

# Systems Advanced: Docker Containers

---

## Docker Introductie

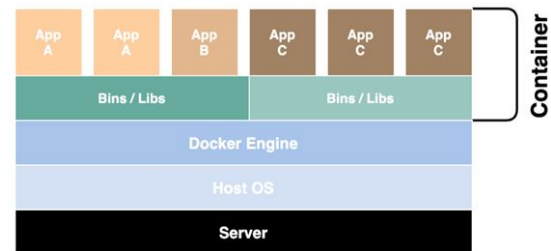


Elfde-Liniestraat 24, 3500 Hasselt, [www.pxl.be](http://www.pxl.be)



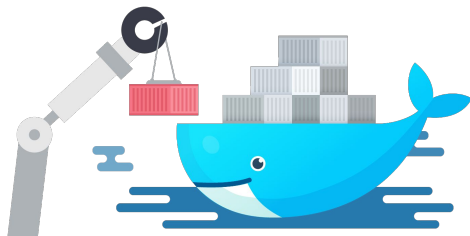
# Wat is een container?

- Een **container image** is
  - een **portable package voor (server) apps**
  - alle componenten die nodig zijn om te draaien, inclusief de Linux-distributie en andere dependencies, zijn verpakt in een enkele image en kunnen worden hergebruikt.
- Een **container** is
  - een **geïsoleerd proces** in een sandbox of jail die de bestanden uit de container image gebruikt
  - **ephemeral**: “vluchtig”, tijdelijke, beperkte lifecycle
  - **immutable**: verandert niet
  - **stateless**: de state verdwijnt nadat de container gestopt is



# Waarom bestaan containers?

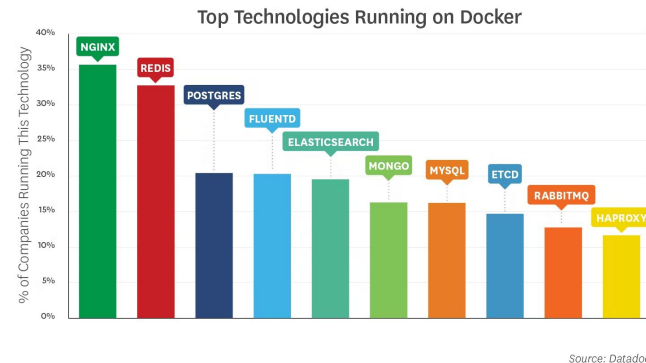
- **Portability**
  - nooit meer "it works on my machine"
- **Scalability**
  - Microservices (cloud-native applications)
  - Machine Learning
- **Agility**
  - Application Deployment (shipping)
  - DevOps CI/CD



# Waar worden containers gebruikt?

Enkele voorbeelden van linux container gebruik

- **Portability**
  - Een MongoDB NoSQL database engine die overal onmiddellijk deployable is.
  - Een Node.js back-end JavaScript runtime environment die overal onmiddellijk deployable is.
- **Scalability**
  - Een onafhankelijke Java microservice met REST API die bv zorgt voor klanten management, binnen een veel groter geheel.
  - Een NGINX web-server, die automatisch **binnen enkele seconden** kan scalen van 5 naar 500 instances, afhangende van het verkeer.
  - Alle mogelijke soorten game servers.
  - ALLE applicaties en services binnen het bedrijf Alphabet en hun Google producten. Google startte 2,000,000,000 ("2 billion") containers per week, **in 2014(!)**.
  - ALLE applicaties en services binnen het bedrijf Meta en hun Facebook/Instagram/WhatsApp/... producten.
  - Zowat alle grote of nieuwe Internet/web applicaties en services ter wereld.
- **Agility**
  - Jouw Java web applicatie, met juist geconfigureerde JVM en alle dependencies, die je makkelijk kan deployen naar PRODUCTION in de cloud met 1 bestand.
  - Een altijd correct geconfigureerde development omgeving voor het hele team.



# Waarom kunnen containers zo goed scalen?

*Waarom gebruiken we niet gewoon VMs op een hypervisor?*

Kort antwoord: omdat een container een geïsoleerd linux proces is, en linux processen starten heel snel.

Een docker container (== proces) opstarten duurt enkele honderden milliseconden. Een VM opstarten duurt vaak meer dan een minuut.

Dus als je snel 100 extra webserver meer of minder nodig hebt, afhankelijk van het gemeten verkeer, is dat met containers absoluut mogelijk.

