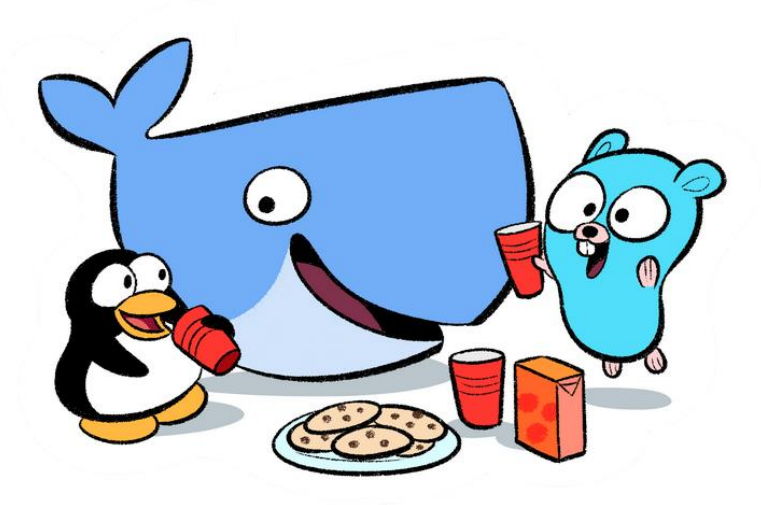
AIM - ICT

DOCKER



Workshop

Jaap Papavoine

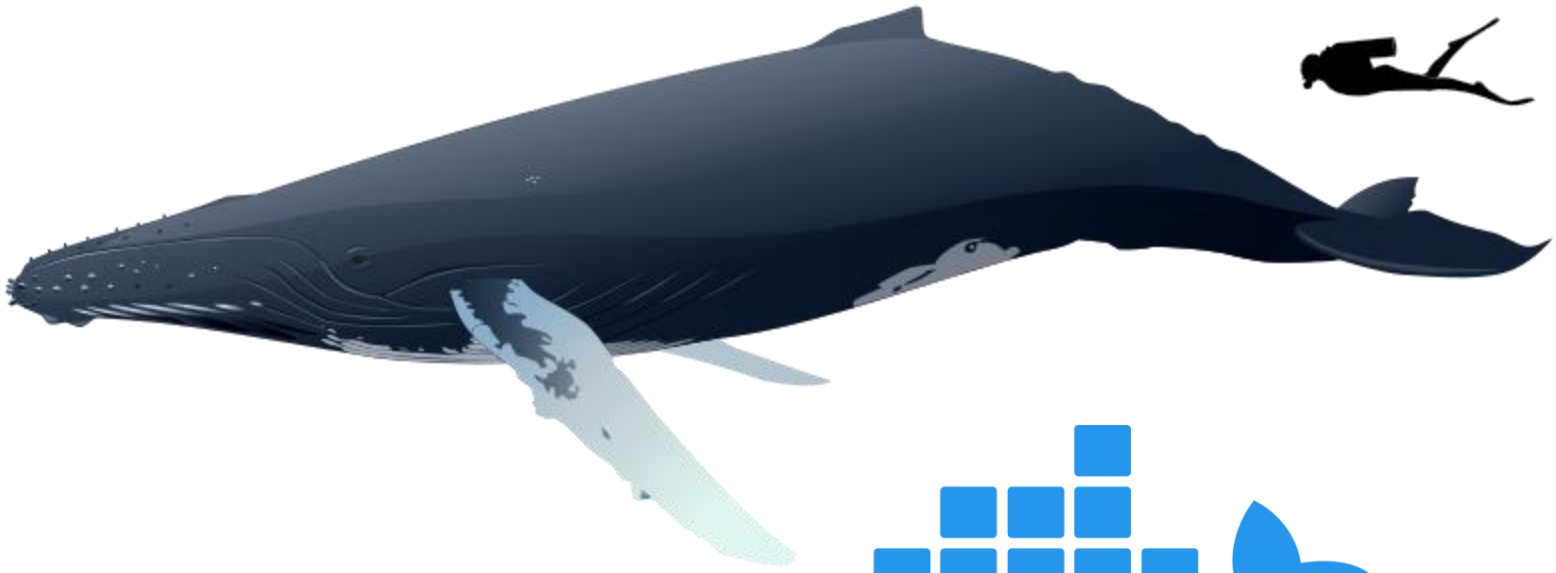
HAN_UNIVERSITY
OF APPLIED SCIENCES

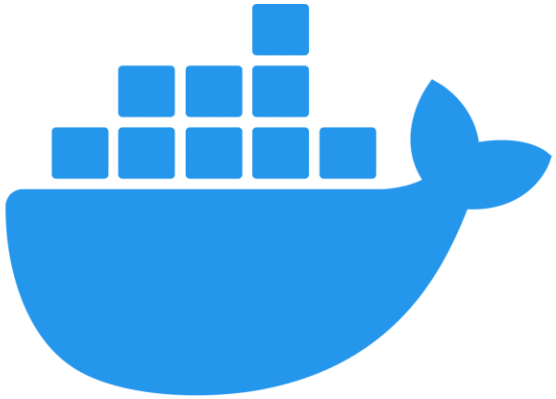
Image source: jaap.papavoine@notagreatdronepilot.nl

Image source: dx.com

Image source: commons.wikimedia.org

Image source: commons.wikimedia.org





docker®

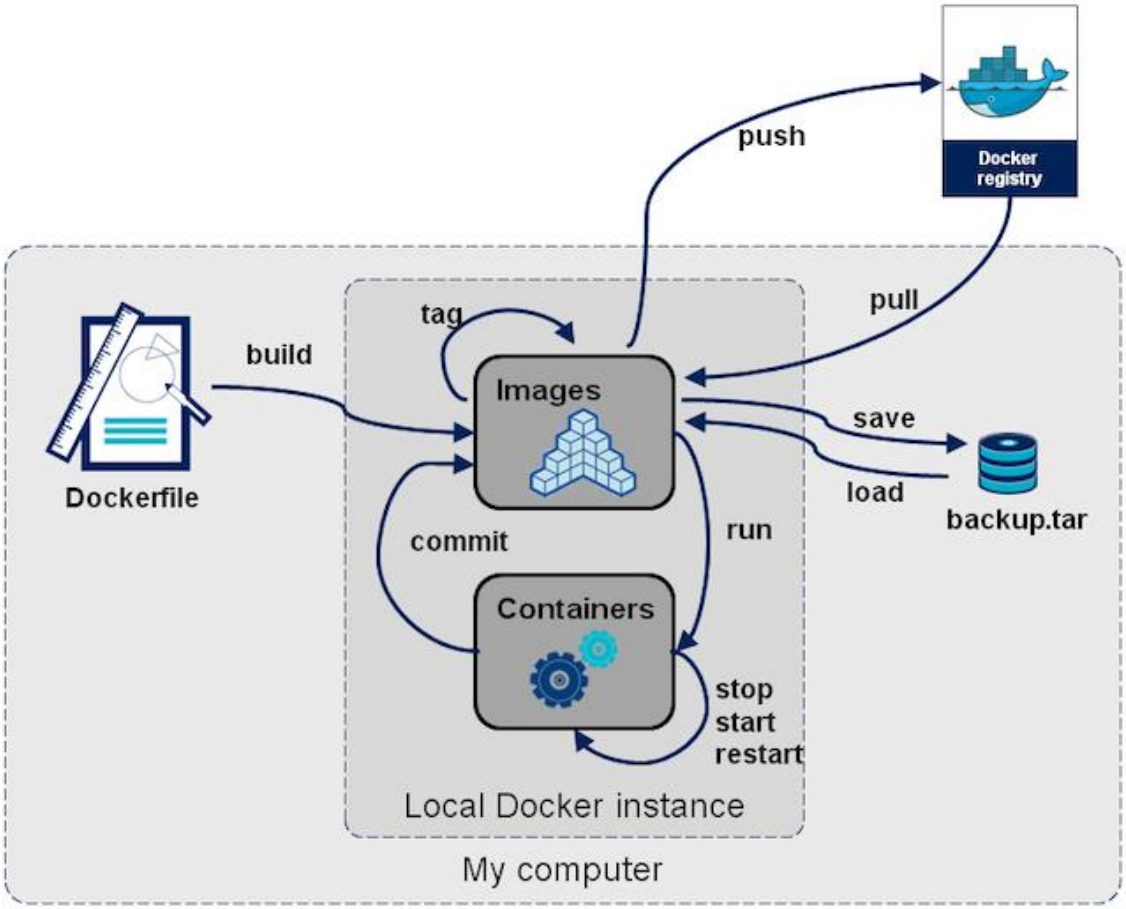
Image source: commons.wikimedia.org, docker.com

The real value of Docker is not technology, it's getting people to agree on something.

