De la chaîne de production au SI géré par des logiciels

Johan Moreau

IRCAD/IHU

14 décembre 2015

Petit sondage ...

- Qui utilise des conteneurs qui ne sont pas basés sur Docker ?
- Qui utilise une gestion de configuration qui n'est pas basée sur Saltstack ?
- Qui utilise une intégration continue qui n'est pas basée sur Jenkins ?
- Qui a déjà vu une présentation Docker ? Saltstack ? Jenkins ?
- Qui utilise Docker ? Saltstack ? Jenkins ?
- Qui utilise 2 des 3 outils ? Les 3 ?

La problématique

Soucis des "Dev" :

- Développer des logiciels scientifiques
- Développer des logiciels pour l'infrastructure du laboratoire

Soucis des "Ops" :

- Distribuer et mettre en production ces logiciels
- Mettre en place les serveurs pour tout cela
- Mettre à disposition les machines pour ces développements

Soucis de l'équipe :

- Satisfaire les utilisateurs de ces logiciels
- Satisfaire les développeurs et administrateurs

Plan

1. Jenkins pour le moteur d'intégration continue
2. Docker pour l'installation des esclaves
3. Saltstack pour l'orchestration globale
4. Architecture globale et points de réflexion

Intégration continue [2]

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" :

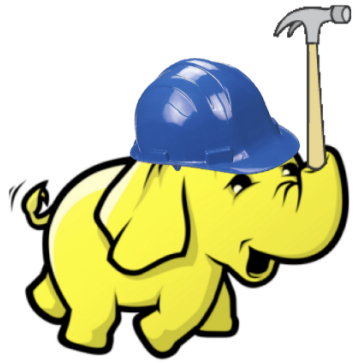
- Compiler le code,
- Lancer les tests unitaires,
- Analyser le code,
- Générer un exécutable ou une archive,
- Générer des rapports, ...

Plus un bug est découvert tôt, moins d'impact aura sa correction [1]

[1] http://martinfowler.com/articles/continuousIntegration.html
[2] http://fr.wikipedia.org/wiki/Int%C3%A9gration_continue

La construction et les artefacts

Ceci implique donc une construction :



- Rapide et flexible
- Standardisée (CMake, Ant, Maven, NAnt, SCons, ...)
- Identique pour toutes les cibles (OS, architecture, ...)

Des artéfacts intéressants :

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

Maintenir un build opérationnel n'est pas facile ! [4]

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

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

Jenkins

Outil très populaire :

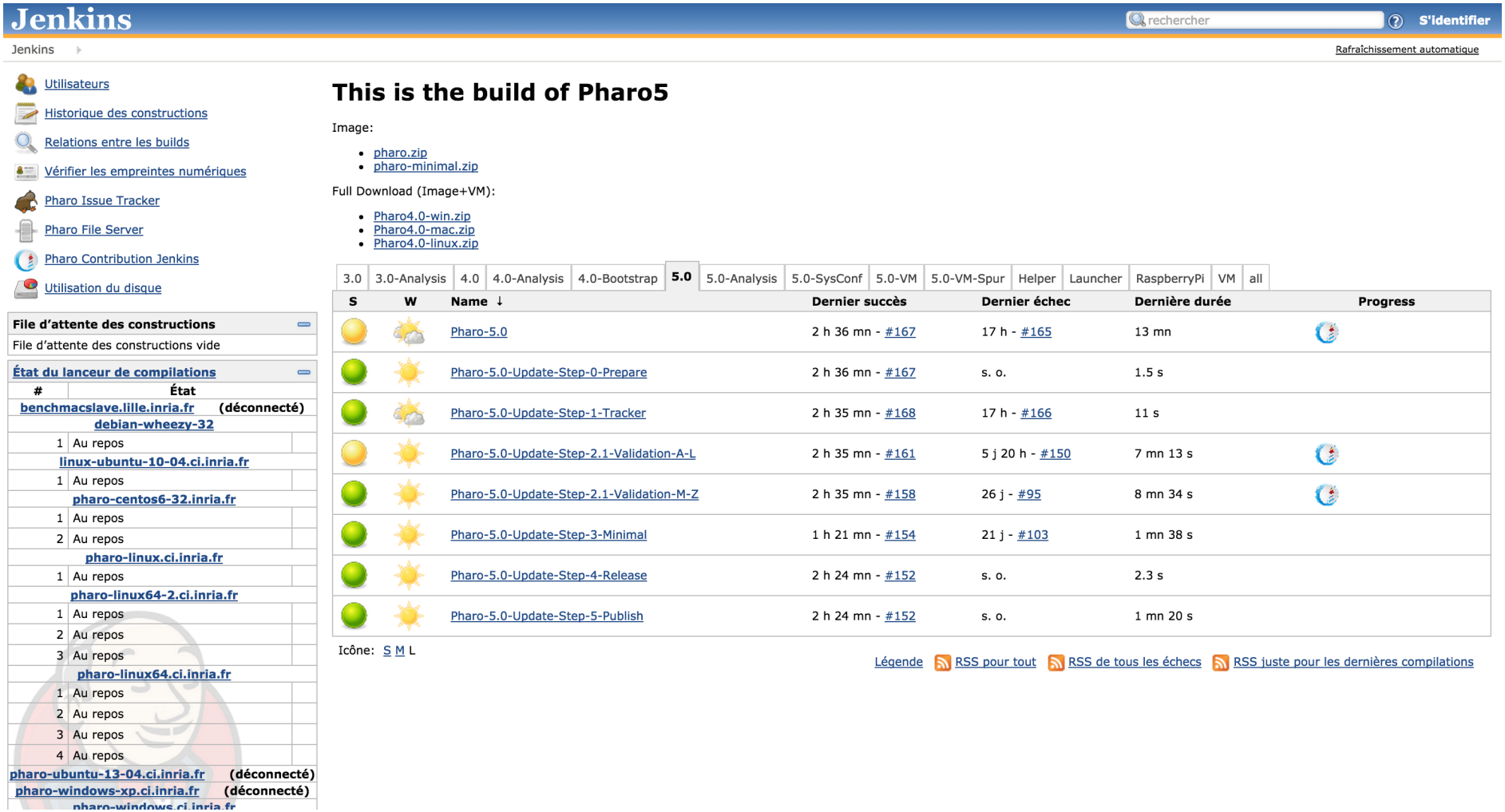
- Opensource, +300 contributeurs en 2014, java (multi-plateforme),
- GUI web, tableau de bord, vue détail d'un "job",
- Gestion des permissions (liaison ldap possible),
- Base de plugins la plus importante dans ce domaine,
- Base d'installations très large, sur des sites très importants :
  - PayPal[5] : 40 000 jobs, 12 000 builds/jours