## Vertical Scaling

( Increase size of instance (RAM , CPU etc.) )



*vroeger*

## Horizontal Scaling

( Add more instances )



Container Architecture

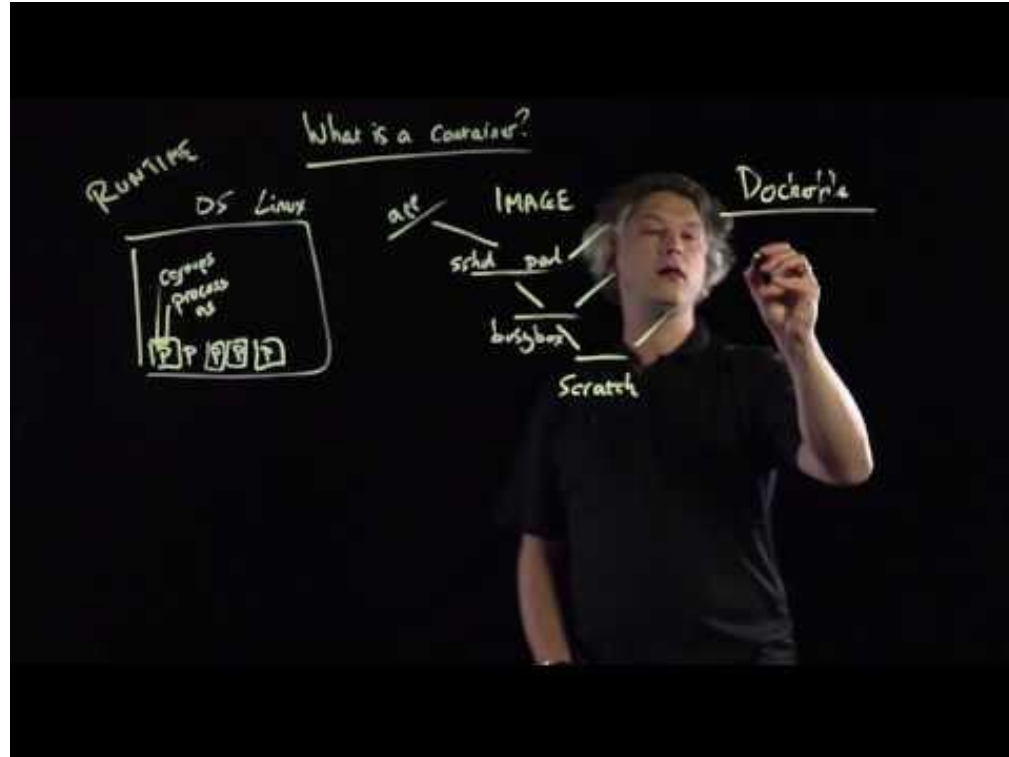
*nu*

# Hoe werken containers in linux?

Ben Corrie

"VMWare Senior Staff Engineer, lead architect behind the Kubernetes integration in vSphere"

<https://youtu.be/EnJ7qX9fkU?t=58> (0:58 tot 4:53)



# Hoe werken containers in linux?

Een geïsoleerd proces in een sandbox / jail

- een container process
- linux kernel support
  - process **namespace**: welke processen kan en mag ik zien en gebruiken?
  - **cgroups**: hoeveel mag ik gebruiken?
    - capabilities
    - resource limits (RAM, CPU)
  - **union filesystem** om container images te gebruiken (zie latere les)
- andere processen kunnen gestart worden vanuit dat ene proces binnen de container

## Linux Kernel: Namespaces

The namespaces are responsible to isolate the workspace for a container. For this, it uses the following namespaces:

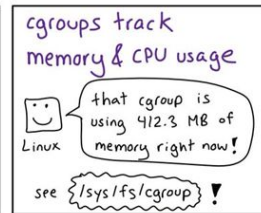
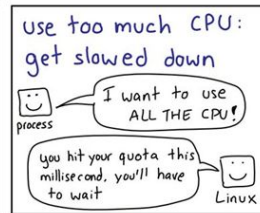
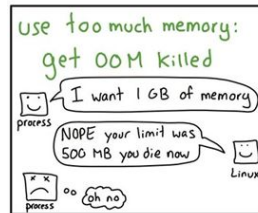
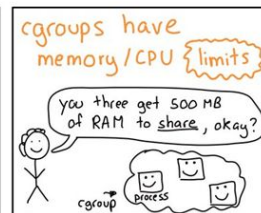
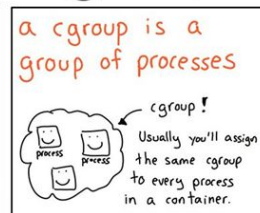
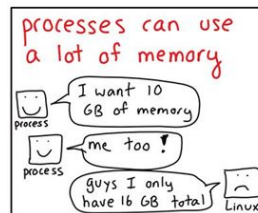
- The *pid* namespace: Process isolation (PID: Process ID).
- The *net* namespace: Managing network interfaces (NET: Networking).
- The *ipc* namespace: Managing access to IPC resources (IPC: InterProcess Communication).
- The *mnt* namespace: Managing filesystem mount points (MNT: Mount).
- The *uts* namespace: Isolating kernel and version identifiers. (UTS: Unix Timesharing System).

