



**HoGent**

Faculteit Bedrijf en Organisatie

Security en Managebility van Docker containers

Tomas Vercautter

Scriptie voorgedragen tot het bekomen van de graad van  
Bachelor in de toegepaste informatica

Promotor:  
Bert Van Vreckem  
Co-promotor:  
Bert Van Vreckem

Instelling: —

Academiejaar: 2015-2016

Tweede examenperiode



Faculteit Bedrijf en Organisatie

Security en Managebility van Docker containers

Tomas Vercautter

Scriptie voorgedragen tot het bekomen van de graad van  
Bachelor in de toegepaste informatica

Promotor:  
Bert Van Vreckem  
Co-promotor:  
Bert Van Vreckem

Instelling: —

Academiejaar: 2015-2016

Tweede examenperiode

## **Samenvatting**

# Voorwoord

In deze bachelorproef wordt er van uitgegaan dat de lezer een basis kennis heeft van docker en linux. De belangrijkste elementen van docker die men best vo

# Inhoudsopgave

<b>1</b>	<b>Inleiding</b>	<b>5</b>
1.1	Probleemstelling en context . . . . .	5
1.2	Doelstelling en onderzoeksvraag/-vragen . . . . .	6
<b>2</b>	<b>Methodologie</b>	<b>7</b>
<b>3</b>	<b>Docker</b>	<b>9</b>
3.1	Geschiedenis van Containers . . . . .	9
3.2	Wat is Docker . . . . .	10
3.3	Hoe gebruikt men Docker . . . . .	11
3.3.1	installatie . . . . .	11
3.3.2	docker images . . . . .	12
3.3.3	docker run . . . . .	15
<b>4</b>	<b>Security</b>	<b>17</b>
4.1	Host Configuration . . . . .	17
4.1.1	Up to date houden van het systeem . . . . .	18
4.1.2	Configureren van het systeem . . . . .	18

4.1.3	Auditen van het systeem . . . . .	19
4.2	Docker daemon configuratie . . . . .	19
4.2.1	Globale configuratie . . . . .	19
4.2.2	netwerk configuratie . . . . .	19
4.2.3	Permissies voor Docker config files . . . . .	20
4.3	Security of Docker images . . . . .	20
4.3.1	Images van de Docker hub . . . . .	20
4.3.2	zelf Docker Images maken . . . . .	22
4.4	Container configuration . . . . .	23
4.4.1	docker run commando configureren . . . . .	23
4.4.2	container configuratie . . . . .	25
<b>5</b>	<b>Manageability</b>	<b>27</b>
5.1	Manueel managen . . . . .	27
5.2	Managen met process manager . . . . .	29
5.3	Managen met tools . . . . .	30
<b>6</b>	<b>Logging en monitoring</b>	<b>31</b>
<b>7</b>	<b>Conclusie</b>	<b>33</b>

# Hoofdstuk 1

## Inleiding

Docker containers is een manier van virtualiseren van services. Hierbij worden services in een

In deze bachelorproef zal ik onderzoek doen naar de security en managebility van Docker containers en deze vergelijken met de security en managebility van het draaien van de services rechteerds op de virtuele machine.

Deze bachelorproef is onderverdeeld in drie grote delen. In de eerste plaats bespreek ik de security zelf. In een tweede deel zal ik het hebben over de managebility van Docker containers. En als laatste onderdeel bespreek ik logging bij docker containers.

### 1.1 Probleemstelling en context

Docker is een nieuwe speler op de virtualisatiemarkt die in korte tijd een substantieel marktaandeel hebben ingenomen. Hiervoor hebben ze linux LXC containers gebruiksvriendelijker gemaakt. Deze vorm van container ligt ergens in het midden tussen een chroot jail (isoleren van een proces van het rest van het systeem) en een volledige virtuele machine. Sinds Docker is uitgekomen voor het grote publiek zijn er altijd al vragen geweest naar in hoeverre deze vorm van virtualizatie veilig genoeg was om in productie te gebruiken. Als we in google iets zoeken rond problemen met docker of mensen die gestopt zijn met docker te gebruiken vinden we heel wat posts met als titel iets in de zin van "Why i stopped using docker". De meeste mensen stoppen met het gebruik van docker omdat ze problemen vinden met ofwel de beveiliging van de



container omgeving ofwel hebben ze problemen met het managen van hun containers. Deze kritieken of problemen met docker zijn er al sinds de eerste uitgave van docker en blijven nu en dan de kop op steken.

## 1.2 Doelstelling en onderzoeksvraag/-vragen

De doelstelling van deze bachelor proef bestaat uit het ophelderen van kritieken op vlak van de manageability en beveiliging van deze containers. Hiervoor gaan we eerst op zoek naar welke kritiekpunten er de kop opsteken. We zullen zo veel mogelijk kijken naar recentere kritieken omdat Docker rap geëvolueerd is kunnen kritieken van twee maand geleden al verouderd zijn. Daarna zullen we in de eerste plaats kijken als deze kritiek gerechtvaardigd is. Indien dit het geval is zal er onderzocht worden hoe we deze kritiekpunten mogelijks kunnen wegwerken met bepaalde zelf toepasbare technieken of best-practices? Of zijn er elementen die zullen moeten geïmplementeerd worden door de Docker community zelf?

Daarnaast zijn er veel mensen die van Docker containers een veiligere en meer handelbare ervaring verwachten dan als ze met virtuele machines werken. Dus zullen we ook kijken hoe kunnen we Docker containers minstens op hetzelfde niveau krijgen zowel qua beheersbaarheid als op beveiligings vlak als een klassieke virtuele machine?

# Hoofdstuk 2

## Methodologie

Een eerste stap in het onderzoek naar problemen zowel op vlak van security als van manageability bij Docker containers was mezelf wegwijs maken in de wereld van docker. Hiervoor heb ik twee maanden lang elke dag docker intensief gebruikt. Hierbij heb ik een volledige omgeving opgezet waarop applicaties gedeployed werden. Hierdoor kon ik zelf een beeld krijgen van de problemen die andere mensen hebben met Docker en kon ik ook de eventuele oplossingen zelf testen.

Tijdens het opzetten van de omgeving ben ik ook op zoek gegaan naar eventuele kritieken die op het internet te vinden waren en hoe men deze kritieken staaften. In de eerste plaats ging ik op zoek naar eventuele problemen in verband met beveiliging en wanneer die onstonden. Daarnaast ga ik ook op zoek naar problemen in verband met monitoring/manageability. Na het intensief gebruiken van docker en het onderzoek naar problemen had ik een beter zicht welke problemen er nu effectief bestaan in verband met docker container en of deze terecht zijn. Maar ook welke problemen er al waren opgelost.

Na het vinden van deze kritieken moet er gekeken worden of deze kritieken ook voorkomen bij virtuele machines. Meerbepaald of er vroeger op dit vlak ook problemen waren met de virtuele oplossingen en indien zo hoe werd dit opgelost? Ook heb ik gekeken naar hoe de beveiliging en manageability van gewone virtuele machines wordt aangepakt. Dit zorgt er voor dat ik een goed idee heb van waar andere systemen problemen mee hadden en hoe dit verholpen werd/wordt.

Hierbij kan ik dan de gevonden problemen zowel voor manageability als security bij Docker containers gaan vergelijken met de probleempunten bij virtuele machines. Zijn er elementen die we kunnen oplossen op dezelfde of een gelijkaardige manier als bij

virtuele machines. En zijn er dingen die we nog niet hebben tegengekomen bij virtuele machines maar nu wel moeten opgelost worden met docker?

