Développement > Intégration > Déploiement

Intégration continue

# Qu’est-ce que l’intégration ?

L’intégration est un ensemble de processus, tels que la compilation, l’exécution des tests, la génération de la configuration, la mise à jour de la base de données, la génération de métrique de qualité… permettant

En génie logiciel l’intégration est l’ensemble des processus permettant la mise en commun de différentes versions de code utilisées par le développeur. Tout comme pour les performances serveurs, les développeurs se sont rapidement rendu compte que la « scalabilité verticale »[[1]](#footnote-1) de leurs compétences était limité. Hypothétiquement même si nous nous plaçons dans un univers parfait un développeur ne peut pas être plus productif indéfiniment. Il éteindra forcément, un jour ou l’autre, son seuil maximal de rentabilité. Les développeurs ont alors pensé à travailler en équipe afin d’accroitre non pas leur propre rendement mais plutôt le rendement du projet informatique.

Le premier obstacle rencontré a été de s’organiser fonctionnellement afin de tirer un maximum avantage de la puissance du travail en équipe. Pour cela les développeurs ont découpé leurs projets initiaux en une multitude de tâches (plus ou moins importantes) qu’ils se sont départagés selon les profils de chacun. Nous parlons alors de « scalabilité horizontale »[[2]](#footnote-2).

# Pourquoi dite « continue » ?

Plus l’on met de temps à faire deux intégrations, plus nous allons ajouter des fonctionnalités, corriger des bugs, modifier les fichiers de configurations donc potentiellement l’intégration sera plus compliquée avec le temps. Donc l’idée de « continue » est de faire régulièrement des intégrations pour minimiser les risques.

L’objectif de l’intégration continue est de transformer le processus d’intégration en un non-événement. Avec pour objectif final le « push button », c’est-à-dire que les développeurs n’auront qu’à appuyer sur un bouton (ou taper une ligne de commande) pour compiler/packager, pour déployer, les testeurs pour lancer les scénarios de tests…

# Composition de l’intégration continue

Qu’est ce qui va composer à minima l’intégration continue ? Il nous faut déjà un mécanisme de surveillance du changement - le développeur a changé une valeur dans un fichier de configuration, une icône, une librairie – un mécanisme pour monitorer (surveiller par le biais de métrique) l’environnement, un gestionnaire de version (de code source), les scripts et les outils pour implémenter les processus dont nous avons parlé plus haut (compilation, packaging …) et un mécanisme de notification parce qu’un workflow qui fonctionne tout seul c’est bien mais si nous ne savons pas ce qu’il fait c’est déjà beaucoup moins intéressant.

# Le cloud

Le cloud nous permet de mettre à disposition rapidement une VM en temps réel.

Jenkins

Serveur d’intégration continue. Jenkins peut effectuer des builds distribués (100 esclaves) et offre un mécanisme de plugins. Jenkins est un produit open source, nous pouvons créer nos propres plugins afin de répondre aux formats de packaging spécifique à notre entreprise.  Ce mécanisme de plugin nous permet d’envisager l’intégration continue sous un angle différent et voir l’intégration continue en tant que fonction.

# Une fonction ?

Nous pouvons imaginer que l’intégration va prendre un objet source (pas nécessairement du code) et nous renvoyer un artefact. Ce qui est important c’est que la fonction n’ait pas d’effet de bord. Mais cette fonction peut-être très couteuse, donc l’idéal est de travailler en asynchrone. Nous allons donc avoir une première phase qui va packager le livrable, et dans un second temps, une fois le livrable packagé, nous pouvons commencer les tests sur le livrable et en parallèle effectuer le calcul des métriques qualités qui pourront-elles mêmes être découpées.

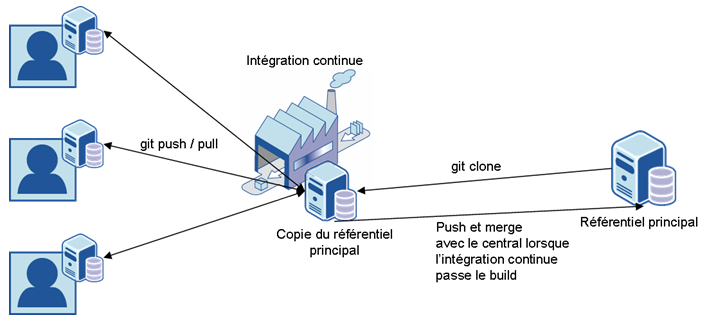
L’objet source de notre intégration continue sera le commit. Il est le candidat naturel comme entrée de la fonction car :

- il détermine le contenue d’une arborescence source,

- il permet une copie simple et rapide d’une machine à l’autre, du fait qu’il ne récupère que le différentiel d’une copie à l’autre. Nous imaginons pour des builds distribuées, même si nous avons notre repository central qui a changé, la récupération par l’ensemble des machines esclaves va être très rapide.

Nous allons utiliser l’intégration continue en tant que Revue de Code automatique. Nous allons avoir une structure avec un repository central partagé (master), un repository local pour chaque chaque développeur (sur son poste) et entre les deux en repository dédié à Jenkins.

Le développeur code, fait un commit – le commit n’est qu’en local chez le développeur – et décide de pousser son commit sur le repository de Jenkins. Jenkins lance alors les différentes tâches qui lui sont allouées et si tous les tests passent Jenkins va pousser le commit sur le repository central. Ce qui nous assure un repository central toujours validé.



1. Possibilité d’augmenter les ressources physiques d’un composant informatiques afin d’augmenter ses performances. [↑](#footnote-ref-1)
2. Possibilité d’ajouter des composants à une entité informatique afin d’augmenter ses performances. [↑](#footnote-ref-2)