Solomon Hykes, Docker founder

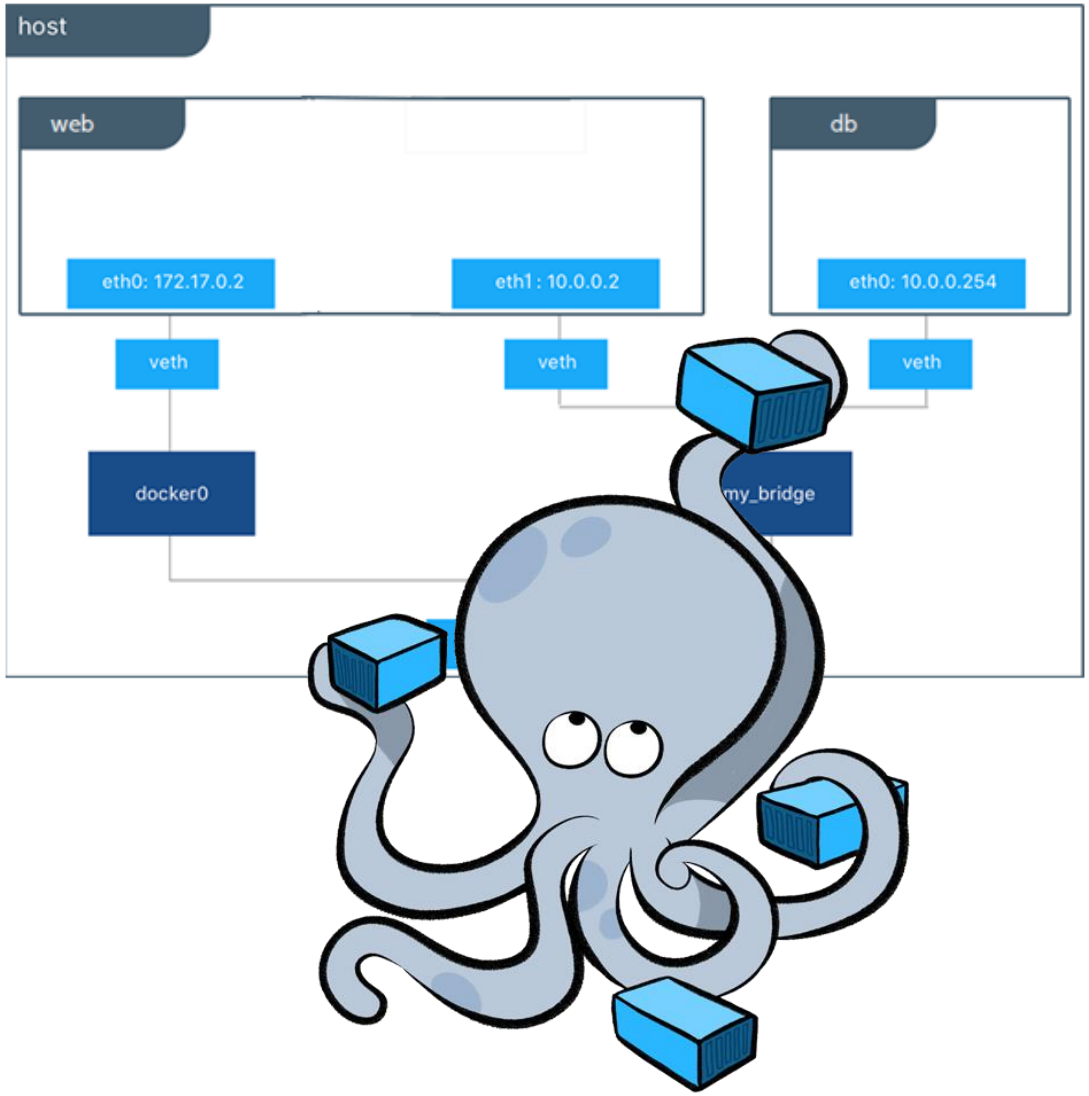
Docker Basics

- Container basics
  - Run, interactive/daemon
  - Stop, restart, remove.
  - Names / id's
  - Host / Container networking
  - Behind the scenes
- Images
  - Create images
    - Using docker commit
    - Using docker build and Dockerfile
  - Docker Hub
  - Base Image
  - Layer basics



Docker Advanced

- Networking
- Compose
- Multistage



AIM - ICT

Containers

Stop! Demo time

Watisdat?