Als laatste ga ik dan kijken hoe we deze kritiekpunten kunnen verhelpen. Meer bepaald zijn er tools, technieken of best-practices die we kunnen gebruiken om dit te verhelpen of zijn er dingen die we niet zelf kunnen oplossen maar moeten opgelost worden door de docker defs?

# Hoofdstuk 3

## Docker

### 3.1 Geschiedenis van Containers

Docker op zich bestaat nog niet zo lang. Versie 0.1.0 is uitgekomen op 26 maart 2016 en versie 1.0.0 en dus de eerste effectieve release kwam uit op 9 juni 2014. Hierdoor is er nog niet veel geschiedenis aan Docker. Maar als we terugkijken naar de evolutie van containers gaat dit verder terug. Het idee van containers gaat terug tot 1979 met de toevoeging van chroot in UNIX V7. chroot is een eerste concept van containerization. Hierbij wordt voor elk proces een geïsoleerde ruimte op de schijf voorzien, ook wel een "chroot jail" genoemd. Na chroot duurde het tot het jaar 2000 tot het volgende op de markt kwam. FreeBSD Jails(2000) en Linux VServer(2001) waren beide jail mechanismen die verder bouwden op chroot. Hierna kwamen enkele oplossingen op de markt die al dichterbij de container die we nu kennen aansluiten namelijk Solaris Containers(2004), OpenVZ(2005), Process Containers(2006) en Control Groups(2007).

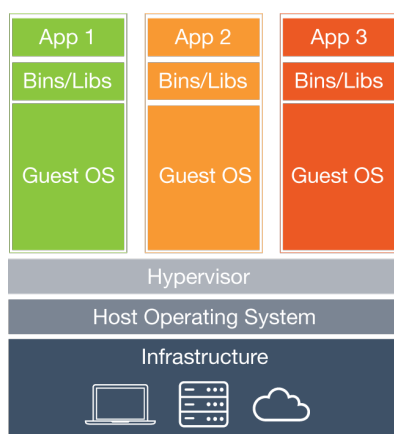
Daarna werd in 2008 LXC (Linux Containers) toegevoegd aan de linux kernel. Dit was de basis voor latere containerizatie technologieën. LXC combineerde het gebruik van cgroups (dit stond in voor isolatie en resource management) en namespaces (dit zorgde ervoor dat groups opgesplitst konden worden waardoor ze elkaar niet kunnen "zien") Ellingwood (2015). In 2014 werd de versie 1.0 van LXC uitgebracht. Hierbij werd onder andere support toegevoegd voor SELinux. De eerste versies van Docker waren ontwikkeld als een interne tool bij het bedrijf dotCloud. Deze tool zorgde voor een versimpeling voor het gebruik van LXC. Later werd dit uitgebracht onder de naam Docker en werd de LXC container vervangen door hun eigen driver LibContainer.

Docker zorgde als een van de eerste voor een volledig ecosysteem voor het managen van containers Gunaratne (2016).

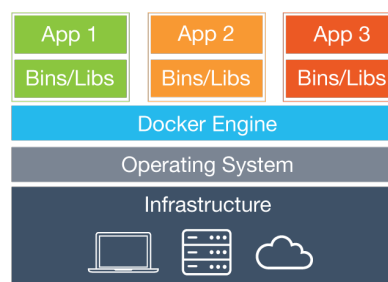
## 3.2 Wat is Docker

Docker is een open source project om een applicatie te verpakken, verzenden en te draaien als een weinig resources gebruikende container. Anders gezegd word in een Docker Container alles om een programma te draaien verpakt in een compleet bestandstelsysteem namelijk: code, runtime, systeem tool en systeem libraries. Alles wat je normaal op een server zou installeren. Dit garandeerd dat de applicatie zich altijd op dezelfde manier zal gedragen in welke omgeving het ook draait Docker (2016j).

Een vergelijking met virtuele machines is rap gemaakt. Voor resource isolatie werkt het op dezelfde manier. Maar waar de verschillen liggen is de grootte, flexibiliteit en prestatie. Doordat Docker gebruik maakt van containerization gebeurt de virtualisatie op de kernel niveau tegenover virtuele machines waarbij dit niet zo is.



Figuur 3.1: Virtualisatie bij Virtuele machines- Docker (2016j)



Figuur 3.2: Virtualisatie bij Docker Containers - Docker (2016j)

Een korte visualisatie van hoe containers werken kunnen we zien op de voorgaande foto's. Hierbij zien we dat traditioneel bij virtuele machines we op de host die zich rechtstreeks bevindt op de hardware een hypervisor nodig hebben die onze hardware gaat virtualiseren voor gebruik van de virtuele gast machines. Op elke gast machine bevindt zich een eigen besturingssysteem die uniek is voor elke gast. Daarnaast moeten ook onze applicatie draaien samen met de binaries en libraries die het nodig heeft. Bij containerisatie en specifiek voor docker containers wordt de hypervisor vervangen door

de docker engine en gaan we voor elke applicatie onze binaries en libraries samen met de applicatie zelf gaan draaien in onze container. Hierdoor krijgen we snellere deployment, opstart tijden en migratie en daarnaast hebben we ook veel minder overhead doordat niet eke applicatie moet draaien op een eigen besturingssysteem en een eigen kernel.

### 3.3 Hoe gebruikt men Docker

Doordat in deze bachelorproef zeer veel met docker gewerkt wordt en het relatief nieuwe software is zal ik in deze sectie uitleggen hoe men zelf met docker aan de slag kan. Het basis gebruik van docker is bewust simpel gehouden. Maar we mogen niet vergeten dat we over een krachtige en flexibele tool beschikken waardoor we docker ook complex kunnen maken.

#### 3.3.1 installatie

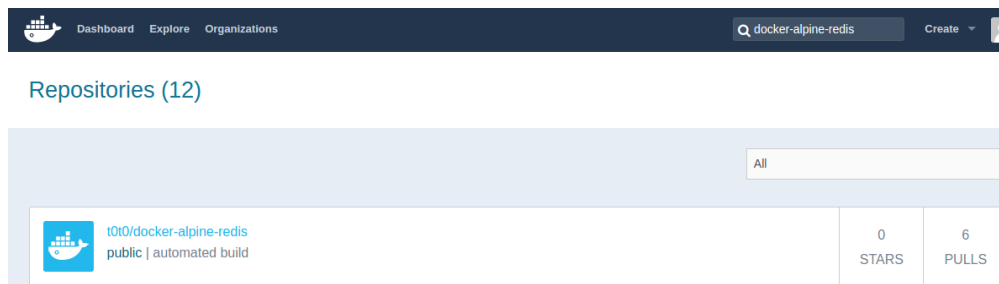
De installatie van docker kan op verschillende manieren. Een van deze manieren is met het 'curl' commando. Daarnaast beschikt Docker ook over een 'apt' en 'yum' repository. Installatie met de package managers voor 'apt' en 'yum' doen we met 'sudo apt-get install Docker' en 'sudo yum install Docker' respectievelijk. Voor de installatie met 'curl' gaan we uit van een Ubuntu systeem maar kan ook uitgevoerd worden met een andere package manager op een ander linux besturingssysteem. Een eerste stap is de installatie van 'curl' zelf. Hiervoor kijken we eerst als we 'curl' al geïnstalleerd hebben. 'sudo wich curl' geeft ons de geïnstalleerde versie van 'curl'. Indien het not niet geïnstalleerd is kunnen we met 'sudo apt-get update' en 'sudo apt-get install curl' het installeren. Deze stap kan op andere besturingssystemen uitgevoerd worden met de package manager van dat besturingssysteem. In veel besturingssysteem wordt curl al megeleverd van de manufacturer. Daarna kunnen we het commando 'curl -fsSL <https://get.docker.com/> | sh' uitvoeren. Dit zal Docker downloaden en installeren. Dit is op de vershillende besturingssystemen hetzelfde. Als we Docker installeren via een user moeten we er rekening mee houden dat we de deze user root rechten geven, meer hierover in het hoofdstuk over security. Na het installeren van Docker kunnen we met het commando 'docker run hello-world' nagaan of onze installatie gelukt is. Indien de installatie succesvol was moeten we nu samen met nog andere output het bericht 'Hello from Docker. This message shows that your installation appears to be working correctly.' krijgen.

