# Workshop Docker Introductie (01)

Hans Feringa: info@linuxadviesburo.nl

## Inhoudsopgave

# Inleiding

In deze workshop gaan we een aantal basis elementen van Docker leren kennen. In de eerste plaats gaan we leren hoe we een image kunnen gaan bouwen die we "stand-alone" kunnen inzetten. Verder leren we hoe we images kunnen beheren en hoe we hiervan containers kunnen "runnen".

Verder gaan we in deze workshop proberen een aantal concepten van Docker uit te werken, waarmee we multi-container applicaties kunnen bouwen. Het gaat hierbij ondermeer om volume's en het linken van containers. We gaan nieuwe images bouwen en daarmee voorbeeld-webapplicaties bouwen a la "Hello World". Dus heel simpel.

De workshop gaat uit van een CentOS 7 VM met voldoende storage (disk ong 10 G). Je kunt hiervoor KVM gebruiken op Linux, of Virtual Box op Windows, OS/X en Linux. Hierop moet je op de avond zelf al een docker hebben draaien. Indien je gebruik maakt van een fysieke machine als docker host, dan zullen een aantal handelingen/configuraties simpeler zijn.

De workshop is grotendeels gebaseerd op "The Dockerbook" van James Turnbull. De electronische versie is niet duur en is een aanrader (http://www.dockerbook.com/). Je kunt natuurlijk ook de dode boom versie kopen. De voorbeelden uit het boek kun je uit github "clonen" (zie later voor het git clone commando).

In principe werken we na de "opwarmer" de opdrachten 1 en 2 tijdens de workshop uit. Opdracht 3 alleen als er voldoende tijd over is.  Opdracht 3 maakt in principe gebruik van de resultaten van de twee eerdere opdrachten en breidt deze uit met een loadbalancer (HAproxy).

# SELinux

***In een aantal gevallen zul de SELinux context van directories/bestanden op je docker host moeten aanpassen om e.e.a werk te krijgen.***

# Voorbereidingen

## *Leeswijzer*

Commando's met een # prompt worden als root op de VM(dockerhost) uitgevoerd.

## *CentOS7 VM*

Installeer een VM met CentOS7 met voldoende diskruimte (>= 10G). Geef de VM 4GB aan geheugen. Wellicht dat minder ook volstaat. Ik heb getest met 4GB

Zorg ook dat je op je VM docker hebt geinstalleerd en dat het opstart tijdens boot (zie onder).
Je moet vanuit je VM kunnen communiceren met het internet.

Deze VM is je "**Docker Host**".

## VM voorbereidingen

Log in als root op je dockerhost (via ssh of via de console).

Update het systeem met de laatste packages en installeer git en wget:

```
# yum -y update
# yum -y install git wget
```

Indien nodig herstart het systeem:

```
# systemctl reboot
```

## Installeren en activeren docker

```
# useradd docker
# yum -y install docker
# systemctl enable docker
# systemctl restart docker
```

## Firewall/Docker issues

***Op dit moment heb ik een reproduceerbaar probleem, waarbij de firewalld tijdens boot niet wil starten (error INVALID_ZONE) als de docker daemon enabled is. Je kunt vervolgens niet met SSH inloggen in je VM (de dockerhost). Via de console kan firewalld alsnog worden opgestart en is de VM weer via SSH bereikbaar. Het advies is dus om docker niet te "enablen", maar na systeemstart handmatig te starten!***

***Een ander probleem is, dat docker altijd (opnieuw) gestart moet worden nadat firewalld (opnieuw) is gestart of een firewall-cmd --reload is gedaan. E.e.a lijkt samen te hangen met de hieronder beschreven firewall rules. Het effect is dat networking in de containers niet (goed) werkt (o.m tijdens build).***

***Dus voorlopig na elke firewall-cmd --reload of een systemctl restart firewalld, ook een systemctl restart docker!***

***Geconstateerd bij software versies (CentOS 7.1.1503):***
***firewalld: 0.3.9-11***

## Creeer een reguliere gebruiker (duser01)

Deze gebruiker moet lid zijn van de group `wheel` (voor `sudo`) en van de group `docker` (voor docker commando's).

```
# useradd duser01
# passwd duser01
# usermod -a -G docker,wheel duser01
```

## Inloggen

Login in je VM als **duser01**. Dit is de user waarmee je de workshop opdrachten uitvoert. Overigens mag je dat ook als root doen. In dat laatste geval hoef je uiteraard geen sudo te gebruiken. Ook als je user `duser01` lid is van de group `docker`, dan hoef je geen sudo te gebruiken.

Zet het ip adres van je docker host in je `/etc/hosts` file (of het equivalent daarvan op Windows of OS/X):

```
192.168.122.110  dockerhost
```

Vervolgens kun je telkens '**dockerhost**' gebruiken ipv het ip adres:

```
$ ssh duser01@dockerhost
```

Inloggen op je Docker host (VM). Je kunt dit het beste doen via een ssh verbinding ipv de console. Je zult de firewall rules in de dockerhost een beetje moeten aanpassen, om vanaf je laptop een connectie naar bijvoorbeeld een webserver te kunnen maken. Standaard is de SSH port (22) wel open.

