Atelier Docker

Yoan Blanc yoan@dosimple.ch

mardi 16 août 2016, Saint-Imier

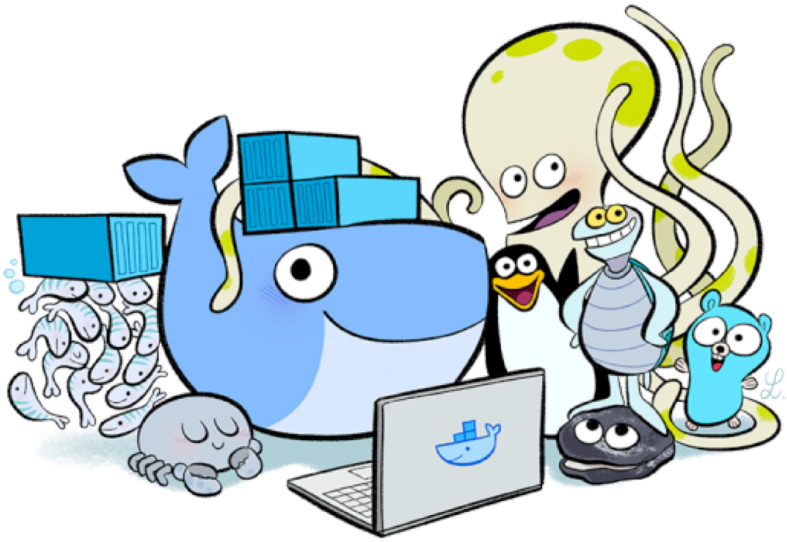

Fig. 1 :

Docker est un système d'orchestration de conteneurs, sous GNU/Linux propulsé par le langage de programmation Go.

Quelles sont vos attentes ? Qu'allez-vous découvrir ?

Préparatifs

```
$ uname -r
4.4+
```

```
$ docker version
Docker version 1.10+

$ docker-compose -v
docker-compose version 1.6+
```

Installation : Windows 10, macOS, GNU/Linux.

Objectifs

1. Se familiariser avec un *container* Linux.
2. Comprendre les enjeux de sécurité.
3. Créer une application PHP dans un conteneur.
4. Scalabilité horizontale de notre application.

Plongeons

```
docker run --interactive \
           --tty \
           --hostname demo \
           --name demo \
           alpine :3.4 \
           /bin/sh

/ # source /etc/profile
demo :/#
```

Alpine Linux est basé sur Busybox et musl libc dans le but de fonctionner en RAM.

Quiz 1

Que manque-t-il ?

```
demo :/# ls -l /
```

---

indice

```
demo :/# uname -r
4.7.0-1-ARCH

(Ctrl-p Ctrl-q)

$ uname -a
4.7.0-1-ARCH
```

Le principe de la virtualisation d'OS est de partager le noyau.

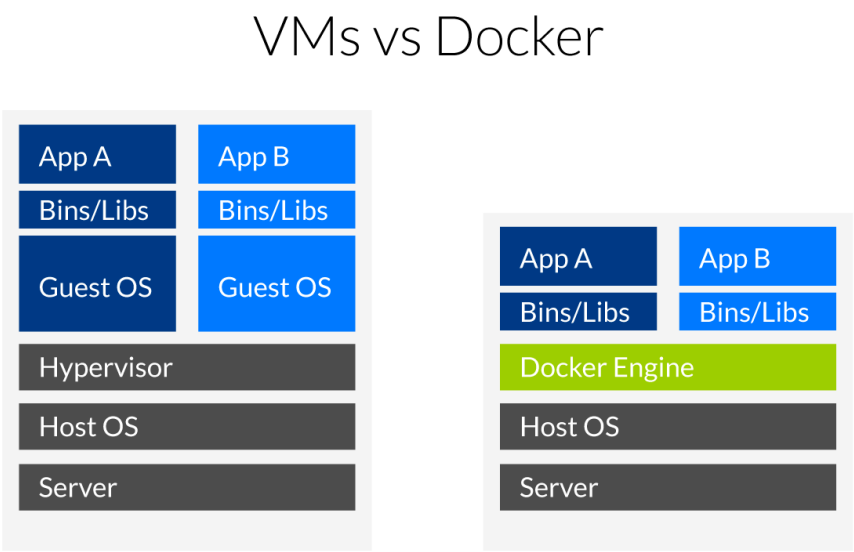

FIG. 2 :

Contrairement à la virtualisation ou para-virtualisation, il n'y a pas de système d'exploitation invité dans un conteneur.

