Introduction à Docker

Alexandre Ancel <alexandre.ancel@ihu-strasbourg.eu>

Contenu original :
Johan Moreau <johan.moreau@ircad.fr>

IRCAD / IHU Strasbourg - Institut de Chirurgie Guidée par l'Image

04 juillet 2017 / 06 juillet 2017

Plan

Un conteneur : kezako ?

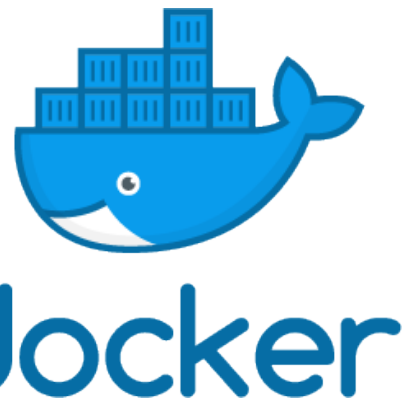
Un conteneur (jolie métaphore), c'est quoi ?

Les mêmes idées que la virtualisation, mais sans virtualisation :

- Agnostique sur le contenu et le transporteur
- Isolation et automatisation
- Principe d'infrastructure consistante et répétable
- Peu d'overhead par rapport à une VM ! [1]

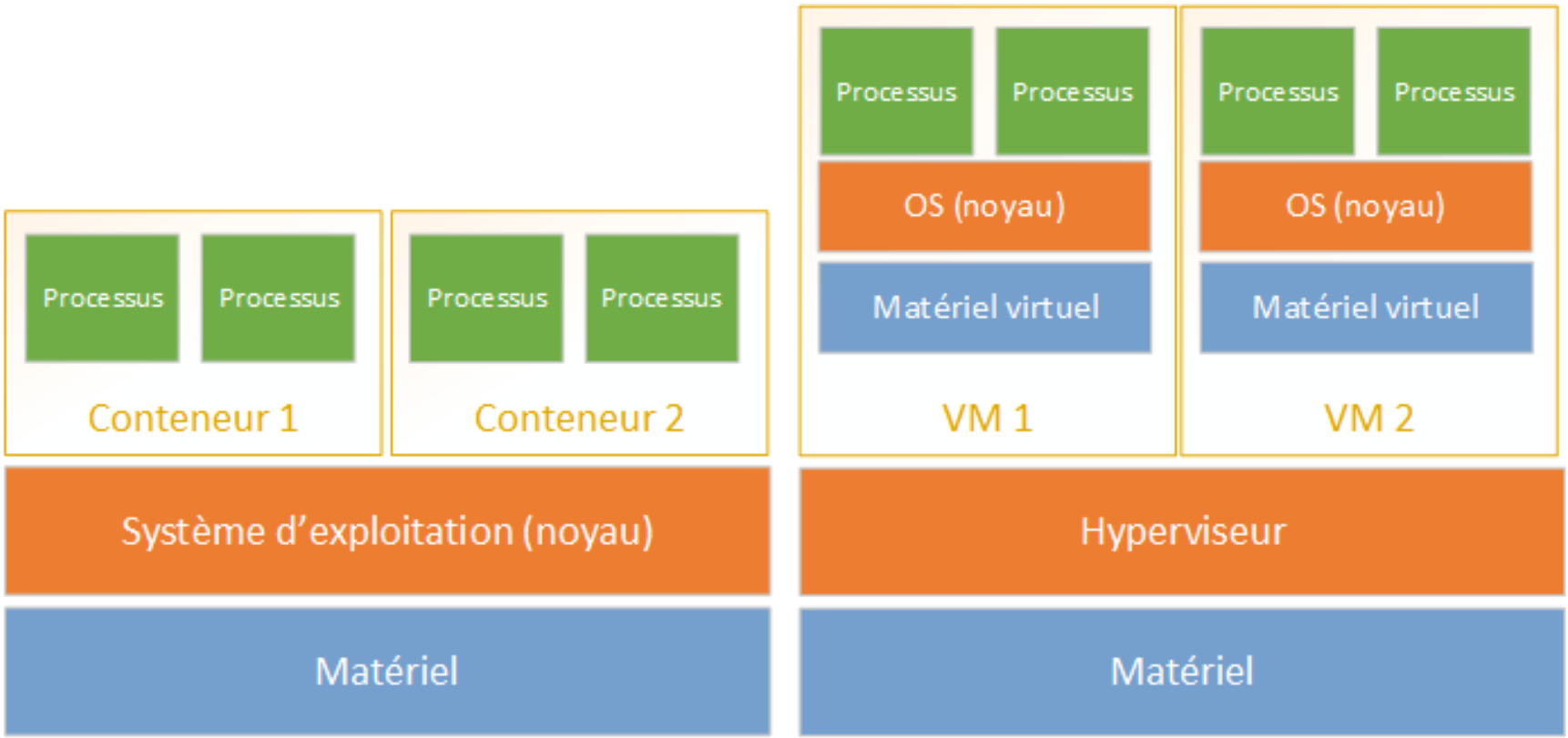
En gros, un super chroot (ou un jails BSD plus sympa) : Un des points forts de Solaris depuis plusieurs années. Techno existante aussi chez Google depuis longtemps. Rien de neuf, mais pourtant ...

Certains parlent de virtualisation "niveau OS" ou "légère", isolation applicative



[1] http://fr.slideshare.net/BodenRussell/kvm-and-docker-lxc-benchmarking-with-openstack

Différences entre VM et conteneur



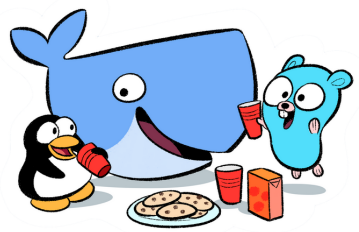
Les conteneurs

Les machines virtuelles

Liaison avec le noyau Linux [2] et autres OS

Une belle évolution de Linux :

- Mount namespace (Linux 2.4.19)
- PID namespace (Linux 2.6.24) - x PID/process
- Net namespace (Linux 2.6.19-2.6.24)
- User namespace (Linux 2.6.23-3.8)
- IPC namespace (Linux 2.6.19-2.6.30)
- UTS namespace (Linux 2.6.19) - host et domain
- cgroups (Linux 2.6.24) - gérer la limitation de ressource
- AUFS/BTRFS: FS/couche, mode union, copy on write



En gros, les éléments noyaux sont en place depuis Linux 2.6.32
Minimum requis pour la stabilité des fonctionnalités : 3.10

Mais natif depuis 2016 sur OSX et Windows

[2] `https://github.com/sdouche/presentation-docker/`

Qu'est ce que Docker vous permet de faire ?