### 3.3.2 docker images

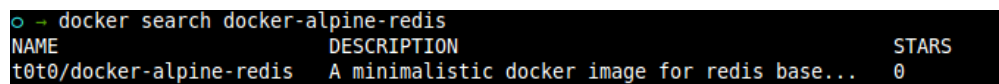
Wanneer we images gebruiken hebben we twee opties. Aan de ene kant kunnen we de docker images van een repository gebruiken (bvb. de Docker hub) aan de andere kant hebben we ook de optie om onze eigen images te maken.

#### Docker images van een repository

In dit voorbeeld gaan we de Docker hub repository gebruiken. Dit is de officiële repository van Docker. Iedereen kan images uploaden naar de Docker hub. Het enige dat daarvoor benodigd is, is een Docker hub account. Om te kijken welke Images er beschikbaar zijn hebben we 2 opties. Ofwel kijken we op de website van docker hub (hub.docker.com). Hier kunnen we met behulp van de search bar (zie image) Docker images zoeken. We kunnen dit ook direct van de command line. Dit doen we met het commando 'docker search ZOEKTERM', hierbij doorzoeken we alle geïnstalleerde registries op onze machine (standaard is dit enkel de Docker hub). Om een voorbeeld te geven gaan we op zoek naar een image genaamd 'docker-alpine-redis' het resultaat van deze zoekopdracht volgens de twee voorgaande methodes kunt u vinden in de volgende figuren.



Figuur 3.3: Zoeken naar images op de docker hub site



Figuur 3.4: Zoeken naar images via command-line

#### Eigen Docker images maken

Naast het gebruik van de Docker hub kunnen we ook onze eigen images maken die we desgewenst kunnen uploaden naar de Docker hub. Het maken van een docker image begint bij een Dockerfile. In deze Docker file definiëren we hoe onze container er moet uitzien wanneer we de image inladen in die container. Een eerste stap is het aanmaken van een map waarin we gaan werken. Deze map is de context die naar de docker daemon wordt gestuurd om de image te maken, dit wil zeggen dat de docker daemon die de image gaat maken voor ons niet buiten deze folder kan op zoek gaan naar files die we willen toevoegen. Dus alle files die we willen toevoegen aan onze image moeten we in deze folder plaatsen. na het aanmaken van deze map voegen we er een file genaamd 'Dockerfile' (case sensitive) aan toe. Een voorbeeld van zo een 'Dockerfile' vinden we hieronder.

```
1 # Base image alpine
2 FROM alpine:3.3
3
4 MAINTAINER Tomas Vercautter & Toon Lamberigts
5
6 # Environment variabelen
7 ENV REDIS_VERSION=3.0.7
8 ENV REDIS_IMAGE=redis-$REDIS_VERSION
9 ENV REDIS_IMAGE_TAR=$REDIS_IMAGE.tar.gz
10 ENV REDIS_DOWNLOAD_URL=http://download.redis.io/releases/
    $REDIS_IMAGE_TAR
11
12 # Install redis en dependencies
13 RUN apk --no-cache add --virtual .dependencies \
14     make \
15     gcc \
16     wget \
17     linux-headers \
18     musl-dev \
19     tcl \
20     tar && \
21     wget "$REDIS_DOWNLOAD_URL" && \
22     tar xzf $REDIS_IMAGE_TAR && \
23     cd $REDIS_IMAGE && \
24     make && \
25     cp src/redis-server /usr/bin/ && \
26     cp src/redis-cli /usr/bin/ && \
27     rm -r /$REDIS_IMAGE && \
28     rm -r /$REDIS_IMAGE_TAR && \
29     apk del .dependencies && \
30     rm -rf /var/cache/apk/* && \
31     mkdir /data
32
33 # Commando die wordt uigevoerd bij het starten van container
34 CMD redis-server --dir /data --appendonly yes
```

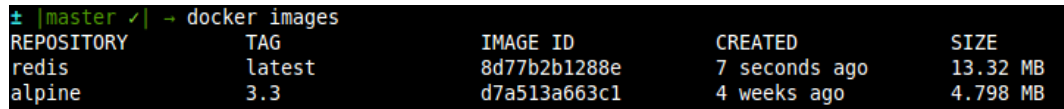


```
35  
36 # Expose de poort voor redis  
37 EXPOSE 6379
```

In deze Dockerfile zien we een aantal sleutelwoorden in het rood. deze zijn de verschillende elementen van de Dockerfile. Docker images kunnen gebaseerd zijn op andere Docker images. Dit wordt gedefinieerd met het sleutelwoord 'FROM' hier kan je zien dat deze image gebaseerd is op een 'alpine' images, meerbepaald de image met tag '3.3'. Het volgende sleutelwoord dat we tegenkomen is 'MAINTAINER' dit voegt aan de image info een 'author' tag toe. Dit kunnen we zien met 'docker inspect NAME'. Dit toont ons alle informatie over de container of image gedefinieerd met 'NAME'. Het 'MAINTAINER' sleutelwoord geeft verder geen invloed op de werking van de image. Daarna komen we het 'ENV' sleutelwoord tegen. Hiermee kunnen we environment variabelen declareren voor de image. Dit wordt vaak gebruikt om versies van software aan te duiden waardoor de Dockerfile makkelijker leesbaar blijft. Het 'RUN' sleutelwoord duid aan welke commando's uitgevoerd worden bij het aanmaken van de image. Dit zijn de commando's die normaal zouden uitgevoerd worden bij het installeren van de gewenste software. In dit geval installeren we redis dus hebben we eerst de dependencies voor het installeren van redis in een groep geïnstalleerd waardoor we die later kunnen verwijderen als groep. Hierna wordt redis gedownload van de officiële site, uitgepakt en geïnstalleerd. Na de installatie ruimen we alles op om de image zo klein mogelijk te houden. Dan hebben we twee sleutelwoorden die het runnen van de container makkelijker maken, namelijk 'CMD' en 'EXPOSE'. 'CMD' zorgt ervoor dat het commando dat erna komt uitgevoerd wordt wanneer we een container met deze image opstarten en 'EXPOSE' zet een poort van de container, in dit geval 6379, open. Door deze twee sleutelwoorden moeten we de twee overeenstemmende opties niet meer meegeven in ons run commando.

Het creëren van een image uit een Dockerfile doen we met het docker build commando doen we met het commando 'docker build [OPTIONS] PATH' waarbij de 'PATH' wordt vervangen door het pad (relatief of absoluut) naar de buildcontext die zal worden meegegeven naar de docker daemon voor het bouwen van de image. In ons geval is dit de map die we daarnet hebben aangemaakt. De Docker daemon zal an op zoek gaan in de root van de buildcontext naar een Dockerfile en de met sleutelwoorden gedefinieerde commando's uitvoeren en een Docker image maken van onze Dockerfile. Indien de dockerfile niet in de root van onze buildcontext bevindt kunnen we met de optie '-f' een relatief pad naar onze Dockerfile toevoegen. Met de optie '-t' kunnen we een eigen tag meegeven aan onze image. Dit helpt ons met het identificeren welke image wat is. In ons voorbeeld met de Dockerfile voor redis is het commando 'docker build -t redis .' indien we ons in de map met de Dockerfile bevinden. Na het uitvoeren van het commando hebben we een image genaamd redis gemaakt met onze Dockerfile.