Nombreux autres outils :

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

[5] https://www.cloudbees.com/sites/default/files/how_to_train_your_jenkins-manivannan.pdf
http://mirrors.clinkerhq.com/jenkins/art/jenkins-logo/1024x1024/logo.png

Exemple de gui Jenkins[7]



[7] https://ci.inria.fr/pharo

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 Swarm [8] **:**

- 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.

[8] https://wiki.jenkins-ci.org/display/JENKINS/Swarm+Plugin

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 [9]

Une solution pour ça ?

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

Plan

1 Jenkins pour le moteur d'intégration continue

2 Docker pour l'installation des esclaves

3 Saltstack pour l'orchestration globale

4 Architecture globale et points de réflexion

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

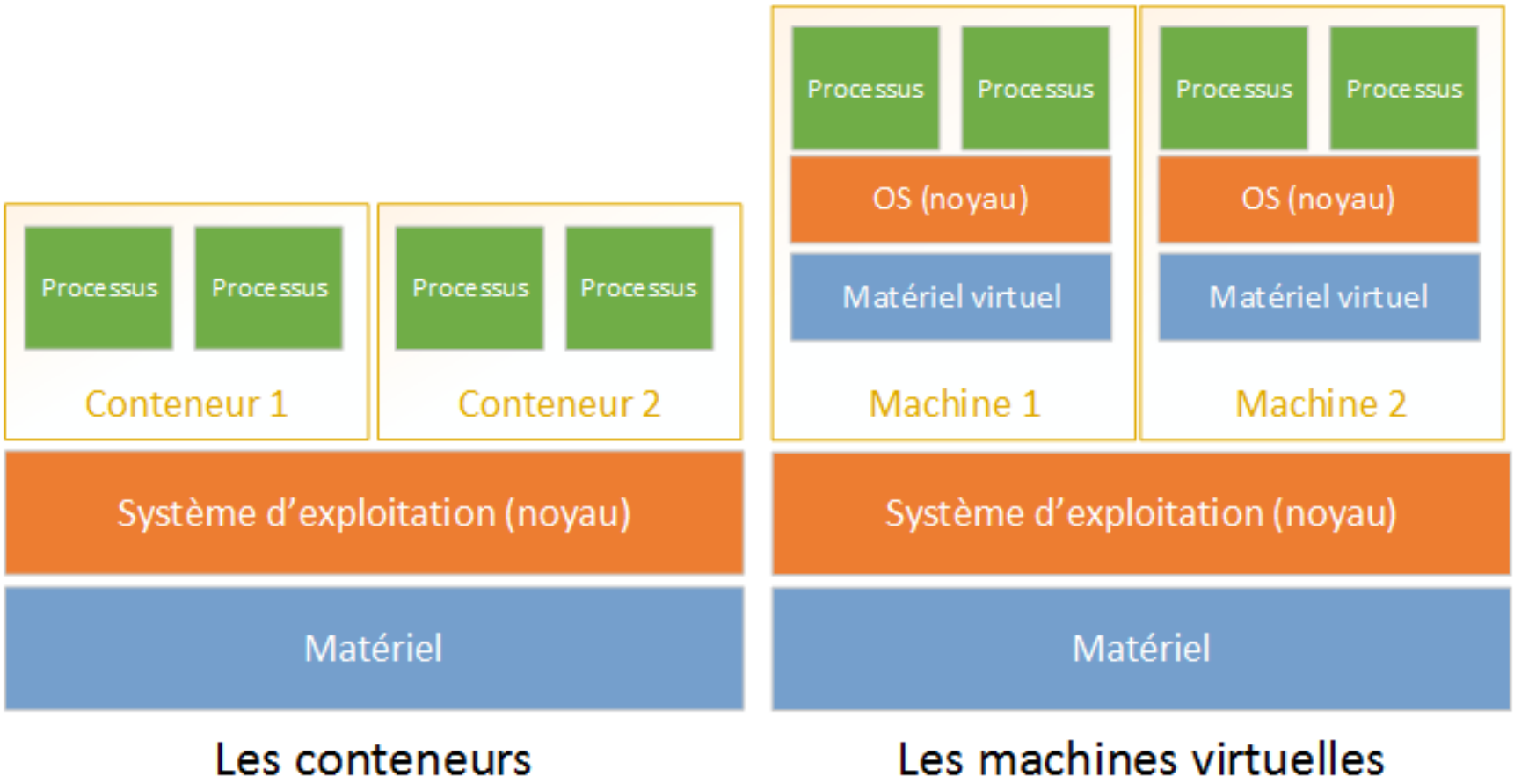
Les mêmes idées que la virtualisation, mais sans virtualisation :

- Agnostique sur le contenu et le transporteur
- Isolation et automatisation
- Principe d'infrastructure consistante et répétable
- Peu d'overhead par rapport à une VM ! [10]

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 ...