ter illustratie

JULIA EVANS  
@b0rk

## cgroups

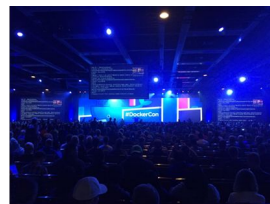
ter illustratie



# "Docker" is een bedrijf met een populaire, open-source, container implementatie



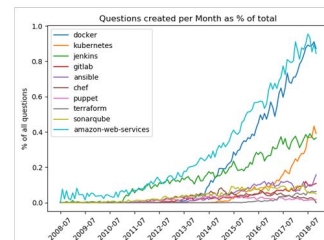
*PyCon, March 2013*



*DockerCon 2014 (Docker 1.0)*



*DockerCon Europe, 2018*



*stackoverflow questions trend 2012-2018*



# Docker: tooling for linux containers

<https://github.com/docker>



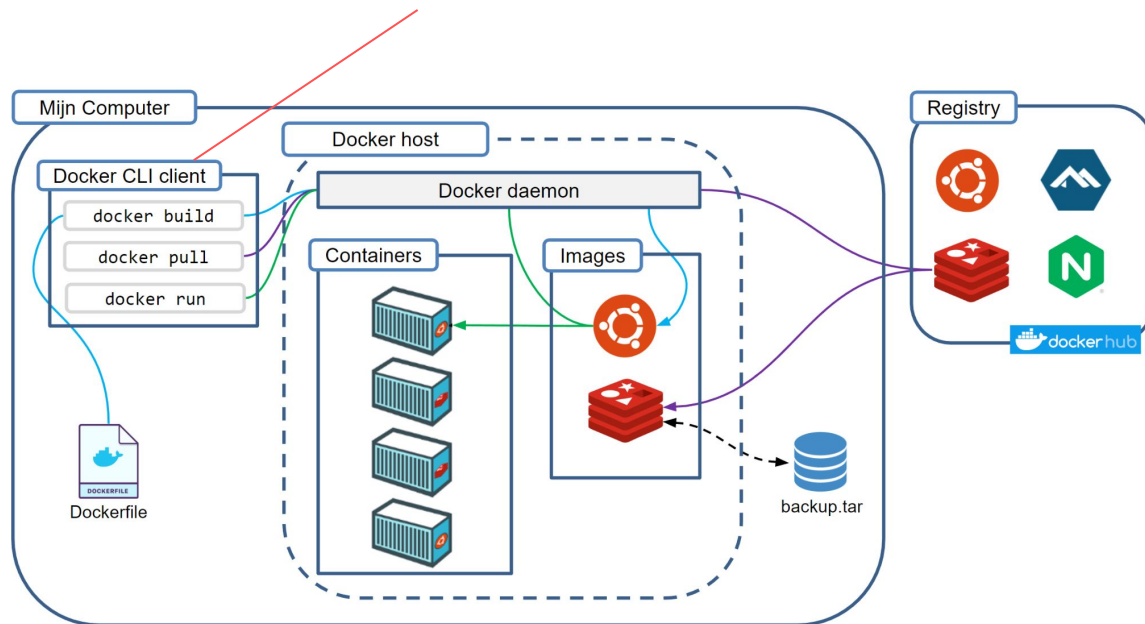
## Docker client ('docker' CLI tool)

- lifecycle management
  - pull
  - create
  - run
  - commit
- network configuration
  - port forwarding
  - overlay networking
- storage configuration
  - (persistent) volumes

<https://docs.docker.com/engine/reference/commandline/cli/>

## Docker Host (linux)

- daemon met exposed API (**dockerd**)
  - ook wel Docker engine of Docker runtime genoemd
  - zorgt ervoor dat er gewerkt kan worden met root-file-systems, process-trees, networking-stacks, enz.
- docker runtime containers
  - Containers zijn running instances van de docker images
- image cache



