

# Jenkins, mettre en place l'intégration continue en java

<b>Objectifs pédagogiques</b>	<b>4</b>
A- Où trouver les ressources du cours	4
<b>Pré-requis</b>	<b>5</b>
<b>I- Introduction à l'intégration continue</b>	<b>6</b>
1- Principes et Prérequis de l'intégration continue.	6
2- Développement agile et intégration continue.	7
3- Techniques de développement adaptées à l'intégration continue.	8
A- Les 12 Facteurs/ The Twelve-Factor App	8
4- Différents types de tests et d'audit d'une application.	9
5- Cycle de vie du processus de développement.	11
6- Outils annexes et intégration.	13
<b>II- Mise en place et automatisation du Build</b>	<b>16</b>
1- Un bref rappel historique	16
2- Mise en place du serveur, les différents types d'installation.	17
A- Exemple d'Installation par docker-compose	19
B- Lancement de jenkins via le navigateur	20
> 127.0.0.1:8080	20
3- Configuration : page principale de la configuration, configuration de Git/SVN, serveur de Mail.	24
A- Création de notre premier job en mode free style	25
B- Exécution de mon premier job	28
C- Automatisation du déploiement : Démo	30
4- Stratégies et techniques de notification.	31
5- Fixer les dépendances entre les travaux de Build.	34
6- Jenkins et Maven : rappel sur Maven, configuration du Build Maven, déploiement dans un repository Maven.	37
A- vocabulaire	38

B- Plugins et buts	38
C- Buts et phases	39
D- Etapes du cycle de vie Maven par défaut	39
E- Présentation fichier pom.xml : Project Object Model	43
7- Jenkins et le Build, les meilleures pratiques et méthodes recommandées.	50
<b>III- Qualité du code</b>	<b>51</b>
1- Introduction, intégration de la qualité dans le processus de build.	51
2- Outils d'analyse : Checkstyle, FindBugs, CPD/PMD.	51
3- Configuration du rapport qualité avec le plugin Warning next generation.	52
4- Rapport de complexité, sur les tâches ouvertes.	55
<b>IV- Automatisation des tests</b>	<b>58</b>
<b>V- Administration d'un serveur Jenkins</b>	<b>59</b>
1- Activation de la sécurité et mise en place simple.	59
2- Différents types de bases utilisateurs.	62
3- Gestion des autorisations et des rôles.	64
4- Journalisation des actions utilisateurs.	69
5- Gestion de l'espace disque.	71
6- Monitoring de la charge CPU.	74
7- Sauvegarde de la configuration.	76
<b>Annexe 1: Installation des outils</b>	<b>81</b>
1- Sous Linux/Ubuntu	81
2- Sous Windows	82
<b>Annexe 2 : Introduction à docker</b>	<b>84</b>
1- Définition	84
Qu'est-ce qu'une image Docker?	87
Qu'est-ce qu'un conteneur Docker?	88
Conclusion	88
2- Les commandes de base à connaître	88
Exemples:	90
<b>Annexe 3: Le Générateur de projet web java jhipster</b>	<b>91</b>

1- Génération d'un projet jhipster et déploie sur heroku(provider cloud avec une offre free)	91
Déployer le projet sur github	93
Quelques liens utils	94
<b>Annexe 4: Ghost (moteur de blog)</b>	<b>95</b>

# Objectifs pédagogiques

À l'issue de la formation, le participant sera en mesure de :

- Comprendre les principes de l'intégration continue en vue de son implémentation;
- Intégrer Jenkins avec les autres outils (SCM, gestionnaire de tickets...);
- Mettre en place un serveur Jenkins automatisant les build;
- Automatiser les tests, les audits de code et les déploiements sur la plateforme d'intégration Jenkins

## A- Où trouver les ressources du cours

Les ressources de cours se trouve sur le dépôt github suivant

<https://github.com/e-metconsulting/client-orsys-cours> :

Pour récupérer les sources en local, lancez la commande suivante si vous avez git sur votre pc.

git clone git@github.com:e-metconsulting/client-orsys-cours.git

## Pré-requis

Pour aborder sereinement les modules de ce cours, la connaissance du langage Java et des notions du cycle de développement (Maven) seront capitales sans toutefois être un frein majeur.

La connaissance de certains outils listés ci-dessous serait un plus.

- Un environnement linux : Ubuntu, Fedora
- Si possible connaître **git** et avoir un compte sur <https://github.com/> et ou <https://gitlab.com/> et être familier aux notions de branch et de repository etc
- Si possible avoir des notions de docker

# I- Introduction à l'intégration continue

## 1- Principes et Prérequis de l'intégration continue.

L'intégration continue (CI) est une pratique logicielle consistant à intégrer en continu les modifications du code de différents développeurs travaillant sur le même code dans un référentiel partagé, sujettes à des validations fréquentes.

La validation du code détecte plus souvent les erreurs plus tôt et réduit la quantité de code dont un développeur a besoin pour déboguer afin de trouver la source d'une erreur.

Les mises à jour fréquentes du code facilitent également la fusion du code des différents membres d'une équipe de développement logiciel.

C'est idéal pour les développeurs, qui peuvent passer plus de temps à écrire du code et moins de temps à déboguer les erreurs ou à résoudre les conflits de merge.

Lorsque vous validez du code dans votre référentiel, vous pouvez en permanence créer et tester le code pour vous assurer que la validation n'introduit pas d'erreurs.

Vos tests peuvent inclure des linters de code (qui vérifient la mise en forme du style), des contrôles de sécurité, la couverture du code via les tests unitaires, des tests fonctionnels et de charges et d'autres contrôles personnalisés.

Un outil tel que Sonar, assure la validation de la qualité des livrables avec un suivi régulier par l'implémentation des règles qualitatives.

Le “**building**” et le **test** de votre code nécessitent un **serveur**. Vous pouvez créer et tester les mises à jour localement avant de pousser le code vers un référentiel (repository manager tel que github, gitlab ou bitbucket etc .), ou vous pouvez utiliser un serveur **CICD** (tel jenkins, github ou gitlab, bamboo par exemple ) qui vérifie les nouveaux commits de code dans un **repository**.

## 2- Développement agile et intégration continue.

Les méthodes Agiles sont des procédures de conception de logiciel qui se veulent plus pragmatiques que les méthodes traditionnelles. En impliquant au maximum le demandeur (client), ces méthodes permettent une grande réactivité à ses demandes, et améliorent sa satisfaction.

La notion de méthode agile a été officialisée en 2001 par un document Manifeste Agile (Agile Manifesto) signé par 17 personnalités impliquées dans l'évolution du génie logiciel, et généralement auteurs de leur propre méthode.

Les méthodes de développement agiles deviennent de plus en plus communes et de plus en plus matures. Elles sont appuyées par des bonnes pratiques dans le développement des logiciels qui ont su faire leurs preuves au fil du temps. Il faut savoir que, sans ces bonnes pratiques, les tentatives d'amélioration de la qualité du produit grâce aux méthodes agiles peuvent échouer.

Ainsi, l'intégration continue/déploiement continu (IC/CD) est un élément essentiel d'un système de développement logiciel Agile, qui s'articule autour du concept de collaboration, de conception à l'échelle et de construction durable.

### 3- Techniques de développement adaptées à l'intégration continue.

On peut définir plusieurs techniques de développement logiciel qui s'adaptent à l'intégration continue, les 12 facteurs donnent une vision synthétique des techniques et pratiques pour réussir son intégration continue.

#### A- Les 12 Facteurs/ The Twelve-Factor App

Les 12 Facteurs définis les 12 règles qui doivent régir les développement d'une application (<https://12factor.net/fr/>)

À l'époque actuelle, les logiciels sont régulièrement délivrés en tant que **services** : on les appelle des applications **web** (web apps), ou logiciels en tant que service (**software-as-a-service**).

L'application des 12 facteurs est une méthodologie pour concevoir des logiciels en tant que service qui :

- Utilisent **des formats déclaratifs** pour mettre en oeuvre l'automatisation, pour minimiser le temps et les coûts pour que de nouveaux développeurs rejoignent le projet;
- Ont un **contrat propre** avec le système d'exploitation sous-jacent, offrant une **portabilité maximum** entre les environnements d'exécution;
- Sont adaptés à des **déploiements** sur des plateformes cloud modernes, rendant inutile le besoin de serveurs et de l'administration de systèmes;
- **Minimisent la divergence** entre le développement et la production, ce qui permet **le déploiement continu** pour une agilité maximum;
- et peuvent **grossir verticalement** sans changement significatif dans les outils, l'architecture ou les pratiques de développement;

La méthodologie 12 facteurs peut être appliquée à des applications écrites dans tout langage de programmation, et qui utilisent tout type de services externes (base de données, file, cache mémoire, etc.)

Ils concernent tout développeur qui construit des applications qui fonctionnent en tant que service, ainsi que les personnes qui déploient et gèrent de telles applications.

1. **Base de code:** Le code à un endroit et versionné (git)
2. **Dépendances:** Outil de gestions des dépendances (maven, composer, npm etc)
3. **Configuration:** Avoir un fichier de configuration pour tous les environnements et recourir aux variables d'environnement pour spécifier les valeurs (user, ip, database) exemple docker-compose
4. **Services Externes:** Les services doivent être gérés comme des éléments externes, exemple les connecteurs de base de données
5. **Build/Release/Run:** Ne pas les mélanger dans les pipelines de CICD
6. **Processus:** Sans état (stateless) doit permettre la scalabilité horizontales
7. **Association de ports:** Les services communiquent sur un port
8. **Concurrence:** La scalabilité horizontale c'est à dire création de processus plutôt que l'augmentation de la CPU/RAM
9. **Jetable:** Arrêt gracieux et relance sans impact; doit savoir où elle en est
10. **Parité développement production:** intégration des développeurs aux déploiements
11. **Logs:** flux d'évènements (logstash, elk)
12. **Processus d'administration:** code admin et applicatif ensemble pas de scission

#### 4- Différents types de tests et d'audit d'une application.

Tout projet logiciel nécessite des tests approfondis et répétés. Cela peut prendre énormément de temps et s'avérer répétitif, c'est pourquoi des outils ont été développés pour automatiser certaines parties du processus de test. Ces outils de test exécutent automatiquement des cas de test sur des parties spécifiques du système.

Lorsqu'un développeur introduit du code dans le référentiel, cela déclenche le démarrage des tests automatisés.

L'importance des tests est capitale dans un processus agile permettant de détecter le plus tôt possible des anomalies en validant tous les aspects des applications. Ils assurent donc la livraison d'une application fiable, stable et robuste avec moins de bugs.

Ainsi, sans être exhaustif définira:

- Les tests unitaires, testent les fonctionnalités codées. On définit généralement une couverture du code à 80% pour s'assurer que tout nouveau code intégré respecte cette règle. C'est donc une discipline pour le développeur de garantir cette exigence.
- Les tests d'intégration, c'est à dire les tests sur les environnements cibles
- Le end-to-end tests (E2E), c'est-à-dire, le test de bout en bout, est une méthodologie utilisée pour tester la fonctionnalité et les performances d'une application dans des circonstances et des données similaires à celles d'un produit pour reproduire les paramètres en direct. L'objectif est de simuler à quoi ressemble un scénario utilisateur réel du début à la fin. La réalisation de ces tests ne sert pas seulement à valider le système testé, mais également à s'assurer que ses sous-systèmes fonctionnent et se comportent comme prévu.

## 5- Cycle de vie du processus de développement.

Le cycle IC/DC est un processus agile en huit étapes, en boucle continue, qui garantit un développement logiciel rapide, efficace et stable. Les étapes une à quatre relèvent de l'intégration continue, tandis que les étapes cinq à huit sont considérées comme faisant partie du processus de déploiement continu.

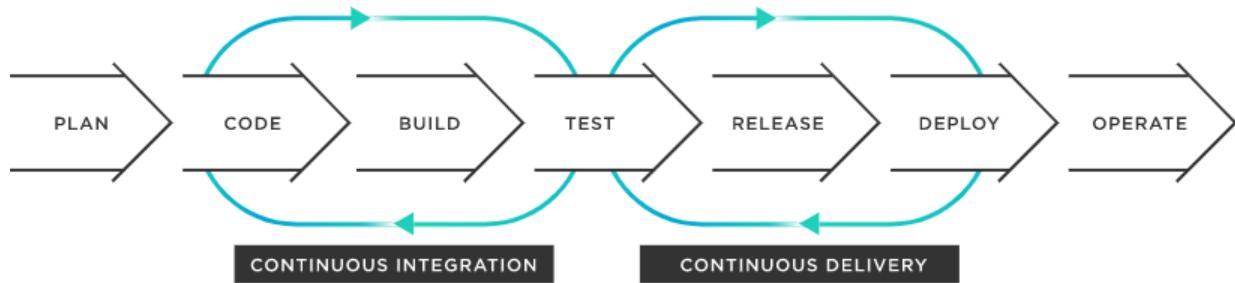


Fig : Cycle de développement CI/CD

1. **Planifier** : les modifications de l'application sont planifiées par l'équipe produit. Il peut s'agir de corrections de bogues, d'améliorations des performances ou de nouvelles fonctionnalités à ajouter à l'application. C'est la phase de construction de la backlog de "stories" ou de son exploitation si elle existe. C'est dans ce référentiel regroupant les fonctionnalités attendues que le product owner (responsable du produit) va sélectionner avec l'équipe, les fonctionnalités qu'il juge prioritaires, à développer durant une courte période de 2 semaines appelée sprint.
2. **Code**: les développeurs codent le logiciel sur leurs machines locales. Chaque développeur a sa propre partie spécifique du système à développer ou un bogue à résoudre. Chaque membre pioche dans la backlog la story à implémenter. Le choix d'une "story" est libre car

l'équipe est autonome. Néanmoins l'équipe peut prioriser certaines stories.

3. **Construction**: une fois que les développeurs ont terminé la deuxième étape, le nouveau code poussé dans le référentiel de code sources. L'application est automatiquement compilée, testée, validée, et déployée dans un environnement de test..
4. **Test** : les testeurs vérifient la fonctionnalité et la facilité d'utilisation du code. Fait-il ce pour quoi il a été conçu ? Ils vérifient également s'il fonctionne avec le reste du code existant. Des tests automatisés doivent être utilisés pour s'assurer que le nouveau code n'interfère pas avec un autre élément faisant déjà partie de l'ensemble: Ce sont les tests de non régression. Si le code est jugé acceptable, il est fusionné. S'il y a des erreurs qui doivent encore être résolues, il est renvoyé au développeur.
5. **Lancement** : une fois terminé, le code affiné est fusionné avec le logiciel et peut être placé dans une version automatisée, en attendant approbation.
6. **Déploiement** : le code est automatiquement déployé en production.
7. **Exploitation** : à ce stade, le nouveau code peut maintenant être exploité dans l'environnement de production.
8. **Contrôle**: les performances de l'application sont surveillées en permanence afin de trouver les bogues et d'identifier les zones où les performances peuvent être améliorées. Il s'agit d'un cycle de retour d'information continu dans lequel les données collectées et analysées à cette étape doivent être transmises à l'équipe produit responsable de la première étape afin qu'elle puisse planifier en permanence les changements et les améliorations à apporter à l'application.

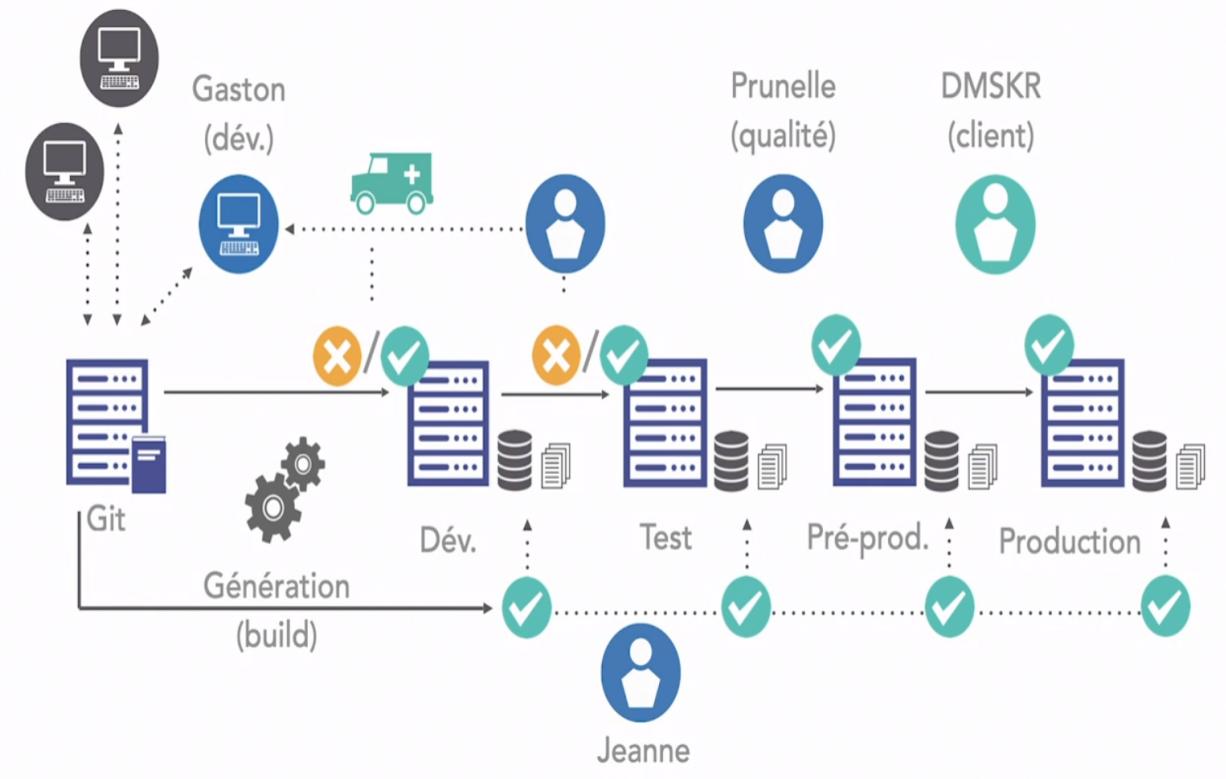


Fig. passer du build à la livraison continue

## 6- Outils annexes et intégration.

L'intégration continue ou CI admet en aval, une composante sous-entendue qui est le déploiement continu ou CD.

Ainsi, en ce qui concerne le déploiement continu, il s'agira de pousser ou déployer régulièrement et fréquemment les nouvelles versions de notre application de façon automatisée dans les environnements cibles, par exemple en production.

Cette notion de déploiement continu peut s'étendre également au fait de stocker dans un référentiel tel que nexus les artefacts de nos composants

applicatifs ou nos images dockers dans une registry tel que docker hub ou gitlab registry etc.

Pour mettre en oeuvre votre intégration et déploiement continue, il faut plusieurs serveurs:

- Un serveur de sources du projet exemple github (git)
- Un serveur pour gérer ses pipelines d'intégration et de déploiement (jenkins, gitlab, github)
- Un repository pour sauvegarder les artefacts de ses projets (nexus, artifactory, github, maven repository ...)
- Un serveur pour la qualité de codes, sonar par exemple
- Des serveurs pour déployer son application pour le test, pour la production, des vps chez des providers cloud ou non, des machines locales, des conteneurs dockers etc.

Il existe donc plusieurs sources de données (serveurs) à provisionner, gravitant autour de son application. Cela demande donc la mise en place d'une architecture dédiée à l'intégration continue et au déploiement continu (CI/CD).

C'est donc au système d'intégration continue et de déploiement que revient la tâche d'orchestrer les workflows et ou pipeline pour provisionner et consommer les données des différents tiers.

Ainsi, il existe différents systèmes mettant en œuvre le concept d'orchestration du workflow pour l'intégration continue et le déploiement continue. On peut citer par exemples

- Azure pipelines
- CircleCI
- Gitlab CI/CD
- Travis CI
- Bamboo
- GitHub Actions
- **Jenkins**

Il est généralement simple avec la documentation fournie de passer d'un orchestrateur à un autre aisément. Cet orchestrateur peut s'appuyer sur un ensemble d'outils tels que Ansible, Consul, des workers pour étendre ses fonctionnalités.