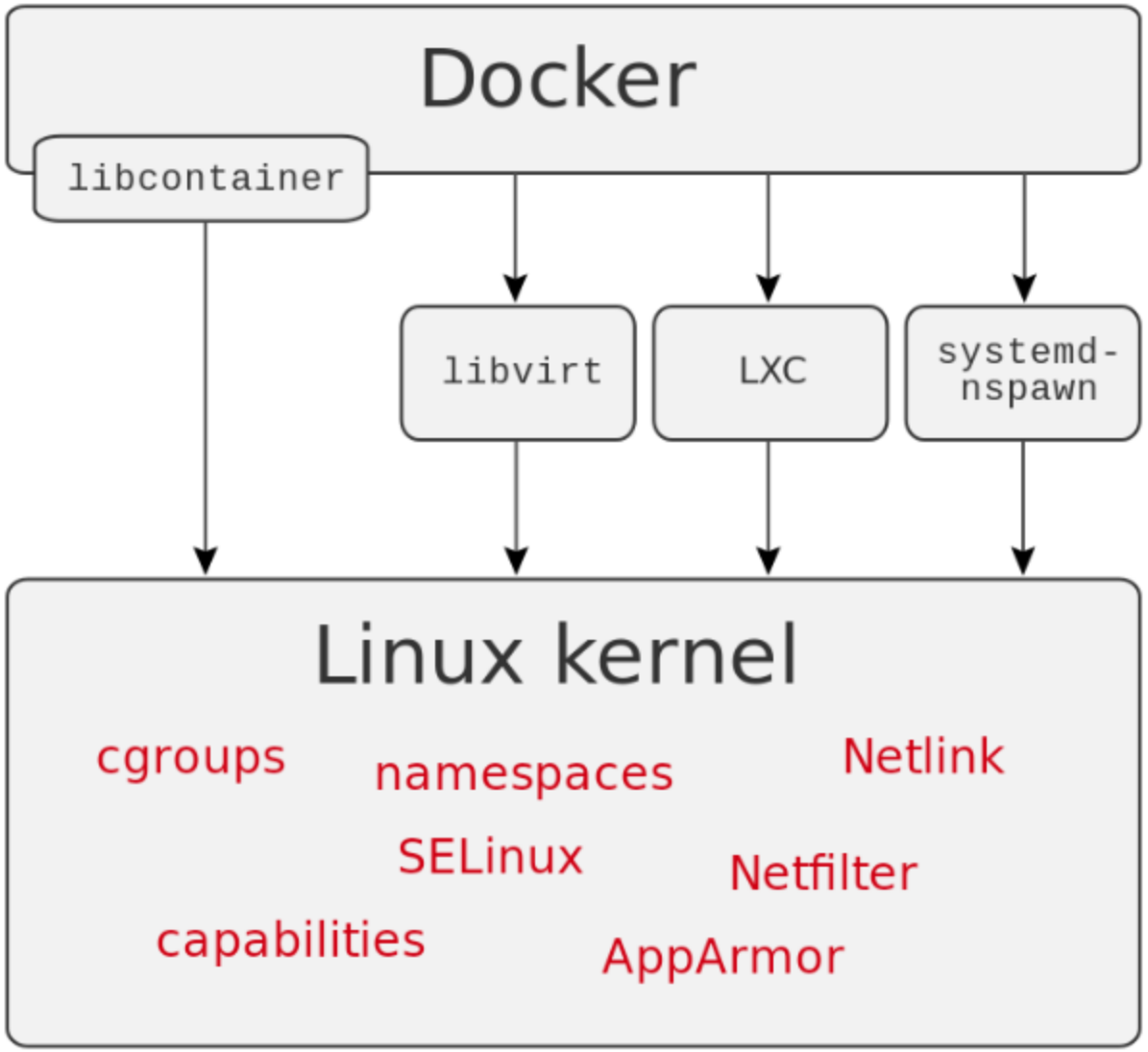


Image source: docker.com

Process isolation with namespaces

Inside container running nginx webserver:

```
root@3e0e00091bd3:/# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 09:46 ?        00:00:00 nginx: master process nginx -g daemon off;
nginx       32     1  0 09:46 ?        00:00:00 nginx: worker process
nginx       33     1  0 09:46 ?        00:00:00 nginx: worker process
nginx       34     1  0 09:46 ?        00:00:00 nginx: worker process
nginx       35     1  0 09:46 ?        00:00:00 nginx: worker process
nginx       36     1  0 09:46 ?        00:00:00 nginx: worker process
nginx       37     1  0 09:46 ?        00:00:00 nginx: worker process
nginx       38     1  0 09:46 ?        00:00:00 nginx: worker process
nginx       39     1  0 09:46 ?        00:00:00 nginx: worker process
nginx       40     1  0 09:46 ?        00:00:00 nginx: worker process
nginx       41     1  0 09:46 ?        00:00:00 nginx: worker process
nginx       42     1  0 09:46 ?        00:00:00 nginx: worker process
nginx       43     1  0 09:46 ?        00:00:00 nginx: worker process
root        44     0  0 10:06 pts/0    00:00:00 /bin/bash
root       389    44  0 10:06 pts/0    00:00:00 ps -ef
root@3e0e00091bd3:/#
```

Process isolation with namespaces

OUTSIDE container running nginx webserver:

```
  868 root      0:00 /usr/bin/containerd-shim-runc-v2 -namespace services.lir
  888 root      0:00 /usr/bin/vpnkit-forwarder -data-connect /run/host-servic
  920 root      0:00 /usr/bin/containerd-shim-runc-v2 -namespace services.lir
  936 root      0:00 /bin/sh
  942 root      0:00 /sbin/vpnkit-tap-vsockd --post-up-script /etc/network/po
 1010 root      0:16 /sbin/vpnkit-tap-vsockd --post-up-script /etc/network/po
 1015 root      0:00 /usr/bin/logwrite -n dockerd /usr/local/bin/dockerd --co
 1020 root      0:36 /usr/local/bin/dockerd --containerd /var/run/desktop-cor
 1386 root      0:00 /usr/bin/containerd-shim-runc-v2 -namespace moby -id 3e0
 1407 root      0:00 nginx: master process nginx -g daemon off;
 1470 101       0:00 nginx: worker process
 1471 101       0:00 nginx: worker process
 1472 101       0:00 nginx: worker process
 1473 101       0:00 nginx: worker process
 1474 101       0:00 nginx: worker process
 1475 101       0:00 nginx: worker process
 1476 101       0:00 nginx: worker process
 1477 101       0:00 nginx: worker process
 1478 101       0:00 nginx: worker process
 1479 101       0:00 nginx: worker process
 1480 101       0:00 nginx: worker process
 1481 101       0:00 nginx: worker process
 1800 root      0:00 /usr/bin/containerd-shim-runc-v2 -namespace moby -id 04c
 1820 root      0:03 /usr/local/openjdk-11/bin/java -Djava.util.logging.confi
 1900 root      0:00 /init
 1901 root      0:00 /init
 1902 root      0:00 /bin/bash
```

Resource sharing with **cgroups**

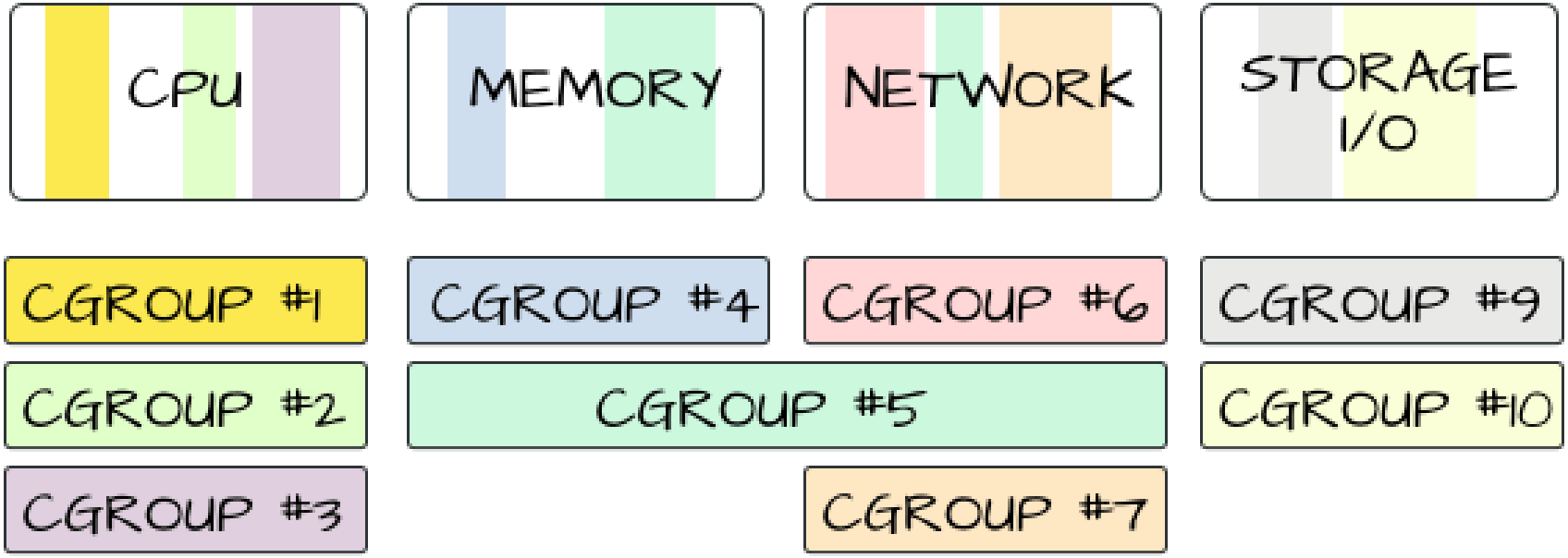


Image source: medium.com

Resc

Doing cgroups the hard way

For simplicity, I will generate a load on the system by running:

Notice that the directories are automatically populated by the controller:

As you see in the screenshot above, the process in the new cgroup receives

We now see that the weighting has taken effect, with the cgroup **user1** taking up about 61% of the CPU time:

$$\frac{2048}{768 + 512 + 2048} = 61.5\%$$

The remaining time is split between **user2** and **user3**.

There are, of course, several problems with our test setup.

1. These are all created by hand. What happens if the process you are putting into a cgroup changes its PID?
2. The custom files and folders created will not survive a reboot.
3. This is a lot of manual work. Where is the tooling?

Have no fear, my friends, systemd has you covered.

