Containers

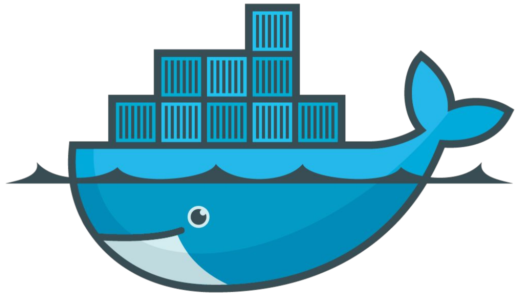
A good way to host web apps

Agenda

- **Containers**
  - What is it?
  - Docker Inc.
  - Why should we use it and where?

- **Work with Docker containers**
  - Basics
  - Volumes
  - Network
  - Logs

  → Labs 1: Run and troubleshoot your first containers



- **Work with Docker images**
  - What is an image?
  - How to create an image?
  - Registry and automated construction

  → Labs 2: Build your PHP application and share it

- **Multi-containers application**
  - Links
  - Docker-compose

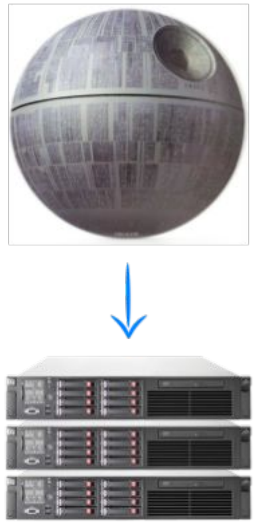
  → Labs 3: Start a Wordpress / Mysql stack

- **Docker in production**

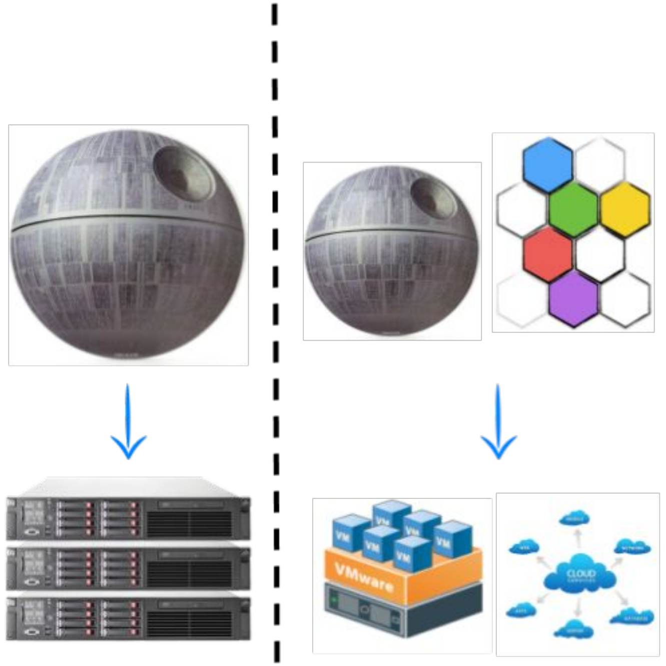
  → Labs 3: Troubleshoot a go web app
  - Orchestration
  - Best practices

About containers and why should we use it?

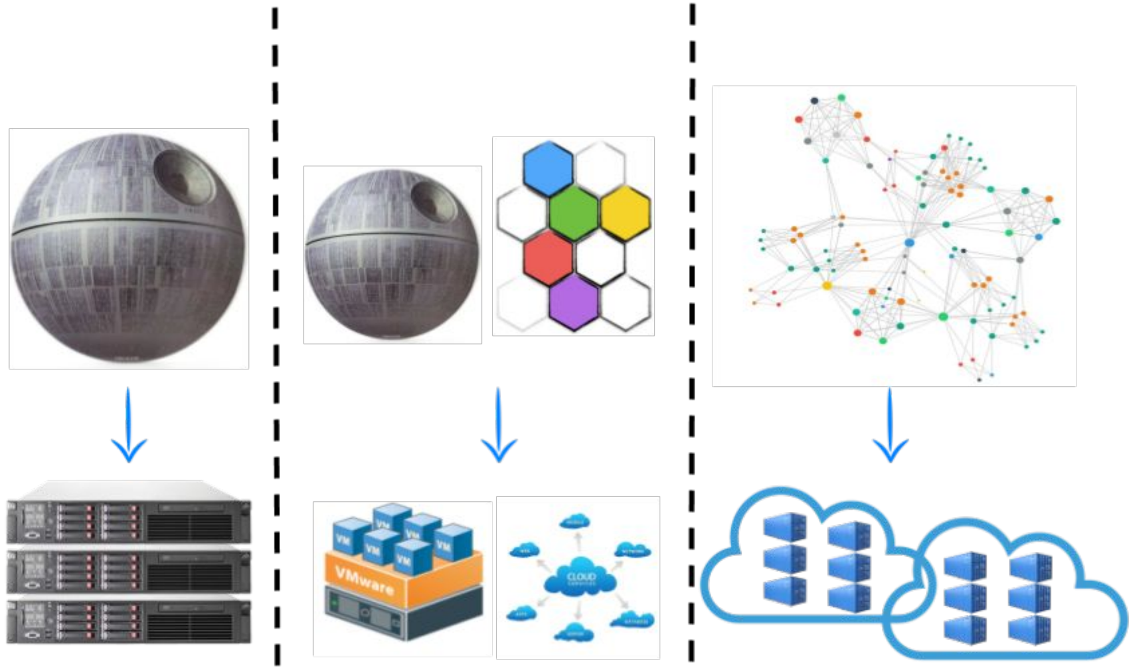
Containers?



Containers?



Containers?



What are containers?

What are containers?

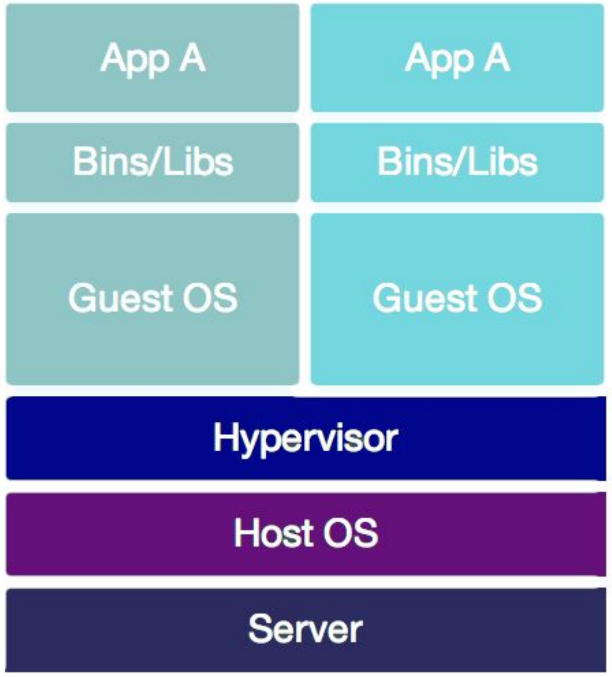
? ? ?

What are containers?

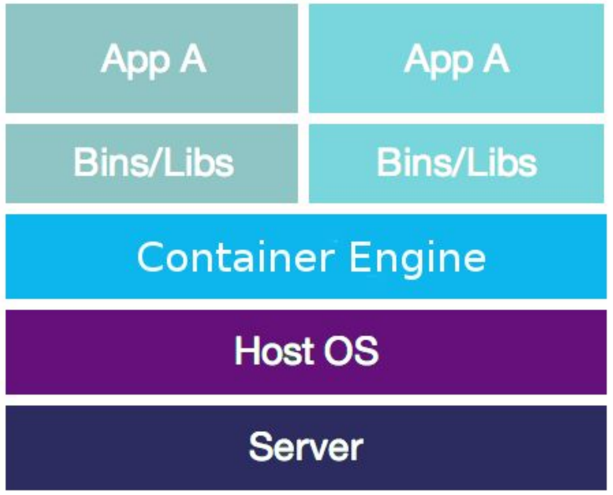
A container consists of everything an application needs to run: the application itself and its dependencies (e.g. libraries, utilities, configuration files), all bundled into one package.

Containers encapsulate only the minimal resources that an application requires to run and function as intended, enabling you to reliably run software when moved from one computing environment to another, which is the main idea behind the use of containers.

Containers vs VMs



Virtual Machines



Containers

History

? ? ?

History

There are many container implementations, with varying degrees of capabilities and isolation:

Operating System Container

- **1982. Chroot**
- **2000. Virtuozzo/Parallels**
- **2005. OpenVZ**
- **2008. LXC**

cgroups and namespaces based Containers

- **2014. Docker**
- **2015. rkt**

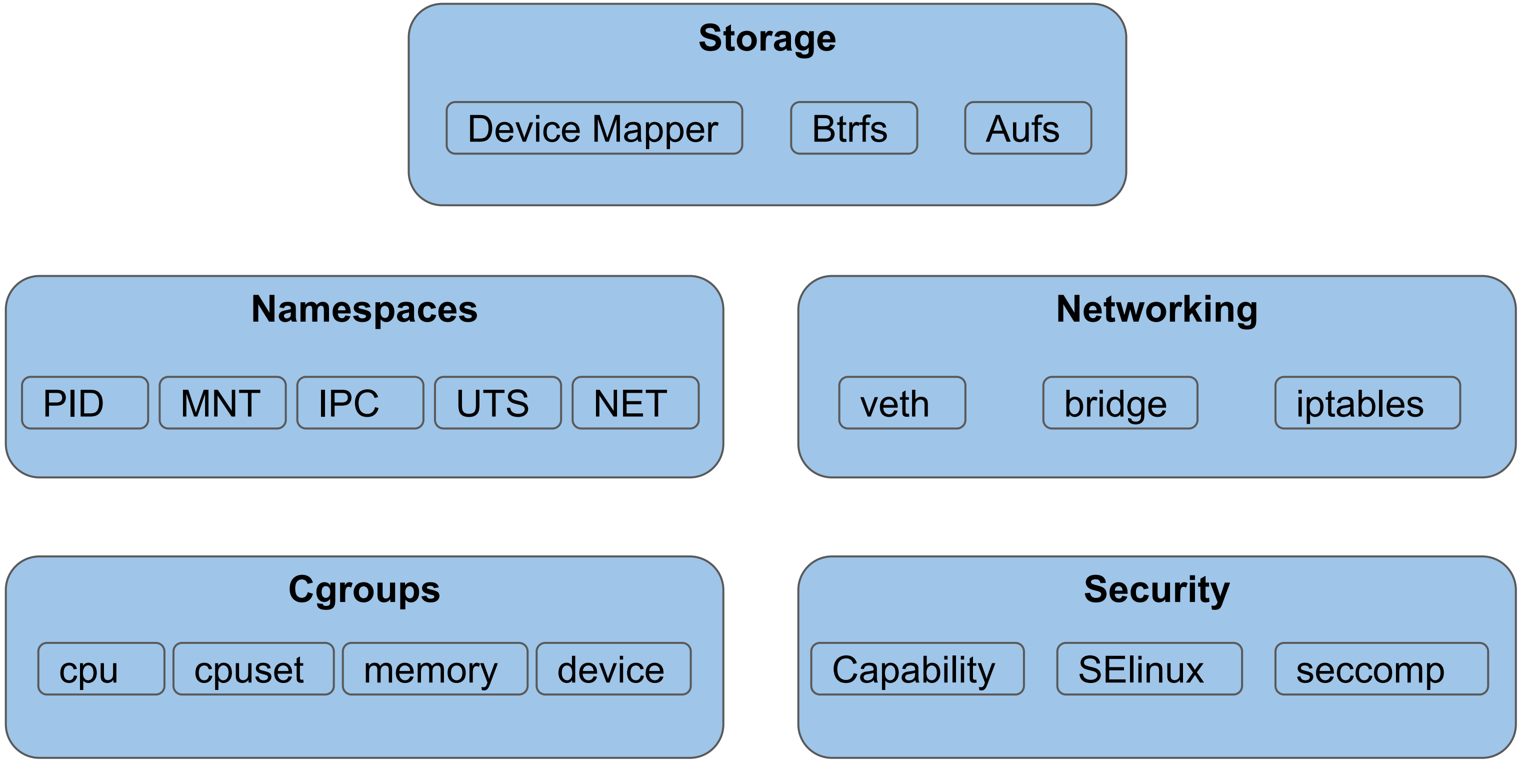
https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016

Isolation: Cgroups and Namespaces ?