Histoire

- 1972 IBM VM/370
- 1979 chroot
- 1999 FreeBSD jails
- 2000 Virtuozzo (OpenVZ)

- 2001 Linux VServer
- 2004 Solaris Zones
- 2008 LXC
- 2011 dotCloud (Docker)
- 2014 CoreOS Rocket (rkt)
- 2015 LXD (Canonical)
- 2015 Open Container Initiative

Virtualisation date des années 70, puis vient le terme *jail* (et du coup, *jailbreak*), pour enfin voir le mot *container* apparaitre avec LXC.

Les *namespaces* et *cgroups* ont permis la naissance de LXC qui a engendré docker, rocket et consœurs.

Espace de noms

```
demo :/# ps -e
```

```
$ ps -e
```

Vos processes sont isolés des autres.

Quiz 2

Quel est le risque d'un système sans `init` (comme `systemd`, `upstart`, `runit`) ?

Qu est-ce qui est étonnant ici ? Pas de `init`. Si votre processus forke, il y a un risque de rencontrer des problème de zombies…

Control Groups

```
docker stats
```

```
docker update --memory 2GB demo
```

Quota sur les ressources telles que mémoire et CPU de manière restreinte (par rapport à `ulimit`).

Initié par Google qui utilise des conteneurs depuis « toujours », voir Borg

Hadoop ou systemd utilisent notamment les *cgroups*.

Copy on Write (CoW)

```
demo :/# echo "Hello World!" > hello.txt
```

```
docker diff ...
```

Union File System via overlayfs, aufs, zfs, etc. (voir : Select a storage driver)

Chaque conteneur possède une petite couche modifiable par dessus les couches existantes. Ceci permet de démarrer quasiment instantanément des instances.

Réseau virtuel

```
demo :/# ip addr
```

```
ip addr
```

```
docker network list
```

```
brctl show
```

Arrêt, redémarrage

```
/ # exit
docker ps -a
```

```
docker start demo
```

```
docker attach demo
/ # more hello.txt
```

```
docker stop demo
```

Ctrl-p + Ctrl-q pour se détacher.

Exportation

```
docker export -o demo.tar demo
```

```
tar tf demo.tar
```

Une des additions de Docker sur son parent, LXC.

Et sa contrepartie, `docker import`.

Sauvegarde, distribution

```
docker commit demo hearc/demo
```

```
docker images
```

`docker push` permet d'envoyer l'image sur Docker Hub (ou un autre *registry*).

Sécurité

En terme de surface d'attaque, la virtualisation d'OS est plus risquée que de la paravirtualisation ou de la virtualisation pure (type 1 ou 2). Par exemple, Docker est exécuté en tant que `root`, donc réussir à en sortir est la porte du paradis.

Capabilities

```
/ # hostname
demo

/ # hostname hello

hostname : sethostname :
 Operation not permitted
```

Le but est de remplacer le trop permissif `setuid` et ainsi d'éviter `root`.

```
--cap-add=SYS_ADMIN
```

```
docker run --rm \
           -it \
           --cap-add=SYS_ADMIN \
           alpine :3.4 \
           /bin/sh
```