Par la suite, nous porterons notre attention sur jenkins. Il s'agira de présenter dans les grandes lignes cet orchestrateur de CI/CD avec un bref aperçu historiques.

Il sera question également de présenter les différentes façons d'installer jenkins tout en privilégiant l'usage de docker. Une fois l'installation de jenkins réalisée nous présenterons son interface, quelques éléments de configurations et nous créerons "from scratch", nos premiers jobs, installerons quelques plugins pour la gestions rapides des utilisateurs et des droits dans un premier temps.

Par la suite nous aborderons des notions de plus en plus avancées que nous citons sans être exhaustifs tels que:

- L'utilisation de git, github et maven
- Le déclenchement automatique d'un job (cascading, triggers, scrutation etc.)
- Le jenkinsFile
- Les pipelines
- Les pipelines multibranch
- La configuration et administration avancées de jenkins
- Les scripts groovy
- Le test, déploiement
- etc.

## **II- Mise en place et automatisation du Build**

Jenkins est un outil open source de serveur d'automatisation. Il aide à automatiser les parties du développement logiciel liées au build, aux tests et au déploiement, et facilite l'intégration continue et la livraison continue.

Écrit en Java, Jenkins fonctionne dans un conteneur de servlets tel qu'Apache Tomcat, ou en mode autonome avec son propre serveur Web embarqué.

Il s'interface avec des systèmes de gestion de versions tels que CVS, Git et Subversion, et exécute des projets basés sur Apache Ant et Apache Maven aussi bien que des scripts arbitraires en shell Unix ou batch Windows.

Ses fonctionnalités peuvent être étendues facilement à l'aide de plugins supplémentaires qui vous permettront, par exemple, de connecter l'outil à d'autres environnements de test.

Les générations de projets peuvent être amorcées par différents moyens, tels que des mécanismes de planification similaires au cron, des systèmes de dépendances entre générations, ou par des requêtes sur certaines URL spécifiques. (wikipedia)

### 1- Un bref rappel historique

Jenkins était à l'origine nommé Hudson et fut renommé en 2011 après des différends entre son auteur, Kohsuke Kawaguchi, et Oracle qui avait fait un fork du projet et revendiquait les droits sur le nom du projet.

La branche d'Oracle, Hudson, continua d'être développée pendant un temps avant d'être donnée à la fondation Eclipse.

Hudson n'est plus maintenu à jour et est annoncé obsolète en février 2017  
Autour de 2008, Hudson est devenu une solution de remplacement populaire à l'outil de référence CruiseControl.

Le 11 janvier 2011, une proposition pour renommer Hudson a été annoncée afin d'éviter des problèmes avec un éventuel enregistrement (marque déposée) du nom par Oracle.

Après l'échec des négociations avec Oracle un vote en faveur du renommage a été entériné le 29 janvier 2011.

Le 20 avril 2016, la version 2.0 est mise en ligne avec le plugin Pipeline activé par défaut. Ce plugin permet la rédaction d'instructions de Build utilisant un langage de domaine spécifique basé sur Apache Groovy. (wikipedia)

Pour résumer Jenkins est un ordonnanceur Scheduler permettant d'automatiser les tâches, les programmer, les tester, les vérifier, les enchaîner. Il permet de lancer des jobs ou tâches de façon intelligente, de créer des pipelines, c'est-à-dire des tunnels d'intégration continue et de déploiement continu.

Jenkins dispose de plus de 1000 plugins, très simple à installer (mais le pendant sera la maintenance de ceci). Ainsi on trouve des plugins pour tous les usages, par exemple pour améliorer la gestion, l'interface, faire tourner les jobs plus facilement ou sur des outils particulier tel que docker.

Il possède une interface graphique, mais peut également s'utiliser en ligne de commande via un CLI.

## 2- Mise en place du serveur, les différents types d'installation.

L'installation de jenkins peut se faire de diverses manières selon les systèmes d'exploitation.

Les environnements linux sont globalement utilisés pour instancier et manager des applicatifs tel que jenkins. C'est ainsi que nous travaillerons essentiellement sur une distribution linux tel qu'ubuntu.

La documentation d'installation de jenkins se trouve dans le site [jenkins.io](https://www.jenkins.io/doc/book/installing/) à la page : <https://www.jenkins.io/doc/book/installing/>

The screenshot shows the Jenkins User Handbook Overview page. On the left, there's a sidebar with links to User Documentation Home, User Handbook (including sections like Docker, Kubernetes, Linux, macOS, WAR files, Windows, Other Systems, Offline Installations, Initial Settings, Using Jenkins, Pipeline, Blue Ocean, Managing Jenkins, Securing Jenkins, System Administration, Scaling Jenkins, Troubleshooting Jenkins, and Glossary), Tutorials (Guided Tour, Jenkins Pipeline, Using Build Tools), and Resources (Pipeline Syntax reference, Pipeline Steps reference, LTS Upgrade guides). The main content area has a header 'Installing Jenkins'. It includes a back link ('< User Handbook Overview'), an index link ('Index'), and a Docker link ('Docker →'). Below the header, there's a section about new installations of Jenkins, mentioning standalone Java servlet containers like Jetty, and a sidebar titled 'Chapter Sub-Sections' listing Docker, Kubernetes, Linux, macOS, WAR files, Windows, Other Systems, Offline Installations, and Initial Settings. At the bottom of the main content area, there's another set of navigation links and a 'Was this page helpful?' button.

Fig: documentation d'installation de jenkins

- installation par paquet

<https://pkg.jenkins.io/debian-stable/>



```
wget -q -O -http://pkg.jenckins-ci.org/debian/jenkins-ci.org.key | sudo apt-key add -
```

```
sudo sh -c 'echo deb http://pkg.jenkins-ci.org/debian binary/ > /etc/apt/sources.list.d/jenkins.list'
```

```
sudo service jenkins start
```

<https://pkg.jenkins.io/debian-stable/>

- sudo apt-get update  
sudo apt-get install jenkins

Attention à la clef nécessaire à la première connection de l'admin ce password se trouve dans les logs ici :

/var/jenkins\_home/secrets/initialAdminPassword

#### A- Exemple d'Installation par docker-compose

```
version: '2'  
services:  
  jenkins:  
    image: 'jenkins/jenkins:lts'  
    labels:  
      kompose.service.type: nodeport  
    ports:  
      - '80:8080'  
      - '443:8443'  
      - '50000:50000'  
    volumes:  
      - 'jenkins_data:/jenkins_config'
```

**volumes:**

**jenkins\_data:**

**driver: local**

Lancement du docker: **docker-compose up -d**

Vérification du run : **docker ps**

Ces commandes doivent être lancées avec un sudo si on a pas défini de user docker.

```
sudo usermod -aG docker $USER  
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose  
sudo service docker restart  
systemctl status docker.service
```

B- Lancement de jenkins via le navigateur

> 127.0.0.1:8080

## Démarrage

# Débloquer Jenkins

Pour être sûr que que Jenkins soit configuré de façon sécurisée par un administrateur, un mot de passe a été généré dans le fichier de logs ([où le trouver](#)) ainsi que dans ce fichier sur le serveur :

`/var/jenkins_home/secrets/initialAdminPassword`

Veuillez copier le mot de passe depuis un des 2 endroits et le coller ci-dessous.

Mot de passe administrateur



Continuer

Fig : débloquer jenkins avec la clé de connection

Cette commande permet d'avoir de récupérer la clé dans les logs  
**docker logs beb254914277** où **beb254914277** est l'id du docker jenkins via docker ps.

On peut également lancer la commande à partir du nom de l'instance docker : **docker logs jenkins\_jenkins\_1**

Il est également possible de récupérer la clé depuis le lien indiqué ci-dessus:  
En se connectant au docker :

- sudo docker exec -it **beb254914277** sh
- cat /var/jenkins\_home/secrets/initialAdminPassword

## Personnaliser Jenkins

Les plugins étendent Jenkins avec des fonctionnalités additionnelles pour satisfaire différents besoins.

### Installer les plugins suggérés

Installer les plugins que la communauté Jenkins trouve les plus utiles.

### Sélectionner les plugins à installer

Sélectionner et installer les plugins les plus utiles à vos besoins.

Fig: Installation des de bases

Installation en cours...

# Installation en cours...

✓ Folders	✓ OWASP Markup Formatter	✓ Build Timeout	✓ Credentials Binding	Folders ** JavaBeans Activation Framework (JAF) API ** JavaMail API ** bouncycastle API ** Instance Identity ** Mina SSHD API :: Common ** Mina SSHD API :: Core ** SSH server <b>OWASP Markup Formatter</b> ** Structs ** Token Macro <b>Build Timeout</b> ** Credentials ** Trilead API ** SSH Credentials ** Pipeline: Step API ** Plain Credentials <b>Credentials Binding</b> ** SCM API ** Pipeline: API ** commons-lang3 v3.x Jenkins API <b>Timestamper</b> ** Caffeine API ** Script Security ** Ionicons API ** JAXB
✓ Timestamper	⌚ Workspace Cleanup	⌚ Ant	⌚ Gradle	
⌚ Pipeline	⌚ GitHub Branch Source	⌚ Pipeline: GitHub Groovy Libraries	⌚ Pipeline: Stage View	
⌚ Git	⌚ SSH Build Agents	⌚ Matrix Authorization Strategy	⌚ PAM Authentication	
⌚ LDAP	⌚ Email Extension	⌚ Mailer		** - dépendance requise

Jenkins 2.361.4

Fig: Installation des plugins de base.

Démarrage

## Créer le 1er utilisateur Administrateur

Nom d'utilisateur:

 ∅

Mot de passe:

 ∅

Confirmation du mot de passe:

 ∅

Nom complet:

Adresse courriel:

 ∅

Jenkins 2.361.4

Continuer en tant qu'Administrateur

Sauver et continuer

Fig: Création d'un premier user

3- Configuration : page principale de la configuration, configuration de Git/SVN, serveur de Mail.

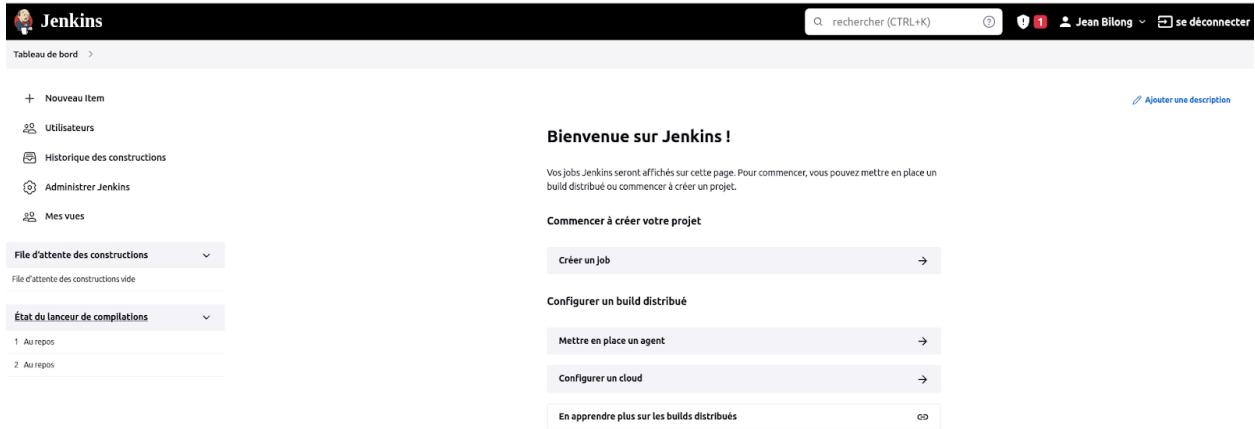


Fig : interface d'accueil de jenkins

L'interface se compose de plusieurs sections :

- **Le Header** est constitué du bandeau noir et du bandeau de file d'ariane pour pister la navigation. Il permet de signaler des informations importante comme des alertes sécurités sur les plugins à mettre à jours
- **Le Footer** où l'on retrouve le lien vers les api de jenkins ainsi que la version installés
- **Le corps de la page** avec une section de gauche donnant des informations sur le statut des jobs et des workers (actuellement au repos car aucun jobs ne tournent), un historique d'exécution des jobs, les différentes vues de l'utilisateur et l'accès à l'administration de Jenkins. Alors la section de gauche permet de lancer rapidement la création de nouveau jobs, de créer des jobs ou pipeline distribués soit en définissant un agent (worker) ou une ressource cloud.

## A- Création de notre premier job en mode free style

La création d'un job peut se faire soit en cliquant sur "Nouveau Item" dans la section de gauche ou "Créer un job" dans la section de droite.

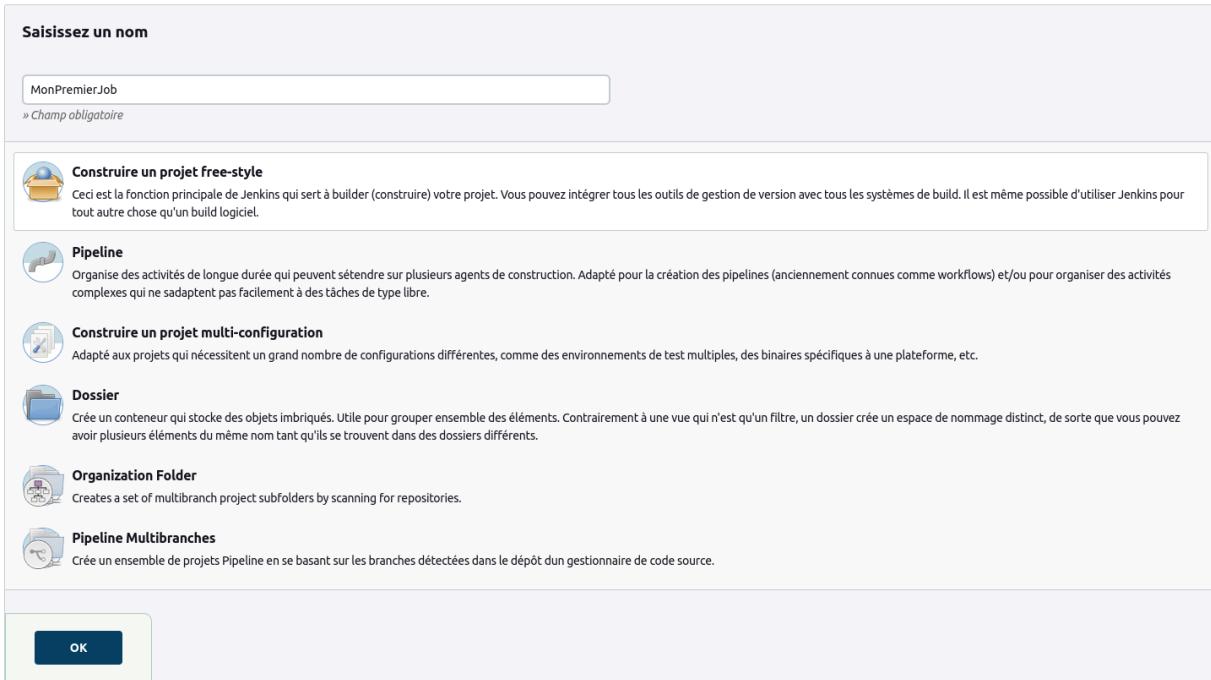


Fig: Choix du type de job à créer (freestyle)

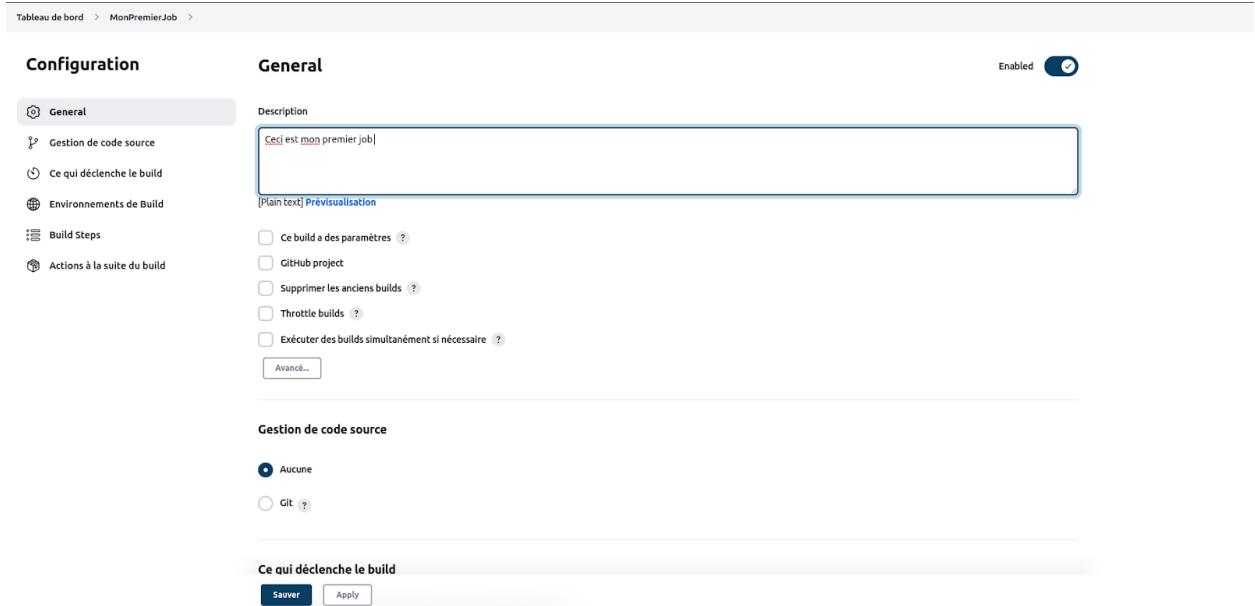


Fig: Définitions des paramètre du job

Dans cette page on a six sections pour définir notre job:

**La section générale:** permet de déclarer la config général du job, par exemple donner une description, déclarer des paramètres, indiquer si le projet que l'on build est un projet github etc.

**La gestion du code source** quant à elle permet de déclarer si on utilise git ou non.

**La section déclenchement du job** sert à définir si l'on souhaite déclencher un job automatique à distance en offrant plusieurs possibilités; par exemple par les triggers, par scrutation du code source, par le déclenchement en cascade suite à l'exécution des jobs précédents ou par définition de la période de déclenchement.

**La section environnement de build** permet de gérer les ressources de travail du build, par exemple le nettoyage de l'espace de travail avant le démarrage du job.

**La section Build step** permet d'ajouter des steps au Job c'est-à-dire les actions ou tâches que doit réaliser le build en exécution par exemple un script shell ou windows.

**La section Action à la suite du build** permet de définir des actions à réaliser à la fin d'exécution du job tels que, la notification par email, la publication d'un tag sur un dépôt git, l'archivage des artefacts ou le nettoyage du workspace, la publication d'un rapport de tests etc.

The screenshot shows the Jenkins dashboard. On the left, there's a sidebar with links: 'Nouveau item', 'Utilisateurs', 'Historique des constructions', 'Administrer Jenkins', and 'Mes vues'. The main area displays a table for the 'MonPremierJob' project. The table has columns: 'Nom du projet', 'Dernier succès', 'Dernier échec', and 'Dernière durée'. The entry for 'MonPremierJob' shows 'S. o.' under 'Dernier succès', 'S. o.' under 'Dernier échec', and 'ND' under 'Dernière durée'. At the bottom of the dashboard, there are sections for 'File d'attente des constructions' (empty), 'État du lanceur de compilations' (status: 'Au repos'), and links for Atom feeds.

Fig: dashboard listant notre premier jobs

## B- Exécution de mon premier job

Pour exécuter notre premier job manuellement il suffit de cliquer sur le bouton lancer un job.

