

Maven - Forge - Intégration continue

Emmanuel Coquery

`emmanuel.coquery@univ-lyon1.fr`

`http://liris.cnrs.fr/~ecoquery`
→ Enseignement

Autour du développement

Au delà du code :

- Tests (unitaires, intégration, fonctionnels)
- Documentation
- Partage des sources
- Suivi de bugs / évolutions
- Qualité du code
- Distribution

→ cycles de développement lourds à gérer

Outils

- Frameworks de tests
- Générateurs de documentation
- Gestionnaires de version
- Gestionnaires de tickets
- Outils d'audit de code
- Scripts, *builders*

Maven

Automatisation du traitement des phases du cycle de vie

- Peut être vu comme un « super Makefile »
 - Java comme langage de script
- Lance l'exécution d'outils :
 - Compilation
 - Test automatisés
 - Archives, Déploiement
 - Génération de documentation
 - ...

Alternatives : CMake, Premake, Grunt, Gulp, etc

Architecture

- Basée sur un système de plugins
 - Un plugin $\leftrightarrow$ un script Java
 - i.e. une classe avec une méthode particulière
 - paramétrable via un morceau de XML
 - Une exécution de maven $\leftrightarrow$ suite d'exécution de plugins
- Nombreux plugins disponibles
 - Pas tous installés au départ
 - Système de téléchargement de plugins à la demande

Phases et cycles de vie

- Une phase regroupe un ensemble de tâches (goals)
 - 1 tâche → 1 plugin
- Un cycle de vie est une suite de phases
 - Le déclenchement d'une phase déclenche les phases précédentes du cycle de vie
- Le cycle de vie dépend du *packaging* (jar,war, ...)
 - *packaging* = type de projet
 - Format d'archive
 - Ordre des phases
 - Affectation tâches → phases
 - Préconfiguration des tâches
 - peut être reconfiguré selon les besoins du projet



Exemple : phases du *packaging* jar

Phase	Tâche(s)
process-resources	<code>resources:resources</code>
compile	<code>compiler:compile</code>
process-test-resources	<code>resources:testResources</code>
test-compile	<code>compiler:testCompile</code>
test	<code>surefire:test</code>
package	<code>jar:jar</code>
install	<code>install:install</code>
deploy	<code>deploy:deploy</code>

Projet maven : organisation des fichiers

- pom.xml ← config. du projet
- src/ ← sources
 - main/ ← à distribuer
 - java/ ← code Java
 - resources/ ← fichiers à distribuer (config appli, images, etc)
 - webapp/ ← ressources web (pour les war : html, jsp, js, images)
 - javacc/ ← grammaire pour générer les *parsers*
 - ...
 - test/ ← uniquement pour les tests
 - java/, resources/, javacc/, etc
- target/ ←
tout ce qui est généré, il est supprimé par clean
il ne faut **pas** le versionner (utiliser .gitignore)

Repository Maven

- Dépôt contenant :
 - Des plugins
 - Des bibliothèques (en général Java)
- Sur le web
 - Téléchargement automatique à la demande
 - Défaut : `http://repo1.maven.org`
 - Miroirs (Nexus, Archiva, etc)
- Local : `/.m2/repository`
 - contient les archives des projets locaux
 - phase `install`
 - cache pour les *repository* web

Classpath et dépendances

Utilisation de libs externes

- Téléchargement
- Gestion des versions
- Transitivité des dépendances
- Configuration du CLASSPATH

Egalement utilisé pour les plugins

Dépendances : *scope*

Scope vs Classpath

	compilation	test	exécution	déploiement
compile	x	x	x	x
provided	x	x	x	
runtime		x	x	x
test		x		

(+ system, import)

Archetypes

- Complexité inhérente aux projets maven
 - Difficultés de mise en œuvre
- Archetype = mini-projet de départ
 - D'un type particulier
 - Préconfiguré
- Exemples
 - maven-archetype-quickstart
 - spring-mvc-jpa-archetype
- En ligne de commande : `mvn archetype:generate`

Intégration dans les EDI

Projets

- IntelliJ : par défaut
- Eclipse :
 - Plugin m2e + connecteurs
 - `mvn eclipse:eclipse`
 - configure un projet Eclipse
 - qui correspond au projet maven
- Netbeans : par défaut

Exécution :

- Intégrée dans tous les EDI (via plugin dans Eclipse)

Forges logicielles

Outil de travail collaboratif pour le développement :

- Espace collaboratif
 - Partage de documents
 - Wiki
 - Dépôt (SVN, Mercurial, Git, etc)
- Gestion des tâches
 - Bug tracking
 - Support, tâches diverses
 - Calendrier, Gantt
 - Suivi via un système de tickets (Issues)

forge.univ-lyon1.fr

- Forge Redmine Gitlab
- Dépôts Mercurial git
- Intégration SI Lyon 1 (LDAP + CAS)
 - Disponible aux étudiants et personnels
 - Utilisable pour les TPs
 - Obligatoire pour le projet transversal (ex-MultiMIF)

Git

- Gestionnaire de versions distribué
 - À la mercurial / darcs / bazaar /etc
- Utilisable
 - En ligne de commande (git)
 - Via une interface dédiée
 - Tortoise git, SourceTree, etc
 - Dans un EDI
 - Intégration Eclipse, Netbeans, IntelliJ, Emacs, etc

Commandes de base

- Création
 - init, clone
- Fichiers
 - add, remove, mv, status
- Versions
 - commit, checkout
- Branches
 - branch, merge, rebase
- Synchronisation de dépôts
 - pull, fetch, push

Scénario simple

① Début du travail

- ① Clone d'un dépôt distant
- ② Modification d'un fichier
- ③ Commit
 - Pour l'instant, pas de modification du dépôt distant
- ④ Push vers le dépôt distant

② Plus tard ...

- ① Pull du dépôt distant
 - ou bien `fetch + rebase`
- ② Update
- ③ GOTO 1.2

Suivi des tâches (gestionnaire de tickets)

- Les tâches ont
 - Une description
 - Un statut : fermé ou ouvert
- Les tâches peuvent avoir :
 - Une échéance
 - Une personne assignée
 - Des étiquettes
 - Une étape (milestone)
- Liens commit (hash 32fb54de)/tâche (numéro 1234) :
 - ref #1234 dans le message de commit
 - ref 32fb54de dans les commentaires de la tâche

Intégration Continue - contexte

Habituellement sur des projets

- de grande taille
- impliquant de nombreuses personnes
- avec des itérations courtes

Technologies actuelles → accessible sur de petits projets

Intégration Continue - principes

- Automatisation des phases du cycle de vie
 - Compilation, test, mise à disposition de binaires
- Institutions de bonnes pratiques
 - Commit réguliers
 - La branche par défaut compile
 - ...
- Surveillance
 - Tableaux de bord, etc

Serveurs d'IC

Permet d'exécuter régulièrement :

- Checkout
- Compilation
- Test
- Audit de code

Pour gitlab :

- peut servir de serveur d'IC ;
- nécessite de gérer et de configurer des « runners » qui exécuteront les tâches.

Alternatives : Jenkins, Travis, etc



SonarQube

Outil d'audit de code

- Analyse exécutée lors du cycle de vie
 - Via e.g. un goal maven
- Fourni des tableaux de bord :
 - Qualité du code
 - Couverture des tests unitaires

Références

- <http://maven.apache.org/>
- <https://git-scm.com/>
- <https://about.gitlab.com/>