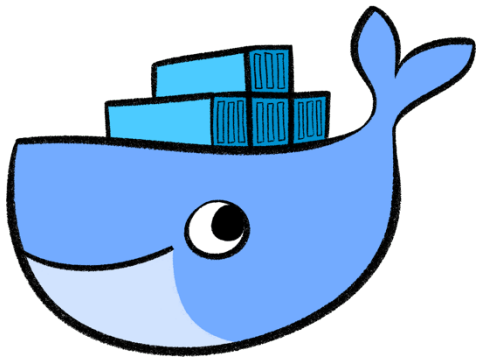
- **Proposer/diffuser des applications dans un environnement maitrisé (Docker/Dockerfiles)**
  - Problèmes de versions et de dépendances pour une application
    - Versions de dépendances : Version spécifique d'OS, installations personnelles ...
  - Proposer un package binaire universel
  - Documenter la procédure d'installation pour soi-même ou les curieux
  - $\Rightarrow$ Répétabilité, consistance
- **Déployer un ensemble d'applications (docker-compose)**
  - Description du fonctionnement d'un ensemble de services et de leurs interactions
  - Par exemple: Apache + PHP + Mariadb + Redis
- **Passage à l'échelle d'un ensemble d'applications (docker-compose/swarm) : multi-hôtes**
  - Augmenter le nombre de service lancé avec répartition de charge entre plusieurs machines.

Docker, la petite histoire

De dotcloud à docker

La petite histoire :

- DotCloud [3] équivalent FR à Heroku (PAAS)
- Construit sur LXC et AUFS + dev. kernel linux
- Développement interne d'un cli en python (dc)
- Pas mal de soucis sur la gestion des conteneurs: coucurrence ...



Docker - vers la simplification :

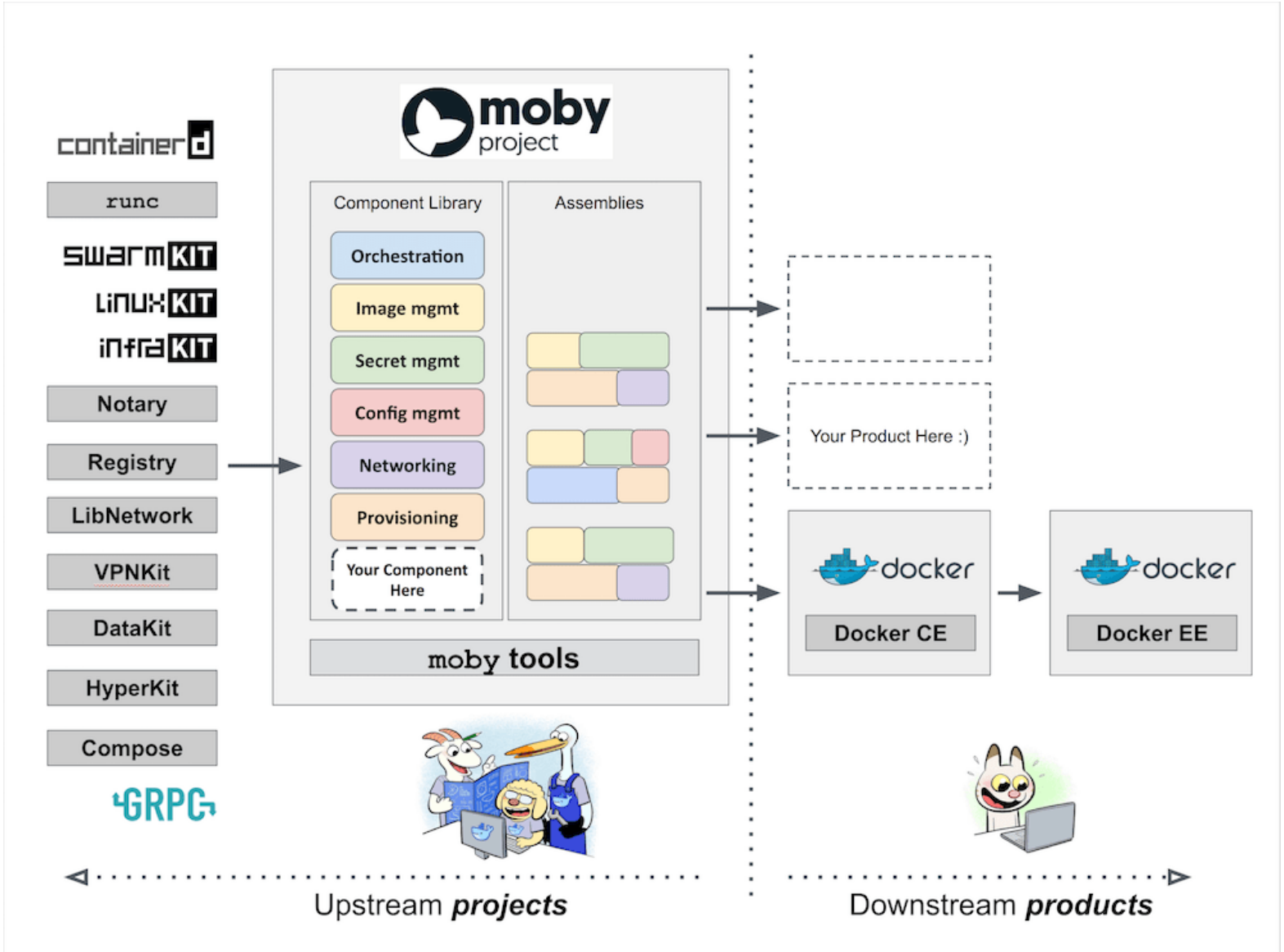
- Projet jeune en Golang [4] : 1er commit le 18/01/2013
- Dans le top15 sur GitHub (approche les 800 contributeurs) en 2015
- 21 millions de téléchargement fin 2014
- Linux 64bits avec LXC/libcontainer [5]

[3] http://fr.slideshare.net/jpetazzo/introduction-to-docker-december-2014-tour-de-france-bordeaux-special-edition

[4] http://linuxfr.org/news/la-folie-docker

[5] https://www.flockport.com/lxc-vs-docker/

En fait docker, c'était pour rire, bienvenu à Moby



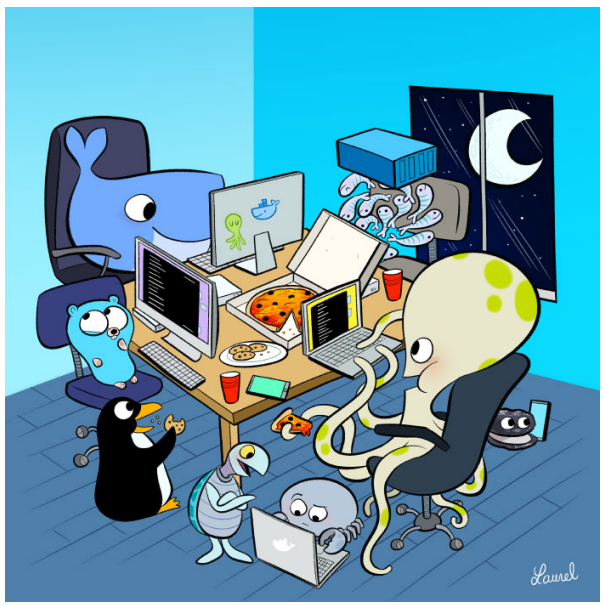
Docker

Terminologie :