L'historique des exécutions de notre job est rempli à chaque clique. On peut voir des icônes vertes en cas de succès et rouge si le job s'est déclenchée avec des erreurs.

The screenshot shows the Jenkins interface for the project 'MonPremierJob'. At the top, there are navigation links: 'Retour au tableau de bord', 'Projet MonPremierJob', and a dropdown menu 'Etat'. Below this is a sidebar with options: 'Modifications', 'Répertoire de travail', 'Lancer un build' (which is highlighted with a black border), 'Configurer', 'Supprimer Projet', and 'Renommer'. The main content area is titled 'Liens permanents' and lists recent builds. A 'Historique des builds' section shows a list of builds from #1 to #6, each with a timestamp of '22 nov. 2022 07:19'. Build #6 is marked with a green checkmark and build #5 with a red cross. At the bottom of the build list are links for 'Atom feed des builds' and 'Atom feed des échecs'.

Build	Date
#6	22 nov. 2022 07:19
#5	22 nov. 2022 07:19
#4	22 nov. 2022 07:19
#3	22 nov. 2022 07:13
#2	22 nov. 2022 07:13
#1	22 nov. 2022 07:13

Fig : Historique d'exécution de notre job

Pour voir le résultat d'exécution de notre job, il suffit de cliquer sur un item dans l'historique des builds.

Les deux figures ci-dessous montrent le détail d'exécution du job en succès et en échec.

The screenshot shows the Jenkins job details for 'MonPremierJob' build #6. The 'Sortie de la console' (Console Output) section is highlighted with a green checkmark icon. The output shows a successful build:

```

Started by user Jean Bilong Mboumba
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/MonPremierJob
[MonPremierJob] $ /bin/sh -xe /tmp/jenkins10299457631609173196.sh
+ echo Hello World
Hello World
Finished: SUCCESS

```

Fig: Succès d'exécution du build

The screenshot shows the Jenkins job details for 'MonPremierJob' build #5. The 'Sortie de la console' (Console Output) section is highlighted with a red X icon. The output shows a failed build due to a command not found:

```

Started by user Jean Bilong Mboumba
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/MonPremierJob
[MonPremierJob] $ /bin/sh -xe /tmp/jenkins4697873627866697277.sh
+ echffo Hello World
/tmp/jenkins4697873627866697277.sh: 2: echffo: not found
Build step 'Execute shell' marked build as failure
Finished: FAILURE

```

Fig: Échec d'exécution du build

## C- Automatisation du déploiement : Démo

Dans cette partie nous allons faire une démo:

- L'installation du plugin blue-ocean.
- L'intégration de git/github
- L'intégration d'un projet maven
- La mise en place du script de déploiement (JenkinsFile)
- La mise à jour des bases de données
- Les tests minimaux. Retour en arrière.
- La configuration de la notification par e-mail, nécessitant la communication entre jenkins et un serveur mails.

## 4- Stratégies et techniques de notification.

Lorsque vous avez défini votre pipeline pour qu'il se déclenche périodiquement ou à chaque modification du code sources, La mise en place de la notification, permet d'être informé des différentes exécutions de son pipeline (échec, succès). Pour ce faire, il existe plusieurs méthodes.

La première est d'utiliser les flux rss; le premier donnant tous les résultats des jobs systématiquement et le second notifiant seulement les échecs.

The screenshot shows the Jenkins dashboard for build #10, which completed successfully on December 4, 2022, at 14:04. It includes links to the Atom feed for builds and failures, and a section titled "Liens permanents" (Permanent links) listing various build logs and statistics.

- Dernier build (#39), il y a 2 h 40 mn
- Dernier build stable (#39), il y a 2 h 40 mn
- Dernier build avec succès (#39), il y a 2 h 40 mn
- Dernier build en échec (#24), il y a 5 h 2 mn
- Dernier build non réussi (#24), il y a 5 h 2 mn
- Last completed build (#39), il y a 2 h 40 mn

La seconde méthode est la notification par email à la fin du script. Pour se faire il convient de configurer son serveur email, dans l'administration de jenkins. Il faut configurer le serveur SMTP grâce au plugin "**Extended E-Mail Notification**", installé par défaut.

The configuration screen for the "Extended E-mail Notification" plugin. It includes fields for the SMTP server (ssl0.ovh.net), port (465), and credentials (jenkins@bilong.fr/\*\*\*\*\* (ovh)). It also includes checkboxes for "Use SSL", "Use TLS", and "Use OAuth 2.0".

Extended E-mail Notification

SMTP server  
ssl0.ovh.net

SMTP Port  
465

credentials  
jenkins@bilong.fr/\*\*\*\*\* (ovh)

+ Ajouter

Use SSL

Use TLS

Use OAuth 2.0

HTML (text/html)

List ID ?

Add 'Precedence: bulk' E-mail Header ?

Default Recipients ?

Reply To List ?

titi@bilong.fr

Emergency reroute ?

Allowed Domains ?

Excluded Recipients ?

Default Subject ?

\$PROJECT\_NAME - Build # \$BUILD\_NUMBER - \$BUILD\_STATUS!

Listes des variables globales que l'on peut utiliser dans son script:  
<http://51.68.80.153:8080/env-vars.html/>

#### Build Steps

≡ Exécuter un script shell ?

Commande

Voir la liste des variables d'environnement disponibles

Un exemple de configuration du script dans la balise post, du JenkinsFile, il est conseillé de se positionner à la fin du script. On a trois possibilités: définir la notification en cas “**success**”, “**failure**” ou tout le temps “**always**”.

```

post {
    success {
        writeFile file: 'test.xml', text: "<?xml version='1.0'
encoding='UTF-8'?>
<testsuites name='Tidy' tests='1' errors='0'>
    <testsuite name='HTML' tests='1' errors='0'>
        < testcase name='HTML tidy'
classname='index.html'></testcase>
    </testsuite>
</testsuites>"
        junit 'test.xml'
    }
    failure {
        writeFile file: 'test.xml', text: "<?xml version='1.0'
encoding='UTF-8'?>
<testsuites name='Tidy' tests='1' errors='1'>
    <testsuite name='HTML' tests='1' errors='1'>
        < testcase name='HTML tidy' classname='index.html'>
            <failure type='HTML'><![CDATA["
                sh 'cat test.txt >> test.xml'
                sh 'echo "]]></failure></testcase></testsuite></testsuites>" >>
test.xml'
            archiveArtifacts artifacts: 'index.html, index.docx, test.xml',
fingerprint: true
            junit 'test.xml'
        }
        always {
            archiveArtifacts artifacts: 'target/demo-0.0.1-SNAPSHOT.jar',
fingerprint: true
            emailext to : 'jenkins@bilong.fr',
            attachLog: true,
            subject: "Jenkins - ${currentBuild.displayName} :
${currentBuild.currentResult}",
            mimeType: 'text/html',
        }
    }
}

```

```

        body: "<p>Info du <a href='${env.BUILD_URL}'>build</a> : </p>
<ul><li>JOB: '${env.JOB_NAME}'</li><li>N° :
${env.BUILD_NUMBER}'</li></ul>"
    }
}

```

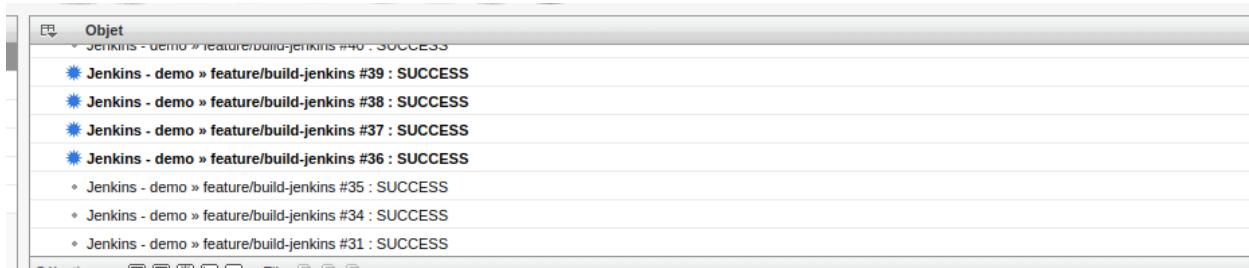


Fig: Notification par email

## 5- Fixer les dépendances entre les travaux de Build.

Il s'agira de configurer la dépendance d'au moins deux Jobs jenkins en définition les conditions de déclenchement des jobs ainsi que leurs liens.

Pour implémenter la dépendances des jobs on va utiliser la notion de trigger de jenkins qui permet d'imbriquer les jobs les uns des autres. Il existe trois type de trigger à savoir :

1. Sur l'échec
2. Sur la réussite
3. Dans tous les cas

C'est-à-dire qu'un job sera lancé lorsque le premier job se termine en erreur, en succès, ou dans tous les cas.

#### Ce qui déclenche le build

Déclencher les builds à distance (Par exemple, à partir de scripts) ?

Construire après le build sur d'autres projets ?

Positionne un déclencheur de build, de façon à ce qu'à la suite du build de certains projets, un build soit lancé pour ce projet. Cela est utile pour lancer un long test après la construction d'un projet, par exemple.

Cette configuration est l'inverse de la section "Construire d'autres projets" dans "Actions post-build". Changer l'un modifiera l'autre automatiquement.

Projet à surveiller

MonPremierJob.

① Pas de projet 'M'. Voulez-vous dire 'MonSecondJob'?

Déclencher que si la construction est stable

Déclencher même si la construction est instable

Déclencher même si la construction échoue

Always trigger, even if the build is aborted

Construire périodiquement ?

GitHub hook trigger for GITScm polling ?

Scrutation de l'outil de gestion de version ?

Fig: Mise en place d'un trigger déclencheur de **MonSecondJob**, en cas de succès de **MonPremierJob**

Icône

File d'attente des constructions (1)

MonSecondJob

État du lanceur de compilations

1 Au repos

2 Au repos

Fig: **MonSecondJob** dans la file d'attente suite au lancement de **MonPremierJob**

		MonPremierJob	9.6 s #9	42 s #8	11 ms		
		MonSecondJob	1.2 s #4	S. O.	11 ms		

Figure: Exécution de **MonSecondJob#4** après une exécution réussie de **MonPremierJob#9**

		MonPremierJob	5 mn 37 s #9	7.1 s #10	11 ms		
		MonSecondJob	5 mn 28 s #4	S. O.	11 ms		

Figure: Pas Exécution de **MonSecondJob#4** après une exécution en échec de **MonPremierJob#10**

Jenkins offre également un trigger permettant de déclencher notre job à distance.

#### Ce qui déclenche le build

Déclencher les builds à distance (Par exemple, à partir de scripts) ?

Jeton d'authentification

Utilisez l'URL qui suit pour lancer un build à distance : JENKINS\_URL/job/MonPremierJob/build?token=TOKEN\_NAME ou /buildWithParameters?token=TOKEN\_NAME  
Optionnaly append &cause=Cause+Text to provide text that will be included in the recorded build cause.

Construire après le build sur d'autres projets ?

Construire périodiquement ?

GitHub hook trigger for GITScm polling ?

Scrutation de l'outil de gestion de version ?

Pour lancer notre job à distance on va donc utiliser l'url fournie:

- <http://51.68.80.153:8080/job/MonPremierJob/build?token=1234>

#### Ce qui déclenche le build

The screenshot shows the 'Triggers' section of a Jenkins job configuration. It includes:

- Déclencher les builds à distance (Par exemple, à partir de scripts) [?](#)  
Jeton d'authentification  
1234  
Utilisez l'URL qui suit pour lancer un build à distance : JENKINS\_URL/job/MonPremierJob/build?token=TOKEN\_NAME ou /buildWithParameters?token=TOKEN\_NAME  
Optionnellement ajouter &cause=Cause+Text pour fournir du texte qui sera inclus dans la cause enregistrée.
- Construire après le build sur d'autres projets [?](#)
- Construire périodiquement [?](#)  
Planning [?](#)  
\*\*\*\*\*  
⚠️ Voulez-vous vraiment dire "chaque minute" avec l'expression "\*\*\*\*\*"? Peut-être voulez-vous dire "H \* \* \* \*"?  
Aurait été lancé à Sunday, December 4, 2022 at 11:03:26 PM Coordinated Universal Time; prochaine exécution à Sunday, December 4, 2022 at 11:03:26 PM Coordinated Universal Time.
- GitHub hook trigger for GITScm polling [?](#)
- Scrutation de l'outil de gestion de version [?](#)

Figure: Déclenchement d'un job périodiquement toute les minutes

6- Jenkins et Maven : rappel sur Maven, configuration du Build Maven, déploiement dans un repository Maven.

Maven est outil de construction d'application java qui :

- Génère une application « déployable » à partir d'un code source
- Compile
- Exécute des tests

C'est aussi outil de gestion de développement qui gère aussi :

- La documentation
- Les Rapports
- Le Site web
- etc.

Intégrer maven dans son processus de développement logiciel est une bonne pratique permettant de :

- **Privilégier la standardisation à la liberté (Convention over Configuration)**

- Structure standard des répertoires d'une application
- Cycle de développement standard d'une application
- Maven se débrouille souvent tout seul !
- 
- **Factoriser les efforts**
  - Un dépôt global regroupant les ressources/bibliothèques communes
  - Des dépôts locaux
  - Un dépôt personnel (~/.m2)
- **Multiplier les possibilités**
  - Une application légère
  - De nombreux plugins, chargés automatiquement au besoin

## A- vocabulaire

**Plugin:** Extension de l'application de base, proposant un ensemble de buts

**But (Goal):** Tâche proposée par un plugin permettant de lancer un certain nombre d'action lorsqu'il est invoqué par mvn plugin:but.  
Paramétré par -Dparam=valeur

**Phase (Maven Lifecycle Phase):** Phase du cycle de développement d'un logiciel, généralement associée à des buts et exécutée par mvn phase

**Artefact (Artifact):** Application dont le développement est géré via Maven

**POM (Project Object Model):** Fichier xml décrivant les spécificités du projets (par rapport à Build.xml, décrit non pas tout, mais juste le « non standard »)

## B- Plugins et buts

### Exemples

#### **Plugin archetype**

Pour créer de nouveaux projets standards  
Buts : generate, create

### **Plugin compiler**

Pour la compilation de code java  
But : compile, test-compile

### **plugin surefire**

Pour l'exécution des tests  
But : test

### **plugin JaCoCo/Cobertura**

pour la générations des rapports de couverture avec  
But : test

## C- Buts et phases

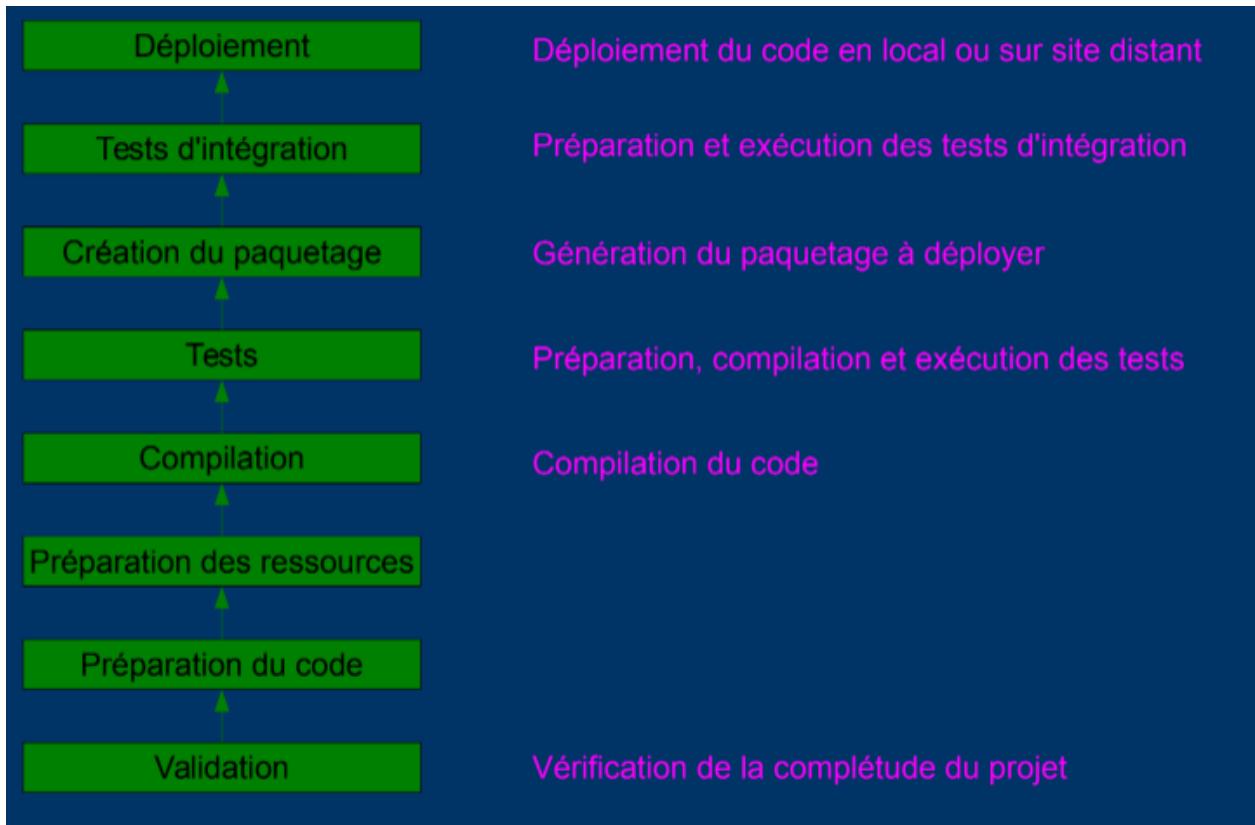
### **Exécution d'un but**

Exécution du but seulement

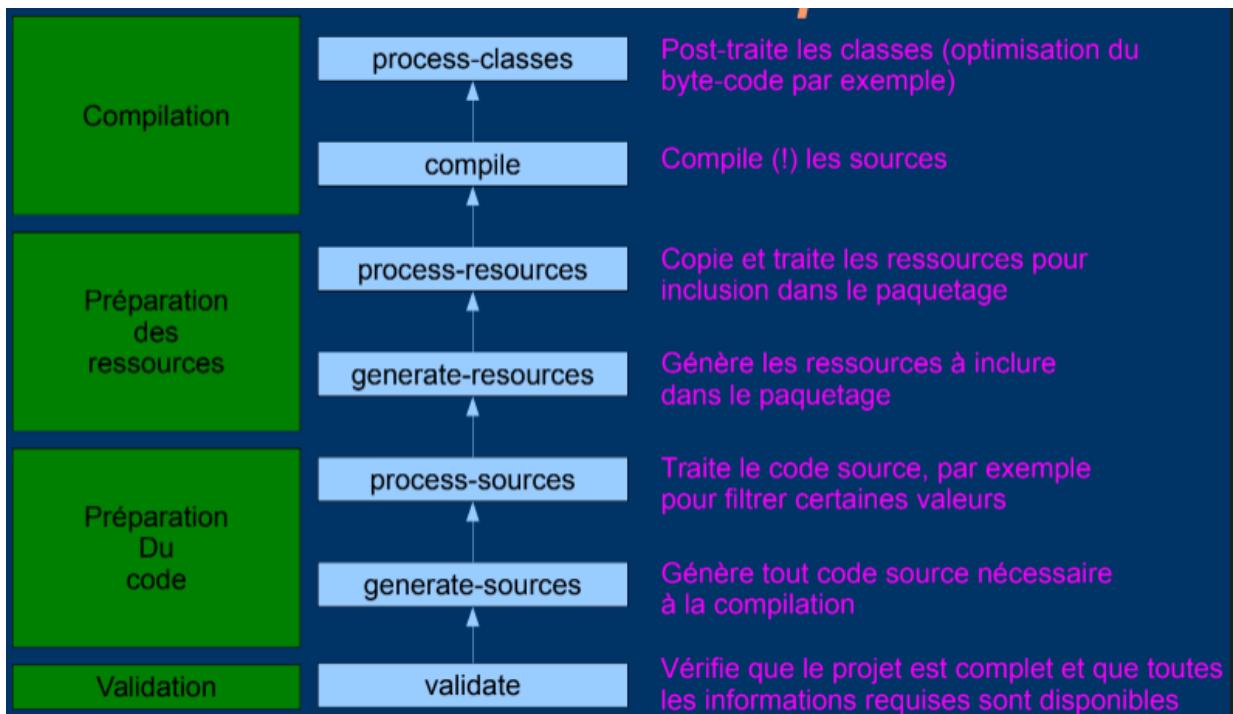
### **Exécution d'une phase**

Exécution de la phase précédente  
Exécution du but associé

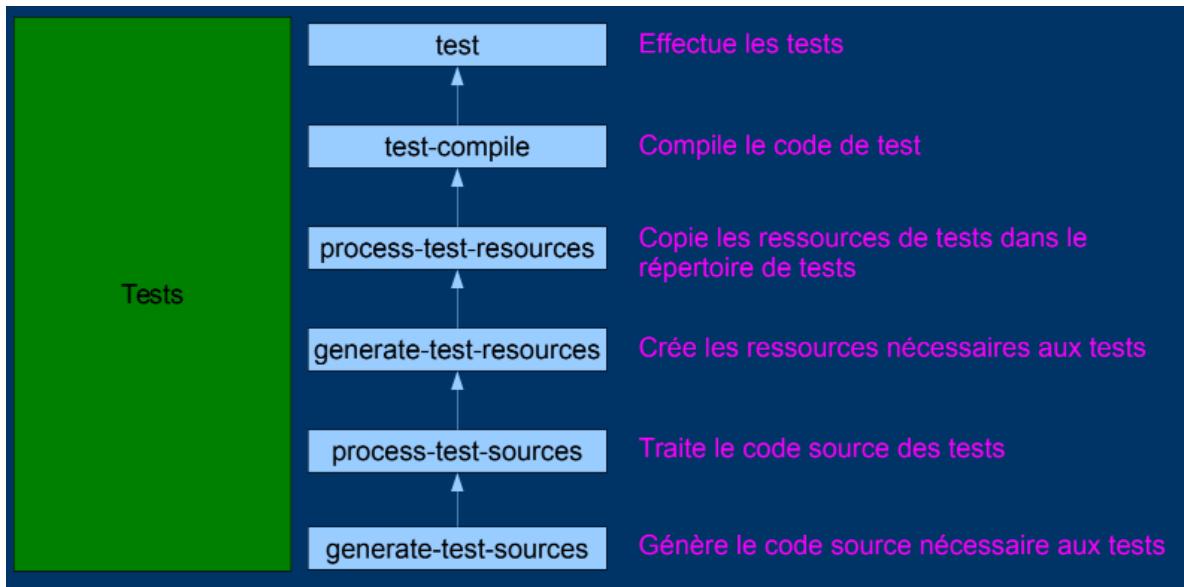
## D- Etapes du cycle de vie Maven par défaut



