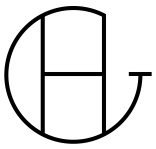


HoGent

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

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, er zal dus 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 uit te testen zal er vertrokken worden van een standaard Windows Server 2016 en CentOS 7.4 installatie waarop Docker geïnstalleerd zal worden. Waarna beide omgevingen getest zullen door middel van performatie-, integratie- 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, waarbij de CentOS server beter zal presteren op vlak van beveiliging en performantie, en dat Windows Server 2016 beter zal presteren op vlak van integratie. De documentatie voor CentOS zal ook meer compleet zijn, maar dat zowel Windows als Docker een grote inhaalbeweging aan het maken zijn.

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?

Voorwoord

Inhoudsopgave

1. Inleiding

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.

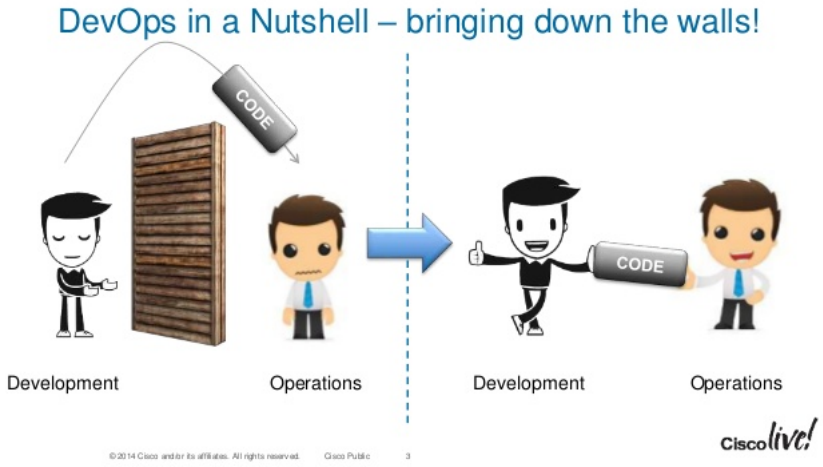
Het zou namelijk een vergissing zijn om Docker zomaar te negeren als men de DevOps-richting wil uitgaan. Het biedt immers verschillende tools aan voor zowel developers als system administrators. Bijvoorbeeld: Containers as a Service (CaaS) en role-based access control voor Operations (system administrators), 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.

1.1 Stand van zaken

1.1.1 DevOps

DevOps is een samenstelling van de woorden 'Development' en 'Operations': ontwikkeling en beheer. Voorheen werkten deze IT-groeperingen strikt gescheiden, waardoor je ofwel bij de ene groep zat ofwel bij de andere. Dit zorgde ervoor 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 hadden met het

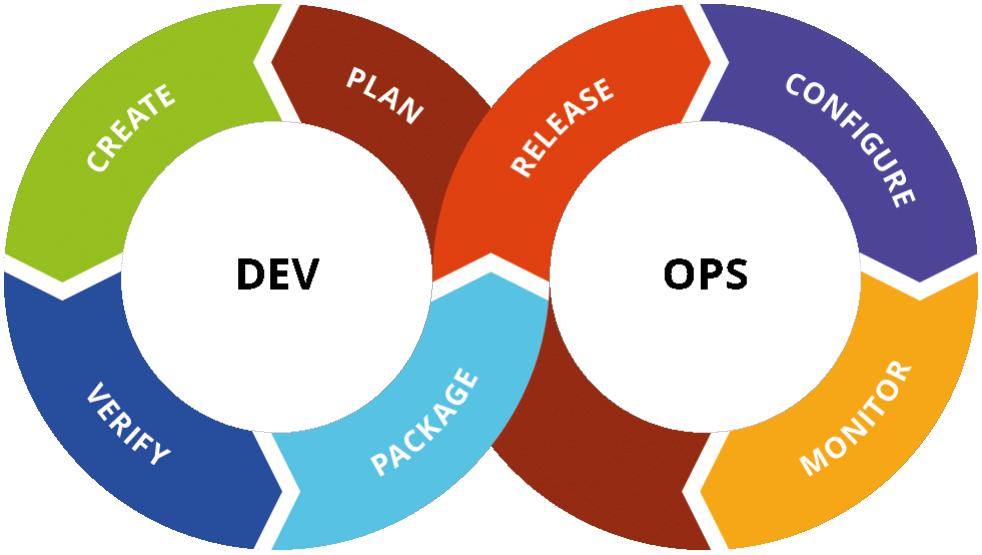
uitrollen, verliep de communicatie veelal stroef. De applicatie werkte immers op de computers van de developers, dus waarom zou dat niet het geval zijn bij de systeembeheerders? Echter, aangezien beide groeperingen onvoldoende kennis hadden over elkaars werkveld, was er veel onbegrip.



