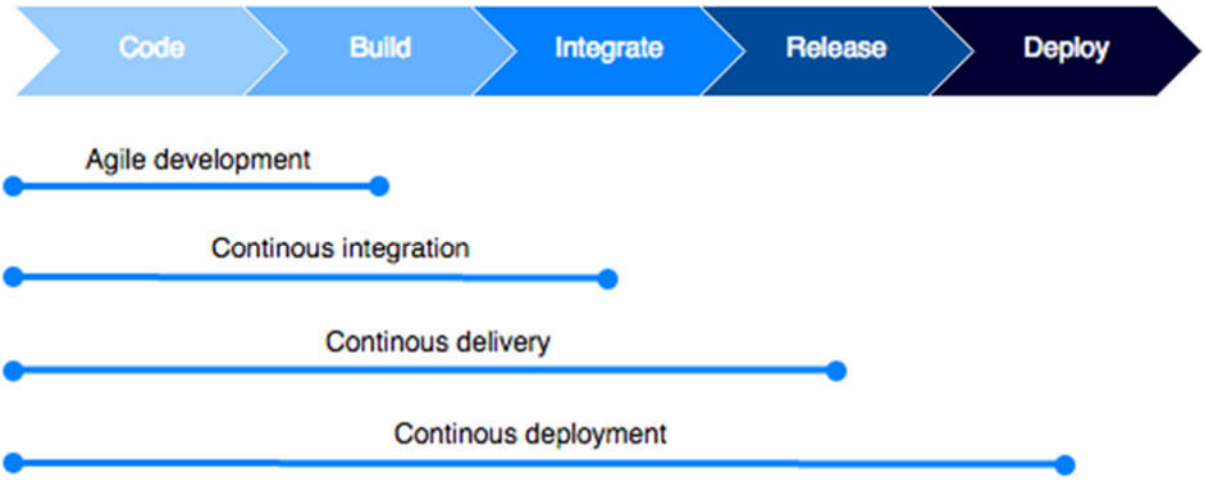
Intégration continue



L'intégration continue est une pratique prévue pour accélérer la mise en production des changements effectués dans un logiciel.

Avec une pratique traditionnelle, on attendrait la création d'une nouvelle version du logiciel, qui regrouperait de nombreuses modifications. La publication d'une version étant une opération lourde, les versions sont forcément espacées. A l'inverse, avec l'intégration continue, une simple modification est immédiatement propagée et on obtient un feedback immédiat sur celle-ci.

La modification est propagée par étapes dans un flux nommé « pipeline » jusqu'à une destination plus ou moins avancée selon l'approche CI (intégration continue) ou CD (livraison continue ou déploiement continu).

Au minimum dans une méthode agile de développement, on met en commun et on compile le code source complet du logiciel sur un serveur partagé.

Avec l'intégration continue, on fait de plus passer automatiquement des tests de vérifications à chaque « commit » sur ce serveur partagé. Cette pratique est fortement associée à la couverture du code par des tests automatisés (comme JUnit) et à des approches comme TDD (développement piloté par les tests). Si un test ne passe pas, le « build » est refusé et on ne va pas plus loin dans le pipeline. Ces tests doivent tourner rapidement pour permettre un feedback rapide (10 minutes par exemple) au développeur.

Mais la mise en production d'un logiciel doit passer par plusieurs étapes correspondant à plusieurs niveaux de tests (tests unitaires, tests d'intégration, tests d'acceptation, tests de performance, …). Avec la livraison continue, le logiciel passe par ces différentes étapes et il est mis en exploitation dans un environnement de tests. Mais la dernière étape de mise en production est encore manuelle. Elle devient automatique avec le déploiement continu.

L'objectif de ce guide est de vous permettre de mettre en place la première étape de l'intégration continue avec des tests unitaires qui seront exécutés sur un serveur partagé.

Guidelines Travis CI

Travis Ci est un service distribué pour l'intégration continue utilisé principalement pour faire des build et tester les projets logiciels se trouvant sur GitHub.

Afin de mettre en place Travis, il vous faut :

1) Avoir un compte GitHub.
2) Vous rendre sur https://travis-ci.com/ (n'allez pas sur https://travis-co.org) et authentifiez-vous avec votre compte GitHub.
3) Allez sur https://travis-ci.com/getting_started et suivre les quelques instructions. Un fichier « *.travis.yml* » vous est fourni et doit être placé à la racine de votre projet. Il devrait ressembler à :

```
language: java

cache:
  directories:
    - $HOME/.m2

script:
  - echo "TestBot at work..."
  - mvn clean test
```

4) S'il n'existe pas, vous devrez également placer à la racine du projet un fichier « *pom.xml* » parent qui indiquera les différents modules contenus dans le projet ou en tout cas ceux que vous désirez tester.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId><nom_du_packaging></groupId>
    <artifactId><nom_du_projet></artifactId>
    <packaging>pom</packaging>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

    <modules>
        <module><nom_du_module_1></module>
        <module><nom_du_module_2></module>
    </modules>
</project>
```

Suivant votre configuration, vous devrez peut-être modifier les versions données dans l'exemple.

Ainsi, lorsque vous pusher de nouveaux commits sur GitHub, Travis lancera une compilation du projet puis lancera les tests unitaires que vous aurez créés. Différentes icônes vous informeront du statut de l'opération sur GitHub ou Travis :

- Orange : build en cours
- Rouge : Certains ou tous les tests ont échoués (fail)
- Vert : Tout est ok

À vous de vérifier si vos commit passent.

GitHub – protection avec les statuts de Travis CI

Sur GitHub vous avez également la possibilité d'appliquer des règles sur vos branches ou une branche en particulier. Par exemple, dans le cas où les tests sur travis ont échoué (fail), il ne devrait pas être possible d'effectuer le merge via une pull request sur la branche master.

Attention : Cette option de configuration n'est disponible que par la personne ayant créé le projet sur la plateforme.

1) Rendez-vous sur le repo de votre projet.
2) Dans l'onglet *Settings* -> *Branches*, cliquez sur le bouton « *Add rule »*
3) Tapez le nom de la branche sur laquelle la règle doit s'appliquer (par exemple *master* pour votre branche par défaut) ou à une série de branches (ajouter un « * » à la fin du nom. Par exemple : *<nom_commun_au_branche>-**).
4) Cocher l'option "Require status checks to pass before merging" ainsi que les options qui s'affichent dessous.
5) Créer la règle en appuyant sur le bouton « Create ».

Maintenant, si vous créez une pull-request pour faire un merge sur une des branches spécifiées (par exemple sur la branche *master*) et que les tests Travis ont échoués. Le bouton qui permet le merge est grisé (sauf pour l'administrateur).

Bonnes pratiques

1) Utiliser des branches pour développer de nouvelles fonctionnalités.
2) Créer des branches en partant de branches qui fonctionnent.
3) Ne faites des merge sur la branche master que si les tests passent et que le code a été vérifié (manuellement ou automatiquement).