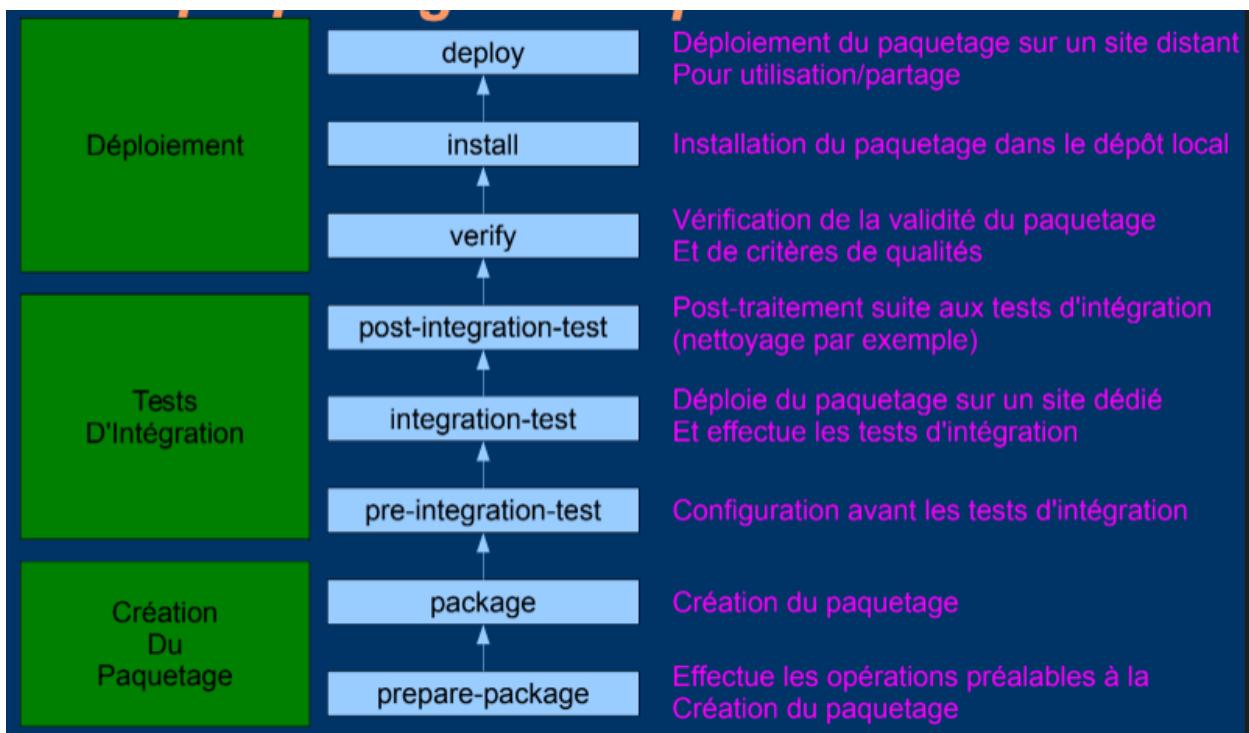
## 1. De la Validation à la compilation



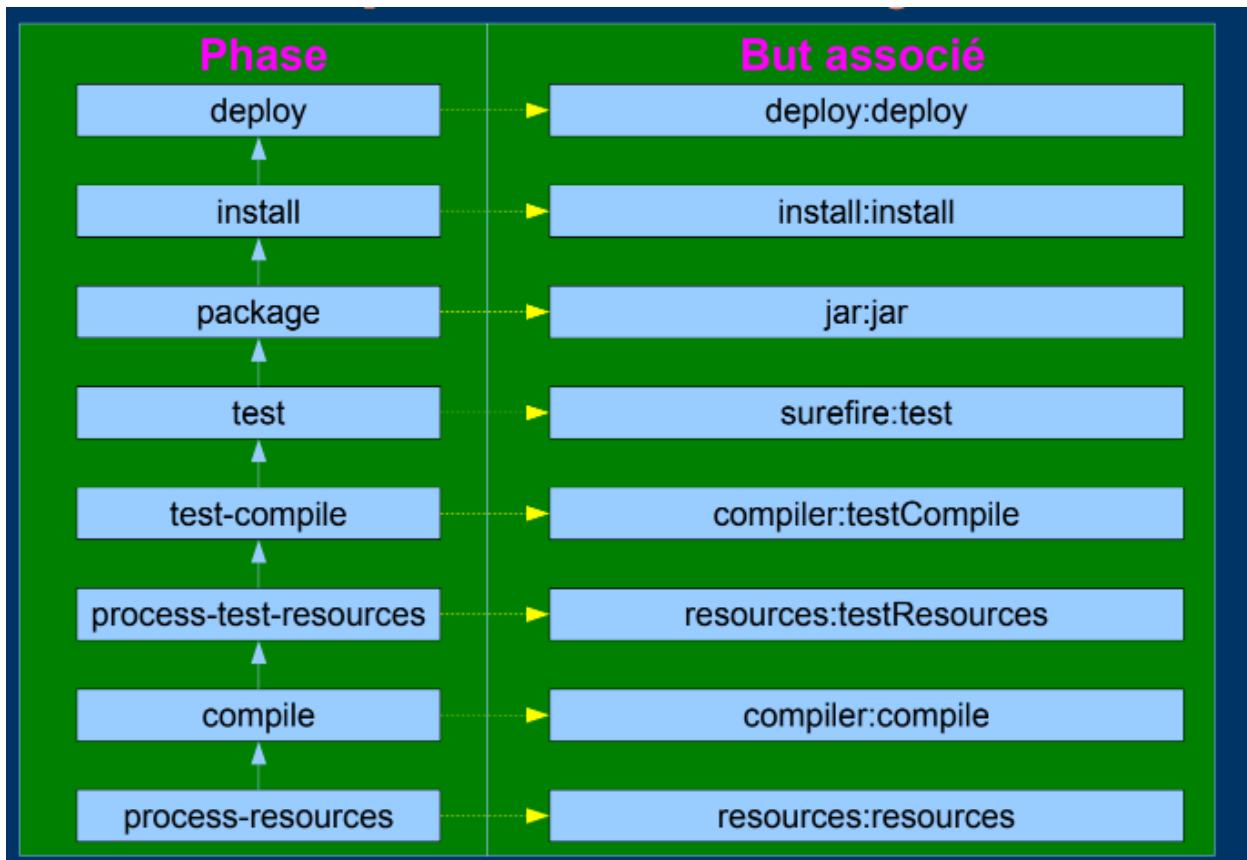
## 2. Tests



## 3. Du paquetage au déploiement



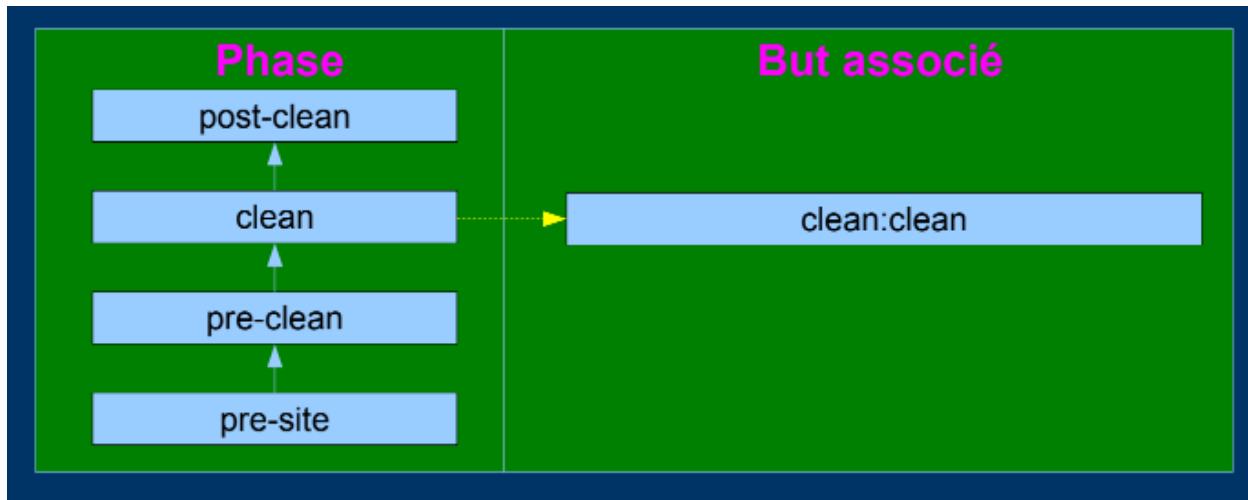
1. Cycle de vie pour un fichier jar:



2. Cycle de vie pour un site:

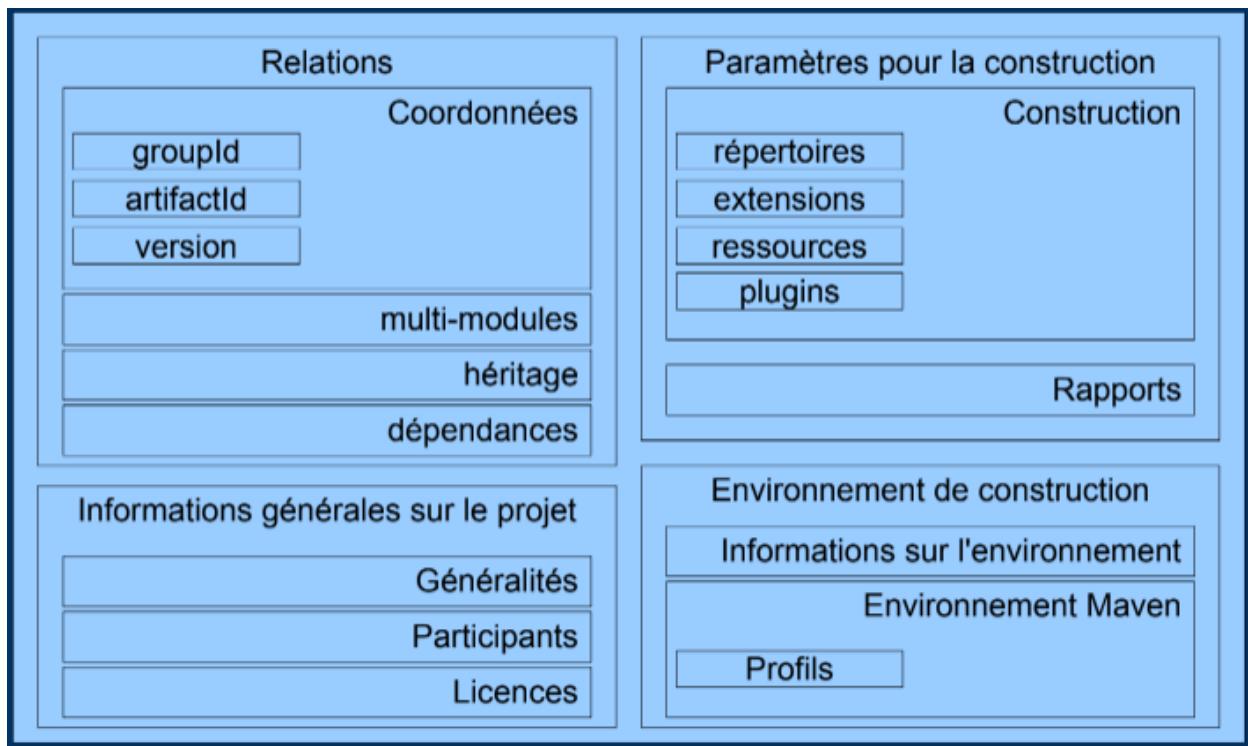


3. Cycle de vie pour le nettoyage



## E- Présentation fichier pom.xml : Project Object Model

Structure générale



## Fichier pom.xml minimal

```
1 <project
2   xmlns="http://maven.apache.org/POM/4.0.0"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-
4   instance"
5   xsi:schemaLocation="http://maven.apache.or-
6   g/POM/4.0.0
7
8   http://maven.apache.org/maven-
9   v4_0_0.xsd">
10
11   <modelVersion>4.0.0</modelVersion>
12   <groupId>edu.mermet</groupId>
13   <artifactId>EssaiProjetMaven</artifactId>
14   <packaging>jar</packaging>
15   <version>1.0-SNAPSHOT</version>
16   <name>EssaiProjetMaven</name>
17   <url>http://maven.apache.org</url>
18
19   <dependencies>
20     <dependency>
21       <groupId>junit</groupId>
22       <artifactId>junit</artifactId>
23       <version>3.8.1</version>
24       <scope>test</scope>
25     </dependency>
26   </dependencies>
27 </project>
```

## Les coordonnées maven d'un projet

- Identifiant
  - GroupId
    - En général, le nom du domaine à l'envers
  - ArtifactId
    - Ce que l'on veut

- Version
  - majorVersion.minorVersion.incrementalVersion-qualifier
  - Permet de spécifier une version minimale lors d'une dépendance avec order <num,num,num,alpha>
    - Si contient SNAPSHOT, remplacé par dateHeureUTC lors de la génération du package
    - Non importé par défaut
- Packaging
  - Type de paquetage à produire (jar, war, ear, etc.)

## Les dépendances au sein d'un projet maven

- Spécifiées par
  - Un identifiant (group/artifact/version)
  - Une portée (scope) : compile par défaut
- Récupération du projet
  - Soit localement
  - Soit sur un dépôt distant
- Version
  - 3.8.1 : si possible 3.8.1
  - [3.8,3.9] : de 3.8 à 3.9 inclus
  - (3.8,4.0) : à partir de 3.8, mais avant 4.0
  - (,4.0) : antérieur à 4.0
  - [3.8,] : à partir de 3.8
  - [3.8.1] : 3.8.1 absolument
- Portée
  - Compile : nécessaire à la compilation, et inclus dans le paquetage
  - Provided : nécessaire à la compilation, non packagé (ex : Servlets)
  - Runtime : nécessaire pour exécution et test, mais pas compilation (ex : driver jdbc)
  - Test : nécessaire uniquement pour les tests
  - System : comme provided + chemin à préciser, mais à éviter

## Structure des répertoires

- Répertoire Du Projet
  - pom.xml
  - src
    - main
      - java
      - resources
    - test
      - java
      - resources
  - target
    - classes
    - test-classes

## Installation de maven

- Variables d'environnement à définir
  - M2\_HOME
    - Doit contenir le répertoire d'installation de maven
  - PATH
    - Rajouter \$M2\_HOME/bin
  - JAVA\_HOME
    - Doit contenir le répertoire d'installation de java

## Création d'un projet Maven

- Plugin archetype
  - Suggestion
    - mvn archetype:generate -DgroupId=monGroupe -DartifactId=monAppli -Dversion=1.0-SNAPSHOT
    - Choisir le type de projet « maven-archetype-quickstart »
    - Confirmer avec « Y »
  - Remarques
    - Les arguments non renseignés seront demandés à l'exécution

Création d'un projet Maven avec spring-boot admin à partir de générateur de projet. Il faut se référer à l'annexe 3.

Générateurs web:

- <https://start.spring.io/>
- <https://start.jhipster.tech/>

Compilation

- Préambule
  - Exécuter toutes les commandes depuis le répertoire contenant le fichier pom.xml
- Compilation
  - commande
    - mvn compile
  - Bilan
    - exemple/target/classes/edu/mermet/App.class
- Exécution (à des fins de test)
  - mvn exec:java -Dexec.mainClass=edu.mermet.App ou / java-jar target/\*\*.java

Packaging et Installation

- Packaging
  - Commande
    - mvn package
  - Bilan
    - Ne recompile pas le code, mais compile la classe de test
    - Exécution avec succès du seul test unitaire
    - Création de
      - exemple/target/test-classes
      - exemple/target/surefire-reports
      - exemple/target/exemple-1.0-SNAPSHOT.jar
- Installation

- Commande
  - mvn install
- Bilan
  - Ré-exécution des tests (pour éviter cela, mvn install -Dmaven.test.skip=true)
  - Création de  
 $\$HOME/.m2/repository/edu/mermet/exemple/1.0-SNAPSHOT/exemple-1.0-SNAPSHOT.jar$ , ...

## Génération d'un site web

- Commande
  - mvn site
- Bilan
  - exemple/target/site
    - CSS
      - maven-base.css, maven-theme.css
      - print.css, site.css
    - images
      - collapsed.gif, expanded.gif
      - external.png
      - icon\_error\_sml.gif, icon\_info\_sml.gif
      - icon\_success\_sml.gif, icon\_warning\_sml.gif
      - logos
        - build-by-maven-black.png
        - build-by-maven-white.png
        - maven-feather.png
    - newwindow.png

## Maven et les tests

- Exécution des tests à chaque invocation de la phase test
  - Évitement
    - En ligne de commande : -Dmaven.test.skip=true
    - Via le POM

```

<plugins>
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <configuration>
      <skip>true</skip>
    </configuration>
  </plugin>
</plugins>

```

- Echec d'un test au moins => blocage de la phase package
  - Évitement
    - En ligne de commande :
 -Dmaven.test.failure.ignore=true
    - Via le POM

```

<plugins>
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <configuration>
      <testFailureIgnore>true</testFailureIgnore>
    </configuration>
  </plugin>
</plugins>

```

## Maven la possibilité de

- Définir des profils permettant soit de tester des configurations différentes (sur sélection manuelle) de définir une configuration automatique selon la plateforme d'exécution de Maven
- Faire l'Héritage entre POMs
- Intégrer subversion ou git via plugin scm (Source Code Management)
- créer un projet multi-modules c'est à dire des sous-répertoire par module, avec un pom par module
- Etc.