- client/server : outil utilisant l'API du serveur/Daemon
- index : répertoire public (https://index.docker.io/)
- image : conteneur en lecture seule (couches = snapshot)
- conteneur : élément manipulable

Analogie avec le développement objet :

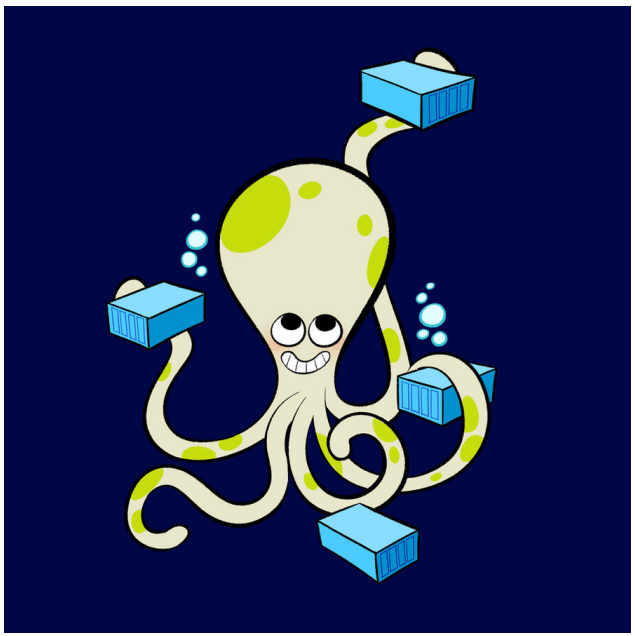
- Images équivalentes aux classes
- Les couches sont équivalentes à l'héritage
- Les conteneurs sont des instances



Docker

Les points forts :

- Installation simple (Linux, OSX, Windows) [6]
- Ligne de commande très sympathique (docker help)
- Langage de description des images (avec notion de parent)
- Communauté très active (trop ?)
- API pour pilotage, écosystème jeune mais déjà énorme :
  - Gui [7], Orchestration, hébergement cloud, intégration continue, OS, ...



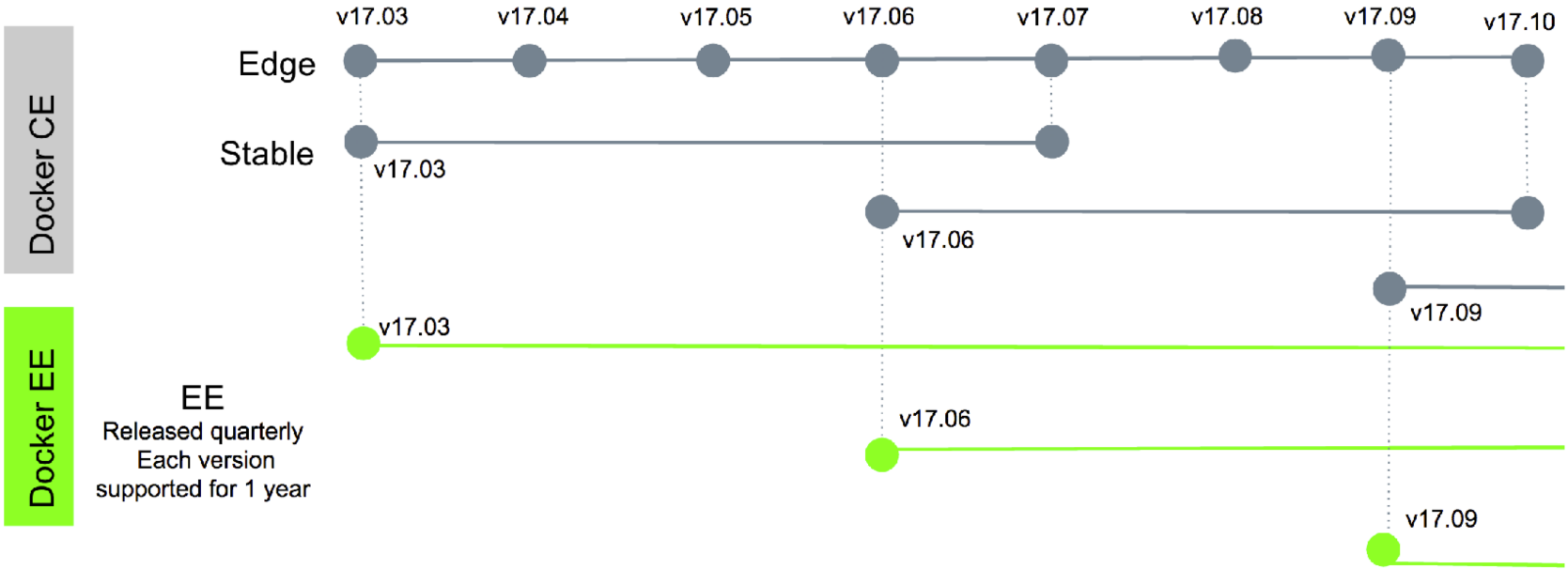
[6] http://docs.docker.com/installation/

[7] http://linuxfr.org/news/logiciels-pour-survivre-avec-docker

Évidement, il faut vivre : CE vs EE[8]

[8] RBAC LDAP/AD, gestion des images avancées, scan d'images, etc ...

Version en MM.YY



Plan

Les commandes de bases

Préambule

- Utilisation de docker
  - Sudo ou non ? Groupe docker
- Syntaxe de la commande
  - A l'ancienne:

```
$ docker <command> <options> <image> <commande>
```

  - Exemple:

```
$ docker run hello-world
```

  - Avec les regroupements logiques (management commands) depuis la version 1.13[9]:

```
$ docker <management_command> <command> <options> <image>
```

  - Exemple:

```
$ docker container run hello-world
```

[9] http://blog.arungupta.me/docker-1-13-management-commands/

Docker 101[10]

```
$ docker container run -i -t ubuntu /bin/bash
```

- run : on veut lancer le conteneur
- -i -t : on veut un terminal et être interactif avec lui
- ubuntu : l'image à utiliser pour ce conteneur
- /bin/bash : on lance bash

```
$ docker container run -i -t ubuntu /bin/bash
root@0bc82356b52d9:/# cat /etc/issue
Ubuntu 14.04.2 LTS
root@0bc82356b52d9:/# exit
```

[10] Les images de base sont très légères pas de ifconfig (net-tools)/ping(iputils-ping)

