

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

Sommaire

Objectifs pédagogiques	4
A- Où trouver les ressources du cours	4
Pré-requis	5
I - Introduction à l'Intégration Continue (IC)	6
1. Pourquoi fait-on de la CI ? (au-delà du "il nous faut Jenkins")	6
2. Où se place Jenkins dans l'écosystème CI/CD ?	6
3. Modèle d'exécution : contrôleur/agents... et agents dynamiques	7
3.1. Le modèle	7
3.2. Agents dynamiques (Docker & Kubernetes)	7
4. Pipeline as Code : Groovy... et YAML (quoi de neuf vraiment ?).....	7
5. Flux CI/CD moderne - du commit à la prod (8 étapes).....	9
6. Branching et règles de merge (ce qui marche en vrai).....	9
7. Tests : quoi, pourquoi, où dans le pipeline ?.....	9
8. Qualité & sécurité.....	10
9. 12-Factor Apps (on garde l'esprit, on actualise)	10
10. Mesurer sans fliquer : DORA metrics	11
11. Environnements & artefacts — fin du "ça marche chez moi"	11
12. Habitudes d'équipe (les vrais différenciateurs)	11
13. Check-list "7 jours" (déploiement progressif de la CI)	12
14. Anti-patterns / Bonnes pratiques (mémo)	12
15. Glossaire express	12
16. Étude de cas — Monorepo polyglotte (Java + Angular).....	13
II - Mise en place & automatisation du build	14
1. Un bref rappel historique	14
2. Installation de Jenkins	15
Option A - Docker Compose minimal (DoD pour dev)	16
Option B - Image custom (plugins préinstallés)	16
Option C - JCasC (Configuration as Code).....	17
3 - Lancement de jenkins via le navigateur	17

4. Configuration : page principale de la configuration, configuration de Git/SVN, serveur de Mail.	22
A- Création de notre premier job en mode free style.....	22
B- Exécution de mon premier job.....	25
C- Automatisation du déploiement : Démo	27
5. Stratégies et techniques de notification.	27
6. Fixer les dépendances entre les travaux de Build.	30
7. Jenkins et Maven : rappel sur Maven, configuration du Build Maven, déploiement dans un repository Maven.....	32
A. vocabulaire	33
B. Plugins et buts	33
C. Buts et phases	34
D. Etapes du cycle de vie Maven par défaut	34
E. Présentation fichier pom.xml : Project Object Model.....	36
8. Jenkins et le Build, les meilleures pratiques et méthodes recommandées.....	42
III - Qualité du code	43
1. Introduction, intégration de la qualité dans le processus de build.....	43
2. Outils d'analyse : Checkstyle, FindBugs, CPD/PMD.....	43
3. Configuration du rapport qualité avec le plugin Warning next generation.....	44
4. Rapport de complexité, sur les tâches ouvertes.	46
IV - Automatisation des tests	48
V - Administration d'un serveur Jenkins.....	49
1. Activation de la sécurité et mise en place simple.	49
2. Différents types de bases utilisateurs.	51
3. Gestion des autorisations et des rôles.	53
4. Journalisation des actions utilisateurs.	56
5. Gestion de l'espace disque.....	57
6. Monitoring de la charge CPU.....	59
7. Sauvegarde de la configuration.....	61
Annexes	65
Annexe 1: Installation des outils	65
1. Sous Linux/Ubuntu/WSL.....	65
2. Sous Windows	68
Annexe 2 : Introduction à docker	70
1. Définition.....	70

2. Les commandes de base à connaître	73
Annexe 3: Le Générateur de projet web java jhipster	75
1. Génération d'un projet jhipster et déploie sur heroku(provider cloud avec une offre free) ...	75
Déployer le projet sur github	77
Annexe 4: Ghost (moteur de blog)	79
Annexe 5 Mise à jour Jenkins 2025	81
1. Évolutions Jenkins (2023–2025)	81
2. Qualité & Analyses modernes	81
3. Intégration Sonar.....	82
4. Artefacts & Registries	82
5. Déploiement Cloud.....	82
6. Administration moderne	83

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 du cours se trouvent sur le dépôt github suivant :

<https://github.com/bilonjea/jenkins-101>

Pour récupérer les sources en local, lancez la commande suivante pc: `git clone git@github.com:bilonjea/cargo-tms.git` par exemple.

Vous pouvez forker ou cloner les deux repos

Pré-requis

Pour aborder sereinement les modules de ce cours, une bonne maîtrise de **Java** et des notions de base du **cycle de développement** (dont **Maven**) sera un atout important, sans constituer pour autant un frein majeur. La connaissance des outils suivants est un plus :

- Un environnement **Linux** (idéalement : **Ubuntu** ou **Fedora**).
- Des notions de **Git** et, si possible, un compte **GitHub** (github.com).
- Être familier avec les notions de **branche** (*branch*) et de **dépôt** (*repository*).
- Des notions de **Docker** (voir **Annexe 2**)

I - Introduction à l'Intégration Continue (IC)

1. Pourquoi fait-on de la CI ? (au-delà du “il nous faut Jenkins”)

Intégration Continue (CI) = intégrer des changements **petits et fréquents** dans une branche partagée et **vérifier automatiquement** qu'ils n'introduisent pas de régressions.

Objectifs concrets :

- **Feedback rapide** (idéalement < 10 min sur une PR) pour que le dev corrige tant que le contexte est frais.
- **Stabiliser la branche principale** (verte par défaut).
- **Réduire le risque** en réduisant la **taille des lots** (principe fondamental : petits lots = moins d'inconnues).
- **Accélérer la livraison** et fiabiliser la qualité.

Avoir Jenkins ≠ faire de la CI. La CI est une **discipline d'équipe** : petits commits, PR systématiques, checks automatiques, critères de merge partagés.

2. Où se place Jenkins dans l'écosystème CI/CD ?

Rôle. Jenkins est un **orchestrateur** : il déclenche et enchaîne *build* → *tests* → *analyses* → *publication* → *déploiement*.

Forces. Ultra-**modulaire** (plugins), **polyglotte** (Java/Node/Python/Go...), **interopérable** (SCM, Sonar, Nexus, Jira, Slack/Discord...).

Concurrents. GitHub Actions, GitLab CI, Azure DevOps, CircleCI.

Quand Jenkins reste pertinent en 2025 ?

- On veut **maîtriser l'infra** (on-prem/hybride),
- On mélange **plusieurs SCM et stacks**,
- On capitalise sur un écosystème de **plugins** et d'**intégrations** déjà éprouvé.

3. Modèle d'exécution : contrôleur/agents... et agents dynamiques

3.1. Le modèle

- **Contrôleur (controller)** : planifie, orchestre, expose l'UI/API.
- **Agents (workers)** : exécutent les étapes.
- **Bonne pratique** : **0 exécuteur sur le contrôleur** en prod ; *tout* s'exécute sur des agents.

3.2. Agents dynamiques (Docker & Kubernetes)

- **Agents Docker dynamiques** : avec le cloud Docker du plugin, Jenkins lance **un conteneur agent éphémère par build** (image = outillage), puis le détruit.
- **Agents Kubernetes (pods)** : **très courant en 2024–2025** ; le plugin K8s crée **un Pod par build** (souvent multi-conteneurs : maven, node, etc.).
Bénéfices : charge **déportée** du contrôleur, **scalabilité** automatique, **reproductibilité** (versions d'outils figées dans l'image), **coûts maîtrisés**.

En 2025, **beaucoup d'entreprises exécutent Jenkins dans Kubernetes** avec des **pods agents dynamiques**. Alternative simple sans K8s : agents Docker dynamiques sur un hôte Docker.