***[Free online course:** Red Hat Enterprise Linux technical overview.]*

Image source: re

Network isolation with virtual networks

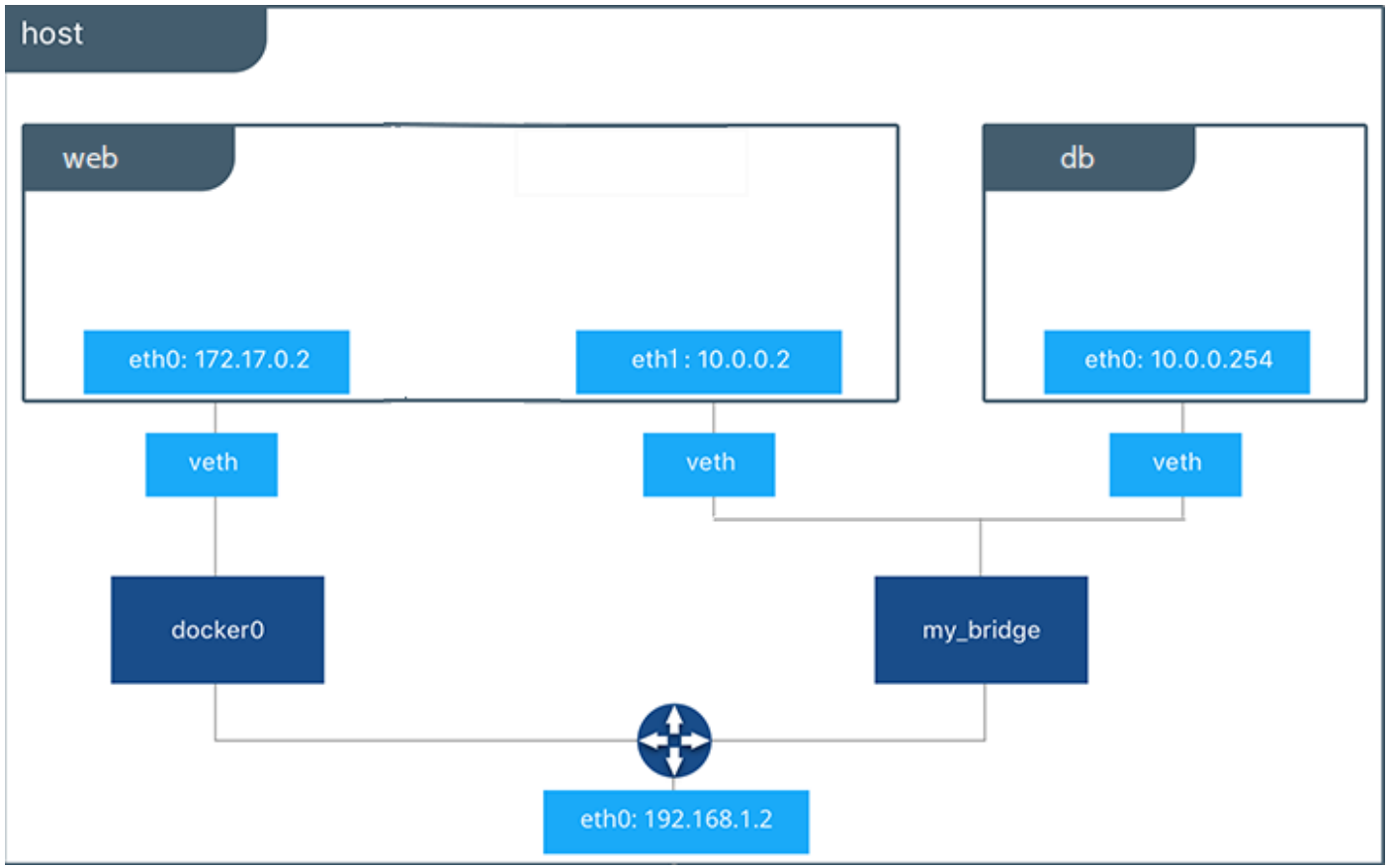
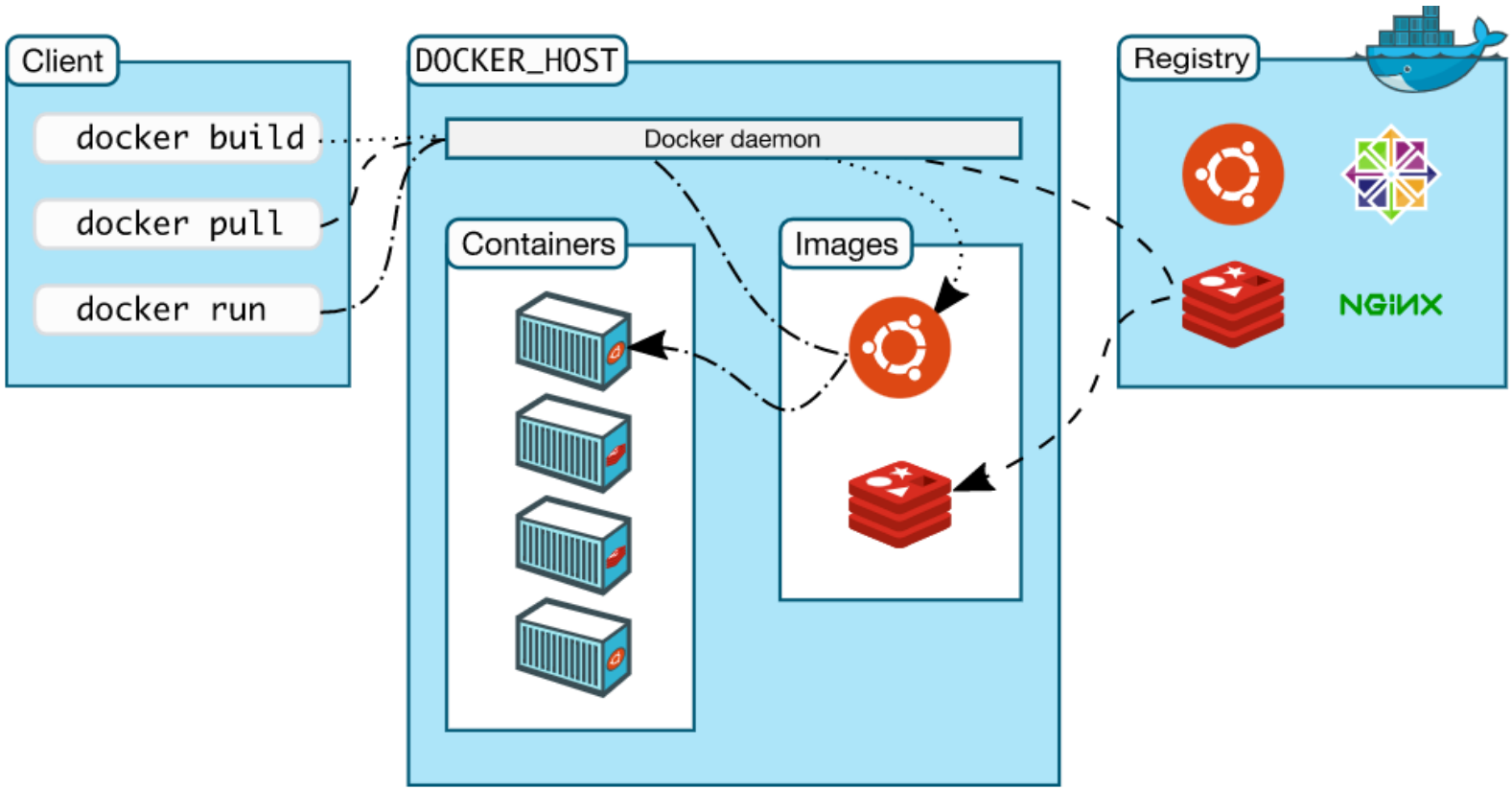


Image source: docker.com

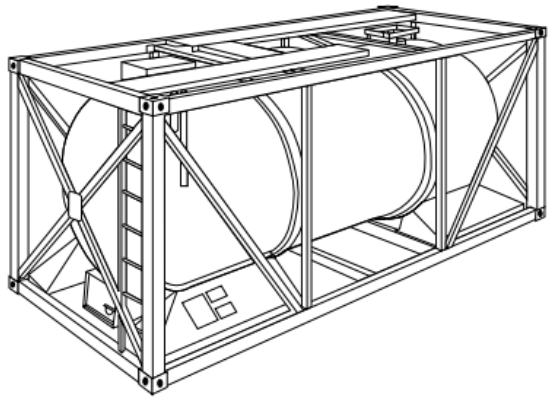
Docker client



Shipping container ‘API’ / spec

TANK CONTAINER

ISO Size Type Group Codes: 20KL, 22KL



- Hapag-Lloyd can provide tank containers which are approved to the highest standards. Depending on the characteristics of the products to be carried, the requirements vary. Hapag-Lloyd offer their services on operational, technical and regulatory questions
- Separate tank fleets are available for:

  FOODSTUFFS, e.g.:
  - Alcohols
  - Fruit juices
  - Edible oils
  - Food additives

  CHEMICAL PRODUCTS, e.g.:
  - Flammables
  - Oxidising agents
  - Toxic substances
  - Corrosives

- Tanks must be filled to not less than 80 % of their capacity to avoid dangerous surge/ swell during transport
- Tanks must not be filled to 100 % of their capacity. Sufficient ullage space shall be left – which must be determined depending on the thermal expansion of the product to be carried
- Certain dangerous products must be carried in tanks having no openings below the surface level of the liquid. Such tanks must be discharged through a syphon pipe by either pressure or pumping

- National road/rail weight limitations to be maintained when arranging la transports
- For the cleaning of tanks and dispos of residues, dedicated rules apply

Docker container images

Freight container

- Cargo!
- ID label
- Destionation label
- Hazard stickers
- Cargo description
- Hooks
- Handles

ALL STANDARDIZED!