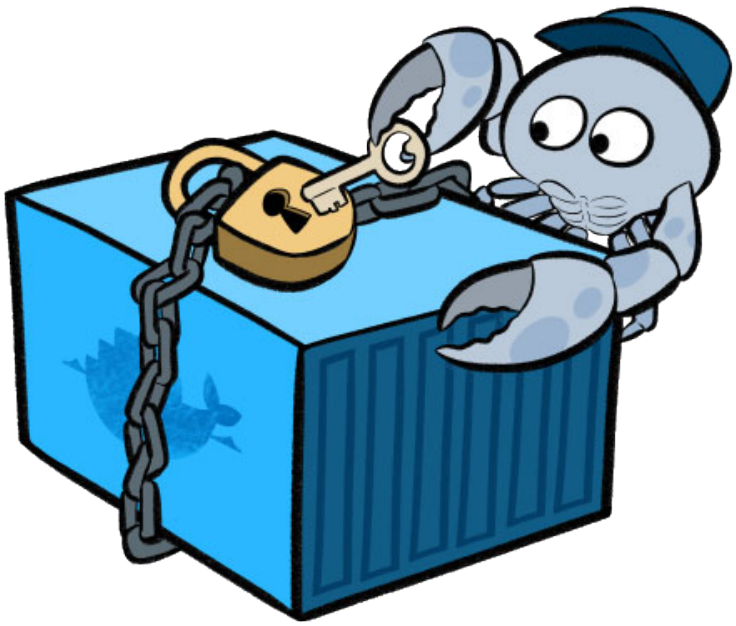

Fig. 3 :

```
/ # hostname hello
```

`SYS_TIME` permettrait d'avoir `ntp`, par exemple.

SECure COMPuting

Masquage de certains *sys calls*.

```
demo :/# apk add --no-cache keyutils
```

```
demo :/# keyctl session
```

```
keyctl_join_session_keyring :
 Operation not permitted
```

Par défaut 44 appels bloqués sur 300+.

Chevauchement avec les *capabilities*.

Merci Google Chrome !

`--pids-limit n`

```
docker run -it --rm \
           --pids-limit 20 \
           alpine :3.4 \
           /bin/sh
```

```
/ # b() { b | b& }; b
```

Avec un `sleep 1 && ( b | b& )` et `docker stats`, c'est intéressant

Disponible depuis Docker 1.11 (requiert Linux 4.3).

Quiz 3

Quel est le problème ici ?

```
docker run -it --rm \
           --volume /etc :/etc \
           alpine :3.4 \
           /bin/sh
```

```
/ # more /etc/passwd
```

Espace utilisateur

Qui peut accéder à votre système de fichiers, devient `root` par définition.

Solution : `--userns-remap=default`.

`CONFIG_USER_NS`.

Disponible depuis Docker 1.10.

Plus de sécurité

- Docker Bench for Security
- AppArmor (SuSE, Ubuntu)
- SELinux (NSA, Red Hat, CentOS, etc.)

Inter-Container Communication

Par défaut, deux containers peuvent communiquer entre eux.

Source : Laurel

ICC (suite)

Désactiver globalement la communication entre conteneurs.

```
# /etc/systemd/system/docker.service

[Config]
ExecStart=/usr/bin/docker daemon --icc=false ...
```

Ceci est recommandé.

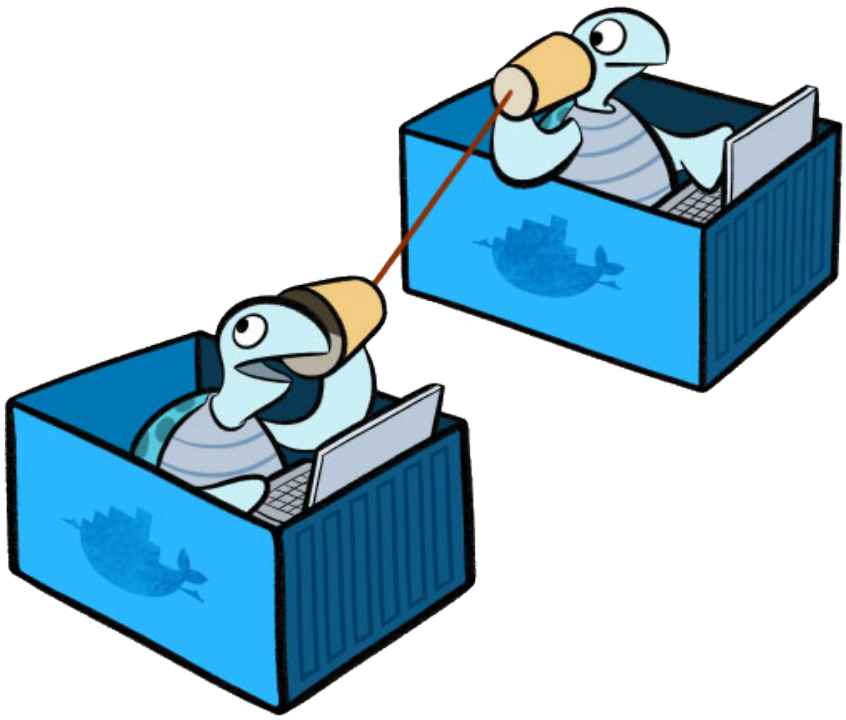

FIG. 4 :

Créer un réseau

```
$ docker network create mynet

$ docker network connect mynet demo

$ docker run -it \
             --net=mynet \
             alpine :3.4 \
             /bin/sh
```

Par défaut, c'est *bridge* qui est choisi.