4. Pipeline as Code : Groovy... et YAML (quoi de neuf vraiment ?)

- **Référence officielle & majoritaire** : **Pipeline Déclaratif (DSL Groovy)** via un **Jenkinsfile** versionné dans le repo.
- **Modernisation côté YAML** : il existe un **plugin “Pipeline as YAML” (incubating)** qui permet d'écrire le pipeline en **YAML** ; **Jenkins convertit ce YAML en Pipeline Déclaratif à l'exécution**.
 - Intérêt : homogénéiser avec des organisations qui standardisent déjà leurs pipelines en YAML (Actions/GitLab).
 - **Réalité 2025** : le **Jenkinsfile Groovy reste la norme** en prod ; le **YAML est possible** mais encore **optionnel**.

Exemple minimal (Déclaratif Groovy) :

```
pipeline {

    agent any

    stages {

        stage('Build') { steps { sh './mvnw -B clean verify' } }

        stage('Tests') { steps { junit 'target/surefire-reports/*.xml' } }

        stage('Qualité') {

            steps {

                recordIssues tools: [spotBugs(pattern: '**/spotbugsXml.xml')]

                recordCoverage tools: [jacoco(pattern: '**/jacoco.xml')]

            }
        }
    }
}
```

Exemple minimal (YAML via plugin) :

```
agent: { any: true }

stages:

- stage: Build

    steps: [ { sh: "./mvnw -B clean verify" } ]

- stage: Tests

    steps: [ { junit: "target/surefire-reports/*.xml" } ]
```

À retenir : **Groovy = standard** ; **YAML = option** qui peut faciliter l’alignement culturel si l’équipe “pense YAML”

5. Flux CI/CD moderne - du commit à la prod (8 étapes)

Idée générale : à chaque changement de code, on cherche un **retour rapide et fiable**, puis on **publie** ce qui a été testé et on **déploie** de façon répétable.

1. **Planifier** : tickets clairs, *Definition of Done* incluant qualité & sécurité.
2. **Coder** : **branches courtes** (heures/jours), PR systématique, conventions de message (PROJ-123: ...).
3. **Construire** : builds **reproductibles** (versions pinnées, *lockfiles*, caches maîtrisés).
4. **Tester** : pyramide (unitaires >> intégration > E2E), feedback PR < **10 min** si possible.
5. **Analyser** : **Checkstyle/PMD/SpotBugs + Warnings NG, JaCoCo (XML), Sonar** (Quality Gate **bloquante**).
6. **Packager/Publier** : artefacts **immutables** (Nexus/JFrog/GitHub Packages), **SNAPSHOT** vs **release** (tag).
7. **Déployer** : staging/prod via VM (systemd+Nginx), containers (Jib/Buildx), Cloud Run/ECS/K8s.
8. **Observer/Améliorer** : logs/metrics/APM + **DORA metrics** (cf. §10).

6. Branching et règles de merge (ce qui marche en vrai)

But : garder une **branche principale verte** et un historique **lisible**.

- **Trunk-Based** recommandé (ou GitFlow **allégé**).
- **PR checks obligatoires** : build OK, tests OK, **Quality Gate Sonar OK**, 1–2 reviews.
- **Rebase/squash** pour un historique lisible.
- **Nom de branche** incluant l'**issue** (ex. CGO-124-ajout-endpoint).

Anti-pattern : “On merge rouge parce que *ça marche chez moi*.” — **Non**. La main doit rester **verte**.

7. Tests : quoi, pourquoi, où dans le pipeline ?

Idée : détecter tôt, au **meilleur coût**.

- **Unitaires** : rapides, isolés (mocks), ROI maximal → **obligatoires en PR**.
- **Intégration** : couvre les **chemins critiques** (DB, broker, API).
- **E2E** : peu nombreux, **valeur de validation** (coûteux → sobriété).
- **Contrats (API-first)** : valider l'**accord** entre producteurs/consommateurs (OpenAPI + générateurs).
- **Non-fonctionnels** : *smoke*, perfis légères, sécurité (SCA), selon contexte.

Placement : tests **rapides** en PR ; campagnes plus lourdes **après merge** (nightly).

8. Qualité & sécurité

Objectif : une **qualité visible**, sans seuils irréalistes.

- **Static analysis** : SpotBugs/PMD/Checkstyle → **Warnings NG** (vue unifiée, tendances, heatmaps).
- **Couverture** : JaCoCo (XML) ; viser une **couverture utile** (ex. 80 % sur le code métier).
- **SonarQube/SonarCloud** : bugs, vulnérabilités, code smells, duplications, couverture → **Quality Gate bloquante**.
- **Sécurité** : SCA (dépendances vulnérables), **secret scanning**, SBOM pour tracer ce qu'on déploie.

Anti-pattern : viser 95 % de couverture du jour au lendemain → **tests cosmétiques**. Mieux vaut cibler les **zones à risque**.

9. 12-Factor Apps (on garde l'esprit, on actualise)

Les **12-Factors** restent une **boussole** pour des apps faciles à builder, déployer, faire tourner et diagnostiquer :

1. **Codebase unique** (Git), 2. **Dépendances** explicites (pom.xml/package.json), 3. **Config par variables** (pas en dur),
2. **Services externes** (DB, cache) comme ressources attachées, 5. **Build/Release/Run séparés**, 6. **Stateless**,
3. **Port binding** (expose ton HTTP), 8. **Concurrence** par process, 9. **Discardable** (démarrage/arrêt rapides),

4. **Parité dev/prod** (mêmes versions d'outils), 11. **Logs** en flux (stdout), 12. **Admin** en tâches uniques.

Compléments 2025 : SBOM, secrets hors dépôt (vault/credentials Jenkins), scans de sécurité “shift-left”.

10. Mesurer sans fliquer : DORA metrics

But : repérer les goulots et progresser, pas surveiller les individus.

- **Lead time for changes** (commit → prod), **Deployment frequency**, **Change failure rate, MTTR**.
But : suivre les **tendances** (équipe) pour trouver les goulots (build trop lent, tests flakies, gate bruyante).
Jamais en outil de sanction individuelle.

11. Environnements & artefacts — fin du “ça marche chez moi”

Principe : ce qu'on **teste** est ce qu'on **déploie** (même binaire).

- **Parité** dev/staging/prod (mêmes runtimes/flags).
- **Artefacts immutables** : ce qui a été testé est ce qui est déployé (même JAR, même image).
- **Registries** : Nexus/JFrog/GitHub Packages (SNAPSHOT vs release).
- **Tags = versions** ; générer un **changelog** à partir des PR/commits.

12. Habitudes d'équipe (les vrais différenciateurs)

Idée : discipline légère, bénéfices forts.

- **Petits incrément**s, merge fréquent, revues bienveillantes et concrètes.
- **Pipelines lisibles** (noms de stages explicites, logs propres, timestamps/couleurs).
- **Post-mortems sans blâme** ; on **outille** après chaque incident.
- **Automatiser** ce qui est pénible (release notes, versions, changelogs).

13. Check-list “7 jours” (déploiement progressif de la CI)

But : passer à une CI utile **sans big bang**.

- **J1** : Jenkins LTS + Java 21 ; contrôleur **0 exécuteur** ; un agent java21.
- **J2 : PR checks** (build rapide + tests + JaCoCo + Warnings NG).
- **J3** : Sonar + **Quality Gate** bloquante.
- **J4** : Artefacts (Nexus/GitHub Packages), mvn deploy / npm publish.
- **J5** : Déploiement staging (VM systemd + Nginx ou container avec Jib).
- **J6** : Notifications (Slack/Discord) ; **Smart commits Jira** (PROJ-123 #comment ...).
- **J7** : Agents **dynamiques** Docker/K8s ; premiers dashboards DORA.

