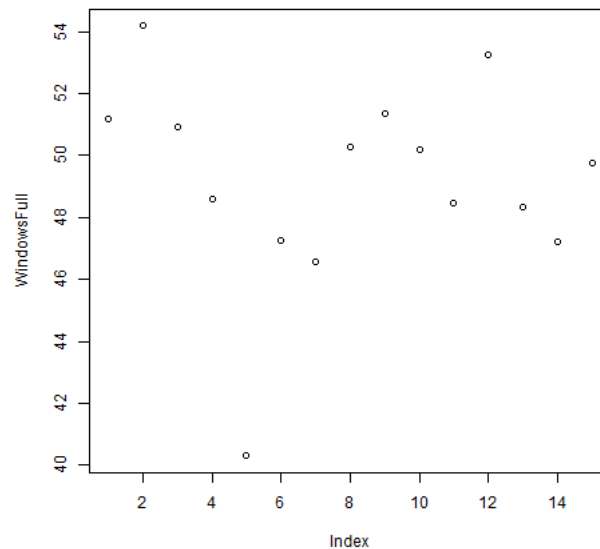
Figuur 6.4: De boxplot voor de resultaten van 'time vagrant provision' voor de CentOS-opstelling



| Meting 1 | Meting 2 | Meting 3 | Meting 4 | Meting 5 | Meting 6 | Meting 7 | Me |
|------------|------------|-----------|------------|------------|------------|------------|-----|
| 51m16.108s | 54m17.881s | 50m9.325s | 48m57.389s | 40m31.108s | 47m27.296s | 46m56.752s | 50m |

Tabel 6.3: De resultaten van 'time vagrant up' voor de Windows-opstelling.

Figuur 6.5: De scatterplot voor de resultaten van 'time vagrant up' voor de Windows-opstelling



6.2 Windows Server 2016

6.2.1 Performantie installatie

Hieronder kan men de resultaten zien van 'time vagrant up' voor de Windows-opstelling.

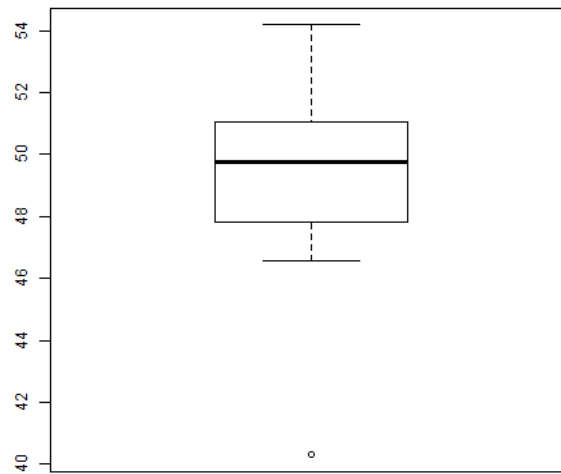
$$\mu = 49.19089\sigma^2 = 10.74354\sigma = 3.277733$$

De installatie van de Windows-opstelling duurde gemiddeld 10 keer langer dan de CentOS-opstelling. De gemiddelde tijd voor een volledige installatie bedroeg immers 45 minuten. De variantie en standaarddeviatie waren wel klein. Hier zijn verschillende redenen voor:

- Installatie Windows OS

Om te beginnen duurde de installatie van het besturingssysteem langer omdat er een GUI gebruikt werd. Als men de Windows Server Core zou gebruiken, zou de performantie al sterk verbeteren, maar dan kon men geen Docker CE for Windows installeren. Windows Server Nano was helaas ook geen optie, omdat deze helemaal niet ondersteund werd door Docker for Windows, zelfs niet voor Docker EE. Ten slotte hoefde CentOS ook niet heropgestart te worden na de installatie van Docker. (xlegalles, 2017)

Figuur 6.6: De boxplot voor de resultaten van 'time vagrant up' voor de Windows-opstelling



| Meting 1 | Meting 2 | Meting 3 | Meting 4 | Meting 5 | Meting 6 | Meting 7 | Meting 8 | Meting 9 | Meting 10 |
|----------|----------|----------|----------|----------|----------|----------|----------|----------|-----------|
| 21.771s | 15.876s | 15.573s | 17.115s | 16.873s | 16.335s | 15.068s | 19.243s | 17.221s | 16.873s |

Tabel 6.4: De resultaten van 'time vagrant provision' voor de Windows-opstelling.

