

Introduction à l'intégration continue

AUTOM

Pré-requis du module

- Connaitre Shell et Linux
- Savoir utiliser une machine virtuelle
- Connaitre méthode agile



Objectifs du module

- Présentation de l'intégration continue
- Prise en main de Jenkins
 - Savoir comment se configure Jenkins (authentification, plugins)
 - Savoir créer ou dupliquer des jobs et les ordonnancer
 - Savoir configurer des jobs
 - Ajout de tâches Maven (run tests Squash TA)
 - Ajout de tâches de script
 - Lien avec un gestionnaire de source
 - Lien avec un gestionnaire d'artefact
- Découverte de Maven
 - Principe (pom.xml)
 - Commandes
- Découverte de SVN
 - Principe (branches, commits, tags)
 - Opérations basiques (tirer, commiter, tagger, créer une nouvelle branche)
- Découverte de Nexus
 - Principe
 - Ajouter des dépendances



Introduction : Présentation de l'intégration continue (qualité logicielle)



- **L'intégration continue** est un ensemble de pratiques utilisées en génie logiciel consistant à vérifier à chaque modification de code source que le résultat des modifications ne produit pas de régression dans l'application développée.
- L'intégration continue est une organisation de travail
- L'intégration continue de test permet au testeur de vérifier la qualité d'une application (qualimétrie, tests de non-régression...) et le respect des normes et bonnes pratiques dans le code (audit de code, qualimétrie, ...)



- Retour sur les différents cycles utilisés pour le développement applicatif
- Présentation de l'intégration continue
- Légende utilisée dans la description des schémas ci-dessous