14. Anti-patterns / Bonnes pratiques (mémo)

À éviter :

- PR énormes, merge rouge, seuils irréalistes, jobs freestyle partout, secrets dans le code.
- À faire :
- **Pipeline as Code** (Groovy **ou** YAML plugin), **pinner** les plugins Jenkins, suivre **tendances** (Warnings NG, couverture, Sonar), caches Maven/npm sur agents.

15. Glossaire express

Pipeline : chaîne automatisée (build/test/qualité/publish/deploy).

Agent : machine/conteneur qui exécute.

Quality Gate : seuil Sonar bloquant.

SBOM : liste des dépendances d'un binaire.

SCA : scan de vulnérabilités des libs.

Agents dynamiques : agents **éphémères** (Docker/K8s) créés à la demande.

16. Étude de cas — Monorepo polyglotte (Java + Angular)

De quoi parle-t-on ?

Un **monorepo** contenant : backend/ (Java multi-modules), sb-admin/ (Java), contract-api/ (OpenAPI pour générer DTO/clients), frontend/ (Angular). Aujourd’hui, les **builds PR sont lents** (~25 min), **main** devient parfois **rouge**, et le **déploiement** sur la VM se fait **manuellement**.

But (simple et concret)

- Obtenir un **feedback PR rapide** ($\approx < 10$ min).
- Garder **main toujours verte** grâce à une **Quality Gate Sonar bloquante**.
- **Publier** automatiquement les **artefacts** (Maven/npm).
- **Déployer** automatiquement en **staging** (VM : JAR via systemd + dist Angular via Nginx).

Ce qu'on change (en pratique)

1. **Pipeline PR (rapide)** : build sélectif par dossier, tests unitaires, Warnings NG + JaCoCo, analyse Sonar → **Gate bloquante**. *Pas* de déploiement ni de publication ici.
2. **Pipeline main (complet)** : build + tests + qualité → mvn deploy (Nexus/GitHub Packages) → **déploiement staging** (scp + systemd + rsync/Nginx).
3. **Agents dynamiques** (Docker ou Kubernetes) : **un agent par build**, éphémère → moins d’attente, outils figés dans l’image (Maven/Node).
4. **API-first** : si contract-api/ change, on **régénère les clients** (Java & TS) et on **rebuilt** backend/front automatiquement dans le même run.

Pourquoi c'est utile ?

- Le **dev a une réponse vite** sur sa PR (utile pour corriger tout de suite).
- On **n'intègre que du vert** (la Gate bloque le mauvais code avant merge).
- Les artefacts sont **tracables et immuables** (ce qu'on teste = ce qu'on déploie).
- Le déploiement staging devient **répétable et sans clics**.

Avant / Après (attendu)

- **Avant** : PR ~25 min, merges parfois rouges, déploiements manuels risqués.
- **Après** : PR ~9–12 min, **main toujours verte**, **publication et staging automatisés**, équipe plus sereine

II - Mise en place & automatisation du build

Jenkins, en deux mots :

Jenkins est un **serveur d'automatisation open source**. Il permet d'**orchestrer** les étapes du cycle de développement : **build, tests, analyses de qualité, publication d'artefacts et déploiements**, au service de l'**intégration continue (CI)** et de la **livraison continue (CD)**.

- **Technologie** : écrit en **Java**, Jenkins s'exécute soit en **mode autonome** (avec un serveur web/servlet embarqué), soit **dans un conteneur de servlets** tel qu'**Apache Tomcat**.
- **Intégrations SCM** : fonctionne avec **Git** (majoritaire aujourd'hui), et reste compatible avec **Subversion** (héritage).
- **Build tools** : support natif d'**Apache Maven** et **Gradle**, exécution de scripts **shell** (Linux/macOS) ou **batch/Powershell** (Windows).
- **Écosystème** : plus de **1000 plugins** pour brancher des outils tiers (qualité, sécurité, artefacts, cloud, notifications, Jira, Docker, Kubernetes, etc.).
- **Déclencheurs** : **webhooks** (PR/commit), planification type **cron**, **dépendances** entre jobs, ou **appel d'URL/CLI**.
- **Usage** : interface **web** complète, **CLI** pour l'automatisation, et surtout **Pipeline as Code** (`Jenkinsfile`) versionné avec le projet.

En 2025 : Jenkins **LTS** fonctionne avec **Java 21**, pousse le **Pipeline as Code**, l'exécution sur **agents dynamiques** (Docker/Kubernetes), et offre une option **YAML** via plugin (le `Jenkinsfile Groovy` restant la référence).

1. Un bref rappel historique

- À l'origine, Jenkins s'appelait **Hudson**. En **2011**, suite à un différend de gouvernance et de marque, la communauté open source a **renommé** le projet **Jenkins**.
- La branche **Hudson** maintenue un temps par Oracle a fini par **s'éteindre** (obsolète depuis **2017**).
- Dès **2011–2012**, Jenkins remplace progressivement **CruiseControl** comme référence CI open source.

- En **avril 2016**, la **version 2.0** sort avec le plugin **Pipeline** activé par défaut : on décrit les builds en **DSL Groovy** (`Jenkinsfile`) — c'est le **tournant Pipeline as Code**.
- De 2018 à 2025, l'écosystème s'étoffe : **Warnings NG** (qualité locale moderne), **Sonar** (Quality Gate), **agents Docker/Kubernetes**, **JCasC** (Configuration as Code), et **Java 21** côté runtime.

En synthèse : Jenkins est un **ordonnanceur (scheduler)** qui **automatise, planifie, enchaîne et vérifie** les tâches. On y crée des **jobs** et surtout des **pipelines** (CI/CD) pour garantir des livraisons **prévisibles et répétables**.

Ses plugins se **posent** en force (richesse) comme en **responsabilité** (maintenance versionnée). L'outil s'administre via l'**UI**, la **CLI**, et de plus en plus via des fichiers **as code** (JCasC, `Jenkinsfile Groovy`, option YAML).

À retenir (2025)

- **Controller sans exécuteur + agents dynamiques** (Docker/K8s) = élasticité & isolation.
- **Jenkinsfile** (Groovy) reste **le standard** ; **YAML** existe via plugin (converti en pipeline déclaratif).
- **Déclencheurs** : favorise les **webhooks SCM** (plus fiables que le polling).
- **Plugins** : **pinner** les versions (fichier `plugins.txt`) et **tester** avant prod.

2. Installation de Jenkins

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 à la page :
<https://www.jenkins.io/doc/book/installing/>

En 2025, on recommande:

- **Jenkins LTS 2.479+** avec **Java 21**.
- Déploiement **containerisé** (Docker/compose) pour isoler, versionner et reproduire.

Option A - Docker Compose minimal (DoD pour dev)

Pour un labo/formation locale (simple) :

```
services:  
  
  jenkins:  
  
    image: jenkins/jenkins:lts-jdk21  
  
    user: "1000:1000"  
  
    ports: ["8080:8080", "50000:50000"]  
  
    volumes:  
  
      - jenkins_home:/var/jenkins_home  
  
        - /var/run/docker.sock:/var/run/docker.sock # DoD: le contrôleur peut  
          piloter Docker hôte  
  
  volumes:  
  
    jenkins_home:
```

- Premier démarrage : récupérer la clé d'init dans .../secrets/initialAdminPassword (ou docker logs) puis installer les plugins conseillés (ou utiliser une image custom avec plugins.txt pinné).
- Sécurité : pour la prod, éviter DoD (risque sécurité), préférer agents dédiés.