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

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

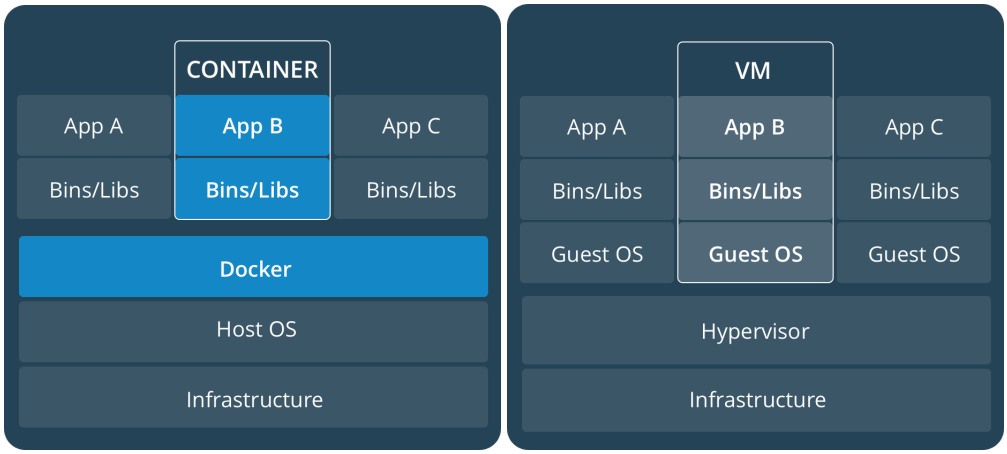
Bij DevOps worden deze groepen gedwongen om samen te werken in één team, zodat het geheel groter wordt dan de som. In praktijk vertaalt dit zich naar een cultuur die gestimuleerd dient te worden, waarin het automatiseren van zoveel mogelijk zaken centraal staat met een focus op continue ontwikkeling 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 snel tractie aan het winnen is.



1.1.2 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 zoals 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, namelijk 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 benodigde 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.



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 benodigdheden eerst in één geautomatiseerd pakket verzamelt. Dit pakket noemt een image. De uitgerolde versie van een image noemt men een Container en deze zal los van het host besturingssysteem werken.

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 aangeroepen 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 23 werkmethodes te onderscheiden, namelijk een standaard-, minimalistische- of 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 die hieronder meer uitgebreid besproken worden. Bij de minimalistische methode worden enkel de eerste en de laatste stap uitgevoerd.

- 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 varia-

bles 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.

Bij de geautomatiseerde methode maakt men ook gebruik van de 4 hierboven beschreven stappen. Maar, voegt men een extra component toe in 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, zullen alle services die hierin beschreven staan in orde gebracht worden door het 'docker-compose up'-commando uit te voeren.

Docker-compose is daarom vooral handig wanneer men meerdere services wil in orde 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.

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 voornaamste staan hieronder beschreven.

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

Doordat elk deel van een applicatie in een 'appartementje' zit, is het maken 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 zijn 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.

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
- More dan 900.000 applicaties die draaien op Docker
- Een verhoging van 40 procent in het toepassen ervan in 1 jaar

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 maandelijkse update, maar zonder garantie op stabiliteit van de runtime (Edge), ofwel voor een viermaandelijkse update voor gebruikers die meer stabiliteit prefereren, zowel op vlak van onderhoud als van beheer (Stable). Daarbovenop heeft de Enterprise Edition ook een meer uitgebreide ondersteuning met verschillende gradaties.

Docker CE

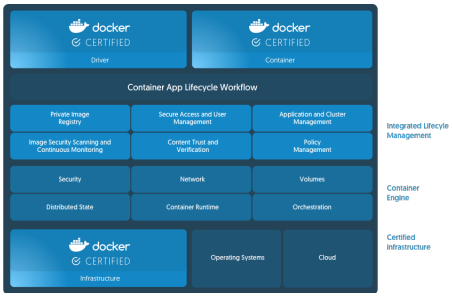
Zoals eerder aangehaald is Docker Community Edition het originele platform waarmee Docker begonnen is. Het is ideaal voor kleinere teams of onafhankelijke developers die willen experimenteren met Container-technologie. Deze technologie is beschikbaar voor zowel Linux als Mac, waardoor het perfect is 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.

Docker CE for Windows

Daarnaast heeft Docker ook een versie ontwikkeld specifiek gericht voor Developers die gebruik willen maken van Windows als ontwikkel platform. Docker CE for Windows maakt gebruik van dezelfde Daemon en Client als Docker CE. Maar, met een paar verschillen zoals: Docker CE for Windows maakt gebruik van de Windows-native Hyper-V Virtualisatie. Wat betekent dat de gebruiker een Windows 10 Pro edition nodig heeft of een Windows Server 2016. 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 commando voor. Ten slotte wordt docker-compose ook geïnstalleerd, naast Docker CE. Docker CE for Windows de versie die gebruikt werd voor de Windows opstelling.

Docker EE

Docker Enterprise Edition is het volledige pakket voor zij die op een professionele manier met een Containers-as-a-Serivce-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., met kenmerken zoals trusted delivery.



1.1.3 Hyper-V

Hyper-V is één van de technologie 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 virtualisatie technologie 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:

- Windows Hypercall

Hypercall voorziet communicatie met de Hypervisor.

- Windows hypervisor

Dit is een softwarelaag wiens primaire verantwoordelijkheid is om geïsoleerde execution enviroments te voorzien. Deze execution enviroments 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 verschil-

lende 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

Tenslotte 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.

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.
- 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.

1.1.4 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 moeten hiervoor requests sturen 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 overgedragen 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

1.1.5 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. Dit 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.

Om 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.