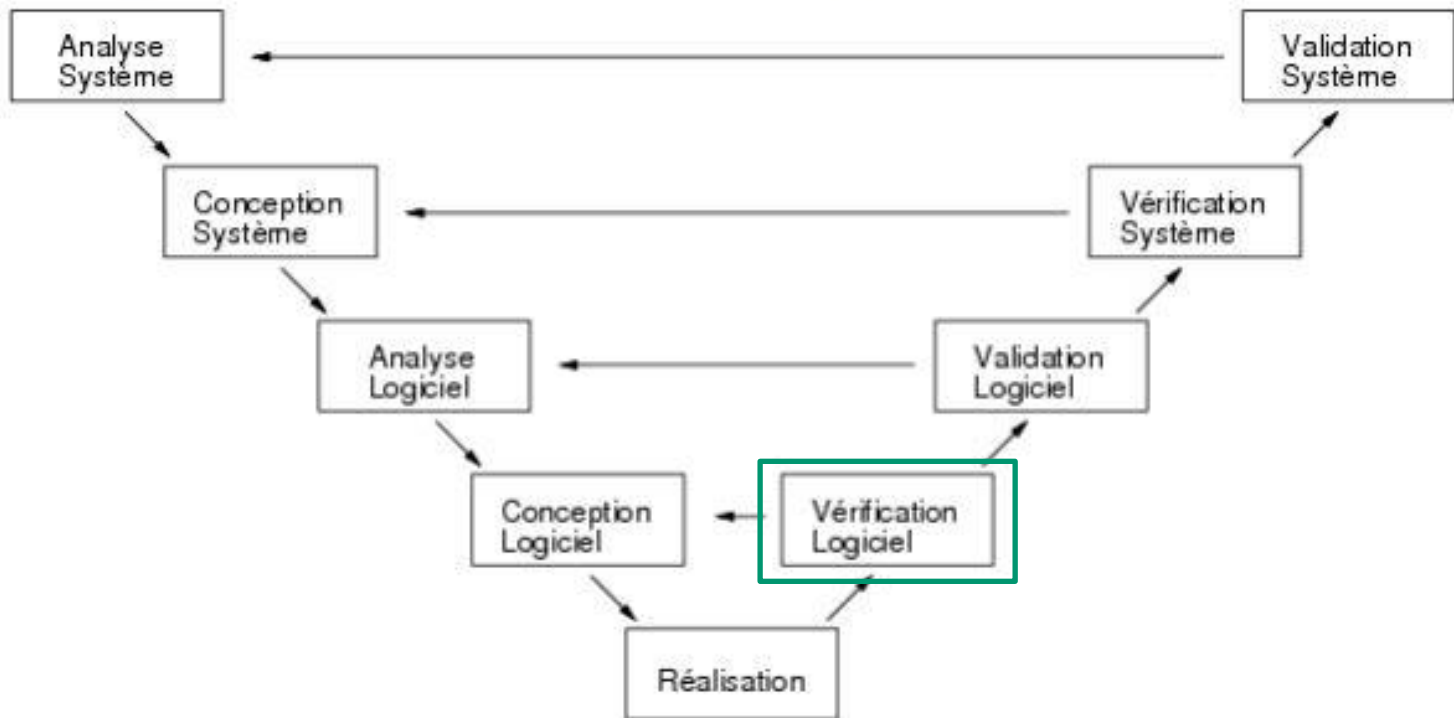


Présence des testeurs dans le processus de création d'une application



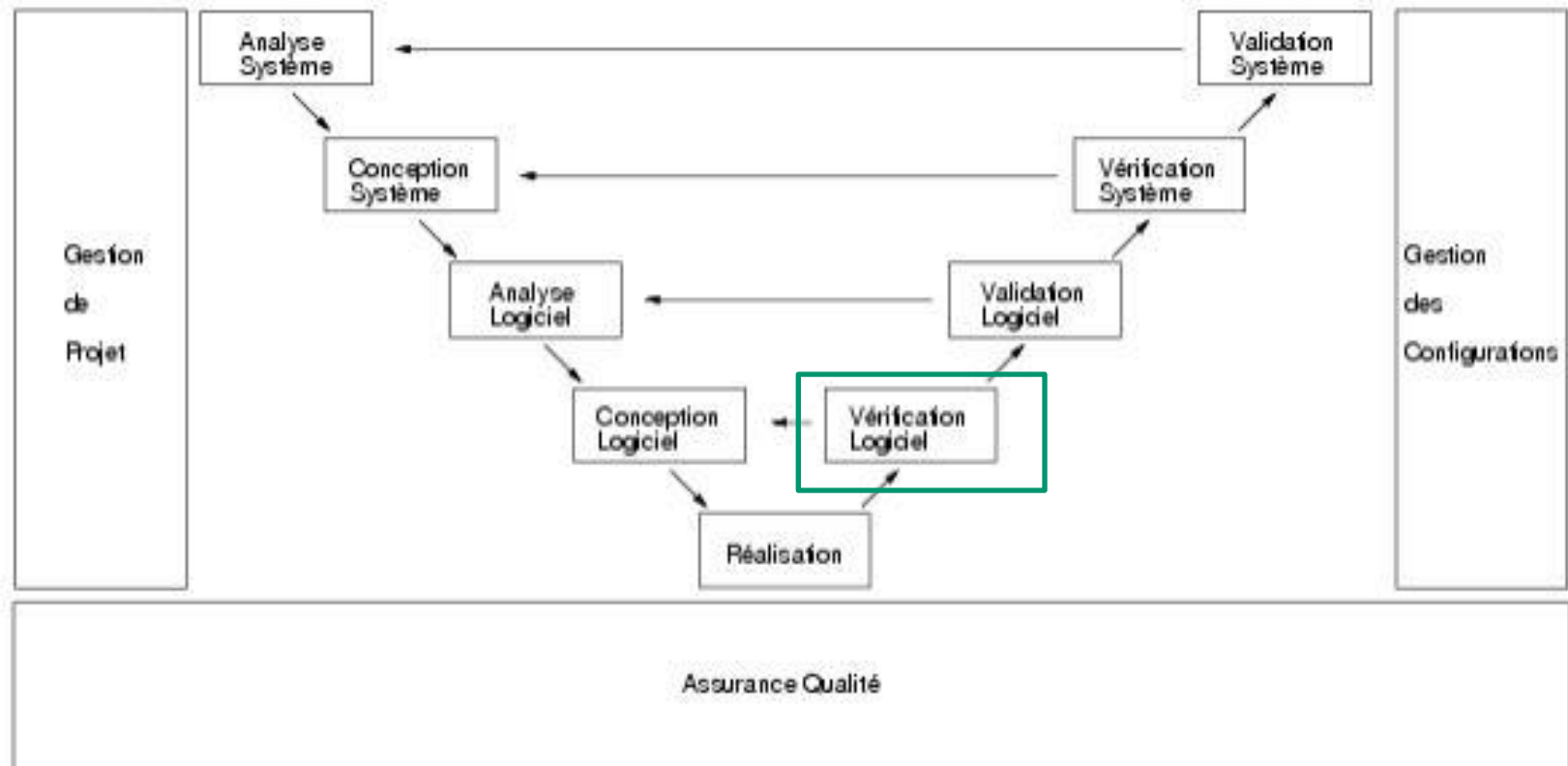
- Cycle en V

Cycles de développement en V

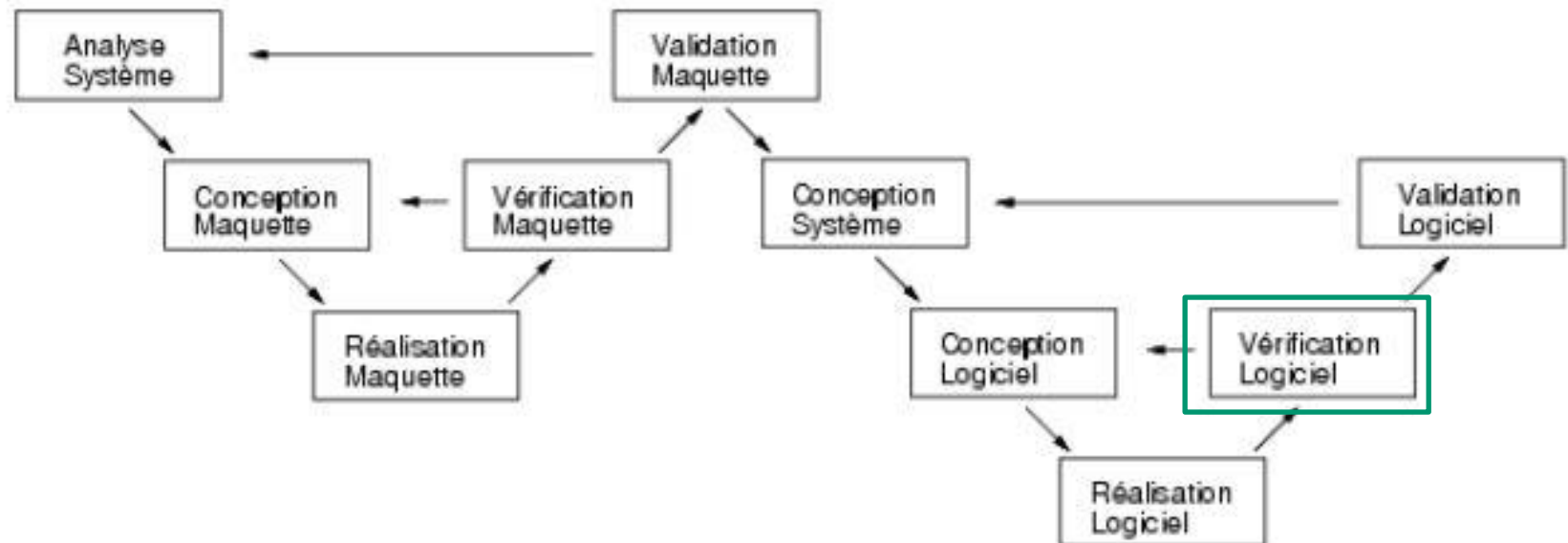


- Cycle en M

Cycles de développement en M



- # Cycles de développement en W



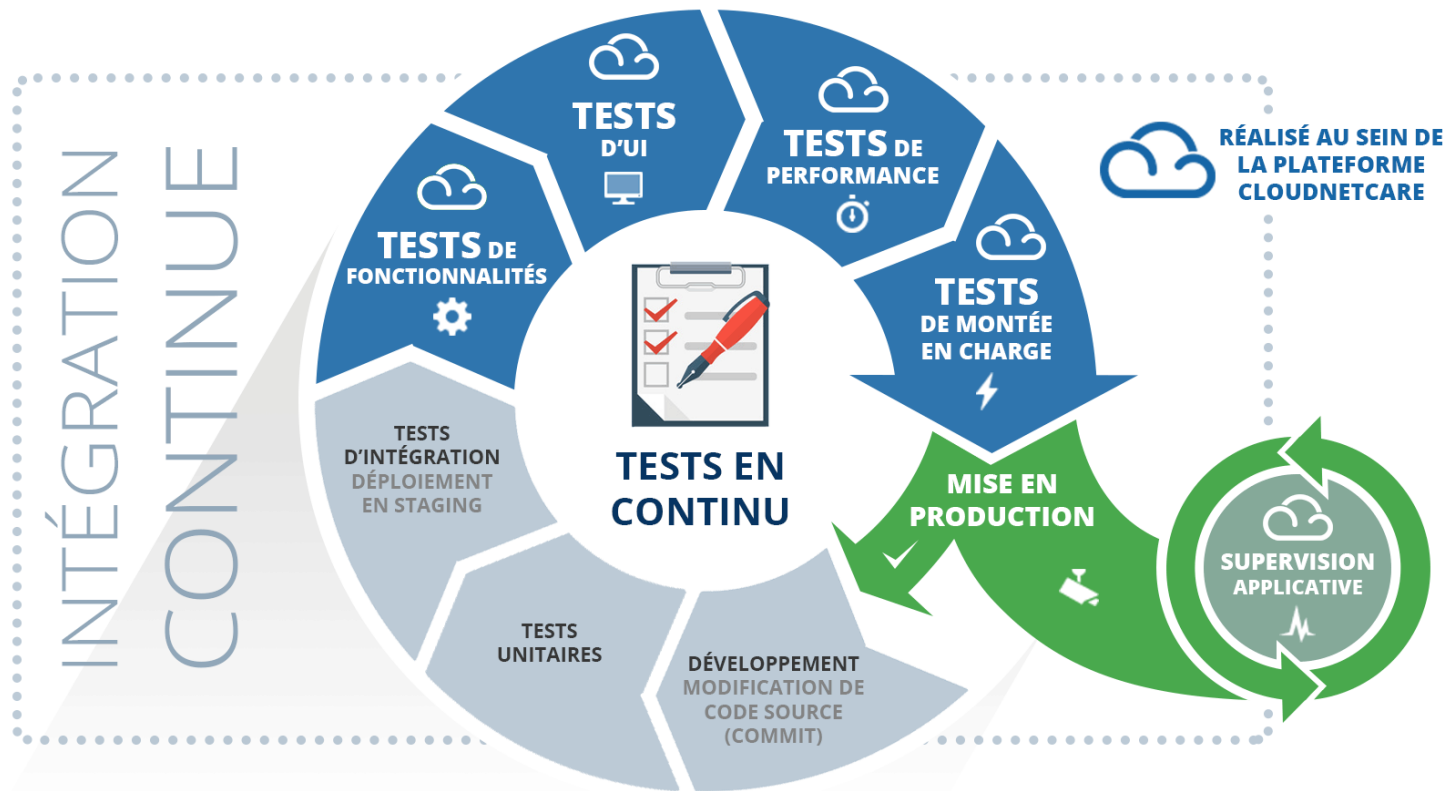
- Limites des cycles traditionnels :
 - Travail souvent dispersé, long.
 - Source de conflits au sein des équipes (MOA, MOE, TRA...)
 - Reprise de code(s) ancien(s), parfois obsolète(s)
 - Prise de retard dans les plannings



Introduction à l'intégration continue

Introduction : l'intégration continue (1/5)

- L'intégration continue permet de repenser tout le processus de développement d'une application.
- Les testeurs sont présents à tous les niveaux du processus



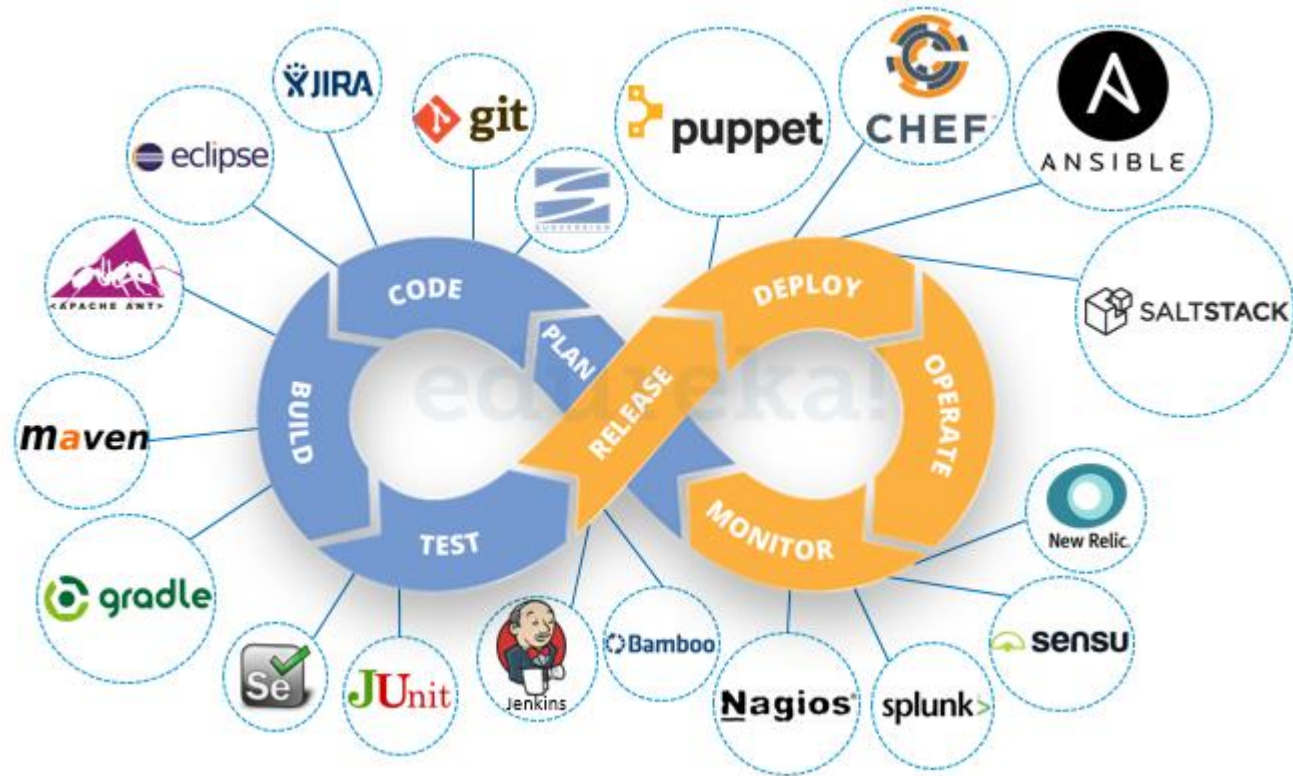
- L'intégration continue dans sa forme simple :
 - Se compose d'outils qui surveillent les modifications de code dans le gestionnaire de configuration (Jenkins et ses plugins)
 - Dès qu'un changement est détecté, cet outil compile (Maven) et teste l'application (Squash TA et autres).
 - Si la moindre erreur arrive alors l'outil alerte les développeurs afin qu'ils puissent tout de suite corriger le problème.
- L'intégration continue dans sa forme complexe :
 - L'Intégration Continue peut aussi suivre la santé de votre projet en surveillant la qualité du code et les métriques de couverture et ainsi aider à maintenir la dette technique à un niveau bas et à abaisser les coûts de maintenance (qualimétrie, Sonar)
 - Créer un reporting public (client, développeurs, MOA & AMOA)
 - Facilite la communication entre les acteurs
 - Indique un état clair du développement d'une application
 - Permet le déploiement automatique d'une application
 - Permet de tester la non régression d'une application

- Synthèse avantages de l'intégration continue :
 - Travail d'équipe et réactivité
 - Assurer la qualité du code au plus tôt
 - Assurer la complétude des livrables (intégrité des livrables)
 - Rend la production d'une application plus « fluide »
 - Permet de générer du reporting de qualité
 - Mise en place d'une méthode de travail DevOps



Introduction à l'intégration continue

Introduction : l'intégration continue (4/5)



Source : edureka!

- La conduite du changement, une étape à prendre en compte!
- Place de l'intégration continue dans les cycles de développement V, M, W?