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

Différences entre VM et conteneur



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

Liaison avec le noyau Linux [13]

Une belle évolution de Linux :

- Mount namespace (Linux 2.4.19)
- PID namespace (Linux 2.6.24) / plusieurs PID par 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 par couche, mode union, copy on write

En gros, les éléments noyaux sont en place depuis 3.8

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

De dotcloud à docker

La petite histoire :

- DotCloud [14] 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 [16] : 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 [17]

[14] http://fr.slideshare.net/jpetazzo/introduction-to-docker-december-2014-tour-de-france-bordeaux-special
https://www.docker.com/sites/default/files/legal/small_v.png

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

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

Docker

Terminologie :

- index : répertoire public (https ://index.docker.io/)
- image : conteneur en lecture seule (snapshot)
- conteneur : élement manipulable
- run : créer un conteneur (à partir d'une image)

Les points forts :

- Installation simple (Linux, OSX, Windows) [18]
- Ligne de commande très sympathique (docker help)
- Langage de description des images (avec notion de parent)
- API pour pilotage, écosystème jeune mais déjà énorme :
  - Gui [19], Orchestration, hébergement cloud, intégration continue, OS, ...

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

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

Docker 101

```
$ 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
```

Docker in the shell [20] [21] [22]

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
```

[20] http://ippon.developpez.com/tutoriels/virtualisation/docker-presentation-part-1/

[21] http://ippon.developpez.com/tutoriels/virtualisation/docker-presentation-part-2/

[22] http://sametmax.com/le-deploiement-par-conteneurs-avec-docker/

Construire sa propre image [23]

Description et construction :

Code Listing 2– Dockerfile

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

Code Listing 3– Exécution d'un build Docker

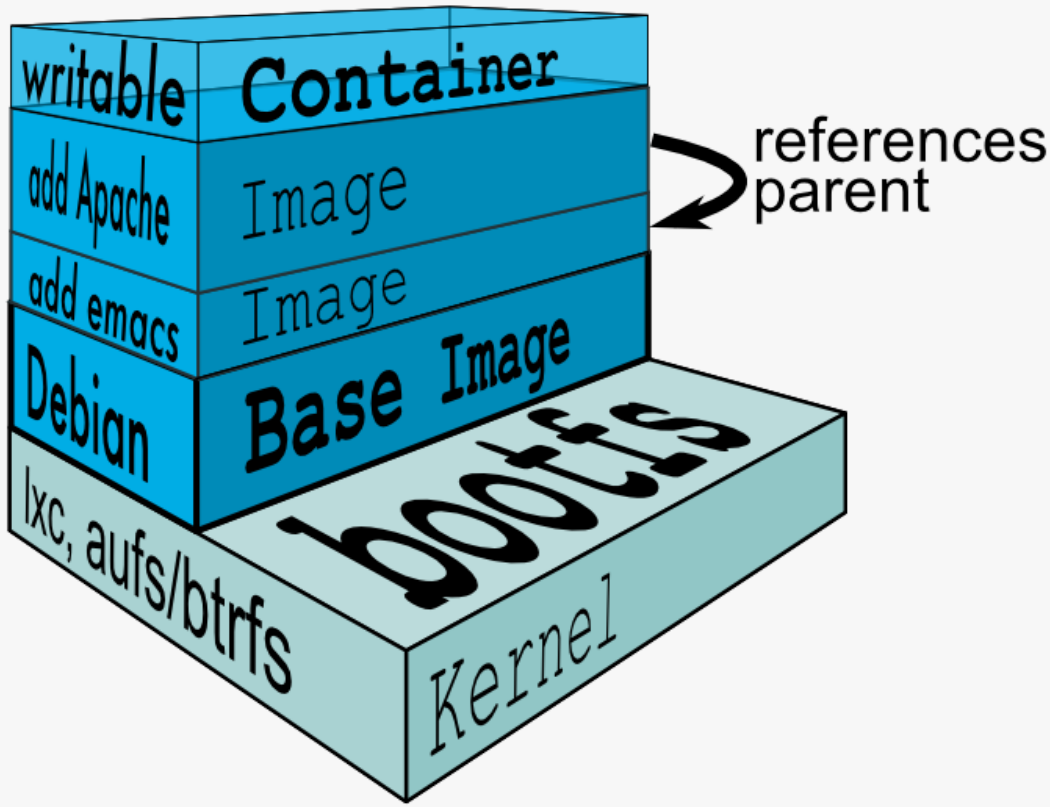
```
$ docker build -q -t readme .
Sending build context to Docker daemon 3.584 kB
Sending build context to Docker daemon
Step 0 : FROM debian:wheezy
 ---> e8d37d9e3476
Step 1 : ADD README.md /tmp/
 ---> 09eabce38f39
Removing intermediate container 3e44a3b6eabe
 Successfully built 09eabce38f39
```

Liens entre conteneurs :

```
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
```

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

Couches d'une image Docker



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

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 [25], ...

pour "l'Ops" :

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

[25] 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
- Permet de garder propre l'OS sous jacent :
  - Latex, environnement lourd, nombreuses dépendances
- Exemple [26] : Latex, Irssi, Mutt, Spotify, Skype

Depuis où :

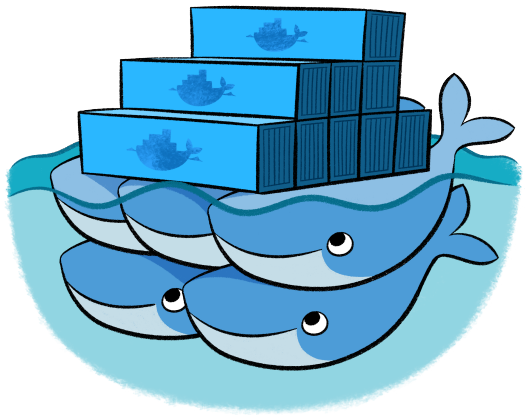
- boot2docker pour Windows et OSX
- Docker Toolbox / kitematic pour OSX/Win

[26] https://blog.jessfraz.com/post/docker-containers-on-the-desktop/
https://avatars3.githubusercontent.com/u/8460345

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

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
```