## 7- Jenkins et le Build, les meilleures pratiques et méthodes recommandées.

Comme on a l'a vu jenkins permet de créer des jobs en mode freestyle en définissant les scripts d'exécution dans les jobs. Ceci est utile pour tester rapidement les actions d'un Job.

Cependant une meilleur pratique est de pourvoir version son script d'exécution pour se faire il préférable d'utiliser le Jenkins File, dans son projet ceci permet de versionner son pipeline.

Par ailleurs, en ce qui concerne les dépendance entre job, la meilleure pratique est d'utiliser les pipelines jenkins (on peut créer rapidement son pipeline avec blueocean ou depuis l'interface de jenkins)

Pour aller plus loin sur les pipeline voir la documentation jenkins :

<https://www.jenkins.io/doc/book/pipeline/>

## **III- Qualité du code**

### **1- Introduction, intégration de la qualité dans le processus de build.**

Maven permet de générer des rapport sur la qualité du code, ces fichiers de rapports peuvent être indexé par jenkins et intégrer dans les jobs pour visualiser, les violations des métriques de qualité statique.

Il convient donc d'intégrer dans nos pipeline une phase de vérification de la qualité du code avant compilation.

### **2- Outils d'analyse : Checkstyle, FindBugs, CPD/PMD.**

Les plugins FindBugs, PMD et CheckStyle Jenkins ont tous été abandonnés au profit de l'outil de génération de rapport d'analyse de code statique Warnings Next Generation. Donc, si vous envisagez de passer à une version plus récente de l'outil d'intégration continue populaire, ou si vous ne savez pas pourquoi vous ne trouvez pas le plugin CheckStyle Jenkins dans la console de gestion de l'outil CI, vous voudrez prendre note des instructions fournies ici.

Le rôle d'Apache Maven en termes d'intégration de Jenkins avec PMD, FindBugs et Checkstyle reste le même. Toute exécution Jenkins qui a l'intention d'utiliser ces outils d'analyse de code statique devra invoquer ces outils dans le cadre de la construction avec l'appel Maven suivant:

- mvn install checkstyle:checkstyle pmd:pmd findbugs:findbugs

Cependant, sans plugin Jenkins pour intégrer les résultats de l'analyse de code statique dans la construction, les divers fichiers XML, JSON et texte créés par ces outils resteront simplement inactifs sur le système de fichiers.

Malheureusement, les anciens plugins FindBugs, PMD et CheckStyle sont obsolètes. Heureusement, les trois plugins obsolètes ont été remplacés par un qui est extensible et beaucoup plus facile à utiliser.

Installez le plug-in Warnings Next Generation de Jenkins (<https://plugins.jenkins.io/warnings-nl/>) à partir de la console de gestion et vous disposerez d'un seul plugin Jenkins capable de traiter les résultats des trois outils.

Avec le plug-in Warnings Next Generation installé, une nouvelle option de post-construction nommée Enregistrer les avertissements du compilateur et les résultats de l'analyse statique devient disponible pour toutes les tâches de build Jenkins. Il fournit une liste déroulante à partir de laquelle des centaines d'outils d'analyse de code statique différents peuvent être choisis, notamment CheckStyle, PMD et FindBugs.

L'étape de construction ne doit être ajoutée qu'une seule fois au job Jenkins, après quoi le bouton Ajouter un outil peut être utilisé pour ajouter des rapports supplémentaires.

Avec les différents outils d'analyse de code statique ajoutés au plugin Jenkins Warnings Next Generation, les rapports pour chacun seront affichés dans la fenêtre d'état de la tâche de build.

### 3- Configuration du rapport qualité avec le plugin Warning next generation.

Pour la configuration du plugin il suffit de se rendre dans le build step et sélectionner : **Invoyer les cibles Maven de haut niveau.**

Pour les projets qui utilisent Maven comme outil de build. Jenkins invoquera Maven avec les cibles et les options spécifiées. Un code de retour différent de zéro indique à Jenkins que le build doit être marqué

comme un échec. Certaines versions de Maven ont un bug qui ne permet pas le retour correct d'un code de sortie.

Jenkins passe certaines variables d'environnement à Maven, auxquelles vous pouvez accéder à l'aide de "\${env.NOMDEVARIABLE}".

Les mêmes variables peuvent être utilisées comme des arguments de ligne de commande, comme si vous faisiez une invocation à partir d'un Shell. Par exemple, vous pouvez spécifier:

- -DresultsFile=\${WORKSPACE}/\${BUILD\_TAG}.results.txt



Puis dans le post build, Record compiler warnings and static analysis results, afin de rajouter les 3 outils de vérification statique du code à savoir Checkstyle, FindBugs, CPD/PMD.

#### Actions à la suite du build

The screenshot shows a configuration dialog for a static analysis tool named 'FindBugs'. The dialog has a header 'Record compiler warnings and static analysis results' and a section 'Static Analysis Tools' with a 'Tool' dropdown set to 'FindBugs'. It includes fields for 'Report File Pattern' (empty), 'Skip symbolic links when searching for files' (unchecked), 'Report Encoding' (empty), 'Use the bug rank when evaluating the severity of the warnings' (checked), 'Custom ID' (empty), and 'Custom Name' (empty). A red 'X' button is visible in the top right corner.

Une fois le job exécuté avec succès on peut voir de nouveaux liens sur le résultat du job lorsqu'on clique sur le numéro de build.

## Construction #2 (5 déc. 2022 à 10:17:01)

 No changes.

 Lancé par l'utilisateur [bilong mboumba](#)

 Revision: eaa279f8359a55dc7c6773f8d80b36cf0ef26624  
Repository: <https://github.com/bilonjea/rps-boot.git>

- refs/remotes/origin/master

 FindBugs: 3 warnings 

 CheckStyle: 221 warnings 

 PMD: 10 warnings 

Figure: Résultat d'exécution du job avec les différents rapport de qualité.

### 4- Rapport de complexité, sur les tâches ouvertes.

Les rapports se présentent et se comportent exactement de la même manière que les plug-ins obsolètes CheckStyle, PMD et FindBugs, de sorte que les professionnels DevOps familiarisés avec l'ancien mode d'intégration ne rencontreront aucune cure d'apprentissage traitant les résultats.

Et ceux qui découvrent l'analyse de code statique dans Jenkins trouveront les commentaires et les informations sur la qualité de leur code extrêmement utiles.

## PMD Warnings

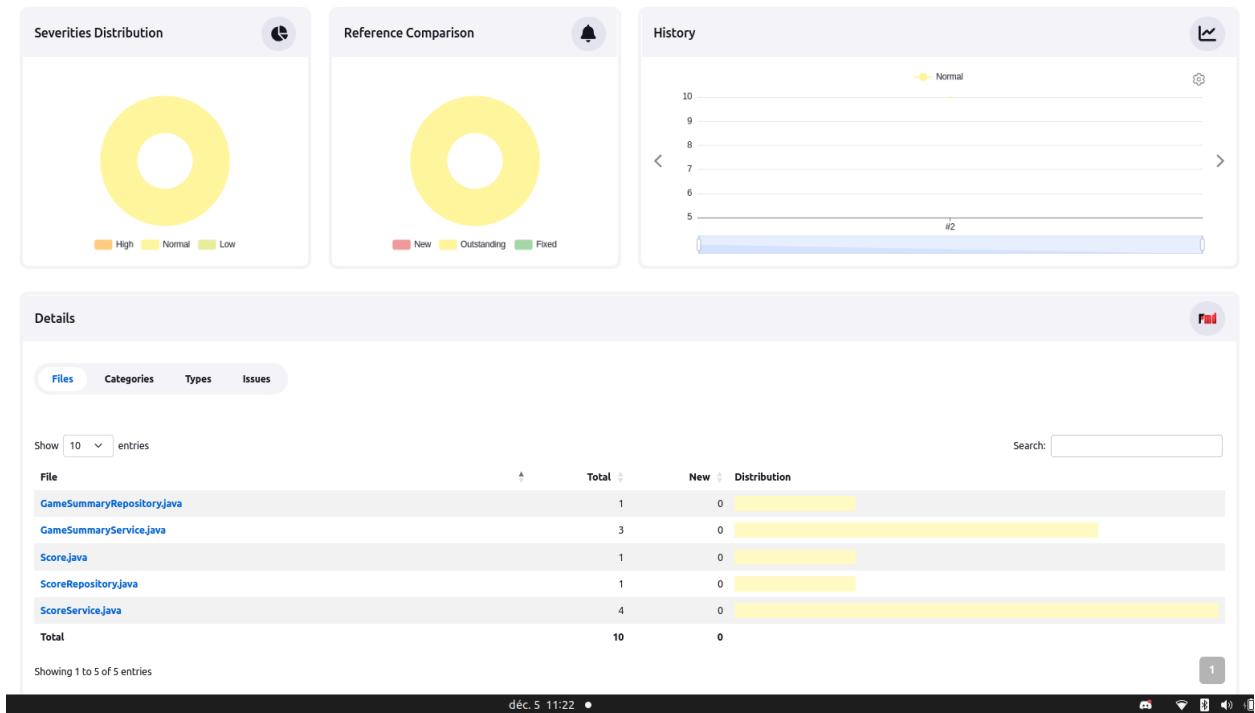


Figure: Exemple de rapport PMD

## CheckStyle Warnings

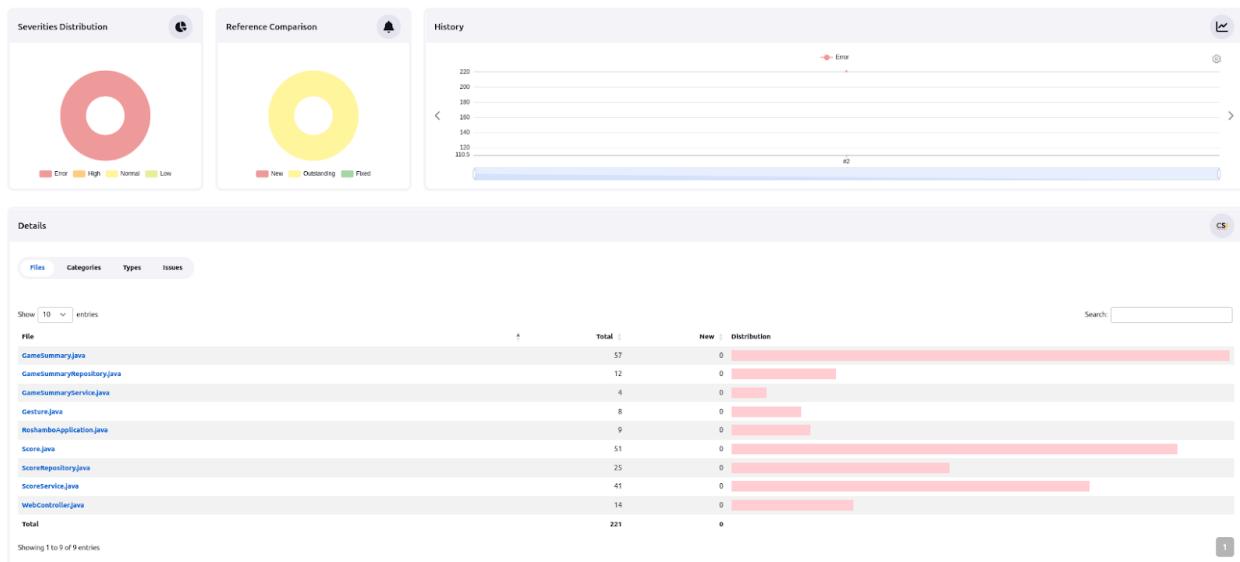


Figure: Exemple de rapport CheckStyle

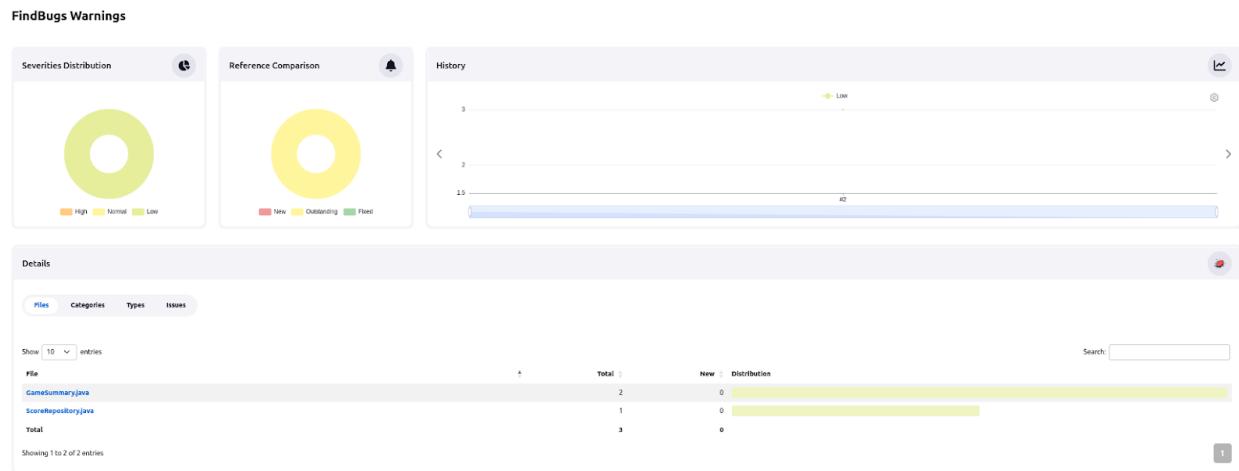


Figure: Exemple de rapport FindBugs

## IV- Automatisation des tests

La section Automatisation des tests sera abordée dans un tutoriel sur le projet maven générer avec jhipster: Une application Java Spring boot possédant une interface web, une base de données.

Dans ce projet on montrera

- Comment mettre en place les tests unitaires et d'intégration.
- Comment gérer les rapports de test grâce au plugin jacoco de maven
- Comment intégrer ses rapport dans un outils tel que sonar
- Comment mesurer la couverture des tests et visualiser son rapport
- On terminera par l'intégration de jmeter pour les test de performance

Toutes ces étapes seront définies dans des stages différents dans le JenkinsFile.

Il faut se référer à l'annexe 3 pour la création du projet maven.

## V- Administration d'un serveur Jenkins

Dans cette section on va faire essentiellement de configuration de jenkins, pour la gestion de sécurité, des utilisateurs et des rôles, la gestion de l'espace disque et le suivi du monitoring du serveur jenkins.

### 1- Activation de la sécurité et mise en place simple.

L'interface d'accueil de jenkins affiche des alerts importantes sur certaines failles de sécurité, l'obsolescence de certains plugins, des alertes qu'il convient de prendre en considération pour protéger son environnement.

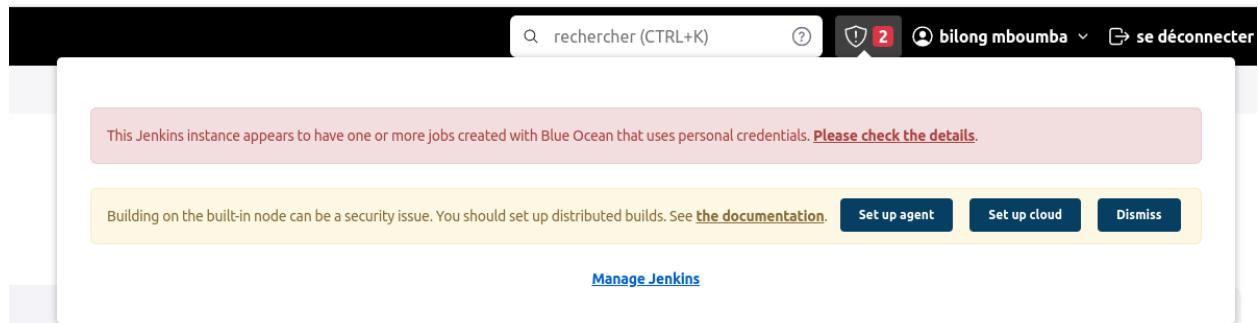


Fig: Icônes d'alerte de jenkins

Dans la page d'administration de jenkins on retrouve les alertes en en-tête de page.

The screenshot shows the Jenkins Administration interface. On the left, there's a sidebar with links like 'Nouveau item', 'Utilisateurs', 'Historique des constructions', 'Relations entre les builds', 'Vérifier les empreintes numériques', and 'Administrer Jenkins' (which is selected). Below that are dropdown menus for 'File d'attente des constructions' and 'État du lanceur de compilations', both currently set to 'Au repos'. The main content area is titled 'Administrer Jenkins' and contains two main sections: 'System Configuration' and 'Security'. Under 'System Configuration', there are four items: 'Configurer le système' (Configure general parameters and file paths), 'Configuration globale des outils' (Configure tools, their location and automatic installers), 'Gestion des plugins' (Manage plugins to enable or disable them), and 'Gérer les nœuds' (Manage nodes to add, remove, control and monitor Jenkins nodes). Under 'Security', there are four items: 'Configurer la sécurité globale' (Secure Jenkins; define who is authorized to access the system), 'Manage Credentials' (Configure credentials), 'Configure Credential Providers' (Configure credential providers and types), and 'Gérer les utilisateurs' (Create/update/delete users who can log in to the server). A red banner at the top right says 'This Jenkins instance appears to have one or more jobs created with Blue Ocean that uses personal credentials. Please check the details.' Below it, a note says 'Building on the built-in node can be a security issue. You should set up distributed builds. See the documentation.' At the bottom right are buttons for 'Set up agent', 'Set up cloud', and 'Dismiss'.

La mise en place de la sécurité est configurée dans le module configurer la sécurité globale. Le menu autorisation permet de limiter les actions utilisateurs:

- **Tout le monde a accès à toutes les fonctionnalités:** Aucune restriction n'est faite sur les autorisations. Tous les utilisateurs ont le contrôle complet de Jenkins, y compris les utilisateurs anonymes qui ne se sont pas authentifiés. Cela est utile dans les situations où vous lancez Jenkins dans un environnement de confiance (comme un intranet d'entreprise) et où vous souhaitez utiliser l'authentification uniquement pour la personnalisation. De cette façon, si quelqu'un a besoin de faire un changement rapide sur Jenkins, cette personne ne sera pas forcée à s'authentifier.
- **Mode Legacy:** Ceci permet un comportement exactement similaire à celui de Jenkins d'avant la version 1.164. C'est-à-dire que, si vous avez le rôle "admin", vous aurez le contrôle complet sur le système. Si vous n'avez pas ce rôle (ce qui est le cas des utilisateurs anonymes), vous n'aurez que l'accès en lecture seule.

