

Intégration continue

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

Johan Moreau

IRCAD/IHU

29 mai 2017

Plan

- 1 Introduction
 - Situation
 - Méthodes
 - Comment améliorer cela ?
- 2 Docker
- 3 Outils Docker
- 4 Jenkins

Situation

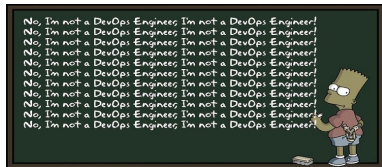
La situation - chez tout le monde ?

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



Méthodes

Quelles réponses des communautés ?

Les principes de l'agilité :

- Cycle en V trop long, cycle rapide²
- 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 ...

DevOps

Les principes du DevOps :

- Les exploitants ont des conditions différentes des développeurs, les faire travailler ensemble
- Mises en exploitation très régulières et rapides^{3 4}
- Très présent chez les leaders :
 - Amazon - toutes les 11,6s⁵,
 - Etsy - 30fois/j⁶,
 - ...
- 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>

⁴http://pages.zereturnaround.com/RebelLabs-AllReportLanders_DevopsProductivityReport.html

⁵<http://assets.en.oreilly.com/1/event/60/VelocityEngine%20Culture%20Presentation.pdf>

⁶<https://codeascraft.com/2012/03/13/making-it-virtually-easy-to-deploy-on-day-one/>

Contextes complémentaires

Microservices⁷ :

- Minimisation d'une application afin de gagner en maintenabilité
- Définition d'un service la plus claire possible
- Principe ancien (Minix vs Linux⁸) 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

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 artefacts 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, Rhodocode, ...
- Réseau social, discussion autours des problèmes, PR/MR, ...

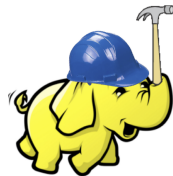
Intégration continue¹³

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

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'infrastructure définie par le code et de l'infrastructure reproductible - même approche que la recherche reproductible¹⁴

Les conteneurs applicatifs (type Docker) peuvent aider pour cela

Plan

1 Introduction

2 Docker

- Un conteneur : kezaiko ?
- Docker, la petite histoire
- Les commandes de bases
- Les images
- La persistance des données
- Le réseau
- La sécurité
- Des exemples

3 Outils Docker

4 Jenkins



Un conteneur : kezako ?

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

Les mêmes idées que la virtualisation, mais sans virtualisation :

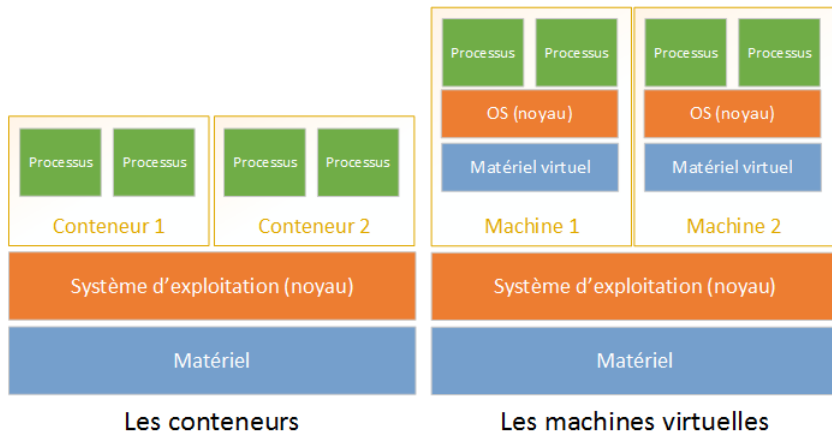
- Agnostique sur le contenu et le transporteur
- Isolation et automatisé
- Principe d'infrastructure consistante et répétable
- Peu d'overhead par rapport à une VM ! ¹⁵

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



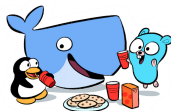
Différences entre VM et conteneur



Liaison avec le noyau Linux¹⁹ 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