1.1.6 PowerShell

De configuratie van de Windows Server gebeurt grotendeels aan de hand van deze command-line inteface. Deze CLI, die bovendien gebruik maakt van het .NET framework om objecten terug te geven, is namelijk 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

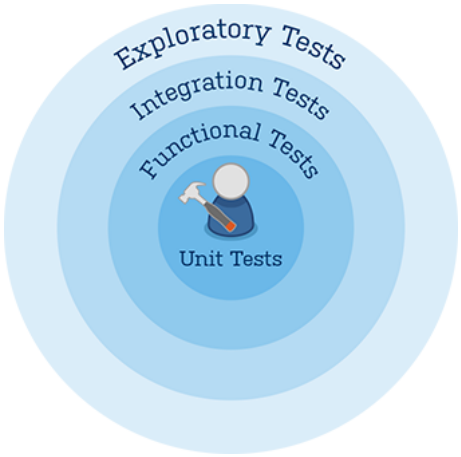
'werkwoord'-'te behandelen onderwerp', bijvoorbeeld: Get-Service.

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

1.1.8 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.



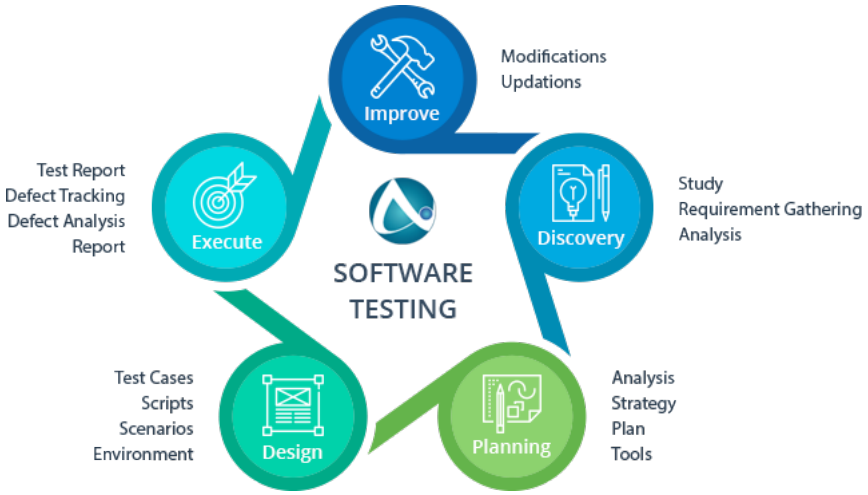
Verder zegt REF over software testing dat het een empirische en technische manier is om software of services te testen, doordat men op een objectieve manier kijkt naar de kwaliteit van een software of service. Dit is de voornaamste reden waarom voor deze methode gekozen is in deze bachelorproef. Daarnaast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

Meer specifiek wordt er gebruik gemaakt worden van performantie-, integratie- 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. (PerformanceTesting)

Bij integratietesten ligt de focus op de datacommunicatie tussen de verschillende modules. In dit geval tussen Docker, het besturingssysteem van de host en de uitgerolde applicatie. (IntegrationTesting)

Bij veiligheidstesten wordt er gekeken naar alle verschillende veiligheidsmaatregelen, en in hoeverre ze in staat zijn om de zwaktes van het systeem te beschermen. (SecurityTesting)



1.2 Probleemstelling en Onderzoeksvragen

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 werkomgeving nog gloednieuw.

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 goed is de technologie geïntegreerd met Hyper-V en met het Windows besturingssysteem in het algemeen?
- Hoe is het gesteld met de veiligheid? Hoe pakt men deze 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.

Dit onderzoek zal vooral een grote meerwaarde zijn voor DevOps-teams die op zoek zijn naar Windows-oplossingen voor hun problemen in verband met automatisatie en continue oplevering.

1.3 Opzet van deze bachelorproef

De rest van deze bachelorproef is als volgt opgebouwd:

In Hoofdstuk 2 wordt de methodologie toegelicht en worden de gebruikte onderzoekstechnieken besproken om een antwoord te kunnen formuleren op de onderzoeksvragen.

In Hoofdstuk 3 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 4 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 5 wordt gekeken naar het resultaat van de werklading dat beide systeem te verduren hebben gekregen en hoe ze beide gepresteerd hebben.

In Hoofdstuk 6 wordt bekeken hoe goed de Docker-module integreert met het hostsysteem, welke resources Docker beschikbaar heeft op het systeem en hoe Docker hiervan gebruik maakt.

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

In Hoofdstuk 8 wordt ten slotte een conclusie gevormd en wordt er een antwoord geformuleerd op de onderzoeksvragen. Hierbij wordt ook een aanzet gegeven naar mogelijk toekomstig onderzoek binnen dit domein.

2. Methodologie

2.1 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.

2.2 Bekijken van documentatie

Vervolgens werd de verzamelde documentatie voor beide omgevingen bekeken en beoordeeld op volledigheid. Aan de hand hiervan werd een conclusie getrokken.

2.3 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.

2.4 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 3 manieren van software testing uitgevoerd:

- Performance testing
- Integration testing
- Security testing

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

2.5 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.

3. 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 in 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.

3.1 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.

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.

3.2 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.

Ook hierin verschilte 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.

3.3 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, zijnde Docker-commando's.

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.

3.4 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.

Ook hier is het weer 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 toch weer wat moeilijker.

3.5 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.

4. 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.