- Microsoft SQL server container

De MSSQL server container voor Windows was meer dan 10 keer zo groot als de Linux variant, meer specifiek: 6GBs tegenover 479MBs. Hierdoor was de installatie een groot deel van de tijd bezig met het downloaden van deze Docker Image. (Moon, 2017)

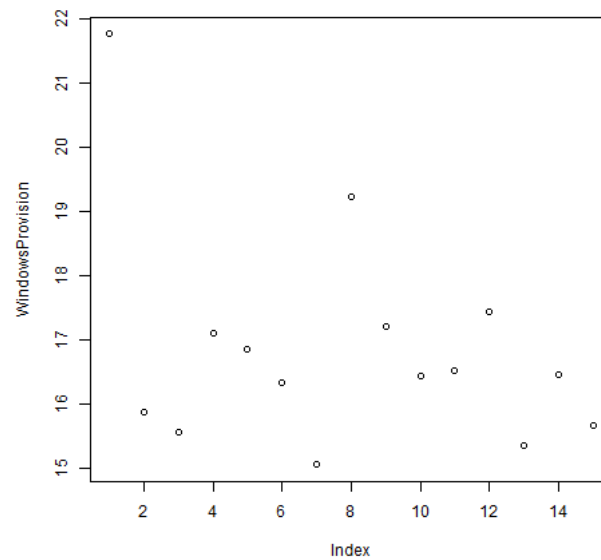
6.2.2 Performantie applicatie

Hieronder kan men de resultaten zien van 'time vagrant provision' voor de Windows-opstelling van alleen de benodigde tijd om de containers te installeren.

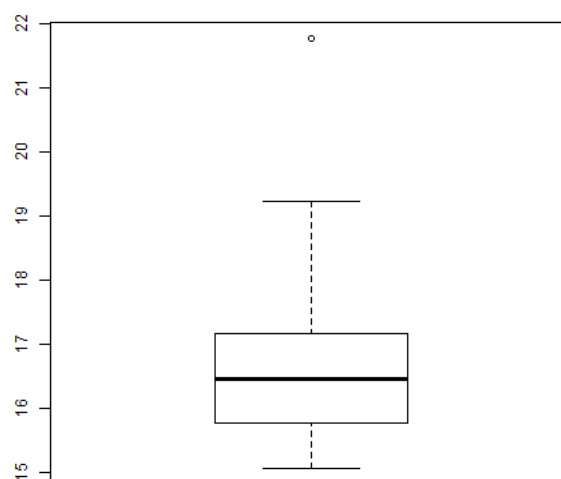
$$\mu = 16.86767\sigma^2 = 1.70055\sigma = 2.89187$$

Hier waren de resultaten al iets meer gelijkaardig aan de CentOS-opstelling, doordat de Windows-opstelling nu geen tijd meer verloor aan de installatie van een GUI, aan heropstarten of aan het downloaden van grotere Docker Images. De voornaamste reden waarom de uitvoeringstijd voor de containers tot drie keer groter was, kwam door de grote van de MSSQL server image. Daarnaast zijn Hyper-V Containers inherent ook groter.

Figuur 6.7: De scatterplot voor de resultaten van 'time vagrant provision' voor de Windows-opstelling



Figuur 6.8: De boxplot voor de resultaten van 'time vagrant provision' voor de Windows-opstelling



6.3 Conclusie

Op elk vlak scoort CentOS beter voor deze test dan Windows. Zoals opgesomd waren hier verschillende redenen voor. Maar, de meeste van deze redenen kunnen en zullen weggewerkt worden door Microsoft en Docker.

7. Security test

Dit hoofdstuk vergelijkt de verschillende manieren waarop beide besturingssystemen, Linux en Windows, in staat zijn om hun systemen te beschermen. Deze security review vergelijkt welke industriestandaarden beide systemen kunnen implementeren om hun systemen te harden tegen aanvallen.

Zoals eerder gezegd is een container automatisch veiliger dan een klassieke VM, omdat er een isolatielaag ligt tussen de container en het besturingssysteem waarop het draait. Echter, als men toch via de container toegang zou kunnen verkrijgen tot het host-systeem, wat dan? Elk systeem is namelijk even sterk als zijn zwakste schakel. Dit werd ook bevestigd voor Solomon Hykes, CTO van Docker op DockerCon 2017 in Austin, USA op vlak van Linux beveiliging. Hij zei: "We denken dat Docker niet de verantwoordelijkheid is om Linux subsystemen te beveiligen. Sterker nog, we denken niet dat deze verantwoordelijkheid bij één bedrijf in het bijzonder moet liggen. Linux is té groot en té belangrijk, dus daarom is het zó belangrijk dat veiligheid vanuit de community komt. En de community is goed bezig: iedereen werkt samen, in plaats van elk in zijn eigen hoekje".

