



## container technologie

"Save the whales. Collect the whole set!"

Oscar Buse  
11 April 2017  
NLUG

# Inleiding

---

Dit praatje gaat over docker: een Linux container technologie.

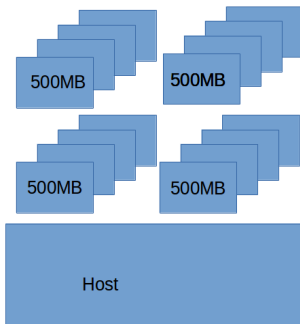
De onderwerpen die aan bod komen:

- Waaron docker?
- Wat is docker?
- Installatie.
- De docker omgeving.
- Images, layers en containers..
- Enkele praktijk voorbeelden I.
- Enkele veel gebruikte commando's.
- cgroups en namespaces.
- Enkele praktijk voorbeelden II.
- Linking containers
- Troubleshooting.
- Uploaden van je container/image naar een repository.
- De voor- en nadelen van docker.
- Best practices

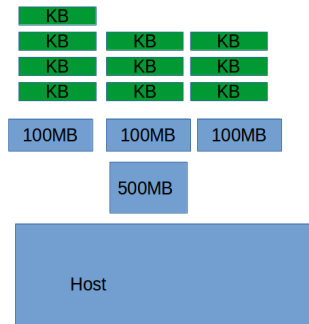
# Waarom docker?

- "Op mijn development omgeving werkt alles prima!"  
Docker: "*build once, run anywhere*"
- Kleine *footprint* en mede daardoor:
  - goed schaalbaar.
  - snelle startup.

In een zeer versimpelde weergave:



VM



Docker

# Wat is docker? 1/2

---

Zomaar wat eigenschappen van docker containers:

- Vrij nieuw: 15/03/2013. Flink groeiende user-base.
- Container technologie (denk aan OpenVZ, LXC, Solaris zones, ...)
- Denk meer aan een single proces dan aan een VM.
- Verspreidbare (software) eenheid voor elke omgeving (als er maar een docker daemon runt). Handig voor software workflows (OTAP).
- "Build once, run anywhere".
- Vluchtig: meer geschikt voor een kortdurend bestaan (maar hoeft niet). Bv. volstrekt normaal om docker eenmalig een extern request te laten doen (later meer).
- Meer geschikt voor stateless applicaties.

## Wat is docker? 2/2

---

- Goed voor "microservices" (grote applicatie opgedeeld in kleinere delen (microservices)).
- Zuinig met diskruimte: *images* worden geshared.
- Snelle startup (voor bv. bijschakelen resources (denk bv. aan extra webserver)).
- "Een *image* voor iedere toepassing" (in de repositories).

# Installatie

---

Docker heeft zijn **eigen** repository voor het package "docker-engine".

Voor bv. CentOS:

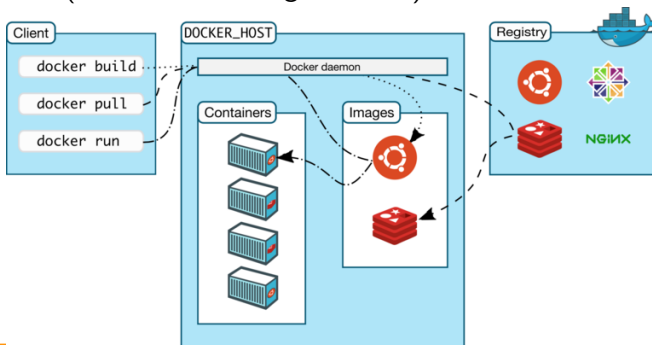
```
# yum install -y yum-utils
# yum-config-manager --add-repo \
    https://docs.docker.com/engine/installation/linux/repo_files/centos/docker.repo
Daarna: install, update (downgrade) docker vanuit de repository:
# yum install docker-engine
(alternatief: curl -sSL http://get.docker.com | sh)
```

Test bv. met "docker run hello-world"

# De docker omgeving

Een overzicht:

- dockerd - de docker daemon
- docker - de cli
- remote API
- repositories met docker images (hub.docker.com)
- compose, machine (voor docker hosts), swarm, k8s (*orchestration*, volgende keer).