4.1 Windows Server 2016

4.1.1 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 namelijk 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 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 de containers vlot en snel kan

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.

```
# -*- mode: ruby -*-

# vi: set ft=ruby :

VAGRANTFILE_API_VERSION = "2"

ENV['VAGRANT_DEFAULT_PROVIDER'] = 'virtualbox'

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|

    config.vm.network "private_network", ip: "192.168.56.16"

    # full version WS2016
    config.vm.box = "gusztavvargadr/w16s"

    # alternative:
    # config.vm.box = "mwrock/Windows2016"

    config.vm.network "public_network", bridge: "Intel(R) Dual Band Wireless-AC 3160"

    # configuration for virtualbox
    config.vm.provider "virtualbox" do |v|
        v.name = "DockerForWinVM"
        v.memory = 4096
        v.cpus = 2
    end

    # configuration for hyperv
    config.vm.provider "hyperv" do |v|
        v.vmname = "DockerForWinVM"
        v.memory = 4096
        v.cpus = 2
    end

    config.vm.boot_timeout = 420

    config.vm.hostname = "DockerForWinVM"

    config.vm.provision :shell, path: "Provisioning/Chocolatey.ps1"

    config.vm.provision :shell, path: "Provisioning/Docker-P1.ps1"

```

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 naar de installatie van Docker.

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

```
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
```

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:

```
#! /usr/bin/bash
# abort on nonzero exitstatus
set -o errexit
# abort on unbound variable
set -o nounset
# don't mask errors in piped commands
set -o pipefail

# Going to the directory which contains de .csproj-file
cd /home/vagrant/VoorbeeldProject/DienstenCheques

# cd /home/vagrant/09solSportsStore-master/SportsStore

# Creates dockerfile
echo "Creating dockerfile"
cat > Dockerfile << EOF
FROM microsoft/dotnet:2.0.5-sdk-2.1.4

RUN mkdir -p /app

WORKDIR /app

COPY . /app

RUN dotnet restore

RUN dotnet build

EXPOSE 5000/tcp

ENV ASPNETCORE_URLS http://*:5000

ENTRYPOINT ["dotnet", "run"]
EOF

# Creates docker-compose.yml
echo "Creating docker-compose-yml"
cat > docker-compose.yml << EOF
version: "3"

services:
  web:
    build: .
    ports:
      - "8080:5000"
    depends_on:
      - db
  db:
    image: "microsoft/mssql-server-linux"
    environment:
      SA_PASSWORD: "Vagrant123"
      ACCEPT_EULA: "Y"
EOF

# Building image
echo "Building docker image"
docker-compose build

# Running container
echo "Building container"
```

4.1.2 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.

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

- 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 namelijk dat het opnieuw wordt opgestart. Deze plug-in is de makkelijkst gevonden manier, zonder in te grijpen in het automatisatie-proces.

```
# 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#"
```

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

```
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
```

- docker-compose.bat

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

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

- 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.

```
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#"
```

- 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.

```
# 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"
```

```
# 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 "#BUIDLING IMAGE#"
docker-compose build

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

4.2 CentOS 7.4

4.2.1 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 basis installatie van een CentOS

7.4 server, maar voorziet wel de mogelijkheid om uitgerold te worden op alle populaire platformen.

```
# -*- mode: ruby -*-

# vi: set ft=ruby :

VAGRANTFILE_API_VERSION = "2"
# Staat standaard op virtualbox maar kan op hyperv gezet worden
# ALS JE HYPERV GEBRUIKT MOET JE config.vm.network IN COMMENTAAR ZETTEN
ENV['VAGRANT_DEFAULT_PROVIDER'] = 'virtualbox'

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|

    # IP Adres toevoegen aan virtuele machine
    # IN COMMENTAAR ZETTEN ALS JE HYPERV GEBRUIKT, NOG NIET ONDERSTEUND DOOR VAGRANT BIJ HYPERV
    config.vm.network "private_network", ip: "192.168.56.15"

    config.vm.box = "centos/7"

```

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.

```
    config.vm.provider "virtualbox" do |v|
        v.name = "CentOSDocker"
        v.memory = 4096
        v.cpus = 2
    end

    # configuration for hyperv
    config.vm.provider "hyperv" do |v|
        v.vmname = "DockerForWinVM"
        v.memory = 4096
        v.cpus = 2
    end

    config.vm.hostname = "CentOSDocker"

    config.vm.provision :shell, path: "scripts/prereq.sh"

    config.vm.provision :shell, path: "scripts/images.sh"

    config.vm.provision "file", source: "D:/Documenten/School/HoGent/StageEnBachelorproef/bachelorproef/BachelorproefDocker/CentOSDocker/VoorbeeldProje

    # config.vm.provision "file", source: "D:/Documenten/School/HoGent/StageEnBachelorproef/bachelorproef/BachelorproefDocker/CentOSDocker/09solSportsS

    config.vm.provision :shell, path: "scripts/applicaties.sh"
end

```

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 dezelfde webapplicatie te voorzien als voor Windows.

4.2.2 Bash

- prereq

In dit gedeelte van het script worden alle functionaliteiten voorzien die nodig zijn voordat men kan beginnen aan 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.