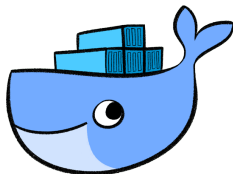
Mais natif depuis 2016 sur OSX et Windows

Docker, la petite histoire

De dotcloud à docker

La petite histoire :

- DotCloud²⁰ 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 :

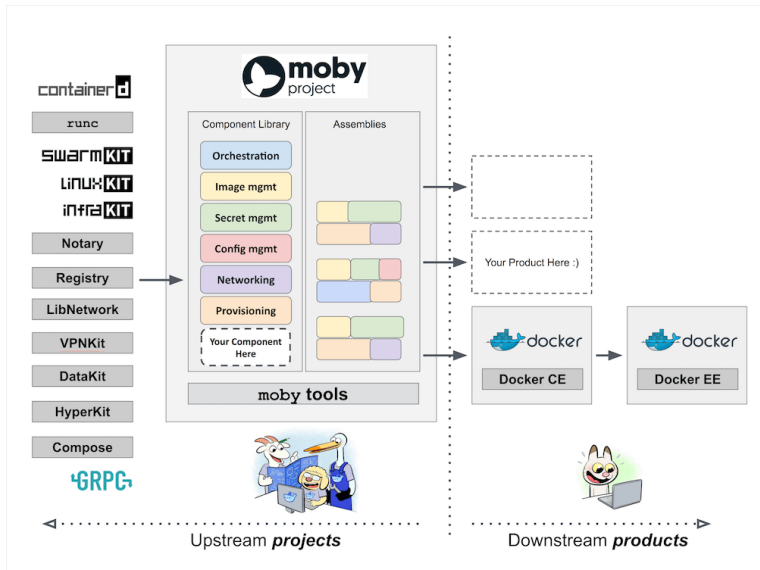
- Projet jeune en Golang²² : 1er commit le 18/01/2013
- Dans le top15 sur GitHub (approche les 800 contributeurs) en 2015
- 21 millions de téléchargement fin 2014
- Linux 64bits avec LXC/libcontainer²³

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



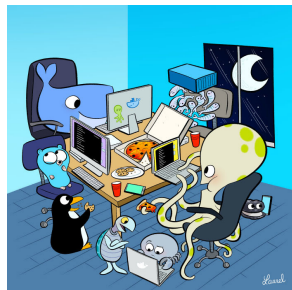
Docker

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 : élément manipulable

Analogie avec le développement objet :

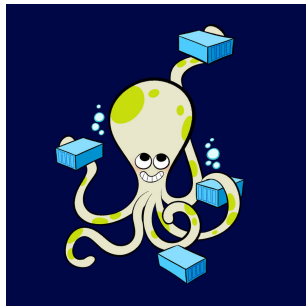
- Images équivalentes aux classes
- Les couches sont équivalentes à l'héritage
- Les conteneurs sont des instances



Docker

Les points forts :

- Installation simple (Linux, OSX, Windows)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 :
 - Gui²⁷, Orchestration, hébergement cloud, intégration continue, OS, ...



²⁶ <http://docs.docker.com/installation/>

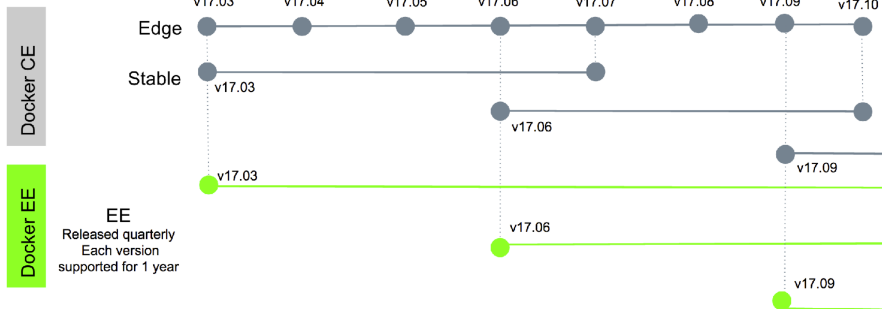
²⁷ <http://linuxfr.org/news/logiciels-pour-survivre-avec-docker>