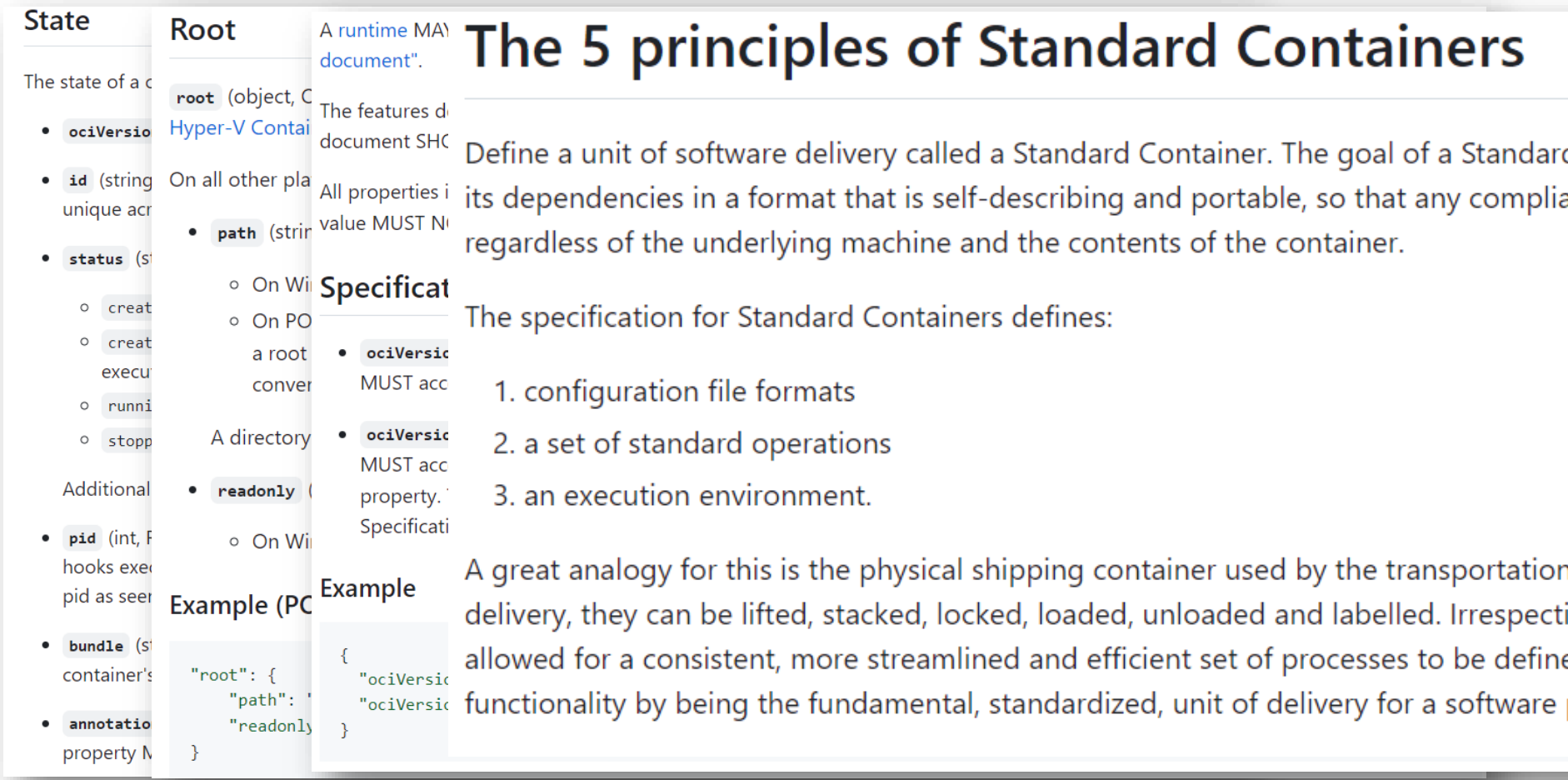
Software (container) image

- Filesystem
- ID
- Origin
- Parameters description
- Start/Stop scripts
- Data hooks
- Network port nrs

ALL STANDARDIZED!

Software container API / Spec

https://github.com/opencontainers/runtime-spec/blob/main/spec.md



State

The state of a c

- `ociVersio`
- `id` (string unique acr
- `status` (st
  - `creat`
  - `creat` execut
  - `runni`
  - `stopp`

Additional

- `pid` (int, F hooks exec pid as seer
- `bundle` (st container's
- `annotatio` property M

Root

`root` (object, C Hyper-V Contai

On all other pla

- `path` (strin
  - On Wi
  - On PO a root conver

A directory

- `readonly`
  - On Wi

Example (PC

```
"root": {
    "path": "
    "readonly
}
```

A runtime MA document".

The features d document SHC

All properties i value MUST N

Specificat

- `ociVersic` MUST acc
- `ociVersic` MUST acc property. Specificati

Example

```
{
  "ociVersic
  "ociVersic
}
```

The 5 principles of Standard Containers

Define a unit of software delivery called a Standard Container. The goal of a Standard its dependencies in a format that is self-describing and portable, so that any compliant regardless of the underlying machine and the contents of the container.

The specification for Standard Containers defines:

1. configuration file formats
2. a set of standard operations
3. an execution environment.

A great analogy for this is the physical shipping container used by the transportation delivery, they can be lifted, stacked, locked, loaded, unloaded and labelled. Irrespecti allowed for a consistent, more streamlined and efficient set of processes to be define functionality by being the fundamental, standardized, unit of delivery for a software

5 Principals of Standard Containers

1. Standard operations

Standard Containers define a set of STANDARD OPERATIONS. They can be created, started, and stopped using standard container tools; copied and snapshotted using standard filesystem tools; and downloaded and uploaded using standard network tools.

2. Content-agnostic

Standard Containers are CONTENT-AGNOSTIC: all standard operations have the same effect regardless of the contents. They are started in the same way whether they contain a postgres database, a php application with its dependencies and application server, or Java build artifacts.

3. Infrastructure-agnostic

Standard Containers are INFRASTRUCTURE-AGNOSTIC: they can be run in any OCI supported infrastructure. For example, a standard container can be bundled on a laptop, uploaded to cloud storage, downloaded, run and snapshotted by a build server at a fiber hotel in Virginia, uploaded to 10 staging servers in a home-made private cloud cluster, then sent to 30 production instances across 3 public cloud regions.

5 Principals of Standard Containers (cont)

4. Designed for automation

Standard Containers are DESIGNED FOR AUTOMATION: because they offer the same standard operations regardless of content and infrastructure, Standard Containers, are extremely well-suited for automation. In fact, you could say automation is their secret weapon.

5. Industrial-grade delivery

Standard Containers make INDUSTRIAL-GRADE DELIVERY of software a reality. Leveraging all of the properties listed above, Standard Containers are enabling large and small enterprises to streamline and automate their software delivery pipelines. Whether it is in-house devOps flows, or external customer-based software delivery mechanisms, Standard Containers are changing the way the community thinks about software packaging and delivery.