Option B - Image custom (plugins préinstallés)

Recommandée pour gagner du temps et geler une version stable de plugins.

```
FROM jenkins/jenkins:lts-jdk21  
  
USER root  
  
COPY plugins.txt /usr/share/jenkins/ref/plugins.txt  
  
RUN apt-get update && apt-get install -y --no-install-recommends curl ca-certificates && rm -rf /var/lib/apt/lists/*  
  
RUN jenkins-plugin-cli --plugin-file /usr/share/jenkins/ref/plugins.txt  
  
USER jenkins
```

plugins.txt (extrait, versions à pinner) :

```
workflow-aggregator:...
pipeline-stage-view:...
git:...
warnings-ng:...
checkstyle:...
pmd:...
spotbugs:...
jacoco:...
sonar:...
configuration-as-code:...
credentials-binding:...
```

Option C - JCasC (Configuration as Code)

Pour **industrialiser** la conf (sécurité, credentials, outils, agents) :

```
jenkins:
    systemMessage: "Jenkins configuré via JCasC"
    numExecutors: 0
    unclassified:
        location:
            url: "https://jenkins.exemple.local"
```

3 - Lancement de jenkins via le navigateur

> 127.0.0.1:8080

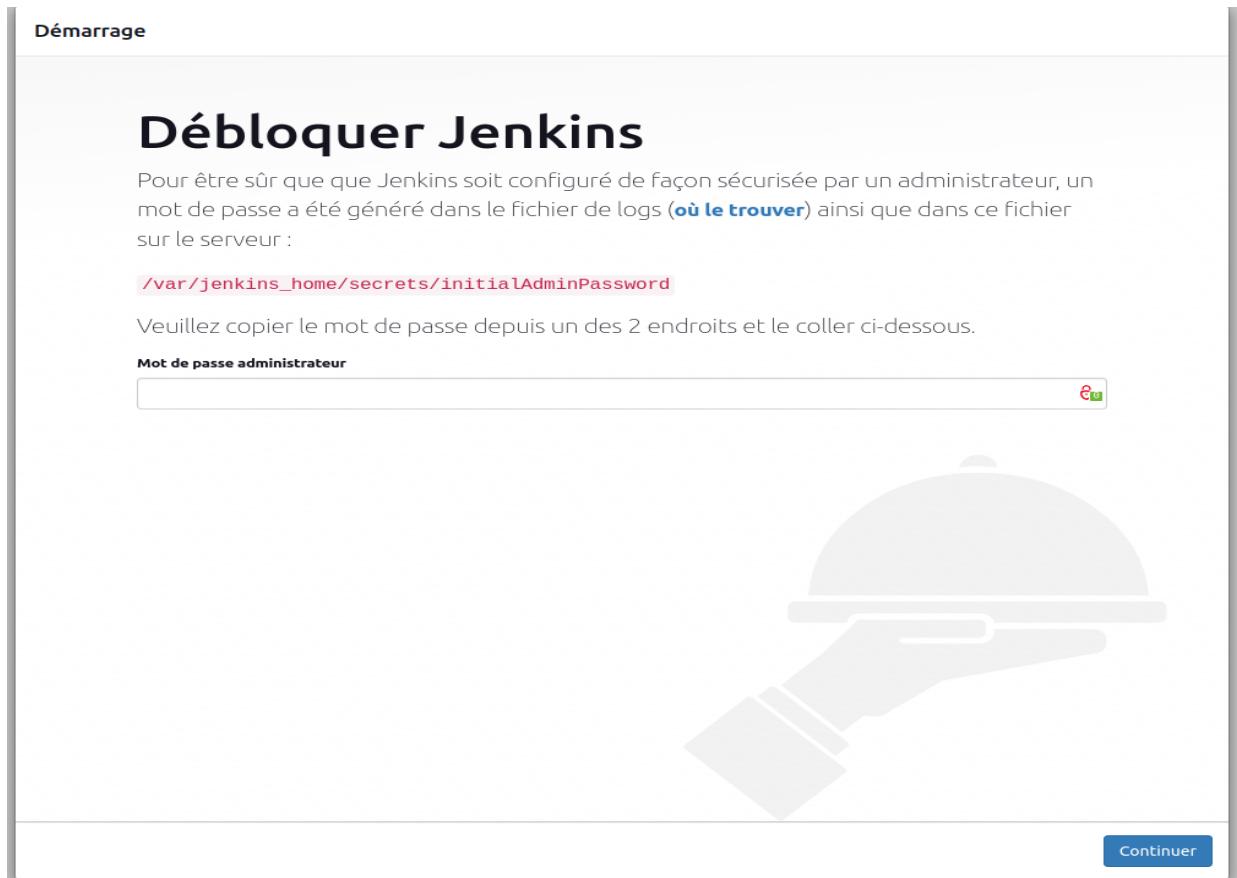


Fig : débloquer jenkins avec la clé de connection

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 jenkinks_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 OWASP Markup Formatter ** Structs ** Token Macro Build Timeout ** Credentials ** Trilead API ** SSH Credentials ** Pipeline: Step API ** Plain Credentials Credentials Binding ** SCM API ** Pipeline: API ** commons-lang3 v3.x Jenkins API Timestamper ** 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

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

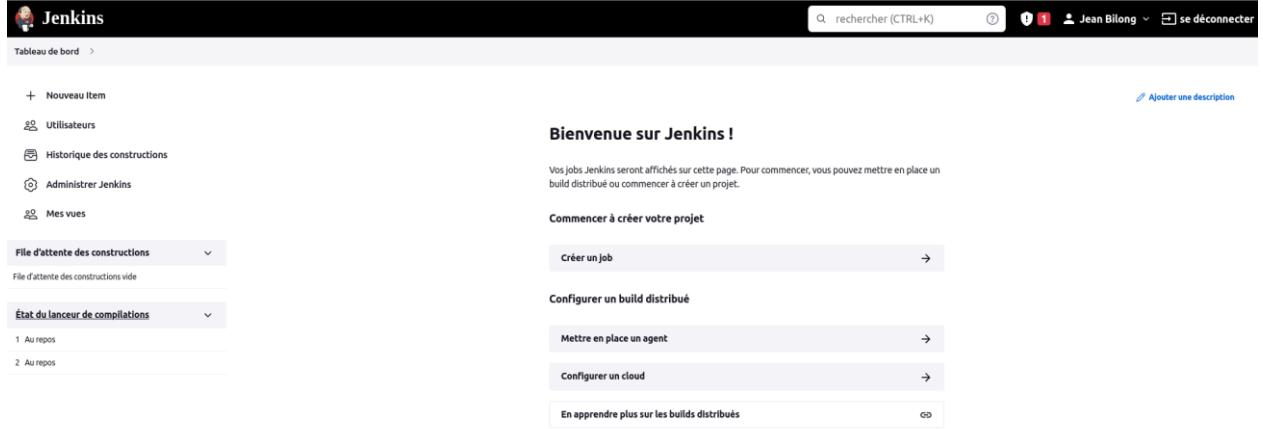


Fig : interface d'accueil de jenkins

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.

Saisissez un nom

» Champ obligatoire

 **Construire un projet free-style**
Ceci est la fonction principale de Jenkins qui sert à builder (construire) votre projet. Vous pouvez intégrer tous les outils de gestion de version avec tous les systèmes de build. Il est même possible d'utiliser Jenkins pour tout autre chose qu'un build logiciel.

 **Pipeline**
Organise des activités de longue durée qui peuvent s'étendre sur plusieurs agents de construction. Adapté pour la création des pipelines (anciennement connues comme workflows) et/ou pour organiser des activités complexes qui ne s'adaptent pas facilement à des tâches de type libre.

 **Construire un projet multi-configuration**
Adapté aux projets qui nécessitent un grand nombre de configurations différentes, comme des environnements de test multiples, des binaires spécifiques à une plateforme, etc.

 **Dossier**
Crée un conteneur qui stocke des objets imbriqués. Utile pour grouper ensemble des éléments. Contrairement à une vue qui n'est qu'un filtre, un dossier crée un espace de nommage distinct, de sorte que vous pouvez avoir plusieurs éléments du même nom tant qu'ils se trouvent dans des dossiers différents.