Voer de volgende reeks commando's als **root** (of gebruik sudo) uit in je **dockerhost**:

```
# firewall-cmd --permanent --direct --add-rule ipv4 filter INPUT 0 \
  -s 192.168.122.0/24 -j ACCEPT
# firewall-cmd --reload
# systemctl restart docker
```

Of

```
# firewall-cmd --zone trusted --add-source 192.168.122.0/24 --permanent
# firewall-cmd --reload
# systemctl restart docker
```

Vervang het ip netwerk adres door het adres wat op jouw laptop gebruikt wordt voor je VM's. Bovenstaande is typisch een kvm default gebruikt (gesloten) netwerk voor geNATte VM's. Als je bridging gebruikt wil je misschien wat restrictiever zijn en niet het hele (V)LAN toelaten.

## Ophalen van de sources van "The Dockerbook"

```
$ git clone https://github.com/jamtur01/dockerbook-code
```

***Deze tree gaan we in deze workshop NIET gebruiken. Deze is hier alleen gegeven voor referentie. Voor de workshop is een aantal wijzigingen in de originele files aangebracht. Verder bevat de tree uit git alle code uit het dockerbook en is dus veel omvangrijker.***

## Creeeren werk directory

Creeer een nieuwe directory van waaruit we gaan werken. **Dit is altijd het startpunt voor de opdrachten.**

```
$ mkdir workshop-01
$ cd  workshop-01
```

## Download de tarball:

```
$ wget http://www.shelob.nl/itgilde/docker-workshop-01.tgz
```

De inhoud van deze tarball is gebruikt om de workshop voor te bereiden. Dee instructies voor het creeeren van directories en bestanden komen daardoor te vervallen omdat deze reeds na het uitpakken van de tar aanwezig zijn.

## Uitpakken tarball in onze werkdirectory

Pak in deze directory de voorbereide tarball uit:

```
$ tar xfz docker-workshop01.tgz
```

Als je deze tarball hebt uitgepakt, dan heb je in principe alle bestanden die in deze workshop worden gebruikt.

## Extra bestanden

In de directory `workshop-files` vindt je ondermeer de volgende bestanden:

```
workshop-files/
```

```
├── div-commands
├── haproxy.cfg
├── into-container
├── logstash.conf
├── server.js
├── start-docker-containers-voor-workshop
└── stop-docker-containers-voor-workshop
```

Mijn ervaring is dat er nogal wat dingen fout gaan met knippen en plakken vanuit een PDF document. Ik heb daardoor ondermeer de Dockerfile(s) in hun respectievelijke directories en andere configuraties en commando's beschikbaar gemaakt (directory workshop-files) in de tarfile. Dit scheelt een hoop frustratie vanwege geintroduceerde spaties, veranderde - (dashes) en andere verfraaiingen die met PDF documenten worden geintroduceerd.

# Opwarmer 1: Run een container direct vanuit de registry

Het starten van een container is zo simpel als het volgende commando:

```
$ docker run -ti centos:latest /bin/bash
[root@548bb95035f5 /]# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  9 19:58 ?        00:00:00 /bin/bash
root        18     1  0 19:58 ?        00:00:00 ps -ef
```

De image wordt opgehaald uit de repository van de docker registry(docker hub) en daarvan wordt een container gestart.

In de container hebben we (interactief) het `ps -ef` commando uitgevoerd. Er is maar 1 process actief in de container (+ het process ps dat we interactief hebben opgestart). Opvallend genoeg is het PID van het bash process 1. Er is geen init of systemd.

Het argument /bin/bash, samen met de flags -t (voorzie een pseudo terminal) en -i (interactieve sessie), zorgen dat we een interactieve (bash) sessie hebben in de container.

We zullen later zien dat dit soort simpele containers heel nuttig kunnen zijn om eenmalige acties uit te voeren zoals backups, of het bekijken van interne log files in een andere container (via bijvoorbeeld volumes). In de regel worden deze containers dan als "wegwerp" containers gebruikt (flag --rm), die na het uitvoeren direct weer worden verwijderd uit de cache.

De container kun je met een exit of CTRL-D weer beeindigen. Weggooien doe je met het commando:

```
$ docker rm <container-id>
```

Het container id kun je met het docker ps commando opvragen:

```
$ docker ps -l
CONTAINER ID     IMAGE               COMMAND             CREATED
STATUS                       PORTS               NAMES
548bb95035f5        centos:latest       "/bin/bash"         5 minutes ago
Exited (0) 15 seconds ago                        sad_galileo
```

Het container-id staat in de eerste kolom. Je kunt eventueel ook de (dynamisch toegewezen) naam gebruiken, in dit geval `sad_galileo`).

# Opwarmer 2: Container met NGINX "Hello World"

In de eerste stap in deze workshop gaan we een simpele Dockerfile definieren en de image hiervan bouwen. Vervolgens gaan we een container runnen van deze image en verifieren of het werkt. De NGINX in deze container moet via een TCP port(8080) op de Docker host (je VM met docker) benaderbaar zijn.

## Dockerfile

De naam `Dockerfile` voor het docker build proces is net zoiets als `Makefile` voor `make`. Als er een `Dockerfile` aanwezig is in de aangewezen directory tijdens een docker build, worden de instructies hieruit gebruikt om een image te "bouwen".

## De Dockerfile

Ga naar `/var/tmp` en creeer de directory `nginx`.