- **Les utilisateurs connectés peuvent tout faire:** Dans ce mode, un utilisateur connecté obtient le contrôle complet de Jenkins. Le seul utilisateur qui n'a pas le contrôle complet est l'utilisateur anonymous, qui n'a que l'accès en lecture seule. Ce mode est utile pour forcer les utilisateurs à se logger avant de faire certaines actions, afin de conserver une trace de qui a fait quoi. Ce mode peut également être utilisé dans le cas d'une installation publique de Jenkins, où vous ne pouvez avoir confiance que dans les utilisateurs qui possèdent des comptes.
- **Sécurité basée sur une matrice:** Cette option vous permet de configurer qui fait quoi dans un grand tableau. Chaque colonne représente une autorisation. Faites glisser la souris au dessus du nom d'une autorisation pour obtenir plus d'information sur ce qu'elle représente. Chaque ligne représente un utilisateur ou un groupe (souvent appelé 'rôle', selon les royaumes -realms- de sécurité). On y trouve un utilisateur spécial 'anonymous' qui représente les utilisateurs non authentifiés, ainsi qu'un utilisateur 'authenticated', qui représente les utilisateurs authentifiés (c-à-d, tout le monde, à l'exception des utilisateurs anonymes). Utilisez le texte sous la table pour ajouter des nouveaux utilisateurs/groupes/rôles à la table et cliquez sur l'icône [x] pour les supprimer. Les autorisations s'ajoutent les unes aux autres. En clair, si un utilisateur X est présent dans les groupes A, B et C, alors les autorisations associées à cet utilisateur sont l'union de toutes les autorisations accordées à X, A, B, C et anonymous. (from Matrix Authorization Strategy Plugin)
- **Stratégie d'autorisation matricielle basée sur les projets:** Ce mode est une extension de la "sécurité basée sur la matrice" qui permet de définir une matrice ACL supplémentaire pour chaque projet séparément (ce qui se fait sur l'écran de configuration du travail.) Cela vous permet de dire des choses comme "Joe peut accéder aux projets A, B et C mais il ne peut pas voir D". Voir l'aide de "Sécurité matricielle" pour le concept de sécurité matricielle en

général. Les ACL sont additives, donc les droits d'accès accordés ci-dessous seront effectifs pour tous les projets.

- **Stratégie basée sur les rôles:** Permet la gestion des autorisations par une stratégie basée sur les rôles.

Autorisations

Les utilisateurs connectés peuvent tout faire

|  Allow anonymous read access ?

The screenshot shows a dropdown menu set to "Les utilisateurs connectés peuvent tout faire" (Authenticated users can do anything). Below it is a checkbox labeled "Allow anonymous read access" which is currently unchecked.

Figure : Définitions des autorisations

## 2- Différents types de bases utilisateurs.

Dans jenkins il est possible de provisionner les utilisateurs depuis différentes sources. Pour se faire il faut se rendre dans la sécurité global pour définir son Realm

Royaume pour la sécurité (Realm)

Base de données des utilisateurs de Jenkins

|  Autoriser les utilisateurs à s'inscrire ?

The screenshot shows a dropdown menu set to "Base de données des utilisateurs de Jenkins". Below it is a checkbox labeled "Autoriser les utilisateurs à s'inscrire" which is currently unchecked.

Figure : Choix du fournisseur des utilisateurs

on listera les sources ci-dessous:

- **Base de données des utilisateurs de Jenkins:** Utilise la liste des utilisateurs gérée par Jenkins pour les authentifier, plutôt que de déléguer à un système externe. Cette solution est pratique pour les configurations simples où vous n'avez pas d'utilisateurs dans une base de données ailleurs

- **Déléguer au conteneur de servlets:** Utilise le conteneur de servlet pour authentifier les utilisateurs, comme défini dans les spécifications des servlets. C'est historiquement ce qu'a fait Jenkins jusqu'à la version 1.163. Cela reste utile principalement dans les situations suivantes :
  - Vous avez utilisé Jenkins avant la version 1.164 et vous désirez conserver le même comportement.
  - Vous avez déjà configuré votre conteneur avec le bon royaume (realm) de sécurité et vous souhaitez le faire utiliser par Jenkins. (parfois le conteneur fournit une meilleure documentation ou des implémentations custom pour se connecter à un royaume utilisateur spécifique)
- **LDAP:** Fournissez des configurations pour les serveurs LDAP que Jenkins doit rechercher, au moins une. Plusieurs serveurs avec des configurations différentes peuvent être fournis et Jenkins les interrogera à tour de rôle dans l'ordre configuré. Pour essayer de fournir une certaine forme de cohérence par rapport aux opérations normales ; s'il y a un problème de communication avec l'un des serveurs lors d'une tentative de connexion ou de recherche avec l'un des serveurs configurés, les serveurs suivants ne seront pas essayés, dans le cas où les mêmes noms d'utilisateur existent sur les différents serveurs, les utilisateurs des serveurs de commandes abaissés seront encore être ombragé par ceux d'ordre supérieur.
- **Unix user/group database:** Délègue l'authentification à la base de données des utilisateurs du système Unix sous-jacent. Avec cette configuration, les utilisateurs se connecteront à Jenkins en entrant leurs noms d'utilisateurs et mots de passe Unix. Le mode vous permet également d'utiliser les groupes Unix pour les autorisations. Par exemple, vous pouvez mettre en place des règles du type "tous les membres du groupe 'developers' auront les droits d'accès administrateur." Cela se fait via la bibliothèque 'PAM', qui définit son

propre mécanisme de configuration. Cela marche également avec les extensions de bases utilisateurs telles que NIS.

- None

La définition des utilisateurs dans jenkins se fait au travers du module **“Gérer les utilisateurs”** voir la figure précédente. Il permet de créer des utilisateurs qui vont se connecter directement à Jenkins.

The screenshot shows the Jenkins user creation interface. The top navigation bar includes links for Tableau de bord, Administrer Jenkins, Base de données des utilisateurs de Jenkins, and Crée un utilisateur. The main title is "Créer un utilisateur". The form fields are as follows:

- Nom d'utilisateur: userjava
- Mot de passe: \*\*\*\*\*
- Confirmation du mot de passe: \*\*\*\*\*
- Nom complet: user java
- Adresse courriel: userjava@yopmail.com

A blue "Créer un utilisateur" button is located at the bottom of the form.

Fig: creation d'un user

### 3- Gestion des autorisations et des rôles.

Pour une gestion plus affinée des rôles et les autorisations on utilisera le plugin **Role-based Authorization Strategy**. Pour plus de détails sur le plugin se rendre sur ce lien : <https://plugins.jenkins.io/role-strategy/>

Le plugin est destiné à être utilisé depuis Jenkins pour ajouter un nouveau mécanisme basé sur les rôles pour gérer les autorisations des utilisateurs.

Les Fonctionnalités suivantes sont prises en charge:

- Création de **rôles globaux**, tels qu'administrateur, créateur de job, anonyme, etc., permettant de définir les autorisations Global, Agent, Job, Run, View et SCM sur une base globale.
- Création **d'item rôles**, permettant de définir des autorisations spécifiques à l'item (par exemple, tâche, exécution ou informations d'identification) sur les tâches, les pipelines et les dossiers.
- Création de **rôles d'agent**, permettant de définir des autorisations spécifiques à l'agent.
- Affectation de ces rôles aux utilisateurs et aux groupes d'utilisateurs
- Étendre la correspondance des rôles et des autorisations via les extensions de macro

## Manage Roles

### Global roles

Role	Overall	Credentials	Manage ownership	Agent				Job				Run				View				SCM				
				Create	Delete	Read	Update	Create	Delete	Read	Update	Create	Delete	Read	Update	Create	Delete	Read	Update	Create	Delete	Read	Update	
admin	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>																					
builder			<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																	
creator			<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>																	

Role to add

### Item roles

Role	Pattern	Credentials				Manage ownership				Agent				Job				Run				View				SCM				
		Create	Delete	Read	Update	Create	Delete	Read	Update	Create	Delete	Read	Update	Create	Delete	Read	Update	Create	Delete	Read	Update	Create	Delete	Read	Update	Create	Delete	Read	Update	
folder-access	<input type="checkbox"/> "(?i)folder"																													
folder-builder	<input type="checkbox"/> "(?i)folder/**"																													
folder-creator	<input type="checkbox"/> "(?i)folder/**"																													
@CurrentUserIsOwner	<input type="checkbox"/> "*"																													

Role to add

Pattern

### Node roles

Role	Pattern	Credentials				Manage ownership				Agent				Job				Run				View				SCM				
		Create	Delete	Read	Update	Create	Delete	Read	Update	Create	Delete	Read	Update	Create	Delete	Read	Update	Create	Delete	Read	Update	Create	Delete	Read	Update	Create	Delete	Read	Update	
agent-builder	<input type="checkbox"/> "build-agent"																													

Fig: Définition des rôles et des autorisations

## Attribuer des rôles

Vous pouvez attribuer des rôles aux utilisateurs et aux groupes d'utilisateurs à l'aide de l'écran Attribuer des rôles

- Les groupes d'utilisateurs représentent les autorités fournies par le domaine de sécurité (par exemple, Active Directory ou le plug-in LDAP peut fournir des groupes)
- Il existe également deux groupes intégrés : authentifié (utilisateurs connectés) et anonyme (tout utilisateur, y compris ceux qui ne se sont pas connectés)

- Le survol de la ligne d'en-tête affichera une info-bulle avec les autorisations associées au rôle et au modèle.
- Survoler une case à cocher affichera une info-bulle avec le rôle, l'utilisateur/groupe et le modèle.

## Assign Roles

### Global roles

User/group	admin	builder	creator	?
administrators	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
global-build-user	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
global-creator-user	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
Anonymous	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

User/group to add ?

Add

### Item roles

User/group	folder-builder	folder-creator	?
item-builder-user	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
item-creator-user	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
Anonymous	<input type="checkbox"/>	<input type="checkbox"/>	

User/group to add ?

Add

### Node roles

User/group	agent-builder	?
Anonymous	<input type="checkbox"/>	
item-builder-user	<input checked="" type="checkbox"/>	

User/group to add ?

Add

---

Une fois le plugin installé, on peut voir dans la menu de gestion de la sécurité, le sous-menu gérer et assigner les rôles.

## Gérer et assigner les rôles



[Gérer les rôles](#)

Manage Roles



[Assigner les rôles](#)

Assign Roles



[Role Strategy Macros](#)

Provides info about macro usage and available macros

## 4- Journalisation des actions utilisateurs.

En plus de configurer les comptes utilisateurs et leurs droits d'accès, il peut être utile de garder des actions de chaque utilisateur : en d'autres termes, qui a fait quoi à votre configuration serveur. Ce type de traçage est même requis dans certaines organisations.

Il y a deux plugins Jenkins qui peuvent vous aider à accomplir cela. Le plugin **Audit Trail** conserve un enregistrement des changements utilisateur dans un fichier de log spécial. Et le plugin **JobConfigHistory** vous permet de garder des copies de versions précédentes des diverses configurations de tâches et du système que Jenkins utilise.

Le plugin Audit Trail garde une trace des principales actions utilisateur dans un ensemble de fichiers de logs tournants. Pour mettre cela en place, allez sur la page Gérer les plugins et sélectionnez le plugin **Audit Trail** dans la liste des plugins disponibles. Ensuite, cliquez sur Installer sans redémarrer Jenkins.

La configuration de l'audit trail s'effectue dans la section Audit Trail de l'écran de configuration principal de Jenkins.

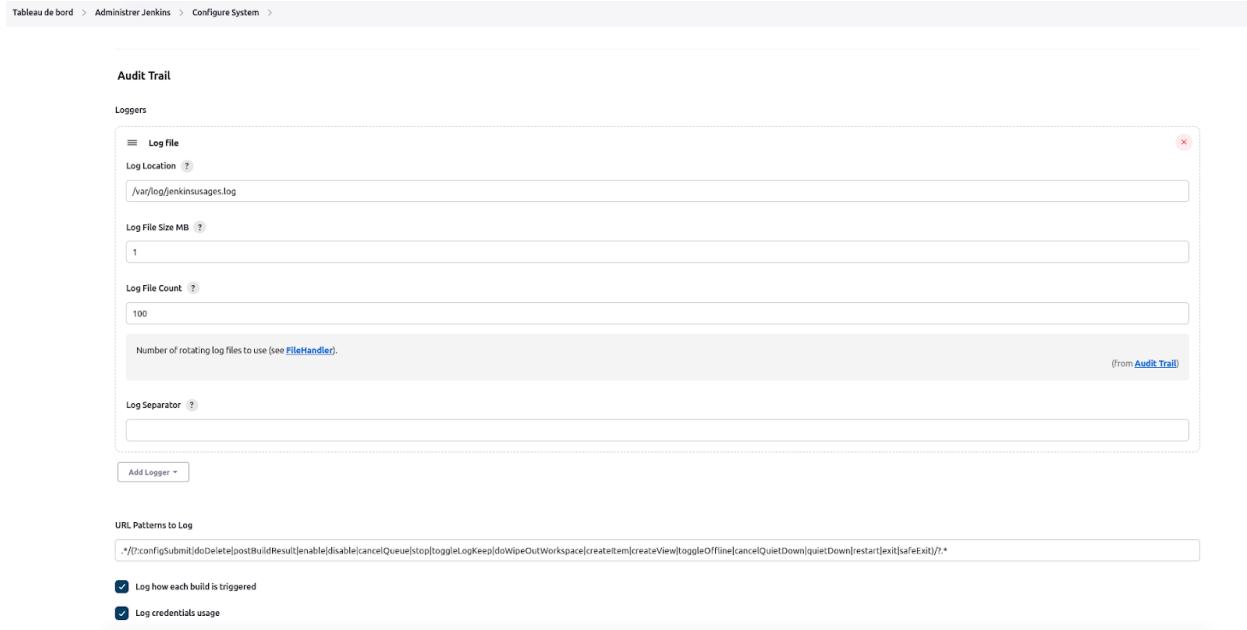


Figure : Configurer le plugin Audit Trail

Le champ le plus important est l'emplacement des logs, qui indique où se trouve le répertoire dans lequel les logs doivent être écrits. L'audit trail est conçu pour produire des logs de style système, qui sont souvent placés dans un répertoire système comme /var/log.

Vous pouvez aussi configurer le nombre de fichiers de logs à maintenir, et la taille maximale (approximative) de chaque fichier. L'option la plus simple est de fournir un chemin absolu (comme /var/log/jenkinsusages.log), auquel cas Jenkins écrira dans des fichiers de logs avec des noms comme /var/log/jenkinsusages.log.1, /var/log/jenkinsusages.log.2, et ainsi de suite.

Bien sûr, vous devez vous assurer que l'utilisateur exécutant votre instance Jenkins peut écrire dans ce répertoire.

## 5- Gestion de l'espace disque.

L'historique des builds prend de l'espace disque. De plus, Jenkins analyse les builds précédents lorsqu'il charge la configuration d'un projet. Ainsi, le chargement d'une tâche avec un millier de builds archivés prendra bien plus de temps qu'une tâche n'en n'ayant que 50. Si vous avez un gros serveur Jenkins avec des dizaines ou des milliers de tâches, le temps total est proportionnellement multiplié.

La façon la plus simple de plafonner l'utilisation de l'espace disque est probablement de limiter le nombre de builds qu'un projet conserve dans son historique.

Cela se configure en cochant la case "Supprimer les anciens builds" en haut de la page de configuration d'un projet voir la figure ci-dessous.

The screenshot shows the Jenkins Pipeline configuration page. Under the 'Strategy' section, 'Log Rotation' is selected. Two fields are visible: 'Nombre de jours de conservation des builds' (5) and 'Nombre maximum de builds à conserver' (20). An 'Avancé...' button is also present.

Figure : Suppression des anciens builds

Si vous dites à Jenkins de ne garder que les 20 derniers builds, il commencera à effacer les plus vieux builds une fois ce nombre atteint. Vous pouvez limiter le nombre d'anciens builds conservés par un nombre de builds ou par date (par exemple les builds de moins de 30 jours). Jenkins fait cela intelligemment: il gardera toujours le dernier build réussi au sein de son historique, ainsi vous ne perdrez jamais votre dernier build réussi.

Le problème avec la suppression des anciens builds est que vous perdez l'historique des builds par la même occasion. Pourtant, Jenkins utilise cet historique pour réaliser différents graphiques sur les résultats des tests et les métriques de build.

Limiter le nombre de build conservé à 20, par exemple, implique que Jenkins affichera des graphiques contenant seulement 20 points. Cela peut être un peu limité. Cette sorte d'information peut être très utile aux développeurs. Il est souvent intéressant de pouvoir afficher l'évolution des métriques sur l'ensemble de la vie du projet, et pas seulement sur les 2 dernières semaines.

Heureusement, Jenkins a un mécanisme à même de rendre les développeurs et les administrateurs systèmes heureux. En général, les éléments prenant le plus de place sont les artefacts de build : fichiers JAR, WAR et ainsi de suite.

L'historique de build en elle-même est principalement constituée de fichiers de log XML, qui ne prennent pas trop de place. Si vous cliquez sur le bouton "Avancé...", Jenkins vous offre la possibilité de supprimer les artefacts mais pas les données du build.

Supprimer les anciens builds ?

**Strategy**

**Log Rotation**

**Nombre de jours de conservation des builds**  
si non vide, les enregistrements de build seront conservés au maximum ce nombre de jours

**Nombre maximum de builds à conserver**  
si non vide, pas plus de ce nombre de builds ne sera conservé

**Nombre de jours durant lesquels conserver les artefacts**  
s'il est renseigné, les artefacts des builds plus anciens que ce nombre de jours seront supprimés, mais les journaux, les rapports, etc ... du build seront conservés  
 7

**Nombre maximum de builds avec artefacts à conserver**  
s'il est renseigné,

Throttle builds ?

Fig : Supprimer les anciens builds - options avancées

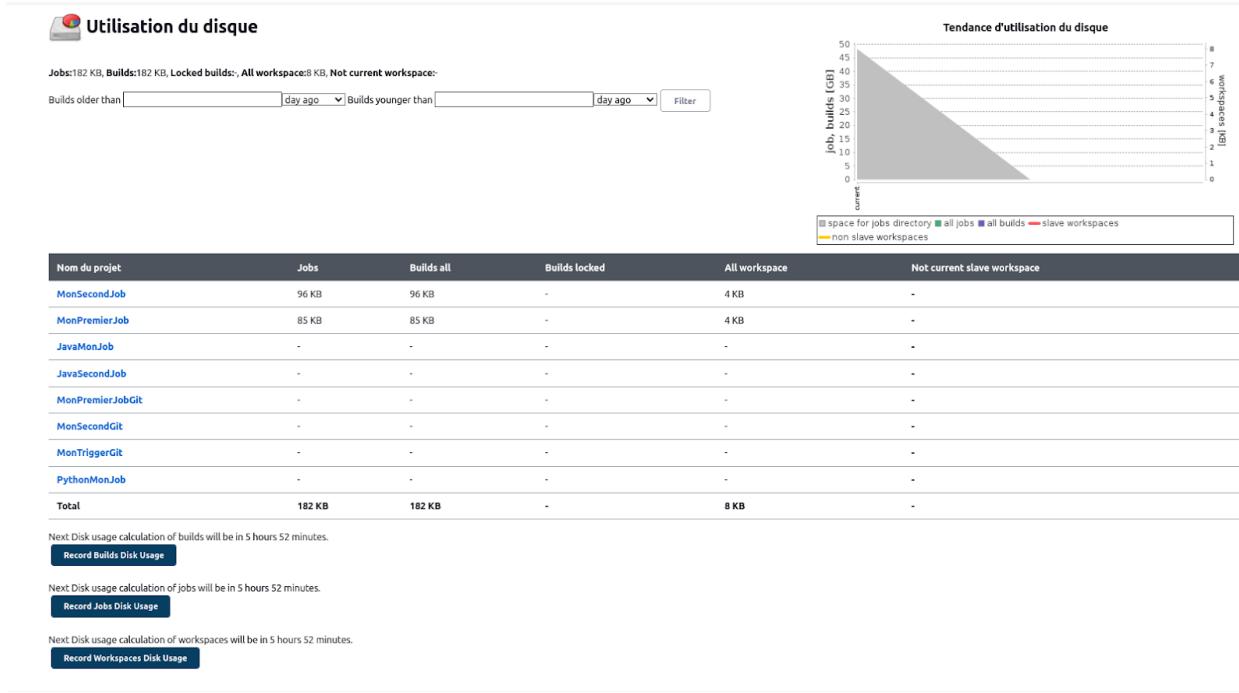
Dans Figure ci-dessus, par exemple, nous avons configuré Jenkins pour qu'il garde les artefacts 7 jours au maximum. Cette option est vraiment pratique si vous avez besoin de limiter l'utilisation du disque tout en désirant fournir l'ensemble des métriques pour les équipes de développement.