Démarrage d'un conteneur

- En quoi consiste le démarrage du container:
- Recherche de l'image
  $\Rightarrow$ Si l'image n'existe pas en local, alors téléchargement via le hub. Construction du système de fichiers au sens Linux
- Démarrage du container
- Configuration de l'adresse IP du container
  $\Rightarrow$ Ainsi que de la communication entre l'extérieur et le container
- Capture des messages entrées-sorties

1 seul processus

- Philosophiquement, n'exécute qu'un seul processus à la fois
- un container = une application (ou processus)
- pas d'exécution de daemons, de services, ssh, etc.
  - même le processus init n'existe pas
  - sinon l'utilisation des outils particuliers tels que supervisord, forever, ...

```
docker container top mycontainer
docker container inspect --format {{.State.Pid}} `docker ps -q`
```

Docker in the shell : Images

Les principales commandes :

Listing 1: Gestion des images

```
# Lance le daemon docker (Si pas de configuration en service)
sudo /usr/bin/docker -d &

# Recherche une image dans la registry (Importance du "/")
docker search ubuntu

# Recuperation de la derniere image stackbrew/ubuntu images
docker image pull stackbrew/ubuntu

# Affiche l'historique d'une image
docker image history stackbrew/ubuntu

# Affiche la liste des images locales
docker images
```

Docker in the shell : Conteneurs

Les principales commandes :

Listing 2: Gestion des conteneurs

```
# Instancie un conteneur a partir d'une image en mode interactif
docker container run -i -t stackbrew/ubuntu /bin/bash
docker container run -i -t --rm --name myUbuntu ubuntu /bin/bash

# Lien de 2 conteneurs
docker container run -ti --link redis:db --name webapp ubuntu bash

# Lance un conteneur en arriere plan
docker container run -d -p 8888:80 ubuntu # export 8888 on master

# Affiche les conteneurs actifs (-a pour les affichers tous)
docker container ps
docker container logs myUbuntu
docker container exec myUbuntu /bin/bash

docker container start myUbuntu # Relance un conteneur
docker container attach myUbuntu # Reprendre la main
docker container stop myUbuntu # SIGTERM suivi d'un SIGKILL
docker container kill myUbuntu # SIGKILL directement
```

Les images

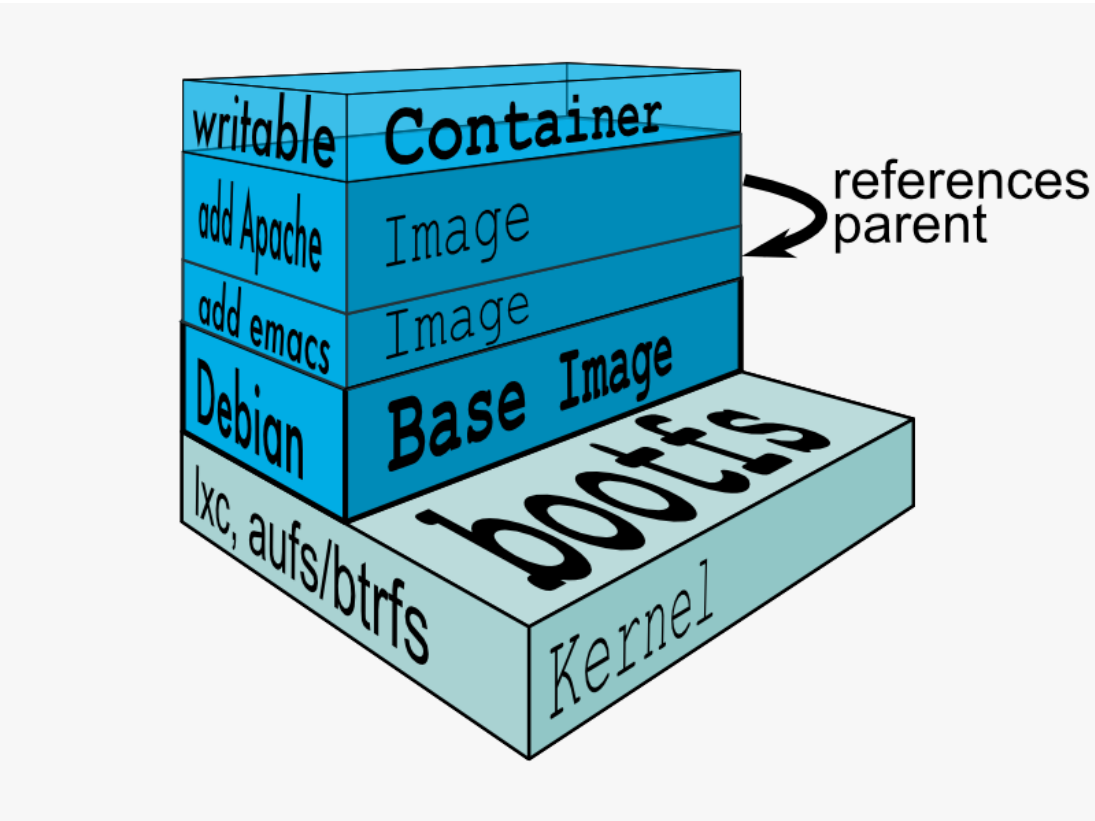
Docker hub

Le hub public :

- Dépôt public (push/pull gratuit)
- Dépôt d'images officielles (sans "/"), et d'images tiers,
  - Systèmes d'exploitation:
    - debian, ubuntu, centos ...
    - Mention spéciale: alpine: Micro-distribution
      alpine:3.5 : 36.5 Mo
      ubuntu:16:04 : 184 Mo
  - Services conteneurisés:
    - php, nginx, mariadb, ...
- Collection de services supplémentaires :
  - Builds automatisés (lier des dépôts github/bitbucket pour lancer un build suite à un commit)
- store.docker.com: Référentiel d'image version entreprise

Couches d'une image Docker

- Performance (*5)
- Ré-utilisabilité
- Lecture seule donc :
  - diff et versioning



Tag et espace de nom des images

Les images peuvent avoir des tags :

- Les tags symbolisent des différences de version d'une image
- C'est le tag :latest qui est utilisé par défaut

Les images disposent de trois espaces de nom :

- Racine : ubuntu
- Utilisateur et organisations : frapsoft/ts-node
- Auto-hébergées (le serveur) : localhost:5000/myapache

Création d'images Docker[11]

Différentes méthodes en ligne de commande:

- image import : charge une archive de fichiers, comme couche de base
- container commit : crée une nouvelle couche (+ image) depuis un conteneur
- image build : script de suite de commandes de création automatisée d'image

Description et construction : Dockerfiles

Listing 3: Dockerfile

```
FROM debian:wheezy
ADD README.md /tmp/
```

Listing 4: Exécution d'un build Docker

```
$ docker build -t readme -f Dockerfile .
Sending build context to Docker daemon 3.072
Step 1/2 : FROM debian:wheezy
 ---> bbd62956fac7
Step 2/2 : ADD README.md /tmp/
 ---> Using cache
 ---> 28ec2deea0ba
Successfully built 28ec2deea0ba
```

[11] http://www.centurylinklabs.com/more-docker-image-cache-tips

Instructions (DSL) du Dockerfile

Les instructions sont peu nombreuses :

- Image de base (ou scratch) : FROM,
- Environnement :
  LABEL, MAINTAINER, ENV, USER, WORKDIR, ARG,
- Ajouts de fichiers (contexte) : ADD, COPY,
- Commande à la construction de l'image : RUN,
- Commandes au lancement du conteneur :
  CMD, ENTRYPOINT[12],
- Ports/volumes accessibles: EXPOSE, VOLUME,
- Autres commandes:
  ONBUILD, STOPSIGNAL, HEALTHCHECK, SHELL
- ...