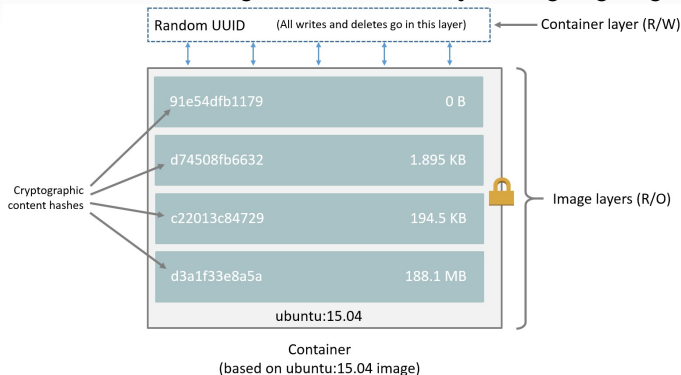
# Images, layers en containers.. 1/2

Voordat we een praktijkvoorbeeld zien eerst wat meer over images, layers en containers..:

**image** Filestelsysteem als read-only basis voor een container. Bestaat uit meerdere (ook read-only) layers. *Distributable unit*.

**layers** "Filestelsysteem verandering in een image". De docker storage engine combineert meerdere layers tot 1 view (filestelsysteem) met *union mounting*.

**container** Docker image met een dunne schrijfbare laag toegevoegd.





## Images, layers en containers.. 2/2

---

- Sharen van image layers: zuinig mbt diskusage, performance winst.
- Copy-on-Write (CoW) toegepast.
- Vóór versie 1.10: layers en images opgeslagen met random UUID. Nadelen hiervan:
  - slechte data integriteit (geen checksums). Vanaf v1.10 sha256 voor layers en images.
  - kans op dubbele ID's.

## cgroups en namespaces

---

Docker maakt gebruik van cgroups en namespaces:

### cgroups

Limit resources.

- iedere container eigen cgroup (onder /sys filesystem).

### namespaces

Gelijke namen mogelijk door isolatie van de namespace:

- mount namespace (bv. / in container != / in host != / in andere container)
- PID namespace
- Netwerk namespace - (bv. port 80 in container != port 80 on host)
- User namespace - root heeft bv. wel eigen namespace maar ja..

# Enkele praktijk voorbeelden I

---

Algemene werkwijze met docker containers:

- build (maak een image mbv een Dockerfile)
  - `docker build -t voorbeeld/mijn_image .`
- run (= create + start container van image)
  - `docker run voorbeeld/mijn_image`

Voorbeeld 1: Hello world

Voorbeeld 2: Een random quote

Voorbeeld 3: Nog een random quote

# Enkele veel gebruikte commando's

---

Maken/Verkrijgen van een image:

- Custom mbv "Dockerfile": `# docker build`
- Downloaden met: `# docker pull`

Enkele commando's:

- `docker version` (tegenwoordig in YY.MM format (*monthly release cycle*))
- `docker info | less`
- `docker build -t example/quote-web .`
- `docker pull <image van reposotory>`
- `docker run -d -m 500m --rm --read-only --name quote-web \`  
    `-p 8081:8080 example/quote-web`
- `docker images`
- `docker ps`
- `docker stop <CONTAINER\_ID/NAME>` (stop container)
- `docker stop $(docker ps -a -q)` (stop alle containers)
- `docker rm <CONTAINER\_ID/NAME>` (remove container)
- `docker rmi <IMAGE\_ID>` (remove image)
- `docker exec -it quote-web /bin/bash` (krijg een shell in de container "quote-web")

## Voorbeeld 3

---

Voorbeeld 3: 2 containers in een apart subnet.  
Enkele commando's:

- `docker network create --subnet=172.18.0.0/16 hours-net`
- `docker network ls` (**toont alle netwerken**)
- `docker run -d --net hours-net --ip 172.18.0.3 -p 8010:8080\`  
`--name web oscarbuse/hours-web`
- `docker run -d --net hours-net --ip 172.18.0.2 --name db\`  
`-v /var/lib/mysql:/var/lib/mysql example/hours-db`

## Linking containers

---

Containers zijn eenvoudig te linken door naar de naam te verwijzen. Je hoeft dan geen eigen subnet te creëren.