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 EE³⁰



Version en MM.YY



Les commandes de bases

Docker 101³¹

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
```

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

- 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 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 Docker³⁵

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

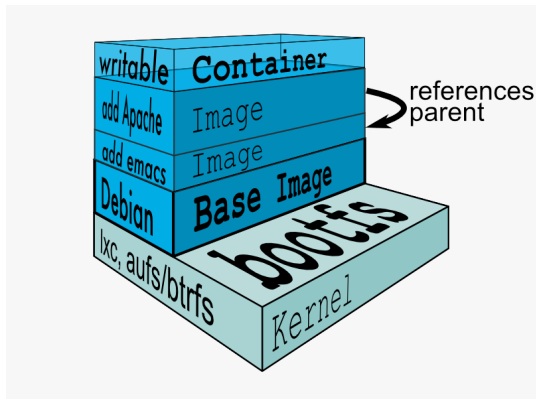
```
FROM debian:wheezy  
ADD README.md /tmp/
```

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
```

Couches d'une image Docker

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



Instructions (DSL) du Dockerfile

Les instructions sont peu nombreuses : FROM, RUN, CMD, LABEL, MAINTAINER, EXPOSE, ENV, ADD, COPY, ENTRYPOINT³⁷, VOLUME, USER, WORKDIR, ARG, ONBUILD, STOPSIGNAL, HEALTHCHECK, SHELL

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

Exercice : Créez un conteneur ping avec argument

En partant de l'image *alpine*

Créez un Dockefile appelant ping et prenant en paramètre 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 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"]
```

La persistance des données

Volume

- Outrepasser le système de COW (copy-on-write) pour profiter des performances natives d'I/O disques,
- Pour partager des fichiers/dossiers entre conteneurs ou hôte vers conteneurs
- 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
```

Exercise

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 ./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"]
```

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 : Seccomp⁴⁰ (profil apparmor, ...), capabilities⁴¹ (exemple : cap-drop ALL , cap-add CHOWN)
- Registry privée

Public vs private Registry

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, ...

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

Mettez 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, Eclipse⁴², ...

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, ...

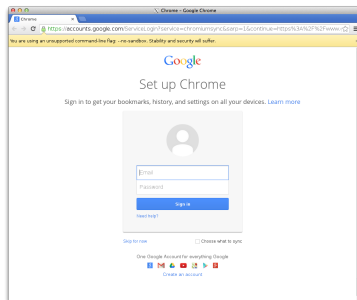
Exemple pour tout le monde

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
- Exemple⁴³ : Latex, Irssi, Mutt, Spotify, Skype

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

1 Introduction

2 Docker

3 Outils Docker

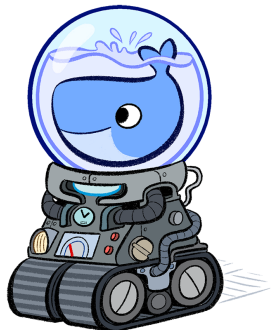
- Docker Machine
- Docker Compose
- Docker Swarm
- Rancher
- Intégration
- Ressources GPU
- SaltStack

4 Jenkins

Docker Machine

Docker Machine⁴⁵

- 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
 - OpenStack / vCenter



Docker Machine⁴⁶

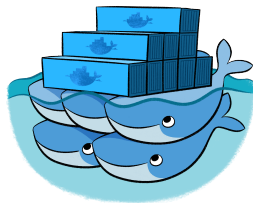
```
docker-machine create --driver virtualbox manager1
docker-machine create --driver virtualbox slave1
docker-machine ssh slave1
docker swarm join ...
docker-machine create --driver virtualbox slave2
docker-machine ssh slave2
docker swarm join ...
```

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

Docker Compose 2/2

`docker run -d --name db mysql`

db:
 image: mysql
 environment:
 MYSQL_ROOT_PASSWORD: password

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

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

`docker run -d --link phpfpd:phpfpd -p 80:80 my-nginx`

nginx:
 build : .
 links :
 - phpfpd:phpfpd
 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 nécessitant 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

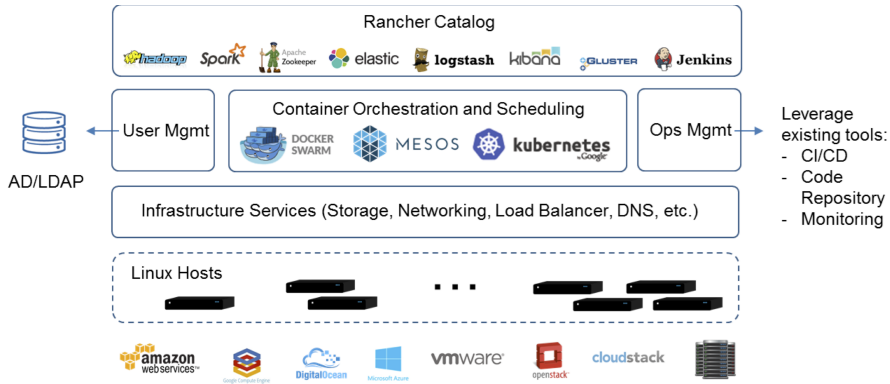
Rancher

Rancher - Basic

- Développé 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

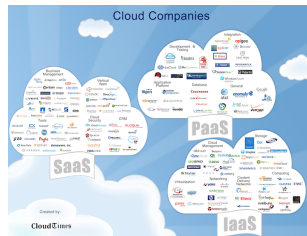


Intégration

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 LXN, ...

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, ...)

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 simple⁵⁷
- Choix technologique : YAML, jinja2, 0MQ, AES256, Python, ...
- Communauté importante⁵⁸ et bonne documentation⁵⁹

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 paquet⁶⁰

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
```