In de directory `nginx` creeer je volgende inhoud in de `Dockerfile`:

```
# Version: 0.0.1
FROM ubuntu:14.10
MAINTAINER James Turnbull "james@example.com"
RUN apt-get update
RUN apt-get install -y nginx
RUN echo 'Hi, I am in your container' \
    >/usr/share/nginx/html/index.html
EXPOSE 80
```

## Build de image

Voor het commando uit in de `nginx` directory. ***Let op de punt (=deze/huidige directory) op de commandline van het build commando!***

Onderstaande output is sterk ingekort!

```
$ docker build -t hferinga/nginx .
```

```
Sending build context to Docker daemon 2.048 kB
Sending build context to Docker daemon
Step 0 : FROM ubuntu:14.10
 ---> 59a878f244f6
Step 1 : MAINTAINER James Turnbull "james@example.com"
 ---> Running in b10d06843d2d
 ---> b0a2b67cd129
Removing intermediate container b10d06843d2d
Step 2 : RUN apt-get update
 ---> Running in 7067110f87c1
...
 ---> 42f8442497fb
Removing intermediate container 7067110f87c1
Step 3 : RUN apt-get install -y nginx
 ---> Running in 1a0b48915306
...
 ---> 4abd062fe0cd
Removing intermediate container 1a0b48915306
Step 4 : RUN echo 'Hi, I am in your container' >
/usr/share/nginx/html/index.html
 ---> Running in 83eedbd890b9
 ---> fe9494537b20
Removing intermediate container 83eedbd890b9
Step 5 : EXPOSE 80
 ---> Running in f32f91e5aae4
 ---> 0eed2f6a7e2c
Removing intermediate container f32f91e5aae4
Successfully built 0eed2f6a7e2c
```

De **FROM** regel in de Dockerfile geeft aan welke base image wordt gebruikt om de uiteindelijke image te bouwen. In dit geval is het **`ubuntu:14.10`**.

Je kunt met het commando "`docker images`" een lijst van lokaal beschikbare images opvragen:

```
$ docker images | grep nginx
hferinga/nginx    latest   0eed2f6a7e2c        27 minutes ago      233.4 MB
```

## Start de container

Zodra de image succesvol is gebouwd, moet je van deze image een "running" container maken. In het "`docker run`" commando geef je aan hoe de interne exposed port (tcp 80) op de Docker host wordt gepresenteerd. Dit doe je met de **`-p`** optie. Je kunt eventueel aangeven dat de "EXPOSED" port op de docker host op een andere poort wordt gemapped. In onderstaand voorbeeld is de EXPOSED port in de container **tcp:80** gemapped op de docker host **tcp:8088**.

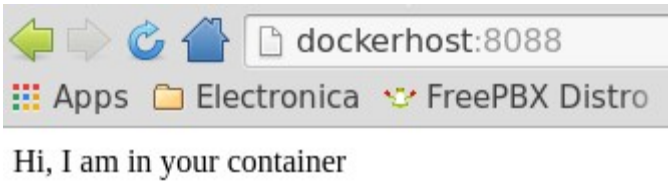
```
$ docker run -p 8088:80 --name nginx-web hferinga/nginx nginx \
  -g "daemon off;"
```

Dit voert de container (soort van) in de **voorgrond** uit (met de **`-d`** optie kun je hem in de achtergrond (detached) laten draaien). In een andere terminal sessie (op je laptop) kun je dan met curl verifieren dat nginx de statische pagina kan serveren (proost). Dat kan natuurlijk ook met een

browser op je laptop door naar het ip adres van je dockerhost te gaan op de juiste tcp port (in dit geval 8088)

Een container blijft in principe achter nadat deze gestopt is. Dat betekent dat je hem niet onder dezelfde naam weer opnieuw kan "**runnen**". Je kunt de container met het commando "**`docker rm nginx-web`**" verwijderen als deze niet meer actief is. Het is ook mogelijk een container met het commando "`docker start <container-id>`" weer opnieuw te starten.

### *Wijs met je browser naar de container poort*




### *Of met het curl commando*

```
# curl dockerhost:8088
Hi, I am in your container
```

## Samenwerkende containers

In de volgende reeks opdrachten uit deze workshop gaan we werken met containers die met elkaar samen werken met behulp van met name volume's en links.

### *Volume's*

De volume optie (`-v/--volumes`) biedt de mogelijkheid om een volume in een container te creeeren van een directory van de dockerhost.

Volumes zijn special ontworpen om data in directories in een of meer containers te delen of "persistent" te maken. Deze volumes zijn geen onderdeel van de file system layering (van het Union File System) die normaal gebruikt wordt. Volumes zijn dus geen onderdeel van een image, maar kunnen dat wel zijn van een container. Gebruikelijk is om volumes op externe (t.o.v containers) storage te definieren.

Volumes blijven toegankelijk, ook als een container gestopt is. In een van de opdrachten wordt hiervan gebruik gemaakt.

### *Links tussen containers*

De `--link` flag creeert een parent-child link tussen twee containers. De flag neemt twee argumenten: de container naam om te linken en een alias voor de link gescheiden door een dubbele punt (:).

vb:

```
--link apache:web02
```

Dit creeert een link tussen de container die wordt opgestart (run) met de container met de naam **apache**. In de container die wordt opgestart wordt is de verbinding met de apache container via de naam **`web02`** beschikbaar (wordt geconfigureerd in `/etc/hosts` van de container). Dit is een private verbinding tussen de twee containers.

Vanuit **security** oogpunt is dit ook een interessante manier om twee container veilig met elkaar te kunnen laten communiceren. Er is namelijk geen externe publiek toegankelijke poort "exposed" op de dockerhost. Alleen containers die expliciet zijn gelinkt met `--link` kunnen met elkaar communiceren over de exposed ports. Ze zitten in een soort van private network.

