Intégration continue

Groupe CNRS Calcul / Johan.Moreau - at - gmail.com|ircad.fr

Johan Moreau

IRCAD/IHU

29 mai 2017

Plan

Situation

La situation - chez tout le monde ?

Les difficultés :

- Augmentations des besoins numériques (donc ↗ des services)
- Fréquence de mises à disposition ultra rapide
- Inteconnexions plus fortes entre les services
- Difficultés de maintenir la documentation d'exploitation
- Multiples environnements (dev, tests, production, ...)
- Cas complexes des clusters, de la HA, des workflow de dev, ...

Réponses partielles :

- Agilité : meilleure production
- DevOps : meilleure exploitation
- Nouveaux outils liés à ces tendances

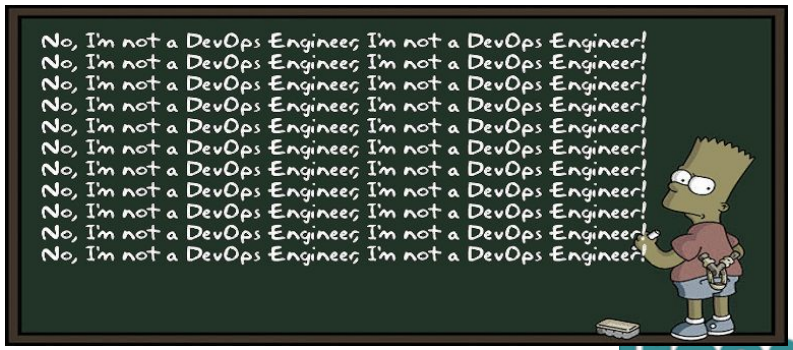


Image from https://3ovyg21t17l11k49tk1oma21-wpengine.netdna-ssl.com/wp-content/uploads/2014/03/bart-simpson-generator-1.jpg

Méthodes

Quelles réponses des communautés ?

Les principes de l'agilité :

- Cycle en V trop long, cycle rapide [2]
- Livraisons très régulières pour retour clients
- Proximité du client pour une meilleure communication
- Mode équipe, le code est à tout le monde, gestion des absences
- Beaucoup d'autres principes :
  - gestion du code, construction simple, ...
- Méthodologies : Scrum, XP, ...
- Outils : DVCS, intégration continue, ...
- *Simple sur le papier, long à mettre en place ...*

[2] http://eureka.sbs.ox.ac.uk/897/1/WP_2011_08_15.pdf

DevOps

Les principes du DevOps :

- Les exploitants ont des conditons différentes des développeurs, les faire travailler ensemble
- Mises en exploitation très régulières et rapides [3] [4]
- Très présent chez les leaders :
  - Amazon - toutes les 11,6s [5],
  - Etsy - 30fois/j [6],
  - ...
- Méthodologies : Kanban, ...
- Outils : VM, conteneurs, gestion de configuration, ...
- *Etape complexe : bon alignement Devs et Ops, + agilité côté Devs !*

[3] https://puppetlabs.com/wp-content/uploads/2013/03/2013-state-of-devops-report.pdf
[4] http://pages.zeroturnaround.com/RebelLabs-AllReportLanders_DevopsProductivityReport.html
[5] http://assets.en.oreilly.com/1/event/60/Velocity%20Culture%20Presentation.pdf
[6] https://codeascraft.com/2012/03/13/making-it-virtually-easy-to-deploy-on-day-one/

Contextes complémentaires

Microservices[7] **:**

- Minimisation d'une application afin de gagner en maintenabilité
- Définition d'un service la plus claire possible
- Principe ancien (Minix vs Linux[8]) mais plus de sens avec REST
- *Impose un design/réflexion très particulier*

Infrastructure définie par le logiciel (SDI ou Infra-as-Code)[9] **:**

- Pour gagner en productivité, automatisation d'infrastructures complexes (service, stockage, réseau, sécurité, ...)
- Outils et API de plus en plus présentes
- TDI : Test-Driven-Infrastructure (suite du TDD)
- *Projet transversal imposant des connaissances larges*

[7] http://en.wikipedia.org/wiki/Microservices
[8] http://en.wikipedia.org/wiki/Tanenbaum-Torvalds_debate
[9] http://en.wikipedia.org/wiki/Software_Defined_Infrastructure

Gestion de sources et artefacts

DVCS, un élément clé dans la réflexion :

- Manipulation de fichiers 'texte', donc gestion réaliste possible
- Fichiers légers pouvant utiliser les fonctions des DVCS (branches, staging, ..)
- Bonnes pratiques compatibles (tout en texte, tout est suivi, ...)
- Intégration continue dans les 2 mondes (devops) et entre les outils

Des artéfacts intéressants :

- Versions journalières("nightly build")
- Déploiements automatiques/continus
- Génération automatique de l'historique (release notes), etc ...

Les leaders :

- git, mercurial, subversion, ...

Les forges sont bien pratiques

Rien de bien nouveau :

- En mode public :
  - Sourceforge, Savannah, ...
- En mode hébergé :
  - Trac, Redmine, ...
- Hébergement des dépôts, de quelques artefacts, forums, ...

Et pourtant, une nouvelle vague :

- En mode public :
  - GitHub, GitLab, ...
- En mode hébergé :
  - GitLab, Rhodecode, ...
- Réseau social, discussion autours des problèmes, PR/MR, ...

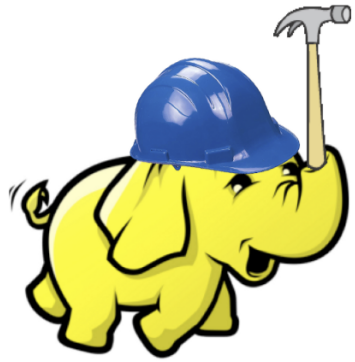
Intégration continue [13]

Définition :

- Ensemble de pratiques vérifiant de manière automatique et régulière que chaque modification de code n'engendre pas de régression

Pour vérifier, on utilise la phase de "construction" :

- Compiler le code (standard, rapide, ...),
- Lancer les tests unitaires,
- Analyser le code, Générer des rapports, ...,
- Générer un exécutable ou une archive



Les leaders :

- Jenkins, Travis-ci, Gitlab-ci, ...

Plus un bug est découvert tôt, moins d'impact aura sa correction [11]

Maintenir un build opérationnel n'est pas facile ! [12]

Image from http://vichargrave.com/wp-content/uploads/2013/02/Hadoop-Development.png

[11] http://martinfowler.com/articles/continuousIntegration.html

[12] http://static.googleusercontent.com/media/research.google.com/de//pubs/archive/42184.pdf

Comment améliorer cela ?

La reproductibilité est le nerf de la guerre

Quelles difficultés avec l'intégration continue :

- Se battre pour faire passer le 1er job est motivant, après ...
- Dans un baremetal ou une VM, il y a des évolutions possibles, qui doit faire le boulot ?
- Comment tester l'évolution de l'esclave jenkins ?
- Gérer la combinatoire croissante de la matrice des jobs !