```
#! /usr/bin/bash

# Provisioning script

# Settings for Bash
# abort on nonzero exitstatus
set -o errexit
# abort on unbound variable
set -o nounset
# don't mask errors in piped commands
set -o pipefail

# The name of the user that is going to manage the Docker service
readonly docker_admin=vagrant

# Install utilities
echo "Installing utilities"

yum -y install epel-release \
               git \
               nano \
               patch

yum -y install python-pip

# Install Docker CE
echo "Installing Docker"

# Docker requires yum-utils and yum-config-manager
yum install -y yum-utils \
    device-mapper-persistent-data \
    lvm2

yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

# Installing Docker CE stable
yum install -y docker-ce

# Optional: Docker CE edge and test repository
# yum-config-manager --enable docker-ce-edge
# yum-config-manager --enable docker-ce-test

```

```
# Set Dockeradmin
echo "Adding user vagrant to group docker"
usermod -aG docker ${docker_admin}

# Service management
echo "Enabling services"
systemctl start docker.service
systemctl start firewalld.service

echo "Starting services"
systemctl enable docker.service
systemctl enable firewalld.service

# Test docker
echo "Testing Docker installation"

docker run hello-world

# Add ports to firewall
echo "Configuring firewall"
firewall-cmd --add-port=8080/tcp --permanent
firewall-cmd --add-port=80/tcp --permanent
firewall-cmd --add-port=5000/tcp --permanent
firewall-cmd --add-port=1433/tcp --permanent
firewall-cmd --reload

# Restarting Docker service due to firewall
echo "Restarting docker.service"
systemctl restart docker.service

# Install Docker-Compose
# sudo curl -L https://github.com/docker/compose/releases/download/1.21.0/docker-compose-

# chmod +x /usr/local/bin/docker-compose
echo "Installing docker-compose"
pip install docker-compose

echo "Testing Docker-compose"
docker-compose --version
```

- 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.

```
#! /usr/bin/bash
# abort on nonzero exitstatus
set -o errexit
# abort on unbound variable
set -o nounset
# don't mask errors in piped commands
set -o pipefail

# Required images
echo "Pulling images"

# MSSQL image
docker pull microsoft/mssql-server-linux

# .NET image
docker pull microsoft/dotnet:2.0.5-sdk-2.1.4
```

- 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.

```
#! /usr/bin/bash
# abort on nonzero exitstatus
set -o errexit
# abort on unbound variable
set -o nounset
# don't mask errors in piped commands
set -o pipefail

# Going to the directory which contains de .csproj-file
cd /home/vagrant/VoorbeeldProject/DienstenCheques

# cd /home/vagrant/09solSportsStore-master/SportsStore

# Creates dockerfile
echo "Creating dockerfile"
cat > Dockerfile << EOF
FROM microsoft/dotnet:2.0.5-sdk-2.1.4

RUN mkdir -p /app

WORKDIR /app

COPY . /app

RUN dotnet restore

RUN dotnet build

EXPOSE 5000/tcp

ENV ASPNETCORE_URLS http://*:5000

ENTRYPOINT ["dotnet", "run"]
EOF

# Creates docker-compose.yml
echo "Creating docker-compose-yml"
cat > docker-compose.yml << EOF
version: "3"

services:
  web:
    build: .
    ports:
      - "8080:5000"
    depends_on:
      - db
  db:
    image: "microsoft/mssql-server-linux"
    environment:
      SA_PASSWORD: "Vagrant123"
      ACCEPT_EULA: "Y"
EOF