In de naam-kolom uitvoer van het `docker ps` commando is de link en alias terug te zien:

```
HAP2/web01,nodeapp
```

De **nodeapp** container heeft hier een link met de **HAP2** container met alias **web01**. In de HAP2 container kan aan de nodeapp container gerefereerd worden met de hostname web01.

In de /etc/hosts file in de HAP2 container, die twee links heeft, staat dan bijvoorbeeld dit:

```
[root@d5f98c6f73dc /]# cat /etc/hosts
172.17.0.58      d5f98c6f73dc
127.0.0.1        localhost
::1              localhost ip6-localhost ip6-loopback
....
172.17.0.8       web01
172.17.0.20      web02
```

Indien we vanuit de HAP2 container met bijvoorbeeld web01 willen communiceren die een website op poort 3000 heeft draaien, kan dat op de volgende manier:

```
$ curl web01:3000
```

# Opdracht 1: multicontainer webapp Jekyll en Apache

We gaan 2 containers maken. Een Jekyll container en een Apache container. In deze containers gaan we werken met volumes. De Jekyll container schrijft in een volume die later door de Apache container wordt gebruikt (in zijn documentroot van de webserver).

## *Bouwen images*

Voordat we zover zijn, moeten we eerst images bouwen. Een voor Jekyll en een voor Apache.

We gaan eerst de Jekyll base image bouwen a.h.v de volgende **Dockerfile**.

```
FROM ubuntu:14.04
MAINTAINER James Turnbull <james@example.com>
ENV REFRESHED_AT 2014-06-01

RUN apt-get -yqq update
RUN apt-get -yqq install ruby ruby-dev make nodejs
RUN gem install --no-rdoc --no-ri jekyll

VOLUME /data
VOLUME /var/www/html
WORKDIR /data

ENTRYPOINT [ "jekyll", "build", "--destination=/var/www/html" ]
```

```
$ cd jykell
$ docker build -t hferinga/jekyll .
```

***Let op: de punt (betekent "deze directory") in het commando!***

Vervolgens gaan we de **Apache** image bouwen met behulp van de volgende **Dockerfile**:

```
FROM ubuntu:14.04
MAINTAINER James Turnbull <james@example.com>
ENV REFRESHED_AT 2014-06-01

RUN apt-get -yqq update
RUN apt-get -yqq install apache2

VOLUME [ "/var/www/html" ]
WORKDIR /var/www/html

ENV APACHE_RUN_USER www-data
ENV APACHE_RUN_GROUP www-data
ENV APACHE_LOG_DIR /var/log/apache2
ENV APACHE_PID_FILE /var/run/apache2.pid
ENV APACHE_RUN_DIR /var/run/apache2
ENV APACHE_LOCK_DIR /var/lock/apache2

RUN mkdir -p $APACHE_RUN_DIR $APACHE_LOCK_DIR $APACHE_LOG_DIR

EXPOSE 80
ENTRYPOINT [ "/usr/sbin/apache2" ]
CMD ["-D", "FOREGROUND"]
```

```
$ cd ../apache
$ docker build -t hferinga/apache .
```

***Let hier ook weer op de '.' (punt).***

Nu hebben we nog **web content** nodig. Hiervoor gebruiken we een stukje blog source van James Turnbull, dat we via een `git clone` ophalen.

```
$ cd ..
$ mkdir blog
$ cd blog
$ git clone https://github.com/jamtur01/james_blog.git
```

Ivm SELinux moeten we een label aanpassing doen in de blog directory tree:

```
$ sudo chcon -Rt svirt_sandbox_file_t `pwd`/blog
```

Controleer nu welke images je hebt gebouwd.

```
$ docker images
```

## Starten containers

Start de **Jekyll** container voor de eerste keer.

```
$ docker run --volume `pwd`/blog/james_blog:/data/ -h james_blog \
  --name james_blog hferinga/jekyll
```

***Let hier op de --volume***

Start nu ook de **apache** container:

```
$ docker run -d -P --volumes-from james_blog -h apache --name apache \
  hferinga/apache
```

In dit geval wordt verwezen naar een volume in een andere container met de naam "**james_blog**". Om nu met de apache server te kunnen communiceren zul je er achter moeten komen welke port op de host (je Docker VM) is gekoppeld aan de container.

```
$ docker ps  -l
CONTAINER ID        IMAGE                     COMMAND                CREATED
STATUS              PORTS                     NAMES
14f3c9fb87d7        hferinga/apache:latest    /usr/sbin/apache2 -D   About a
```

```
minute ago   Up About a minute   0.0.0.0:49153->80/tcp   apache
```

Of met het volgende docker port commando:

```
$ docker port apache 80
0.0.0.0:49153
```

80 is de poort die door de apache image is "exposed" en deze is dynamisch gemapped naar 49153 op de docker host (en is dus elke keer anders). Je kunt deze ook statisch mappen naar een eigen gekozen port door de **`-p`** optie te gebruiken ( `-p 8080:80`).

Je kunt nu door met je browser vanaf je laptop met de url: http://<vm-ip-address>:49153 de blog benaderen.

Zoals je hebt kunnen zien is de jekyll container vrijwel direct weer gestopt. Jekyll genereert van de configuratie in de blog directory tree statische web pagina's, en maakt deze beschikbaar op het volume. Daarmee is de jekyll container klaar met zijn werk en stopt ermee en kan apache de pagina's serveren. Volumes hebben de eigenschap beschikbaar te blijven, ook als de container op dat moment niet actief is.