Principes de l'infrastruture définie par le code et de l'infrastructure reproductible - même approche que la recherche reproductible [14]

Les conteneurs applicatifs (type Docker) peuvent aider pour cela

[14] http://www.sciencemag.org/content/334/6060/1226

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 ![15]

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

[15] http://fr.slideshare.net/BodenRussell/kvm-and-docker-lxc-benchmarking-with-openstack
Image from https://msopentech.com/wp-content/uploads/DockerUbuntu.png

Différences entre VM et conteneur

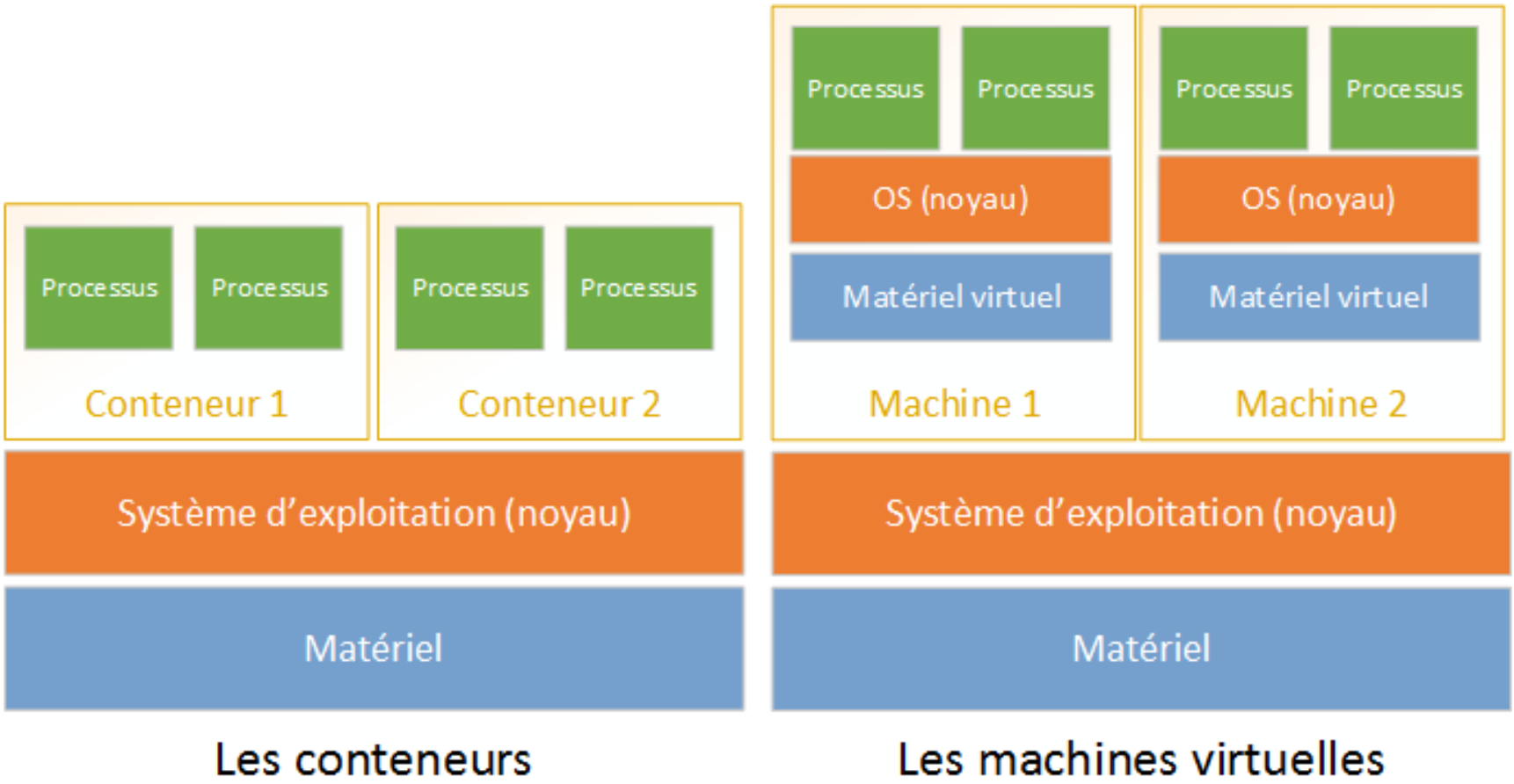
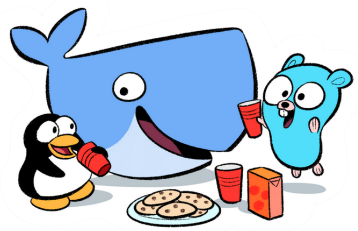


Image from `http://uploads.lightcode.fr/articles/33-conteneurs-linux/conteneurs-vs-vms.png`

Liaison avec le noyau Linux[19] 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 : gérer la limitation de ressource
- AUFS/BTRFS : FS/couche, mode union, copy on write



En gros, les éléments noyaux sont en place depuis Linux 3.8

Mais natif depuis 2016 sur OSX et Windows

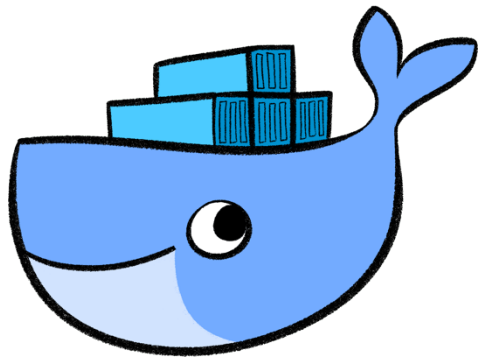
[19] Image from https://cdn-images-1.medium.com/max/1600/1*TXashznqPJG_9iNPar9a7w.png
https://github.com/sdouche/presentation-docker/

Docker, la petite histoire

De dotcloud à docker

La petite histoire :

- DotCloud [20] équivalent FR à Heroku (PAAS)
- Construit sur LXC et AUFS + dev. kernel linux
- Pas mal de soucis sur la gestion des conteneurs
- Développement interne d'un démon en python