Daarom zal deze security review vertrekken vanuit de populairste veiligheidsopties voor het harden van een CentOS machine, waarna deze vergeleken zullen worden met equivalente opties voor Windows.

7.1 CentOS

7.1.1 Linux containers

Als eerste werd er gekeken naar hoe Linux-containers tot staande gebracht worden en hoe sterk het isolatieniveau ervan is.

Zoals in de inleiding reeds vermeld hebben Linux-containers hun eigen netwerk resources, geheugen, geïsoleerde CPU en I/O block, maar maken ze wel gebruik van dezelfde kernel als het host besturingssysteem. Dit zorgt ervoor dat containers licht en gestroomlijnd kunnen zijn, maar dit kan zorgen voor zowel sterktes als zwaktes.

Om te beginnen is het veel moeilijker om diepe toegang te krijgen tot de hoogst mogelijke privileges bij een container. Dit is een direct gevolg van de isolatie tussen de container en zijn host. Echter, als dit wel lukt, is er een reëel probleem. De Linux Container deelt immers de kernel met zijn host-systeem, waardoor system calls een groot probleem vormen. Door een geminimaliseerde OS te gebruiken kan men dit probleem verminderen, maar niet verhelpen.

Daarnaast is er wel het voordeel dat Root-toegang tot de container niet automatisch betekent dat men Root-toegang heeft op het host-systeem, maar dan moet men wel oppassen voor privileged containers. Deze worden namelijk wel standaard gebonden aan het host uid 0, waardoor deze containers niet root-safe zijn en ook niet root-safe gemaakt kunnen worden. Daarom maakt men best gebruik van Mandatory Access Control, zoals SELinux of AppArmor, seccomp filters en het laten vallen van namespaces.

Vervolgens is er het probleem dat Cgroups standaard geen limiet heeft, waardoor men gemakkelijk een Denial of Service-attack kan uitvoeren op een CentOS-systeem. Dit risico kan verlaagd worden door memory, cpu en pods aan te passen in de configuratie van lxc.cgroup.

Ten slotte zijn aanvallers vaak teleurgesteld als ze in een container geraken, want best practice zegt "geen data in containers", waardoor de voornaamste aanvalreden niet veel zal opleveren. Kijk maar naar de ransomware attacks die steeds populairder worden. Deze hebben als doel om data 'gevangen' te nemen tot het losgeld ervoor betaald is. (Weissig2013) (Reno, 2016) (Wang, 2017)

7.1.2 SELinux

SELinux of Security-Enhanced Linux gebruikt Linux Security Modules (LSM) om een Linux-systeem te harden tegen aanvallen. De verschillende modules kunnen worden aan- of uitgezet via een reeks van booleans. Daarnaast kan men ook handgemaakte waardes toevoegen aan deze lijst om ze te blokkeren of door te laten.

SELinux bestaat uit onderwerpen, zoals users of applicatieprocessen. Daarnaast bestaat het uit objecten, zoals bestanden en folders, en uit een set van regels. Deze regels bepalen

wat een onderwerp mag doen met een object.

Voor Docker bestaat er ook een SEmodule. Deze module moet eerst enabled worden via 'semodule -v -e docker'. Daarna dient Docker te worden herstart met de volgende waarde: `--selinux-enabled`. Deze waarde dient te worden geplaatst in `docker.service` configuratiebestand. Ten slotte dient men Docker opnieuw op te starten, waarna Docker altijd met SELinux enabled zal opstarten.

De invloed van deze module is veelvoudig. Het belangrijkste gevolg is dat men kan beperken tot welke folders Docker toegang heeft, want de 'privileged' Docker-processen krijgen niet dezelfde privileges als andere privileged processen. Daarnaast kan men via het 'docker run'-commando de kernel-mogelijkheden beperken door er '`--cap-drop=`' aan toe te voegen.

