Conteneurs et outils de production de code

https://etherpad.in2p3.fr/p/JDEV2015.T6.GT02

Johan Moreau

IRCAD/IHU

2 juillet 2015

Entre 1000 et 1 million de LOC

Les petits :

- Annuaire de laboratoire en microservices,
- Conteneur de backup,
- Outil de gestion de médicaments pour animaux, ...

Les gros :

- Logiciel de traitement d'image pour la modélisation anatomique
- Logiciel de suivi intra-opératoire de réalité augmentée, ...

Problématiques différentes (performance, distribution, ...) et des choix différents

Gestion de version

VCS :

- CVS, Subversion, ...

DVCS :

- Mercurial, Git, Bazaar, ...

Forge/frontal pour les accès :

- GitHub, Bitbucket, GitLab, Kallithea, ...
- Redmine, Trac, Bugzilla, ...

Quelle organisation de dépôts en fonction de l'outil ?

Quelle communication entre les dépôts ?

Comment faire de la revue de code par rapport au VCS ?

Construction du job

Build :

- CMake, Ant, Maven, NAnt, SCons, Gradle, Gulp, Grunt ...

Complémentaire :

- DOxygen, Javadoc, Sphinx, ...
- XUnit, Checkstyle, Sonar, FindBugs, CppCheck, JDepend, Selenium, ...

A quel point les rapports sont-ils utilisés ?

Intégration continue

Outils :

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

Combien de jobs par "target" ?
Quelle liaison avec la forge ou l'outil de tickets ?
Comment lier cela à la revue de code et à la méthodologie ?

Conteneurs/VM - x86

Emulateurs, VM :

- QEMU, VMWare, HyperV, Xen, KVM, ...

Conteneurs/Isolateurs :

- VServeur, OpenVZ, LXC, Docker, ...

Comment ces outils s'intègrent-il dans la CI ?

Les notes de l'etherpad des participants 1/5

- on peut se servir d'un client git (git-svn)pour ensuite se brancher sur le client subversion => transition plus facile (git over svn) et inversement :
  https ://fr.atlassian.com/git/tutorials/migrating-overview
- Les commandes de base de mercurial sont proches de svn cote utilisation et permettent une migration d'un VCS vers un DVCS
- intégration possible de git ou mercurial dans redmine
- Atlassian : 10€pour moins de 10 développeurs , ça passe à 1200€dès 11 développeurs/produit (information à vérifier suivant le contexte)
- montée de version sous fusionForge pas évidente (perte de données possibles)

Les notes de l'etherpad des participants 2/5

- les forges pour la communication interne sont une erreur selon Johan => il faut des outils de ticketing, intégration continue, etc -> les forges ne semblent plus adaptées
- par contre pour une communication externe cela peut-être approprié (surtout si demande insitutionnelle)
- une architecture hybride est une solution elegante : developpements internes avec des outils efficaces et souples puis depot vers une forge externe 'à la mode'
- conseil : poser ses sources sur plusieurs dépôts, comme ça on n'est pas très impacté en cas de fermeture d'un dépôt

Les notes de l'etherpad des participants 3/5

Questions importantes à se poser :

- Quelle organisation de dépots en fonction de l'outil ?
- Quelle communication entre les dépôts ?
- Comment faire de la revue de code (ou autres usages) par rapport au VCS ?

on peut utiliser mercurial forest pour pointer à partir d'un dépot principal vers des dépôts extérieurs (par exemple des librairies) => git submodule.
les personnes qui travaillent sur plusieurs dépots passent par une autre commande car git submodule ne fonctionne pas très bien

Les notes de l'etherpad des participants 4/5

- Jenkins peut écouter ce qui se passe sur les dépôts
- Questions à se poser :
  - Combien de jobs par target ?
  - Quelle liaison avec la forge ou l'outil de tickets ?
  - Comment lier cela à la revue de code et à la méthodologie ?
- Avec tous les différents dépôts, l'intégration continue peut ne plus "savoir" quoi faire, il faut alors un outil au-dessus pour orchestrer tout ça
- lors de l'utilisation de librairies externes on peut etre confronté à ces problèmes
- conseil de Johan : Github + Travis-ci = très bien pour l'intégration continue publique

L'organisation des equipes se fait au cas par cas. par ex : https ://fr.atlassian.com/git/tutorials/comparing-workflows

Les notes de l'etherpad des participants 5/5

Virtualisation permet d'éviter déjà de démarrer un serveur longtemps alors qu'on en a besoin que ponctuellement. Conteneurs :

- évitent de choisir un OS qui peut ne pas etre supporté par notre infra
- plus rapide que les VM
- consommation de la macchine de dev concentrée sur ce qui nous intéresse et pas à des taches autres (lancer des pilotes, taches bg, etc)
- noyau utilisé par les conteneurs c'est celui de l'OS hote => cela peut affecter les conteneurs. Mais dans la majeure partie des cas on peut réutiliser les conteneurs, et donc on sait que c'est notre modif qui coince s'il y a un souci
- certaines plateformes d'intégration continue permettent d'attacher les conteneurs que l'on a utilisé et de les recréer d'après les fichiers de conf des conteneurs (docker file = du shell)
- si l'IT fait une mise à jour cela peut etre intégré directement dans docker