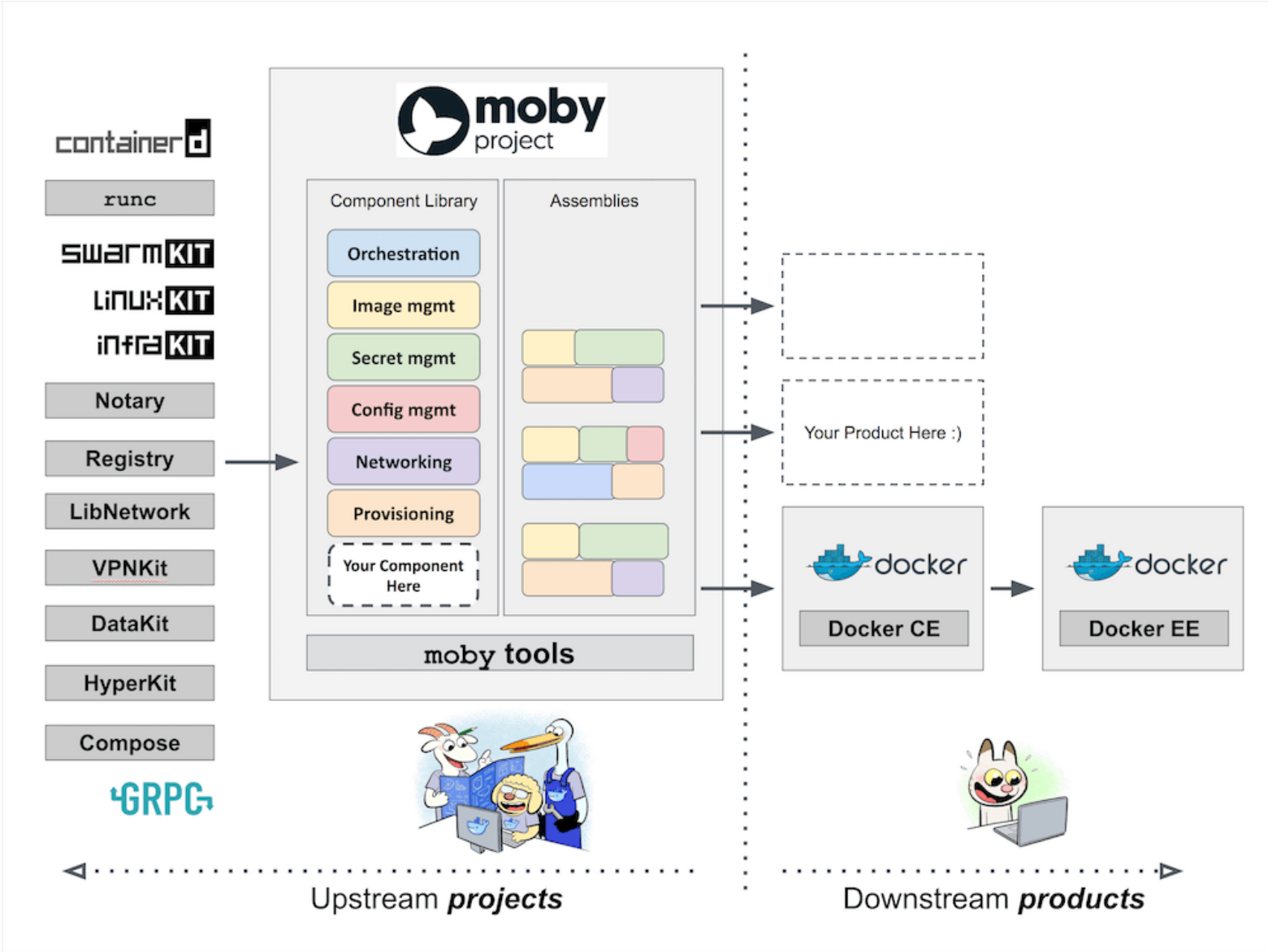


Docker - vers la simplification :

- Projet jeune en Golang [22] : 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 [23]

[20] http://fr.slideshare.net/jpetazzo/introduction-to-docker-december-2014-tour-de-france-bordeaux-special-e
Image from https://www.docker.com/sites/default/files/Whale%20Logo332%402x_5.png
[22] http://linuxfr.org/news/la-folie-docker
[23] https://www.flockport.com/lxc-vs-docker/

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



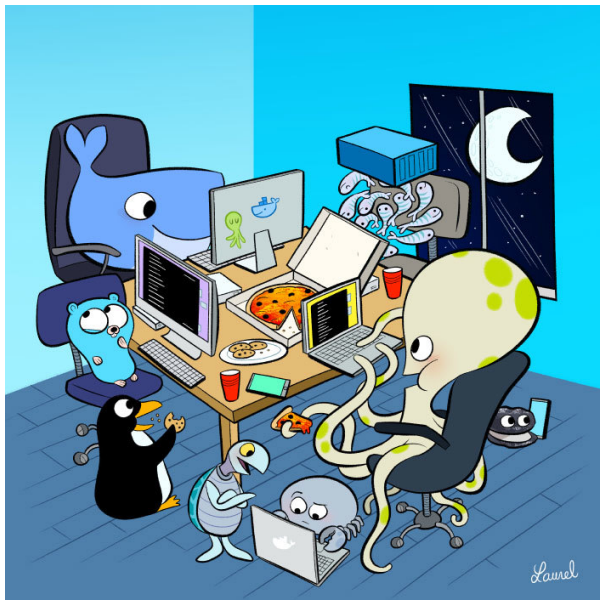
Docker

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 : élement manipulable

Analogie avec le développement objet :

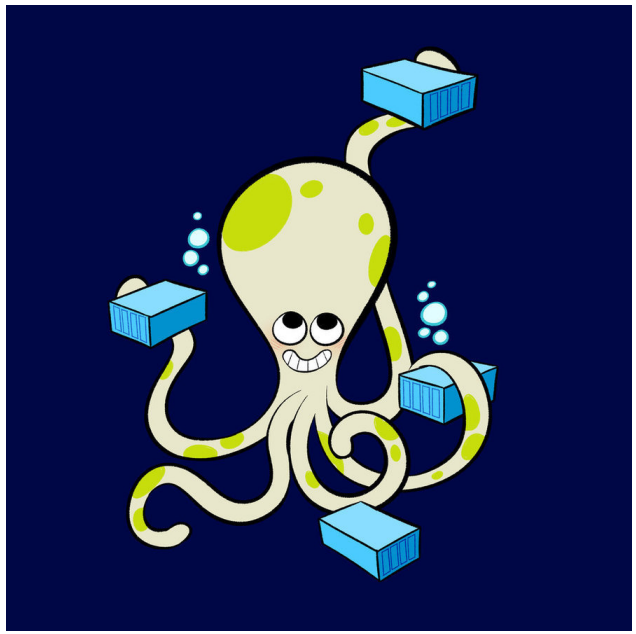
- Images équivalentes aux classes
- Les couches sont équivalentes à l'héritage
- Les conteneurs sont des instances



Docker

Les points forts :

- Installation simple (Linux, OSX, Windows) [26]
- 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 [27], Orchestration, hébergement cloud, intégration continue, OS, ...