[12] commande de base != de CMD, commande par défaut

Commande RUN

- Pour chaque instruction RUN, un conteneur temporaire (8xxxxxxxx) est créé depuis l'image de base.
- La commande RUN est exécutée dans ce conteneur,
- Le conteneur est commité en une image intermédiaire (7yyyyyyyy),
- Le conteneur intermédiaire (8xxxxxxxx) est supprimé, Le résultat, l'image intermédiaire, servira d'image de base pour l'étape suivante,
- etc..

Point d'entrée: ENTRYPOINT/CMD

- Combinaisons:
  - ENTRYPOINT ou CMD:
    Spécifie la commande à lancer au démarrage
  - ENTRYPOINT et CMD:
    La commande prend alors la forme:
    ${ENTRYPOINT} ${CMD}
    CMD est, dans ce cas, surchargeable au lancement avec run
- Formes de lancement (valable aussi pour RUN):
  - Forme shell (mod. SHELL) : RUN echo hello
    ⇒ /bin/sh -c echo hello
  - Forme exec : RUN ["echo", "hello"]
    ⇒ echo hello
- Forme shell :
  l'application ne recevra pas les signaux envoyés par stop et kill

Exercice : Créez un conteneur ping avec argument

En partant de l'image *alpine*
Créez un Dockerfile appelant ping et prenant en paramètre du conteneur le host (localhost par défaut)

Exercice

Listing 5: Dockerfile pour ping

```
FROM alpine
ENTRYPOINT ["ping"]
CMD ["localhost"]
```

Multi-stage build [13] [14]

- Être capable d'avoir plusieurs FROM dans un Dockerfile
- Récupérer des artefacts issus des "stages" précédents les nièmes FROM
- L'intérêt réside dans la réduction du poids d'une image
- Cela est utile pour des artefacts "construits" dans dans des stages de l'image (ou pour réduire des installations trop lourdes)
- Nécessite une version récente de docker (17.05)

[13] `https://stefanscherer.github.io/use-multi-stage-builds-for-smaller-windows-images` & Builder pattern

[14] Présent que sur les dernières versions (edge ou 17.05), sinon voir Builder Pattern avec un shell script

Multi-stage build & Builder pattern[15]

Builder pattern

```
FROM golang:latest
COPY hello.go .
RUN go build -o hello hello.go
```

```
FROM alpine:latest
COPY hello .
CMD ["./hello"]
```

```
docker image build --tag hgo:bp \
--file Dockerfile1 .

docker container create \
--name hgo-bp hgo:bp
docker container \
cp hgo-bp:/go/hello ./hello

docker image build \
--tag hgo:latest \
--file Dockerfile2 .
```

Multi-stage builds

```
FROM golang:latest
COPY hello.go .
RUN go build -o hello hello.go

FROM alpine:latest
COPY --from=0 ./hello .
CMD ["./hello"]
```

[15] https://gobyexample.com/hello-world

La persistance des données

Volume

- Sortir du système image/read-only et conteneur/writable
  - Pour profiter des performances natives d'I/O disques,
  - Pour ne pas intégrer les modifications de fichiers dans une couche (Pour ne pas commiter)
- Interêt:
  - Conserver des données quand un conteneur est supprimé
  - Partager des fichiers/dossiers entre conteneurs
  - Partager des fichiers/dossiers entre hôte et conteneurs
- Options:
  - Utilisation du -v (–volume)
  - Utilisation du –volumes-from (Masquage possible dans le conteneur)

Volume

- Initialisation du volume à la création du conteneur avec copie des données.
- Volumes anonymes (docker inspect):

```
$ docker run -t -i -v /data ubuntu /bin/bash
```

- Volumes nommés :

```
$ docker volume create --name dataVolume
$ docker run -t -i -v dataVolume:/data ubuntu /bin/bash
```

- Montage inter-conteneur:

```
$ docker volume create --name dataVolume
$ docker run -t -i -v dataVolume:/data:rw --name myUbuntu ubuntu /bin/bash
$ docker run -t -i --volumes-from myUbuntu ubuntu /bin/bash
```

- Montage depuis le système hôte :

```
$ docker run -t -i -v ${PWD}:/data:ro ubuntu /bin/bash
```

Exercice: Volumes

- Créer un conteneur Ubuntu avec un volume initialisé avec le contenu de /etc (du conteneur)
- Utiliser un second conteneur pour faire une archive du contenu du volume créé et récuperez cette archive sur votre système avec un point de montage

Exercice: Volumes

- Créer un conteneur Ubuntu avec un volume initialisé avec le contenu de /etc (du conteneur)

```
#!/bin/bash

docker container create -ti -v myEtc:/etc \
  --name myUbuntu ubuntu
```

- Utiliser un second conteneur pour faire une archive que vous récuperez sur votre système avec un point de montage

```
#!/bin/bash

docker container run -d --rm \
  -v myEtc:/data \
  -v ${PWD}:/backup ubuntu \
  bash -c 'cd /data && tar czvf /backup/backup.tar.gz .'
```

Le réseau

Network - au démarrage

Au démarrage du daemon docker:

- création du bridge "docker0"
- une adresse IP privé ainsi qu'une adresse MAC sont assignées au bridge
- configuration des tables de routage (route et iptables)

Toute création de container entraîne la création de deux paires d'interfaces:

- une dans le container : eth*
- une autre dans la machine hôte : veth*
- toutes les deux reliées au bridge et fonctionne comme un pipe
- génération d'une adresse IP ainsi qu'une adresse MAC pour le container
- configuration de la route par défaut dans le container

Network

Publication des ports internes :

- –publish-all, -P
  - Pensez au docker inspect dans ce cas !
- -p x:y

Liens entre conteneurs (deprecated) :

```
docker run -d --name mydb mysql
docker run -d --name myphp --link mydb:mysql php
docker run -d --link myphp:php -p 80:80 my-nginx-img
```

Isolation des réseaux conteneurs (+ DNS interne) :

```
docker network create --subnet 10.0.9.0/24 --opt encrypted myNetwo
docker run -d --network myNetwork --name mydb mysql
```

Exercice : Accédez à votre application en réseau

En utilisant l'image *php:7.0-apache*
Afficher l'index.php contenant :
<?php echo "Hello tout le monde";?>

Exercice

Listing 6: Lancer un conteneur Php avec un montage

```
docker run -d -p 8080:80 --rm --name my-apache2 \
  -v "$PWD:/var/www/html" php:7.0-apache
```

Exercice : Netcat

- Utilisation de la commande netcat pour faire communiquer 2 hôtes (image alpine)
- Côté serveur:

```
while true; do nc -v -l -p 1234 ; done
```

- Coté client:

```
<cmd> | nc <IP ou hostname> 1234
```

- Envoyez, par exemple, le nom d'hote et la date.