Introduction à l'intégration continue

Introduction : L'intégration continue dans l'entreprise

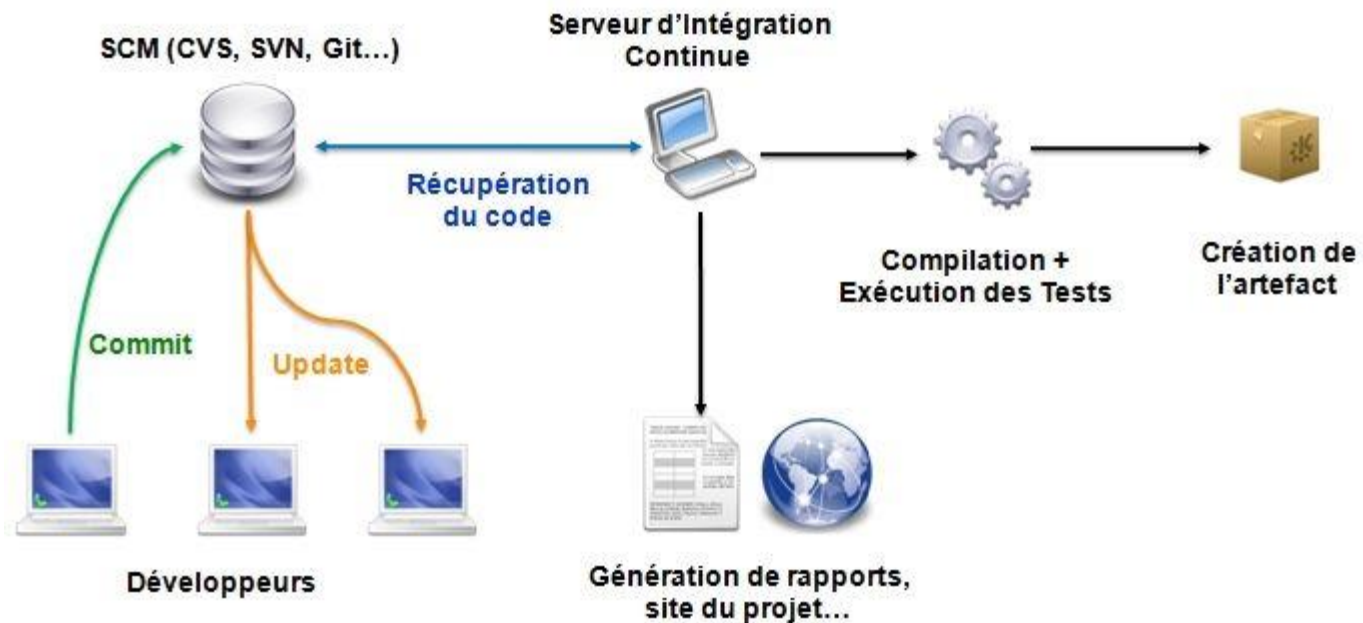
- **Phase 1** → Pas de serveur de build
- **Phase 2** → Builds quotidiens
- **Phase 3** → Builds quotidiens et tests automatisés basiques
- **Phase 4** → Arrivée des métriques
- **Phase 5** → Les tests, une phase longtemps moquée
- **Phase 6** → Tests d'acceptation
- **Phase 7** → Le déploiement continue



Introduction à l'intégration continue

Introduction : Phase 1 → Pas de serveur de build

- **Phase 1** → Pas de serveur de build → à mettre en place (développeur)
 - Il s'agit principalement du travail du développeur
 - Le testeur peut participer à la création d'un serveur de build



- **Phase 2 → Builds quotidiens**
 - Le serveur de build existe
 - Les développeurs « commit » le code tous les jours (dans le meilleur des cas)
 - Le code est « buildé » et testé (de préférence la nuit)



Introduction à l'intégration continue

Introduction : Phase 3 → Builds quotidiens et tests automatisés basiques

- **Phase 3 → Builds quotidiens et tests automatisés basiques**
 - Dès qu'un nouveau code est « commité », des tests automatisés sont exécutés (non régression).
 - Exécution également de tests unitaires (développeurs)
 - Les erreurs sont remontées immédiatement
 - Les développeurs corrigent le code
 - Le client suit en temps réel l'avancée d'amélioration de la qualité du code
 - Suivi reporting par email et tableau de bord



- **Phase 4 → Arrivée des métriques**
 - Des métriques de qualité et de couverture de code sont maintenant mesurées pour aider à évaluer la qualité du code (qualimétrie)
 - Le build de métrique de qualité du code génère automatique l'API de l'application (API Application Programming Interface)
 - Oblige un respect des bonnes pratiques (dans la mesure du possible)
 - Montage d'un tableau de bord de suivi de qualimétrie (voir partie qualimétrie, module 2)



- **Phase 5 → Les tests, une phase longtemps moquée**
 - Très longtemps les tests n'ont pas été pris au sérieux.
 - Dans l'intégration continue, les tests font maintenant partie du cycle de vie
 - Différents types de tests sont utilisés et améliorent la confiance du client
 - ... améliorent aussi (et surtout) la qualité de l'application livrée
 - Le nouveau processus peut être développé → testé → déployé
 - Le déploiement permet de réaliser des tests de bout en bout, des tests de stress et des tests de performances.



- **Phase 6 → Tests d'acceptation automatisés (fitness)**
 - Signifie s'assurer que l'application est conforme aux spécifications techniques et fonctionnelles
 - Tests de validation
 - Acceptation fonctionnelle automatique et manuelle
 - Acceptation technique : tests unitaires, et tests d'intégration
 - Tests alpha et beta



Introduction à l'intégration continue

Introduction : Phase 7 → Le déploiement continu (1/2)

- **Phase 7 → Le déploiement continu**
 - Mise en production automatique de l'application
 - Confiance importante dans la qualité du code livré
- Distinguer livraison continue et déploiement continu :
- **Déploiement continu :**
 - Un déploiement continu correspond à la livraison d'une version applicative testée.
 - Ce processus est automatique et sous la responsabilité des techniques
 - Les utilisateurs ont accès immédiatement à la nouvelle version livrée
- **Livraison continue :**
 - Une livraison continue correspond à la livraison d'une version applicative testée.
 - Cependant, c'est le métier et non le technique qui décide du moment où les utilisateurs auront accès à la nouvelle version livrée



Partie 1 : Présentation des outils (non exhaustive)



- Présentation sommaire des outils que nous utiliserons :
- **Jenkins** : création et gestion de jobs pour l'intégration continue. Fonctionne par exemple sur un serveur d'application Tomcat
- **Maven** : outil pour la gestion et l'automatisation de production des projets logiciels Java (semblable à l'outil ant)
- **SVN et/ou GIT** : logiciel de gestion de version
- **Nexus** : gestionnaire de dépôts open source