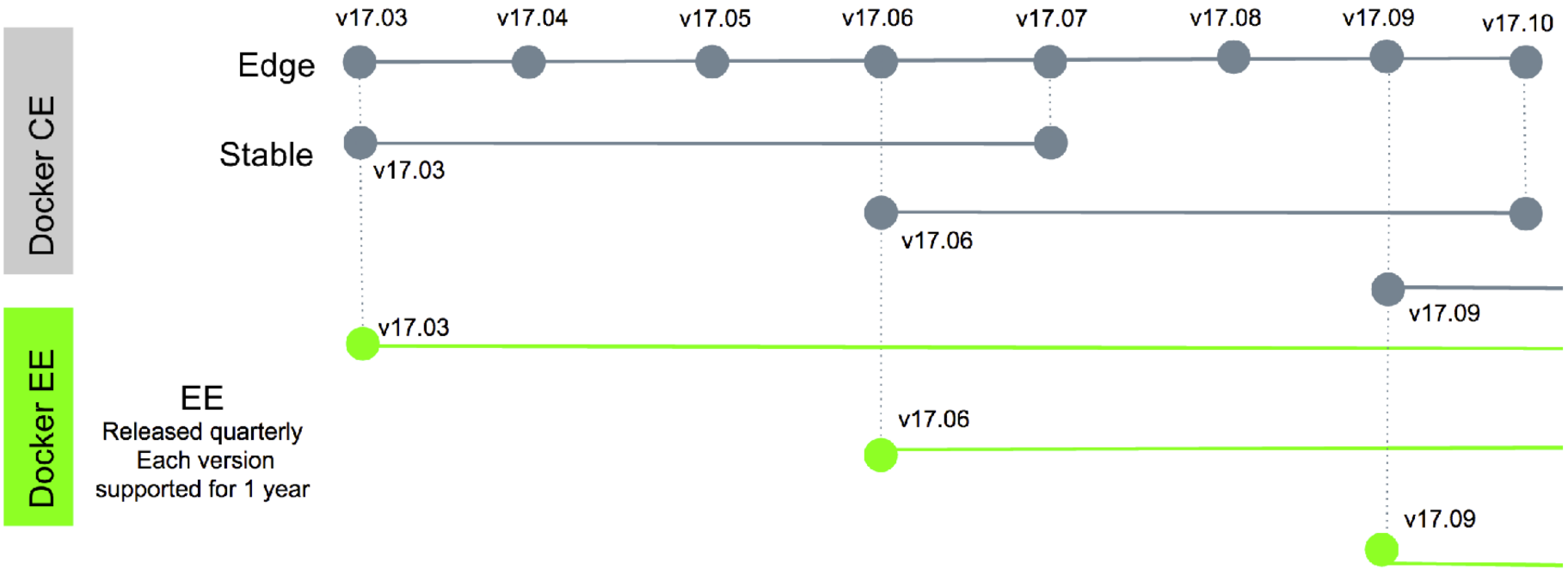
[26] http://docs.docker.com/installation/

[27] http://linuxfr.org/news/logiciels-pour-survivre-avec-docker
Image from http://pre12.deviantart.net/9476/th/pre/i/2015/253/c/7/octopus_docker_bloglaurel_by_bloglaurel-d99468m.jpg

Evidement, il faut vivre : CE vs EE [30]

[30] Image from `https://i1.wp.com/blog.docker.com/wp-content/uploads/ee.png`
RBAC LDAP/AD, gestion des images avancées, scan d'images, etc ...

Version en MM.YY



Les commandes de bases

Docker 101 [31]

A l'ancienne :

```
$ docker 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

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

Depuis les dernières version de Docker :

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

[31] 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 -> 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 -> Ainsi que de la communication entre l'extérieur et le container
- Capture des messages entrées-sorties

1 seul processus

- Phylosophiquement, 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 top mycontainer
docker inspect --format {{.State.Pid}} 'docker ps -q'
```

Docker in the shell [32] [33] [34]

Les principales commandes :

Code Listing 1 – les commandes docker utiles

```
sudo /usr/bin/docker -d & # run the daemon
sudo docker search ubuntu # give ubuntu images from public index ( official /trusted )
sudo docker pull stackbrew/ubuntu # pull latest stackbrew/ubuntu images
sudo docker history stackbrew/ubuntu # view history for this images
sudo docker images # show local images
sudo docker run -i -t stackbrew/ubuntu /bin/bash # run this images / create container
sudo docker run -t -i -link redis:db -name webapp ubuntu bash # link 2 containers
sudo docker ps # show active containers (-a to show all containers)
sudo docker logs myUbuntu
sudo docker attach myUbuntu # retake the hand on the container
sudo docker run -d -p 8888:80 ubuntu # export 8888 on master
sudo docker stop # SIGTERM suivi d'un SIGKILL
sudo docker kill # SIGKILL directement
```

[32] http://ippon.developpez.com/tutoriels/virtualisation/docker-presentation-part-1/
[33] http://ippon.developpez.com/tutoriels/virtualisation/docker-presentation-part-2/
[34] http://sametmax.com/le-deploiement-par-conteneurs-avec-docker/

Les images

Création d'images Docker [35]

Différente méthode en ligne de commande :

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

Description et construction :

Code Listing 2 – Dockerfile

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

Code Listing 3 – Exécution d'un build Docker

```
$ docker build -q -t readme .
Sending build context to Docker daemon 3.584 kB
Sending build context to Docker daemon
Step 0 : FROM debian:wheezy
 ---> e8d37d9e3476
Step 1 : ADD README.md /tmp/
 ---> 09eabce38f39
Removing intermediate container 3e44a3b6eabe
 Successfully built 09eabce38f39
```