Latest generation of Linux containers are based on Linux cgroups and Linux namespaces

Mount namespace	**Linux 2.4.19**	**: File system isolation**
PID namespace	**Linux 2.6.24**	**: ID process isolation**
Net namespace	**Linux 2.6.19-2.6.24**	**: Network isolation**
User namespace	**Linux 2.6.23-3.8**	**: Right management isolation**
IPC namespace	**Linux 2.6.19-2.6.30**	**: Interprocess isolation**
UTS namespace	**Linux 2.6.19**	**: Time isolation**
Control Groups	**Linux 2.6.24**	**: Resources isolation**

Anatomy


Storage
Device Mapper
Btrfs
Aufs
Namespaces
PID
MNT
IPC
UTS
NET
Networking
veth
bridge
iptables
Cgroups
cpu
cpuset
memory
device
Security
Capability
SElinux
seccomp

Docker Inc.

Docker Inc.

- Docker Enterprise Edition

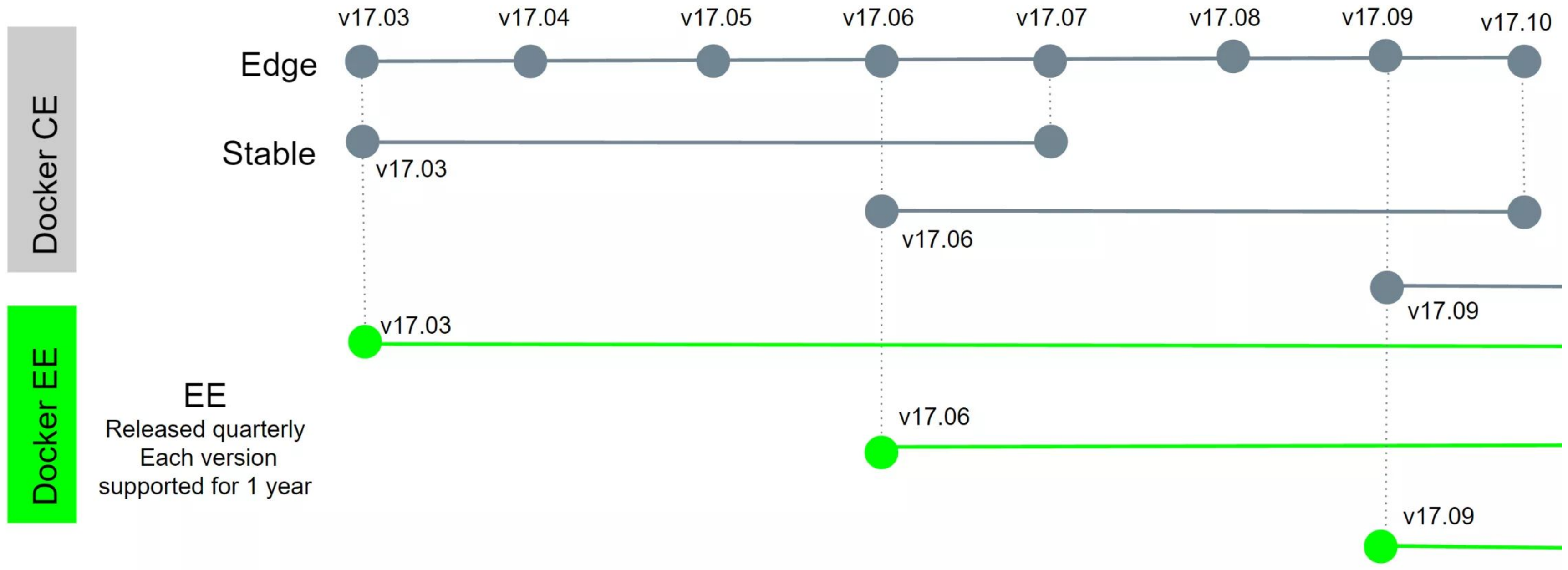
- Docker Community Edition
  - Edge is for users wanting a drop of the latest and greatest features every month
  - Stable is released quarterly and is for users that want an easier-to-maintain release pace

- Docker official public registry: Dockerhub

- Docker swarm

https://blog.docker.com/2017/03/docker-enterprise-edition/

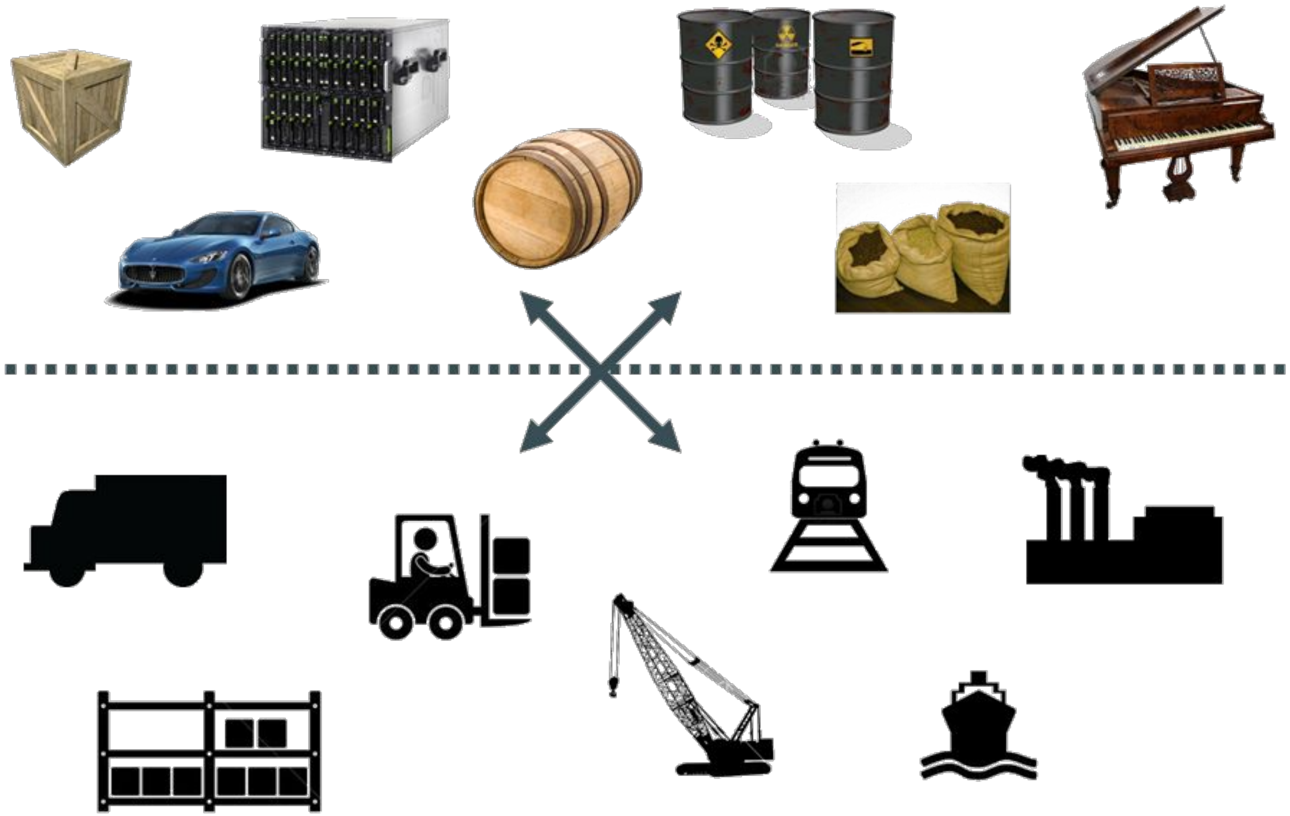
Docker Inc.


Docker CE
Docker EE
Edge
Stable
v17.03
v17.04
v17.05
v17.06
v17.07
v17.08
v17.09
v17.10
v17.03
v17.06
v17.09
v17.03
v17.06
v17.09
EE
Released quarterly
Each version
supported for 1 year

Docker philosophy

Multiplicity of Goods

Do I worry about how goods interact (e.g. coffee beans next to spices)



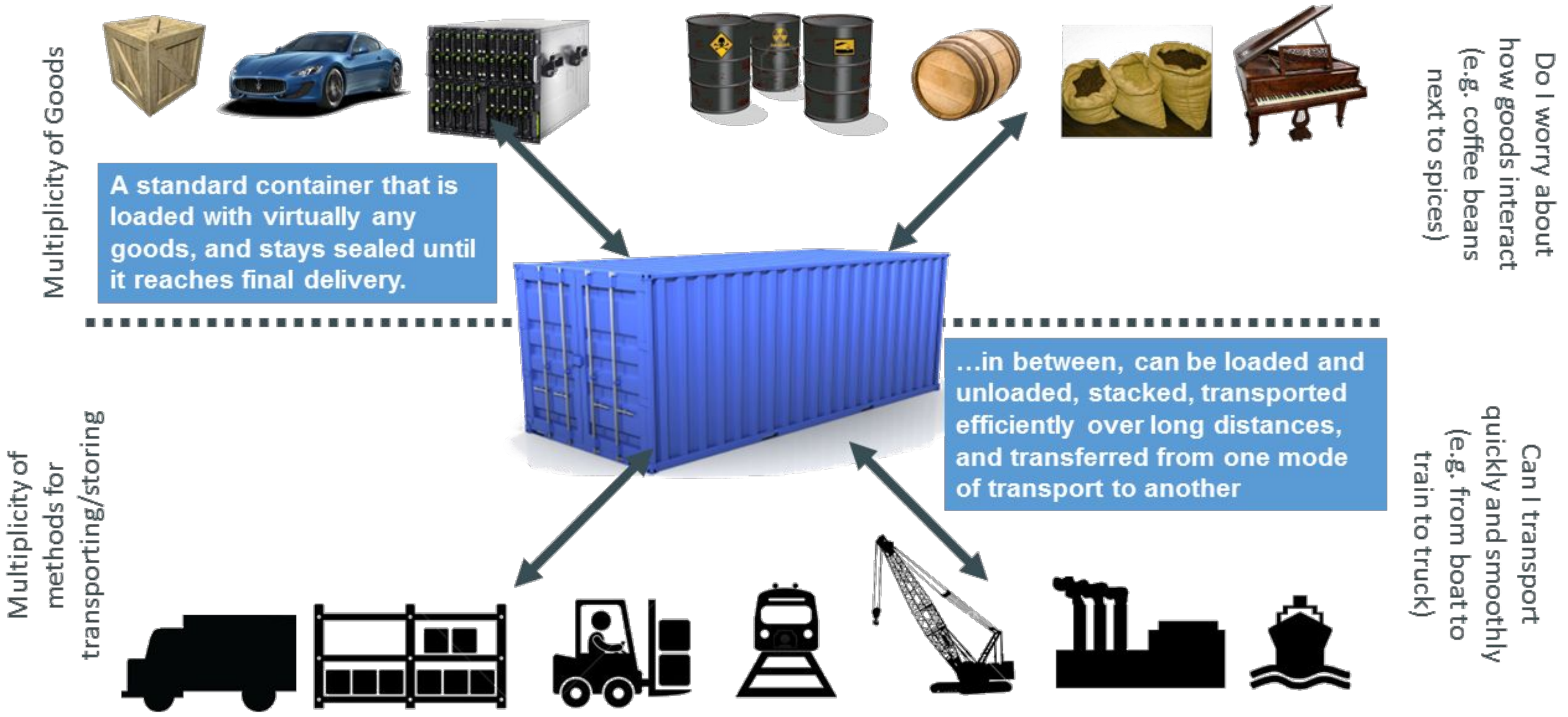
Multipilicity of methods for transporting/storing

Can I transport quickly and smoothly (e.g. from boat to train to truck)

Docker philosophy

	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	🚆	🗄️	🚜	🏗️	🚢	🏭	🚚

Docker philosophy



Multiplicity of Goods

A standard container that is loaded with virtually any goods, and stays sealed until it reaches final delivery.