Van alle tools die beschikbaar zijn om een Linux-server te beveiligen, is SELinux veruit de krachtigste. Het grootste nadeel bij SELinux is echter dat de gebruiksvriendelijkheid niet evident is; het is een moeilijke tool. Ondanks dit nadeel loont het wel om vaardig te worden binnen deze module. (Henry-Stocker, 2012) (Juggery, 2017) (Walsh, 2013)

7.1.3 Firewalld

Daarnaast kan men op CentOS Firewalld configureren om bepaalde poorten te openen of te blokkeren. In essentie gebruikt Firewalld gewoon IpTables met een XML-overlay, wat het gebruik ervan veel vergemakkelijkt heeft. Hiermee kan men de verschillende zones en poorten instellen die men wil blokkeren of doorlaten, maar men kan ook gekende services toevoegen aan Firewalld, waarbij deze automatisch de standaardpoorten ervoor opent of sluit. Standaard is de zone 'public' en worden de meeste poorten geblokkeerd. (K. Brown, 2014)

7.1.4 Seccomp

Ten slotte kan men ook Secure Computing gebruiken (Seccomp) gebruiken. Hiermee kan men de system calls beperken die een process kan maken. Hierdoor kan het systeem beschermd worden tegen hackers, wanneer ze system calls willen maken die niet eerder gedeclareerd zijn. Hoewel Seccomp in het verleden stroef was om te gebruiken, is het gebruik (Libseccomp) ervan sterk gegroeid, onder andere door de toevoeging van BPF (Berkeley Packet Filters). BPF werd in het verleden gebruikt om netwerk pakketten te filteren, maar door het potentieel ervan groeiden de toepassingen ervoor ook.

Seccomp's voordelen liggen vooral in het blokkeren van system calls vanuit de containers naar het host-systeem, als deze niet van toepassingen zijn. Daardoor kunnen aanvallen vanuit de container end-points tegengehouden worden. (Arora, 2012) (Spijkers, 2013)

7.2 Windows

7.3 Hyper-V containers

Hyper-V Containers, zoals die gebruikt worden door Docker for Windows, verschillen op één belangrijke manier van Linux Containers. Hyper-V containers gebruiken namelijk de base image om een VM aan te maken, door deze in de Hyper-V partition wrapper te steken, waarna er in deze VM een Windows-container wordt aangemaakt om de applicatie in te steken. Hierdoor is er een extra isolatielaag tussen de container en de kernel, namelijk de Hypervisor van de VM. Dit verhelpt veel van de problemen die de Linux Containers hebben op vlak van kwetsbaarheden.

Hier staat echter wel een kost tegenover. Dit zorgt er namelijk voor dat de Hyper-V containers in het algemeen groter zullen zijn dan Linux Containers, en dat ze meer tijd zullen nodig hebben om opgezet te worden. De last is gelukkig niet al te groot, aangezien Hyper-V een type 1 Hypervisor is en dus rechtstreeks op de hardware draait. Ten slotte bestaat er ook het gevaar dat de VM crasht wanneer men in de container zit, waardoor men niet meer uit de container boundaries zal geraken. (Savill, 2015)

7.4 Integrity levels

Windows Integrity Checks (WIC) is het Windows-equivalent van SELinux. Deze functie zat reeds ingebouwd in Windows Vista, waar het als doel had om objecten te beschermen, zoals bestanden, printers en pipes. WIC werkt door de betrouwbaarheid van alle objecten en de interactie hiermee een waarde te geven, zogezegd een level of trustworthiness. Als men een actie wil uitvoeren op dit object, zal men het niveau op zijn minst moeten evenaren. WIC heeft een grotere prioriteit dan normale permissies.

Er zijn in totaal zes niveaus die WIC kan geven aan objecten. Van laag naar hoog zijn dit:

- untrusted (anonieme processen)
- low (interactie met het internet)
- medium (meeste objecten, onder andere de shell)
- high (Administrators)
- system (system)
- installer (speciaal voor .EXE's)

Hoewel deze functie praktisch onopgemerkt is gebleven, heeft dit de veiligheid van Windows wel sterk verbeterd. Hiervoor had Windows namelijk geen enkele manier om aan Mandatory Access Control te doen. Helaas is er wel nog werk aan de winkel. Er is bijvoorbeeld een gebrek aan management of een configuratietool voor administrators. Daarnaast is het systeem ook niet feilloos, zoals er in Vyacheslav Rusakov wordt aangetoond. (Khanse, 2009) (Bradley, 2007) (Rusakov, 2016)

7.5 Windows Defender

Windows Defender is begonnen als een anti-spyware systeem, waarnaast men nog andere antivirus-functies moest installeren, zoals Microsoft Security Essentials, om een complete verdediging te hebben. Sinds Windows 8 heeft het echter ook definitief deze rol overgenomen.