 **Organization Folder**
Creates a set of multibranch project subfolders by scanning for repositories.

 **Pipeline Multibranches**
Crée un ensemble de projets Pipeline en se basant sur les branches détectées dans le dépôt dun gestionnaire de code source.

OK

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

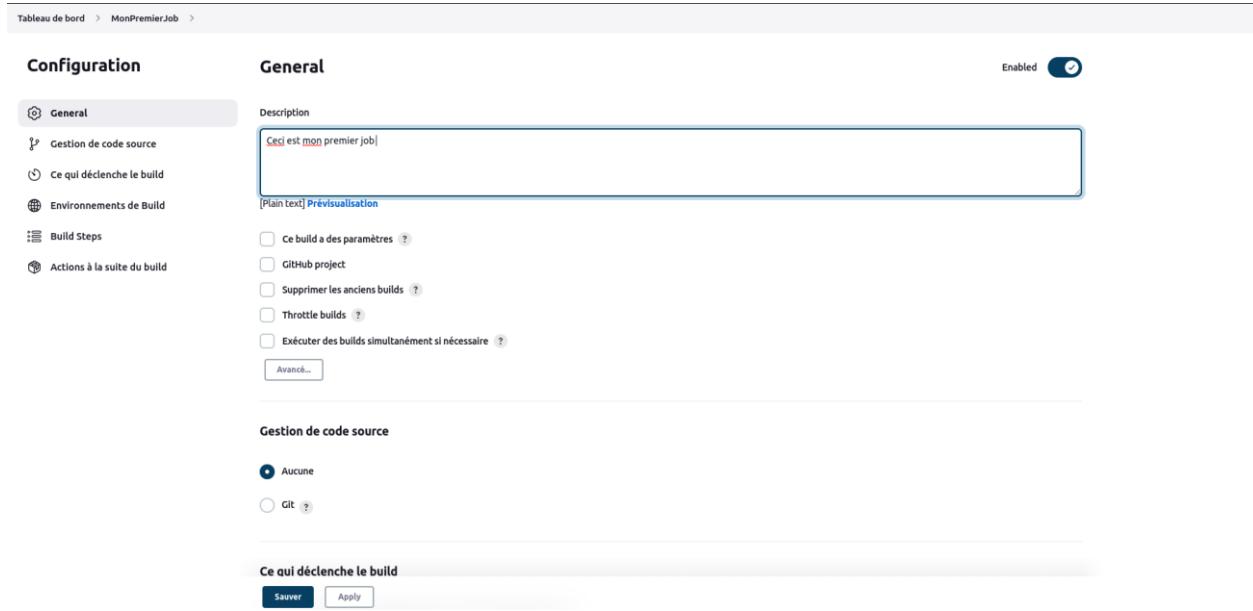


Fig: Définitions des paramètres du job

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 des jobs 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. At the top, there's a navigation bar with links for 'Tableau de bord', 'Nouveau item', 'Utilisateurs', 'Historique des constructions', 'Administrer Jenkins', and 'Mes vues'. A search bar says 'rechercher (CTRL+K)'. On the right, it shows 'Jean Bilong Mboumba' and a 'se déconnecter' button. Below the navigation, there's a section for 'Historique des constructions' with a table. The table has columns: 'S' (Status), 'M' (Last Result), 'Nom du projet' (Project Name), 'Dernier succès' (Last Success), 'Dernier échec' (Last Failure), and 'Dernière durée' (Last Duration). One row is shown for 'MonPremierJob' with status 'S.O.', last success 'S.O.', last failure 'ND', and duration 'ND'. There are also 'Ajouter une description' and a green 'D' icon. Below the table, there are dropdown menus for 'File d'attente des constructions' and 'État du lanceur de compilations', both currently set to 'Vide'. A legend at the bottom indicates that green icons represent success and red icons represent failure.

Fig: dashboard listant notre premier job

B- Exécution de mon premier job

Pour exécuter notre premier job manuellement il suffit de cliquer sur le bouton lancé 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 déclenché avec des erreurs.

The screenshot shows the Jenkins interface for the 'MonPremierJob' project. At the top, there are navigation links: 'Retour au tableau de bord', 'État' (selected), 'Modifications', 'Répertoire de travail', 'Lancer un build' (highlighted in a red box), 'Configurer', 'Supprimer Projet', and 'Renommer'. Below this is a section titled 'Liens permanents' with a list of recent builds. The main area is titled 'Historique des builds' with a 'tendance' dropdown set to 'tendance'. A search bar 'Filter builds...' is present. The build history table lists six builds (#6 to #1) from 22 nov. 2022 07:19 to 22 nov. 2022 07:13. Each build row has a green checkmark icon and a link to its details. At the bottom, there are links for 'Atom feed des builds' and 'Atom feed des échecs'.

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 'Sortie de la console' (Console Output) page for build #6. The left sidebar includes links: 'Retour au projet', 'État', 'Modifications', 'Sortie de la console' (selected), 'Voir en texte brut', 'Informations de la construction', 'Supprimer le build #6', and 'Build précédent'. The main content area is titled 'Sortie de la console' with a green checkmark icon. It displays the command-line output of the 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 interface for a job named 'MonPremierJob'. The left sidebar has links for 'Retour au projet', 'État', 'Modifications', 'Sortie de la console' (which is selected and highlighted in grey), 'Voir en texte brut', 'Informations de la construction', 'Supprimer le build "#5"', 'Build précédent', and 'Build suivant'. The main content area is titled 'Sortie de la console' with a red 'X' icon. It displays the following Jenkins log output:

```

Started by user Jean Bilong Mboumba
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/MonPremierJob
[MonPremierJob] $ /bin/sh -xe /tmp/jenkins4697873627868697277.sh
+ echffo Hello World
/tmpp/jenkins4697873627868697277.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.

5. 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.

✓ #10 4 déc. 2022 14:04
[Atom feed des builds](#) [Atom feed des échecs](#)



Derniers résultats des tests (aucune erreur)

Liens permanents

- 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.

Extended E-mail Notification

SMTP server

ssl0.ovh.net

SMTP Port

465

Credentials

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

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.fullDisplayName} : ${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>'

}

}

```

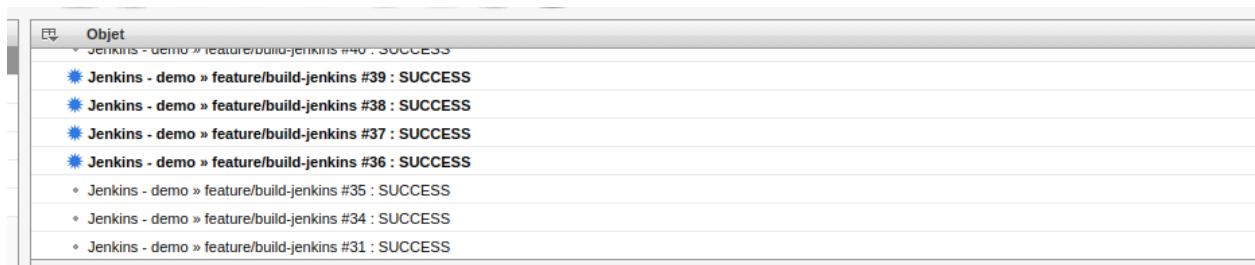


Fig: Notification par email

6. 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 dépendance 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

! 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**

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

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
Optionally 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

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
Optionally 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 ?