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

Couches d'une image Docker

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

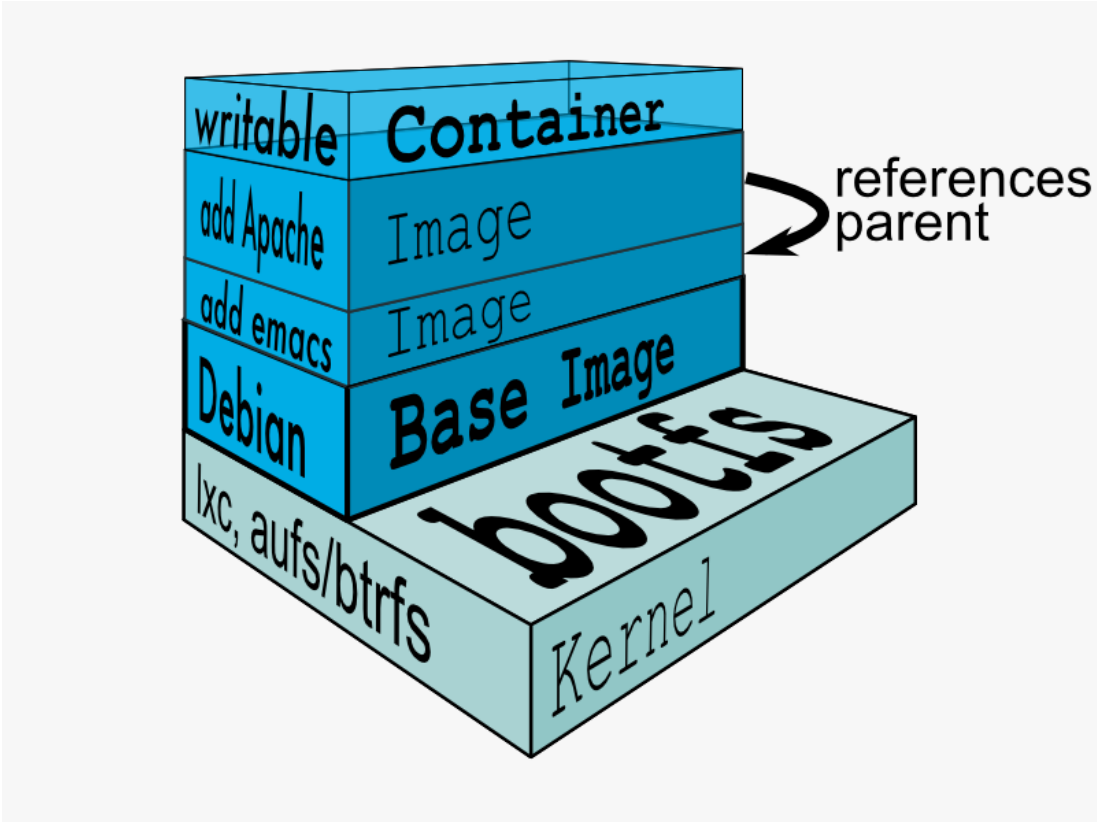


Image from https://raw.githubusercontent.com/radiomix/poc-docker-jenkins/master/img/docker-filesystems-multilayer.png

Instructions (DSL) du Dockerfile

Les instructions sont peu nombreuses : FROM, RUN, CMD, LABEL, MAINTAINER, EXPOSE, ENV, ADD, COPY, ENTRYPOINT[37], VOLUME, USER, WORKDIR, ARG, ONBUILD, STOPSIGNAL, HEALTHCHECK, SHELL

- 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 intermédiare (7yyyyyyyy),
- Le conteneur intermédiare (8xxxxxxxx) est supprimé, Le résultat, l'image intermédiaire, servira d'image de base pour l'étape suivante,
- etc..

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

Exercice : Créez un conteneur ping avec argument

En partant de l'image *alpine*
Créez un Dockefile appelant ping et prenant en paramêtre du conteneur le host

Exercice

Code Listing 4 – Dockefile pour ping

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

Multi-stage build [38] [39]

- Etre 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 point d'une image
- Cela est utile pour des artefacts "construits" dans dans des stages de l'image (ou pour réduire des installations trop lourdes)

Code Listing 5 – Dockerfile go multistage

```
FROM golang:1.7.3
COPY app.go .
RUN go build -o app app.go

FROM scratch
COPY --from=0 /go/app .
CMD ["./app"]
```

[38] https://stefanscherer.github.io/use-multi-stage-builds-for-smaller-windows-images

[39] Présent que sur les dernières versions, sinon voir Builder Pattern avec un shell script

La persistance des données

Volume

- Outrepasser le système de COW (copy-on-write) pour profiter des performances natives d'I/O disques,
- Outrepasser le système de COW pour ne pas intégrer les modifications de fichiers dans une couche :
  - Pour ne pas les commiter,
  - Pour persister ses données sur le diques,
- Pour partager des fichiers/Dossiers entre conteneurs,
- Pour partager des fichiers/dossiers entre hôte et conteneur

- Utilisation du -v (–volume)
- Utilisation du –volumes-from

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 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 :

```
docker network create ––driver overlay ––subnet 10.0.9.0/24 ––opt encrypted myApp1
docker run –d ––network myApp1 ––name mydb mysql
```

Autre stratégie les reverse-proxy : HAProxy, traefik, ...

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

Code Listing 6 – Lancer un conteneur Php avec un montage

```
#!/bin/bash
docker run -d -P --name my-apache-php-app \
    -v "$PWD":/var/www/html php:7.0-apache
```

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

Partez d'une image *alpine*

Récupérez `https://github.com/JohanMoreau/dev-koans/tree/master/Python/FlaskBasic`

Installez pip :

apk add –update py2-pip

Récupérez le nécessaire

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

Exercice

Code Listing 7 – Dockerfile pour Flask

```
# our base image
FROM alpine:latest
# Install python and pip
RUN apk add --update py2-pip
# install Python modules needed by the Python app
COPY src/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 src/app.py /usr/src/app/
COPY src/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"]
```

Code Listing 8 – Lancer un conteneur Flask

```
#!/bin/sh
docker container run -p 8888:5000 --name mywebapp webapppy
```

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 [40] (profil apparmor, ...), capabilities [41] (example : cap-drop ALL , cap-add CHOWN)
- Registry privée

[40] `http://training.play-with-docker.com/security-seccomp/`
[41] `http://training.play-with-docker.com/security-capabilities/`

Private Registry

- Stock 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, ...

Public vs private Registry

Le hub public :

- Dépôt public (push/pull gratuit)
- Dépôt d'images officielles, et d'images tiers,
- Dépôt privé (dépôt d'images non publiques payant)
- Collection de services supplémentaires :
  - Automated builds (lier des dépôts github/bitbucket pour lancer un build suite à un commit)