Qua performantie krijgt Windows Defender gemengde reviews. Vooral de firewall van Windows Defender kan een last vormen op de CPU. Daardoor is de meest gezochte term voor Windows Defender hoe je het uitschakelt. Vaak is het beter om een extra third-party systeem te gebruiken voor de firewall, zoals een PFSense of een WatchGuard. (Hall, 2017) (Lefferts, 2017)

7.5.1 ACL

Het Windows-equivalent voor Seccomp is Windows Access Control Lists. De ACL bestaat uit een lijst van Access Control Entries (ACE's), waarbij elke ACE een trustee aanduidt en bepaalt welke Access Rights deze heeft. Helaas wordt het niet aangeraden om rechtstreeks met de inhoud van een ACL te werken. Er zijn functies en modules om deze te wijzigen, maar ook hier kan de administrator geen rechtstreekse invloed hebben op het systeem. (Rapid7, 2017)

7.6 Conclusie

Linux heeft een meer hands-on aanpak om zijn systemen te beveiligen, waarbij de administrator van het Linux-systeem veel zelf kan en moet bepalen. Dit betekent wel dat de veiligheid van een Linux-opstelling even goed is als de kennis van zijn administrator, wat zowel positief als negatief kan zijn. Windows daarentegen neemt veel controle van de administrator weg, en doet het werk voor hem. Dit betekent dat er een basisniveau is qua veiligheid, maar dat dit niveau ver onder het niveau van een Linux-opstelling kan liggen, vooral als de administrator reeds veel kennis verworven heeft.

8. Conclusie

De onderzoeksvraag voor deze bachelorproef luidde als volgt: "Hoe goed presteert Docker for Windows op vlak van performantie en veiligheid tegenover Docker for Linux?" Hiervoor werden er twee systemen opgesteld, waarbij de eerste opstelling een Windows Server 2016 met Docker for Windows is. De tweede opstelling bevatte een CentOS 7.4 met Docker for Linux. Vervolgens werd de performantie van beide systemen vergeleken door te kijken hoe lang beide nodig hadden om een volledige installatie uit te voeren en hoeveel tijd ze nodig hadden voor enkel de containers. Ten slotte werd de beveiliging van beide systemen onder de loop genomen, waarbij er vertrokken werd uit de veiligheidsmaatregelen die men kan nemen bij CentOS, en of Windows hier een equivalent voor had.

Welk systeem de betere performantie had was duidelijk. Bij de volledige installatie had Windows namelijk tot wel 10 keer meer tijd nodig om alles te installeren. Dit komt onder andere omdat Docker for Windows een GUI nodig heeft om te installeren. Daarnaast zijn de containers en base images groter dan die bij Linux. Dit zijn twee hekelpunten die zeker nog weggewerkt moeten worden door Docker en Microsoft, zeker als men Docker for Windows ook verkocht wil krijgen aan DevOps teams.

Aan de andere kant maakt deze grafische interface Docker wel veel toegankelijker voor beginnende developers die op een Windows OS werken. Deze hebben immers geen nood aan een volledig geautomatiseerde omgeving, maar hechten veel meer waarde aan een toegankelijke applicatie. Daarnaast zijn deze developers niet meer verplicht om een Linux VM te draaien op hun systeem. Ten slotte zullen deze beginnende developers ook niet evenveel containers draaien als een professionele onderneming, waardoor de grootte ervan minder belangrijk is.

Mijn conclusie voor performantie is dat Docker for Linux meer geschikt zal zijn voor

bedrijven die op een professionele manier willen omgaan met container-technologie, maar dat Docker for Windows meer geschikt is voor beginnende developers; zeker als men al werkt op een Windows-systeem.

Op vlak van veiligheid ligt de bal een beetje meer in het midden. Linux heeft enkele sterke voordelen, zoals SELinux, Firewalld en Seccomp, maar deze vereisen wel al enige voorkennis. De beveiliging van Linux is dus voor een groot deel afhankelijk van de kennis van zijn administrator. Daartegenover is de beveiliging van Windows veel meer basic, maar is er wel slechts een minimum aan veiligheid doordat Microsoft bepaalde beslissingen uit de handen van zijn administrators neemt en deze instelt voor hem. Dit is zowel een voor- als een nadeel, want ook Microsoft is niet feilloos. Hyper-V Containers zijn dan wel weer een pak veiliger dan de Linux Containers, doordat ze een extra niveau van isolatie hebben.