# Building image
echo "Building docker image"
docker-compose build

# Running container
echo "Building container"
```

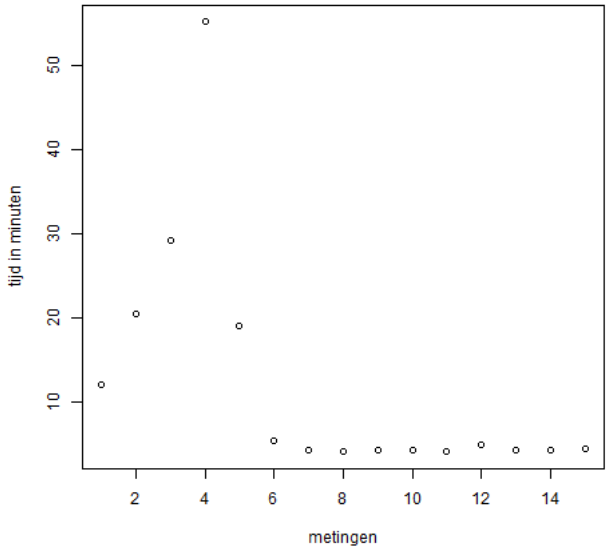
5. Performantie test

In dit hoofdstuk worden de resultaten besproken van de uitgevoerde performantietests. De performantie van beide opstellingen werden getests Door gebruik te maken van 'time vagrant up' en 'time vagrant provision'. Hierdoor werd er op het einde van de installatie de tijd getoond die Vagrant nodig had om het commando uit te voeren. Beide commando's werden elk 15 keer uitgevoerd.

5.0.1 CentOS 7.4

Performantie installatie

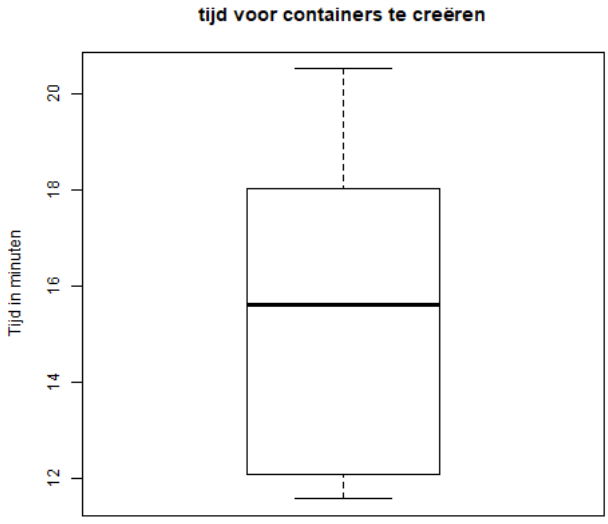
Als eerste kan men hieronder de resultaten zien van 'time vagrant up' voor de CentOS-opstelling. Dit toont hoeveel tijd Vagrant nodig had om de CentOS opstelling te installeren vanaf nul tot een werkende webapplicatie.



$\mu = 12.06457$

$\sigma^2 = 203.7204$

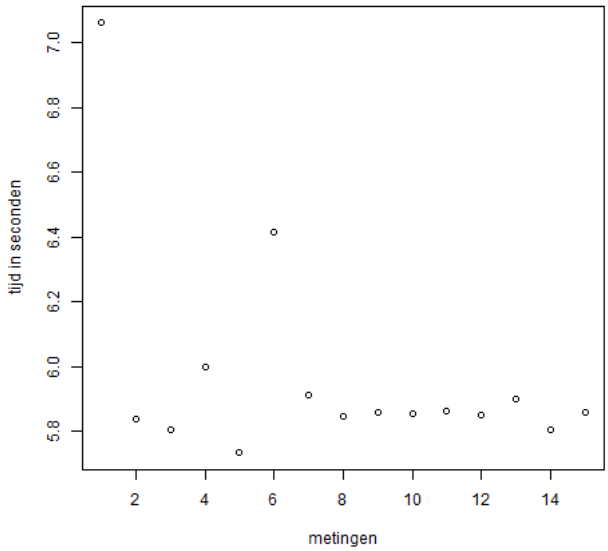
$\sigma = 14.27307$



De gemiddelde performatie-maat van de CentOS-opstelling is laag en vrijwel stabiel, met maar een paar uitschieters. De reden voor deze uitschieters komt omdat deze genomen zijn in een omgeving met een slechtere internetverbinding. Dit is dus de bottleneck bij deze opstelling en kan een grote impact hebben op de installatie, zoals te zien valt aan de hoge variatie en standaarddeviatie. Ondanks dit is de performantie van de CentOS-opstelling heel goed te noemen.

Performantie applicatie

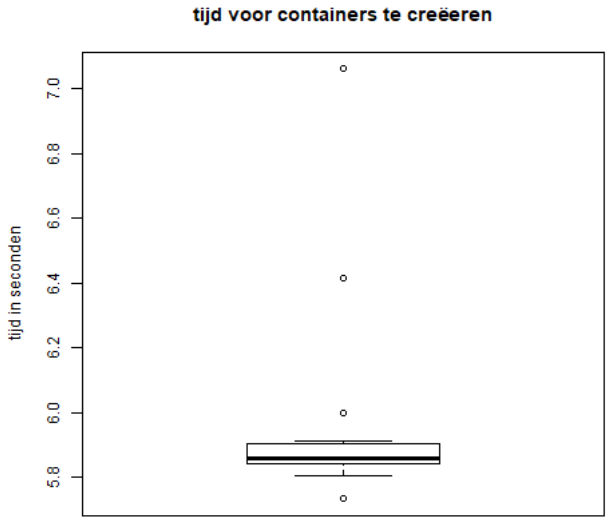
Daarnaast kan men hier de resultaten zien van 'time vagrant provision'. Bij deze resultaten moet het Operating System niet meer geïnstalleerd worden, alleen nog maar de containers voor de applicatie.



$\mu = 5.972333$

$\sigma^2 = 0.1150491$

$\sigma = 0.3391889$



Ook het opstellen van alleen de container gaat vlot. Met een stabielere gemiddelde tijd en een kleinere variantie en standaarddeviatie. Doordat er geen tijd meer nodig was om de Docker Images te downloaden lagen de tijden veel lager.

5.0.2 Windows Server 2016

Performantie installatie

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

De Windows opstelling brengt het er veel slechter vanaf dan de Linux opstelling. De gemiddelde tijd is immers 45 minuten en de variantie en standaarddeviatie liggen veel groter. Hier zijn er verschillende redenen voor:

- Installatie Windows OS

De installatie van het besturingssysteem zelf duurt langer omdat de Docker CE for Windows een GUI vereist. Met een Windows Server core zou de performantie al sterk verbeterd kunnen worden. Windows Server nano is helaas geen optie, omdat deze niet ondersteund wordt door Docker CE for Windows.

- Microsoft SQL server container

De MSSQL server container voor Windows is meer dan 10 keer zo groot als de Linux variant, 6GBs tegenover 479MBs. Hierdoor is het systeem een groot deel van zijn tijd bezig met downloaden van de Docker Image.

Performantie applicatie

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

Hier zijn de resultaten al iets meer genormaliseerd tegenover de CentOS-opstelling. Omdat de Windows-opstelling nu de extra bestanden voor de grafische interface of gigantische MSSQL image moet binnenhalen.

6. Integratie test

Etiam pede massa, dapibus vitae, rhoncus in, placerat posuere, odio. Vestibulum luctus commodo lacus. Morbi lacus dui, tempor sed, euismod eget, condimentum at, tortor. Phasellus aliquet odio ac lacus tempor faucibus. Praesent sed sem. Praesent iaculis. Cras rhoncus tellus sed justo ullamcorper sagittis. Donec quis orci. Sed ut tortor quis tellus euismod tincidunt. Suspendisse congue nisl eu elit. Aliquam tortor diam, tempus id, tristique eget, sodales vel, nulla. Praesent tellus mi, condimentum sed, viverra at, consectetuer quis, lectus. In auctor vehicula orci. Sed pede sapien, euismod in, suscipit in, pharetra placerat, metus. Vivamus commodo dui non odio. Donec et felis.

Etiam suscipit aliquam arcu. Aliquam sit amet est ac purus bibendum congue. Sed in eros. Morbi non orci. Pellentesque mattis lacinia elit. Fusce molestie velit in ligula. Nullam et orci vitae nibh vulputate auctor. Aliquam eget purus. Nulla auctor wisi sed ipsum. Morbi porttitor tellus ac enim. Fusce ornare. Proin ipsum enim, tincidunt in, ornare venenatis, molestie a, augue. Donec vel pede in lacus sagittis porta. Sed hendrerit ipsum quis nisl. Suspendisse quis massa ac nibh pretium cursus. Sed sodales. Nam eu neque quis pede dignissim ornare. Maecenas eu purus ac urna tincidunt congue.