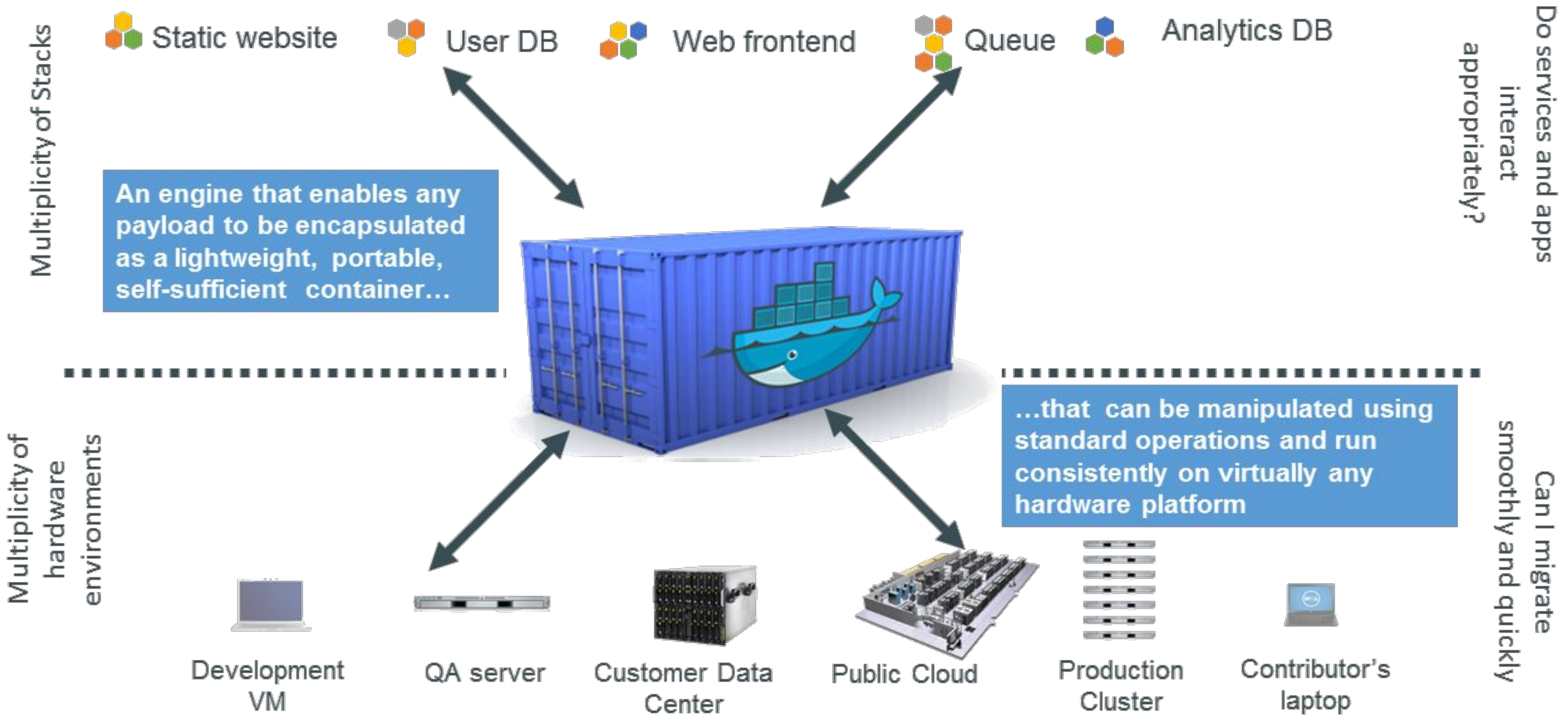
Do I worry about how goods interact (e.g. coffee beans next to spices)

Multiplicity of methods for transporting/storing

…in between, can be loaded and unloaded, stacked, transported efficiently over long distances, and transferred from one mode of transport to another

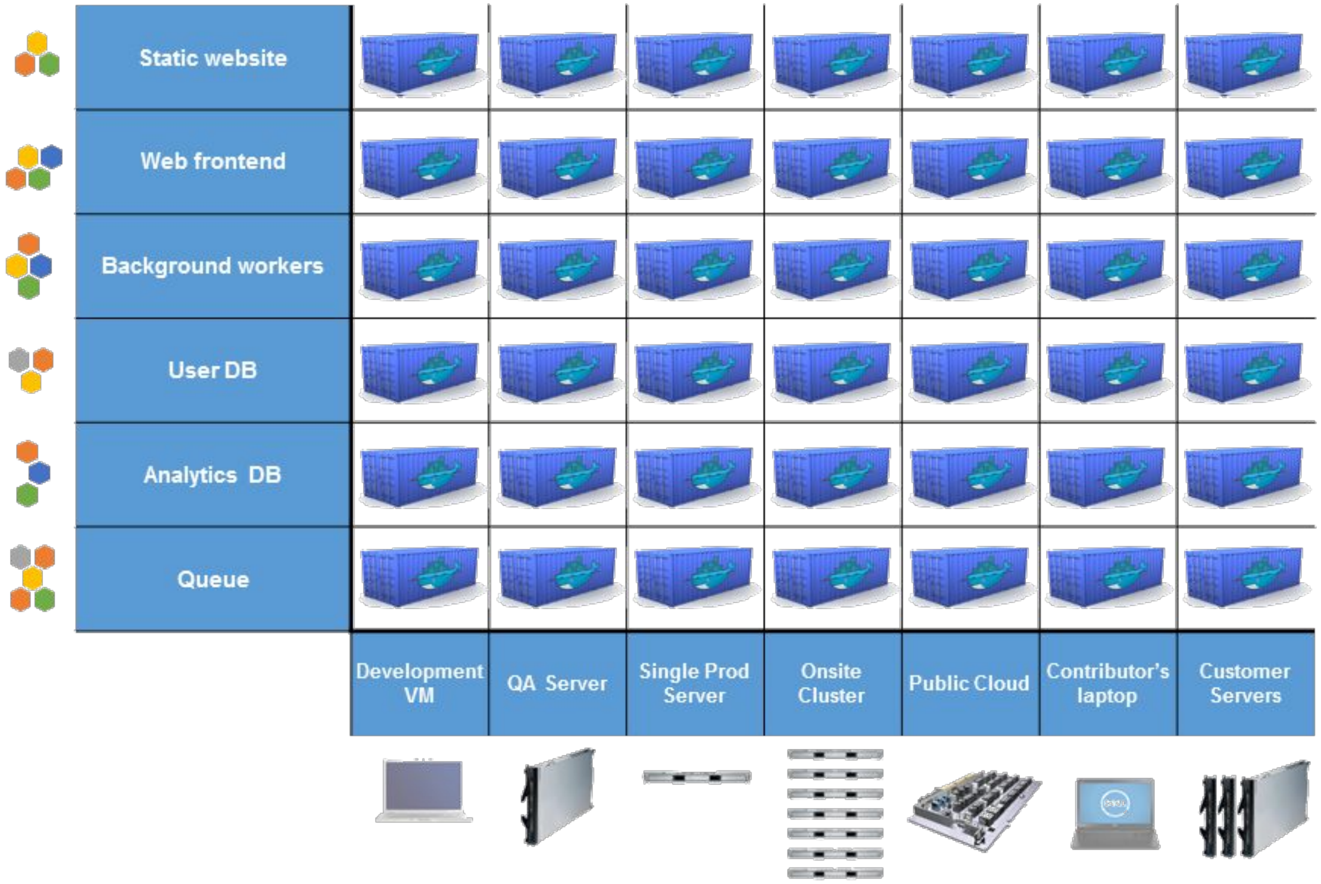
Can I transport quickly and smoothly (e.g. from boat to train to truck)

Docker philosophy

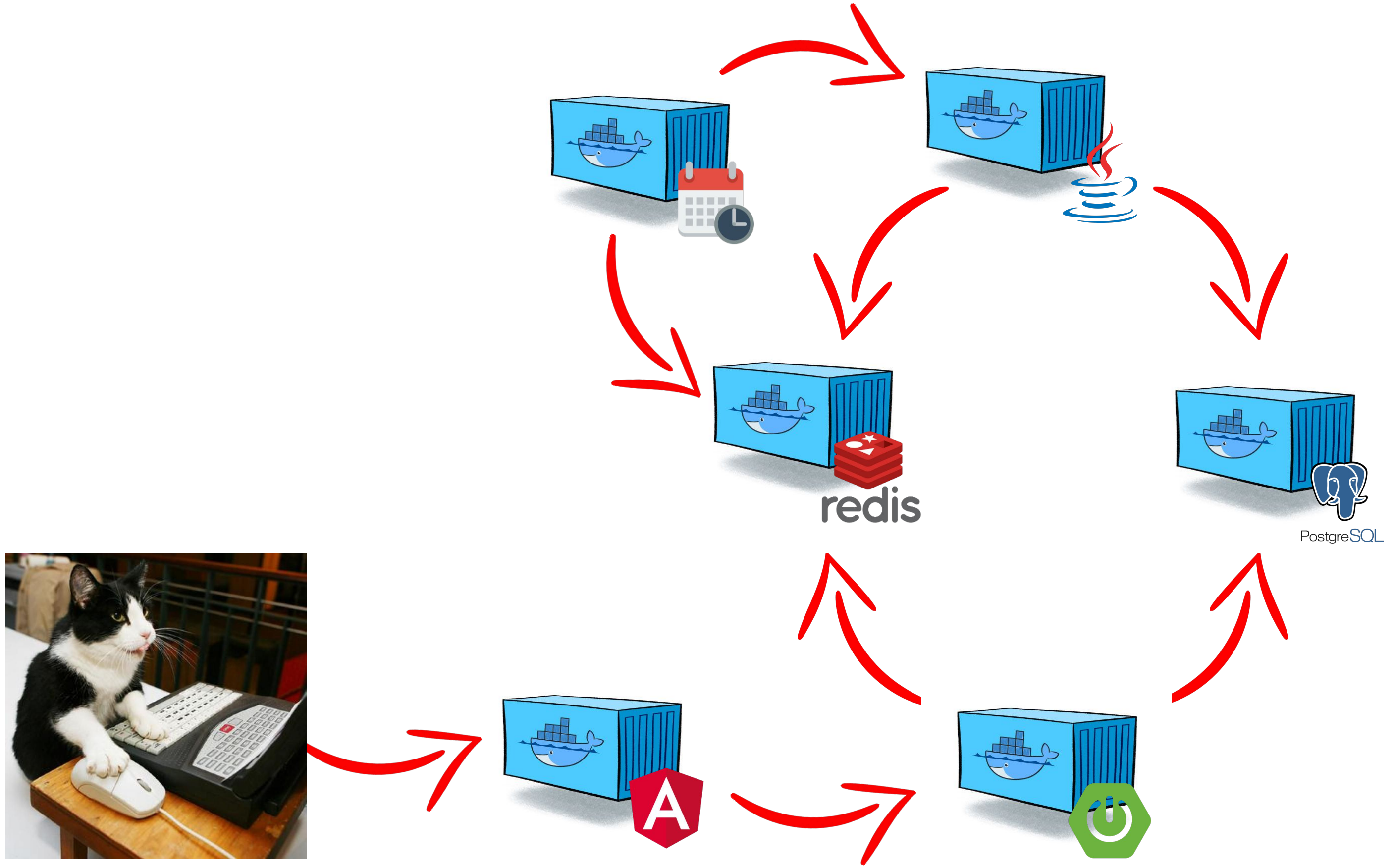

An engine that enables any payload to be encapsulated as a lightweight, portable, self-sufficient container…

…that can be manipulated using standard operations and run consistently on virtually any hardware platform

Docker philosophy



Micro services example



Why is Docker so important?

? ? ?

Why Developers Care

- Build once... (finally) run anywhere*
- A clean, safe, hygienic, portable runtime environment for your app.
- No worries about missing dependencies, packages and other pain points during subsequent deployments.
- Run each app in its own isolated container, so you can run various versions of libraries and other dependencies for each app without worrying.
- Automate testing, integration, packaging...anything you can script.
- Reduce/eliminate concerns about compatibility on different platforms, either your own or your customers.
- Cheap, zero-penalty containers to deploy services. A VM without the overhead of a VM. Instant replay and reset of image snapshots.