De resultaten van dit onderzoek vallen min of meer binnen de lijnen van de verwachtingen. Alleen de performantie kan ietwat schokkend genoemd worden. Als Docker en Microsoft Docker for Windows even competitief willen maken als Docker for Linux, is er nog werk aan de winkel. Op vlak van veiligheid zijn beide systemen equivalent aan elkaar, waar vooral de kennis van de administrator een grote rol speelt.

Verdere onderzoeksvragen die uit deze bachelorproef oprijzen zijn:

- Hoe goed integreert Docker for Linux met het Windows-besturingssysteem?
- Kun je de installatie van Docker for Windows beter automatiseren door PowerShell-modules aan te maken?
- Zijn er manieren om de grootte van de Hyper-V containers te verkleinen?

A. Onderzoeksvoorstel

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

A.1 Introductie

Voorheen kon men Docker enkel installeren op Linux servers. Veel keuze had men dus niet wat betreft het OS waarop men het wou deployen. Maar recent werd Docker ook geïntroduceerd voor Windows Server 2016, waarin men handig gebruik maakt van de Hyper-V containers die er ingebouwd in zitten. In de afgelopen 4 jaar waarin Docker op de markt is gekomen, heeft het veel gedaan voor DevOps-teams. Het is dus zeker de moeite waard om eens te kijken of de Hyper-V containers mooi integreren in Docker en hoe dat precies gebeurt. Het doel en de onderzoeksvraag van deze bachelorproef is dus: Hoe vlot werkt Docker op Windows Server 2016, zeker voor het deployen van .Net-applicaties, ten opzichte van Docker voor Linux?

A.2 State-of-the-art

Er werden reeds verschillende tests uitgevoerd met Docker, (**Boettiger2014**), waarin het gebruikt werd om makkelijk reproduceerbaar onderzoek uit te voeren. In de conclusie worden verschillende best practices voorgesteld. Er zal in deze bachelorproef onderzocht worden in hoeverre het mogelijk is om deze na te bootsen op een Windows Server. Het uiteindelijke doel is om te proberen de volledige opstelling te repliceren. Een andere studie, (**Salman2016**), heeft ook al de voordelen besproken van het werken met containers op

vlak van beveiliging, zoals bijvoorbeeld Sandboxing. Ook dit is een goede studie om te proberen reproduceren in de Windows omgeving, echter met een grotere uitdaging. In hun conclusie vermeldt men vooral cgroups en SELinux, functies die uniek voor Linux bestaan. Er zal dus onderzocht worden hoe deze nagebootst kunnen worden op een Windows Server. Verder zal in deze bachelorproef de beschikbare literatuur ook grondig worden getest, aangezien Docker nog maar sinds september 2016 beschikbaar is voor Windows en dat deze misschien nog dient te worden bijgewerkt. De belangrijkste officiële bronnen hiervoor zijn: (Docker, 2016), (Friis2016) en (Container2016). Deze zullen één voor één bekeken en getest worden op hun compleetheid.

A.3 Methodologie

Eerst en vooral zal er gestart worden met een literatuurstudie van de beschikbare informatie op de website en blogs van Docker. Daarbovenop zullen er ook verschillende onafhankelijke Ops-bronnen geraadpleegd worden, om na te gaan hoe zij hun Docker hebben opgesteld. Vervolgens zullen er een Windows Server 2016 en een CentOS-server met Docker worden opgesteld en zal er vergeleken worden hoe goed beiden het doen in het opstellen van containers. Als virtualisatie-platform zal VirtualBox gebruikt worden, waarbij de Windows Server opgesteld zal worden via Vagrant, Chocolatey en Powershell, en de CentOS met Vagrant en Ansible.

A.4 Verwachte resultaten

De verwachting is dat de Windows Server het op zijn minst even goed zal doen als Linux. Zeker op vlak van .Net-applicatie zou Windows met de Hyper-V containers geen problemen mogen hebben. De benodigde resources voor het efficiënt draaien van de Windows Server zullen waarschijnlijk wel hoger liggen dan de Linux Server.

A.5 Verwachte conclusies

De documentatie voor de Windows Server zal wel moeilijker te verkrijgen zijn, aangezien de technologie hierop vrij nieuw is en de meeste mensen hiervoor nog steeds Linux gebruiken. Echter de installatie zou vrij vlot moeten gaan via Powershell en Chocolatey. Beide zijn namelijk krachtige tools in hun eigen recht.