Donec et nisl id sapien blandit mattis. Aenean dictum odio sit amet risus. Morbi purus. Nulla a est sit amet purus venenatis iaculis. Vivamus viverra purus vel magna. Donec in justo sed odio malesuada dapibus. Nunc ultrices aliquam nunc. Vivamus facilisis pellentesque velit. Nulla nunc velit, vulputate dapibus, vulputate id, mattis ac, justo. Nam mattis elit dapibus purus. Quisque enim risus, congue non, elementum ut, mattis quis, sem. Quisque elit.

Maecenas non massa. Vestibulum pharetra nulla at lorem. Duis quis quam id lacus dapibus interdum. Nulla lorem. Donec ut ante quis dolor bibendum condimentum. Etiam egestas

tortor vitae lacus. Praesent cursus. Mauris bibendum pede at elit. Morbi et felis a lectus interdum facilisis. Sed suscipit gravida turpis. Nulla at lectus. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Praesent nonummy luctus nibh. Proin turpis nunc, congue eu, egestas ut, fringilla at, tellus. In hac habitasse platea dictumst.

Vivamus eu tellus sed tellus consequat suscipit. Nam orci orci, malesuada id, gravida nec, ultricies vitae, erat. Donec risus turpis, luctus sit amet, interdum quis, porta sed, ipsum. Suspendisse condimentum, tortor at egestas posuere, neque metus tempor orci, et tincidunt urna nunc a purus. Sed facilisis blandit tellus. Nunc risus sem, suscipit nec, eleifend quis, cursus quis, libero. Curabitur et dolor. Sed vitae sem. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Maecenas ante. Duis ullamcorper enim. Donec tristique enim eu leo. Nullam molestie elit eu dolor. Nullam bibendum, turpis vitae tristique gravida, quam sapien tempor lectus, quis pretium tellus purus ac quam. Nulla facilisi.

7. Security test

Etiam pede massa, dapibus vitae, rhoncus in, placerat posuere, odio. Vestibulum luctus commodo lacus. Morbi lacus dui, tempor sed, euismod eget, condimentum at, tortor. Phasellus aliquet odio ac lacus tempor faucibus. Praesent sed sem. Praesent iaculis. Cras rhoncus tellus sed justo ullamcorper sagittis. Donec quis orci. Sed ut tortor quis tellus euismod tincidunt. Suspendisse congue nisl eu elit. Aliquam tortor diam, tempus id, tristique eget, sodales vel, nulla. Praesent tellus mi, condimentum sed, viverra at, consectetuer quis, lectus. In auctor vehicula orci. Sed pede sapien, euismod in, suscipit in, pharetra placerat, metus. Vivamus commodo dui non odio. Donec et felis.

Etiam suscipit aliquam arcu. Aliquam sit amet est ac purus bibendum congue. Sed in eros. Morbi non orci. Pellentesque mattis lacinia elit. Fusce molestie velit in ligula. Nullam et orci vitae nibh vulputate auctor. Aliquam eget purus. Nulla auctor wisi sed ipsum. Morbi porttitor tellus ac enim. Fusce ornare. Proin ipsum enim, tincidunt in, ornare venenatis, molestie a, augue. Donec vel pede in lacus sagittis porta. Sed hendrerit ipsum quis nisl. Suspendisse quis massa ac nibh pretium cursus. Sed sodales. Nam eu neque quis pede dignissim ornare. Maecenas eu purus ac urna tincidunt congue.

Donec et nisl id sapien blandit mattis. Aenean dictum odio sit amet risus. Morbi purus. Nulla a est sit amet purus venenatis iaculis. Vivamus viverra purus vel magna. Donec in justo sed odio malesuada dapibus. Nunc ultrices aliquam nunc. Vivamus facilisis pellentesque velit. Nulla nunc velit, vulputate dapibus, vulputate id, mattis ac, justo. Nam mattis elit dapibus purus. Quisque enim risus, congue non, elementum ut, mattis quis, sem. Quisque elit.

Maecenas non massa. Vestibulum pharetra nulla at lorem. Duis quis quam id lacus dapibus interdum. Nulla lorem. Donec ut ante quis dolor bibendum condimentum. Etiam egestas

tortor vitae lacus. Praesent cursus. Mauris bibendum pede at elit. Morbi et felis a lectus interdum facilisis. Sed suscipit gravida turpis. Nulla at lectus. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Praesent nonummy luctus nibh. Proin turpis nunc, congue eu, egestas ut, fringilla at, tellus. In hac habitasse platea dictumst.