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

7. 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 génération des rapports de couverture avec
- But : test

C. Buts et phases

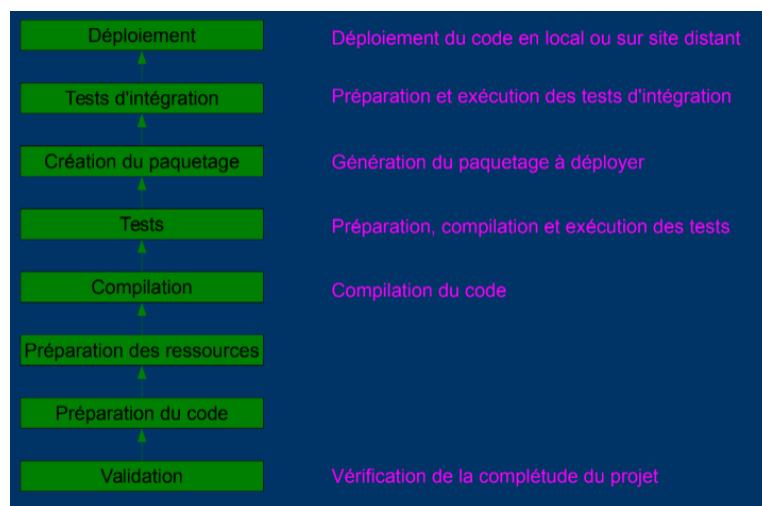
Exécution d'un but

- Exécution du but seulement

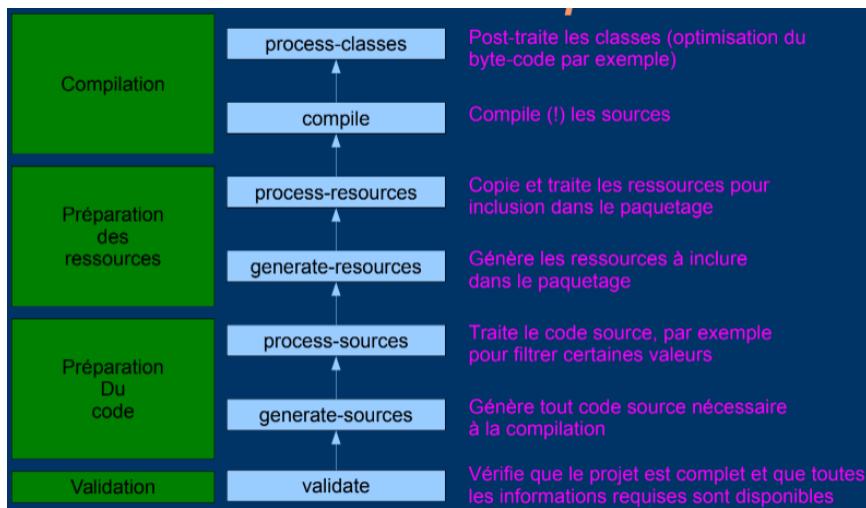
Exécution d'une phase

- Exécution de la phase précédente
- Exécution du but associé

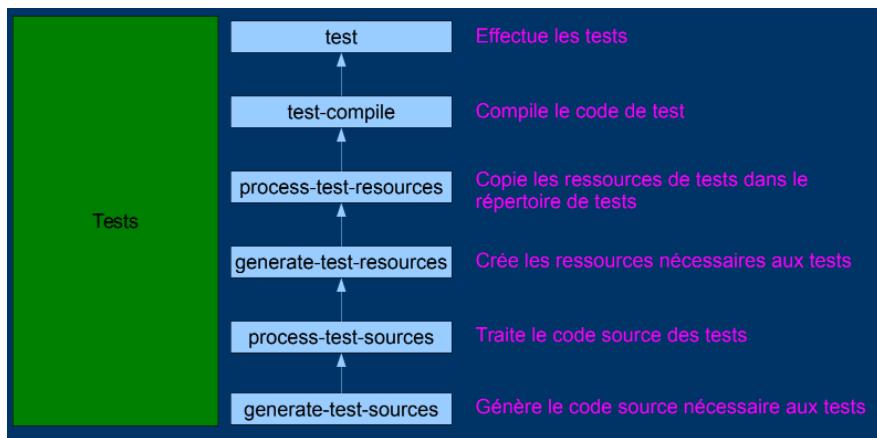
D. Etapes du cycle de vie Maven par défaut



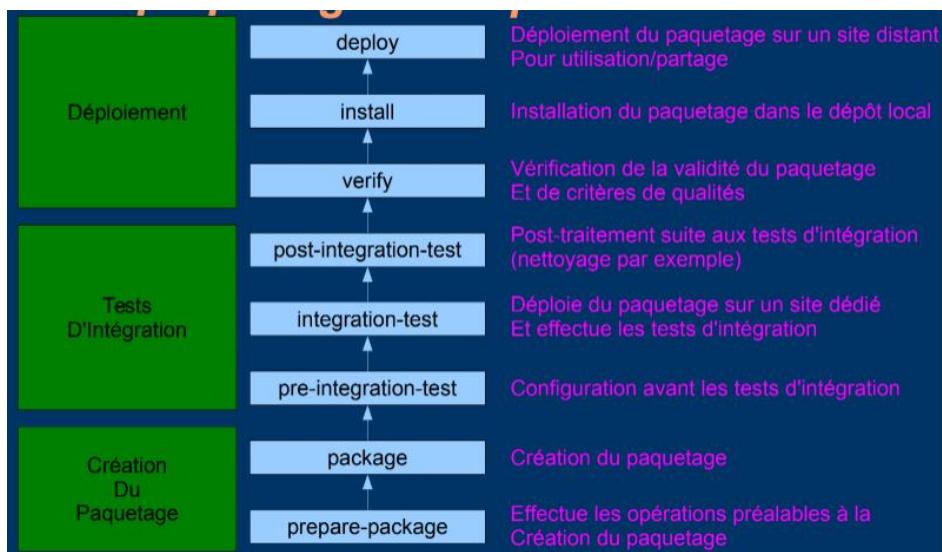
1. De la Validation à la compilation



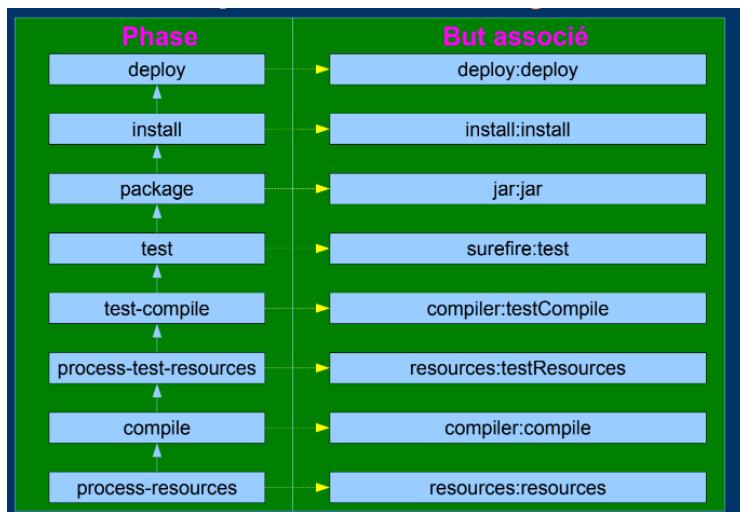
2. Tests



3. Du paquetage au déploiement



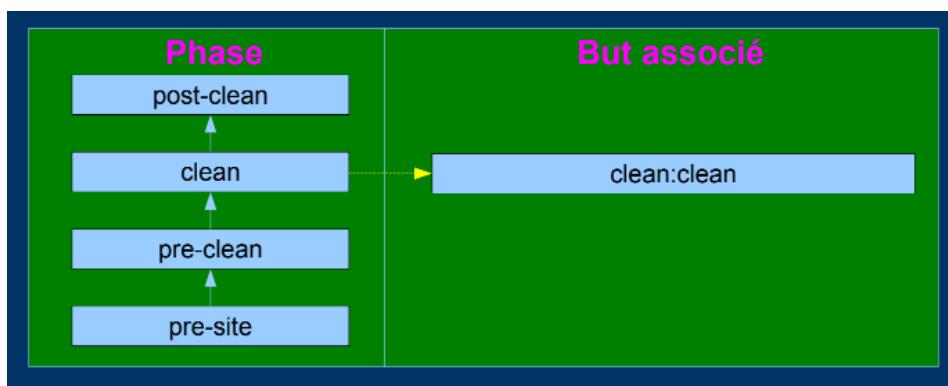
1. Cycle de vie pour un fichier jar:



2. Cycle de vie pour un site:

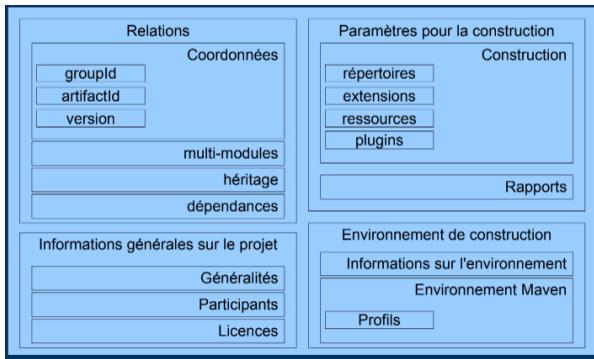


3. Cycle de vie pour le nettoyage



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/**.jar

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 offre 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.

8. Jenkins et le Build, les meilleures pratiques et méthodes recommandées.

Comme on 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 meilleure 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 dépendances 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 rapports 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 pipelines 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-ng/>) 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 : **Invoquer 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 the Jenkins configuration interface for a 'Static Analysis Tools' step. It is specifically set up for the 'FindBugs' tool. The configuration includes:

- Report File Pattern:** A text input field containing the placeholder 'Fileset 'includes'' syntax, which specifies the files to scan for issues. It notes that if left empty, the default file pattern '**/findbugsXml.xml' will be used.
- Skip symbolic links when searching for files:** An unchecked checkbox.
- Report Encoding:** A dropdown menu currently set to 'UTF-8'.
- Use the bug rank when evaluating the severity of the warnings:** A checked checkbox.
- Custom ID:** A text input field for optional custom ID, currently empty.
- Custom Name:** A text input field for optional custom display name, currently empty.

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](#)
- git** Revision: eaa279f8359a55dc7c6773f8d80b36cf0ef26624
Repository: <https://github.com/bilongjea/rps-boot.git>
 - refs/remotes/origin/master
- FindBugs:** 3 warnings [i](#)
- CheckStyle:** 221 warnings [i](#)
- PMD:** 10 warnings [i](#)

Figure: Résultat d'exécution du job avec les différents rapports 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.

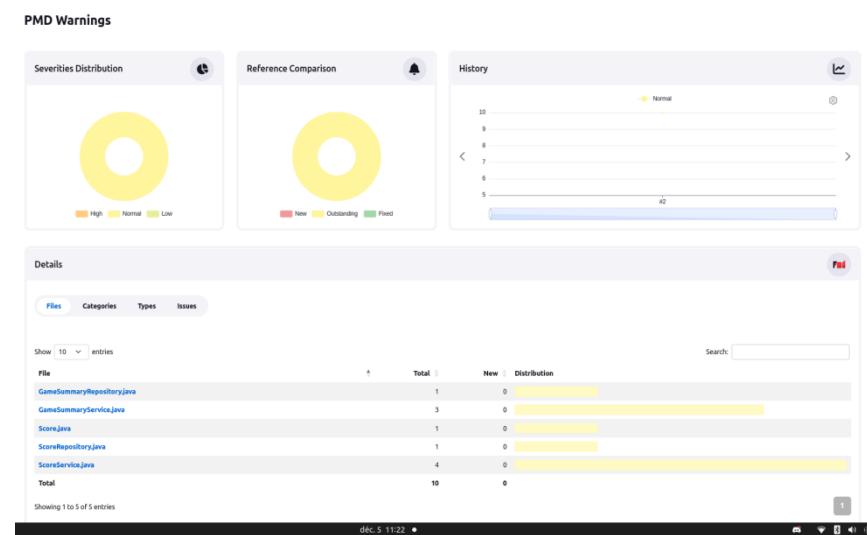


Figure: Exemple de rapport PMD

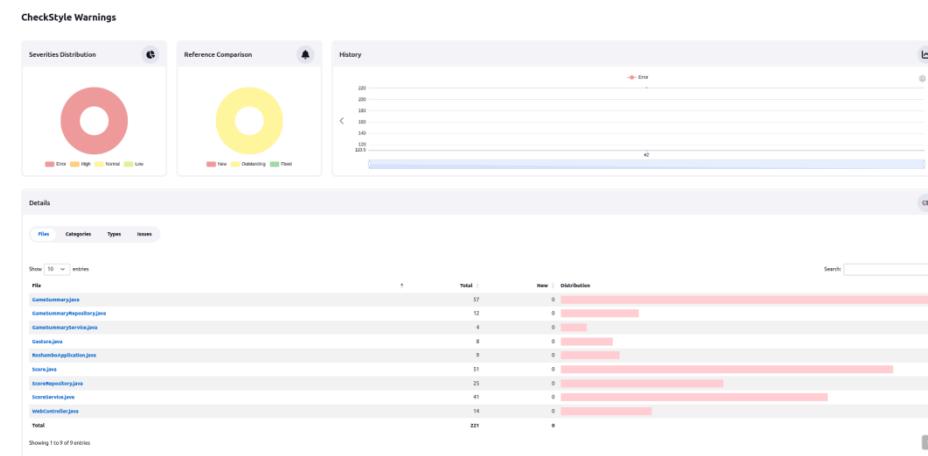


Figure: Exemple de rapport CheckStyle

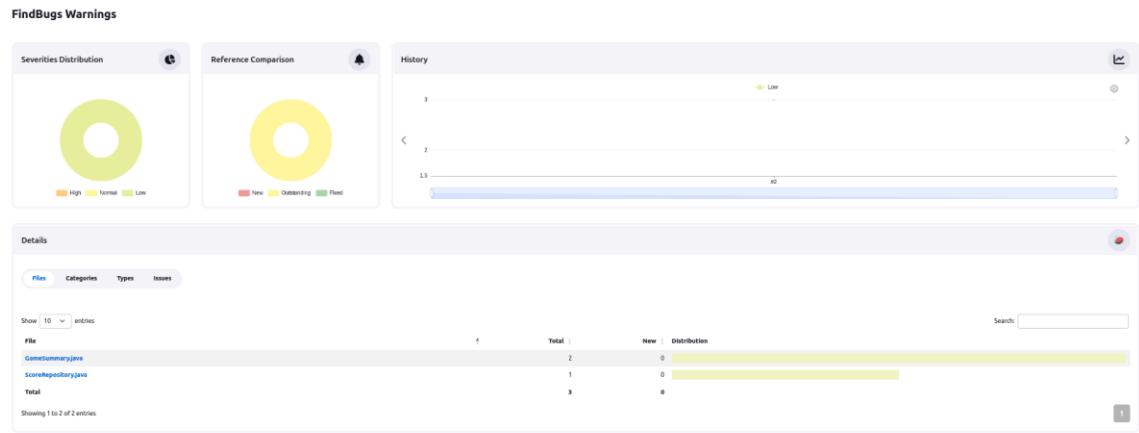


Figure: Exemple de rapport FindBugs

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.