Bibliografie

- Arora, H. (2012). Linux system calls. Verkregen van https://www.ibm.com/developerworks/community/blogs/58e72888-6340-46ac-b488-d31aa4058e9c/entry/linux_system_calls20?lang=en
- Baker, I. M. J. (2018, mei 16). Introduction to the Bash Command Line. Verkregen van <https://programminghistorian.org/en/lessons/intro-to-bash>
- Billemont, M. (2018, mei 16). What is bash, and where does it live? Verkregen van <https://guide.bash.academy/inception/>
- Bradley, T. (2007). Introduction to Windows Integrity Control. Verkregen van <https://www.symantec.com/connect/articles/introduction-windows-integrity-control>
- Brown, K. (2014). The Beginner's Guide to iptables, the Linux Firewall. Verkregen van <https://www.howtogeek.com/177621/the-beginners-guide-to-iptables-the-linux-firewall/>
- Brown, T. (2017). Containers - Bringing Docker To Windows Developers with Windows Server Containers. Verkregen van <https://msdn.microsoft.com/en-us/magazine/mt797649.aspx>
- Cezar, M. (2016). Install Docker and Learn Basic Container Manipulation in CentOS and RHEL 7/6 – Part 1. Verkregen van <https://www.tecmint.com/install-docker-and-learn-containers-in-centos-rhel-7-6/>
- Christopher, T. (2017, januari 11). What Do Containers Have to Do with DevOps, Anyway? Verkregen van <https://containerjournal.com/2017/01/11/containers-devops-anyway/>
- Cloud, E. (2017, mei 1). Docker by the Numbers: DockerCon Shines Light on Containers in the Enterprise. Verkregen van <http://electric-cloud.com/blog/2017/05/docker-numbers-dockercon-shines-light-containers-enterprise/>
- Cooley, S. (2018, april 7). Introduction to Hyper-V on Windows 10. Verkregen van <https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/about/>
- Docker. (2016). *Docker for the Virtualization Admin*.

- Docker. (2018a). Install Docker CE from binaries. Verkregen van <https://docs.docker.com/install/linux/docker-ce/binaries/>
- Docker. (2018b). Install Docker for Windows. Verkregen van <https://docs.docker.com/docker-for-windows/install/#what-to-know-before-you-install>
- Ernest, M. (2017, juli 24). What Is DevOps? Verkregen van <https://theagileadmin.com/what-is-devops/>
- Evans, J. (2016). What even is a container: namespaces and cgroups. Verkregen van <https://jvns.ca/blog/2016/10/10/what-even-is-a-container/>
- Fulton, S. M. (2017, april 24). Dockercon 2017: How Docker Changed Windows So Windows Could Change Docker. Verkregen van <https://thenewstack.io/docker-changed-windows-windows-change-docker/>
- Grubor, S. (2017, november 1). *Deployment with Docker*.
- Hall, B. L. J. (2017). Windows Defender Firewall with Advanced Security. Verkregen van <https://docs.microsoft.com/en-us/windows/security/identity-protection/windows-firewall/windows-firewall-with-advanced-security>
- Henry-Stocker, S. (2012). Why SELinux is more work, but well worth the trouble. Verkregen van <https://www.networkworld.com/article/2717423/security/why-selinux-is-more-work--but-well-worth-the-trouble.html>
- Janetakis, N. (2017, augustus 8). Differences between a Dockerfile, Docker Image and Docker Container. Verkregen van <https://nickjanetakis.com/blog/differences-between-a-dockerfile-docker-image-and-docker-container>
- Jean-François, M. (2017). Een introductie tot Docker, veelgemaakte fouten en hoe ze te voorkomen. *Scriptie*.
- Juarez, S. C. J. T. H. (2018, januari 11). Hyper-V Architecture. Verkregen van <https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/reference/hyper-v-architecture>
- Juggery, L. (2017). Docker & SELinux: 30.000 foot view. Verkregen van <https://medium.com/lucjuggery/docker-selinux-30-000-foot-view-30f6ef7f621>
- Kaner, C. (2006, november 17). Exploratory Testing. Verkregen van <http://www.kaner.com/pdfs/ETatQAI.pdf>
- Khanse, A. (2009). Mandatory Integrity Control in Windows 10/8/7. Verkregen van <http://www.thewindowsclub.com/mandatory-integrity-control>
- Lefferts, R. (2017). Introducing Windows Defender Security Center. Verkregen van <https://blogs.windows.com/windowsexperience/2017/01/23/introducing-windows-defender-security-center/>
- Mark, H. (2012, januari 18). 13 Steps to Learn & Perfect Security Testing in your Org. Verkregen van <https://www.atlassian.com/blog/archives/13-steps-to-learn-perfect-security-testing-in-your-org>
- Maurice, S. (2018, april 5). Software testing terminologies explained. Verkregen van <https://www.capgemini.com/2018/04/software-testing-terminologies-explained/#>
- Mays, J. (2015). How To Install Docker on CentOS 7. Verkregen van <https://www.liquidweb.com/kb/how-to-install-docker-on-centos-7/>
- Mike, C. (2016). Containers are not VMs. Verkregen van <https://blog.docker.com/2016/03/containers-are-not-vms/>
- Moon, S. (2017, oktober 20). SQL container image size is 10x bigger for Windows than Linux. Verkregen van <https://github.com/Microsoft/mssql-docker/issues/186>