N'hésitez pas à être drastique, en gardant un nombre maximal d'artefact assez faible. Souvenez vous que Jenkins gardera toujours le dernier build stable et le dernier réussi, quelque soit sa configuration.

Ainsi, vous aurez toujours au moins le dernier build réussi (à moins bien sûr qu'il n'y ait pas encore eu de build réussi). Jenkins offre également de marquer un build particulier à "Conserver ce build sans limite de temps", afin que certains builds importants ne puissent être supprimés automatiquement.

Le plugin **Disk Usage** est un des plus utiles pour un administrateur Jenkins. Ce plugin conserve et reporte l'espace disque utilisé par vos projets. Il vous permet de repérer et corriger les projets qui utilisent trop d'espace.

Vous pouvez installer le plugin Disk Usage de la façon habituelle, depuis l'écran "Gestion des plugins". Après installation du plugin et redémarrage de Jenkins, le plugin Disk Usage enregistre la quantité d'espace disque utilisée par chaque projet. Il ajoute également un lien "Disk usage" sur la page "Administrer Jenkins". Ce lien vous permet d'afficher la quantité totale d'espace utilisé par vos projets.

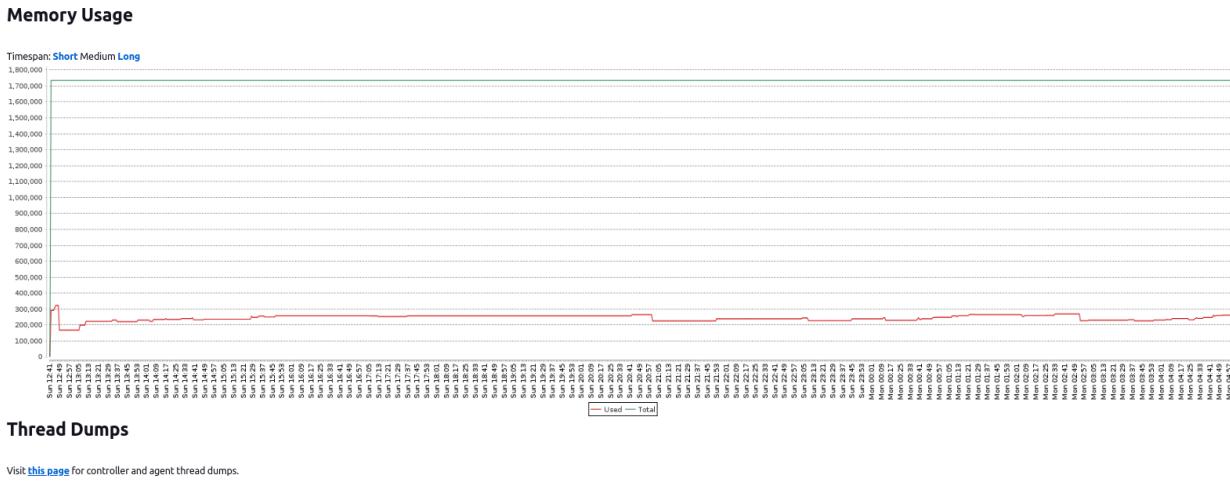


## 6- Monitoring de la charge CPU.

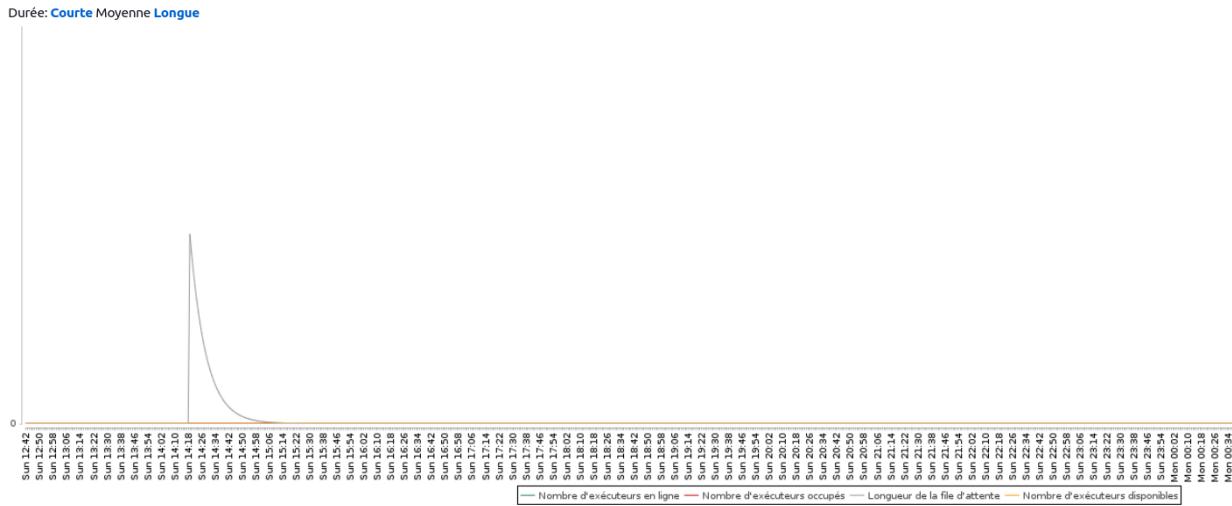
Dans l'administration Jenkins le menu "Status Information" permet de suivre la charge du serveur jenkins.



Le module "Propriétés système" produit un graphique sur l'utilisation de la mémoire, voir figure ci-dessous.



Un autre module permet d'avoir les statistiques de l'utilisation du système



Les statistiques d'utilisation du système permettent de garder trace de trois métriques de mesure de la charge:

**Nombre total d'exéuteurs:** Pour un ordinateur, il s'agit du nombre d'exéuteurs de cet ordinateur. Pour un libellé, cela correspond à la somme des exéuteurs sur tous les ordinateurs possédant ce libellé. Pour Jenkins, il s'agit de la somme de tous les exéuteurs disponibles sur tous les ordinateurs rattachés à cette installation de Jenkins. En dehors des changements de configuration, cette valeur peut également changer quand les agents se déconnectent.

**Nombre d'exéuteurs occupés:** Cette ligne donne le nombre d'exéuteurs (parmi ceux comptés ci-dessus) qui s'occupent des builds. Le ratio de ce nombre avec le nombre total d'exéuteurs donne l'utilisation des ressources. Si tous vos exéuteurs sont occupés pendant une période prolongée, pensez à ajouter plusieurs d'ordinateurs à votre cluster Jenkins.

**Longueur de la file d'attente:** C'est le nombre de jobs qui sont dans la file des builds, en attente d'un exéiteur disponible (respectivement pour cet ordinateur, pour ce libellé ou pour Jenkins en général). Cela n'inclue pas les jobs qui sont dans la période silencieuse (quiet period ou période de délai), ni les jobs qui sont dans la file à cause de builds précédents toujours en cours. Si cette ligne dépasse 0, cela signifie que Jenkins pourra lancer plus de builds en ajoutant des ordinateurs.

Ce graphe est une moyenne glissante exponentielle de données collectées périodiquement. Les périodes de mise à jour sont respectivement toutes les 10 secondes, toutes les minutes et toutes les heures.

## 7- Sauvegarde de la configuration.

Sauvegarder vos données est une pratique universellement recommandée, et vos serveurs Jenkins ne devraient pas y faire exception. Par chance, sauvergarder Jenkins est relativement aisé. Dans cette section nous allons regarder plusieurs façons de réaliser cela.

Dans la plus simple des configurations, il suffit de sauvegarder périodiquement votre dossier **JENKINS\_HOME**.

```

ubuntu@b2-7-flex-sbg5-1:/var/lib/jenkins$ ls
audit-trail.xml
caches
com.cloudbees.hudson.plugins.folder.config.AbstractFolderConfiguration.xml
com.cloudbees.jenkins.plugins.bitbucket.endpoints.BitbucketEndpointConfiguration.xml
config-history  trash
config.xml
credentials.xml
fingerprints
github-plugin-configuration.xml
hudson.model.UpdateCenter.xml
hudson.plugins.build_timeout.global.GlobalTimeOutConfiguration.xml
hudson.plugins.build_timeout.operations.BuildstepOperation.xml
hudson.plugins.emailext.ExtendedEmailPublisher.xml
hudson.plugins.git.GitSCM.xml
hudson.plugins.git.GitTool.xml
hudson.plugins.gradle.injection.InjectionConfig.xml
hudson.plugins.timestamper.TimestamperConfig.xml
hudson.tasks.Mailer.xml
hudson.tasks.Shell.xml
hudson.triggers.SCMTrigger.xml
identity.key.enc
io.jenkins.plugins.junit.storage.JUnitTestResultStorageConfiguration.xml
jenkins.fingerprints.GlobalFingerprintConfiguration.xml
jenkins.install.InstallUtil.lastExecVersion
jenkins.install.UpgradeWizard.state
jenkins.model.ArtifactManagerConfiguration.xml
jenkins.model.GlobalBuildDiscarderConfiguration.xml
jenkins.model.JenkinsLocationConfiguration.xml
jenkins.plugins.git.GitHooksConfiguration.xml

jenkins.security.QueueItemAuthenticatorConfiguration.xml
jenkins.security.ResourceDomainConfiguration.xml
jenkins.security.UpdateSiteWarningsConfiguration.xml
jenkins.security.apitoken.ApiTokenPropertyConfiguration.xml
jenkins.tasks.filters.EnvVarsFilterGlobalConfiguration.xml
jenkins.telemetry.Collector.xml
jobConfigHistory.xml
jobs
logs
nodeMonitors.xml
nodes
org.jenkinsci.plugins.gitclient.GitHostKeyVerificationConfiguration.xml
org.jenkinsci.plugins.github_branch_source.GitHubConfiguration.xml
org.jenkinsci.plugins.github_branch_source.GitHubSCMProbe.cache
org.jenkinsci.plugins.resourcedisposer.AsyncResourceDisposer.xml
org.jenkinsci.plugins.workflow.flow.FlowExecutionList.xml
org.jenkinsci.plugins.workflow.flow.GlobalDefaultFlowDurabilityLevel.xml
org.jenkinsci.plugins.workflow.libs.GlobalLibraries.xml
plugins
queue.xml
scriptApproval.xml
secret.key
secret.key.not-so-secret
secrets
updates
userContent
users
workspace

```

Figure: Jenkins Home /var/lib/jenkins

Il contient la configuration de toutes vos tâches de build, les configurations de vos nœuds esclaves et l'historique des builds. La sauvegarde peut se faire pendant que Jenkins tourne. Il n'y a pas besoin de couper votre serveur pendant la sauvegarde.

L'inconvénient de cette approche est que le dossier **JENKINS\_HOME** peut contenir un volume très important de données. Si cela devient un problème, vous pouvez en gagner un peu en ne sauvegardant pas les dossiers suivants, qui contiennent des données aisément recréées à la demande par Jenkins :

**\$JENKINS\_HOME/war** : Le fichier WAR éclaté

**\$JENKINS\_HOME/cache** : Outils téléchargés

**\$JENKINS\_HOME/tools**: Outils décompressés

Vous pouvez aussi être sélectif concernant ce que vous sauvegarder dans vos tâches de build. Le dossier **\$JENKINS\_HOME/jobs** contient la configuration de la tâche, l'historique des builds et les fichiers archivés pour chacun de vos builds. La structure d'un dossier de tâche de build est présentée dans Figure ci-dessous.

```
ubuntu@b2-7-flex-sbg5-1:/var/lib/jenkins/jobs$ ll
total 44
drwxr-xr-x 11 jenkins jenkins 4096 Dec  5 03:58 .
drwxr-xr-x 20 jenkins jenkins 4096 Dec  5 05:29 Administer Jenkins
drwxr-xr-x  3 jenkins jenkins 4096 Dec  5 04:43 JavaMonJob/
drwxr-xr-x  3 jenkins jenkins 4096 Dec  5 04:00 JavaSecondJob/
drwxr-xr-x  3 jenkins jenkins 4096 Dec  5 05:29 MonPremierJob/
drwxr-xr-x  3 jenkins jenkins 4096 Dec  5 04:43 MonPremierJobGit/
drwxr-xr-x  3 jenkins jenkins 4096 Dec  5 00:24 MonSecondGit/
drwxr-xr-x  3 jenkins jenkins 4096 Dec  5 05:29 MonSecondJob/
drwxr-xr-x  3 jenkins jenkins 4096 Dec  5 00:32 MonTriggerGit/
drwxr-xr-x  3 jenkins jenkins 4096 Dec  5 04:01 PythonMonJob/
drwxr-xr-x  4 jenkins jenkins 4096 Dec  4 13:05 demo/
ubuntu@b2-7-flex-sbg5-1:/var/lib/jenkins/jobs$
```

Fig : Le dossier des builds.

Pour savoir comment optimiser vos sauvegardes Jenkins, vous devez comprendre comment sont organisés les dossiers de tâche de build. Au sein du dossier jobs , il y a un dossier pour chaque tâche de build. Ce dossier contient deux dossiers : builds et workspace . Il n'y a pas besoin de sauvegarder le dossier workspace , vu qu'il sera simplement restauré via une simple récupération si Jenkins constate son absence.

Au contraire, le dossier builds requiert plus d'attention. Il contient l'historique de vos résultats de build et de vos artefacts générés précédemment, avec un dossier horodaté pour chaque build précédent. Si vous n'êtes pas intéressés par la restauration de votre historique des builds ou d'anciens artefacts, vous n'avez pas besoin de sauver ce dossier. Si vous l'êtes, continuez à lire! Dans chacun de ces dossiers, vous trouverez l'historique des builds (stockés sous la forme de fichiers XML, par exemple les résultats des tests JUnit) et les artefacts archivés. Jenkins utilise les fichiers texte et XML pour réaliser les graphiques affichés sur le tableau de bord des tâche de build.

Le dossier archive contient les fichiers binaires ayant été générés et stockés par les builds précédents. Les binaires peuvent vous être importants ou non, mais ils peuvent prendre beaucoup de place. Aussi, si vous les excluez de vos sauvegardes, vous pourriez économiser beaucoup d'espace.

De même qu'il est sage de réaliser des sauvegardes fréquentes, il est également sage de tester votre procédure de sauvegarde. Avec Jenkins, cela est facile à faire. Les répertoires racine de Jenkins sont totalement portables, pour tester votre sauvegarde, il suffit donc de l'extraire dans un dossier temporaire et de lancer une instance Jenkins. Par exemple, imaginons que vous ayez extrait votre sauvegarde dans un dossier temporaire nommé `/tmp/jenkins-backup`. Pour tester cette sauvegarde, assigner le chemin du dossier temporaire à la variable **JENKINS\_HOME** :

- `export JENKINS_HOME=/tmp/jenkins-backup`

Puis démarrer Jenkins sur un port différent et regardez s'il fonctionne :

- `java -jar jenkins.war --httpPort=8888`

Vous pouvez maintenant voir Jenkins tourner sur ce port et vérifier que votre sauvegarde fonctionne correctement.

L'approche décrite dans la section précédente est suffisamment simple pour s'intégrer dans vos procédures normales de sauvegardes, mais vous pourriez préférer quelque chose de plus spécifique à Jenkins. Le **plugin Backup Manager** fournit une interface utilisateur simple que vous pouvez utiliser pour sauvegarder et restaurer vos configurations et données Jenkins.



## Sauvegarde des fichiers de configuration

### Backup configuration

Répertoire racine

/var/lib/jenkins

Répertoire destination ?

Format

Zip

Format du nom de fichier ?

Custom exclusions ?

Mode bavard ?

Configuration files (.xml) only ?

No shutdown ?

### Contenu de la sauvegarde

Sauvegarder le répertoire de travail des jobs

Sauvegarder l'historique des build

Sauvegarder les archives des artifacts maven

Sauvegarder les empreintes des fichiers

# Annexe 1: Installation des outils

## 1- Sous Linux/Ubuntu

- **Visual Studio Code**, Sublime-text
- **Git**: sudo apt install git
- **Git-flow**: sudo apt-get install git-flow
- **Création d'un compte github**:
- **Génération et Ajout de la clé ssh**: ssh-keygen -o -t rsa
- **Maven**: sudo apt install maven
- **Open jdk 8**: sudo apt install openjdk-8-jdk
- **Open jdk 11**: sudo apt install openjdk-11-jdk
  - update-alternatives --list java
  - sudo update-alternatives --config java
- **NodeJs && NPM**: sudo apt install nodejs npm
- **yarn**: sudo apt install yarn
- **Jhispter Generator**: sudo npm install -g generator-jhipster:
- **angular/cli**: npm install -g @angular/cli
- **Docker**:
  - sudo apt-get update
  - sudo apt-get remove docker docker-engine docker.io
  - sudo apt install docker.io
  - sudo systemctl start docker
  - sudo systemctl enable docker
  - docker --version
  - sudo apt install curl
  - sudo curl https://github.com/docker/compose/releases/download/1.24.1/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose
  - sudo chmod +x /usr/local/bin/docker-compose
  - docker-compose --version

## 2- Sous Windows

Installation via scoop(<https://scoop.sh/>) ou choco (<https://chocolatey.org/>):

- <https://github.com/lukesampson/scoop/wiki>
- <https://github.com/lukesampson/scoop/wiki/Quick-Start>
- **Configuration local**
  - \$env:SCOOP='C:\DEV\Tools\scoop'
  - [environment]::setEnvironmentVariable('SCOOP',\$env:SCOOP,'User')
  - exemple : scoop install curl
- **Configuration global:**
  - \$env:SCOOP\_GLOBAL='c:\apps'
  - [environment]::setEnvironmentVariable('SCOOP\_GLOBAL',\$env:SCOOP\_GLOBAL,'Machine')
  - Exemple: scoop install -g pandoc
- **Recherche d'un install**
  - Scoop search maven
  - scoop info maven
- **Installation:**
  - scoop bucket add extras
  - scoop bucket add java
  - scoop install vscode  
<https://code.visualstudio.com/docs/languages/java>
  - scoop install idea (intellij)
  - scoop install eclipse-jee
  - scoop info sublime-text
  - scoop install cmder
  - scoop install git
  - Installation des jdk
    - scoop bucket add java
    - scoop install ojdkbuild8/adopt8-upstream/openjdk11
    - <https://adoptopenjdk.net/>

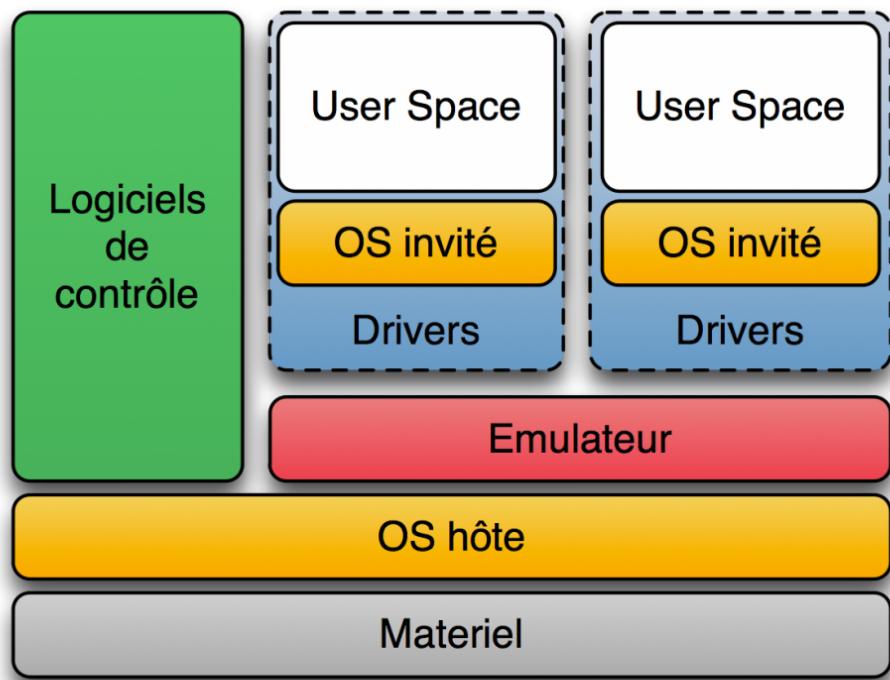
- <https://www.oracle.com/java/technologies/javase-downloads.html>
- scoop install maven
- scoop install nodejs / scoop install nodejs-lts
  - scoop install nvs
  - scoop install nvm
- scoop install yarn
- npm install -g generator-jhipster
- npm install -g @angular/cli
  - <https://cli.angular.io/>
- Docker
  - Boot2docker
  - Docker Toolbox
    - <https://blog.lecacheur.com/2015/10/14/docker-au-revoir-boot2docker-bonjour-docker-toolbox/>
  - Scoop
    - <https://github.com/lukesampson/scoop/wiki/Docker>
  - Docker Desktop
    - <https://docs.docker.com/docker-for-windows/install-windows-home>
    - <https://docs.microsoft.com/en-us/windows/wsl/install-win10>
    - <https://docs.microsoft.com/fr-fr/windows/wsl/wsl2-kernel>
    - <https://docs.docker.com/docker-for-windows/install/>
  - Running docker :
    - docker-compose -f  
src\main\docker\postgresql.yml up -d

## Annexe 2 : Introduction à docker

### 1- Définition

Docker est un **logiciel libre** permettant de lancer des **applications** dans **des conteneurs logiciels**.