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
```

Gestion des grains^{63 64}

Code Listing 13 – Exemple d'exécutions 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
```

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

Pilier 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'exécution 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 :

- Module⁷⁰ et State⁷¹
- 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/>

Plan

- 1 Introduction
- 2 Docker
- 3 Outils Docker
- 4 Jenkins

L'intégration continue

Origines du côté de Grady Booch :

L'intégration continue est un ensemble de pratiques utilisées en génie logiciel consistant à vérifier à chaque modification de code source que le résultat des modifications ne produit pas de régression dans l'application développée.

L'intégration continue

Ok il y a des outils, mais ce qui comptent ici :

C'est de savoir ce qu'on a, ce qu'on veut, vers où on veut aller. Chacun a des projets qui sont différents avec une équipe différente, on est proche de la méthodologie de l'équipe, il n'existe pas une seule manière de faire avec les outils.

La naissance

Les origines :

- Serveur d'intégration continue,
- Ecrit en Java (Kohsuke Kawaguchi) et pour Java à l'origine,
- OpenSource (licence MIT),
- Fork en 2011 de Hudson (Sun en 2005), plus ou moins suite au rachat par Oracle



Nombreux autres outils :

- Cruisecontrol, Buildbot, GitLab CI, ...
- Travis-ci, Heroku, Codeship, CircleCI, Drone.io, ...

Jenkins

Outil très populaire :

- Opensource, +300 contributeurs en 2014, java (multi-plateforme),
- GUI web, tableau de bord, vue détail d'un "job",
- Gestion des permissions (liaison ldap possible),
- Simple d'installation (.war, distribution linux, conteneurs, recettes, ...) avec IHM,
- Groovy accessible depuis quasiment partout dans Jenkins,
- Base de plugins la plus importante dans ce domaine :
 > 1500 plugins
- Base d'installations très large, sur des sites très importants :
 - PayPal⁸² : 40 000 jobs, 12 000 builds/jours