Exercice

```
docker run -d --rm --name alpine_1 alpine \
  /bin/sh -c 'while true; do nc -v -l -p 1234 ; done'
```

```
# Does not work
docker run -ti --rm --name alpine_2 alpine \
  /bin/sh -c 'cat /etc/issue | nc alpine_1 1234'

# Works (with IP of alpine_1)
docker run -ti --rm --name alpine_2 alpine \
  /bin/sh -c 'cat /etc/issue | nc 172.17.0.3 1234'

# Works
docker run -ti --rm --name alpine_2 --link alpine_1 alpine \
  /bin/sh -c 'cat /etc/issue | nc alpine_1 1234'

# With networks
docker network create net_nc_srv
docker run -d --rm --name alpine_1 --network net_nc_srv alpine \
  /bin/sh -c 'while true; do nc -v -l -p 1234 ; done'
docker run -d --rm --name alpine_2 --network net_nc_srv alpine \
  /bin/sh -c 'cat /etc/issue | nc alpine_1 1234'
```

Exercice : Créez un Dockerfile et accédez à votre application en réseau

- Récupérez (git clone) `https://github.com/JohanMoreau/dev-koans/tree/master/Python/FlaskBasic`
- Partez d'une image *alpine*
  - Copier le répertoire récupéré dans le conteneur (avec arborescence)
  - Installer pip :

```
apk add --update py2-pip
```

  - Installer les dépendances :

```
pip install --no-cache-dir -r /usr/src/app/requirements.txt
```

  - Lancer l'application app.py avec python.

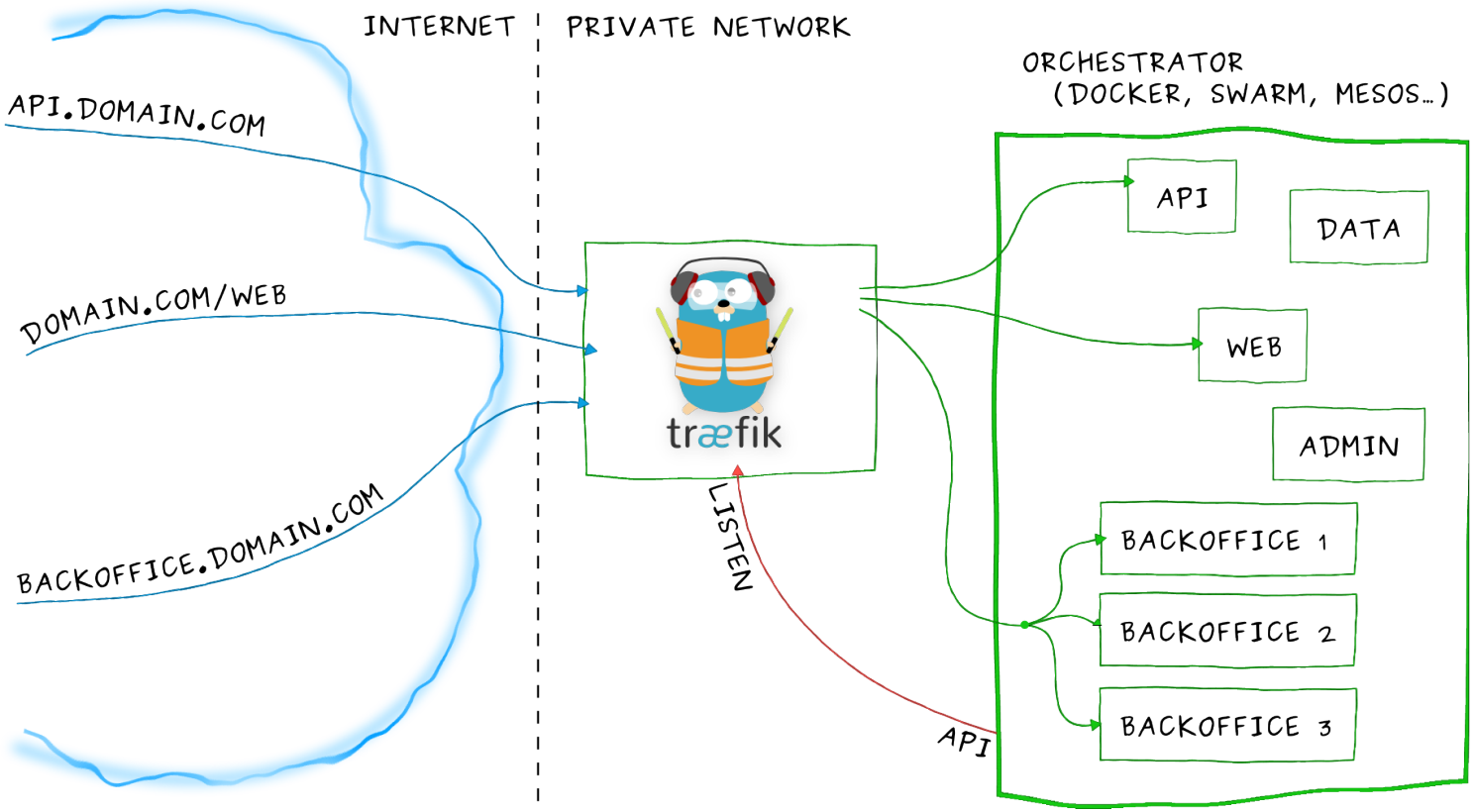
Exercice

Listing 7: Dockerfile pour Flask

```
# our base image
FROM alpine:latest
# Install python and pip
RUN apk add --update py2-pip

# Make missing directories
RUN mkdir -p /usr/src/app/templates

# install Python modules needed by the Python app
COPY ./requirements.txt /usr/src/app/
RUN pip install --no-cache-dir -r /usr/src/app/requirements.txt

# copy files required for the app to run
COPY ./app.py /usr/src/app/
COPY ./templates/index.html /usr/src/app/templates

# tell the port number the container should expose
EXPOSE 5000

# run the application
CMD ["python", "/usr/src/app/app.py"]
```

Network

- Reverse-proxy :
  - nginx (https://github.com/jwilder/nginx-proxy),
  - traefik[16] (https://github.com/containous/traefik), HAProxy, ...