***Een volume die alleen IN een container aanwezig is, verdwijnt (inclusief data) als er geen container (aktief of inaktief) meer aan refereert!***

Nu gaan we een kleine wijzing aanbrengen in de `blog/index.md` pagina. Bijvoorbeeld in de "**title**". Maak hier een wijziging en start de container opnieuw op. Deze keer niet met een run, maar met een start. Het run commando maakt van een image een container instance. Maar we hebben al een container gemaakt onder de naam "**james_blog**" tijdens de eerste run. Als je deze run opnieuw uitvoert, krijg je een foutmelding dat de container met die naam al bestaat.

```
$ docker run -v `pwd`/blog/james_blog:/data/ --name james_blog\
  hferinga/jekyll
2015/01/22 20:49:31 Error: Conflict, The name james_blog is already assigned
to 575a385083ea. You have to delete (or rename) that container to be able to
assign james_blog to a container again.
```

We gaan de bestaande container dus herstarten, zodat jekyll de aanpassingen kan verwerken.

```
$ docker start james_blog
james_blog
```

We kunnen hier de naam james_blog gebruiken ipv de random naam die door docker aan de container wordt gegeven, of het container-id. De naam hadden we tijdens het run commando aan de container gegeven (met het argument `--name`).

Als je nu je browser refreshed, dan zul je zien dat jekyll je wijzigingen heeft omgezet naar webpagina's. De Apache container hoeft hiervoor niet te worden herstart.

## Interactieve shell starten in een running container

Met behulp van het `docker exec` commando kun je een commando uitvoeren in een running container. In het volgende voorbeeld kun je een interactieve shell opstarten in de running container. (Deze optie is vanaf Docker 1.3 beschikbaar.)

Het opzetten van de interactieve shell gaat als volgt:

```
$ docker exec -t -i apache /bin/bash
root@apache:/var/www/html# exit
exit
```

Je krijgt dan een prompt in de container (in dit geval de container met de naam "apache").

Met het exit commando (of CTRL-D) verlaat je deze interactieve shell weer.

## Backup maken van data in een container

Om een backup te maken van data in een container, kun je gebruikmaken van een wegwerp container.

***De directory waarin de backup wordt geschreven moet de juiste selinux context hebben!***

Maak de backup directory aan op de dockerhost.

```
$ mkdir /var/tmp/backupdir
$ sudo chcon -Rt svirt_sandbox_file_t /var/tmp/backupdir
```

Bijvoorbeeld om een tar backup van de data in de james_blog container volume te maken kun je volgende commando gebruiken.

```
$ docker run --rm --volumes-from james_blog -v                    \
       /var/tmp/backupdir:/backup                                    \
       ubuntu tar cvf /backup/james_blog_backup.tar /var/www/html
```

Door middel van het `--rm` argument, wordt de container na run direct weer weggegooid. De `-v` optie duidt hier de backup volume aan waarop de backup wordt geschreven op het filesysteem van de dockerhost. In de wegwerp container wordt dit door de `/backup` directory gepresenteerd. Er wordt gebruik gemaakt van een ubuntu base image.

# Opdracht 2: multicontainer app Nodejs, Redis, logstash

In deze opdracht gaan we een multicontainer Nodejs applicatie opzetten met een redundante Redis (key/value pair) data store en een log(stash) server. In deze opdracht werken we met volumes en links tussen containers.

Wat hebben we hier voor nodig:

1 nodejs container
1 redis primary container
2 redis replica containers
1 logstash container

Dus eerst hebben we de images nodig.

## *Bouwen images*

We bouwen de **nodejs** image van de volgende Dockerfile:

```
FROM ubuntu:14.04
MAINTAINER James Turnbull <james@example.com>
ENV REFRESHED_AT 2014-06-01

RUN apt-get -yqq update
RUN apt-get -yqq install nodejs npm
RUN ln -s /usr/bin/nodejs /usr/bin/node
RUN mkdir -p /var/log/nodeapp

ADD nodeapp /opt/nodeapp/

WORKDIR /opt/nodeapp
RUN npm install

VOLUME [ "/var/log/nodeapp" ]
EXPOSE 3000

ENTRYPOINT [ "nodejs", "server.js" ]
```

**Bouwen**:

```
$ cd nodeapp
$ sudo docker build -t hferinga/nodejs .
```

We gaan nu een **redis basis image** bouwen waarop we zowel de primary als de replica's baseren.

In de base image zijn de **`VOLUME`** definities gemaakt die zowel voor de redis_primary als voor de redis_replica nodig zijn.

```
$ mkdir redis_base
$ cd redis_base
$ vi Dockerfile
```

Dockerfile:

```
FROM ubuntu:14.10
MAINTAINER James Turnbull <james@example.com>
ENV REFRESHED_AT 2014-06-01

RUN apt-get -yqq update
RUN apt-get install -yqq software-properties-common python-software-properties
RUN add-apt-repository ppa:chris-lea/redis-server
RUN apt-get -yqq update
RUN apt-get -yqq install redis-server redis-tools

VOLUME [ "/var/lib/redis", "/var/log/redis" ]
CMD []
```

Bouwen:

```
$ docker build -t hferinga/redis_base .
```

Vervolgens bouwen we de **redis primary** image.

```
$ mkdir redis_primary
$ cd redis_primary
$ vi Dockerfile
```

Dockerfile voor **redis_primary**:

```
FROM hferinga/redis_base
MAINTAINER James Turnbull <james@example.com>
ENV REFRESHED_AT 2014-06-01

VOLUME [ "/var/lib/redis", "/var/log/redis" ]

EXPOSE 6379
ENTRYPOINT [ "redis-server", "--logfile /var/log/redis/redis-server.log" ]
```

Hiervan bouwen we de primary redis image als volgt:

```
$ docker build -t hferinga/redis_primary .
```

De **redis replica** wordt op eenzelfde manier gemaakt.