# Docker: tooling for linux containers

<https://github.com/docker>



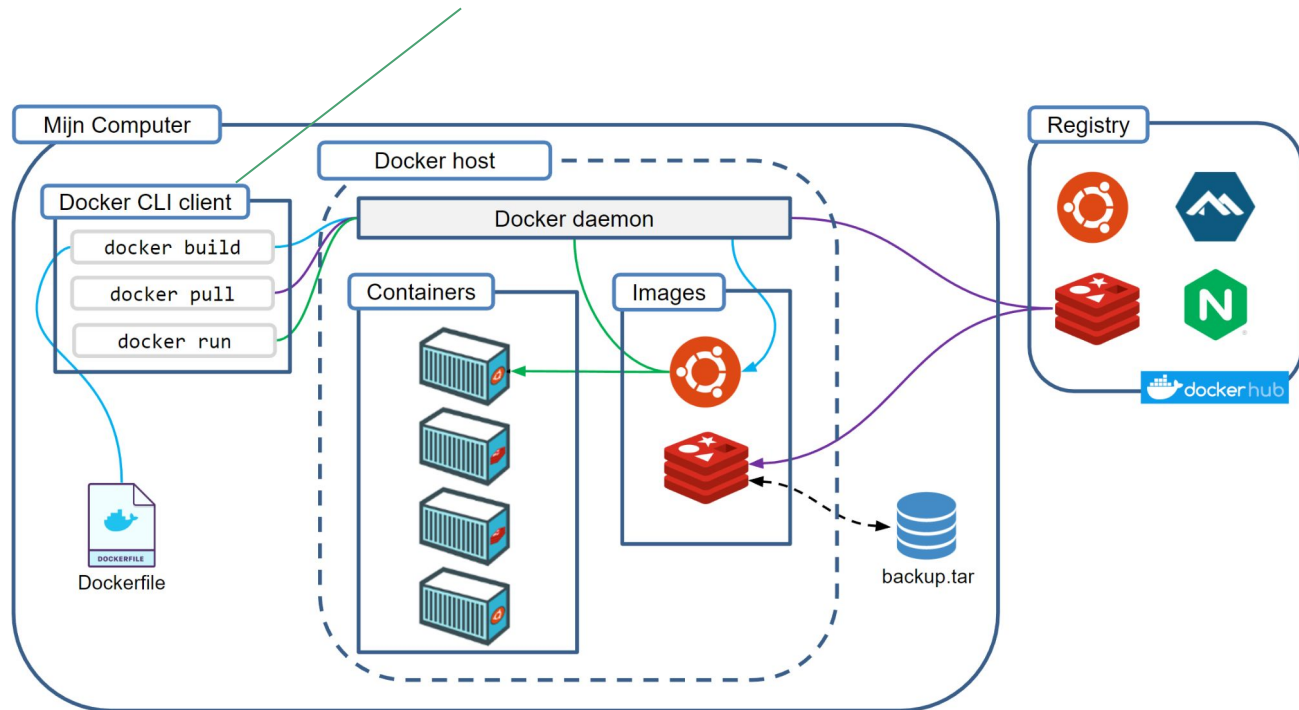
<https://docs.docker.com/engine/reference/commandline/cli/>

## Docker images

- 1 bestand met alle dependencies
- kan makkelijk gedeeld worden
- aangeboden als een union file system (later meer)
- geeft aan met welk OS een container moet worden uitgevoerd
- een container is de runtime instance van een container image

## Registry voor docker images

- <https://hub.docker.com/>
- bevat onder andere de repositories, verzamelingen van images, voor Ubuntu, Fedora, etc
- de Ubuntu repo bvb bevat dan images van de verschillende versies van Ubuntu
- ondersteunt pull en push
- (zie volgende les)



# Hoe gaan we docker gebruiken in deze lessen?

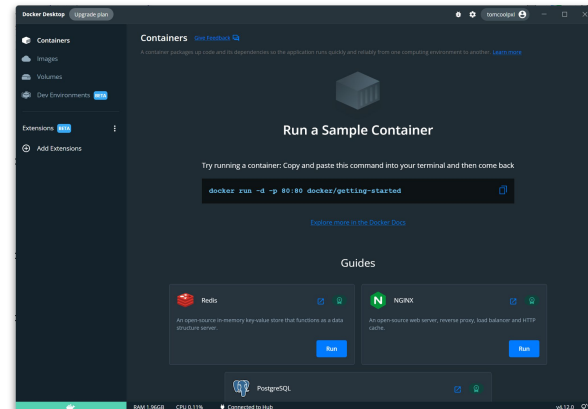
- Main use cases
  - Application development op een desktop operating system (Windows, macOS, Linux)
    - Containers worden gebouwd dmv configuratie files.
    - Werken met containers dmv Docker CLI en Docker Desktop.
    - Op non-linux desktop operating systems wordt Docker aangeboden via een verborgen, geïntegreerde hypervisor (Windows Hyper-V of macOS Hypervisor)
  - Deployment op (cloud) infrastructure
    - Containers worden gedownload via een private/public registry (zie volgende les).
    - Containers worden beheerd door een container orchestration system in de cloud (bv. [Kubernetes](#)), of door de cloud provider zelf (bv. [AWS Fargate](#)). Die technologieën zijn geen onderdeel van dit OLOD.

Wij leren in de eerste plaats werken met Docker CLI, maar de Docker Desktop GUI wordt af en toe gebruikt als illustratief tool.

De installatie van docker op een ubuntu VM is verplicht. De installatie van Docker Desktop op Windows is optioneel.