Network isolation with virtual networks

```
docker run
   - p <hostport>:<containerport>
   <imagename>

docker run -p 8000:5000 web

OR

docker run -P web

docker container list
```

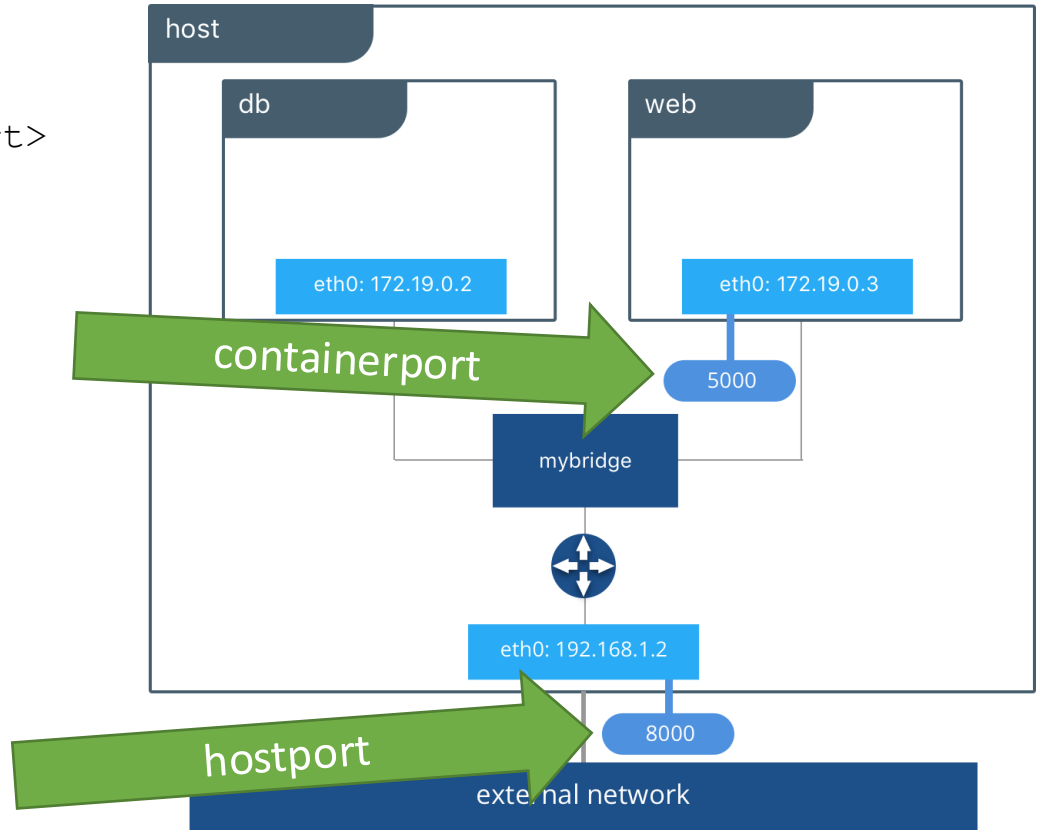
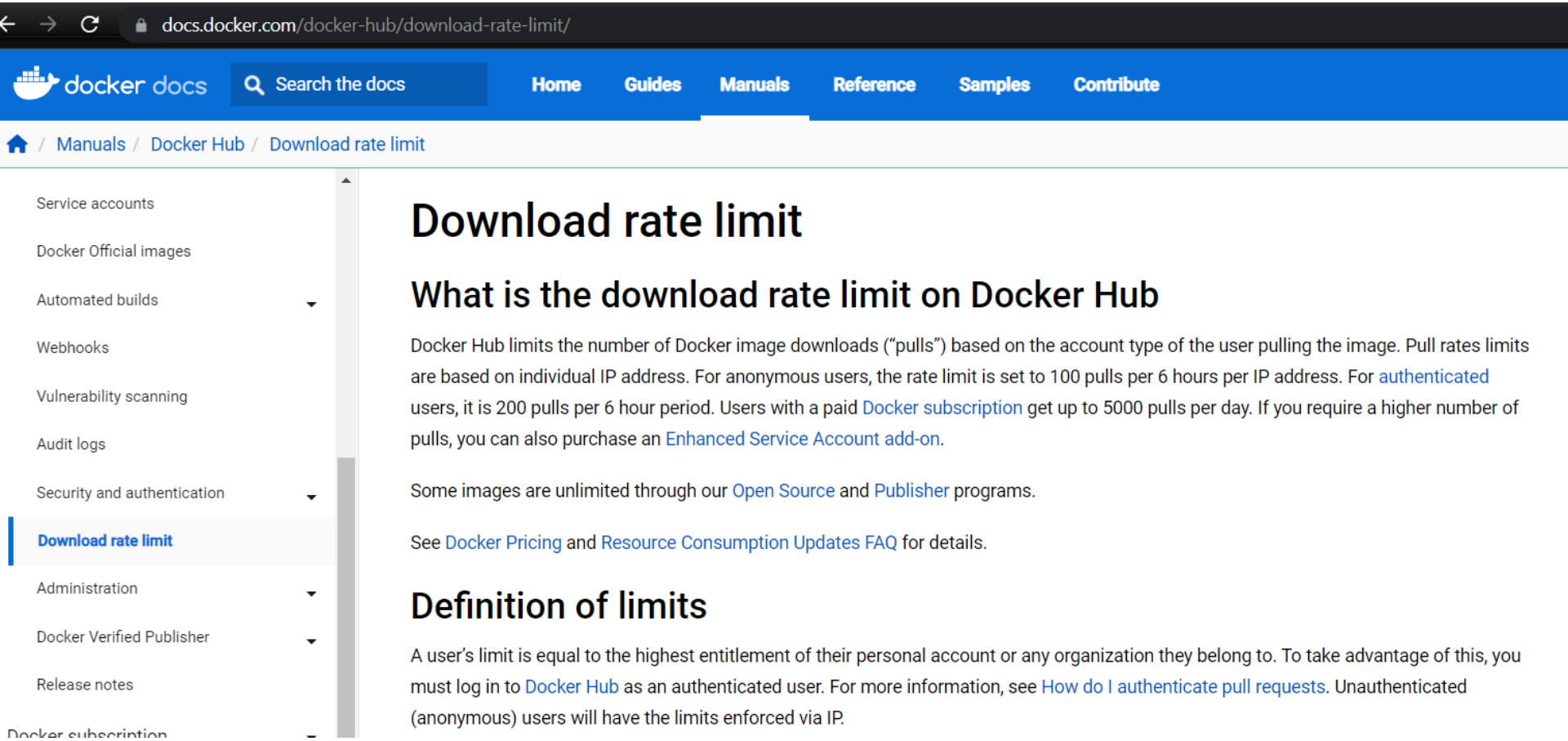
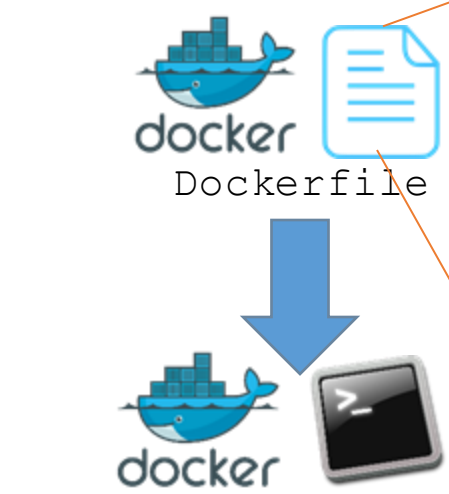


Image source: docker.com


docs.docker.com/docker-hub/download-rate-limit/
docker docs
Search the docs
Home
Guides
Manuals
Reference
Samples
Contribute
Manuals / Docker Hub / Download rate limit
Service accounts
Docker Official images
Automated builds
Webhooks
Vulnerability scanning
Audit logs
Security and authentication
Download rate limit
Administration
Docker Verified Publisher
Release notes
Download rate limit
What is the download rate limit on Docker Hub
Docker Hub limits the number of Docker image downloads (“pulls”) based on the account type of the user pulling the image. Pull rates limits are based on individual IP address. For anonymous users, the rate limit is set to 100 pulls per 6 hours per IP address. For authenticated users, it is 200 pulls per 6 hour period. Users with a paid Docker subscription get up to 5000 pulls per day. If you require a higher number of pulls, you can also purchase an Enhanced Service Account add-on.
Some images are unlimited through our Open Source and Publisher programs.
See Docker Pricing and Resource Consumption Updates FAQ for details.
Definition of limits
A user’s limit is equal to the highest entitlement of their personal account or any organization they belong to. To take advantage of this, you must log in to Docker Hub as an authenticated user. For more information, see How do I authenticate pull requests. Unauthenticated (anonymous) users will have the limits enforced via IP.