Exemple de gui Jenkins⁸⁴

Jenkins

Jenkins

rechercher

S'identifier

Rafraîchissement automatique

- Utilisateurs
- Historique des constructions
- Relations entre les builds
- Vérifier les empreintes numériques
- Pharo Issue Tracker
- Pharo File Server
- Pharo Contribution Jenkins
- Utilisation du disque

This is the build of Pharo5

Image:

- pharo.zip
- pharo-minimal.zip

Full Download (Image+VM):

- Pharo4.0-win.zip
- Pharo4.0-mac.zip
- Pharo4.0-linux.zip

3.0	3.0-Analysis	4.0	4.0-Analysis	4.0-Bootstrap	5.0	5.0-Analysis	5.0-SysConf	5.0-VM	5.0-VM-Spur	Helper	Launcher	RaspberryPi	VM	all
S	W	Name												
		Pharo-5.0												
		Pharo-5.0-Update-Step-0-Prepare												
		Pharo-5.0-Update-Step-1-Tracker												
		Pharo-5.0-Update-Step-2.1-Validation-A-L												
		Pharo-5.0-Update-Step-2.1-Validation-M-Z												
		Pharo-5.0-Update-Step-3-Minimal												
		Pharo-5.0-Update-Step-4-Release												
		Pharo-5.0-Update-Step-5-Publish												

Icône: S M L

Légende RSS pour tout RSS de tous les échecs RSS juste pour les dernières compilations

File d'attente des constructions

File d'attente des constructions vide

État du lanceur de compilations

#	État
benchmacslave.lille.inria.fr (déconnecté)	
deblan-wheezy-32	
1 Au repos	
linux-ubuntu-10-04.ci.inria.fr	
1 Au repos	
pharo-centos6-32.inria.fr	
1 Au repos	
2 Au repos	
pharo-linux.ci.inria.fr	
1 Au repos	
pharo-linux64-2.ci.inria.fr	
1 Au repos	
2 Au repos	
3 Au repos	
pharo-linux64.ci.inria.fr	
1 Au repos	
2 Au repos	
3 Au repos	
4 Au repos	
pharo-ubuntu-13-04.ci.inria.fr (déconnecté)	
pharo-windows-xp.ci.inria.fr (déconnecté)	
pharo-wolfvare.ci.inria.fr	

Plugins Jenkins

Plugins en tout genre :

- Outils de constructions,
- Outils d'analyse de code,
- Outils de gestion de sources,
- Outils de gestion de configurations, de VM, de conteneurs, ...

Jenkins Swarm⁸⁵ :

- Plugin d'auto-découverte du master pour les esclaves,
- Prévu pour la gestion des clusters, ...

Jenkins est capable d'installer ses plugins tout seul.

Jenkins est capable de lire sa configuration de job depuis un fichier.

Les plugins

Leur installation :

- Téléchargé depuis <https://updates.jenkins-ci.org>
- Peuvent-être installés depuis l'IHM ou en CLI

Leur création :

- Nombreux tuto
- Mécanismes de dépendances pour aller vite au fonctionnel
- Architecture de tests et distribution proposé par la communauté

Jenkins en 2016/2017

Aujourd'hui :

- A dépasser le stade du simple outil d'intégration continue ;
- Certains l'utilisent aussi pour la gestion des artefacts et le déploiement continu ;
- Devient un outil d'orchestration à part entière (discutable)
- Les droits des jobs ne sont pas nécessairement ceux du dépôt de sources

2 limites début 2016 :

- Pour ceux qui veulent un job de CI lié au code sources ...
 - Jenkins pipeline
- IHM discutable ...
 - Blue Ocean

Liaison possible avec GitLab⁸⁶

mkdir jenkins redis postgresql gitlab registry helloWorldC

Ajouter la registry dans le docker-compose.yml précédent

Bien utilisé Azerty123 comme password, root comme user (mail
root@root.com)

Au lancement de Jenkins, ne garder que les plugins "Pipeline" et git
Ajouter à Jenkins, les plugins "Self-Organizing Swarm Plug-in Modules",
"Config File Provider Plugin" et "Gitlab Plugin"
Naviguer dans les menus Gitlab et Jenkins

La console

Outil pratique pour comprendre l'état du serveur ou automatiser quelques tâches (administration, qualité, ...)