# Installatie op een desktop OS

- Docker Desktop
  - bevat Docker CLI en Docker Desktop GUI (alsook Kubernetes voor lokaal gebruik)
  - beschikbaar voor Windows / macOS / Linux (Ubuntu, debian, Fedora)
  - In Windows is dezelfde docker CLI ook beschikbaar op geïnstalleerde linux distributies die draaien op WSL 2
- Docker Desktop for Windows Requirements
  - Windows 11 (aangeraden) of Windows 10 (vanaf 21H1)
  - *Hyper-V and Containers Windows features must be enabled.*
  - *WSL 2 must be enabled.*
  - *Zie link installatie procedure*
- Docker Desktop for Windows installatie
  - <https://docs.docker.com/desktop/install/windows-install/>
  - Optioneel: command-line auto completion for PowerShell
    - <https://github.com/matt9ucci/DockerCompletion>



# Installatie op een Ubuntu VM

Je kan de lessen ook volgen met enkel een Ubuntu (VM) waar docker op geïnstalleerd is.

We maken eerst een nieuwe Ubuntu VM aan OF we gebruiken 1 van onze beschikbare server VMs.  
Opmerking: vergeet geen snapshot te maken voor de docker installatie.

- Maak een nieuwe Virtuele Machine aan
  - gebaseerd op de ISO van Ubuntu Live Server 22.04
  - 4 GB geheugen
  - 2 Processoren
  - 64 GB Harde schijf
  - NAT-netwerk
  - Standaard installatie
    - met SSH
    - een gebruiker, genaamd "student"
    - een hostname "docker"

# Installatie op een Ubuntu VM

De installatie kan uitgevoerd worden via een script dat je downloadt.

zie <https://docs.docker.com/engine/install/ubuntu/#install-using-the-convenience-script>

We downloaden het convenience script en voeren het dan uit met sudo rechten

- `curl -fsSL https://get.docker.com -o get-docker.sh`
- `sudo sh get-docker.sh`

We zorgen er voor dat we als gewone user ook met docker kunnen werken (zonder sudo telkens te moeten gebruiken)

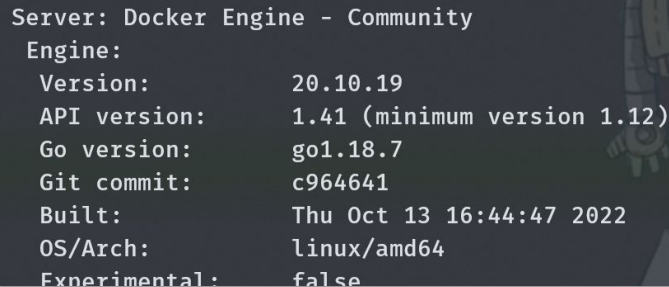
- `sudo usermod -aG docker $USER`

We loggen uit en terug in om de group membership te activeren in onze shell

- `exit`
- `ssh <username>@<ip_address>`

Test

- `docker version`

A terminal window with a dark background and light-colored text. It displays the output of the 'docker version' command. The text is as follows:

```
Server: Docker Engine - Community
Engine:
 Version:      20.10.19
 API version:  1.41 (minimum version 1.12)
 Go version:   go1.18.7
 Git commit:   c964641
 Built:        Thu Oct 13 16:44:47 2022
 OS/Arch:      linux/amd64
 Experimental: false
```

# Rechten via de docker group

Waarom wordt de group **docker** gebruikt bij de installatie?

Docker heeft normaal gezien root nodig omdat het namespaces, cgroups, networks moet aanmaken en aanpassen.

De docker daemon luistert naar een bepaalde socket: `/var/run/docker.sock` of `/run/docker.sock` (of beide, afhankelijk van de distributie)

Dus als we kijken met `ls -l /run/docker.sock` zien we de rechten.

```
student@ubuntu-server-2:~$ ls -l /run/docker.sock
srw-rw--- 1 root docker 0 Oct 16 08:18 /run/docker.sock
student@ubuntu-server-2:~$ ls -l /var/run/docker.sock
srw-rw--- 1 root docker 0 Oct 16 08:18 /var/run/docker.sock
student@ubuntu-server-2:~$ _
```

We zien dat de groep docker ook rechten heeft op deze socket. Dus kunnen we ook als gewone gebruiker inloggen en zonder sudo alle docker-commando's uitvoeren, zolang we maar in de docker group zitten.

# Mijn eerste container applicatie runnen

Indien we een container willen opstarten die we nog niet bezitten, wordt de overeenkomstige docker-image eerst automatisch gedownload van Docker-Hub.