- Nick, J. (2017, september 26). Docker Community Edition. Verkregen van <https://nickjanetakis.com/blog/docker-community-edition-vs-enterprise-edition-and-their-release-cycle>
- Noga, C. (2016, december 1). Performance Testing vs. Load Testing vs. Stress Testing. Verkregen van <https://www.blazemeter.com/blog/performance-testing-vs-load-testing-vs-stress-testing>
- Pichardo, D. (2014, december 2). Provisioning Vagrant Windows Environments with PowerShell Desired State Configuration. Verkregen van <https://dennypc.wordpress.com/2014/12/02/vagrant-provisioning-powershell-dsc/>
- Ramandeep, K. (2018, april 16). What is Docker. Verkregen van <https://dzone.com/articles/what-is-docker-1>
- Rapid7. (2017). Understanding Access Control Lists. Verkregen van <https://blog.rapid7.com/2017/01/11/understanding-access-control-lists/>
- Reno, J. (2016). Linux containers vs. VMs: A security comparison. Verkregen van <https://www.infoworld.com/article/3071679/linux/linux-containers-vs-vm-a-security-comparison.html>
- Rickard, J. (2018, januari 3). Install Docker and run containers on Windows. Verkregen van <https://4sysops.com/archives/install-docker-and-run-containers-on-windows/>
- Rusakov, V. (2016). Malicious code and the Windows integrity mechanism. Verkregen van <https://securelist.com/malicious-code-and-the-windows-integrity-mechanism/76751/>
- Savill, J. (2015). The differences between Windows Containers and Hyper-V Containers in Windows Server 2016. Verkregen van <http://www.itprotoday.com/windows-8/differences-between-windows-containers-and-hyper-v-containers-windows-server-2016>
- Siron, E. (2017, januari 31). Client Hyper-V vs. VirtualBox. Verkregen van <https://www.altaro.com/hyper-v/client-hyper-v-vs-virtualbox/>
- Spijkers, D. A. (2013). Linux Kernel Security (it is necessary). Verkregen van <http://drsalbertspijkers.blogspot.be/2013/06/linux-kernel-security-it-is-necessary.html>
- Steven, V.-N. J. (2018, maart 21). What is Docker and why is it so darn popular? Verkregen van <https://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular/>
- Sturgeon, P. (2012). Vagrant: What, Why, and How. Verkregen van <https://code.tutsplus.com/tutorials/vagrant-what-why-and-how--net-26500>
- ThinkSys. (2017, juni 29). Strategies for Security Testing. Verkregen van <https://www.thinksys.com/qa-testing/strategies-for-security-testing/>
- TutorialsPoint. (2018). Docker - Images. Verkregen van https://www.tutorialspoint.com/docker/docker_images.htm
- Uhrig, T. (2014, mei 19). What is the difference between a Docker image and a container? Verkregen van <https://stackoverflow.com/questions/23735149/what-is-the-difference-between-a-docker-image-and-a-container>
- Vivek, S. (2018, april 17). Announcing Docker Enterprise Edition 2.0. Verkregen van <https://blog.docker.com/2018/04/announcing-docker-enterprise-edition-2-0/>
- Walsh, D. J. (2013). Your visual how-to guide for SELinux policy enforcement. Verkregen van <https://opensource.com/business/13/11/selinux-policy-guide>

- Wang, C. (2017, juni 29). What is Docker? Linux containers explained. Verkregen van <https://www.infoworld.com/article/3204171/linux/what-is-docker-linux-containers-explained.html>
- Weissig, J. (2013). Episode 14 Introduction to Linux Control Groups (Cgroups). Verkregen van <https://sysadminecasts.com/episodes/14-introduction-to-linux-control-groups-cgroups>
- xlegalles. (2017). Support Windows Nano Server. Verkregen van <https://github.com/chocolatey/choco/issues/1371>