Dit kunnen we controleren met het 'docker images' commando. Dit geeft ons een lijst met alle images die zich lokaal op onze machine bevinden. In ons geval geeft dit de volgende output. Hierop kunnen we zien dat de image voor 'alpine:3.3' is gedownload doordat we dit als base image hebben gedefinieerd en dat onze 'redis' image is aangemaakt.



REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
redis	latest	8d77b2b1288e	7 seconds ago	13.32 MB
alpine	3.3	d7a513a663c1	4 weeks ago	4.798 MB

Figuur 3.5: Output van het docker images commando

### 3.3.3 docker run

Met het 'docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]' commando kunnen we een image naar keuze inladen in een container. Het basis commando hebben we al eerder gebruikt met 'docker run hello-world' dit neemt de Docker image 'hello-world' en laad deze in een container waardoor ze kan interageren het environment. Bij het uitvoeren van het 'docker run' commando kunnen we zowel een locale image als een image uit een repository kiezen. Om een image te kiezen van uit een repository moeten we niets extra doen. De docker software kijkt bij het uitvoeren van het commando eerst als we image lokaal hebben. Indien dit het geval is wordt de image in een container geladen en opgestart. Indien dit niet het geval is zoekt de docker software naar te image in de voorgedefinieerde repositories (standaard enkel docker hub) en als het de gekozen image vind wordt het gedownload en ingeladen en opgestart.

Daarnaast kunnen we ook opties meegeven aan het run commando. Hiermee kunnen we volumes mounten met de '-v' optie, extra poorten exposen die nog niet in de image gedefinieerd zijn met '--expose', poorten van de container mappen op poorten van de host met de '-p' optie, een naam megeven aan onze container met de optie '--name' of de container detached laten draaien met de optie '-d' indien we dit niet meegeven wordt de container attached aan ons terminal venster en kunnen we het voor niets anders meer gebruiken. Dit zijn slechts enkel voorbeelden van opties die we hebben de rest van de opties zijn te vinden in de docker run reference (TODO INSERT LINK TO REFERENCE BIJLAGE?). Ook kunnen we commando's meegeven bij het runnen van een container. Het meegegeven commando wordt uitgevoerd als we de container opstartn. Er moet altijd een commando meegegeven worden als we een container willen starten maar dit kan ook worden meegegeven in de image. Als voorbeeld zullen we de redis image dat we daarnet hebben gemaakt laden in een

container. Hiervoor gebruiken we het commando 'docker run -d --name redisContainer redis'. Dit maakt van onze image genaamd redis een container genaamd redisContainer die detached draait. Om te controleren als onze container nu draait gebruiken we het commando 'docker ps' dit geeft ons een lijst van alle draaiende containers op onze machine.

```
root@master ~# docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS          NAMES
5b000f28ed5d   redis     "/bin/sh -c 'redis-se"  6 seconds ago  Up 6 seconds  6379/tcp       redisContainer
```

Figuur 3.6: Output van het docker ps commando

# Hoofdstuk 4

## Security

Op vlak van security ben ik gestart van CIS Docker 1.6 Benchmark geschreven door Pravin Goyal (2015). Hierin is in samenwerking met de "Center for Internet Security-onderzoek gedaan naar de beveiliging van docker containers. Dit onderzoek is onderverdeeld in zes grote onderdelen namelijk Host configuration, Docker daemon configuration, docker daemon configuration files, Container images and build file, Container runtime en docker security operations. Hierbij wordt er gekeken welke van deze elementen standaard in docker worden uitgevoerd. In dit hoofdstuk zal ik dezelfde zes onderdelen behandelen en vergelijken. Hierbij zal ik kijken hoe we deze beveiliging aanpakken wanneer we onze services in een Docker container draaien tegenover wanneer we deze native in onze virtuele machine draaien.

Hierbij heb ik ook telkens deze methodes voor Docker containers uitgetest om te kijken hoe makkelijk deze kunnen geïmplementeerd worden in een nieuwe omgeving of bestaande omgeving. Een grote hulp hiervoor was de github repository "docker-bench-security" van docker een grote hulp. Dit is een script die alle elementen die worden aangehaald in CIS Docker 1.6 Benchmark automatisch getest en feedback over gegeven.

### 4.1 Host Configuration

Een van de belangrijkste onderdelen in het beveiligen van de Docker stack is het beveiligen van de host machine waarop de docker containers zich zullen bevinden. Dit heeft drie onderdelen namelijk het up to date houden van het systeem, het juist

configureren van het systeem, en het auditten van het systeem.

### **4.1.1 Up to date houden van het systeem**

Om de veiligheid van een systeem te kunnen garanderen moet er ten allen tijde gezorgd worden dat het systeem up to date blijft. Dit zorgt ervoor dat security flaws die gekend en gepatcht zijn ook op het systeem beveiligd zijn. Doordat we Docker gebruiken komt hier nog een extra laag bij. Bij het Updaten van een machine waarop de services rechtstreeks draaien moet enkel het host systeem, de kernel van het host systeem en de services up to date blijven. Doordat Docker een kernel deelt met het hostsysteem wordt het up to date houden van de kernel belangrijker. Naast het onderhouden van de voorvernoemde elementen moet er nu, doordat we Docker gebruiken, nog met twee extra dingen rekening gehouden worden. Enerzijds moet Docker zelf nu ook upgedate worden, maar een onderdeel die soms vergeten wordt is dat elke container nu ook zijn eigen besturingssysteem gebruikt. Hierover wordt er verder bij het beveiligen van Container images meer uitleg over gegeven.

### **4.1.2 Configureren van het systeem**

Bij het configureren van het hostsysteem wordt net zoals bij het up to date houden van het systeem dezelfde stappen van een native applicatie op een hostsysteem configureren gevolgd. Maar door de extra laag komen daar nog extra stappen bij. Bij een systeem waar Docker op draait wordt er aangeraden om een aparte partitie te creëren voor de containers. Dit is een simpele operatie waardoor alle docker gerelateerde files zich niet meer tussen de files van het hostsysteem bevinden.

Daarnaast moet ook onder controle gehouden worden wie toegang krijgt tot de docker daemon. De Docker daemon heeft 'root' access, gebruikers die worden toegevoegd aan de 'docker' usergroup en geen 'root' privileges hebben kunnen hierdoor 'root' access verkrijgen tot het hostsysteem. Men directories sharen tussen het host systeem en een guest container. Doordat containers standaard altijd runnen als 'root' heeft de container als de '/' directory gemount is op deze container ongelimiteerde toegang tot het volledige host systeem. Hierdoor kunnen zonder restricties aanpassingen aan het hostsysteem gedaan worden. Dit betekent dat een gebruiker die toegang heeft tot de docker daemon verhoogde rechten kan verkrijgen door gewoon een container op te starten.

Zoals bij een virtuele machine waarop de services rechtstreeks wordt ook bij een

systeem waar docker containers op draaien aangeraden overbodige services uit te schakelen of te verwijderen. Moest een gebruiker door een van deze services toegang krijgen tot het hostsysteem. Zou hij makkelijker 'root' rechten kunnen vergrijpen volgens de manier beschreven in de vorige paragraaf.