Créer un volume

```
docker volume create --name myvolume

docker run -it --rm \
           --volume myvolume :/data \
           alpine :3.4 /bin/sh

docker run -it --rm \
           -v myvolume :/data :ro \
           alpine :3.4 /bin/sh
```

E.g. une base de données, un site web (nginx vs php-fpm), etc.

Plugins : GlusterFS, GCE, Contiv (Ceph), etc.

Tour d'horizon

CoreOS rkt

```
$ rkt run --insecure-options=image \
          docker ://alpine :3.4 \
          --exec=/bin/sh \
          --interactive
```

Une image docker peut-être exécutée non plus via `runc` mais `rkt`, une autre spécification (appc vs oci).

Utilisé par Blablacar, notamment.

Canonical LXD (LXC)

```
$ lxc launch images :alpine/3.4/amd64 alpine
$ lxc exec alpine -- /bin/sh
```

Historiquement Docker et CoreOS utilisaient LXC. Remplacé par `runC` et `rkt` depuis. LXD est la réponse de Canonical. Support notable de CRIU pour des migrations en live.

Il y a également systemd-nspawn

Le *"Cloud"*

- Amazon (Xen)
- Citrix Cloud.com (Xen)
- Google Compute Engine (KVM)
- Microsoft Azure (Hyper-V)
- Digital Ocean (KVM)
- Samsung Joyent (SmartOS)
- Verizon (VMware)
- Heroku (LXC)
- Infomaniak (KVM)
- HE-Arc (OpenVZ)

SmartOS (Joyent, Samsung) = illumos (OpenSolaris) + KVM.

---

mi-temps

Problèmes actuels

1. Systèmes découplés
2. Itérations rapides
3. Environnement hétérogène
4. Montée en charge horizontale

Créer un système découplé

```
docker run -d \
           --publish-all \
           --name php \
           php :7.0-apache
```

```
docker cp index.php php :/var/www/html
```

Téléchargement et exécution d'une image PHP préconstruite.

Les ports sont exportés de manière aléatoire.

Ajout de Redis

```
docker exec -it php bash
```

```
# pecl install redis
# docker-php-ext-enable redis
# apache2ctl restart
```

Rafraichir le `phpinfo` va montrer que Redis est à présent là.

`Dockerfile`

```
FROM php :7.0-apache

RUN pecl install redis \
 && docker-php-ext-enable redis

COPY index.php /var/www/html
```

Construction d'une image.

```
docker build -t myapp .
```

Notre nouvelle image ajoute deux couches à l'image de base.

---

index.php

```php
<?php
$redis = new Redis();
$redis->connect('redis');
$redis->incr('test');

echo $redis->get('test');
```

Mise-à-jour de notre image.

```
docker build -t myapp .
```

Notre ancienne image a disparu de notre liste. Il est néanmoins possible de la retagguer à volonté.

Redis

```
docker run -d \
           --name redis \
           redis :3.2-alpine

docker run -dP \
           --link redis \
           --name php \
           myapp
```

Malgré `--icc=false`, les ports ouverts de `redis` ont été ouverts dans `php` et seulement ceux-là.

```
/ # ping -c 3 redis
/ # apk --no-cache add nmap
/ # nmap -p1-6400 redis
```

Docker-compose

```yaml
version : '2'
services :
  php :
    image : myapp
    links :
      - redis
    ports :
```

```
      - 80 :80
  redis :
    image : redis :3.2-alpine
```

Exécution

```
docker-compose up
```

`-d` permet de lancer docker-compose en arrière plan.

(re)construire

```
services :
  php :
    image : myapp
    build : .
```

Mise à jour.

```
docker-compose build
docker-compose up frontend
```

Préparez vos cartes !