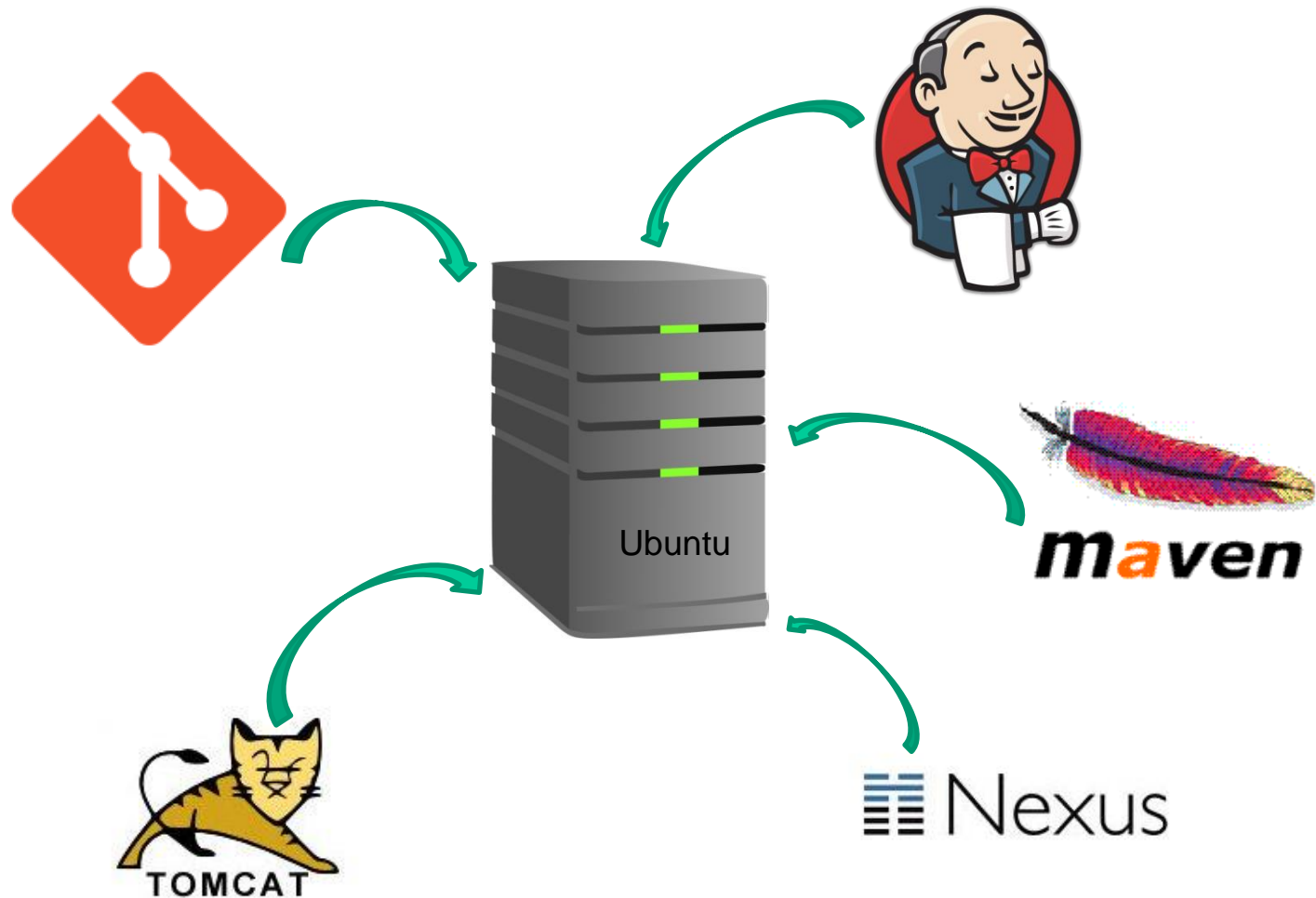


Introduction à l'intégration continue

Présentation des outils : Jenkins vs Hudson



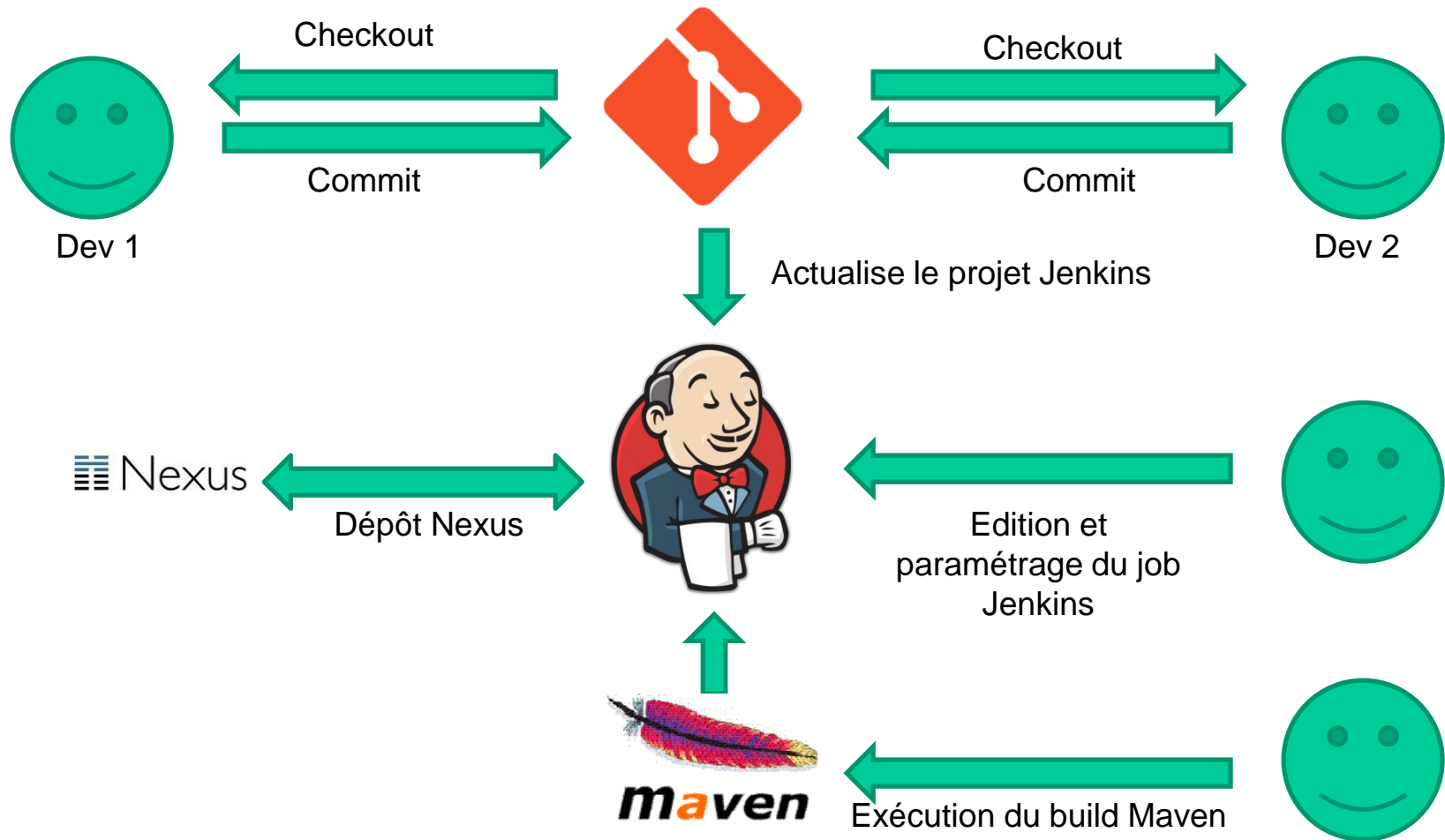
- Contenu de la machine Ubuntu



Introduction à l'intégration continue

Présentation des outils : schéma de fonctionnement

- Processus simplifié (la complexité varie selon le contexte)



Partie 2 : Installation de Jenkins



- **Pourquoi utiliser Jenkins?**

- Permet de détecter au plus tôt les anomalies d'intégration
- Ces anomalies sont corrigées progressivement
- Les vérifications automatisées sous Jenkins permettent de revérifier la complétude d'une livraison à chaque fois
- Permet de connaître et suivre précisément les versions des livraisons

- **Les pré-requis nécessaires**

- Le code source doit être partagé entre les développeurs et les testeurs (svn, git...)
- Les développeurs doivent pousser régulièrement leur code pour le suivi
- Des tests automatisés doivent être mise à disposition

- **Principe de fonctionnement de Jenkins**

- Exécuter un job (un build → par exemple avec Maven) qui déroule l'ensemble des tests automatisés sur le code source livré par les développeurs.



- Les testeurs et Jenkins : cas d'utilisation
- Démonstration d'exécution d'un build maven sous Jenkins
- Rappel :
 - La bonne pratique consiste à installer chaque outil sur une plateforme virtuelle dédiée, avec un utilisateur et un groupe dédié.
 - Dans le cadre de nos exercices, tous les outils seront installés localement sur une seule et même machine virtuelle et Windows.



- **Exercice** : La version précédemment installée est la 2.45 de Jenkins.
 - Utiliser le jenkins.war 2.46 et mettre à jour votre Jenkins. Vous pourrez vérifier la version avant et après mise à jour depuis le GUI Jenkins (15 minutes)
 - Info : penser à vérifier les droits (utilisateur/groupe)
- **Exercice** : Installation de Squash TF Serveur sur Windows
Télécharger Squash TF serveur pour Windows
 - Installer Squash TF sur votre poste Windows
 - Démarrer le et vérifier son bon fonctionnement (voir les logs)



- **Rappel général sur l'arborescence de Tomcat :**
 - Emplacement de Tomcat : selon les cas 😊
 - bin : contient les scripts Tomcat
 - catalina.sh → démarre, stop, indique si Tomcat tourne
 - daemon.sh → démarre Tomcat comme service
 - configtest.sh → vérifier la configuration système de Tomcat
 - startup.sh → démarre Tomcat
 - shutdown.sh → stop Tomcat
 - version.sh → indique la version de Tomcat
 - conf : configuration de Tomcat
 - catalina.properties → configuration de Tomcat
 - logging.properties → log4j
 - serveur.xml → configuration de votre serveur Tomcat



- **Rappel général sur l'arborescence de Tomcat :**
 - lib : contient les librairie Tomcat (ne pas toucher)
 - LICENSE : la licence Tomcat
 - logs : contient les logs Tomcat. Pour rappel, catalina.out contient les logs de lancement. Voir détail
 - NOTICE : xsd utilisées par Tomcat
 - RELEASE-NOTES : mises à jour
 - RUNNING.txt configuration pour installation Tomcat
 - temp : répertoire pour les fichiers temporaires de Tomcat
 - webapps : répertoire déploiement application
- **Exercice 5 (10 minutes) :** depuis la configuration, modifier le port Tomcat par 8082



- **Eteindre Tomcat**
 - Dans bin, lancer la commande `./shutdown.sh` puis vérifier que Tomcat est coupé `ps aux | grep tomcat`
- **Copier le fichier jenkins.war dans le répertoire webapps**
- **Lancer Tomcat et vérifier ses logs**
 - Dans bin, lancer la commande `./startup.sh && tail -f ../logs/catalina.out`
- **Vérifier que Tomcat tourne**
 - `ps aux | grep tomcat`
- **Accéder à Jenkins**
 - Dans un navigateur Web : `http://{ip_machine}:{port}/jenkins`



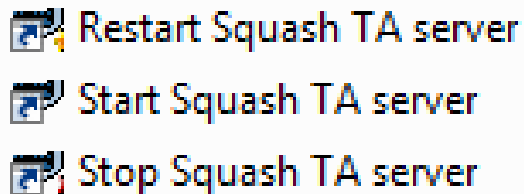
- Une fois Tomcat démarré, tout le contenu se trouve dans TOMCAT_HOME/.jenkins
- Contenu du répertoire .jenkins : détails vu ensemble
- Le répertoire .jenkins/workspace contient les jobs créés depuis Jenkins, nous y reviendrons avec précision!



Installation et paramétrage des outils

Installation de Jenkins : Présentation de Squash TA serveur

- Squash TA server intègre un serveur d'application Tomcat qui héberge Jenkins
- Vous trouverez dans le Squash TA server tous les paramètres de lancement nécessaires pour le bon fonctionnement de build Maven Squash TA
- Squash TA server intègre une JDK
- Le fonctionnement et le paramétrage est identique à nos connaissances Tomcat Linux.
- Les raccourcis :



Restart Squash TA server
Start Squash TA server
Stop Squash TA server



Partie 2 : Présentation Maven



- Historique
 - Projet Apache
 - Versions maintenues : Maven 2 et Maven 3
- Principes
 - Gestion de build de projet (comme Ant , Gradle...)
 - Déclaration de build (basée sur le fichier pom.xml)
 - Métadonnées complètes sur le projet
 - Dans le test, très utilisé pour l'exécution de tests automatisés

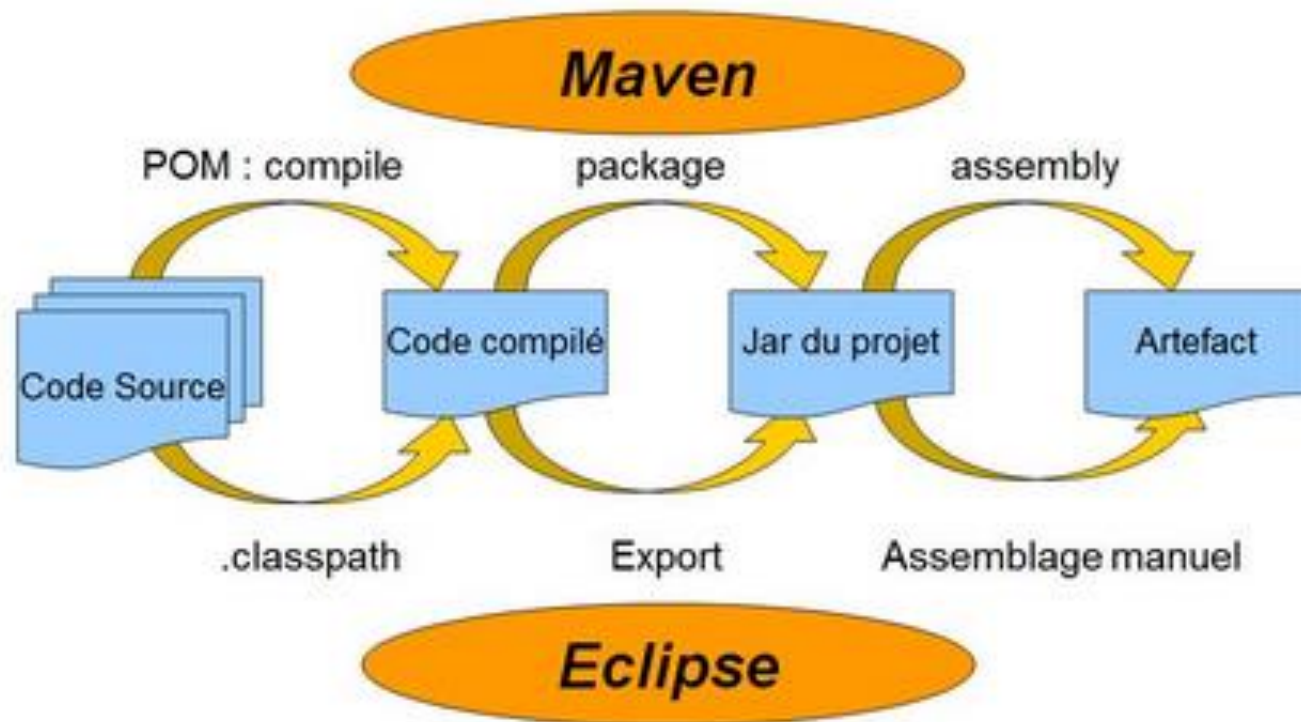


- Fonctionnalités :
 - construction , compilation
 - documentation
 - rapport
 - gestion des dépendances
 - gestion des sources
 - mise à jour de projet
 - déploiement