### 4.1.3 Auditen van het systeem

## 4.2 Docker daemon configuratie

Bij het beveiligen van de docker stack is ook de configuratie van de docker daemon een belangrijk element. Dit heeft twee grote onderdelen. Enerzijds de configuratie van de docker daemon zelf, dit houdt in dat we de daemon zelf zo gaan configureren dat deze zo veilig en robuust mogelijk wordt. Aan de andere kant moeten we ook zorgen dat de configuratie files zelf de juiste file permissies en owners hebben. Hierbij is de vergelijking met een virtuele machine, waar de services rechtstreeks op draaien, moeilijk aangezien er hier geen docker daemon op te vinden is. Omdat het configureren van de Docker daemon dingen verandert die voor elke container toegepast worden kunnen we het wel vergelijken met hoe de individuele services globaal geconfigureerd worden.

### 4.2.1 Globale configuratie

Oorspronkelijk was (zoals uitgelgd in 3.1 Geschiedenis van Containers) Docker gemaakt als een interne tool voor dotCloud om het gebruik van LXC (LinuX Containers) te vergemakkelijken. Docker kan nu (op het moment van het schrijven van deze Bachelorproef) nog gebruik maken van de LXC Dirver om containers uit te voeren. Al is er ondersteuning voor deze uitvoering met LXC wordt dit niet aangeraden. Zonder expliciet mee te geven aan de Docker daemon dat LXC als 'execution driver' moet gebruikt worden wordt libcontainer van Docker zelf gebruikt. Hier ligt ook de verdere ontwikkeling en ondersteuning voor onder andere meer geavanceerde netwerking, firewall en beveiliging.

### 4.2.2 netwerk configuratie

Docker heeft om netwerk misconfiguratie te vermijden toegang nodig to de iptables. Dit wordt gebruikt voor het opzetten, onderhouden en inspecteren van filter regels

voor IP pakketten in de linux kernel Eychenne (2016). Deze permissie is standaard toegewezen aan Docker. Naast de permissies voor de iptables is er ook standaard communicatie toegestaan tussen de containers onderling zelfs zonder containers te linken met `'-link=CONTAINERNAME'`. Dit wordt aangeraden om dit af te zetten door de docker daemon op te starten met de flag `'-icc=false'`. Indien we dit niet doen kan er ongewenste communicatie zijn tussen twee containers. Containers kunnen na het toevoegen van de inter container communication flag nog altijd communiceren met elkaar mits het expliciet linken van de gewenste containers met `'-link=CONTAINERNAME'`.

### 4.2.3 Permissies voor Docker config files

Het configureren van de Docker daemon met de juiste instellingen voor zowel netwerking, registry gebruik en algemene configuratie heeft enkel nut als we ook de configuratie files waarin deze instellingen worden bewaard correct beveiligd zijn. Standaard worden de bestanden die Docker gebruikt pas aangemaakt wanneer de daemon ze nodig heeft. Deze worden ook aangemaakt met de correcte permissies. De `'/etc/docker'` folder waarin alle files in verband met containers en images onder andere worden in opgeslagen. Deze is standaard beveiligd met toegewezen gebruikers groep en eigenaar op `'root'` en permissies hiervoor lezen, schrijven en uitvoeren voor de eigenaar en enkel lees en uitvoerrechten (dus ook niet verwijderen) voor de niet-eigenaars Shotts (2016).

## 4.3 Security of Docker images

Als we het hebben over de beveiliging van Docker images zijn er twee grote categorieën die we moeten onderscheiden. Een eerste categorie is wanneer we werken met Docker images die afkomstig zijn van de officiële Docker Hub Registry, een repository waar iedereen zijn images op kan posten en kan distribueren. Daarnaast hebben we ook de mogelijkheid om onze eigen images te maken.

### 4.3.1 Images van de Docker hub

Wanneer we Docker images gebruiken van de officiële Docker Hub Registry is een handig hulpmiddel het commando `'docker search zoekterm'`. Dit geeft ons een lijst van alle docker images die voldoen aan deze zoekterm. Een voorbeeld hiervan vindt u hieronder.

\$ docker search busybox

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
busybox	Busybox base image.	316	[OK]	
progrium/busybox		50		[OK]
radial/busyboxplus	Full-chain, Internet enabled, busybox made...	8		[OK]
odise/busybox-python		2		[OK]

Een van de dingen die hierbij opvallen is de kolom 'OFFICIAL'. Een image die als official is aangeduid, is door het approval proces van docker zelf gegaan. Dit houdt onder andere in dat de images onderhouden worden, bij herhaaldelijk bouwen altijd hetzelfde resultaat bekomen wordt, consistent maar toch duidelijk zijn en goed bevijld zijn. Meer hierover in het deel over eigen docker images beveiligen.

De officiële images zijn in het algemeen veiliger om te gebruiken dan niet officiële images. Dit wil niet zeggen dat niet officiële images niet veilig kunnen zijn, maar deze zijn niet gecontroleerd geweest door docker zelf. Doordat we bij het downloaden van een Docker image niet direct kunnen zien wat deze zal doen wordt het dus sterk afgeraden om Docker images van niet vertrouwlijke bronnen te gebruiken.

Van elke image die gevonden wordt met 'docker search' en dus op de officiële repositories staat (ervan uitgaand dat er manueel geen andere repositories zijn toegevoegd) is online op de docker hub webpagina een Dockerfile te vinden. Met deze Dockerfile kunnen we ook zelf kijken hoe de image precies is opgebouwd. Dit kan wel snel een tijdrovend proces worden aangezien docker images kunnen gebaseerd zijn op andere images. Hierdoor kan voorkomen dat er meer dan een paar Dockerfiles moeten bekeken worden voordat men weet wat er precies allemaal inbegrepen zit in de onderzochte image.

Een ander belangrijk element dat met de officiële docker images probeert opgelost te worden is dat iedereen zijn eigen manier heeft om de dockerfiles op te bouwen en manieren van software te installeren. Door een methodiek aan te bieden die gevolgd moet worden voordat de image kan opgenomen worden in de officiële docker hub zorgen ze voor een gelijkaardige opbouw van Dockerfiles bij officiële docker images.

De vergelijking met een virtuele machine zonder docker kan gemaakt worden door de officiële docker hub te vergelijken met een package archive. In beide gevallen kan zowel van de officiële repositories afgehaald worden als zelf niet officiële repositories toegevoegd. Het verschil ligt dan uiteraard in de aangeboden software, Docker images t.o.v. packages.



### 4.3.2 zelf Docker Images maken

Als we zelf Docker images willen maken is goede referentie om aan te voldoen de eisen die gesteld worden voor officiële docker images. Uiteraard zijn er hier elementen die voor eigen gebruik niet uiterst noodzakelijk zijn maar deze vormen een goede basis om van te vertrekken.

Als eerste belangrijk punt moeten we rekening houden met welke packages we willen gebruiken in onze images. Hierbij zijn dezelfde risico's aan verbonden als moesten we een gewone virtuele machine gebruiken. Enkel packages installeren die gebruikt worden en het goed configureren (indien nodig) van packages is hier even belangrijk als bij een gewone virtuele machine. Dus bij het installeren van packages is het aangewezen om de fingerprint te controleren.

Bij het maken van onze docker images kunnen we ook files kopiëren in onze container. Dit wordt best zo veel mogelijk vermeden maar indien toch vereist is het best om de file specifiek te kopiëren en niet een hele map. Bij het copieren van een volledige map kunnen bij herbouwen ongewenste files in de map bijgekomen zijn waardoor er een onverwacht of ongewenst resultaat kan bekomen worden.