Docker Desktop
Upgrade plan
Search Ctrl+K
jaapapa
Containers
Give feedback
A container packages up code and its depend the application runs quickly and reliably from one computing environment to another.
INLOGGEN
EXT
Run a Sample Container
Try running a container: Copy and paste this command into your terminal and then come back
docker run -d -p 80:80 docker/getting-started
Explore more in the Docker Docs
RAM 1.83GB
CPU 0.00%
Connected to Hub
v4.14.1

Docker container intro




```
FROM nginx:1.9.14
ADD MyApp.html
/usr/share/nginx/html/inde
x.html
ENV DB_URL mydbserver:3306
ENV DB_USER admin
ENV DB_PW admin
VOLUME /var/log/nginx/
EXPOSE 80
CMD nginx -g daemon off;
```

Contents

Config Parameter

Data location

Networking

Start command

jaapp/myawesomeapp:1.0

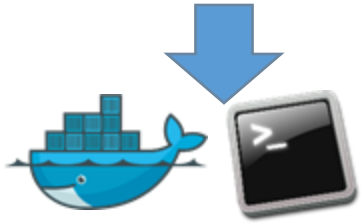
Image name

Docker container intro

jaapp/myawesomeapp:1.0

```
FROM nginx:1.9.14
ADD MyApp.html
/usr/share/nginx/html/index.html
ENV DB_URL mydbserver:3306
ENV DB_USER admin
ENV DB_PW admin
VOLUME /var/log/nginx/
EXPOSE 80
D nginx -g daemon off;
```

```
docker run
-e DB_URL:mydbserver:3066
-e DB_USER:admin
-e DB_PW:Welkom01
-v /temp:/var/logs/nginx
-p 80:80
jaapp/myawesomeapp:1.0
```

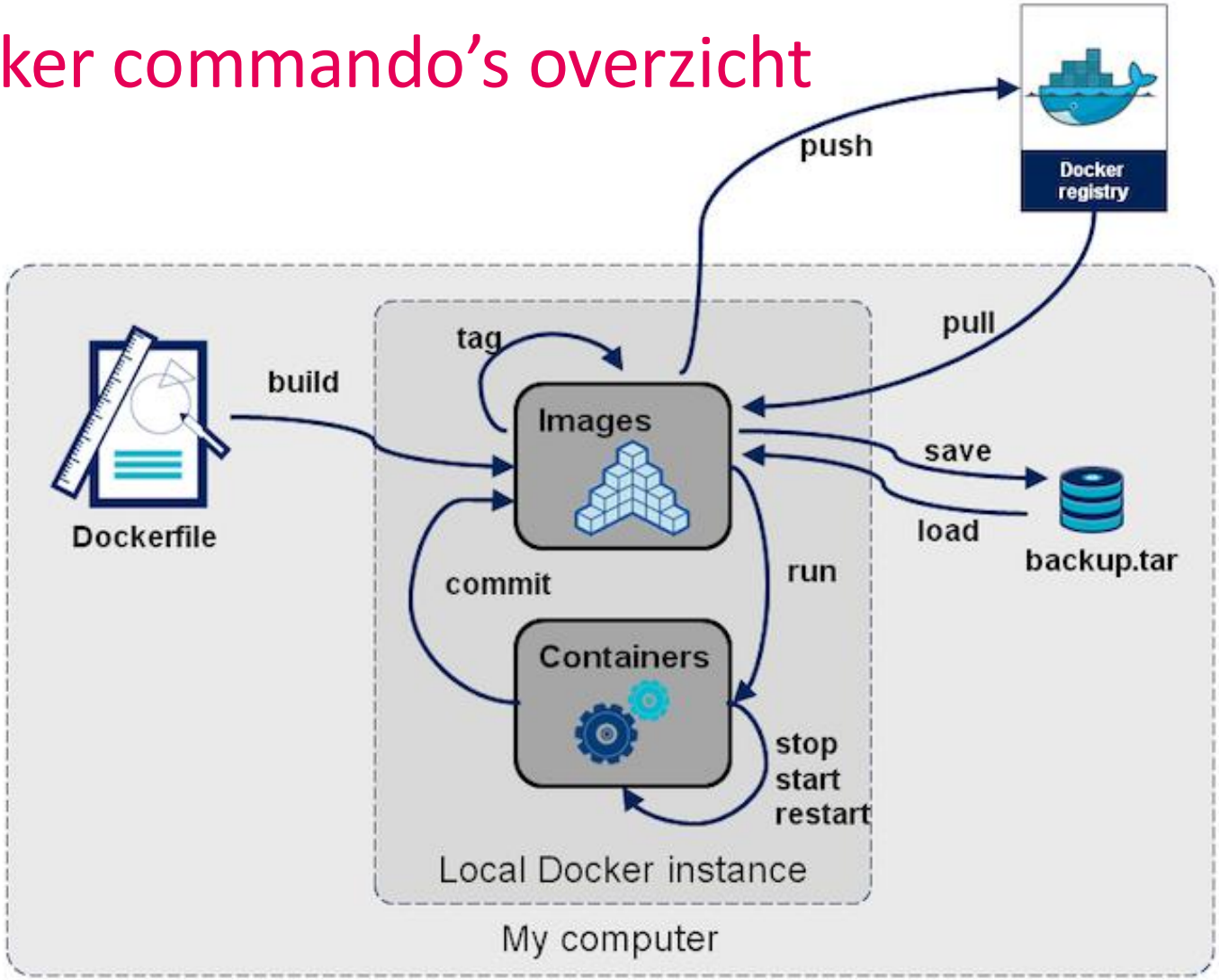


```
docker run
-e DB_URL:db.acc.local:3066
-e DB_USER:AppDBOwner01
-e DB_PW:Pa$$w0rd
-v /mnt/data:/var/logs/nginx
-p 8089:80
jaapp/myawesomeapp:1.0
```

```
docker run
-e DB_URL:db.corp.local:3066
-e DB_USER:AppDBOwner04
-e DB_PW:$&1A0Lf@--lm7t7
-v /mnt/datassd:/var/logs/nginx
-p 8065:80
jaapp/myawesomeapp:1.0
```

HAN_UNIVERSITY OF APPLIED SCIENCES

Docker commando's overzicht



Container images vs containers

Container

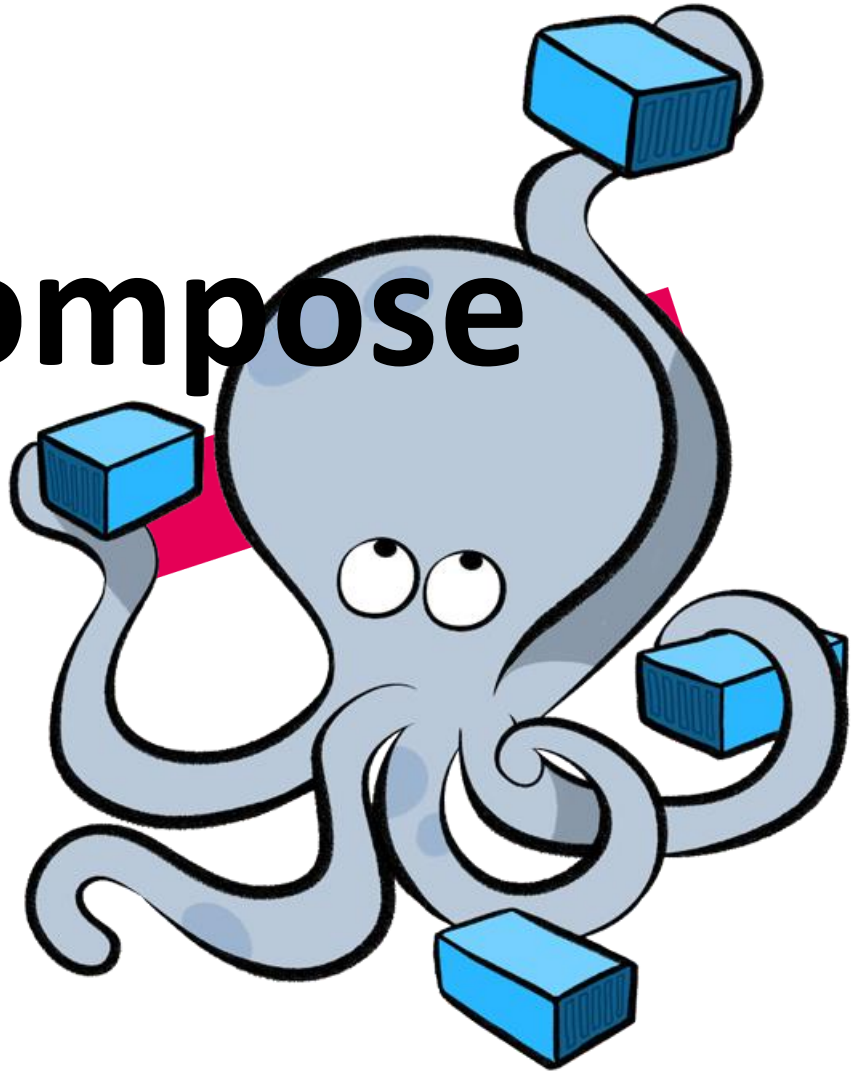
- Created by docker
  - ‘docker run <>’
- Running process
- Uses kernel features for process isolation
  - Cpu/memory
  - Process ID’s / namespace
  - Networking
  - File system
- Based on an image!
- Can write to mutable layer on top of image