## docker run hello-world

De applicatie zorgt voor deze output. Eens de output gedaan is, heeft de applicatie ook gedaan. Daarom wordt de container dan ook afgesloten.

zie <https://docs.docker.com/engine/reference/commandline/docker/> voor reference documentatie van het docker cli commando.

```
# thraa @ DESKTOP-TOMC in ~ [00:33:10]
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:18a657d0cc1c7d0678a3fba8b7eb4918bba25968d3e1b0adebfa71caddbc346
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/

# thraa @ DESKTOP-TOMC in ~ [00:33:17]
$ _
```



# Een overzicht van de containers

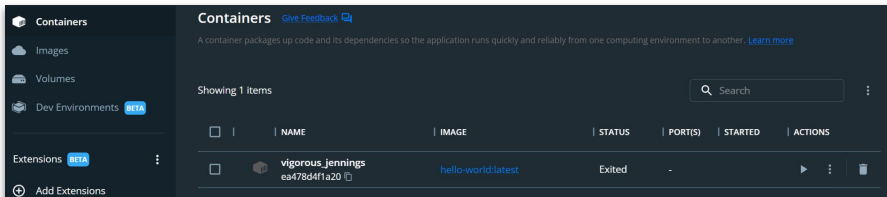
Om de containers te bekijken die aan het draaien zijn

- `docker container ls` OF `docker ps`

Om alle containers te bekijken (ook die gestopt zijn)

- `docker container ls -a` OF `docker ps -a`

In de Docker Desktop GUI kan je dat via de Containers tab ook zien.



```
# thraa @ DESKTOP-TOMC in ~ [00:34:18]
$ docker container ls
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES

# thraa @ DESKTOP-TOMC in ~ [00:34:21]
$ docker container ls -a
CONTAINER ID   IMAGE     COMMAND   CREATED        STATUS      PORTS   NAMES
ea478d4f1a20   hello-world   "/hello"   About a minute ago   Exited (0)   About a minute ago   vigorous_jennings

# thraa @ DESKTOP-TOMC in ~ [00:34:33]
$ _
```

# Een overzicht van de container images

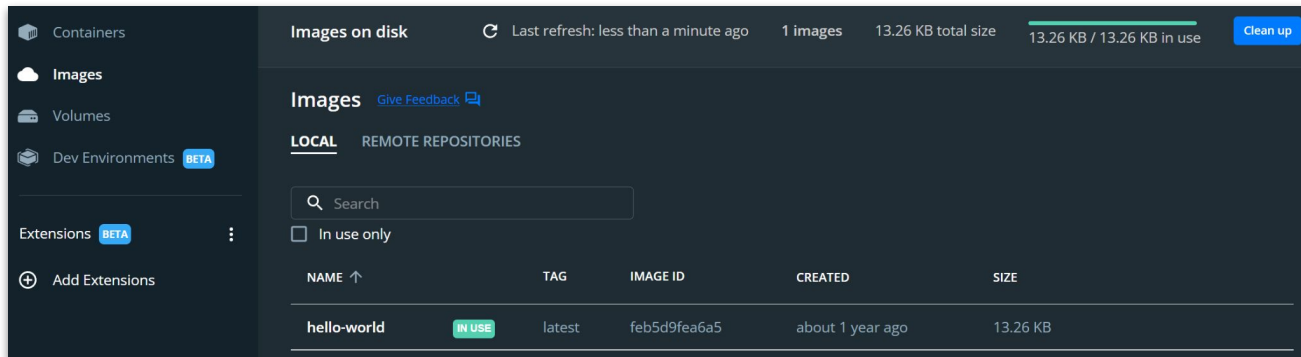
Om de lijst van gedownloade images te bekijken

- `docker image ls` OF `docker images`

In de Docker Desktop GUI kan je dat ook in de Images tab zien.

```
# thraa @ DESKTOP-TOMC in ~ [00:34:33]
$ docker image ls
REPOSITORY      TAG       IMAGE ID       CREATED        SIZE
hello-world     latest    feb5d9fea6a5   12 months ago  13.3kB

# thraa @ DESKTOP-TOMC in ~ [00:39:39]
$ _
```



# Help!

We kunnen altijd  
in de help gaan  
kijken.

```
# thraa @ DESKTOP-TOMC in ~ [00:41:55]
$ docker --help
```

Usage: docker [OPTIONS] COMMAND

A self-sufficient runtime

Options:

--config string

-c, --context string

-D, --debug

-H, --host list

-l, --log-level string

--tls

--tlscacert string

--tlscert string

--tlskey string

--tlsverify

-v, --version

```
# thraa @ DESKTOP-TOMC in ~ [00:43:04]
$ docker image --help
```

Usage: docker image COMMAND

Manage images

Commands:

build	Build an image from a Dockerfile
history	Show the history of an image
import	Import the contents of a directory as a new image
inspect	Display detailed information on one or more images
load	Load an image from a tar archive
ls	List images
prune	Remove unused images
pull	Pull an image or a repository from a registry
push	Push an image or a repository to a registry
rm	Remove one or more images
save	Save one or more images as a tar archive
tag	Create a tag TARGET_IMAGE that refers to SRC_IMAGE

Run 'docker image COMMAND --help' for more information on a command

```
# thraa @ DESKTOP-TOMC in ~ [00:43:04]
$ -
```

```
# thraa @ DESKTOP-TOMC in ~ [00:44:01]
$ docker image ls --help
```

Usage: docker image ls [OPTIONS] [REPOSITORY[:TAG]]

List images

Aliases:

ls, list

Options:






-a, --all	Show all images (default hides intermediate images)
--digests	Show digests
-f, --filter filter	Filter output based on conditions provided
--format string	Pretty-print images using a Go template
--no-trunc	Don't truncate output
-q, --quiet	Only show image IDs

```
# thraa @ DESKTOP-TOMC in ~ [00:44:05]
$ -
```

# TAB completion

Je kan ook altijd gebruik maken van [TAB] auto-completion.