Why Administrators Care

- Configure once... run anything
- Make the entire lifecycle more efficient, consistent, and repeatable
- Eliminate inconsistencies between development, test, production, and customer environments.
- Support segregation of duties.
- Significantly improves the speed and reliability of continuous deployment and continuous integration systems.
- Because the containers are so lightweight, address significant performance, costs, deployment, and portability issues normally associated with VMs.

Why is it so important for all?

- Optimize hardware usage.
- Prospective hires like containers.
- Containers are open source.
- The learning curve is manageable.
- You can deploy faster.
- Containers give you deployment flexibility
- They work with the infrastructure you already use.

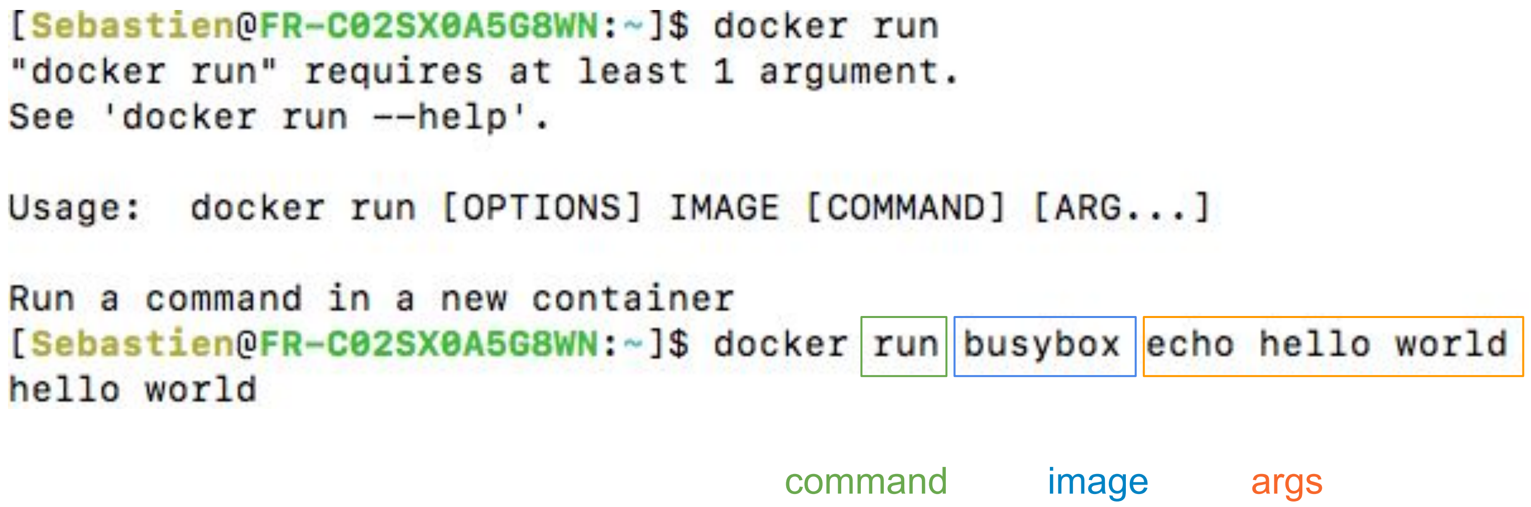
Where we should use it?

? ? ?

Where we should use it?

Work with containers

Hello World

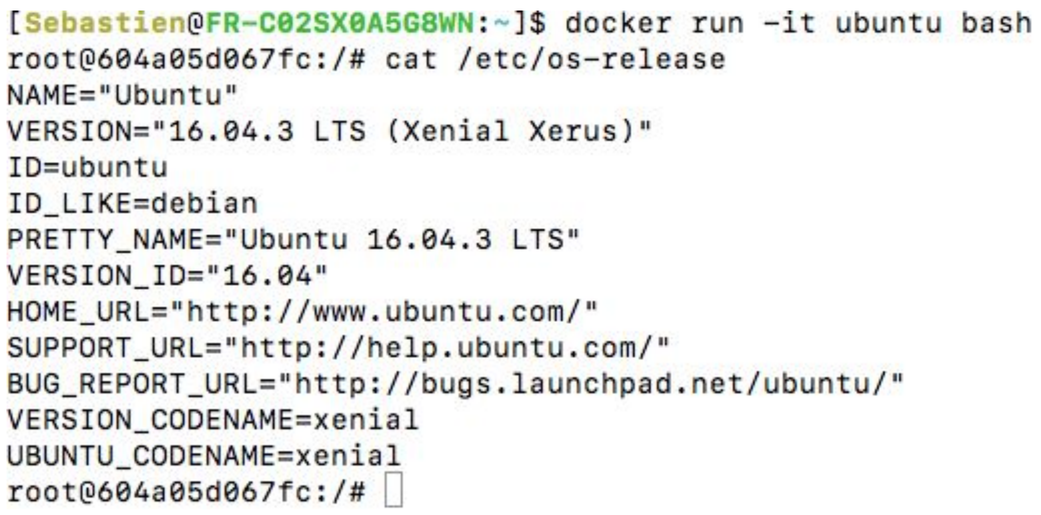


```
[Sebastien@FR-C02SX0A5G8WN:~]$ docker run
"docker run" requires at least 1 argument.
See 'docker run --help'.

Usage:  docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

Run a command in a new container
[Sebastien@FR-C02SX0A5G8WN:~]$ docker run busybox echo hello world
hello world
```

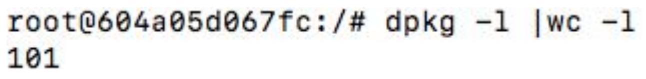
Interactive container



```
[Sebastien@FR-C02SX0A5G8WN:~]$ docker run -it ubuntu bash
root@604a05d067fc:/# cat /etc/os-release
NAME="Ubuntu"
VERSION="16.04.3 LTS (Xenial Xerus)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 16.04.3 LTS"
VERSION_ID="16.04"
HOME_URL="http://www.ubuntu.com/"
SUPPORT_URL="http://help.ubuntu.com/"
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
VERSION_CODENAME=xenial
UBUNTU_CODENAME=xenial
root@604a05d067fc:/#
```

Run a simple ubuntu container

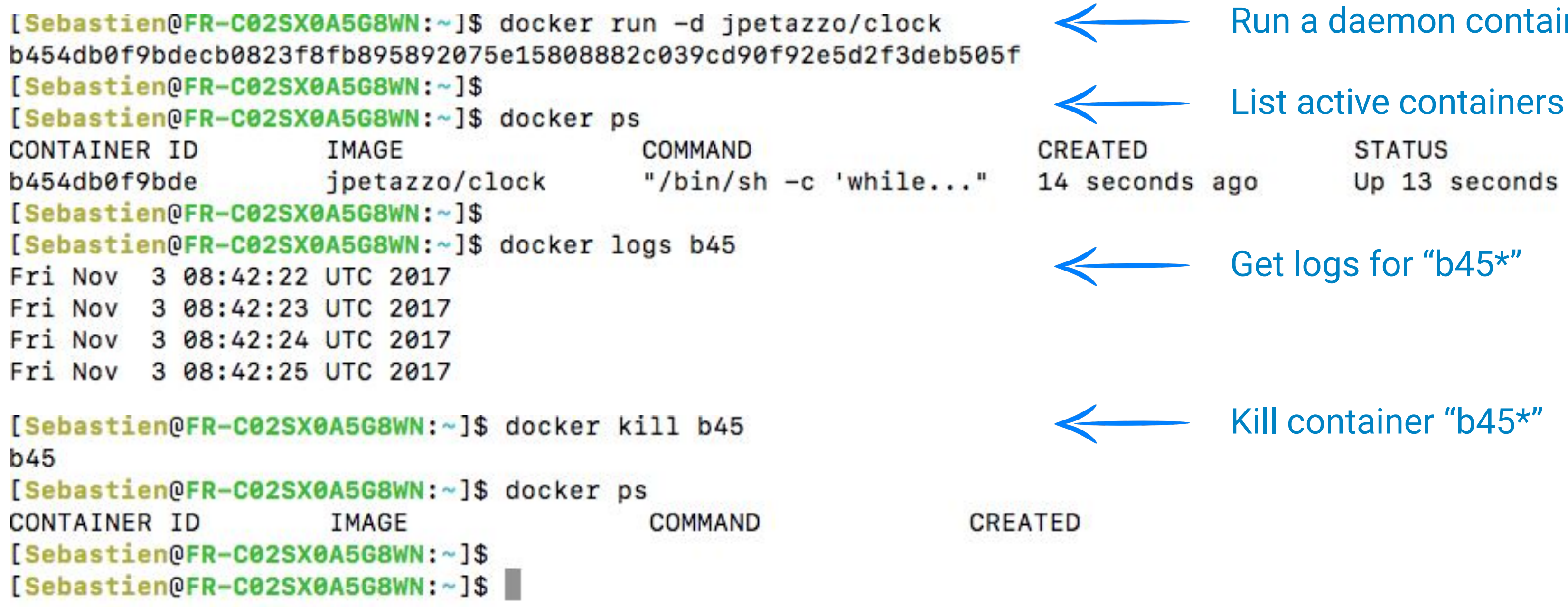
Get OS information



```
root@604a05d067fc:/# dpkg -l |wc -l
101
```

Count installed packages

Background container



```
[Sebastien@FR-C02SX0A5G8WN:~]$ docker run -d jpetazzo/clock
b454db0f9bdecb0823f8fb895892075e15808882c039cd90f92e5d2f3deb505f
[Sebastien@FR-C02SX0A5G8WN:~]$
[Sebastien@FR-C02SX0A5G8WN:~]$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
b454db0f9bde        jpetazzo/clock      "/bin/sh -c 'while..."   14 seconds ago      Up 13 seconds
[Sebastien@FR-C02SX0A5G8WN:~]$
[Sebastien@FR-C02SX0A5G8WN:~]$ docker logs b45
Fri Nov  3 08:42:22 UTC 2017
Fri Nov  3 08:42:23 UTC 2017
Fri Nov  3 08:42:24 UTC 2017
Fri Nov  3 08:42:25 UTC 2017

[Sebastien@FR-C02SX0A5G8WN:~]$ docker kill b45
b45
[Sebastien@FR-C02SX0A5G8WN:~]$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
[Sebastien@FR-C02SX0A5G8WN:~]$
[Sebastien@FR-C02SX0A5G8WN:~]$ █
```

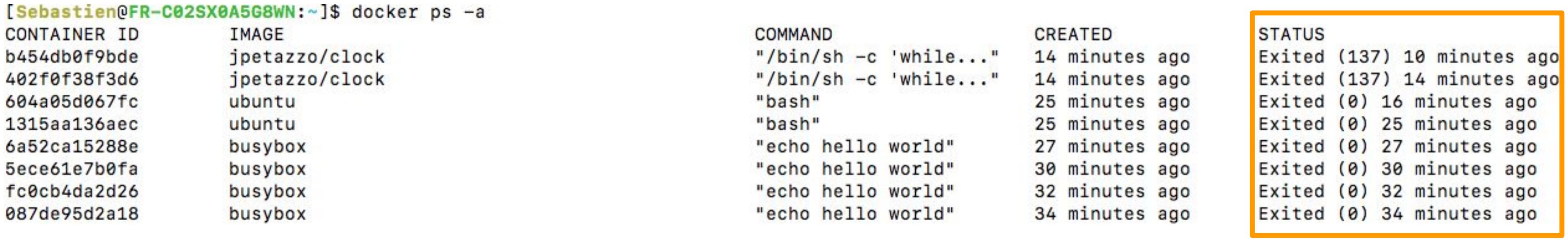
← Run a daemon container

← List active containers

← Get logs for “b45*”

← Kill container “b45*”