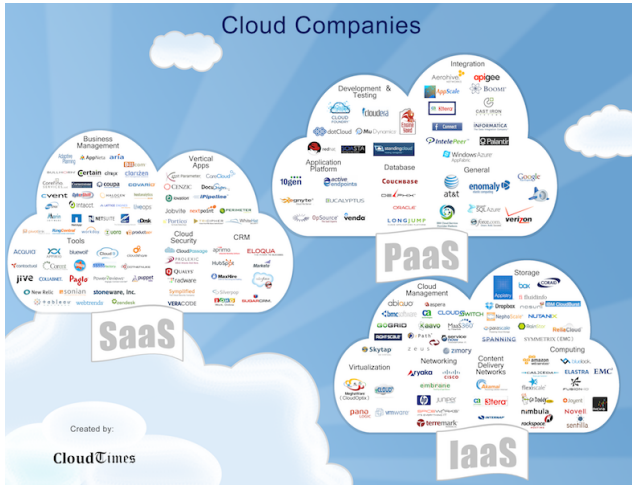
docker run -d --link phpfpm :phpfpm -p 80 :80 my-nginx

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

Intégration [29]

Divers outils :

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



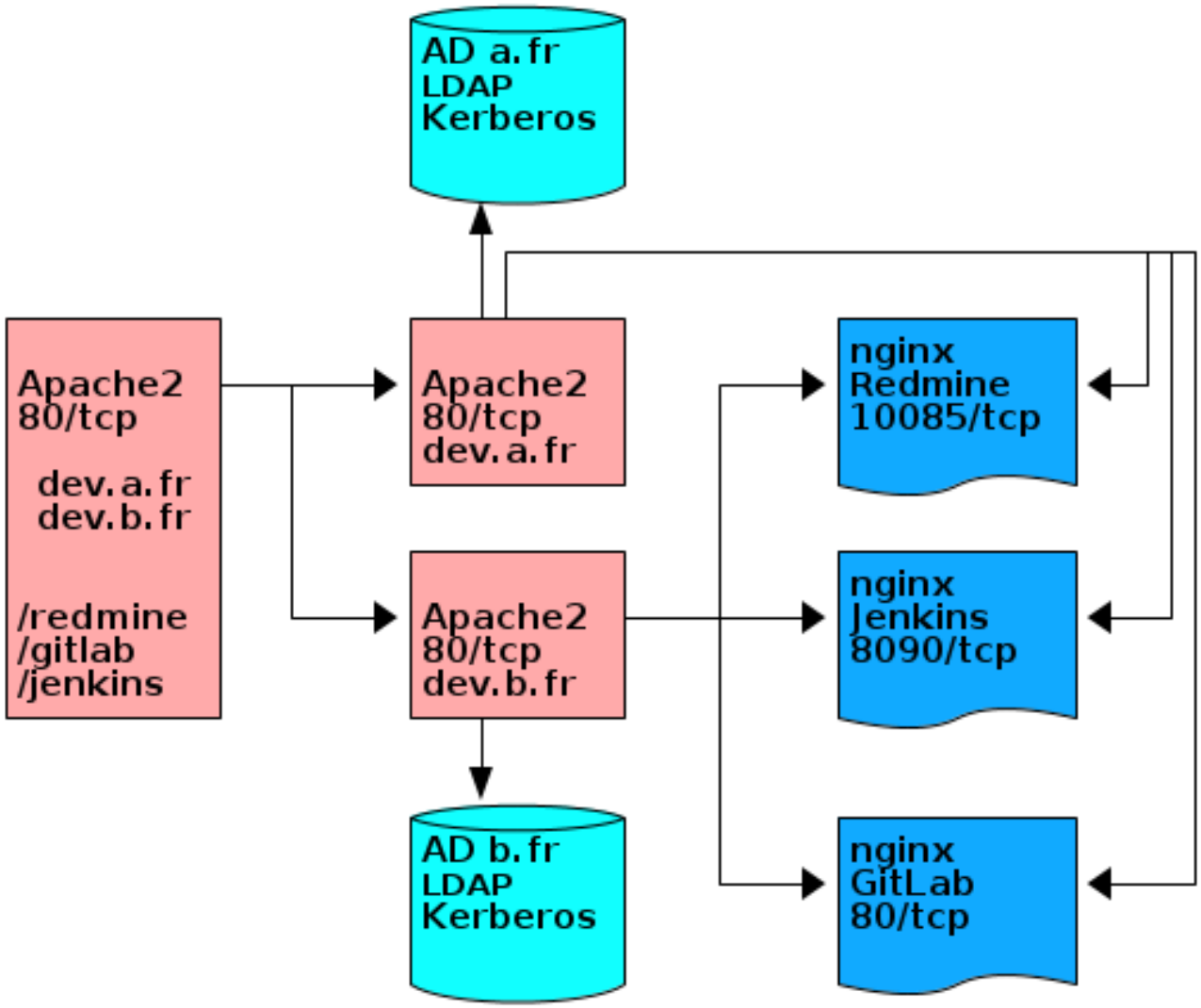
Et bientôt des nouvelles concurrences : CoreOS Rkt, Ubuntu LXD, ...

[29] http://cloudtimes.org/wp-content/uploads/2011/11/Clouds.cloudtimes-1024x787.png

Quelle usage de Docker dans notre exemple ?