Voorbereiding:

```
$ cd ..
$ mkdir redis_replica
$ cd redis_replica
$ vi Dockerfile
```

van de volgende Dockerfile:

```
FROM hferinga/redis_base
MAINTAINER James Turnbull <james@example.com>
ENV REFRESHED_AT 2014-06-01

VOLUME [ "/var/lib/redis", "/var/log/redis" ]

ENTRYPOINT [ "redis-server", "--logfile /var/log/redis/redis-replica.log", "--
slaveof redis_primary 6379" ]
```

***Let op de port 6379. Dit is de port waarmee de redis replica communiceert met de primary redis server.***

Bouwen van de redis replica image:

```
$ sudo docker build -t hferinga/redis_replica .
```

Nu hebben we alleen nog de **logstash** image nodig om onze applicatielogs op te vangen.

Voorbereiding:

```
$ mkdir logstash
$ cd logstash
$ vi logstash.conf
$ vi Dockerfile
```

Inhoud van de `logstash.conf` file:

```
input {
  file {
    type => "syslog"
    path => ["/var/log/nodeapp/nodeapp.log", "/var/log/redis/redis-
server.log"]
  }
}
output {
  stdout {
    codec => rubydebug
  }
}
```

Dockerfile van logstash:

```
FROM ubuntu:14.04
MAINTAINER James Turnbull <james@example.com>
ENV REFRESHED_AT 2014-06-01

RUN apt-get -yqq update
RUN apt-get -yqq install wget
RUN wget -O - http://packages.elasticsearch.org/GPG-KEY-elasticsearch |  apt-
key add -
RUN echo 'deb http://packages.elasticsearch.org/logstash/1.4/debian stable
main' > /etc/apt/sources.list.d/logstash.list
RUN apt-get -yqq update
RUN apt-get -yqq install logstash

ADD logstash.conf /etc/

WORKDIR /opt/logstash

ENTRYPOINT [ "bin/logstash" ]
CMD [ "--config=/etc/logstash.conf" ]
```

Bouwen:

```
$ docker build -t hferinga/logstash .
```

## *Applicatie cluster opstarten.*

Eerst de primary redis server:

```
$ docker run -d -h redis_primary \
    --name redis_primary hferinga/redis_primary
```

Dan de 2 redis replica's. We gebruiken hiervoor telkens dezelfde redis replica image:

```
$ docker run -d -h redis_replica1 \
    --name redis_replica1 \
    --link redis_primary:redis_primary \
    hferinga/redis_replica
```

Check de log:

```
$ docker run -ti --rm --volumes-from redis_replica1 \
    ubuntu cat /var/log/redis/redis-replica.log
```

***We gebruiken hiervoor een wegwerp container omdat redis naar een log file logt en dus het docker logs commando niets laat zien!***

Vervolgens voegen we de tweede replica toe:

```
$ docker run -d -h redis_replica2 \
```

```
    --name redis_replica2 \
    --link redis_primary:redis_primary \
    hferinga/redis_replica
```

Check ook de log van de tweede replica:

```
$ docker run -ti --rm --volumes-from redis_replica2 ubuntu \
    cat /var/log/redis/redis-replica.log
```

Nu de redis backend in de lucht is, kunnen we onze "NodeJS applicatie" lanceren:

```
$ docker run -d \
    --name nodejs -h nodejs -p 3000:3000 \
    --link redis_primary:redis_primary \
    hferinga/nodejs
```

Je kunt nu met je browser naar: <ipadres-van-je-dockerhost>:3000 op je laptop.

## *Logstash*

De extra component logstash kun je nu opstarten om de logs van de applicatie op te vangen.

```
$ docker run -d --name logstash -h logstash \
    --volumes-from redis_primary \
    --volumes-from nodejs \
    hferinga/logstash
```

Je ziet dat de logstash container de volumes gebruikt van zowel de redis backend als van onze NodeJS applicatie.

Logs bekijken die door logstash worden verzameld:

```
$ docker logs -f logstash
```

# Opdracht 3: Toevoegen van een HAproxy loadbalancer

**HAproxy Dockerfile**

(origineel van git clone https://github.com/dockerfile/haproxy)

Dockerfile: (aangepast van het origineel voor CentOS, was Ubuntu)

```
#
# Haproxy Dockerfile
#
# https://github.com/dockerfile/haproxy
#

# Pull base image.
FROM centos:latest
MAINTAINER hferinga
ENV REFRESHED_AT 2014-12-01

# Install Haproxy.

ADD yum.repos.d /etc/yum.repos.d
ADD ./rpm-gpg/ /var/tmp
ADD ./rpm-gpg/ /etc/pki/rpm-gpg/
#RUN rpm --import /var/tmp/RPM-GPG-KEY-EPEL-7
RUN rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-7

RUN yum -y update
RUN yum -y install wget
RUN yum -y install tcpdump
RUN yum -y install curl
RUN yum -y install tar
RUN yum -y install net-tools
RUN yum -y install e2fsprogs
RUN yum -y install haproxy
#RUN yum -y install python-pip
#RUN pip install requests==2.2.1

RUN yum clean all


# Add files.
ADD haproxy.cfg /etc/haproxy/haproxy.cfg
ADD start.bash /haproxy-start

# Define mountable directories.
VOLUME ["/haproxy-override"]

# Define working directory.
WORKDIR /etc/haproxy

# Define default command.
CMD ["bash", "/haproxy-start"]

# Expose ports.
EXPOSE 80
EXPOSE 443
```

## Build

```
$ docker build -t hferinga/haproxy .
```

## HAProxy configuratie (`haproxy.cfg`).

Aangepaste versie in `/var/tmp/haproxy` via volume.
Copieer deze uit de workshop-files directory naar de "override" volume.