Code Listing 21 – Ping en groovy avec retour std du shell

```
println "ping -c1 gitlab".execute().text
```

Les esclaves

Les esclaves :

- On peut travailler sur le serveur, mais ce n'est pas une bonne pratique.
- Il existe plusieurs types d'esclaves :
 - ssh, jnlp, swarm

Exemple pour un swarm c/c++ en docker :

- heliostech/jenkins-slave-gcc :centos-7 -master
- Exercice ...

Exercice : jenkins swarm c/c++ en docker

Code Listing 22 – Conteneur Jenkins Swarm avec env C

```
#!/bin/sh
docker run -v "$PWD"/jenkins/.ssh:/home/jenkins-slave/.ssh:ro --name slaveC --rm
--network ci_default -d heliostech/jenkins-slave-gcc:centos-7 --master http://
jenkins:8080/ --username root --password "Azerty123" --executors 2 --labels "linux
centos-7-gcc"
```

Notre 1er code jenkins

Il nous faut un bout de code !

Code Listing 23 – HelloWorld en C++

```
#include <iostream>
int main()
{
    std :: cout << "Hello, world!" << std::endl;
}
```

Code Listing 24 – Makefile HelloWorld.cpp

```
all :
    g++ main.cpp -o hello
```

Droits minimaux⁸⁹

Il va nous falloir :

- un token gitlab (création) pour que jenkins (enregistrement) puisse accéder à l'interface Gitlab
- une clé SSH depuis Jenkins (création) vers GitLab (enregistrement) pour la récupération du code

Jenkins > Credentials > System > Global credentials (unrestricted) >

[Back to credential domains](#)

[Add Credentials](#)

Kind:

Scope:

API token:

ID:

Description:

Profile Settings

[Profile](#) [Account](#) [Applications](#) [Access Tokens](#) [Emails](#) [Password](#) [Notifi](#)

Public Avatar

Image from https://cdn-images-1.medium.com/max/1600/1*9rVAy2J90-Itn4aG89T8wA.png
 Image from https://cdn-images-1.medium.com/max/1600/1*dDwVPvBGr1ndWzyoYRo9rA.png
<https://medium.com/@teeks99/continuous-integration-with-jenkins-and-gitlab-fa770c62e88a>

ih
89

Droits Jenkins/Gitlab - Exercice

Aller sur votre profil Gitlab et récupérez la commande openssl pour créer votre clé

Faite un docker exec sur le conteneur jenkins et exécutez la commande (copier le contenu de .ssh/id_rsa.pub)

Faire un test avec ssh git@gitlab

Aller sur votre profile Gitlab et créer un token pour jenkins

Aller dans la configuration de Jenkins, créez et testez une connection Gitlab

API - Jenkins

Code Listing 25 – Create a job with Jenkins API

```
#!/bin/sh
if [ $# -ne 4 ] then
    echo "No arguments supplied"
    echo "1st argument : is your Jenkins user"
    echo "2nd argument : is your Jenkins password"
    echo "3th argument : is your job"
    echo "4th argument : is the jenkins type of your job [ freestyle | pipeline ]"
    exit 1
fi
USER=$1
APITOKEN=$2
JOB=$3
GROOVYSCRIPT=list-jobs.groovy
SERVER=127.0.0.1:10081
CRUMB='curl "http://$USER:$APITOKEN@$SERVER/crumbIssuer/api/xml/?xpath=concat(//crumbRequestField,%22:%22,//crumb)"'
curl -v -X POST --data-binary "@Jenkins/"$4".xml" -H "Content-Type:application/xml" -H "$CRUMB" "http://$USER:Azerty123@$SERVER/createItem?name=$JOB"
```