*Opmerking: de Windows auto-completion scrollt 1 voor 1 door de mogelijke commando's ipv ze in een handige lijst te tonen zoals op linux.*

```
student@ubuntu-server-2:~$ docker     
attach    context  help      kill      node      rename    secret    system    volume  
build     cp       history   load      pause     restart   service   tag       wait  
builder   create   image     login     plugin    rm        stack     top  
commit    diff     images    logout    port      rmi       start     trust  
compose   events   import    logs      ps        run       stats     unpause  
config    exec     info      manifest  pull      save      stop      update  
container export    inspect   network   push      search    swarm     version  
  
student@ubuntu-server-2:~$ docker image    
build     import    load      prune     push      save  
history    inspect   ls        pull      rm        tag  
  
student@ubuntu-server-2:~$ docker image ls  
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE  
hello-world   latest    feb5d9fea6a5   12 months ago  13.3kB  
  
student@ubuntu-server-2:~$
```

# Lesmateriaal via github

Sommige docker lessen gebruiken lesmateriaal zoals Dockerfiles en gebruikte source code, dat je kan terugvinden in onze github repo <https://github.com/PXL-Systems-Advanced/docker-lessen>.

Clone deze repo naar je laptop

- `git clone https://github.com/PXL-Systems-Advanced/docker-lessen.git`

Git installatie op Windows

- <https://scoop.sh>
- `scoop install git`

Git installatie op Ubuntu

- `sudo apt install -y git`

# DEMO

Met docker kan je heel makkelijk (server) software installeren zonder je iets te moeten aantrekken van linux distributie, dependencies, downloads, ...

- [Minetest](#)
- <https://github.com/minetest/minetest#Docker>

Download, installeer en run de multiplayer server met 1 commando:

```
docker run -it -p 30000:30000/tcp -p 30000:30000/udp registry.gitlab.com/minetest/minetest/server:latest
```

Download, pak uit en run de `bin/minetest.exe` client:

- <https://www.minetest.net/downloads/>
- selecteer "Join spel", gebruik `127.0.0.1` als adres, `30000` als poort, typ een speler naam en druk op "Register"

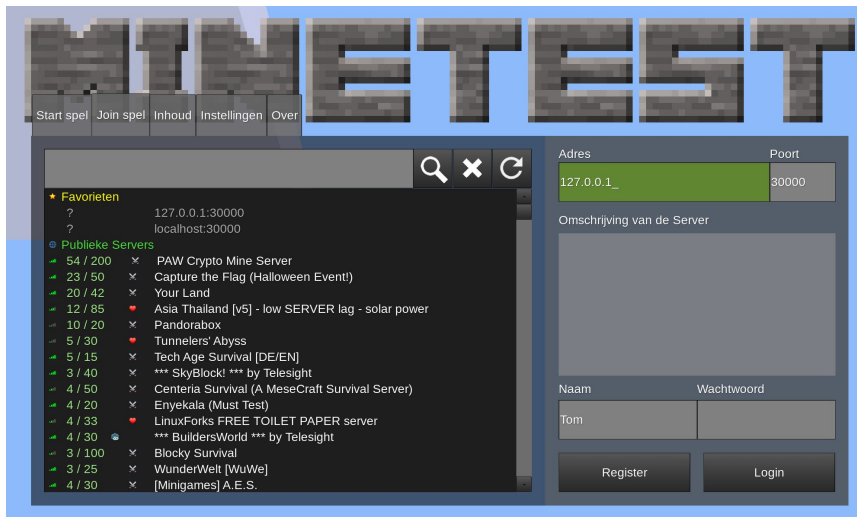
Als je de poorten `30000/tcp` en `30000/udp` zou openzetten op je laptop zou iedereen in het lokale netwerk je multiplayer server kunnen bereiken.

Je zou deze server software ook kunnen installeren op een klein, goedkoop, low-powered linux servertje zoals een Raspberry Pi en zelfs beschikbaar maken via het internet.