- Les serveurs de test/pré-prod des applications légères (node.js, python, ...)
- Infrastructure Jenkins : master / esclaves (applications lourdes, applications légères)
- Redmine, Owncloud, wordpress (en cours)
- Simplification d'architecture complexe (authentification)
- Batchs de backup
- Latex, ditaa, ...

Exemple avec 6 conteneurs



Orchestrer Docker c'est bien, mais peut-on faire plus ?

Docker :

- L'écosystème se développe bien, mais tout n'est pas un conteneur

Le reste donc :

- Cela serait plus simple si l'esclave jenkins avait la même configuration que le développeur (possible dans certains avec Docker)
- Notre infrastructure a d'autres OS
- Notre infrastructure n'est pas encore bien pensée pour du microservice

Plan

1 Jenkins pour le moteur d'intégration continue

2 Docker pour l'installation des esclaves

3 Saltstack pour l'orchestration globale

4 Architecture globale et points de réflexion

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, ...)

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 [31] [32] [33] [34])
- Multiplateforme (Linux, OSX, Windows), installation simple [35]
- Choix technologique : YAML, jinja2, 0MQ, AES256, Python, ...
- Communauté importante [36] et bonne documentation [37]

[31] http://ryandlane.com/blog/2014/08/04/moving-away-from-puppet-saltstack-or-ansible/

[32] https://missingm.co/2013/06/ansible-and-salt-a-detailed-comparison/

[33] http://blog.xebia.fr/2014/07/18/test-driven-infrastructure-avec-chef-episode-2/

[34] http://jensrantil.github.io/salt-vs-ansible.html

[35] http://docs.saltstack.com/en/latest/topics/installation/ubuntu.html

[36] http://saltstarters.org

[37] 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 [38]

Code Listing 4– top.sls la racine des états

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

Code Listing 5– 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
```

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

Formules simples : installation du paquet avec config. [39] [40]

Code Listing 6– 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
```

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

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

Gestion des grains [41] [42]

Code Listing 7– 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
```

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

[42] 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 8– 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 9– le fichier de conf snmp

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

Code Listing 10– le fichier pillar