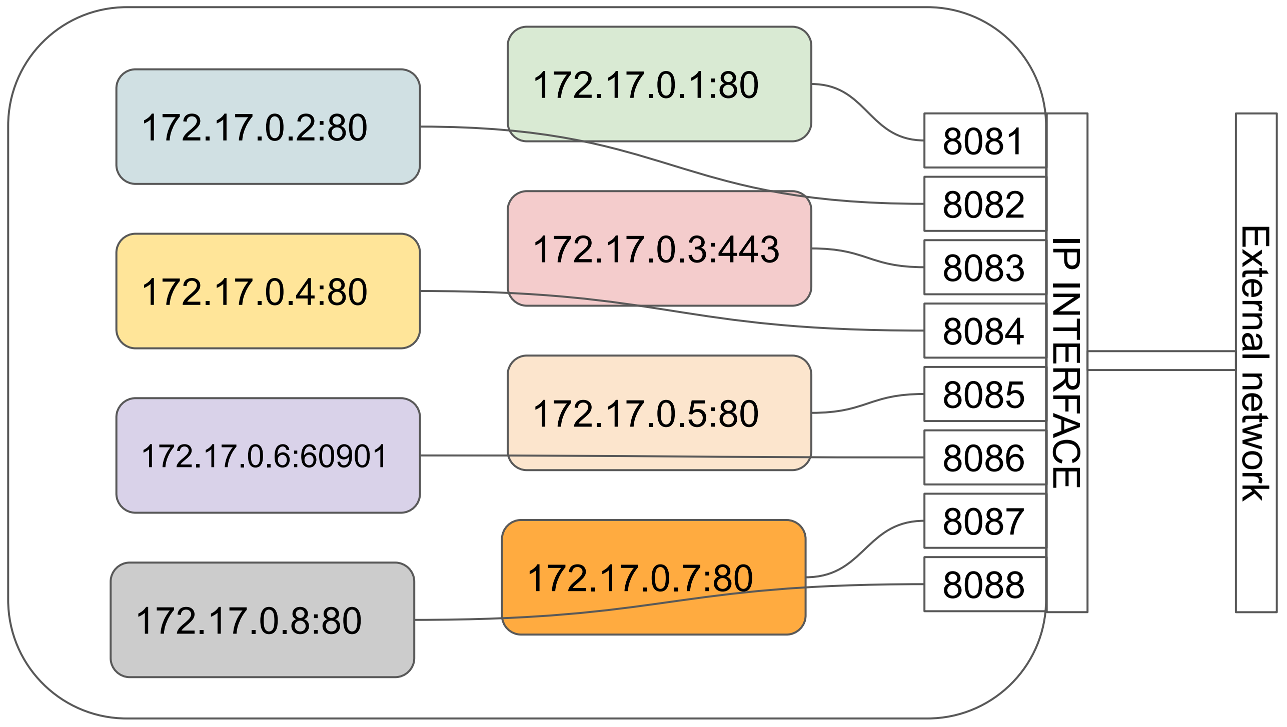
Dead/stopped containers



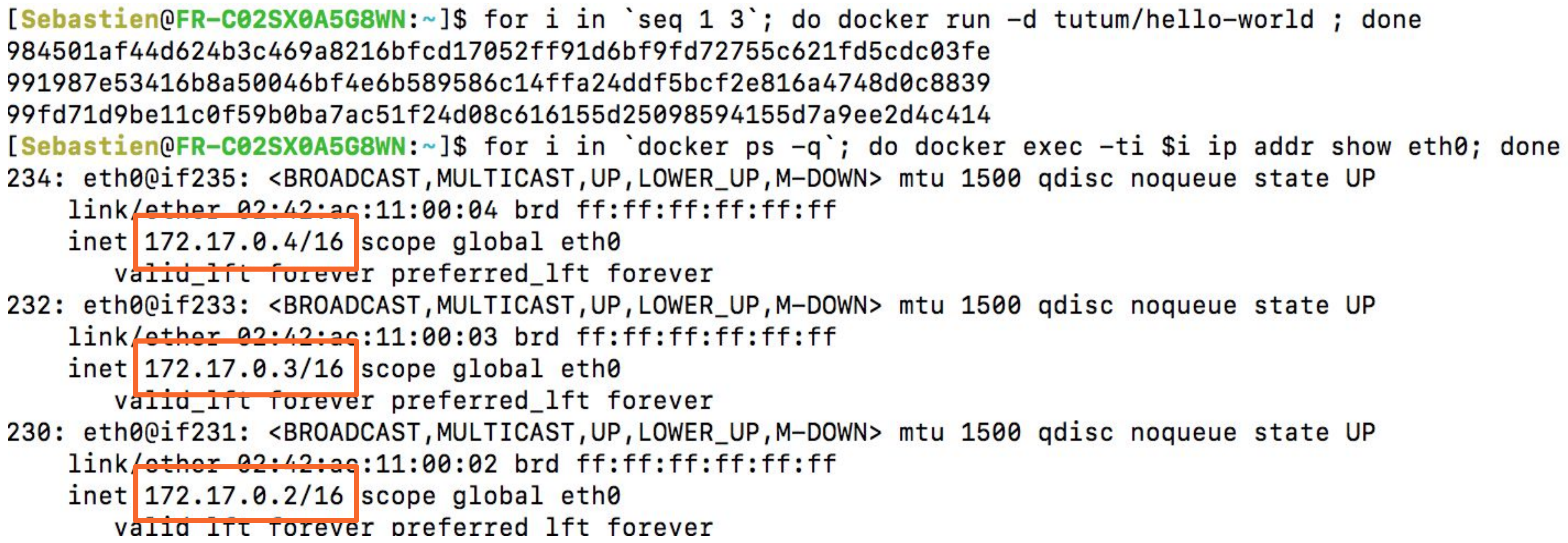
```
[Sebastien@FR-C02SX0A5G8WN:~]$ docker ps -a
CONTAINER ID        IMAGE                                          COMMAND                  CREATED             STATUS
b454db0f9bde        jpetazzo/clock                                 "/bin/sh -c 'while..."   14 minutes ago      Exited (137) 10 minutes ago
402f0f38f3d6        jpetazzo/clock                                 "/bin/sh -c 'while..."   14 minutes ago      Exited (137) 14 minutes ago
604a05d067fc        ubuntu                                         "bash"                   25 minutes ago      Exited (0) 16 minutes ago
1315aa136aec        ubuntu                                         "bash"                   25 minutes ago      Exited (0) 25 minutes ago
6a52ca15288e        busybox                                        "echo hello world"       27 minutes ago      Exited (0) 27 minutes ago
5ece61e7b0fa        busybox                                        "echo hello world"       30 minutes ago      Exited (0) 30 minutes ago
fc0cb4da2d26        busybox                                        "echo hello world"       32 minutes ago      Exited (0) 32 minutes ago
087de95d2a18        busybox                                        "echo hello world"       34 minutes ago      Exited (0) 34 minutes ago
```

Network isolation & ports mapping

Network isolation & ports mapping

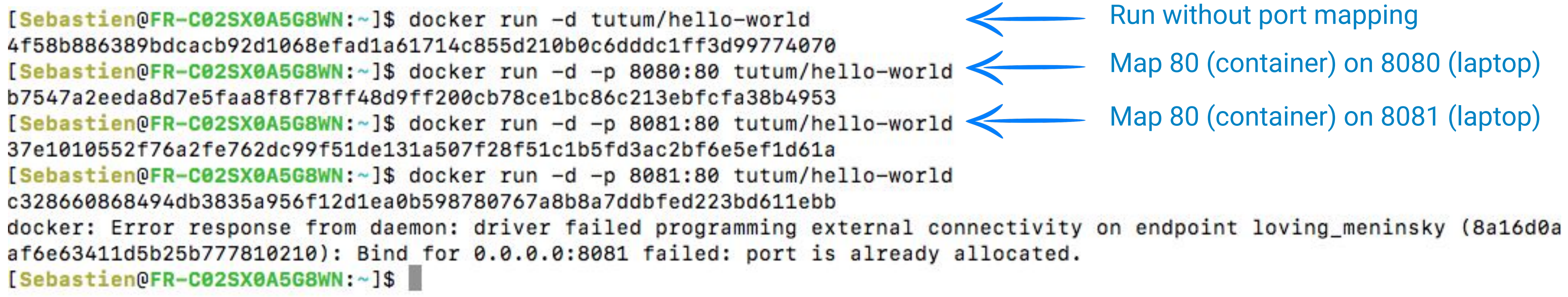


Network isolation & ports mapping



```
[Sebastien@FR-C02SX0A5G8WN:~]$ for i in `seq 1 3`; do docker run -d tutum/hello-world ; done
984501af44d624b3c469a8216bfcd17052ff91d6bf9fd72755c621fd5cdc03fe
991987e53416b8a50046bf4e6b589586c14ffa24ddf5bcf2e816a4748d0c8839
99fd71d9be11c0f59b0ba7ac51f24d08c616155d25098594155d7a9ee2d4c414
[Sebastien@FR-C02SX0A5G8WN:~]$ for i in `docker ps -q`; do docker exec -ti $i ip addr show eth0; done
234: eth0@if235: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:ac:11:00:04 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.4/16 scope global eth0
       valid_lft forever preferred_lft forever
232: eth0@if233: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:ac:11:00:03 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.3/16 scope global eth0
       valid_lft forever preferred_lft forever
230: eth0@if231: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.2/16 scope global eth0
       valid_lft forever preferred_lft forever
```

Network isolation & ports mapping



```
[Sebastien@FR-C02SX0A5G8WN:~]$ docker run -d tutum/hello-world
4f58b886389bdcacb92d1068efad1a61714c855d210b0c6dddc1ff3d99774070
[Sebastien@FR-C02SX0A5G8WN:~]$ docker run -d -p 8080:80 tutum/hello-world
b7547a2eeda8d7e5faa8f8f78ff48d9ff200cb78ce1bc86c213ebfcfa38b4953
[Sebastien@FR-C02SX0A5G8WN:~]$ docker run -d -p 8081:80 tutum/hello-world
37e1010552f76a2fe762dc99f51de131a507f28f51c1b5fd3ac2bf6e5ef1d61a
[Sebastien@FR-C02SX0A5G8WN:~]$ docker run -d -p 8081:80 tutum/hello-world
c328660868494db3835a956f12d1ea0b598780767a8b8a7ddbfed223bd611ebb
docker: Error response from daemon: driver failed programming external connectivity on endpoint loving_meninsky (8a16d0a
af6e63411d5b25b777810210): Bind for 0.0.0.0:8081 failed: port is already allocated.
[Sebastien@FR-C02SX0A5G8WN:~]$
```

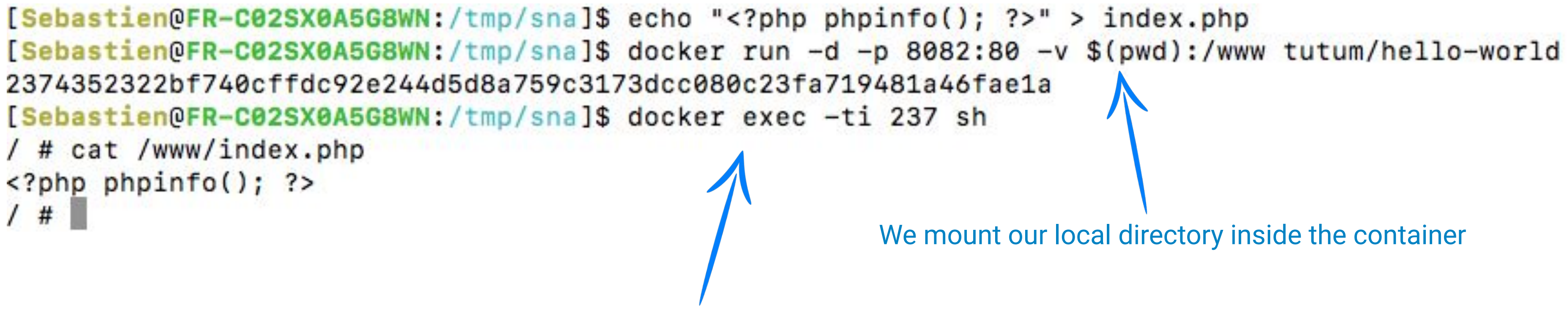
Run without port mapping

Map 80 (container) on 8080 (laptop)

Map 80 (container) on 8081 (laptop)