Je kan je ook indenken dat het heel makkelijk is om zo'n docker server later naar een cloud machine of platform te deployen, zolang die machine of dat platform docker containers kan runnen.

```
# thraa @ DESKTOP-TOMC in ~ [23:37:24]
$ docker run -p 30000:30000/tcp -p 30000:30000/udp registry.gitlab.com/minetest/minetest/server:latest

2022-10-18 21:38:25: ACTION[Main]: World at [/var/lib/minetest/.minetest/worlds/world]
2022-10-18 21:38:25: ACTION[Main]: Server for gameid="minetest" listening on 0.0.0.0:30000.
2022-10-18 21:38:43: ACTION[Server]: Tom [172.17.0.1] joins game. List of players: Tom
```



# Het eerste grote docker succes was een game server

De grote cloud providers gebruiken al minstens sinds 2011 container technologie ipv VMs voor hun services. (google search, gmail, youtube, facebook, ...)

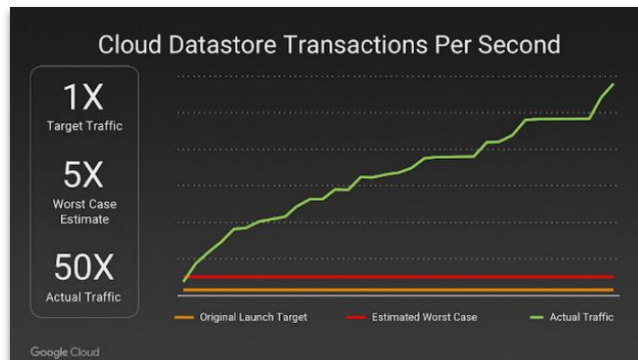
De Google Kubernetes 1.0 release vond plaats in 2015. Met kubernetes kan je grote verzamelingen docker containers beheren en scalen.

Pokémon GO van Niantic was het eerste grote software product in 2015 dat met docker en kubernetes werd gedeployed. Het plotse wereldwijde succes van het game overtrof alle voorspellingen en bezorgde de ontwikkelaars slapeloze nachten.

Gelukkig bleek supersnelle scaling met docker containers perfect te werken: het game had meer dan 500 miljoen downloads en kon heel snel upscalen om meer dan 20 miljoen Daily Active Users te verwerken. 50x meer dan het voorspelde worst case scenario.

Het was meteen ook het eerste docker/kubernetes succesverhaal.

Game servers zijn niet meer dan real-time gedistribueerde databases met extra logic en dus zeer geschikt voor containerization, zoals alle client-server web/internet software. Dat geldt dus ook voor de full-stack software die je in PXL leert ontwikkelen.



[Catch 'em all with Kubernetes: Pokemon Go case Study](#)

# Oefening: Werken met containers

- Maak een nieuwe Virtuele Machine aan.
  - gebaseerd op de ISO van Ubuntu Live Server 22.04
  - 4 GB geheugen
  - 2 Processoren
  - 64 GB Harde schijf
  - NAT -netwerk
  - Standaard installatie
    - met SSH
    - een gebruiker, genaamd "student"
    - een hostname "docker"
- Installeer Docker en zorg er voor dat gewone gebruikers Docker-commando's kunnen uitvoeren.
- Start een getting-started container:
  - `docker run -d -p 80:80 docker/getting-started`
- Zoek uit wat volgende commando's doen en voer ze uit.
  - `docker system info`
  - `docker system df`
  - `docker system events` (hiervoor heb je een tweede tty nodig)
- Zoek uit welke informatie je kan vinden over de container in de Docker Desktop applicatie, kijk een beetje rond in de applicatie.
- Surf naar `http://<vm_ip_address>:80/` en kijk eens rond
- Stop en verwijder de container via de cli. Zoek daarvoor eerst het CONTAINER ID op.
  - `docker container ls -a`
  - en gebruik dat ID om de container te stoppen en te verwijderen
  - `docker container rm <CONTAINER_ID>`

## Getting Started

### The command you just ran

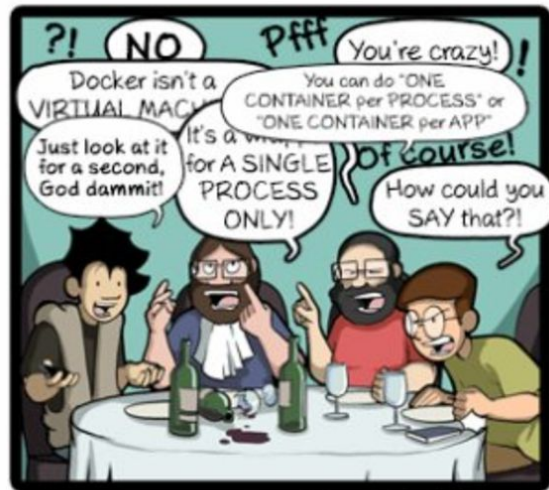
Congratulations! You have started the container for this tutorial! Let's first explain the command that you just ran. In case you forgot, here's the command:

```
docker run -d -p 80:80 docker/getting-started
```

You'll notice a few flags being used. Here's some more info on them:

- `-d` - run the container in detached mode (in the background)
- `-p 80:80` - map port 80 of the host to port 80 in the container
- `docker/getting-started` - the image to use





CommitStrip.com