Installation et paramétrage des outils

Présentation Maven : Maven dans Eclipse



Source : developpez.com



- Autre exemple :

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://maven.apache.org/POM/4.0.0" 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>
  <parent>
    <groupId>fr.afcepf</groupId>
    <artifactId>Yparent</artifactId>
    <version>0.0.1-SNAPSHOT</version>
  </parent>
  <artifactId>YWSRest</artifactId>
  <packaging>war</packaging>

  <dependencies>
    <dependency>
      <groupId>javax</groupId>
      <artifactId>javaee-api</artifactId>
    </dependency>
    <dependency>
      <groupId>org.jboss.resteasy</groupId>
      <artifactId>resteasy-jaxrs</artifactId>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>${project.groupId}</groupId>
      <artifactId>YApiBusiness</artifactId>
      <version>${project.version}</version>
    </dependency>
  </dependencies>
</project>
```



- pom.xml :
 - pom : project object model
 - Est un fichier xml
 - Il décrit le projet et sa mise en place
- Les balises :
 - **project** : c'est la balise racine de tous les fichiers pom.xml
 - **pom.xml.modelVersion** : cette balise indique la version de POM utilisée (obligatoire)
 - **groupId** : cette balise permet d'identifier un groupe qui a créé le projet (fonction d'indexation du projet)
 - **artifactId** : nom des artefacts à construire
 - **packaging** : type de packaging du projet (ex. : JAR, WAR, EAR, etc.).
 - **version** : version de l'artefact généré par le projet.
 - **name** : nom du projet.
 - **url** : adresse du site du projet.
 - **description** : description du projet.
 - **dependencies** : balise permettant de gérer les dépendances.
 - Plugins → gère les plugins
 - D'autres balises peuvent être utilisées selon les projets

- **Archetype** : un archetype est un template de projet. Permet de respecter les règles de bonnes pratiques.
- **Dépendances** : Une dépendance est un lien vers un artefact spécifique contenu dans un repository.
- **Artefact** : dans Maven, un artefact est un élément spécifique issu de la construction du logiciel (exemple : fichiers .jar, .ear, .zip, .war...)
- **Artifact : groupeid** : identifiant du groupe à l'origine du projet.
- **Artifact : artifactid** : l'artifactid est l'identifiant du projet au sein de ce groupe. Il est souvent utilisé comme nom final du projet.

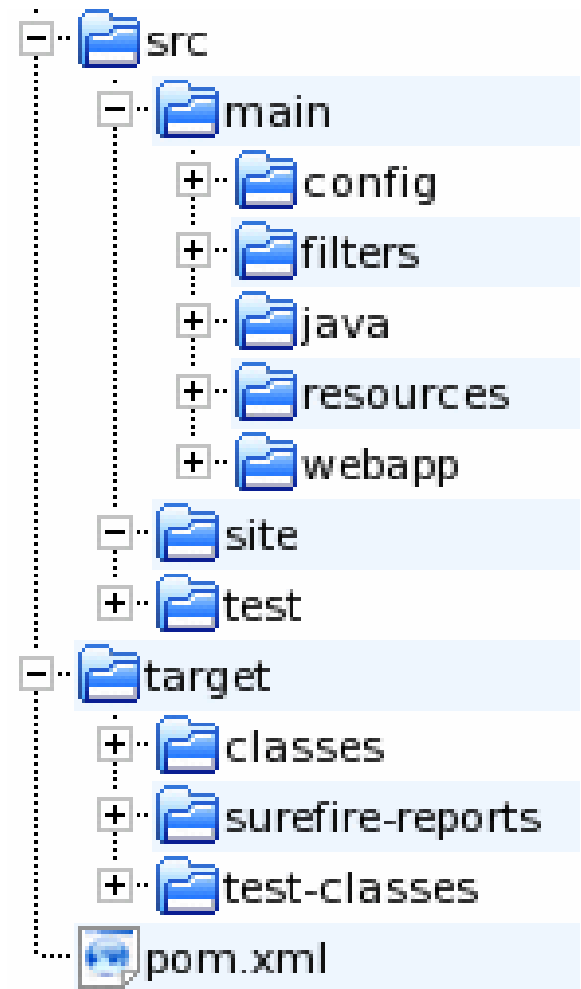


Installation et paramétrage des outils

Présentation Maven : principales commandes Maven

Structure d'un projet Maven

- **src/main/java:** Votre code java va ici (étonnamment)
- **src/main/resources:** Les autres ressources dont votre application a besoin
- **src/main/filters:** Les filtres de ressources, sous forme de fichier de propriétés, qui peuvent être utilisés pour définir des variables connues uniquement au moment du build.
- **src/main/config:** Les fichiers de configuration
- **src/main/webapp:** Le répertoire d'application web pour les projets WAR.
- **src/test/java:** Les tests unitaires
- **src/test/resources:** Les ressources nécessaires aux tests unitaires, qui ne seront pas déployées
- **src/test/filters:** Les filtres nécessaires aux tests unitaires, qui ne seront pas déployées
- **src/site:** Les fichiers utilisés pour générer le site web du projet Maven



Source : developpez.com

Installation et paramétrage des outils

Présentation Maven : arborescence

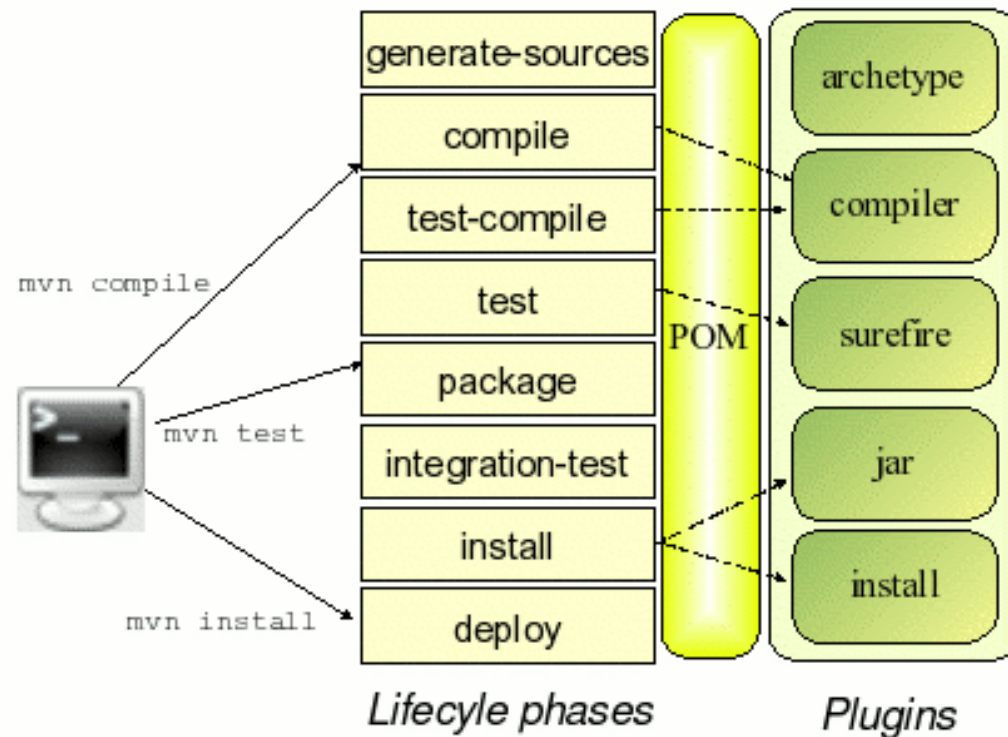


Figure 2. Les phases du cycle de vie Maven 2

Source : developpez.com

compile: Compile le code source du projet

test-compile: Compile les tests unitaires du projet

test: Exécute les tests unitaires dans le répertoire src/test

package: Mets en forme le code compilé dans son format de diffusion (JAR, WAR, etc.)

integration-test: Réalise et déploie le package si nécessaire dans un environnement dans lequel les tests d'intégration peuvent être effectués.

install: Installe les produits dans l'entrepôt local, pour être utilisé comme dépendance des autres projets sur votre machine locale.

deploy: Réalisé dans un environnement d'intégration ou de production, copie le produit final dans un entrepôt distant pour être partagé avec d'autres développeurs ou projets.

Source : developpez.com

- **Aide Maven** : `mvn --help`
- **Tester un projet Maven**
 - La commande `mvn test` depuis le répertoire où se trouve votre fichier `pom.xml`
- **Générer un site d'information à propos de notre projet Maven**
 - Commande `mvn site`
- **Supprimer le répertoire target**
 - Commande `mvn clean`
- **Compiler, tester et packager (Java)**
 - `mvn compile`
 - `mvn test-compile`
 - `mvn package`
- **Installer un jar**
 - `mvn install`



Installation et paramétrage des outils

Présentation Maven : Mise en place d'un environnement de travail:

Pour pouvoir télécharger des dépendances autre que le repoMaven Central, il faut pouvoir renseigner un repository :

```
<?xml version="1.0" encoding="UTF-8"?>
<settings>
  <profiles>
    <profile>
      <repositories>
        <repository>
          <snapshots>
            <enabled>>false</enabled>
          </snapshots>
          <id>squashTA</id>
          <name>squashTA</name>
          <url>http://repo.squashtest.org/maven2/releases/</url>
        </repository>
      </repositories>
      <pluginRepositories>
        <pluginRepository>
          <snapshots>
            <enabled>>false</enabled>
          </snapshots>
          <id>squashTA</id>
          <name>squashTA</name>
          <url>http://repo.squashtest.org/maven2/releases/</url>
        </pluginRepository>
      </pluginRepositories>
    </profile>
  </profiles>
  <activeProfiles>
    <activeProfile>squashTA</activeProfile>
  </activeProfiles>
</settings>
```

Pour configurer un proxy:

```
<?xml version="1.0" encoding="UTF-8"?>
<settings>
  <proxies>
    <proxy>
      <active/>
      <protocol>http</protocol>
      <username>dupont</username>
      <password>lupin</password>
      <port>8080</port>
      <host>my.proxy.url</host> <id/>
    </proxy>
  </proxies>
</settings>
```

Source : developpez.com

- **Créer un projet.**

- La commande `archetype:generate` permet de créer un projet Maven
- C'est par exemple la commande que nous utilisons lors de la création d'un projet Maven Squash TA :
- `mvn archetype:generate -DarchetypeGroupId=org.squashtest.ta -DarchetypeArtifactId=squash-ta-project-archetype -DarchetypeVersion=1.11.0-RELEASE -DarchetypeRepository=http://repo.squashtest.org/maven2/releases`
- Tester la commande depuis cmd (copier/coller ne marche pas)
- Une fois la commande passée, l'arborescence du projet est créée, ainsi que le fichier `pom.xml`



Installation et paramétrage des outils

Présentation Maven : créer un projet