[16] https://traefik.io/, image sous https://creativecommons.org/licenses/by/3.0/fr/

La sécurité

Sécurité

- Produit jeune, niveau théorique plus faible qu'un hyperviseur
- Qualité des images, nouveaux outils :
  - Lynis : `https://cisofy.com/lynis/`
  - Docker-bench-security :`https://dockerbench.com/`
- Possibilité de limiter l'accès aux ressources : Seccomp[17] (profil apparmor, ...), capabilities[18] (example : cap-drop ALL , cap-add CHOWN)
- Registry privée

[17] `http://training.play-with-docker.com/security-seccomp/`

[18] `http://training.play-with-docker.com/security-capabilities/`

Sécurité: Accès root et volumes

- Problème:
  - Par défaut, root dans un conteneur
  - on peut monter n'importe quel volume

```
$ docker run -t -i -v /etc:/data alpine
```

  - Modifications des fichiers du système hôte (environnement, configuration ...)
- Solutions:
  - Attention à qui sont donnés les doits pour l'accès à docker ![19]
  - OK pour une machine personnelle, moins pour des machines de calcul (Singularity)
  - Créer des utilisateurs dans les conteneurs (USER)
  - Utiliser l'option "–user"

[19] https://docs.docker.com/engine/security/security/#docker-daemon-attack-surface

Public vs private Registry

Private Registry :

- Stocke et distribue les images Docker
- De nombreux Registry hébergés disponibles:
  - Docker Hub, AWS ECR, ...
- Peut être self-hosted avec plusieurs intégrations pour le stockage:
  - Local, AWS S3, Ceph, OpenStack Swift, ...

Exercice : Private Registry

- Lancez une registry (https://hub.docker.com/_/registry/) en localhost sur le port 5000 avec un stockage local
- Récupérez l'image hello-world
- Mettre un tag sur l'image ci-dessus pour votre registry
- Poussez cette nouvelle image vers votre registry

Exercice : Private Registry

Listing 9: Unsecure private registry language

```
#!/bin/sh
echo 'DOCKER\_OPTS="--insecure-registry 127.0.0.1:5000"' \
  >> /etc/docker/docker

docker run -d -p 5000:5000 --name registry \
  -v $(pwd)/registry-data:/var/lib/registry registry:2

docker tag hello-world 127.0.0.1:5000/hello-world

docker push 127.0.0.1:5000/hello-world
```

Haute-disponibilité vs Migration

Stratégie :

- Les conteneurs ne sont pas migrables à chaud
- Choix d'une architecture scalable permettant la haute-disponibilité, tout en négligeant la perte d'un noeud
- L'architecture logiciel doit tenir compte sur de ce design

Des exemples

Exemples DevOps

pour "le Dev" :

- Multiples environnements (tests, dev, branches, ...)
- Compilation/Exécution multi-[os—jvm—tools—...]
- Utilisation de conteneurs pré-chargés avec des data pour les tests
- Outils spécifiques disponibles : plugins IntelliJ, Eclipse [20], ...

pour "l'Ops" :

- Rapidité de déploiement
- Force les bonnes pratiques (microservice, description donc documentation, ...)
- Déploiement récurrent de serveur
- Gestion des vulnérabilités : mise à jour d'une des couches, test, ...

[20] https://www.voxxed.com/blog/2015/06/docker-tools-in-eclipse/

Exemple pour tout le monde

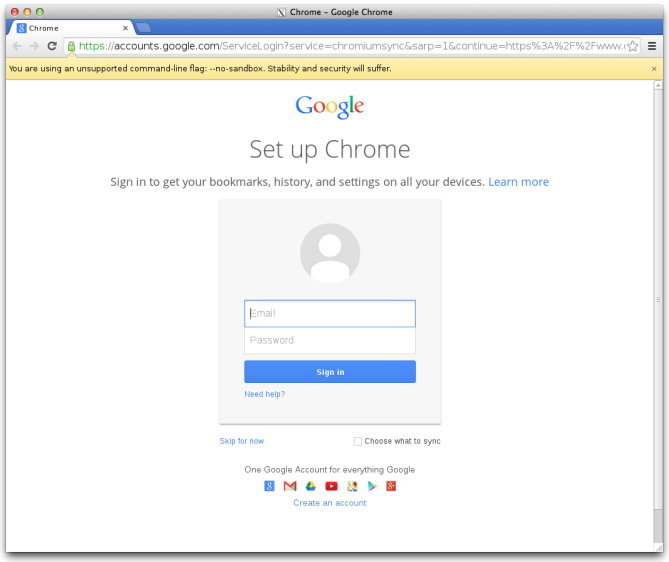
Quoi :

- Les évangélistes Docker font quasiment tout tourner en conteneur
- Permet de limiter la contagion virale depuis un logiciel
- Moyen de tester des applications en gardant un système clean
- Permet de garder propre l'OS sous jacent :
  - Latex, environnement lourd, nombreuses dépendances
- Exemple [21] : Latex, Irssi, Mutt, Spotify, Skype

[21] https://blog.jessfraz.com/post/docker-containers-on-the-desktop/

Exemple: Sandbox pour navigateurs

```
# docker run -t -i -p 22 magglass1/docker-browser-over-ssh
IP address:      172.17.0.4
Password: N24DjBM86gPubuEE
Firefox:  ssh -X webuser@172.17.0.4 firefox
Google Chrome:  ssh -X webuser@172.17.0.4 google-chrome --no-sandbox
```



Ou via module VNC dans Chrome :

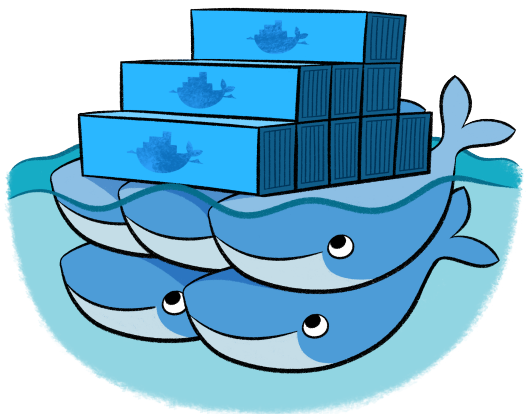
`https://hub.docker.com/r/siomiz/chrome/`

Plan

Docker Compose

Installation et ligne de commande

- Lancer une stack de conteneur via 1 fichier : docker-compose.yml
  $\Rightarrow$ "meta-Dockerfile"
- "Compose" gère des groupes et les liens
- Configuration en YAML (Simplicité)