***Het origineel werkt niet. Gebruik daarom voor dit onderdeel altijd de override configuratie.***

Dit is de override config (via `-v/ --volume`).

```
global
  log 127.0.0.1 local0
  log 127.0.0.1 local1 notice
  # chroot werkt alleen bij run --privileged=true
  # chroot /var/lib/haproxy
  user haproxy
  group haproxy
  # niet in daemon mode anders stopt container direct !!!
  # daemon

defaults
  log global
  mode http
  option httplog
  option dontlognull
  timeout connect 5000ms
  timeout client 50000ms
  timeout server 50000ms

listen stats :88
  stats enable
  stats uri /hapstats

frontend localnodes
    bind *:80
    mode http
    default_backend nodes

backend nodes
    mode http
    balance roundrobin
    option forwardfor
    http-request set-header X-Forwarded-Port %[dst_port]
    http-request add-header X-Forwarded-Proto https if { ssl_fc }
    option httpchk HEAD / HTTP/1.1\r\nHost:localhost
    server web02 172.17.0.37:80 maxconn 32 check
    server web01 172.17.0.108:3000 maxconn 32 check
```

## Run en prerequisites

Creeer de directory `/var/tmp/haproxy` op de dockerhost:

```
$ mkdir /var/tmp/haproxy
```

En copieer de bovenstaande (override) config hierin (kun je vinden in de `workshop-files` directory):

```
$ vi /var/tmp/haproxy/haproxy.cfg
of
$ cp ~/workshop-01/workshop-files/haproxy.cfg /var/tmp/haproxy/
```

Zet vervolgens de selinux context goed.

## SELinux

```
$ sudo chcon -Rt svirt_sandbox_file_t /var/tmp/haproxy/
```

Start vervolgens de container voor de eerste keer:

```
$ docker run -d -p 8000:80 -p 8188:88 -v /var/tmp/haproxy:/haproxy-override \
       --name HAP2 \
       hferinga/haproxy
```

***De laatste twee regels van de configuratie bevatten ip adressen en die worden dynamisch toegekend aan de containers die een webservice aanbieden. Beter is om hier gebruik te maken van links met een naam/alias.***

Nu met `--link name:alias`. Hierbij wordt de **alias** in de `/etc/hosts` gezet inclusief het actuele ip adres. De '`name`' is de naam van de container en deze wordt op deze manier gekoppeld aan een alias.

De container wordt dan op de volgende manier voor de eerste keer gestart:

```
$ docker run -d -p 8000:80 -p 8188:88 \
        -v /var/tmp/haproxy:/haproxy-override \
        --name HAP2 \
        --link apache:web02\
        --link nodejs:web01\
        hferinga/haproxy
```

In de voorbeeld configuratie kunnen we dan de 'server' regels in de `haproxy.cfg` vervangen door:

```
    server web02 web02:80 maxconn 32 check
    server web01 web01:3000 maxconn 32 check
```

Nu werkt het dus zonder dat we het ip adres hoeven op te zoeken en te configureren.

Je kunt nu met je browser de statistieken bekijken op de URL: http://dockerhost:8188/hapstats en de websites achter de loadbalancer op de URL: http://dockerhost:8000. Omdat we voor web01 en web02 andere websites gebruikt hebben (niet gebruikelijk natuurlijk voor een loadbalancer), zie je alternerend de web01 en de web02 pagina. (gewoon om dat het kan).

Eventueel met curl vanaf de commandline:

```
$ watch curl http://dockerhost:8000
```

***In een browser wil het nog wel eens niet lukken vanwege de caching van de browser en zie je meestal maar een van de websites. Met het curl commando gaat het wel goed.***