```
— mvn archetype:create -DgroupId=com.javaworld.hotels -  
  DartifactId=HotelWebapp -Dpackagename=com.javaworld.hotels -  
  DarchetypeArtifactId=maven-archetype-webapp
```

la commande depuis cmd (copier/coller ne marche pas)

- Dans le module HotelDatabase, trouvez la classe HotelModel et ajouter deux hotels, un à Tokyo et un à New-York, afin de comprendre à quoi correspond chaque paramètre de Hotel, trouvez la classe correspondante et ajouter les informations nécessaires.
- Produisez un .war utilisable
- Déployer votre application dans un serveur Tomcat



- **Réaliser l'exercice source developpez.com.**
 - Exercice source
 - <http://dcabasson.developpez.com/articles/java/maven/introduction-maven2/#references>



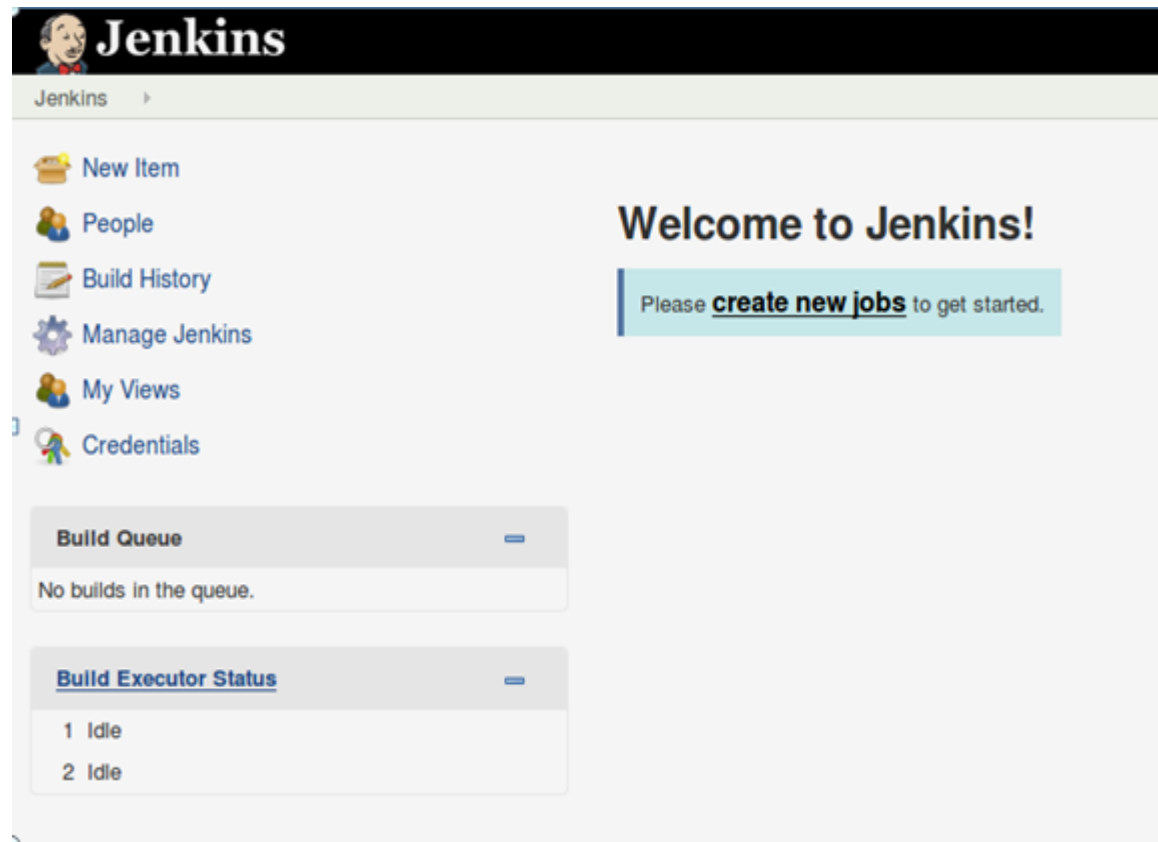
Partie 3 : Prise en main de Jenkins



Installation et paramétrage des outils

Prise en main de Jenkins

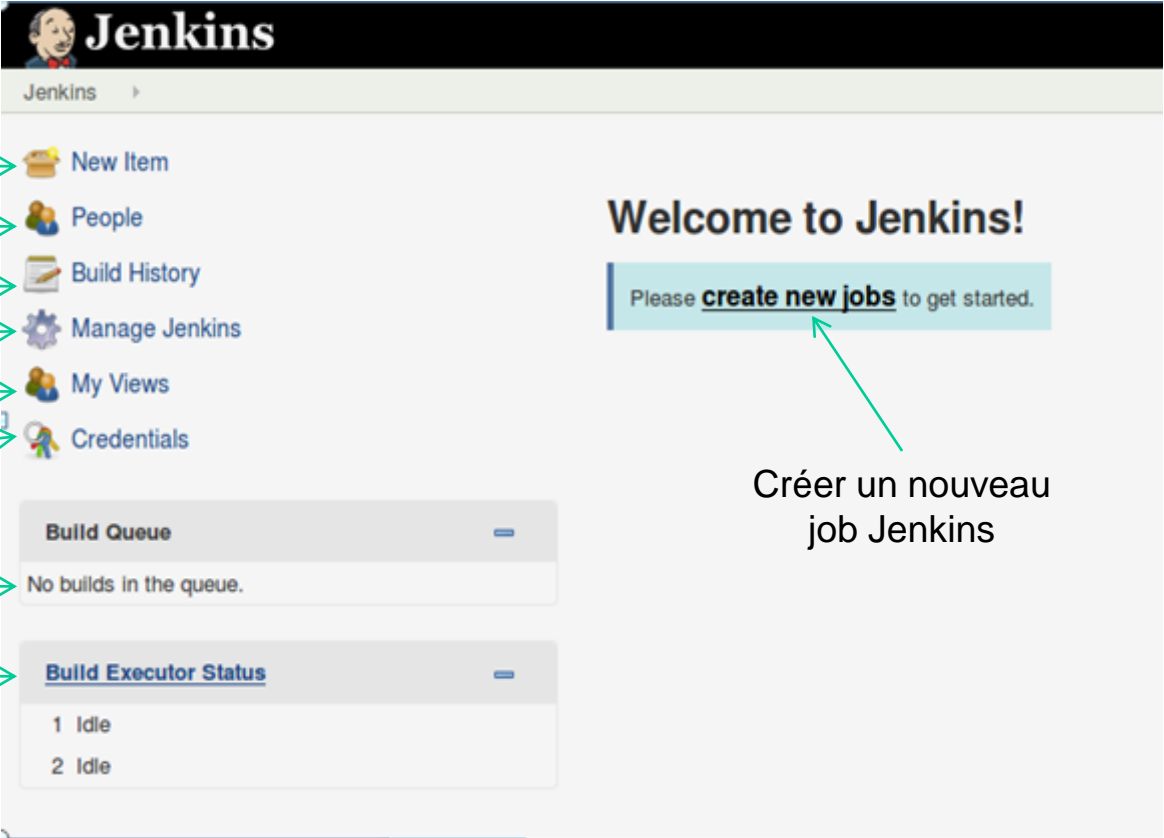
- **Premier lancement de Jenkins :**
 - Exercice (5 minutes) : récupérer le mot de passe administrateur et accéder à Jenkins pour la première fois.
 - Vous arrivez sur la page d'accueil suivante :



Installation et paramétrage des outils

Prise en main de Jenkins

- Ecran d'accueil de l'application Jenkins (écran admin) (1/3)



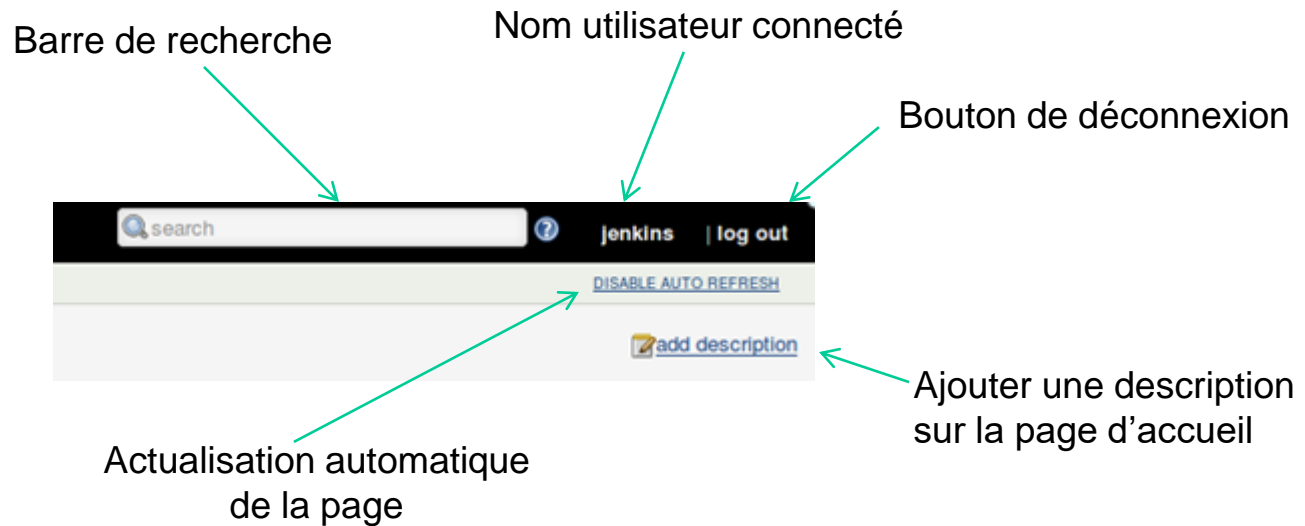
The screenshot shows the Jenkins administration interface. On the left, a sidebar contains several menu items: 'New Item', 'People', 'Build History', 'Manage Jenkins', 'My Views', and 'Credentials'. Below these are sections for 'Build Queue' (showing 'No builds in the queue') and 'Build Executor Status' (showing two 'Idle' executors). On the right, a large area displays 'Welcome to Jenkins!' and a message: 'Please create new jobs to get started.'.

Annotations with arrows point from the following text to the interface:

- Nouveau job → New Item
- Liste utilisateur(s) → People
- Historique des jobs → Build History
- Configuration Jenkins → Manage Jenkins
- Vue projet → My Views
- Gestion profils → Credentials
- Jobs en attente → Build Queue
- Descriptif des jobs exécutés depuis l'écran d'accueil → Build Executor Status

Créer un nouveau job Jenkins

- Ecran d'accueil de l'application Jenkins (écran admin) (2/3)



Installation et paramétrage des outils

Prise en main de Jenkins

- **Ecran d'accueil de l'application Jenkins après le lancement d'un job et création d'un projet (écran admin) (3/3)**
 - Possibilité de créer de nouvelles vues
 - Ligne du job

The screenshot displays the Jenkins administration page. On the left is a sidebar with navigation links: New Item, People, Build History, Project Relationship, Check File Fingerprint, Manage Jenkins, My Views, and Credentials. The main content area features a 'test description' section with an 'edit description' link. Below this is a table of jobs. The table has columns for 'S' (Status), 'W' (Weather icon), 'Name', 'Last Success', 'Last Failure', and 'Last Duration'. One job, 'test_maven_project', is listed with a status of 'S' and a duration of '35 sec'. Below the table are links for 'Icon: S M L' and several RSS feeds. At the bottom, there are two expandable sections: 'Build Queue' (showing 'No builds in the queue.') and 'Build Executor Status' (showing '1 Idle' and '2 Idle').

S	W	Name	Last Success	Last Failure	Last Duration
		test_maven_project	46 min - #4	N/A	35 sec

Icon: [S](#) [M](#) [L](#)












[Legend](#) [RSS for all](#) [RSS for failures](#) [RSS for just latest builds](#)

Build Queue
No builds in the queue.

Build Executor Status
1 Idle
2 Idle

Installation et paramétrage des outils

Prise en main de Jenkins

	Configure System Configure global settings and paths.	→ Panneau de configuration de Jenkins
	Configure Global Security Secure Jenkins; define who is allowed to	→ Panneau de configuration de Jenkins
	Configure Credentials Configure the credential providers and typ	→ Gestion des mots de passe
	Global Tool Configuration Configure tools, their locations and auton	→ Configuration des outils utiliser pour la construction des builds
	Reload Configuration from Disk Discard all the loaded data in memory an	→ Réinitialise les tâches en mémoire
	Manage Plugins Add, remove, disable or enable plugins th	→ Panneau de gestion des plugins
	System Information Displays various environmental informati	→ Informations de l'installation Jenkins sur le serveur d'intégration
	System Log System log captures output from java.	→ Logs Jenkins
	Load Statistics Check your resource utilization and see i	→ Statistiques Jenkins
	Jenkins CLI Access/manage Jenkins from your shell,	→ Liste des lignes de commandes Jenkins
	Script Console Executes arbitrary script for administrati	→ Console de création de scripts Jenkins

Installation et paramétrage des outils

Prise en main de Jenkins



[Manage Users](#)

Create/delete/modify users that can log in to this Jenkins

- Permet de gérer et créer les utilisateurs Jenkins.
 - **Exercice** : créer un nouvel utilisateur (10 minutes)



[Manage Plugins](#)

Add, remove, disable or enable plugins that can extend the functionality of Jenkins.

- Gestion des plugins.
 - Les plugins permettent d'ajouter de nouvelles fonctionnalités à Jenkins
 - Par exemple Maven Integration Plugin permet de créer un nouveau projet Maven. Sans ce plugin, la fonctionnalité n'est pas proposée.
 - Présentation de l'interface plugins :

Updates			
Available			
Installed			
Advanced			
Install	Name ↓	Version	Installed
No updates			

Update information obtained: 3 hr 15 min ago

[Check now](#)

Select: [All](#), [None](#)

This page lists updates to the plugins you currently use.

- **Exercice (10 minutes) :**
 - Ajouter dans Jenkins le plugin suivant :



- Mettre à jour les plug-ins si possible



Installation et paramétrage des outils

Prise en main de Jenkins



Global Tool Configuration

Configure tools, their locations and automatic installers.

- Permet de configurer les outils utilisés par les jobs Jenkins
- Les outils dépendent des plugins précédemment installés
- Vous pouvez par exemple paramétrer votre jdk, vos dépôts git et svn et maven
- Présentation du Global Tool Configuration
- **Exercice (10 minutes) :**
 - Paramétrez vos outils JDK et Maven
 - Pensez à appliquer et valider vos modifications



- Solution de l'exercice 10 (1/2)

JDK

JDK installations

JDK

Name

☒ Install automatically



Install from java.sun.com

Version

☒ I agree to the Java SE Development Kit License Agreement

Delete installer

Add Installer

Delete JDK

Add JDK

List of JDK installations on this system



- Solution de l'exercice 10 (2/2)

Maven

Maven installations

Maven

Name

☒ Install automatically



Install from Apache