Volume management

Volume management

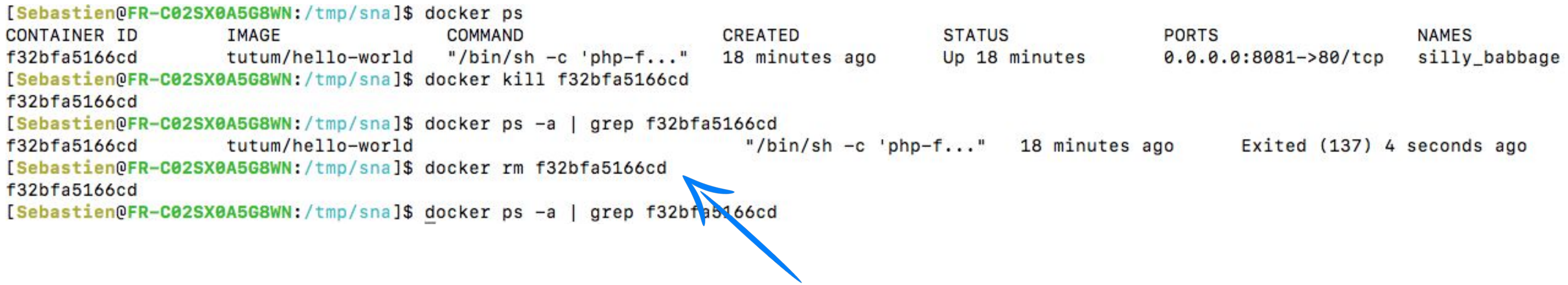


```
[Sebastien@FR-C02SX0A5G8WN:/tmp/sna]$ echo "<?php phpinfo(); ?>" > index.php
[Sebastien@FR-C02SX0A5G8WN:/tmp/sna]$ docker run -d -p 8082:80 -v $(pwd):/www tutum/hello-world
2374352322bf740cffdc92e244d5d8a759c3173dcc080c23fa719481a46fae1a
[Sebastien@FR-C02SX0A5G8WN:/tmp/sna]$ docker exec -ti 237 sh
/ # cat /www/index.php
<?php phpinfo(); ?>
/ #
```

We mount our local directory inside the container

Troubleshooting: We want to enter in a background container

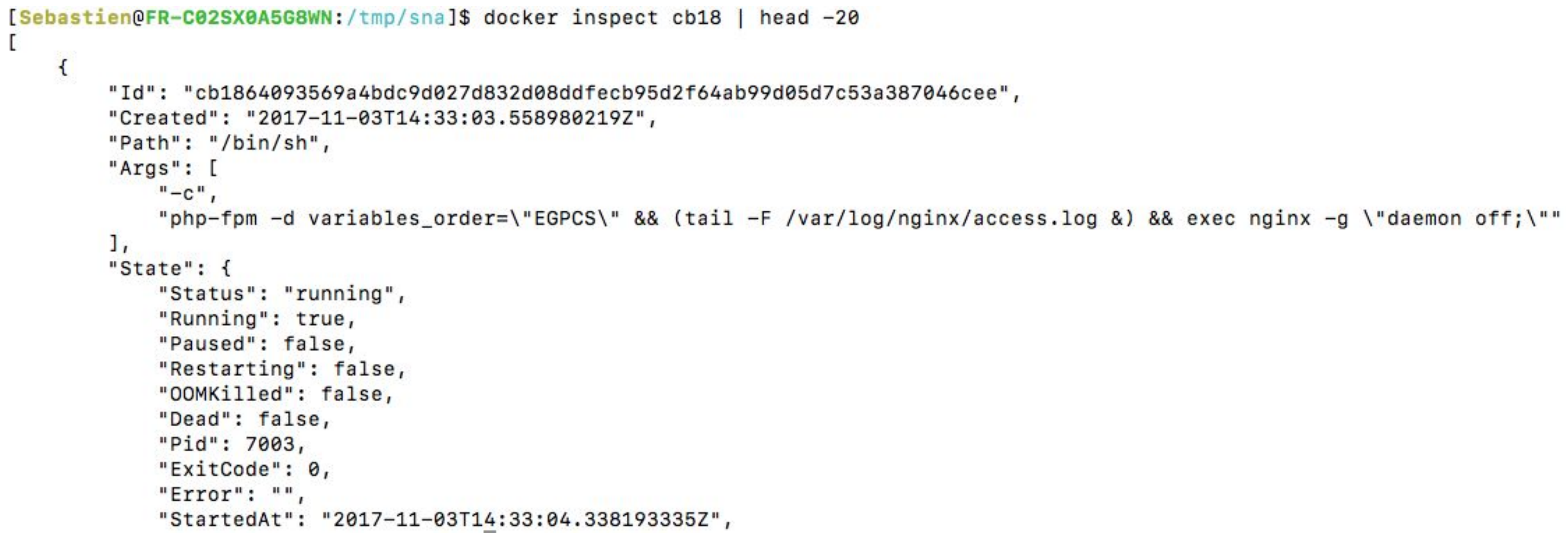
Clean container



```
[Sebastien@FR-C02SX0A5G8WN:/tmp/sna]$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                  NAMES
f32bfa5166cd        tutum/hello-world   "/bin/sh -c 'php-f..."   18 minutes ago      Up 18 minutes       0.0.0.0:8081->80/tcp   silly_babbage
[Sebastien@FR-C02SX0A5G8WN:/tmp/sna]$ docker kill f32bfa5166cd
f32bfa5166cd
[Sebastien@FR-C02SX0A5G8WN:/tmp/sna]$ docker ps -a | grep f32bfa5166cd
f32bfa5166cd        tutum/hello-world                            "/bin/sh -c 'php-f..."   18 minutes ago      Exited (137) 4 seconds ago
[Sebastien@FR-C02SX0A5G8WN:/tmp/sna]$ docker rm f32bfa5166cd
f32bfa5166cd
[Sebastien@FR-C02SX0A5G8WN:/tmp/sna]$ docker ps -a | grep f32bfa5166cd
```

We remove **definitively** this container

Get all information about container



```
[Sebastien@FR-C02SX0A5G8WN:/tmp/sna]$ docker inspect cb18 | head -20
[
    {
        "Id": "cb1864093569a4bdc9d027d832d08ddfecb95d2f64ab99d05d7c53a387046cee",
        "Created": "2017-11-03T14:33:03.558980219Z",
        "Path": "/bin/sh",
        "Args": [
            "-c",
            "php-fpm -d variables_order=\"EGPCS\" && (tail -F /var/log/nginx/access.log &) && exec nginx -g \"daemon off;\""
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 7003,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2017-11-03T14:33:04.338193335Z",
```

Lab1: Docker install

$ git checkout lab1

- Install docker on your server :

```
yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
yum install docker-ce
systemctl enable docker
systemctl start docker
```

- Install git on your server :

```
yum install git
```

- Clone the repo to download the lab resources :

```
git clone github.com/geothery/cesi
```

Lab1: Your first containers

- Start a tutum/hello-world container with daemon and port mapping options (80:80)

```
docker run -p 80:80 tutum/helloworld
```

- Try to access to http://your_public_ip:80

- Enter into your container and change the /www/index.php . Refresh your website. What's appends?

- Kill your container and start a new one. What happen ?

- Let's add a LoadBalancer on top of your web servers

Summary Part 1

We’ve learned how to:

- Manage containers lifecycle (start/stop/kill/rm)
- Get container information
- Expose a network port
- Mount a volume

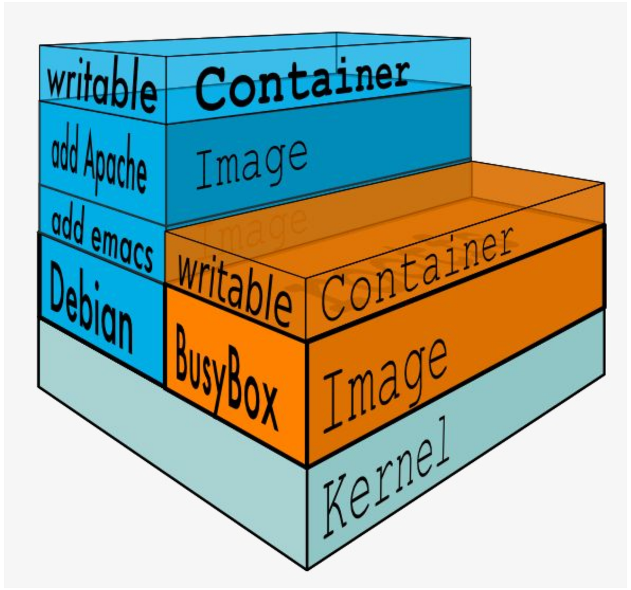
There are a lot of other possibilities like:

- Injecting environment variables
- Limit memory usage
- Changing the log driver
- Changing DNS / hostname / etc
- etc

Work with images

What is an image?

- An image is a collection of files + some meta data. (Technically: those files form the root filesystem of a container.)
- Images are made of layers, conceptually stacked on top of each other.
- Each layer can add, change, and remove files.
- Images can share layers to optimize disk usage, transfer times, and memory use.
- An image is a read-only filesystem



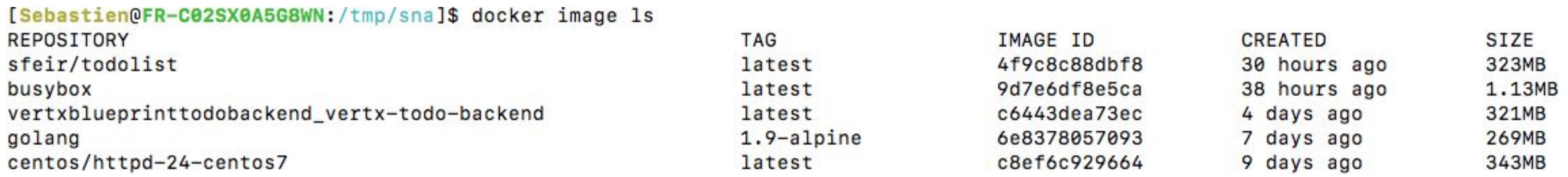
Images naming

An image is always like that: *<namespace>/<image>:<version>*

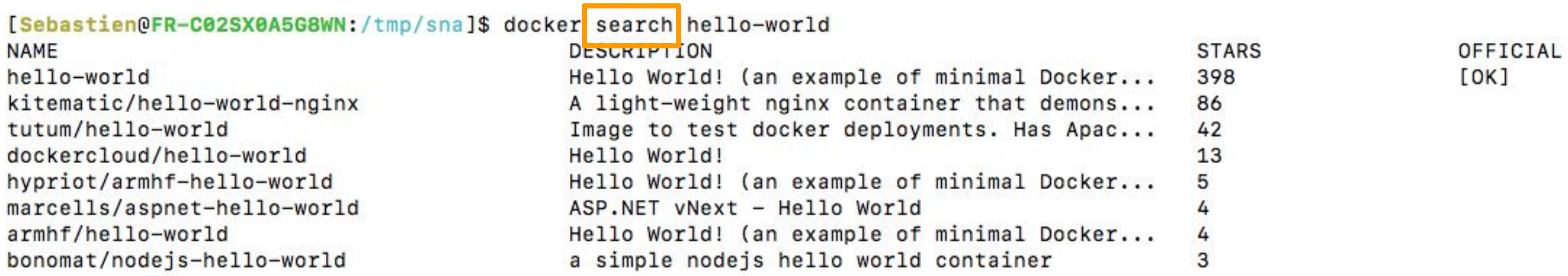
- namespaces can be:
  - Root like: ubuntu
  - User/org: jpetazzo/clock
  - Self-hosted: geoffrey.thery.org/dev/myimage

- Default version is “latest”

Showing current images

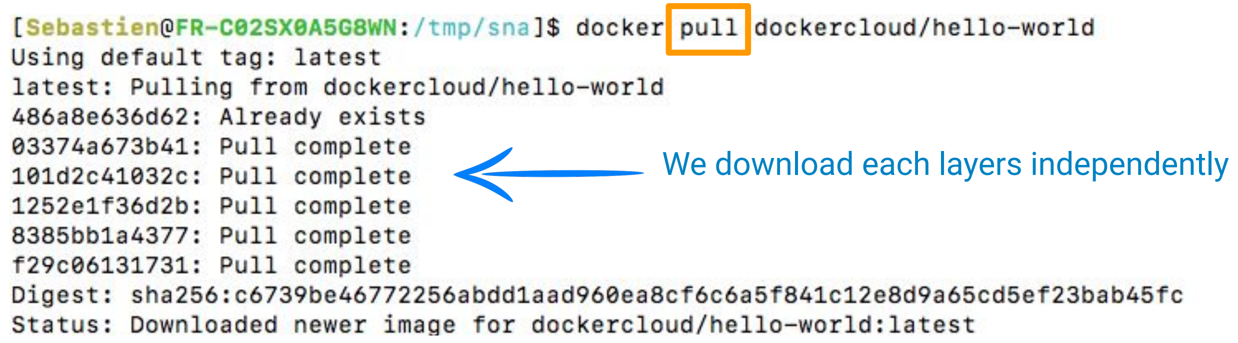

```
[Sebastien@FR-C02SX0A5G8WN:/tmp/sna]$ docker image ls
REPOSITORY                                                  TAG                 IMAGE ID            CREATED             SIZE
sfeir/todolist                                              latest              4f9c8c88dbf8        30 hours ago        323MB
busybox                                                     latest              9d7e6df8e5ca        38 hours ago        1.13MB
vertxblueprinttodobackend_vertx-todo-backend                latest              c6443dea73ec        4 days ago          321MB
golang                                                      1.9-alpine          6e8378057093        7 days ago          269MB
centos/httpd-24-centos7                                     latest              c8ef6c929664        9 days ago          343MB
```