# Diverse commando's

Deze kun je terugvinden in de `workshop-files` directory in het bestand `div-commands`.

```
#############################################################################
# build images (telkens vanuit de directory met de Dockerfile)
# Als de user die de docker commando's uitvoert in de group docker zit,
# kunnen de docker commando's zonder sudo worden uitgevoerd!
#############################################################################
# jekyll/apache (Opdracht 1)
sudo docker build -t hferinga/jekyll .
sudo docker build -t hferinga/apache .

# redis,nodejs,logstash (Opdracht 2)
sudo docker build -t hferinga/redis_base .
sudo docker build -t hferinga/redis_primary .
sudo docker build -t hferinga/redis_replica .
sudo docker build -t hferinga/nodejs .
sudo docker build -t hferinga/logstash .

# haproxy
sudo docker build -t hferinga/haproxy .
#############################################################################
# run containers
```

```
#############################################################################
# jekyll/apache (Opdracht 1)
sudo docker run --volume `pwd`/blog/james_blog:/data/ -h james_blog   \
     --name james_blog hferinga/jekyll
#sudo docker run -d -P --volumes-from james_blog -h apache\
     --name apache hferinga/apache
# of
sudo docker run -d --volumes-from james_blog -h apache \
     --name apache -p 8080:80 hferinga/apache

# redis(primary,replica{1,2},nodejs,logstasch
sudo docker run -d -h redis_primary --name redis_primary \
     hferinga/redis_primary
sudo docker run -d -h redis_replica1 --name redis_replica1 \
     --link redis_primary:redis_primary hferinga/redis_replica
sudo docker run -d -h redis_replica2 --name redis_replica2 \
     --link redis_primary:redis_primary hferinga/redis_replica
sudo docker run -d --name nodejs -h nodejs -p 3000:3000 \
     --link redis_primary:redis_primary hferinga/nodejs
sudo docker run -d --name logstash -h logstash --volumes-from \
     redis_primary --volumes-from nodejs hferinga/logstash

# haproxy
sudo docker run -d -p 8000:80 -p 8188:88 -v \
     /var/tmp/haproxy:/haproxy-override -h HAP2 \
     --name HAP2 --link apache:web02 --link nodejs:web01 \
     hferinga/haproxy

#############################################################################
# start containers
#############################################################################
sudo docker start redis_primary
sudo docker start redis_replica{1,2}
sudo docker start nodejs
sudo docker start logstash
sudo docker start apache
sudo docker start HAP2
sudo docker start james_blog
#############################################################################
# stop containers
#############################################################################
sudo docker stop redis_primary
sudo docker stop redis_replica{1,2}
sudo docker stop nodejs
sudo docker stop logstash
sudo docker stop apache
sudo docker stop HAP2
#############################################################################
# wegwerp containers:
#############################################################################
sudo docker run -ti --rm --volumes-from=redis_replica1 \
     ubuntu cat /var/log/redis/redis-replica.log
sudo docker run--rm --volumes-from=james_blog -v $(pwd):/backup \
     ubuntu tar cvf /backup/james_blog_backup.tar /var/www/html
sudo docker run--rm --volumes-from=james_blog \
      -v /var/tmp/backupdir:/backup \
```

```
     ubuntu tar cvf /backup/james_blog_backup.tar /var/www/html

################################################################################
# diversen
################################################################################
sudo docker logs -f logstash
sudo chcon -Rt svirt_sandbox_file_t `pwd`/blog
sudo chcon -Rt svirt_sandbox_file_t /var/tmp/haproxy/
sudo chcon -Rt svirt_sandbox_file_t /var/tmp/backupdir
```

# Starten en stoppen van de containers

Na de eerste 'run' moet je een 'start' in plaats van een 'run' doen.

## *Start*

Onderstaande reeks start de containers uit deze workshop in de juiste volgorde op.

```
docker start redis_primary
docker start redis_replica{1,2}
docker start nodeapp
docker start logstash
docker start apache
docker start james_blog
docker start HAP2
```

## *Stop*

Met de volgende reeks commando's stop je de containers weer.

```
docker stop redis_primary
docker stop redis_replica{1,2}
docker stop nodeapp
docker stop logstash
docker stop apache
docker stop HAP2
docker stop james_blog
```

## into-container script (niet meer nodig met docker exec)

Dit script maakt gebruik van `nsenter` om een shell in de actieve container te krijgen. Copieer het in het bestand `/usr/local/bin/into-container` en maak het executable. **Overigens is sinds docker 1.3 het `docker exec` commando beschikbaar, waardoor je zonder onderstaand script nog makkelijker een interactieve shell (of ander commando) in een running container kunt krijgen.** Dus gebruik ipv onderstaand script het `docker exec` commando!

```
#!/bin/bash

DockerOpts=""

errorexit() {
  echo "ERROR: $*"
  exit 1
}

usage() {
  echo "usage: ${0##*/} image-name"
  exit 1
}

[[ $# -ne 1 ]] && usage

IMAGE=$1

# Get container id
ID=$(docker ${DockerOpts} ps -a | grep Up | grep $IMAGE | awk '{print $1'})

echo ID:$ID

[[ "x${ID}" == "x" ]] && errorexit "Could not find Container ID: Is it
running?"

# Now get PID of Container
PID=$(docker ${DockerOpts} inspect --format '{{.State.Pid}}' $ID)

echo PID: $PID

# now enter container
nsenter --target ${PID} --mount --uts --ipc --net --pid
```

```
$ sudo vi /usr/local/bin/into-container
$ sudo chmod +x /usr/local/bin/into-container
```

Gebruik het als volgt:

(458124bb22d5 is het container-id)

```
$ sudo /usr/local/bin/into-container 458124bb22d5
```

# Docker common commands

| Purpose | Command |
|---|---|
| Build an image | docker build –rm=true . |
| Install an image | docker pull ${IMAGE} |
| List of installed images | docker images |
| List of installed images (detailed listing) | docker images –no-trunc |
| Remove an image | docker rmi ${IMAGE_ID} |
| Remove all untagged images | docker rmi $(docker images \| grep “^” \| awk “{print $3}”) |
| Remove all images | docker rm $(docker ps -aq) |
| Run a container | docker run |
| List containers | docker ps |
| Stop a container | docker stop ${CID} |
| Find IP address of the container | docker inspect –format ‘{{ .NetworkSettings.IPAddress }}’ ${CID} |
| Attach to a container | docker attach ${CID} |
| Remove a container | docker rm ${CID} |
| Remove all containers | docker rm $(docker ps -aq) |
| Start interactief process in container | docker exec -t -i ${CID} /bin/bash |

Start interactief process in container docker exec ${CID} /bin/bash