Version

Delete Installer

Add Installer

Delete Maven

Add Maven

List of Maven installations on this system



- Menus de nouveau item (job) (change selon les versions et les plugins installés) :
 - Construire un projet freestyle : projet nu où tout doit être paramétré manuellement
 - Construire un projet Maven : projet qui utilise un build Maven
 - Pipeline : permet d'orchestrer plusieurs jobs freestyles → présentation rapide
 - Construire un projet multi-configuration : projet complexe
- Une fois créé, le job apparaît dans la page d'accueil
- Il est possible de facilement supprimer ou modifier un job
- Pour aller sur un job, cliquer dessus



- Création d'un job de test Squash TF - Maven
 - Choisir le type de job souhaité
 - La création d'un job passe par la création d'une configuration globale du job
 - Vous devez savoir ce que vous souhaitez mettre dans votre job
 - Les plugins permettent d'ajouter de nouvelles fonctionnalités (complexe)
 - Exercice (45 minutes) : création d'un projet run test Squash TA
 - Suivre la procédure suivante : <https://sites.google.com/a/henix.fr/wiki-squash-ta/ta-serveur-guide/create-a-job>
 - Vous importerez un projet local créé pour l'occasion
 - Vérifier les paramètres
 - Essayer d'exécuter votre premier build (marche rarement du premier coup)
 - Retour en groupe sur le projet créé
 - Les plugins utilisés
 - Les paramètres
 - L'environnement global du projet



Installation et paramétrage des outils

Prise en main de Jenkins

Gérer un job depuis la page d'accueil

Ajouter un job

Onglets de rangement

Ligne du job

Modifier le job

Répertoire qui contient
le projet

Exécuter le job

Supprimer le job

Configurer le job

Accès dépôts

The screenshot shows the Jenkins job management page. At the top, there are tabs for 'Tous' and 'test', with a '+' button to add a new job. Below the tabs is a table with columns: 'S' (Status), 'M' (Maintenance), 'Nom du projet ↓' (Project Name), and 'Dernier succès' (Last Success). The table lists several jobs, each with a status icon (red, blue, or grey sphere) and a maintenance icon (cloud or sun). A context menu is open over the first job, showing options: 'Modifications', 'Répertoire de travail', 'Lancer un build', 'Supprimer Maven project', 'Configurer', and 'Modules'. Arrows point from the text labels on the left to specific elements in the interface.

S	M	Nom du projet ↓	Dernier succès
		test	S. O.
			1 h 48 mn - #4
			S. O.
			S. O.

Context menu options:

- Modifications
- Répertoire de travail
- Lancer un build
- Supprimer Maven project
- Configurer
- Modules

Installation et paramétrage des outils

Prise en main de Jenkins

- Planifier des jobs
- Orchestrer des jobs



Partie 4 : Introduction à Git



5.2. Les gestionnaires de versions

1.Intégration continue avec Squash-TF + Jenkins + Git > 5.2.Les gestionnaires de versions

- **INTERETS DES GESTIONNAIRES DE VERSIONS (VCS : VERSION CONTROL SYSTEMS)**

- Initialement dédiés à la gestion de code source pour les projets logiciels
- Travail collaboratif :
 - Facilite l'échange
 - Traçabilité
 - Historique
 - Gestion des conflits

- **DIFFERENTS TYPES DE GESTIONNAIRES DE VERSIONS**

- **Systèmes centralisés** : un seul dépôt comme référence

- Subversion (SVN), CVS (Concurrent Versioning System)

Avantages	Inconvénients
Simplifie la gestion des versions	Problématique pour le travail sans connexion

- **Systèmes décentralisés** : Chaque utilisateur possède l'intégralité du dépôt sur sa machine

- GIT, Mercurial, Bazaar

Avantages	Inconvénients
Travail possible sans accès réseau	Cloner un dépôt est long car comprend tout l'historique
Facilite l'utilisation individuelle	

5.3. Git et ses principales commandes

1.Intégration continue avec Squash-TF + Jenkins + Git > 5.3.Git et ses principales commandes

- HISTORIQUE DE GIT

Git est un logiciel de gestion de versions décentralisé créé en 2005 par Linus Torvald, créateur du noyau Linux.

- PRINCIPALES PLATEFORMES GIT



GitHub



Bitbucket

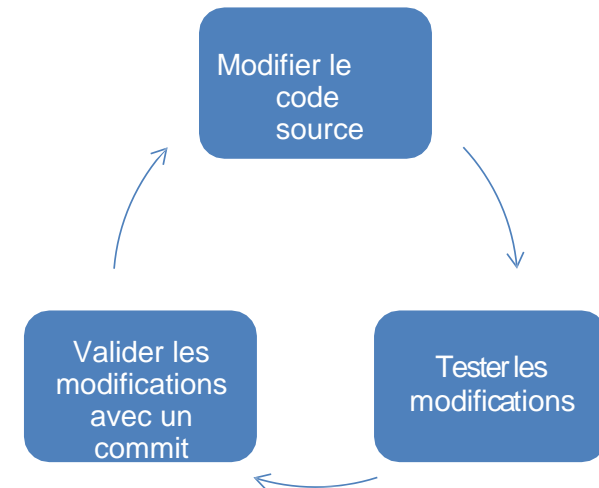


GitLab

- PRESENTATION DE GIT

Lorsque l'on travaille avec Git, on suit en général toujours ces étapes :

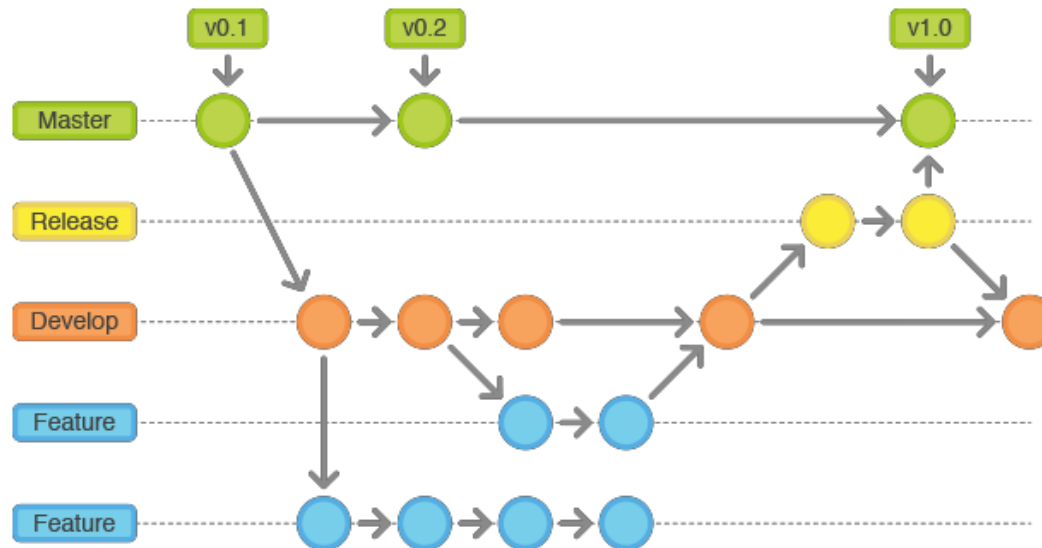
1. Modifier le code source
2. Tester votre programme pour vérifier si cela fonctionne
3. Faire un commit pour « enregistrer » les changements et les faire connaître à Git
4. Recommencer à partir de l'étape 1 pour une autre modification



5.4. Git et ses principales commandes

1. Intégration continue avec Squash-TF + Jenkins + Git > 5.4. Git et ses principales commandes

- WORKFLOW GIT DANS LE DEVELOPPEMENT D'UN PROJET



5.5. Git et ses principales commandes

1.Intégration continue avec Squash-TF + Jenkins + Git > 5.5.Git et ses principales commandes

• PRINCIPALES COMMANDES GIT

Git help

Affiche l'aide sur les différentes fonctionnalités git

- git help
- git help *command* (ex : *git help config...*)

Git config

Configure les préférences utilisateur (dont mail et nom d'utilisateur)

- git config --global user.name Marty McFly
- git config --global user.mail martymcfly@google.com

Git init

Crée un dépôt git

- git init

Git add

Ajoute un fichier à l'index

- git add MaClasse.java

Git commit

Valide les modifications apportées au HEAD

- git commit -m « *description du commit* »

5.6. Git et ses principales commandes

1.Intégration continue avec Squash-TF + Jenkins + Git > 5.6.Git et ses principales commandes

• PRINCIPALES COMMANDES GIT

Git clone

Récupère un dépôt existant et le copie à l'endroit souhaité

- git clone <https://github.com/CompteGithub/DépôtDistant.git>
- git clone utilisateur@serveur:/chemin/vers/dépôt.git

Git status

Affiche la liste des fichiers modifiés ainsi que ceux qu'il reste à ajouter à l'index ou à valider

- git status

Git push

Envoie les modifications locales apportées à la branche courante

- git push origin master

Git branch

Crée, liste ou supprime une branche en fonction de l'option choisie

- git branch
- git branch -d *nomDeLaBrancheASupprimer*

Git checkout

Change de branche vers la branche spécifiée ou crée une nouvelle branche

- git checkout -b *nomBrancheCrééePuisPositionSurCetteBranche*
- git checkout *brancheOùSePositionner*

5.7. Git et ses principales commandes

1.Intégration continue avec Squash-TF + Jenkins + Git > 5.7.Git et ses principales commandes

• PRINCIPALES COMMANDES GIT

Git remote

Permet de se connecter un dépôt distant ou d'ajouter un dépôt où se connecter

- git remote -v
- git remote add origin *adresse*

Git pull

Fusionne les modifications du dépôt distant dans le dépôt local

- git pull

Git merge

Fusionne la branche désirée avec la branche courante