```
snmpcom: myCom
```

Gestion de vulnérabilités : exemple avec ShellShock

Code Listing 11– 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 [43] [44] [45] [46] [47] , serveur sans données, ...)

[43] https://github.com/saltstack/salt-jenkins

[44] http://www.afewmorelines.com/an-example-jenkins-job-using-saltstack-as-a-configuration-manager/

[45] http://www.afewmorelines.com/a-full-deployment-pipeline-using-vagrant-saltstack-and-jenkins/

[46] https://registry.hub.docker.com/u/maestrodev/build-agent/

[47] http://www.logilab.org/blogentry/204921

Docker et Salt, exemple avec nodejs 1/3

Docker dans salt :

- Module [48] et State [49]
- Très récent, en gros avant juin 2014, difficile à utiliser [50] [51] [52]

Code Listing 12– docker dans le top.sls

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

[48] http://docs.saltstack.com/en/latest/ref/modules/all/salt.modules.dockerio.html

[49] http://docs.saltstack.com/en/latest/ref/states/all/salt.states.dockerio.html

[50] http://serverfault.com/questions/606099/salt-cloud-docker-state-handle-volumes

[51] http://thomason.io/automating-application-deployments-across-clouds-with-salt-and-docker/

[52] https://www.bountysource.com/issues/1378476-dockerio-module-can-t-get-volumes-to-work

Docker et Salt, exemple avec nodejs 2/3

Code Listing 13– 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 14– 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 [53] [54] [55] [56]

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. [57] [58]

[53] https://github.com/toscanini/docker-salt-minion

[54] https://github.com/ipmb/docker-salt-minion

[55] https://github.com/kstaken/dockerfile-examples/tree/master/salt-minion

[56] http://blog.xebia.com/2014/06/14/combining-salt-with-docker/

[57] http://serverfault.com/questions/587205/lxc-and-saltstack-minion-id-configuration-on-ubuntu

[58] http://karlgrz.com/testing-salt-states-rapidly-with-docker/

Plan

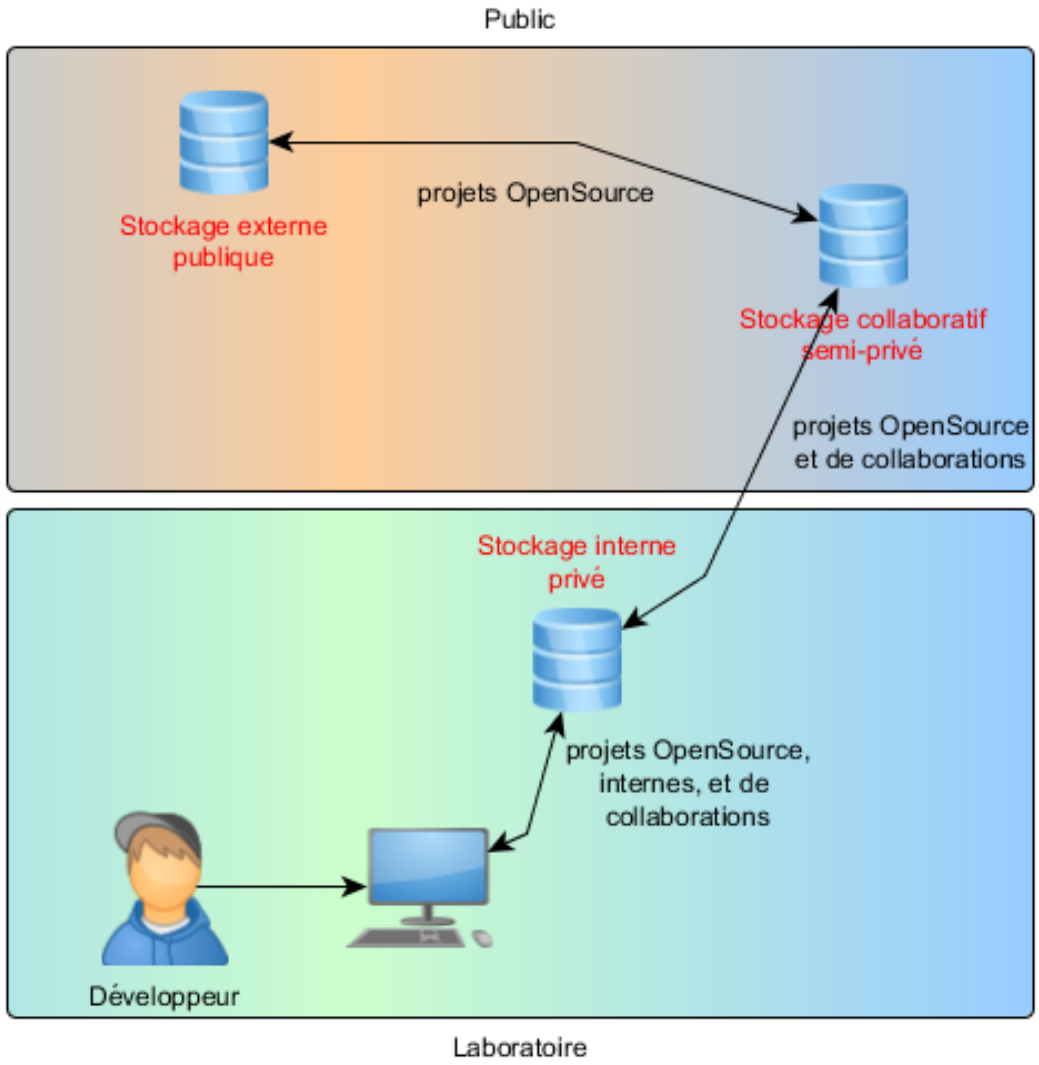
1 Jenkins pour le moteur d'intégration continue

2 Docker pour l'installation des esclaves