(Container) image

- Blueprint for a container
- Contains:
  - Filesystem
  - Start command
  - Metadata
    - Environment Vars
    - Networking
    - Data storage
- Is immutable

AIM - ICT

Docker compose



Multi-container aps

HAN_UNIVERSITY
OF APPLIED SCIENCES

Multi container apps 1/3

DB Dockerfile:
FROM Ubuntu:14.04
RUN apt-get install mysql -y
EXPOSE 3306
VOLUME /var/lib/mysql
VOLUME /logs/mysql
CMD mysqld

App Dockerfile:
FROM Ubuntu:14.04
RUN apt-get install jdk –y
ADD myapp.jar /myapp.jar
EXPOSE 8080, 9090
CMD java /myapp.jar

Proxy Dockerfile:
FROM Ubuntu:14.04
RUN apt-get install nginx -y
EXPOSE 80, 443
VOLUME /opt/nginx/logs
CMD nginx -g -daemonoff

Multi container apps 2/3

```
run-all.sh:

docker run -d –n mydb -p 3066:3066
  -v /testdata/mysql:/var/mysql/lib
  –v /testdata/mysql-
logs:/logs/mysql
  jaapp/mydb:3.1

docker run -d –n myapp -p 8080:8080
  -e DB_URL:mydb:3066
  jaapp/myapp:3.1

docker run -d –n myproxy
  -p 80:80 –p 443:443
  -e WEBSERVER_URL:myapp:8080
  jaapp/myproxy:3.1
```

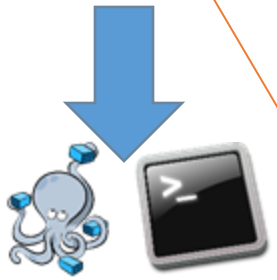
A. Wat als ik een enkele container wil bijwerken?

B. Wat gebeurt er als ik een enkele container stop waar een andere container van afhankelijk is?

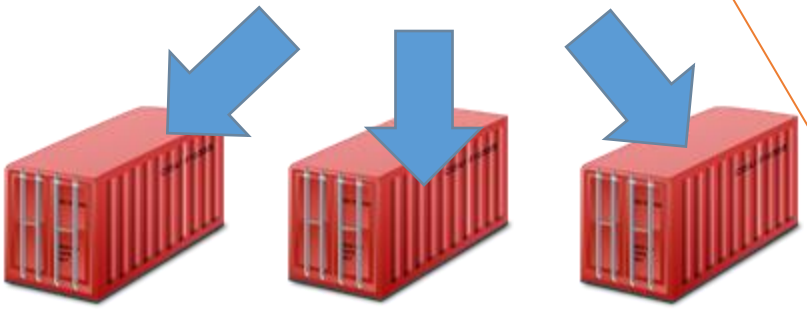
C. Hoe sla ik deze configuratie op?

Multi container apps 3/3

docker-compose.yml



docker-compose up



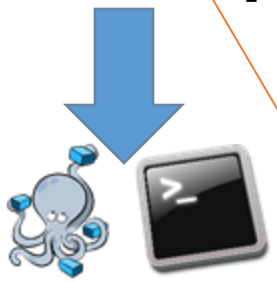
Mydb Myapp Myproxy

Docker-compose.yml:

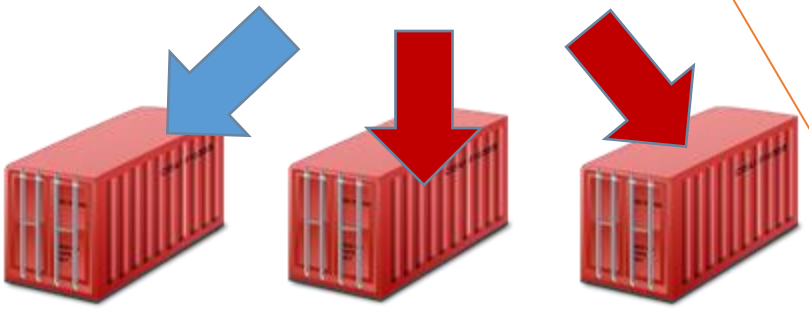
```
Mydb:
  image: jaapp/mydb:3.1
  ports:
    - 3066:3066
  volumes:
    - /testdata/mysql:/var/mysql/lib
    - /testdata/mysql-logs:/logs/mysql
Myapp:
  image: jaapp/myapp:3.1
  environment:
    - DB_URL:Mydb:3066
  ports:
    - 3066:3066
  links:
    - Mydb
Myproxy:
  image: jaapp/myprox:3.1
  environment:
    - WEBSERVER_URL:myapp:8080
  links:
    - Myapp
```

Multi container apps 3.5/3

docker-compose.yml



docker-compose up



Mydb Myapp Myproxy

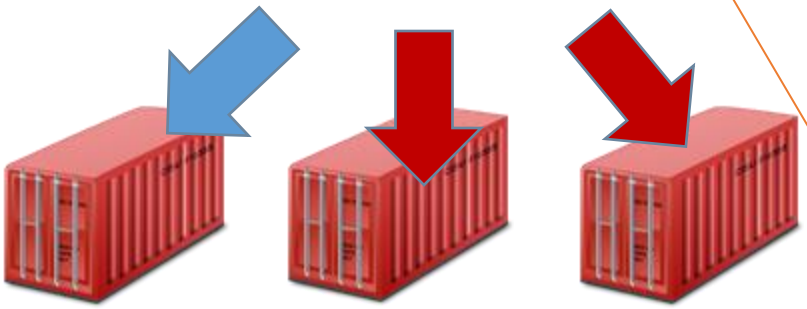
Docker-compose.yml:

```
Mydb:
  image: jaapp/mydb:3.1
  ports:
    - 3066:3066
  volumes:
    - /testdata/mysq    ar/mysql/lib
    - /testdata/mys     s:/logs/mysql
Myapp:
  image: jaapp/myapp:3.2
  environment:
    - DB_URL:Mydb:3066
  ports:
    - 3066:3066
  links:
    - Mydb
Myproxy:
  image: jaapp/myprox:3.1
  environment:
    - WEBSERVER_URL:myapp:8080
  links:
    - Myapp
```

Variables

docker-compose.yml

docker-compose up



Mydb Myapp Myproxy

Docker-compose.yml:

```
Mydb:
  image: jaapp/mydb:3.1
  ports:
    - 3066:3066
  volumes:
    - /testdata/mysq     ar/mysql/lib
    - /testdata/mys      s:/logs/mysql
Myapp:
  image: jaapp/myapp:3.2
  environment:
    - DB_URL:Mydb:3066
  ports:
    - 3066:3066
  links:
    - Mydb
Myproxy:
  image: jaapp/myprox:3.1
  environment:
    - WEBSERVER_URL:myapp:8080
  links:
    - Myapp
```

Image source: docker.com