- git merge *nomBrancheAFusionner*

Git log

Génère le log d'une branche, affiche l'historique des commits ainsi que leurs auteurs

- git log

Git rm

Supprime des fichiers de l'index et du répertoire de travail

- git rm monFichier.txt

5.8. Git et ses principales commandes

1.Intégration continue avec Squash-TF + Jenkins + Git > 5.8.Git et ses principales commandes

• PRINCIPALES COMMANDES GIT

Git fetch

Extrait tous les fichiers du dépôt distant qui ne sont pas encore dans le répertoire de travail local

- `git fetch origin`

Git diff

Liste tous les conflits entre les branches à fusionner avant de les fusionner

- `git diff brancheSource brancheCible`

Git tag

Marquage pour identifier des commits spécifiques

- `Git tag 1.2.0 IDduCommit`

Git stash

Enregistre les changements qui ne doivent pas être commit immédiatement (=commit temporaire)

- `git stash`

Git reset

Réinitialise l'index et le répertoire de travail à l'état du dernier commit

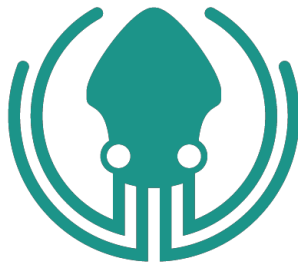
- `git reset --hard HEAD`

5.8.1. Outils Git avec interface graphique

1. Intégration continue avec Squash-TF + Jenkins + Git > 5.8. Git et ses principales commandes



Github client



GitKraken

- Exercice final (2 heure) : Importer votre projet sélénium dans Jenkins et le lancer



Annexes



- Jenkins est le produit d'un développeur visionnaire, Kohsuke Kawaguchi, qui a commencé ce projet comme un loisir sous le nom d'Hudson à la fin de l'année 2004 alors qu'il travaillait chez Sun. Comme Hudson évoluait au cours des années, il était de plus en plus utilisé par les équipes de Sun pour leurs propres projets. Début 2008, Sun reconnaissait la qualité et la valeur de cet outil et demandait à Kohsuke de travailler à plein temps sur Hudson, commençant ainsi à fournir des services professionnels et du support sur Hudson. En 2010, Hudson était devenu la principale solution d'Intégration Continue avec une part de marché d'environ 70%.
- En 2009, Oracle racheta Sun. Vers la fin 2010, des tensions apparurent entre la communauté de développeurs d'Hudson et d'Oracle, dont la source était les problèmes de l'infrastructure Java.net, aggravées par la politique d'Oracle au sujet de la marque Hudson. Ces tensions révélaient de profonds désaccords sur la gestion du projet par Oracle. En effet, Oracle voulait orienter le projet vers un processus de développement plus contrôlé, avec des livraisons moins fréquentes, alors que le cœur des développeurs d'Hudson, mené par Kohsuke, préférait continuer à fonctionner selon le modèle communautaire ouvert, flexible et rapide qui avait si bien fonctionné pour Hudson par le passé. (source : Guide complet Jenkins)



- En Janvier 2011, la communauté des développeurs Hudson vota pour renommer le projet en Jenkins. Par la suite ils migrèrent le code originel d'Hudson vers un nouveau [projet Github](#) et y poursuivirent leur travail. La grande majorité des développeurs du coeur et des plugins leva le camp et suivit Kohsuke Kawaguchi et les autres principaux contributeurs dans le camp de Jenkins, où le gros de l'activité de développement peut être observée aujourd'hui.
- Après ce fork, la majorité des utilisateurs suivit la communauté des développeurs Jenkins et passa à Jenkins. Au moment où ces lignes sont écrites, les sondages montrent qu'environ 75% des utilisateurs d'Hudson sont passés à Jenkins, 13 % utilisent encore Hudson et 12% utilisent Jenkins et Hudson ou sont en cours de migration vers Jenkins.
- Cependant, Oracle et Sonatype (la compagnie derrière Maven et Nexus) ont poursuivi leur travail à partir du code d'Hudson (qui est maintenant lui aussi hébergé chez GitHub à <https://github.com/hudson>), mais selon un axe différent. En effet, les développeurs de Sonatype se sont concentrés sur des modifications dans l'architecture, notamment sur l'intégration avec Maven, sur le framework d'injection de dépendances, et sur l'architecture des plugins. (source : Guide complet Jenkins)



- **Exercice 1 (facultatif)** : Depuis l'application VirtualBox, créer une nouvelle machine virtuelle Ubuntu (système Linux). (50 minutes)
- La machine doit :
 - Avoir suffisamment d'espace disque (il est conseillé de définir directement une taille de mémoire conséquente)
 - Avoir suffisamment de ressource machine
 - Avoir un nom et mot de passe simple
 - Les utilisateurs de Windows 10 peuvent directement créer une machine Ubuntu
 - Ne pas mettre à jour la machine virtuelle (chez un client peut représenter une faille de sécurité)



- **Exercice 2 (facultatif)** : Monter votre environnement Tomcat en vous aidant de la documentation Tomcat/Ubuntu (internet). Pour rappel, Tomcat est un conteneur web. Il intégrera l'application Jenkins (1 heures)
 - Afin de vérifier la bonne installation de serveur d'application, vérifiez vos logs (Tomcat → catalina.out)
 - Systématiquement, avant de passer à l'étape suivante, assurez-vous que tout fonctionne parfaitement.
- **Exercice 3 (facultatif)** : Vérifier et paramétrer votre environnement Java. Paramétrer votre JAVA_HOME (/etc/environment). Vérifiez votre version de Java (java – version) (15 minutes)



- **Exercice 4 (facultatif) :** En utilisant le support Jenkins Ubuntu, installez votre application Jenkins dans le serveur Tomcat. (1h20)
 - Vous n'utiliserez pas le dépôt apt-get
 - Veillez à ce que Jenkins appartienne à l'utilisateur et groupe Tomcat
 - Adaptez les commandes d'installation à votre version de Tomcat (Tomcat7)
 - En suivant la documentation, vérifiez vos variables d'environnement
 - Depuis les logs Tomcat (catalina.out), vérifiez que l'application Jenkins se déploie correctement
 - Tester une première connexion depuis l'URL suivante `http://localhost:8080/jenkins`. Trouvez la solution pour la première authentification
 - Choisissez les plugins dont vous avez besoin.
 - Vous créerez deux scripts (en shell), un script de lancement de la servlet Jenkins et un script de déploiement de Jenkins. Vos scripts doivent avoir une gestion d'erreur et des logs (obligatoire).



- **Retour sur la correction mini-projet Unix**
- **Exercice 1** : Test d'intégration et installation de Jenkins (une demie journée)
 - Reprendre une livraison d'installation de Jenkins réalisée lors du cours Unix
 - Tester la livraison complète (livrables + documentations + suivi installation manuelle + utilisation des scripts automatisés)
 - Vous devez organiser votre plateforme de test de manière à tester la procédure manuelle et automatique (chacun est libre dans la procédure de test) sans tout refaire plusieurs fois
 - Jenkins doit être ensuite utilisable
 - Vous m'enverrez par email votre retour, les anomalies trouvées.