Exercice : Private Registry

Lancez une 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

Code 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
```

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

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[42], ...

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, ...

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

Exemple pour tout le monde

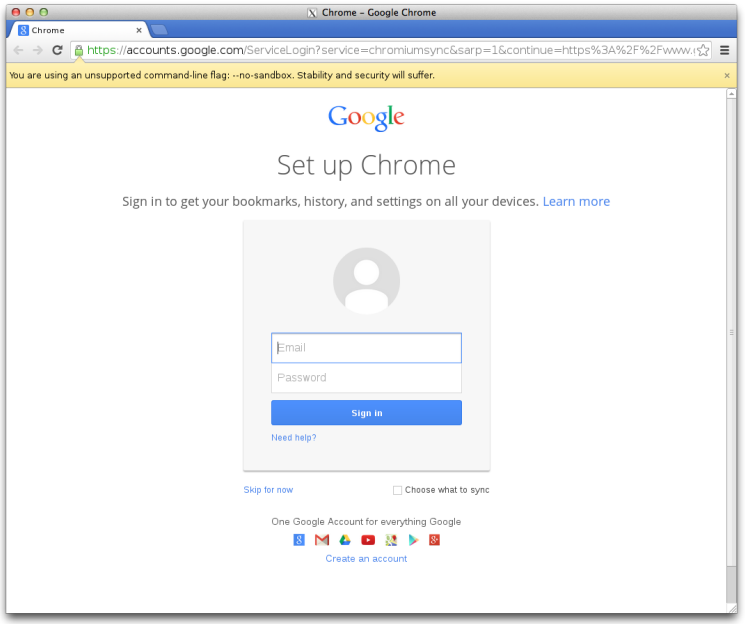
Quoi :

- 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 [43] : Latex, Irssi, Mutt, Spotify, Skype

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

Example : 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 Machine

Docker Machine [45]

- Provisionne et configure Docker sur un serveur distant
- Fonctionne avec la plus part des cloud providers
  - AWS / GCE / Azure / DO / IBM
- Fonctionne aussi avec des technologies standards
  - OpenStack / vCenter

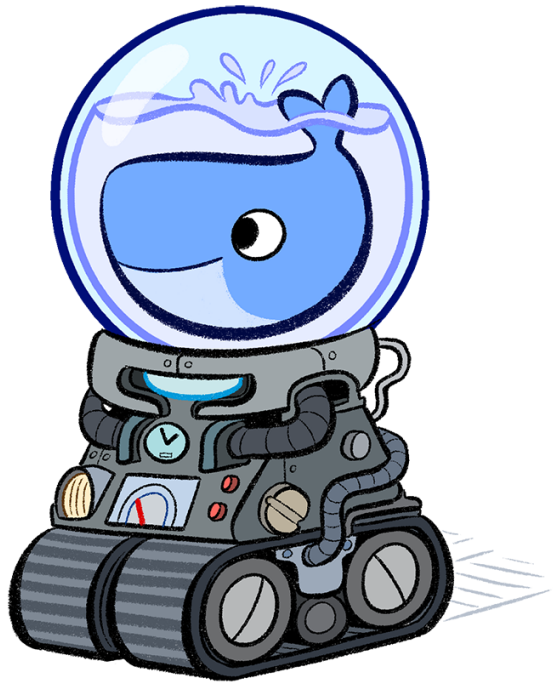


Image from `https://www.docker.com/sites/default/files/products/docker_machine.png`

[45] Remplace boot2docker même si celui est toujours disponible

Docker Machine [46]

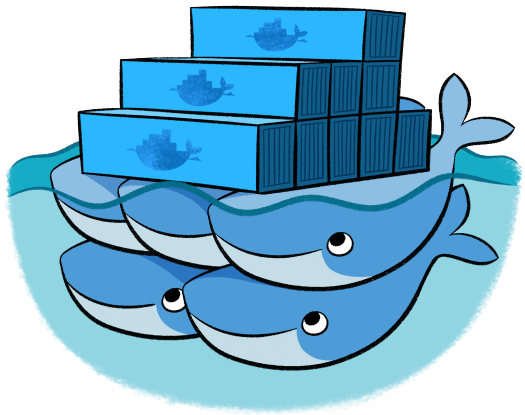
```
docker-machine create -d virtualbox manager1
docker-machine create -d virtualbox slave1
docker-machine ssh slave1
docker swarm join  ...
docker-machine create -d virtualbox slave2
docker-machine ssh slave2
docker swarm join  ...
```

Docker Compose

Docker Compose 1/2

Installation et ligne de commande

- Lancer une stack de conteneur via 1 fichier : docker-compose.yml
- "Compose" gère des groupes et les liens
- Simplicité du YAML



Installation et ligne de commande

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

Image from http://sdtimes.com/wp-content/uploads/2015/02/0226.sdt-docker-2.png

Docker Compose 2/2

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

Récupérez le docker-compose de *sameersbn/gitlab*
Lancez le compose (modifications probables)
Configurez votre mot de passe gitlab (Azerty123) en allant sur
http ://127.0.0.1 :10080
Ajoutez *jenkins :latest* avec un stockage permanent et port 10081
Vérifiez que gitlab voit jenkins et inversement

Exercice : Docker Compose

wget https://raw.githubusercontent.com/sameersbn/docker-gitlab/master/docker-compose.yml

Ajouter

```
jenkins :
  image: jenkins
  ports :
  - "10081:8080"
  - "50000:50000"
  links :
  - gitlab : gitlab
  volumes:
  - ./jenkins :/var/jenkins_home
```