C'est donc un outil qui peut **empaqueter une application et ses dépendances** dans un **conteneur isolé**, qui pourra être exécuté sur n'importe quel serveur.



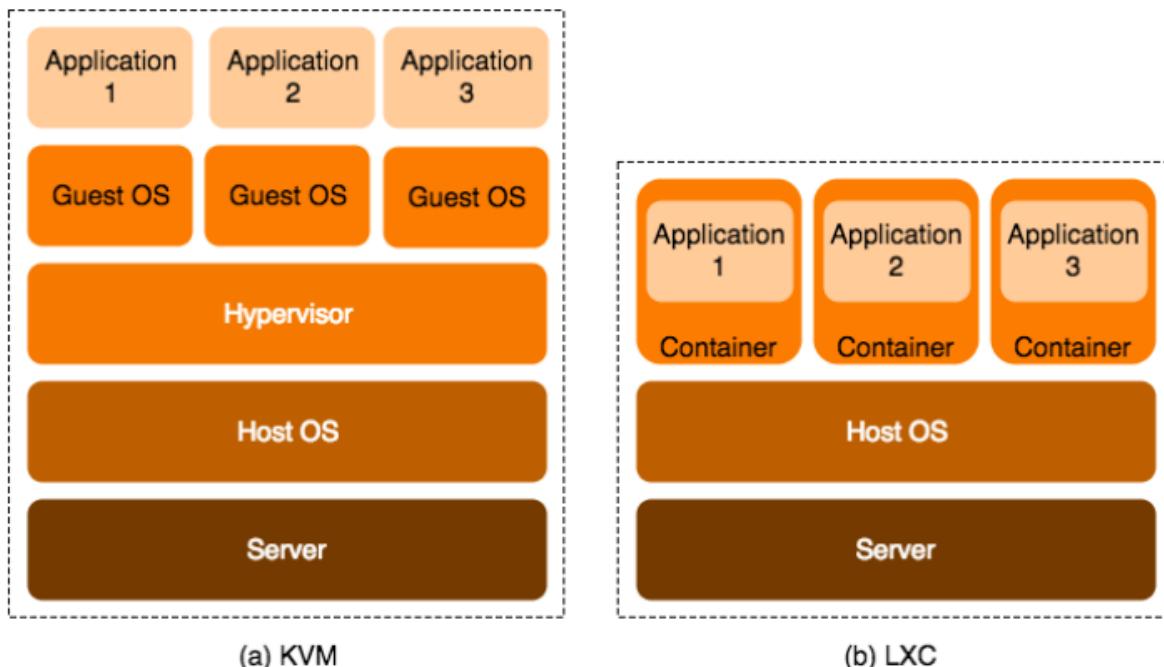
Il ne s'agit pas de **virtualisation**, mais de **conteneurisation**, une forme plus légère qui s'appuie sur certaines parties de la machine hôte pour son fonctionnement.

Cette approche permet d'accroître la **flexibilité** et la **portabilité** d'exécution d'une application, laquelle va pouvoir tourner de façon **fiable** et **prévisible** sur une grande variété de machines hôtes, que ce soit sur la machine **locale**, un **cloud privé** ou **public**, une **machine nue**, etc

Plus généralement, lorsqu'on parle de **conteneur**, on parle de **virtualisation d'OS** qui s'appuie sur le standard de conteneur **linux LXC** (contraction de l'anglais **Linux Containers** est un **système de virtualisation**, utilisant l'**isolation** comme méthode de cloisonnement au niveau du **système d'exploitation**).

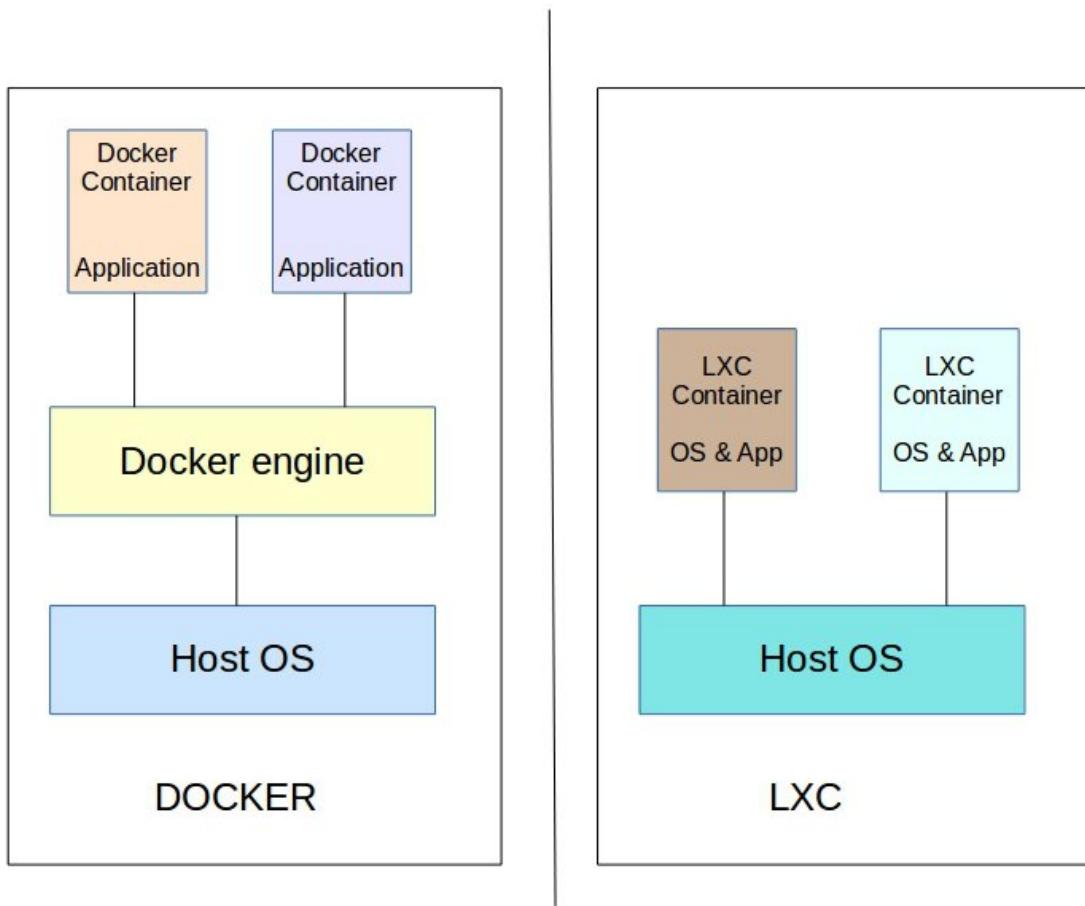
Il est utilisé pour faire fonctionner **des environnements Linux isolés les uns** des autres dans **des conteneurs**, partageant le même noyau et une plus ou moins grande partie du système hôte.

Le conteneur apporte une **virtualisation de l'environnement d'exécution** (processeur, mémoire vive, réseau, système de fichier...) et non pas de la machine.

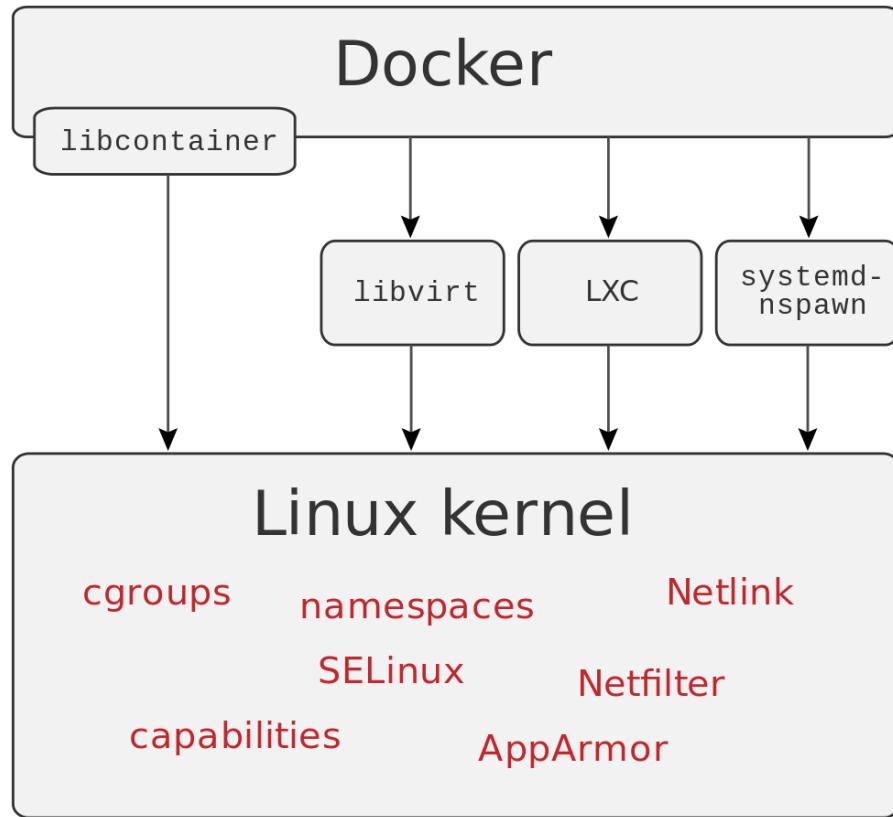


Pour cette raison, on parle de « **conteneur** » et non de « **machine virtuelle** ». au niveau du **système d'exploitation**: contrairement à la **virtualisation** qui émule par logiciel différentes **machines** sur une machine physique, **la conteneurisation émule différents OS sur un seul OS**.

Techniquement, Docker étend le format de conteneur **Linux standard**, **LXC**, avec une **API** de haut niveau fournissant une solution pratique de virtualisation qui exécute les processus de **façon isolée**.



Les Conteneurs docker sont basés sur **Alpine Linux** qui est une **distribution Linux ultra-légère, orientée sécurité et basée sur Musl** (en) et **BusyBox** (est un logiciel qui implémente un grand nombre des commandes standard sous Unix, à l'instar des GNU Core Utilities).



### *Qu'est-ce qu'une image Docker?*

Une image Docker est un fichier immuable, qui constitue une capture instantanée d'un conteneur.

Généralement, les images sont créées avec la commande « docker build ». Et puis, ils vont produire un conteneur quand ils sont lancés avec la commande « run ».

En revanche, dans un registre Docker, les images sont stockées comme « registry.hub.docker.com ».

Comme elles peuvent devenir assez volumineuses, les images sont conçues pour composer des couches de d'autres images, ce qui permet

d'envoyer une quantité minimale de données lors du transfert des images sur le réseau.

Les images sont stockées dans **un registre Docker**, tel que **Docker Hub**, et peuvent être téléchargées à l'aide de la commande **docker pull**.

### ***Qu'est-ce qu'un conteneur Docker?***

Un conteneur Docker est une instance exécutable d'une image. En utilisant l'API ou la CLI de Docker, nous pouvons **créer, démarrer, arrêter, déplacer ou supprimer** un conteneur.

De manière avantageuse, nous pouvons connecter un **conteneur** à un ou plusieurs **réseaux**, y attacher de la **mémoire** ou créer une nouvelle image **sur la base de son état actuel**.

Si on applique le concept de **l'orienté objet**. **Si une image est une classe, un conteneur est une instance d'une classe**, c'est-à-dire un **objet d'exécution**

### ***Conclusion***

Ce que vous devez retenir, c'est que les images sont des instantanés figés de conteneurs vivants. Alors que les conteneurs exécutent les instances d'une image.

<https://www.youtube.com/watch?v=caXHwYC3tq8>

## 2- Les commandes de base à connaître

**docker pull <nom-image>** : téléchargement d'une image docker depuis le repo <https://hub.docker.com/>; c'est la version latest qui est téléchargée

**docker pull <nom-image:versionId>** : Récupère une version spécifique

**docker images** : liste les images télécharger sur son pc

**docker system prune -a** : supprime toutes les images docker

**docker run -d <nom-image>**: création d'une instance docker à partir de l'image

- Télécharge l'image si elle n'est pas sur le disque
- L'option <-d> exécute l'instance en daemon

**docker run -d --name <nom-local> <nom-image>**: création d'une instance docker à partir de l'image avec attribution d'un nom local.

**docker ps**: affiche l'id des instances docker en cours d'exécution

**docker rm --force \$(docker ps -a -q)**: supprime toutes les instances docker en cours d'exécution

**docker exec -it <instance-name>/<instance-id> <shell>**: permet de se loguer dans l'instance docker à partir de l'id ou du nom de l'instance et sur un shell quelconque

**docker stop <instance-name>/<instance-id>** : permet de se loguer dans l'instance docker à partir de l'id ou du nom de l'instance et sur un shell quelconque.

**docker-compose -f <stack.yml> up -d**: exécution multi conteneurs docker à partir d'un fichier YML

***Exemples:***

1. [https://hub.docker.com/\\_/ghost](https://hub.docker.com/_/ghost)
2. <https://github.com/ynovmaster2/docker-intro>

## Annexe 3: Le Générateur de projet web java jhipster

Jhipster est un générateur d'application libre open source très prisé et populaire; utilisé pour le développement d'applications web modernes en utilisant Angular/React/Kotlin et le framework Spring.

C'est donc une plateforme full stack permettant de créer aussi bien des applications monolithiques que des applications respectant l'architecture de micro services .

Il est nativement responsive design et déployable sur le cloud comme en on-premise sur une vm dédiée et facilement scalable via micro services exposant une api rest. C'est donc une application maven complète

1- Génération d'un projet jhipster et déploie sur heroku(provider cloud avec une offre free)

- > création du dossier **jhdemo**
- > lancer la commande jhipster
- > répondez aux questions par défaut en générant un projet monolithique

```
INFO! Using JHipster version installed globally
INFO! Executing jhipster:app


https://www.jhipster.tech

Welcome to JHipster v6.10.4
Application files will be generated in folder: /home/bilonjea/Desktop/testo

Documentation for creating an application is at https://www.jhipster.tech/creating-an-app/
If you find JHipster useful, consider sponsoring the project at https://opencollective.com/generator-jhipster

JHipster update available: 6.10.5 (current: 6.10.4)
Run npm install -g generator-jhipster to update.

? Which *type* of application would you like to create? Monolithic application (recommended for simple projects)
? [Beta] Do you want to make it reactive with Spring WebFlux? No
? What is the base name of your application? prosoccer
? What is your default Java package name? com.pro.soccer.com
? Do you want to use the JHipster Registry to configure, monitor and scale your application? No
? Which *type* of authentication would you like to use? JWT authentication (stateless, with a token)
? Which *type* of database would you like to use? SQL (H2, MySQL, MariaDB, PostgreSQL, Oracle, MSSQL)
? Which *production* database would you like to use? PostgreSQL
? Which *development* database would you like to use? PostgreSQL
? Do you want to use the Spring cache abstraction? Yes, with the Ehcache implementation (local cache, for a single node)
? Do you want to use Hibernate 2nd level cache? Yes
? Would you like to use Maven or Gradle for building the backend? Maven
? Which other technologies would you like to use?
? Which *Framework* would you like to use for the client? Angular
? Would you like to use a Bootswatch theme (https://bootswatch.com/)? Flatly
? Choose a Bootswatch variant navbar theme (https://bootswatch.com/)? Light
? Would you like to enable internationalization support? Yes
? Please choose the native language of the application French
? Please choose additional languages to install English
? Besides JUnit and Jest, which testing frameworks would you like to use? Protractor
? Would you like to install other generators from the JHipster Marketplace? No

Installing languages: fr, en
```

Fig : Installation de jhipster

## Déployer le projet sur github

```
> Création d'un repo sur github et le linker avec son repo local  
>git init  
>git add .  
>git commit -m "first commit"  
>git remote add origin  
git@github.com:login/jhdemo.git  
>git push -u origin master
```

```
> builder le projet  
    > maven clean install  
    > lancer le docker de la base données : docker-compose -f  
src/main/docker/postgres.yml up  
    > lancer l'application jhipster via java -jar  
target/monapplication.jar  
> Intégration du projet sur heroku https://dashboard.heroku.com/  
    > Création d'un nouvelle application exemple : jhdemovm  
    > Sélectionnez la méthode de déploiement github  
    > Autorisez votre compte github sur heroku  
    > Sélectionner l'application jhipster à déployer  
    > cliquez sur connecter  
    > Sélectionnez la branch de déploiement  
    > Cliquez sur Deploy Branch  
    > à la fin du déploiement vous aurez l'url de l'application  
> intégration d'un schéma avec jdl https://www.jhipster.tech/jdl/  
    > jdl studio (https://start.jhipster.tech/jdl-studio/)  
    > télécharger le fichier jdl dans votre projet dans src/main/jdl  
    > générer vos entités avec la commande : jhipster import-jdl  
jhipster-jdl.jh
```

Pour gagner du temps il est également possible de générer le projet jhipster en ligne directement dans son repo github ou gitlab en se rendant ici : <https://start.jhipster.tech/>

## Quelques liens utils

> Présentation de jhipster en 5 min :

<https://www.jhipster.tech/presentation/#/>

> Site officiel : <https://www.jhipster.tech/>

> Un exemple de génération de projet :

<https://www.bearstudio.fr/blog/jhipster-generateur-projet-hipsters>

Ce projet jhipster nous servira de base pour mettre en place les notions de CI/CD (Continuous integration Continuous Deployment) via Jenkins.

Il sera également possible de générer rapidement un petit projet maven spring-boot admin via l'outil spring <https://start.spring.io/>.

## Annexe 4: Ghost (moteur de blog)

Ghost est un moteur de blog libre et open source écrit en JavaScript (NodeJs) et distribué sous licence MIT. Ghost est conçu pour simplifier le processus de publication en ligne par des blogueurs. (wikipedia)

```
# by default, the Ghost image will use SQLite (and thus requires no
separate database container)
# we have used MySQL here merely for demonstration purposes
# especially environment-variable-based configuration)

version: '3.1'

services:

ghost:
  image: ghost:3-alpine
  restart: always
  ports:
    - 2368:2368
  environment:
    # see
    https://docs.ghost.org/docs/config#section-running-ghost-with-config-env-variables
    database_client: mysql
    database_connection_host: db
    database_connection_user: root
    database_connection_password: example
    database_connection_database: ghost
    # this url value is just an example, and is likely wrong for your
    environment!
  url: http://localhost:2368
```

```
db:  
  image: mysql:5.7  
  restart: always  
  environment:  
    MYSQL_ROOT_PASSWORD: example
```