Scalabilité horizontale

```
frontend :
  image : dockercloud/haproxy
  links :
    - php
  ports :
    - 80 :80
  volumes :
    - /var/run/docker.sock :/var/run/docker.sock
```

Go big !

```
docker-compose scale php=2
```

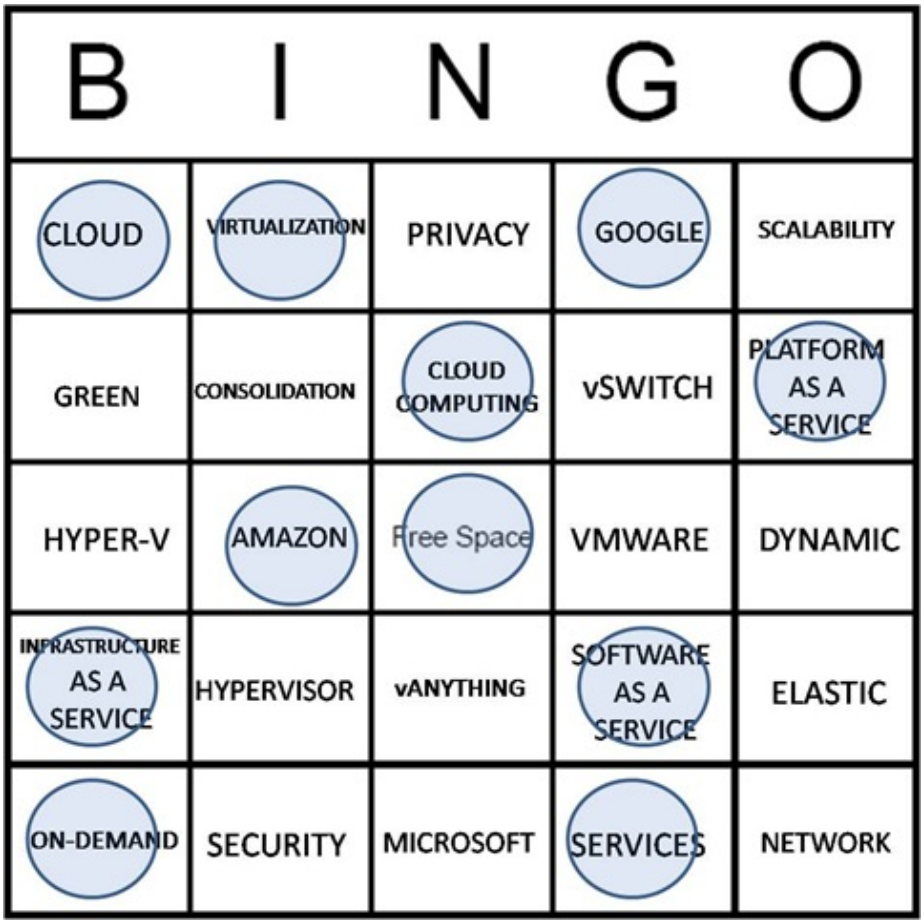
B I N G O
CLOUD
VIRTUALIZATION
PRIVACY
GOOGLE
SCALABILITY
GREEN
CONSOLIDATION
CLOUD COMPUTING
vSWITCH
PLATFORM AS A SERVICE
HYPER-V
AMAZON
Free Space
VMWARE
DYNAMIC
INFRASTRUCTURE AS A SERVICE
HYPERVISOR
vANYTHING
SOFTWARE AS A SERVICE
ELASTIC
ON-DEMAND
SECURITY
MICROSOFT
SERVICES
NETWORK

FIG. 5 :

Going bigger !

- Docker Swarm (1.12)
- Kubernetes (Google)
- Amazon EC2 Container Service
- Apache Mesos (Berkeley)
- CoreOS (Google Ventures)

Futur

- Docker + CUDA
- containerd
- Unikernel
- Triton Cloud

Lectures

- Docker Curriculum, Prakhar Srivastav
- Slideshare, Jérôme Petazzoni
- Understanding and Hardening Linux Containers
- Run containers on bare metal already !
- Life and death of a container, Luis Herrera Benítez
- Docker for PHP Developers, Chris Tankersley

End