Een belangrijk element om bij stil te staan als de gemaakte Docker image robuust wenst gemaakt te worden is het definiëren van vaste package versies. Hierdoor blijven onze images bij het herbouwen van dezelfde versie altijd hetzelfde. Extra beveiliging kan hierdoor verkregen worden, door het gebruiken van specifieke versies van packages weten we ook altijd exact welke versies van deze packages er in onze image gebruikt worden. Een nadeel hieraan is dat bij het uitkomen van nieuwe updates en eventuele security updates en bugfixes we onze docker images moeten aanpassen. In het algemeen wordt aangeraden om voorgedefinieerde versies te gebruiken, nieuwe versies van software worden best eerst uitvoerig getest vooraleer in gebruik genomen te worden. Door het definiëren van de versie dat we willen gebruiken hebben we hierover volledige controle.

Het maken van onze eigen Docker images kunnen we dus volledig vergelijken aan het configureren van een virtuele machine. Hierbij moet er ook gekeken worden naar welke files we willen gebruiken op deze machine alsook welke packages er op geïnstalleerd moeten staan. Dezelfde veiligheids voorschriften die gebruikt worden bij het configureren van een virtuele machine zijn dus ook van toepassing op het maken van een eigen Docker image.

## 4.4 Container configuration

Als we Docker containers gebruiken moeten de containers zelf ook goed geconfigureerd zijn. Een groot deel van de veiligheid van de containers zelf hangt af van hoe we ons 'docker run' commando gebruiken. Daarnaast zijn er ook nog een aantal punten waarvoor we moeten oppassen of kunnen optimaliseren op de container zelf.

### 4.4.1 docker run commando configureren

Elke container word opgestart met het commando 'docker run [OPTIONS] IMAGE [COMMAND] [ARG...]'. Met dit commando kunnen aan de hand van onze zelfgemaakte Docker image of een gedownloade Docker image een container aanmaken en direct opstarten. Doordat dit een van de belangrijkste commando's is die bij Docker hoort is het uiteraard ook een van de krachtigste, dus moeten we wel beter enkele dingen in gedachten houden wanneer we het uitvoeren.

Bij het uitvoeren van het standaard 'docker run' commando namelijk 'docker run IMAGE' zijn er veel dingen die standaard in orde zijn. Uiteraard zijn er dingen waar we normaal geen problemen mee mogen hebben maar soms toch problemen kunnen veroorzaken.

Een van deze dingen zijn de standaard Linux Kernel Capabilities die de container heeft. Over Linux Kernel Capabilities zou kunnen een Bachelorproef op zich geschreven worden en gaat valt hierdoor niet binnen de scope van deze Bachelor proef. Maar sterk vereenvoudigd is dit de basis instructieset die een linux besturingssysteem bezit, deze instructies zijn als gewone gebruiker niet allemaal toegankelijk maar de root user kan elke instructie uitvoeren. Docker heeft uit voorzorg hiervoor deze lijst met Capabilities al sterk verminderd in vergelijking met de gewone Linux Kernel Capabilities, dit zorgt ervoor dat de root user in een container veel minder kan dan een root user op een gewoone machine. De meeste use cases hebben containers ook geen echte root rechten nodig. Hiervoor kunnen we dus de Linux Kernel Capabilities per container gaan tweak en enkel privileges toewijzen aan de containers die ze echt nodig hebben. Om deze beperking te omzeilen kunnen Docker containers ook met de optie '-privileged' opstarten. Dit geeft de container alle Kernel Capabilities die het hostsysteem heeft en wordt dus best buiten in enkele zeer specifieke use cases vermeden.

Naast de mogelijkheid om de standaard Kernel Capabilities aan te passen van de containers hebben we ook de mogelijkheid om het processor en geheugen gebruik te limiteren. Docker containers kunnen zonder verdere configuratie het complete geheugen

gebruiken van de host, hierdoor kunnen andere containers zonder geheugen geraken en dit kan voor problemen zorgen. De mogelijkheid een container het complete geheugen van het hostsysteem in beslag neemt is zeker iets waar rekening mee gehouden moet worden. Hiervoor gebruiken we de optie `'-m'`. Hiermee kunnen de maximale hoeveelheid geheugen die een container mag gebruiken bepalen. Een optie om de minimale hoeveelheid geheugen voor containers met een kritieke rol vast te leggen ontbreekt daarbij heelaas. Hierdoor kunnen deze kritieke containers ook zonder geheugen geraken waardoor de werking van ons systeem in gedrang komt. Een mogelijkheid om dit te vermijden zou kunnen zijn aan elke container een bepaalde hoeveelheid geheugen toe wijzen waardoor als we de som maken van alle geheugengebruik we lager blijven dan de hoeveelheid geheugen van de host. Maar dan we moeten we elke keer als we een nieuwe container toevoegen elke container opnieuw opstarten met een andere hoeveelheid geheugen toegewezen. Hierdoor verliezen we onze flexibiliteit die we net verkrijgen door docker te gebruiken en dit is dus niet aan te raden. Daarnaast hebben we ook CPU gebruik. Elke container gebruikt standaard een gelijke hoeveelheid van de cpu. De optie die gebruikt wordt om het cpu gebruik te bepalen is de `'-c'` of de `cpu-shares` optie. Standaard heeft deze een waarde van 1024, als alle containers deze waarde hebben, krijgen ze allemaal gelijke prioriteit op de cpu. Door deze waarde te verhogen of verlagen kunnen we desgewenst een hogere of lagere prioriteit verkrijgen voor onze container.

Het is maar op het moment dat we dit standaard `run` commando beginnen aanpassen met opties dat er enkele dingen zijn waarvoor we best opletten. Wanneer een `docker run` commando wordt uitgevoerd zijn er twee dingen die we in opties kunnen toevoegen waar we toch iets voorzichtiger mee moeten zijn. Namelijk netwerk configuratie en mounts van host directories in de container.

Netwerk configuratie van docker is als er niets aan is aangepast tamelijk robuust. Maar als we hierin dingen beginnen aanpassen is het mogelijk om deze robuustheid wat te ondermijnen. Een van de elementen hierbij is poorten. Bij het connecteren naar buiten de machine waarop een container opereert moet een zowel een poort op de host als op de container worden opengesteld, waarna deze poorten "verbonden worden" waardoor het netwerkverkeer die op de gedefinieerde poort van de host toekomt wordt doorgezonden naar de gekozen poort op de container. Een best practice hierbij is om altijd de volledige connectie te definiëren, dit wil zeggen dat we zowel de externe interface, de externe host poort als de interne container poort meegeven met ons `run` commando. Hierbij worden op de container best enkel poorten geopend die effectief gebruikt worden door deze container. Dit verkleint het aanvalsvlak via netwerk op deze container. TCP/IP poorten onder 1024 op de host machine worden best ook vermeden om op te binden. Uiteraard zijn er services die hier op moeten binden zoals een http proxy moet luisteren op poort 80 om te kunnen functioneren

en zijn ook veiliger als ze hierop luisteren. Een goede maatstaf om te weten als een container op een privileged port gebonden moet worden is kijken indien men de service die in de container draait ook wanneer die op een virtuele machine draait op een poort onder 1024 zou binden. Daarnaast hebben we net zoals bij Kernel Capabilities ook bij netwerking ook een manier om het intern Docker netwerk te omzeilen. Een container gebruikt standaard een bridged netwerk, hierdoor wordt deze container in een aparte netwerk stack gestoken. Als we de optie `–net=host` meegeven overriden we dit en geven we de container toegang tot alle netwerk interfaces van de host machine. Dit moet tenzij in zeer specifieke toepassingen niet gebrijkt worden.