API - Gitlab

Code Listing 26 – Create a project with GitLab API

```
#!/bin/sh
if [ $# -ne 2 ]
then
    echo "No arguments supplied"
    echo "First argument is your GitLab token"
    echo "Second argument is your new project"
    exit 1
fi
curl --header "PRIVATE-TOKEN:$1" -X POST "http://127.0.0.1:10080/api/v3/projects?
name=$2"
IDS='curl --header "PRIVATE-TOKEN:$1" GET "http://127.0.0.1:10080/api/v4/projects?
search=$2&simple=true"'
ID='echo "$IDS"|tr " ," '\n'|sed -e "s/\"//g"|grep "id:"|sed -e 's/\"/{id :/'
curl --header "PRIVATE-TOKEN:$1" -X POST "http://127.0.0.1:10080/api/v4/projects/
$ID/hooks?url=http://jenkins:8080/project/$2&push_events=true&token=
d738cdacfb22fcce95ae5cfa7c065972"
```

API - Les 2 en mêmes temps

Code Listing 27 – Appel les 2 précédents scripts

```
#!/bin/bash

if [ $# -ne 5 ]
then
    echo "No arguments supplied"
    echo "1st argument : is your Jenkins user"
    echo "2nd argument : is your Jenkins password"
    echo "3th argument : is your Gitlab token"
    echo "4th argument : is your job/ project"
    echo "5th argument : is the jenkins type of your job [ freestyle | pipeline ]"
    exit 1
fi

./Gitlab/ gitlabcreateProject .sh $3 $4
./Jenkins/jenkinscreateJob.sh $1 $2 $4 $5
```

Job Jenkins

De quoi s'agit-il ?

- Projet de travail pouvant (mais pas nécessairement) être lié à un dépôt de sources
- Les jobs peuvent travailler ensemble
- Les droits des jobs ne sont pas nécessairement ceux du dépôt de sources
- Il existe plusieurs type de job :
 - Principalement : "freestyle" et "pipeline"
 - Multibranch Pipeline, Folder, ...

Job freestyle - création


Jenkins

[Jenkins](#) > [All](#) >

 **New Item**

 People

 Build History

 Manage Jenkins

 Credentials

Build Queue

No builds in the queue.

Build Executor Status

1 Idle
 2 Idle

☐ **Freestyle project**
 This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

☐ **Maven project**
 Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

☐ **External Job**
 This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system. See [the documentation for more details](#).

☐ **Multi-configuration project**
 Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.


Jenkins

[admin](#) | [log out](#)
[Jenkins](#) > [All](#) >

Enter an item name

* This field cannot be empty, please enter a valid name


Freestyle project
 This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.


Pipeline
 Orchestrates long-running activities that can span multiple build slaves. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.


Multi-configuration project
 Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.


External Job
 This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system. See [the documentation for more details](#).


Folder
 Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.


Multibranch Pipeline
 Creates a set of Pipeline projects according to detected branches in one SCM repository.

Job freestyle - param

Project name: HelloWorld

Description: [Empty text area]

[Raw HTML] [Preview](#)

- ☐ Discard Old Builds
- ☐ This build is parameterized
- ☐ Disable Build (No new builds will be executed until the project is re-enabled.)
- ☐ Execute concurrent builds if necessary

Advanced Project Options [Advanced...](#)

Source Code Management

- ☐ CVS
- ☐ CVS Projectset
- ☐ Git
- ☒ None
- ☐ Subversion
- ☐ Team Foundation Server

Build Triggers

- ☐ Build after other projects are built
- ☐ Build periodically
- ☐ Poll SCM

Build Environment

Exercice

Créer un projet test1 sur gitlab, créer un job freestyle test1 sur jenkins
Lier Jenkins/Gitlab

Pipeline

2 versions :

- Imperative (aka scripted) Pipeline (2016)
- Declarative Pipeline (2017)

Jenkins Jobs as Code :

- Jenkinsfile
- Proche d'un travis.yml ou d'un .gitlab-ci.yml

Pipeline C++

Code Listing 28 – Pipeline CPP

```
node("centos-7-gcc") {  
    stage('checkout') {  
        checkout([$class: 'GitSCM',  
            userRemoteConfigs: [[  
                credentialsId: 'sshkey',  
                url: "ssh:// git@gitlab /root/ test . git "]]  
            ])  
    }  
    stage('make') {  
        gitlabCommitStatus("build") {  
            sh "make"  
        }  
    }  
}
```