Vivamus eu tellus sed tellus consequat suscipit. Nam orci orci, malesuada id, gravida nec, ultricies vitae, erat. Donec risus turpis, luctus sit amet, interdum quis, porta sed, ipsum. Suspendisse condimentum, tortor at egestas posuere, neque metus tempor orci, et tincidunt urna nunc a purus. Sed facilisis blandit tellus. Nunc risus sem, suscipit nec, eleifend quis, cursus quis, libero. Curabitur et dolor. Sed vitae sem. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Maecenas ante. Duis ullamcorper enim. Donec tristique enim eu leo. Nullam molestie elit eu dolor. Nullam bibendum, turpis vitae tristique gravida, quam sapien tempor lectus, quis pretium tellus purus ac quam. Nulla facilisi.

8. Conclusie

Curabitur nunc magna, posuere eget, venenatis eu, vehicula ac, velit. Aenean ornare, massa a accumsan pulvinar, quam lorem laoreet purus, eu sodales magna risus molestie lorem. Nunc erat velit, hendrerit quis, malesuada ut, aliquam vitae, wisi. Sed posuere. Suspendisse ipsum arcu, scelerisque nec, aliquam eu, molestie tincidunt, justo. Phasellus iaculis. Sed posuere lorem non ipsum. Pellentesque dapibus. Suspendisse quam libero, laoreet a, tincidunt eget, consequat at, est. Nullam ut lectus non enim consequat facilisis. Mauris leo. Quisque pede ligula, auctor vel, pellentesque vel, posuere id, turpis. Cras ipsum sem, cursus et, facilisis ut, tempus euismod, quam. Suspendisse tristique dolor eu orci. Mauris mattis. Aenean semper. Vivamus tortor magna, facilisis id, varius mattis, hendrerit in, justo. Integer purus.

Vivamus adipiscing. Curabitur imperdiet tempus turpis. Vivamus sapien dolor, congue venenatis, euismod eget, porta rhoncus, magna. Proin condimentum pretium enim. Fusce fringilla, libero et venenatis facilisis, eros enim cursus arcu, vitae facilisis odio augue vitae orci. Aliquam varius nibh ut odio. Sed condimentum condimentum nunc. Pellentesque eget massa. Pellentesque quis mauris. Donec ut ligula ac pede pulvinar lobortis. Pellentesque euismod. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent elit. Ut laoreet ornare est. Phasellus gravida vulputate nulla. Donec sit amet arcu ut sem tempor malesuada. Praesent hendrerit augue in urna. Proin enim ante, ornare vel, consequat ut, blandit in, justo. Donec felis elit, dignissim sed, sagittis ut, ullamcorper a, nulla. Aenean pharetra vulputate odio.

Quisque enim. Proin velit neque, tristique eu, eleifend eget, vestibulum nec, lacus. Vivamus odio. Duis odio urna, vehicula in, elementum aliquam, aliquet laoreet, tellus. Sed velit. Sed vel mi ac elit aliquet interdum. Etiam sapien neque, convallis et, aliquet vel, auctor non, arcu. Aliquam suscipit aliquam lectus. Proin tincidunt magna sed wisi. Integer blandit

lacus ut lorem. Sed luctus justo sed enim.

Morbi malesuada hendrerit dui. Nunc mauris leo, dapibus sit amet, vestibulum et, commodo id, est. Pellentesque purus. Pellentesque tristique, nunc ac pulvinar adipiscing, justo eros consequat lectus, sit amet posuere lectus neque vel augue. Cras consectetuer libero ac eros. Ut eget massa. Fusce sit amet enim eleifend sem dictum auctor. In eget risus luctus wisi convallis pulvinar. Vivamus sapien risus, tempor in, viverra in, aliquet pellentesque, eros. Aliquam euismod libero a sem.

Nunc velit augue, scelerisque dignissim, lobortis et, aliquam in, risus. In eu eros. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Curabitur vulputate elit viverra augue. Mauris fringilla, tortor sit amet malesuada mollis, sapien mi dapibus odio, ac imperdiet ligula enim eget nisl. Quisque vitae pede a pede aliquet suscipit. Phasellus tellus pede, viverra vestibulum, gravida id, laoreet in, justo. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Integer commodo luctus lectus. Mauris justo. Duis varius eros. Sed quam. Cras lacus eros, rutrum eget, varius quis, convallis iaculis, velit. Mauris imperdiet, metus at tristique venenatis, purus neque pellentesque mauris, a ultrices elit lacus nec tortor. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent malesuada. Nam lacus lectus, auctor sit amet, malesuada vel, elementum eget, metus. Duis neque pede, facilisis eget, egestas elementum, nonummy id, neque.

Lijst van figuren

Lijst van tabellen