Een voorbeeld van een Wordpress container gelinkt met een MySQL database container:

- `docker pull wordpress:latest`
- `docker pull mysql:latest`
- `docker run --name mysqlwp -e MYSQL_ROOT_PASSWORD=changeme -d mysql`
- `docker run --name wordpress --link mysqlwp:mysql -p 8080:80 -d wordpress`
- `docker ps`

Met joomla:

- `docker pull joomla:latest`
- `docker run --name joomla --link mysqlwp:mysql -p 8081:80 -d joomla`
- `docker ps`

# Troubleshooting

---

- `docker logs <CONTAINER\_ID/NAME>`
- `docker exec -it <CONTAINER\_ID/NAME> bash`
- `docker inspect <CONTAINER\_ID/NAME>`
- `docker stats (monitoring)`
- `docker events`
- `docker diff (handig voor read-only maken)`
- `docker volume prune (opruimen, bespaar diskruimte)`

Monitor je containers.

Verschillende tools mogelijk:

- cAdvisor (geen alerting)
- Prometheus (wel alerting)
- Scout, Datadog (hosted, wel alerting, kost geld)
- ...

Waarschijnlijk goed om de monitoring te integreren in een gemanagede container omgeving (volgende keer meer).

## Up/downloaden van je container/image naar een repository.

---

- `docker login`
- `docker push oscarbuse/hours-web`
- `docker pull oscarbuse/hours-web`

Bij een gewijzigde container eerst:

```
docker commit <container_id/name> <image name>
```



# Voordelen

---

- + Geen dependency issues ("but it worked on my development environment..??").
- + Geen conflicten, isolatie van software.
- + Makkelijk OTAP mogelijk.
- + Lichtgewicht, lage overhead.
- + CLI (docker) en web API.
- + Meer richting een *immutable* omgeving. CM (Configuration Management) groeit in complexiteit.
- + Goed te gebruiken als virtualisatie op virtualisatie: bv. docker containers in een kvm guest. Of in een aws/google omgeving.
- + grote userbase, veel standaard tooling.
- + Goede integratie met andere tooling (voor CI/CD)

# Nadelen

---

- Security..
  - dockerd is (extra) root proces.
  - root in container <-> root op host (..): alleen scheiding door gebruik van namespaces.  
Dit kun je opvangen door de docker daemon te starten met `--userns-remap=default`:

```
# cat << 'EOF' >> /etc/systemd/system/docker.service.d/userns.conf
[Service]
ExecStart=
ExecStart=/usr/bin/dockerd --userns-remap=default -H fd://
EOF
# systemctl daemon-reload
# systemctl restart docker.service
```

Creert wel nieuwe images!
  - Zorg dat processen in de container niet als root runnen.
  - Run dockerd in een dedicated VM.
- Docker bepaald eigenschappen van de container tijdens het bouwen. Het monitort/managed niet.
- Ongebruikte images/volumes kan veel zijn en dient opgeruimd te worden.
- Docker daemon alleen voor Linux.
- Minder geschikt voor *stateful* applicaties (bv. databases).
- Default geen limiet op resources.
- Té dominant?

## Best practices

---

- pas op met "latest".
- pas op met `apt-get update` (duurt lang).
- beperk container interactie (focus op container interactie onderling).
- denk aan patching/upgrading your "base" images.
- pas read-only toe waar mogelijk.
- gebruik TLS voor remote connectie met de docker daemon.
- focus op stateless.
- hou het simpel (iedere container specifieke taak, containers combineren met linking).
- monitor.

## Referenties

---

- <https://www.google.com>
- <https://www.docker.com>
- <https://docs.docker.com>
- <https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-16-04>
- <https://hub.docker.com>

## Vragen?

---

Bier?