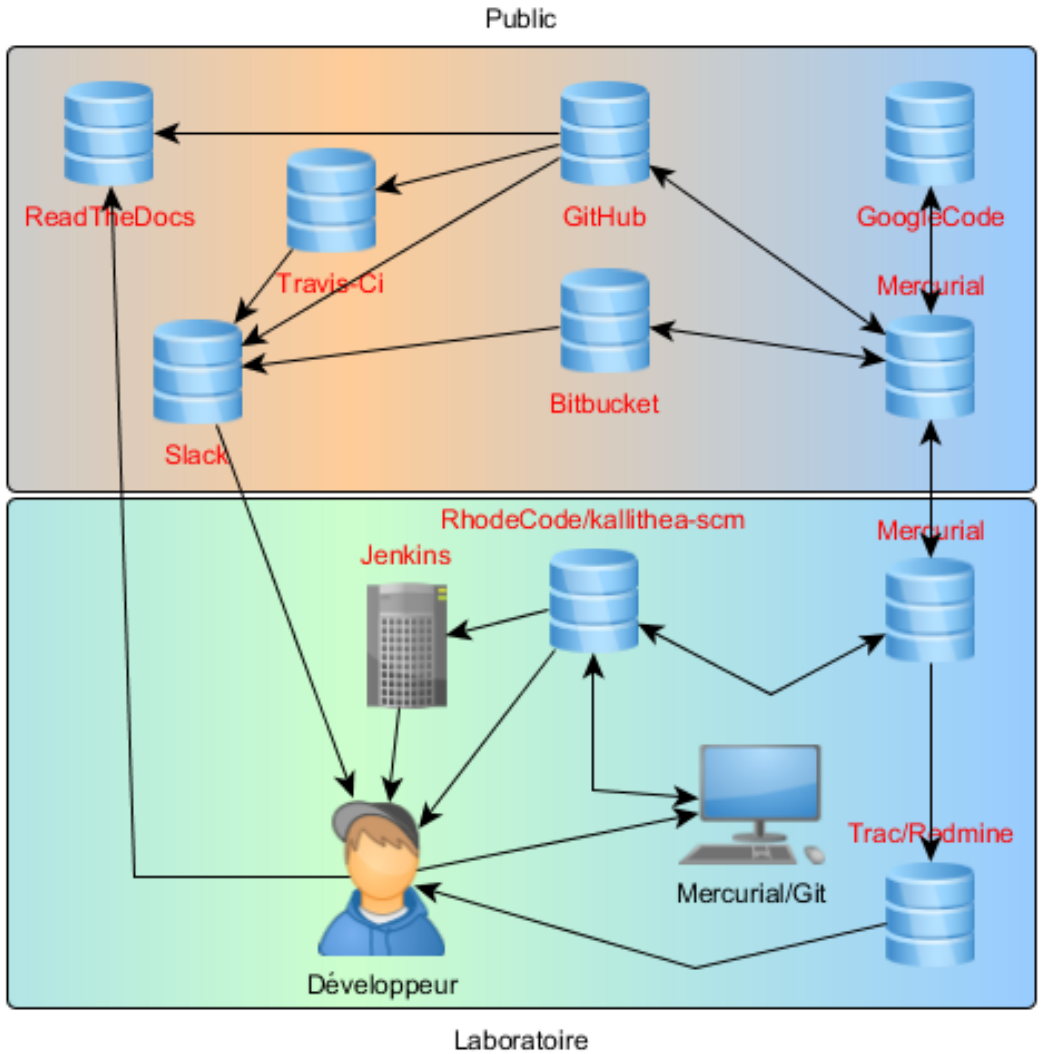
3 Saltstack pour l'orchestration globale

4 Architecture globale et points de réflexion

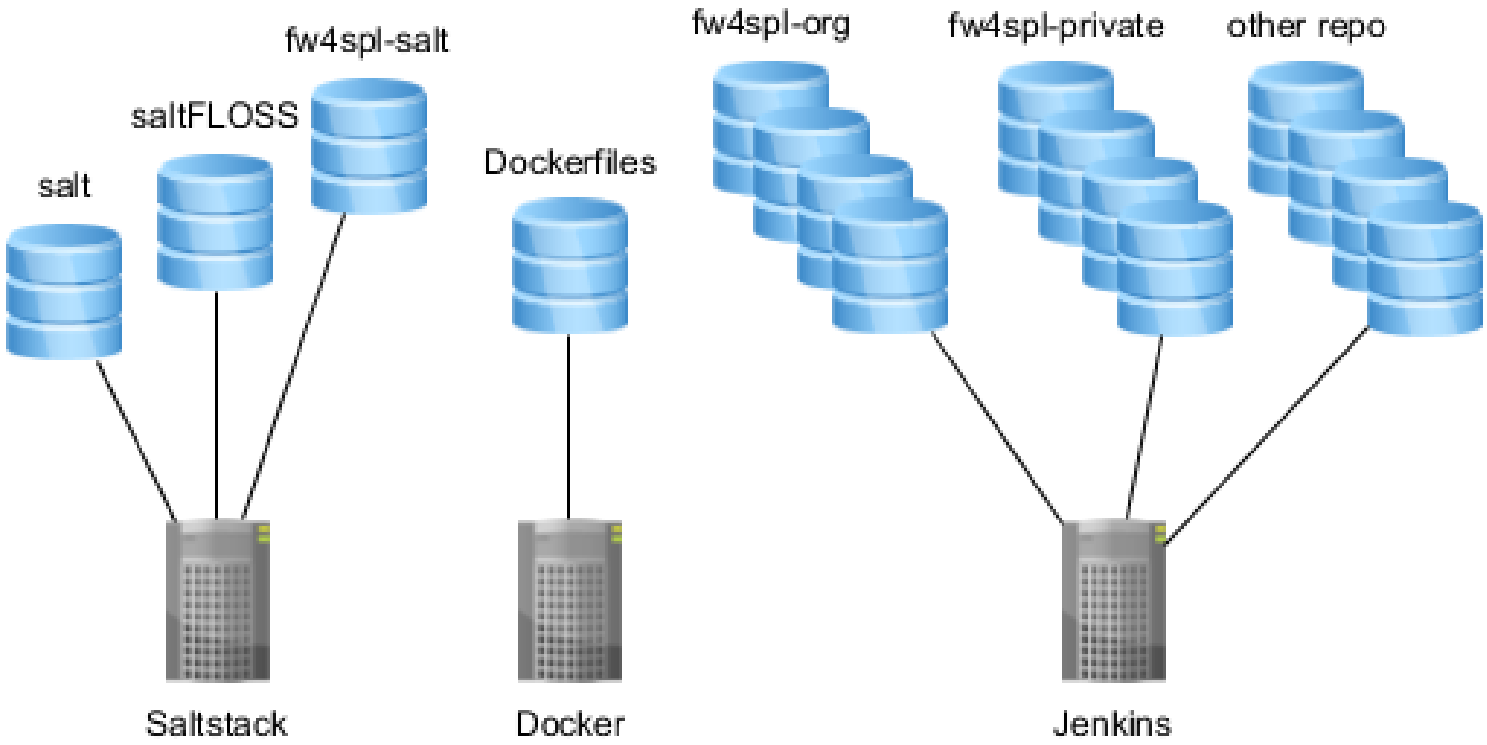
Etape sur les dépôts de sources



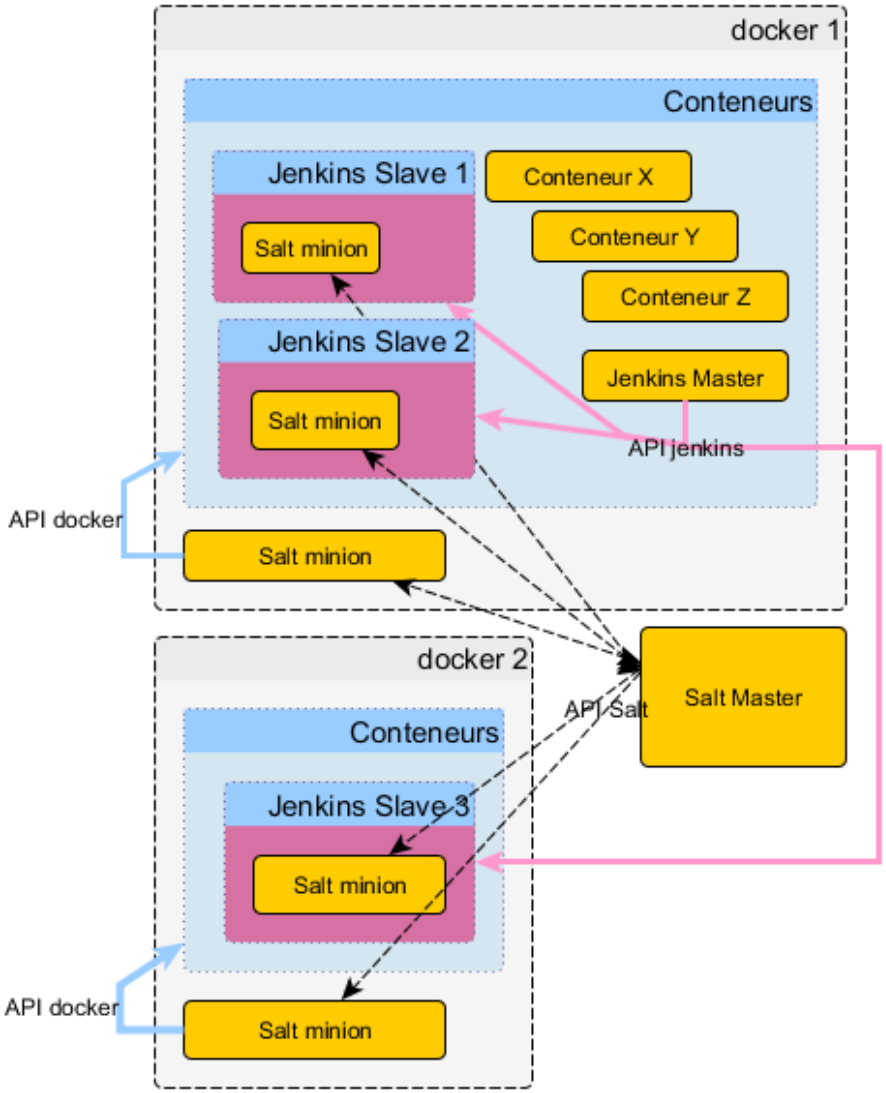
Les outils du processus de build



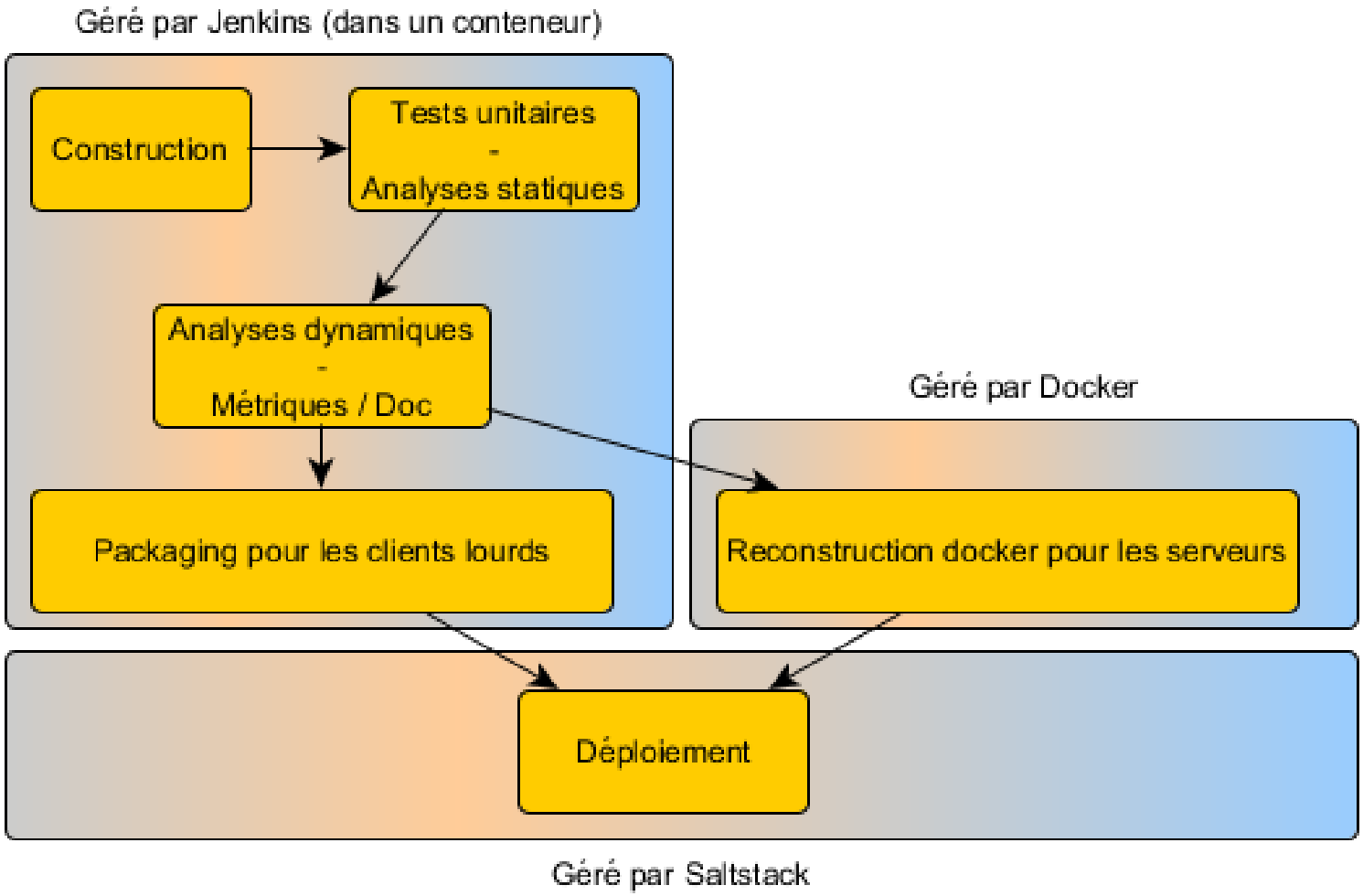
Les dépôts pertinents pour l'orchestration



Architecture globale



Séquence lors d'un push



Conclusion

En l'état :

- Convergence Dev <-> Ops
- Aucun développement local, uniquement de la configuration
- 4 outils : Mercurial/SaltStack/Docker/Jenkins
- Persistence de l'information (tout est en fichiers texte)

Evolutions :

- Supervision dédiée aux conteneurs (actuellement shinken)
- Inventaire automatique des conteneurs dans la CMDB ITIL
- Diffusion et prise en main par plus de personnes
- Registre Docker privé pour les images docker
- Haute-disponibilité des frontend (HAProxy)
- Haute-disponibilité des dockers (Clustering : Docker Swarm)
- Haute-disponibilité de l'infrastructure (OpenStack/CloudStack)