### 4.4.2 container configuratie

Om extra beveiliging te voorzien voor de docker containers is het mogelijk (indien ondersteund door de host OS) om AppArmor en SELinux te gebruiken. Met AppArmor is het mogelijk om voor elke container apart een AppArmor profiel te maken. SELinux voor docker moet aangezet worden bij het opstarten van de Docker Daemon, dit kan simpel gedaan worden door de Daemon bij het starten de optie `–selinux-enabled` mee te geven. Hierna kunnen we SELinux security opties meegeven aan containers. Zowel AppArmor en SELinux zorgen ervoor dat we elke container individueel kunnen configureren op vlak van security. Het individueel configureren van SELinux en AppArmor voor containers kan vergeleken worden met het configureren voor processen op een niet-Docker systeem. Dit is buiten de scope van deze Bachelorproef en hierom gaan we hier niet verder op in.

Daarnaast is een element die we moeten bekijken wat we nu precies in een container willen draaien qua processen. Het idee achter docker is om één enkele applicatie per container op te delen. Docker luistert standaard enkel op één hoofdprocess, dus indien we meerdere processen willen uitvoeren op 1 container die niet gemaakt zijn om vanuit 1 hoofdprocess op te starten en informatie terug door te geven via dit hoofd process moeten we een process manager gaan bijvoegen in de container. Dit zorgt er onder andere voor dat een container die algemeen gezien een simpel iets is weer zeer complex wordt. Anderzijds zorgt dit ervoor dat je de processen binnen de container niet meer op een optimale manier kunt monitoren met docker. Een voorbeeld hiervan zou zijn indien we zowel onze applicatie als de database op eenzelfde container plaatsen. Indien we deze configuratie zouden gebruiken merken we al snel dat het heel wat makkelijker zou zijn op elke vlak om gewoon de database uit de applicatie container te halen en in zijn eigen container op te starten, dit heeft ons naast een robuuster systeem (als de applicatie crasht zou het kunnen dat dit de database mee doet crashen indien het op dezelfde container draait) ook een meer schaalbaar systeem door gewoon

meer applicatie containers te linken met dezelfde database container. Een ander voorbeeld is het gebruik van SSH, container hebben in de meeste gevallen geen ssh nodig. Alle containers zijn toegankelijk via het hostsysteem met het commando 'docker exec CONTAINER' dus is daar de meest aangewezen plaats om SSH te installeren.

# Hoofdstuk 5

## Manageability

Wanneer we de manageability van Docker containers gaan bekijken hebben we drie opties, in de eerste plaats kunnen we alles manueel managen. Daarnaast kunnen we ook alles automatisch laten managen door een proces manager. Ofwel kunnen we tools gebruiken om onze containers te managen. Als we bezig zijn met het managen van Docker containers moeten we ook rekening houden met het managen van onze docker images.

### 5.1 Manueel managen

Een optie waarmee de meesten beginnen is het manueel managen van onze Docker containers. Bij Deze optie gaan we Manueel onze docker containers gaan uitvoeren of met behulp van een kort script of een makefile. Op deze manier kunnen we zowel onze containers managen als onze Docker images builden. Op deze manier onze containers managen is doenbaar indien we enkele containers moeten managen zoals in een development omgeving of een stabiele kleine omgeving. Het manueel managen van onze containers verwacht ook meer stabiliteit van onze containers. Indien er niemand de containers aan het monitoren zou zijn wanneer een container crasht, Moet er gewacht worden tot er iemand deze weer herstart. Hierdoor kan het dus gebeuren dat een container relatief langer onbeschikbaar is. In een development omgeving zijn de vereisten voor uptime veel lager. Hier is het manueel managen van onze containers en images wel een optie. Bij het manueel managen wordt er ook gebruik gemaakt van scripts of makefiles. Een voorbeeld van beide vind u hieronder.

```
1 CONTAINER ?= redis
```

```
2 IMAGE_NAME ?= redis
3 IMAGE_VERSION ?= latest
4 HOSTNAME ?= redis-host
5
6
7 # full image name
8 IMAGE = $(IMAGE_NAME):$(IMAGE_VERSION)
9 # full ssh connect vars
10 CONNECT = $(REMOTE_USER_NAME)@$(IP)
11
12 build:
13     docker build -t $(IMAGE) .
14
15 run:
16     docker run -d --name $(CONTAINER) -h $(HOSTNAME) $(IMAGE)
17
18 logs:
19     docker logs $(CONTAINER)
20
21 kill_container:
22     docker kill $(CONTAINER)
23
24 remove_container:
25     docker rm $(CONTAINER)
26
27 remove_image:
28     docker rmi $(IMAGE)
29
30 update: kill_container remove_container remove_image build run
```

In deze makefile hebben we toegang tot de verschillende commando's die we normaal volledig met de hand zouden uitschrijven maar nu kunnen we deze aanroepen met 'make COMMANDO'. Als we dit bekijken met de voorgaande makefile zien we dat we voor onze redis container uit te voeren enkel 'make run' moeten uitvoeren terwijl we anders 'docker run --name redis redis' moesten uitvoeren. Bij dit voorbeeld is het commando dat we zonder makefile zouden moeten uitvoeren nog beperkt maar als we meer opties meegeven met ons commando wordt het interessanter voor ons om een kort make commando uit te voeren dan elke keer dat we dezelfde container willen starten opnieuw het complete commando uit te typen. Ook behouden we onze consistentie bij het uitvoeren van de commando's en elimineren we eventuele menselijke fouten zoals typfouten. Het voordeel dat we hier hebben is dat we make commando's ook kunnen aanroepen in andere make commando's hierdoor kunnen we de het commando 'make update' maken. Deze verwijdt onze oude bestaande container en image. Daarna wordt een nieuwe image gemaakt en deze ingeladen in een nieuwe container. Bovenaan de makefile vinden we ook variabelen die gebruikt kunnen worden in onze make

commando's. Met behulp van deze variabelen kunnen we een makefile snel aanpassen voor het gebruik met een ander commando.

Bij het gebruik van scripts verkrijgen we ongeveer dezelfde functionaliteit als bij makefiles, enkel is de syntaks voor het uitvoeren anders en de manier waarop het geschreven is. Hierboven vinden we een script die dezelfde functionaliteit zou hebben als het 'make update' commando.

## 5.2 Managen met process manager

Een andere optie die we hebben is onze containers managen met een process manager. Hierbij kunnen we onze containers gaan bekijken alsof het processen zijn. Als we onze containers managen met een process manager gaan we zelf configuratie bestanden gaan schrijven voor de gekozen process manager. Die gekozen manager zal dan deze bestanden inlezen en vervolgens de met de parameters vanuit de configuratie bestanden onze containers starten en managen zoals het een ander process zou beheren. Een voorbeeld hiervan is het gebruik van Systemd Unit files. Dit vinden we hieronder uitgewerkt voor de Redis Container uit hoofdstuk drie.

```
1 [Unit]
2 Description=Run a redis container
3 Author=Toon Lamberigts & Tomas Vercautter
4 Requires=docker.service
5
6 [Service]
7 ExecStartPre=/usr/bin/docker rm redis-container
8 ExecStart=/usr/bin/docker run --rm --name redis-container -h redis-
  host redis
9 ExecStop=/usr/bin/docker stop -t 2 redis
```

Deze unit file zal bij het proberen starten van de bij behorende service ingeladen worden. Het 'ExecStartPre' gedeelte wordt uitgevoerd voor de service effectief zal starten. Deze zal de container verwijderen indien deze bestaat, dit is om zeker te zijn dat er geen container genaamd 'redis-container' bestaat. Dit zou een fout genereren bij het opstarten van waardoor de service zou falen. Dus uit voorzorg proberen we te verwijderen voor het opstarten. Het onderdeel 'ExecStart' is het effectieve commando die de service zal uitvoeren en zich aan zal binden. In dit geval is dat onze redis container. We geven bij het opstarten via unit files geen flag '-d', detached mee omdat we willen dat de container zich bind aan de service waardoor we de container kunnen monitoren via de aangemaakte service. Hiervoor geven we ook de '-rm' tag mee, deze zorgt ervoor dat bij het stoppen van de container hij ook direct verwijderd wordt. Dit



gebeurt jammer genoeg enkel bij een correcte terminatie van de container, dus indien de docker service crasht zal deze niet verwijderd worden maar wel gestopt.