```
docker exec -it ci_jenkins_1 /bin/bash
ping gitlab
docker exec -it ci_gitlab_1 /bin/bash
ping jenkins
```

Docker Swarm

Docker Swarm

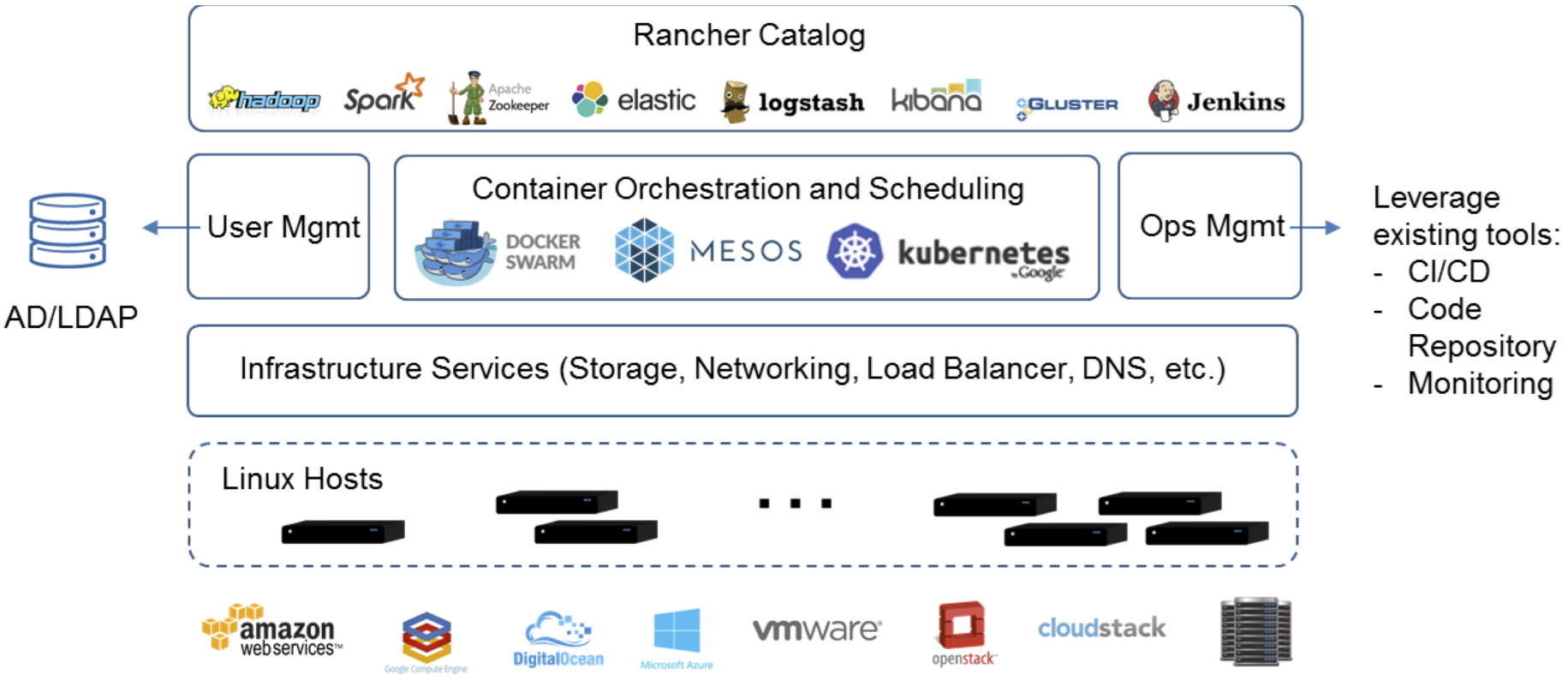
- Docker en multihost (possiblement windows/linux)
- 2 versions : docker swarm et docker-swarm (ancien)
  - docker-swarm : outil externe nécessisant l'utilisation d'un outil de découverte de services (consul, etcd, ...)
  - docker swarm : intégré depuis 1.12 :
    - docker swarm init
    - docker swarm join
    - docker swarm ps

Rancher

Rancher - Basic

- Developpé par RancherLabs, 100% OpenSource
- Outil de pilotage d'infrastructure Docker
- Modulaire : IHM (WebApp), Cattle, API REST, ...
- Mode : Serveur<->Agent(s)
- Support de docker-compose/rancher-compose
- Support de Mesos/Kubernetes/Swarm
- Support des principaux IaaS/PaaS via docker-machine
- RBAC (AD, LDAP, SHIBBOLETH, comptes locaux, Github
- Rancher Catalog (public ou privé) (dépôt git)

Rancher - Archi

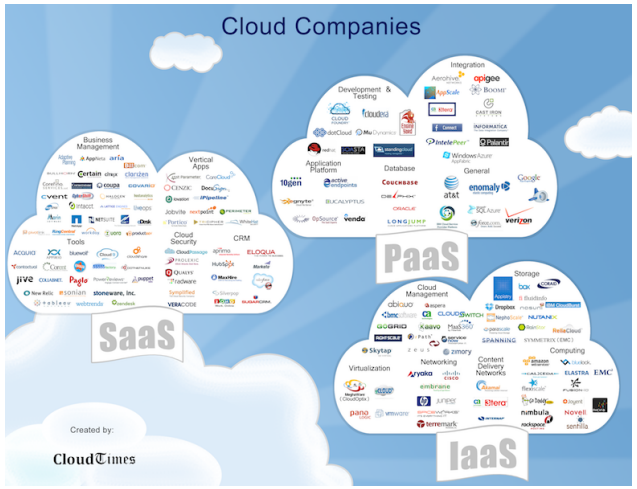


Intégration

Intégration [51]

Divers outils :

- IAAS Public : Amazon AWS, Google Cloud Platform, Microsoft Azure
- IAAS Privé : OpenStack, CloudStack
- PAAS : Scalingo [48] ,wercker [49] ,stage1.io [50] Drone.io, Codeship, CircleCI, ...
- Ansible, Puppet, Chef, Salt ...



Et bientôt des nouvelles concurrences : CoreOS Rkt, Ubuntu LXD, ...

[48] https://scalingo.com/
[49] http://wercker.com/
[50] http://stage1.io/
[51] http://cloudtimes.org/wp-content/uploads/2011/11/Clouds.cloudtimes-1024x787.png

Ressources GPU

Contexte graphique

A titre d'information car jamais testé :

- https ://github.com/NVIDIA/nvidia-docker
- https ://github.com/thewtex/docker-opengl-nvidia
- http ://gernotklingler.com/blog/howto-get-hardware-accelerated-opengl-support-docker/

SaltStack

L'idée de SaltStack

Les objectifs :

- Un framework d'exécution à distance
- Un outil de gestion de configurations/orchestration
- Un gestionnaire de configurations centralisées

Quelques exemples d'utilisation :

- Installation d'une application utilisateur
- Configuration des imprimantes, des partages réseaux, ...
- Lancer des commandes sur un ensemble de machines
- Gérer des VM ou conteneurs (construire, lancer, arrêter, ...)

Image from `http://thomason.io/wp-content/uploads/2014/04/saltstack-logo.png`

Pourquoi Salt ?

Nombreuses solutions/techniques :

- Images disques (PXE, Ghost, ...)
- Scripts maisons (Python, Bash, PowerShell, ...)
- Logiciels de gestion de conf (CFEngine, Puppet, Chef, Ansible, ...)

Le choix de Salt pour nous :

- Nouvelle génération d'outil (comparable à Ansible [53] [54] [55] [56])
- Multiplateforme (Linux, OSX, Windows), installation simple [57]
- Choix technologique : YAML, jinja2, 0MQ, AES256, Python, ...
- Communauté importante [58] et bonne documentation [59]

[53] http://ryandlane.com/blog/2014/08/04/moving-away-from-puppet-saltstack-or-ansible/
[54] https://missingm.co/2013/06/ansible-and-salt-a-detailed-comparison/
[55] http://blog.xebia.fr/2014/07/18/test-driven-infrastructure-avec-chef-episode-2/
[56] http://jensrantil.github.io/salt-vs-ansible.html
[57] http://docs.saltstack.com/en/latest/topics/installation/ubuntu.html
[58] http://saltstarters.org
[59] http://docs.saltstack.com/en/latest/topics/tutorials/quickstart.html

Architecture

Mode de fonctionnement :

- En solitaire (à titre perso, ça peut aider)
- En client-serveur (en pull ou push) : maître/larbin (master/minion)

Le lexique :

- Grain : propriétés d'un larbin (OS, RAM, ...)
- Formule/Etat (state) : états à appliquer dans un .sls en YAML
- Pilier (pillar) : paramètres à transmettre aux larbins

Les commandes :

- Salt est conçu pour exécuter des commandes à distance (modules au sens python) : test.ping (module test), cmd.run (module cmd)
- Module important : state (gestion de configuration)

Formules simples : installation d'un paquet [60]

Code Listing 10 – top.sls la racine des états

```
base: # Master base configuration
  '*': # all minion
    - apache # call apache state
```

Code Listing 11 – apache.sls fichier décrivant l'état apache

```
apache2: # Id, using like main parameter for this state
  pkg. installed # state to call : this pack must be installed
```

[60] http://docs.saltstack.com/en/latest/topics/tutorials/states_pt1.html

Formules simples : installation du paquet avec config. [61] [62]

Code Listing 12 – apache.sls fichier décrivant l'état apache avec fichier