Searching for images



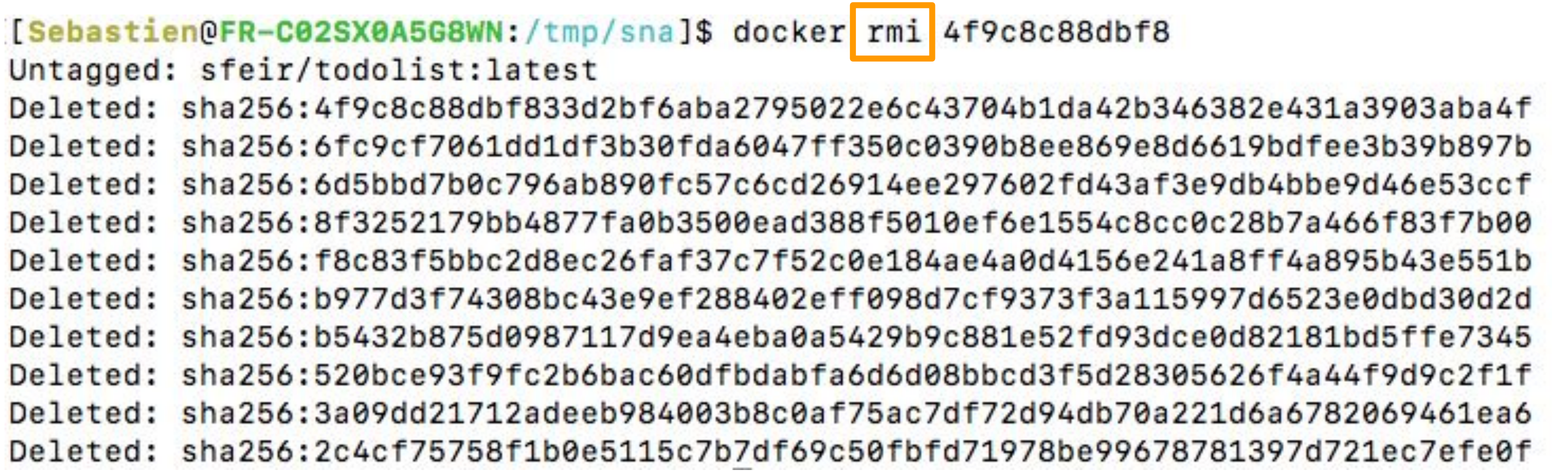
```
[Sebastien@FR-C02SX0A5G8WN:/tmp/sna]$ docker search hello-world
NAME                           DESCRIPTION                                     STARS     OFFICIAL
hello-world                    Hello World! (an example of minimal Docker...   398       [OK]
kitematic/hello-world-nginx    A light-weight nginx container that demons...   86
tutum/hello-world              Image to test docker deployments. Has Apac...   42
dockercloud/hello-world        Hello World!                                    13
hypriot/armhf-hello-world      Hello World! (an example of minimal Docker...   5
marcells/aspnet-hello-world    ASP.NET vNext - Hello World                     4
armhf/hello-world              Hello World! (an example of minimal Docker...   4
bonomat/nodejs-hello-world     a simple nodejs hello world container           3
```

Pulling images



```
[Sebastien@FR-C02SX0A5G8WN:/tmp/sna]$ docker pull dockercloud/hello-world
Using default tag: latest
latest: Pulling from dockercloud/hello-world
486a8e636d62: Already exists
03374a673b41: Pull complete
101d2c41032c: Pull complete
1252e1f36d2b: Pull complete
8385bb1a4377: Pull complete
f29c06131731: Pull complete
Digest: sha256:c6739be46772256abdd1aad960ea8cf6c6a5f841c12e8d9a65cd5ef23bab45fc
Status: Downloaded newer image for dockercloud/hello-world:latest
```

Cleaning images

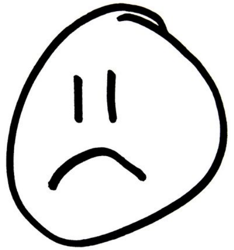


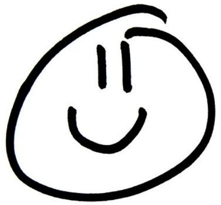
```
[Sebastien@FR-C02SX0A5G8WN:/tmp/sna]$ docker rmi 4f9c8c88dbf8
Untagged: sfeir/todolist:latest
Deleted: sha256:4f9c8c88dbf833d2bf6aba2795022e6c43704b1da42b346382e431a3903aba4f
Deleted: sha256:6fc9cf7061dd1df3b30fda6047ff350c0390b8ee869e8d6619bdfee3b39b897b
Deleted: sha256:6d5bbd7b0c796ab890fc57c6cd26914ee297602fd43af3e9db4bbe9d46e53ccf
Deleted: sha256:8f3252179bb4877fa0b3500ead388f5010ef6e1554c8cc0c28b7a466f83f7b00
Deleted: sha256:f8c83f5bbc2d8ec26faf37c7f52c0e184ae4a0d4156e241a8ff4a895b43e551b
Deleted: sha256:b977d3f74308bc43e9ef288402eff098d7cf9373f3a115997d6523e0dbd30d2d
Deleted: sha256:b5432b875d0987117d9ea4eba0a5429b9c881e52fd93dce0d82181bd5ffe7345
Deleted: sha256:520bce93f9fc2b6bac60dfbdabfa6d6d08bbcd3f5d28305626f4a44f9d9c2f1f
Deleted: sha256:3a09dd21712adeeb984003b8c0af75ac7df72d94db70a221d6a6782069461ea6
Deleted: sha256:2c4cf75758f1b0e5115c7b7df69c50fbfd71978be99678781397d721ec7efe0f
```

But wait...

How can I create my own images?

Manual process = *docker commit* = bad.





Automated process = *Dockerfile* = good.

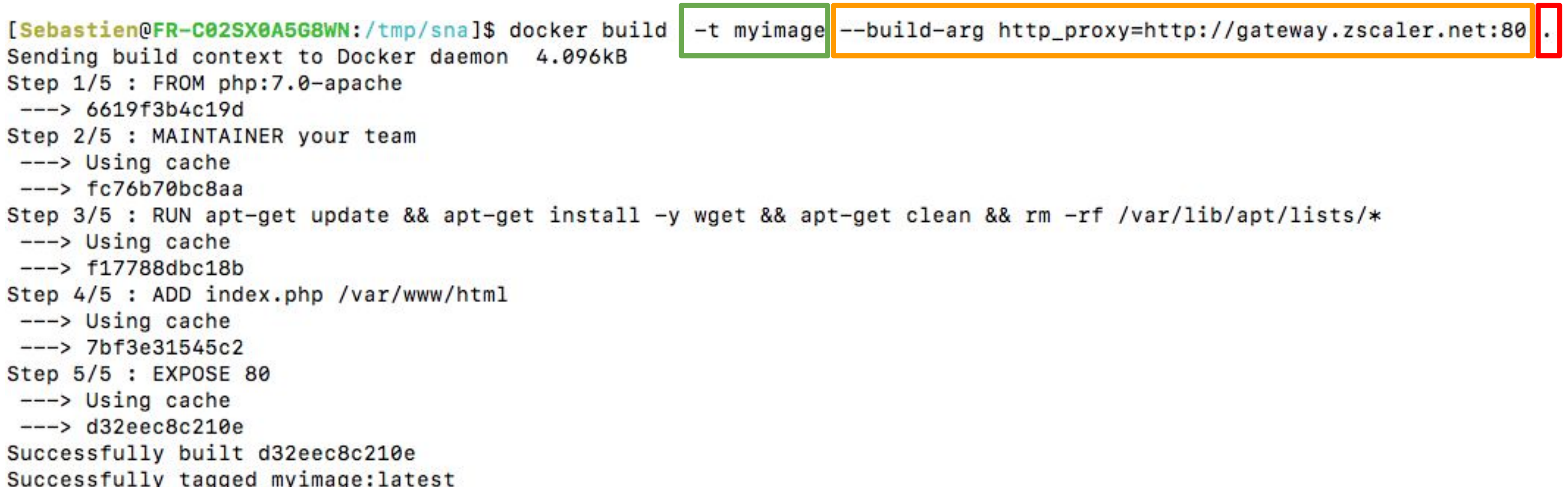
Dockerfile

```
# A basic apache server.
FROM php:7.0-apache


MAINTAINER your team
RUN apt-get update && apt-get install -y curl && apt-get clean && rm -rf /var/lib/apt/lists/*
ADD index.php /var/www/html


EXPOSE 80
CMD ["apache2-foreground"]
```

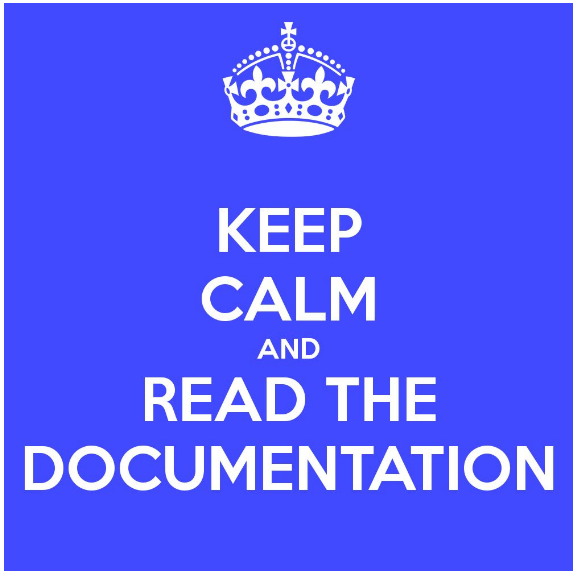
Building image



```
[Sebastien@FR-C02SX0A5G8WN:/tmp/sna]$ docker build  -t myimage --build-arg http_proxy=http://gateway.zscaler.net:80 .
Sending build context to Docker daemon  4.096kB
Step 1/5 : FROM php:7.0-apache
 ---> 6619f3b4c19d
Step 2/5 : MAINTAINER your team
 ---> Using cache
 ---> fc76b70bc8aa
Step 3/5 : RUN apt-get update && apt-get install -y wget && apt-get clean && rm -rf /var/lib/apt/lists/*
 ---> Using cache
 ---> f17788dbc18b
Step 4/5 : ADD index.php /var/www/html
 ---> Using cache
 ---> 7bf3e31545c2
Step 5/5 : EXPOSE 80
 ---> Using cache
 ---> d32eec8c210e
Successfully built d32eec8c210e
Successfully tagged myimage:latest
```