## 5.3 Managen met tools

Een derde optie die we hebben is het managen met tools. In dit onderdeel zal ik niet uitgebreid beschrijven hoe elke tool werkt en hoe je die moet gebruiken, dit moet geval per geval bekeken worden als dit een meerwaarde betekend voor het project met docker. Ik zal hier proberen een voorbeeld te geven hoe men tools zou kunnen gebruiken in een docker omgeving. De tool waarmee ik wat meer uitleg zal geven is vagrant. Deze tool is enkel een voorbeeld en is zeker niet de enige tool hiervoor. Vagrant een tool ontwikkeld voor het gebruik in development en wordt door hashicorp ontwikkeld. Daarnaast hebben ze ook Nomad, een nieuwe tool die in moet staan voor de deployment van containers. Andere bedrijven hebben andere tools en zoals hierboven vermeld moet voor elke use case gekeken worden hoe tools eventueel kunnen geïntegreerd worden in het ontwikkelproces met docker. Het voorbeeld van Vagrant dient om een idee de vormen van hoe tools gebruikt kunnen worden. Docker bied zelf ook een tool aan voor het managen van docker containers namelijk Docker Compost. Een voorbeeld van een configuratiefile voor Vagrant en Docker Compose worden hieronder weergegeven. Beide zetten een docker container genaamd redis op van een image genaamd redis.

```
1 Vagrant.configure("2") do |config|
2   config.vm.provision "docker" do |d|
3     d.image = "redis"
4     d.name = "redis"
5   end
6 end
```

Dit is een voorbeeld van een Vagrant file voor docker. Deze kan opgestart worden met het commando 'vagrant up'

```
1 #redis image
2 redis:
3   image: redis
```

Dit is een voorbeeld van een Docker-compose file voor docker. Deze kan opgestart worden met het commando 'docker-compose up'

# Hoofdstuk 6

## Logging en monitoring

We kunnen onze Docker containers zo goed beveiligen en managen als we willen, het is bijna nooit de vraag zal er iets verkeerd gaan? Maar eerder de vraag wanneer iets verkeerd zal gaan. Om te kunnen weten wanneer er iets verkeerd gaat en wat er precies verkeerd gaat is het monitoren van onze containers en loggen van wat er gebeurd zeer belangrijk. Daarom heb ik besloten van kort om kort uit te leggen wat een mogelijke manier is om onze containers te monitoren en hoe we onze containers kunnen loggen. De tools die ik hier zal uitleggen zijn zeker niet de enige die hiervoor gebruikt kunnen worden. De focus ligt hier eerder op wat te monitoren en loggen en hoe we dit persistent kunnen doen dan op hoe we dit doen. We zouden kunnen zelf alles opzetten om te monitoren en loggen maar waarom zoveel tijd steken in hoe alles te loggen terwijl we kunnen bezig zijn met het belangrijker aspect namelijk wat te monitoren en loggen

Patrick Debois.

Bij het loggen en monitoren van onze containers is er niet zo veel verandert in vergelijking met het monitoren en loggen van een virtuele machine. We hebben nog altijd zowel de optie om alles zelf te configureren of een tool installeren die het voor ons doet. Hierbij moet er enkel rekening gehouden worden met het feit dat er nu op de machine waar de containers op staan evenveel logs worden geproduceerd als bij een aanpak zonder containers op verschillende virtuele machines samen worden geproduceerd. Bij monitoren van containers is het belangrijk dat zowel het hostsysteem wordt gemonitord)als de services of software in de containers. Waarbij we bij het monitoren van het hostsysteem nog altijd de zelfde elementen monitoren als bij een gewone virtuele machine maar nu ook docker specifieke

elementen moeten bijhouden, zoals onder andere traffic tussen containers en welke containers zich bevinden op dat hostsysteem.

## **Hoofdstuk 7**

### **Conclusie**

# Bibliografie

(1996). Unix - Frequently Asked Questions (6/7) [Frequent posting]Section - A very brief look at Unix history.

Anderson, C. (2015). Docker [Software engineering]. *IEEE Software*, 32(3):102–c3.

Brockmeier, J. (2014). Is It Safe? A Look at Docker and Security from LinuxCon.

Docker (2015). Introduction to Container Security. Technical report.

Docker (2016a). Automatically start containers.

Docker (2016b). build.

Docker (2016c). docker realeases.

Docker (2016d). Docker run.

Docker (2016e). Docker security.

Docker (2016f). Find & run the whalesay image.

Docker (2016g). Install Docker.

Docker (2016h). Official Repositories on Docker Hub.

Docker (2016i). Understand images & containers.

Docker (2016j). What is Docker?

Ellingwood, J. (2015). The Docker Ecosystem: An Overview of Containerization.

Ewindisch (2014). On the Security of Containers.

- Eychenne, H. (2016). iptables(8) - Linux man page.
- FreeBSD (2012). chroot(2).
- Gunaratne, I. (2016). Evolution of Linux Containers and Future.
- Hildred, T. (2015). The History of Containers.
- Jennings, C. (2009). VMware: five biggest challenges of server virtualisation.
- Jessie Frazelle (2016). Docker Engine 1.10 Security Improvements.
- Johnston, S. (2014). Docker 1.3: signed images, process injection, security options, Mac shared directories.
- Kerrisk, M. (2016a). capabilities(7) - Linux manual page.
- Kerrisk, M. (2016b). chroot(2) - Linux manual page.
- Lenny Zeltser (2015). Security Risks and Benefits of Docker Application Containers.
- Mónica, D. (2015). Understanding Docker Security and Best Practices.
- Mouat, A. (2015a). *Docker Security*.
- Mouat, A. (2015b). Running Docker Containers with Systemd.
- Petazzoni, J. (2013). Containers & Docker: How Secure Are They?
- Pravin Goyal (2015). CIS Docker 1.6 Benchmark v1.0.0. Technical report.
- Rudenberg, J. (2014). Docker Image Insecurity.
- Shotts, W. E. (2016). Learning the shell - Lesson 7: Permissions.
- Thijssen, J. (2012). Why putting SSH on another port than 22 is bad idea.
- Toni de la Fuente (2015). Docker Security Tools: Audit and Vulnerability Assessment.
- Voelker, A. (2015). Why I don't use Docker much anymore - Abe Voelker.

## Lijst van figuren

3.1	Virtualisatie bij Virtuele machines- Docker (2016j) . . . .	10
3.2	Virtualisatie bij Docker Containers - Docker (2016j) . . .	10
3.3	Zoeken naar images op de docker hub site . . . . .	12
3.4	Zoeken naar images via command-line . . . . .	12
3.5	Output van het docker images commando . . . . .	15
3.6	Output van het docker ps commando . . . . .	16

## **Lijst van tabellen**