Declarative pipeline

Code Listing 29 – Declarative Pipeline Junit

```
pipeline {
  agent {
    docker 'maven:3.3.3'
  }
  stages {
    stage('Build') {
      steps {
        git url: 'http://github.com/JohanMoreau/JunitTest.git'
        sh "mvn -B clean install -Dmaven.test.failure.ignore=true"
      }
    }
    stage('Archive') {
      steps {
        junit '**/target/ surefire -reports/TEST-*.xml'
        sh 'ls -al target/'
        archive 'target/*.jar'
      }
    }
  }
  post {
    always {
      deleteDir ()
    }
  }
}
```

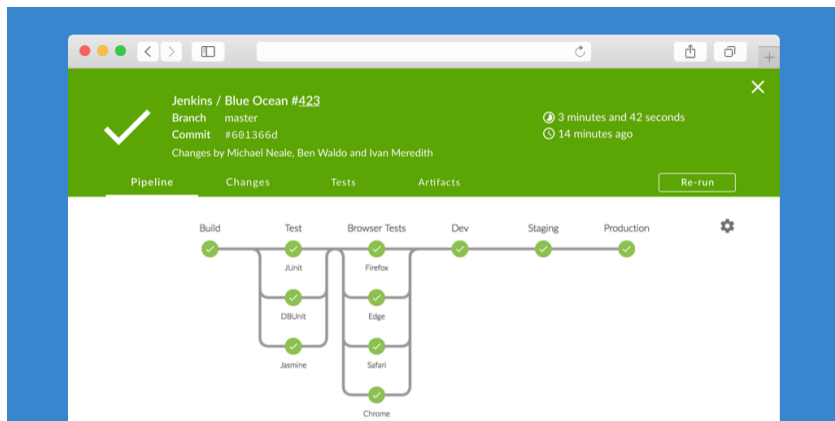
Pipeline Docker

Code Listing 30 – Pipeline Docker

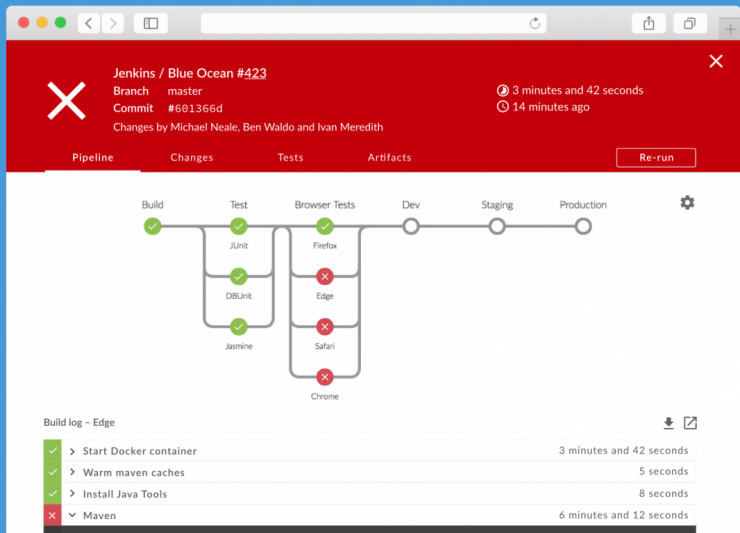
```
node('docker'){
    stage('Checkout') {
        git url: 'http://github.com/JohanMoreau/dev-koans.git'
    }

    stage('Build docker') {
        sh("cp Docker/webapppy/Dockerfile Python/FlaskBasic/")
        sh("cd Python/FlaskBasic && ls -al && docker build -t webapppy . ")
    }
}
```

BlueOcean



BlueOcean



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](mailto:johan.moreau@gmail.com)
- Ce document est distribué sous licence Creative Commons Attribution-ShareAlike 2.0