Installation et ligne de commande

- Installation simple
- docker-compose run — up — stop — rm — build

Docker Compose: Syntaxe

```
version: "2"

services:
  service_name_1:
    image: <image_name_1>
    ports:
      - "80:80"
    volumes:
      - ./service.conf:/etc/service/service.conf
    command: --verbose

  whoami:
    image: <image_name_1>
    build: ./path/to/dockerfile
    depends_on:
      - service_name_1
    environment:
      - MY_ENV_VAR=environement
```

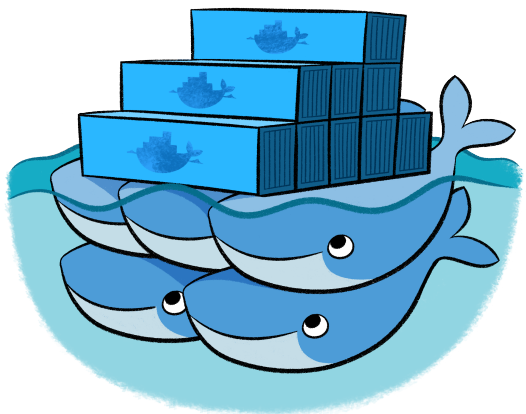
Docker Compose

docker run -d --name db mysql

```
db:
  image: mysql
  environment:
    MYSQL_ROOT_PASSWORD: password
```

docker run -d --name phpfpm --link db:mysql jprjr/php-fpm

```
phpfpm:
  image: jprjr/php-fpm
  volumes:
    - .:/srv/http
    - timezone.ini:/etc/php/conf.d/timezone.ini
  links:
    - db:mysql
```



docker run -d --link phpfpm:phpfpm -p 80:80 my-nginx

```
nginx:
  build: .
  links:
    - phpfpm:phpfpm
  ports:
    - 80:80
```

Exercice : Docker Compose

- Créer un fichier docker-compose.yml qui:
  - Créé un conteneur avec une base de données mysql https://hub.docker.com/_/mysql/
  - Créé un conteneur avec phpmyadmin et le lie à la base de données https://hub.docker.com/r/phpmyadmin/phpmyadmin/
- Créer une nouvelle base dans mysql avec phpmyadmin:
- Vérifiez que celle-ci a été créée dans le conteneur:
  - commande: mysql -u root -p
  - mysql: show databases;

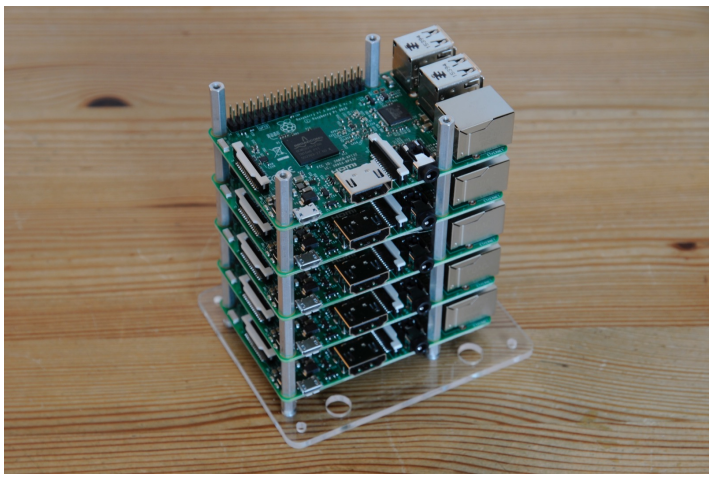
Exercice : Docker Compose

```
version: "2"

services:
  my-mysql-server:
    image: mysql
    environment:
      - MYSQL_ROOT_PASSWORD=root

  my-phpmyadmin-server:
    image: phpmyadmin/phpmyadmin
    depends_on:
      - my-mysql-server
    environment:
      - MYSQL_ROOT_PASSWORD=root
      - PMA_HOST=my-mysql-server
    ports:
      - "8080:80"
```

Plan

Docker Swarm[22]

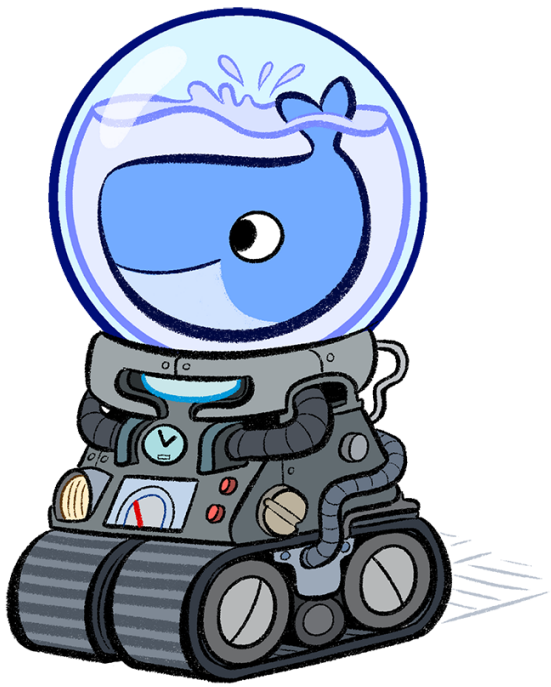
- Docker en multihost
  - Possiblement windows/linux
  - Faire passer des services à l'échelle
  - Tagger des machines: lieux (France, US ...), type (production, dev ...)
- 2 versions : docker swarm et docker-swarm (ancien)
- docker swarm (depuis 1.12) :
  docker swarm init – join – ps



[22] https://blog.hypriot.com/, https://creativecommons.org/licenses/by-nc/4.0/

Docker Machine[23]

- Permet d'utilise docker sur d'anciennes version de Windows/Mac
- Provisionne et configure Docker sur un serveur distant
  - Fonctionne avec la plupart des cloud providers
    - AWS / GCE / Azure / DO / IBM
  - Fonctionne aussi avec des technologies standards
    - Hyper-V (Windows) / HyperKit (MacOS)
    - OpenStack / vCenter



[23] Remplace boot2docker même si celui est toujours disponible

Docker Machine[24]

Utilisation de docker swarm en local (avec docker-machine):

```
docker-machine create -d virtualbox manager1
docker-machine ssh manager1
docker swarm init --advertise-addr=...
docker-machine create -d virtualbox slave1
docker-machine ssh slave1
docker swarm join ...
docker-machine create -d virtualbox slave2
docker-machine ssh slave2
docker swarm join ...
```

Depuis le manager:

```
docker node ls
docker service create --name whoami --replicas 1 \
  -p 80:8000 jwilder/whoami
docker service ls
```

[24] http://www-public.tem-tsp.eu/~berger_o/docker/install-docker-machine-virtualbox.html

Le document :

- Cette présentation a été faite avec des outils opensource et libre dans le but de présenter des outils de construction d'applications eux-aussi opensource.
- N'hésitez pas à nous envoyer vos remarques ou corrections à: alexandre.ancel sur ihu-strasbourg.eu ou johan.moreau sur gmail.com
-