```
apache2: # Id, using like main parameter for this state
  pkg:
    - installed # state to call : this pack must be installed
  service :
    - running # check if running
    - watch: # reload if change
      - file : /etc/apache2/httpd.conf
/etc/apache2/httpd.conf:
  file .managed:
    - source: salt://apache/httpd.conf # find the file on salt master
    - user: root
    - group: root
    - mode: 644
    - require:
      - pkg: apache2 # update this file only if apach2 is installed
```

[61] Le serveur Salt sert de serveur de fichier avec l'url "salt://"

[62] http://docs.saltstack.com/en/latest/topics/tutorials/states_pt2.html

Gestion des grains [63] [64]

Code Listing 13 – Exemple d'éxecutions distantes

```
# from master to specific minion
sudo salt serveur.example.com cmd.run reboot
sudo salt -G 'os:debian' cmd.run apt-get update
# from master to multiples minion
sudo salt 'serveur(01|02).example.com' state.sls apache
sudo salt --grain-pcre 'os:(debian|ubuntu)' state.highstate
# from minion
sudo salt-call -l debug state.highstate
```

[63] Possibilité de tester avant éxécution en ajoutant test=True

[64] aide en ligne avec sys.doc, exemple : sudo salt-call sys.doc user

Pilier/Pillar

Utilité du pilier :

- Découplage état et paramètres : variations en fonction d'un OS, d'une distribution, etc ...
- Sécurité : les larbins ont accès à tous les états, éviter d'y mettre des informations confidentielles

Création du pilier :

- Même syntaxe que les recettes et pillar.get() dans les formules

Pillier simple : communauté snmp

Code Listing 14 – snmp.sls l'état snmp

```
snmpd.conf:
    file :
      - name: /etc/snmp/snmpd.conf
      - source: salt://nux/snmp/files/snmpd.conf
      - template: jinja
      - context: snmpcom: {{ pillar.get('snmpcom', 'public') }}
```

Code Listing 15 – le fichier de conf snmp

```
rocommunity {{ snmpcom }}
```

Code Listing 16 – le fichier pillar

```
snmpcom: myCom
```

Gestion de vulnérabilités : exemple avec ShellShock

Code Listing 17 – Séquence d'éxécution salt pour ShellShock

```
sudo salt -l debug -v -G 'kernel:Linux' cmd.run "env x='() { :;}; echo _bad' bash -c '
    echo _good' 2>&1 |grep _"
srv1.domain : _bad
srv2.domain : _bad
srv3.domain : _good
srv4.domain : _bad
srv5.domain : _good
...
> sudo salt -l debug -v -G 'kernel:Linux' pkg. install bash refresh=True
> sudo salt -l debug -v -G 'kernel:Linux' cmd.run "env x='() { :;}; echo _bad' bash -
    c 'echo _good' 2>&1 |grep _"
srv1.domain : _good
srv2.domain : _good
srv3.domain : _good
srv4.domain : _good
srv5.domain : _good
```

SaltStack, c'est encore plus

Une boîte à outils en évolution :

- salt-syndic : mandataire pour les infrastructures lourdes
- returners : la suite à vos exécutions
- salt-cloud : pilotage des clouds privés et publics
- salt-ssh/ Salt Rosters
- ...

Saltstack vs Docker

Alors on fait quoi ?

- En gros, on automatise avec les 2 ...
- Mais l'un répond au besoin général et l'autre aux besoins spécifiques
- Saltstack pour tout, mais on peut gagner du temps en déploiement avec Docker
- Idéalement il faut pouvoir les intégrer pour la vue d'ensemble
- Cela sera d'autant plus vrai avec les évolutions telles qu'OpenStack
- Docker particulièrement utile pour la R&D et pour l'infra (la préprod, les fermes [65] [66] [67] [68] [69] , serveur sans données, ...)

[65] https://github.com/saltstack/salt-jenkins
[66] http://www.afewmorelines.com/an-example-jenkins-job-using-saltstack-as-a-configuration-manager/
[67] http://www.afewmorelines.com/a-full-deployment-pipeline-using-vagrant-saltstack-and-jenkins/
[68] https://registry.hub.docker.com/u/maestrodev/build-agent/
[69] http://www.logilab.org/blogentry/204921

Docker et Salt, exemple avec nodejs 1/3

Docker dans salt :

- Module [70] et State [71]
- Très récent, en gros avant juin 2014, difficile à utiliser [72] [73] [74]

Code Listing 18 – docker dans le top.sls

```
base:
    'roles:docker':
        - match: grain
        - roles.docker
        - roles.docker-nodejs
```

[70] http://docs.saltstack.com/en/latest/ref/modules/all/salt.modules.dockerio.html
[71] http://docs.saltstack.com/en/latest/ref/states/all/salt.states.dockerio.html
[72] http://serverfault.com/questions/606099/salt-cloud-docker-state-handle-volumes
[73] http://thomason.io/automating-application-deployments-across-clouds-with-salt-and-docker/
[74] https://www.bountysource.com/issues/1378476-dockerio-module-can-t-get-volumes-to-work

Docker et Salt, exemple avec nodejs 2/3

Code Listing 19 – Etat salt de docker

```
docker_repo:
    pkgrepo.managed:
      - repo: 'deb http://get.docker.io/ubuntu docker main'
      - file: /etc/apt/sources.list.d/docker.list
      - keyid: 36A1D7869245C8950F966E92D8576A8BA88D21E9
      - keyserver: keyserver.ubuntu.com
      - require_in:
          - pkg: mydocker
mydocker:
    pkg:
        - installed
        - name: lxc-docker
    service:
        - running
        - name: docker
        - require:
            - pkg: mydocker
            - pip: docker-python-dockerpy
```

Docker et Salt, exemple avec nodejs 3/3

Code Listing 20 – Etat salt du conteneur nodejs docker

```
docker_nodejs_image:
  docker.pulled:
    - name: dockerfile/nodejs
    - tag: latest
    - force: True
    - require:
      - service: mydocker
docker_nodejs_container:
  docker.installed:
    - name: mynodejs
    - hostname: mynodejs
    - image: dockerfile/nodejs
    - command: /data/test.sh
    - detach: False
    - require:
      - docker: docker_nodejs_image
docker_nodejs_running:
  ...
```

Salt dans docker

Le salt-minion :

- Déployer des images docker directement avec le client salt intégré [75] [76] [77] [78]

Montage d'une maquette salt en quelques minutes :

- Il est possible d'installer le serveur et de nombreux clients, le tout dans des conteneurs pour tester rapidement les recettes. [79] [80]

[75] https://github.com/toscanini/docker-salt-minion
[76] https://github.com/ipmb/docker-salt-minion
[77] https://github.com/kstaken/dockerfile-examples/tree/master/salt-minion
[78] http://blog.xebia.com/2014/06/14/combining-salt-with-docker/
[79] http://serverfault.com/questions/587205/lxc-and-saltstack-minion-id-configuration-on-ubuntu
[80] http://karlgrz.com/testing-salt-states-rapidly-with-docker/

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 m'envoyer vos remarques ou corrections sur johan.moreau sur gmail.com
- Ce document est distribué sous licence Creative Commons Attribution-ShareAlike 2.0