Dockerfile keywords

- FROM
- MAINTAINER
- ADD
- COPY
- ENV
- EXPOSE
- LABEL
- STOPSIGNAL
- USER
- VOLUME
- WORKDIR
- CMD
- ENTRYPOINT



https://docs.docker.com/engine/reference/builder/

Lab2: PHP Dockerisation

- Go in directory lab2
- Build an image for your php app with the Dockerfile

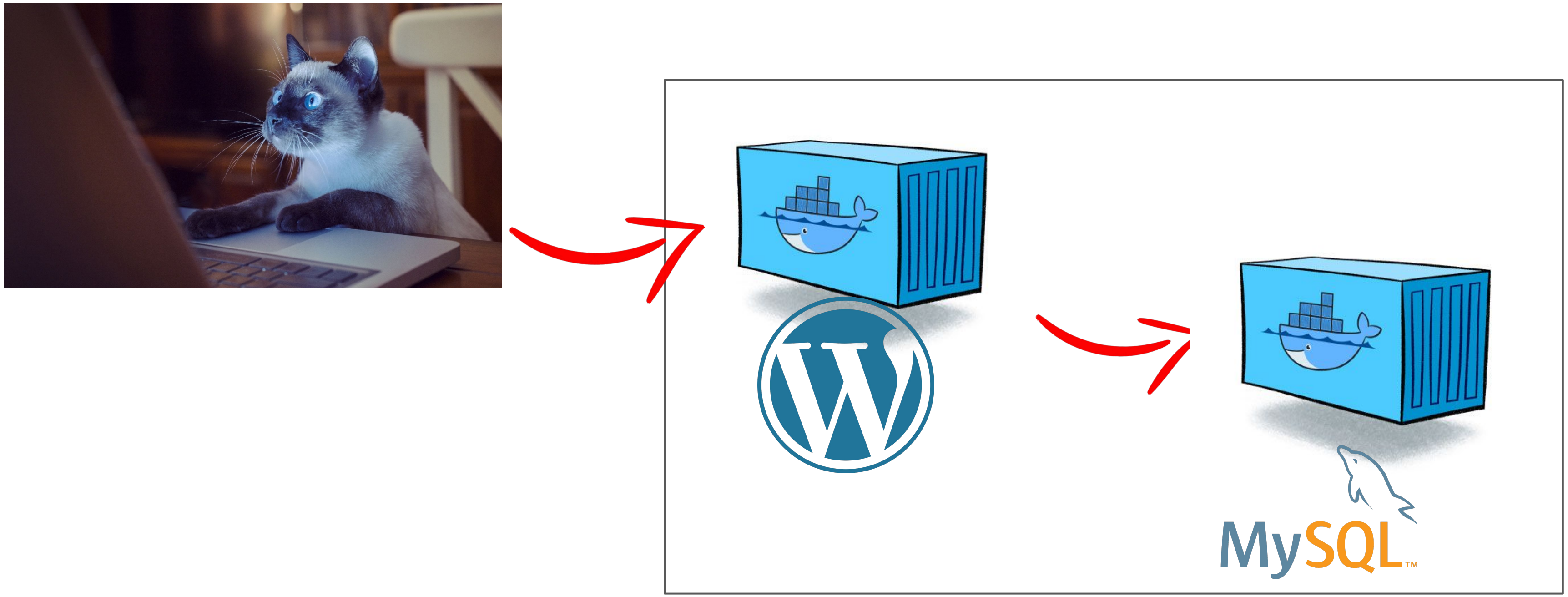
```
docker build -t myphpapp .
```

- Run your container

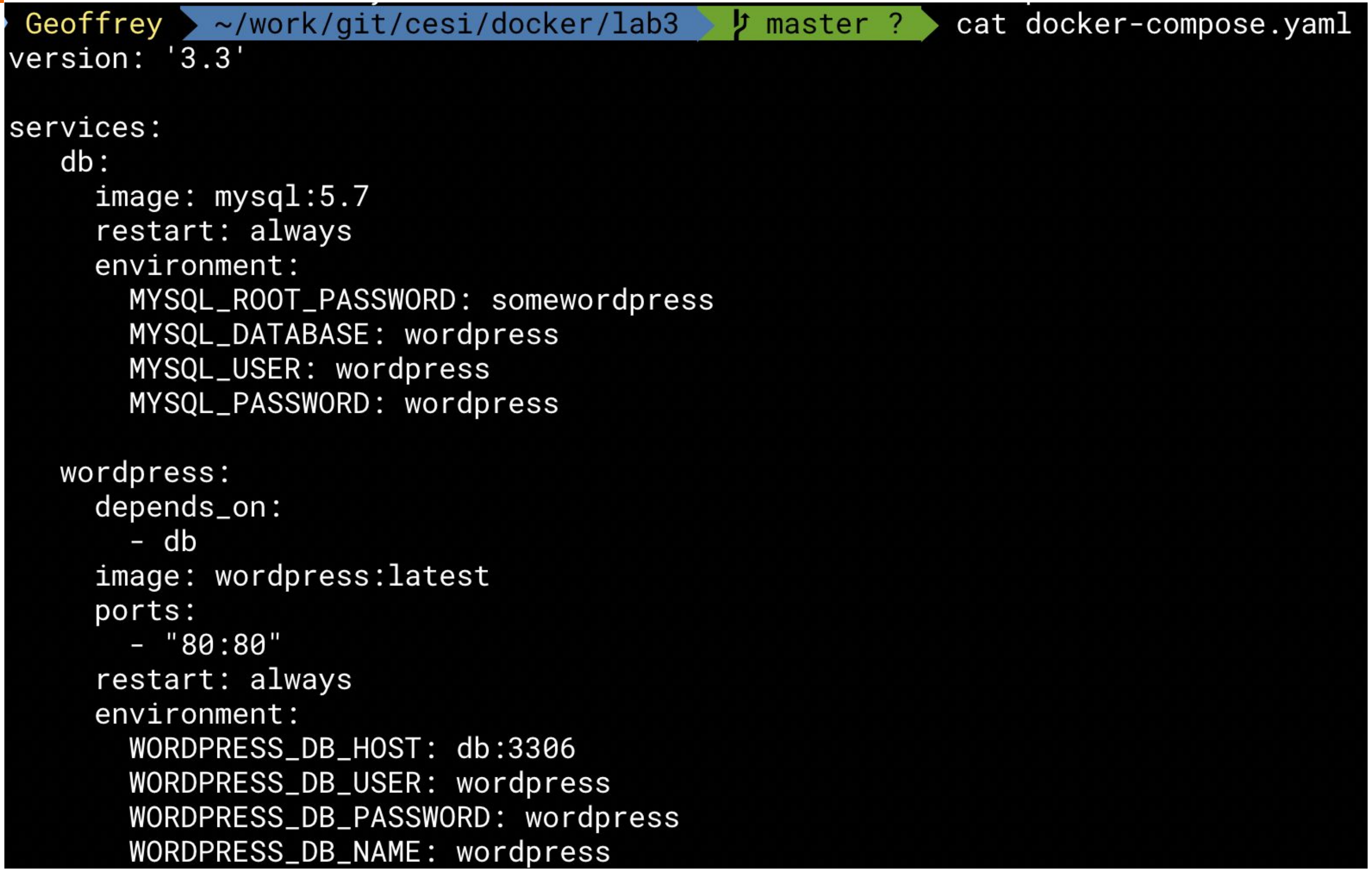
```
docker run -p 80:80 myphpapp
```

Multi-containers applications

Docker link



Docker-compose



```
 Geoffrey  ~/work/git/cesi/docker/lab3  master ?  cat docker-compose.yaml
version: '3.3'

services:
   db:
     image: mysql:5.7
     restart: always
     environment:
       MYSQL_ROOT_PASSWORD: somewordpress
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: wordpress

   wordpress:
     depends_on:
       - db
     image: wordpress:latest
     ports:
       - "80:80"
     restart: always
     environment:
       WORDPRESS_DB_HOST: db:3306
       WORDPRESS_DB_USER: wordpress
       WORDPRESS_DB_PASSWORD: wordpress
       WORDPRESS_DB_NAME: wordpress
```

Lab3: Compose my app with Wordpress

- go in directory lab3
- Install docker compose on your server (commands in installcompose file)
- look at the file docker-compose.yaml
- Start the stack: docker-compose up and check http://your_public_ip:80
- Finish the wordpress install

Summary Part 3

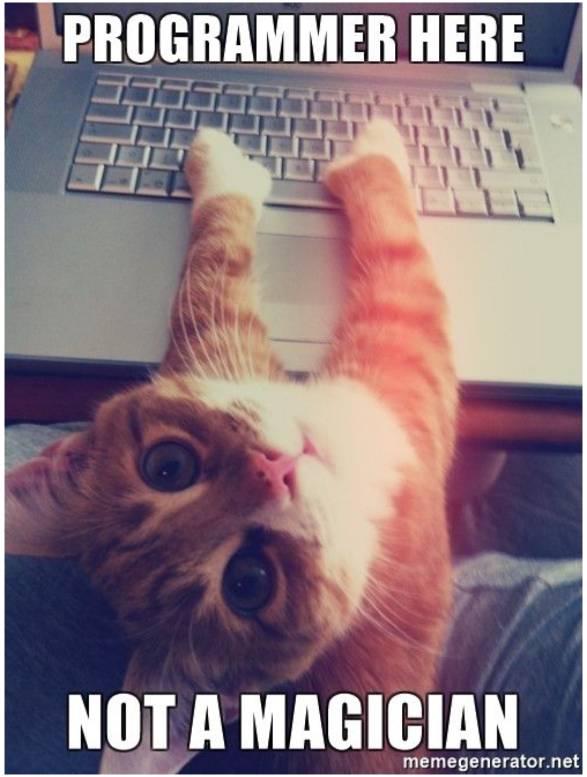
We've learned how to:

- Work with docker-compose (start/stop stack)

- Customize a container using environment variables

Docker in production

Lab: Troubleshooting

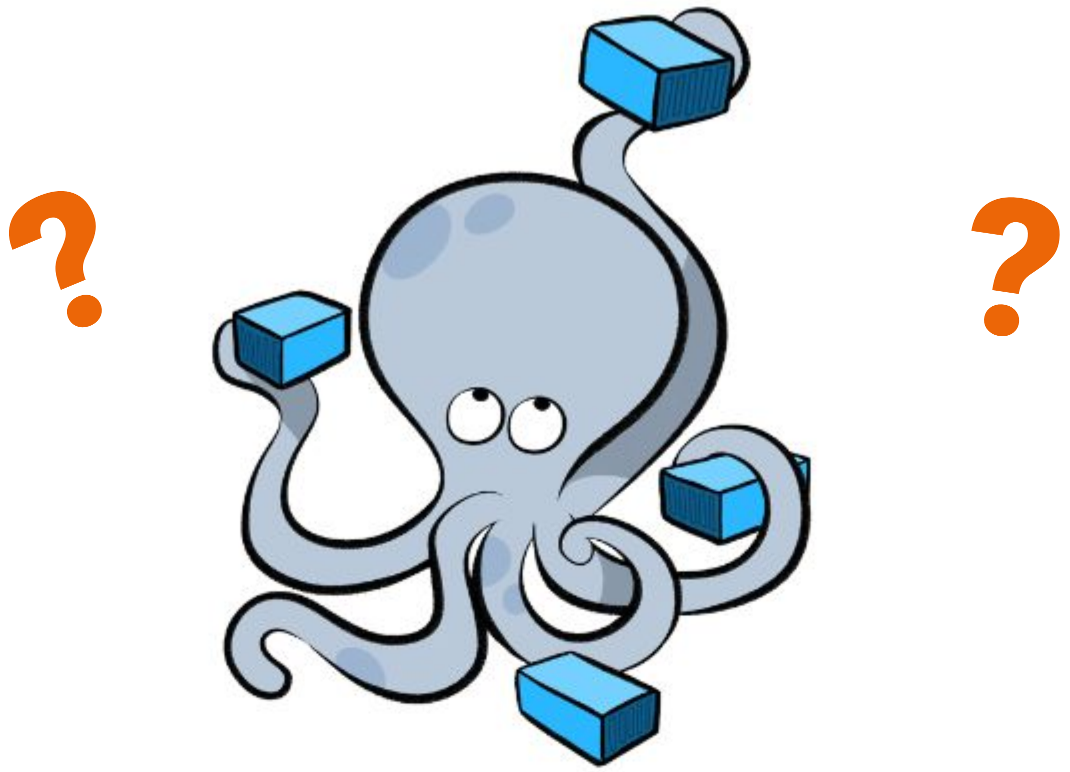


- Build the container and run it

  docker build -t myimage .

  docker run -d -p 80:8080 myimage
- Try to reach your application 5 times with your browser
- Whats appends ? Try to troubleshoot :)

Orchestration



Orchestration

kubernetes by Google

scaling

network orchestration

multiple hosts

pipeline compatible

healthcheck

rolling update

placement control

high availability

affinity & anti-affinity

loadbalancing

infra as code

RANCHER®

Thank you