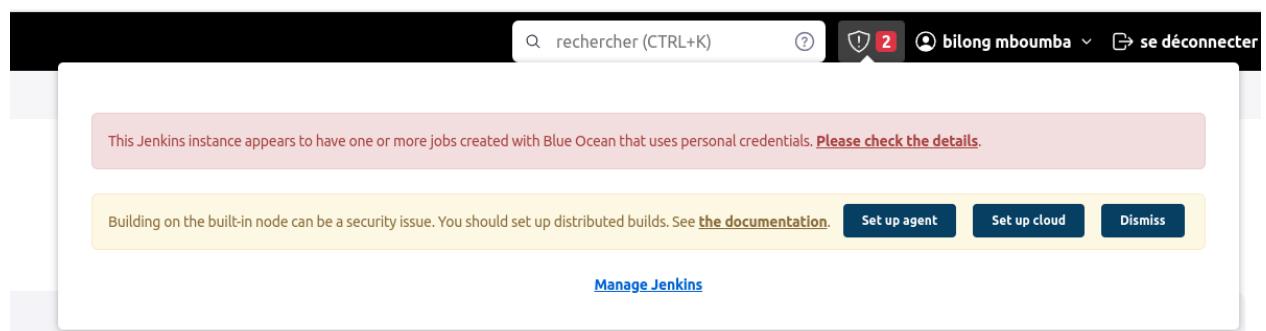


Fig: Icônes d'alerte de jenkins

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: Nouveau Item, Utilisateurs, Historique des constructions, Relations entre les builds, Vérifier les empreintes numériques, Administrer Jenkins (which is selected), Mes vues, Open Blue Ocean, File d'attente des constructions, and État du lanceur de compilations. The main area is titled "Administrer Jenkins". It has several sections: "System Configuration" with links to Configurer le système, Configuration globale des outils, Gestion des plugins, and Gérer les nœuds; "Security" with links to Configurer la sécurité globale, Manage Credentials, Configure Credential Providers, and Gérer les utilisateurs; and a "Plugins" section with a list of installed plugins like Blue Ocean, CloudBees Jenkins Pipeline, etc. There are also status messages at the top: "This Jenkins instance appears to have one or more jobs created with Blue Ocean that uses personal credentials. Please check the details" and "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 ?

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 globale pour définir son Realm



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 'Créer un utilisateur' (Create User) form. The 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 (with a red error icon)

A blue 'Créer un utilisateur' (Create User) button is at the bottom.

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	Tag
	Read	Create	Delete	Configure	Update	Run	View	Read	Tag
Administrator	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Builder	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>						
Creator	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>						

Role to add

Add

Item roles

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

Role to add

Pattern

.*

Add

Node roles

Role	Pattern	Credentials	Manage ownership	Agent
		Jobs	Notes	Discover
agent-builder	<input checked="" type="checkbox"/> "build-agent"	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

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

User/group to add ?

Add

Node roles

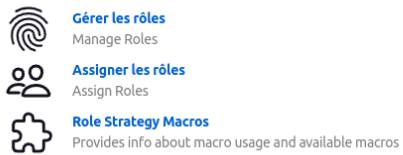
User/group	agent-builder
Anonymous	<input type="checkbox"/>
item-builder-user	<input checked="" type="checkbox"/>

User/group to add ?

Add

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

Gérer et assigner les rôles



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.

The screenshot shows the Jenkins 'Configure System' page with the 'Audit Trail' section selected. The 'Loggers' section contains the following configuration:

- Log file**:
 - Log Location**: /var/log/jenkinsusages.log
 - Log File Size MB**: 1
 - Log File Count**: 100
 - Number of rotating log files to use (see [FileHandler](#))**: (from [Audit Trail](#))
- Log Separator**: (empty field)

Below the loggers, there is a section for **URL Patterns to Log** with the value `*(?:configSubmit|doDelete|postBuildResult|enable|disable|cancelQueue|stop|toggleLogKeep|doWipeOutWorkspace|createItem|createView|toggleOffline|cancelQuietDown|quietDown|restart|exit|safeExit)/*`. At the bottom, there are two checked checkboxes:

- Log how each build is triggered
- Log credentials usage

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.



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.

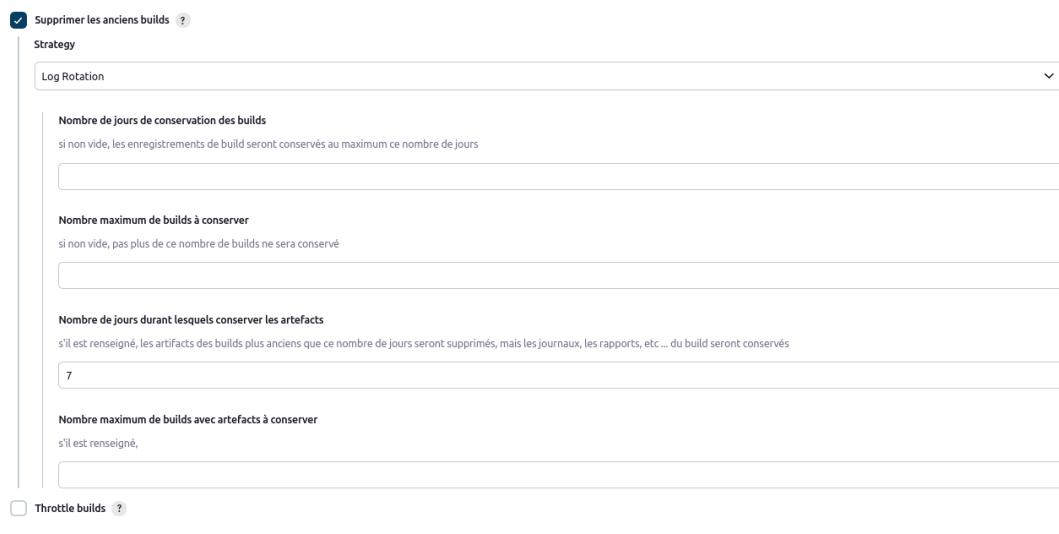


Fig : Supprimer les anciens builds - options avancées

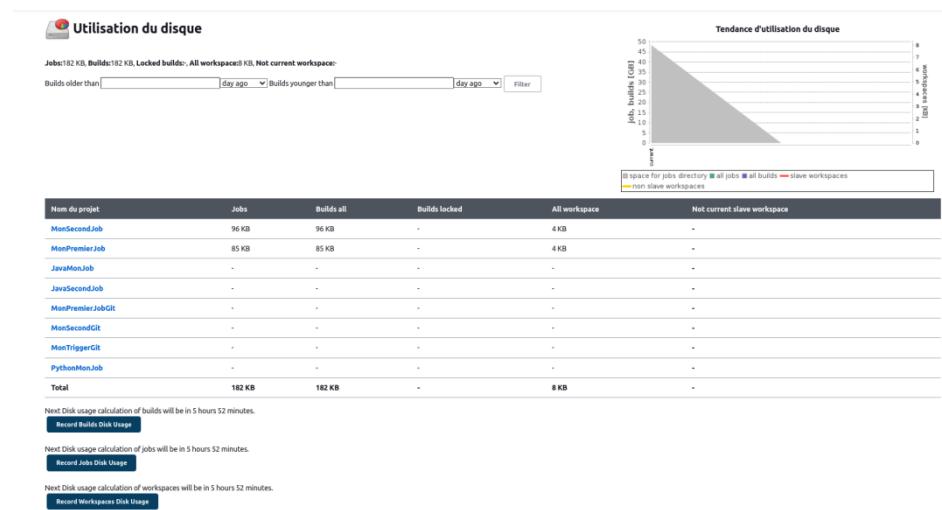
Dans Figure ci-dessus, par exemple, nous avons configuré Jenkins pour qu'il garde les artefacts 7 jours au maximum. Cette option est vraiment pratique si vous avez besoin de limiter l'utilisation du disque tout en désirant fournir l'ensemble des métriques pour les équipes de développement.

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.

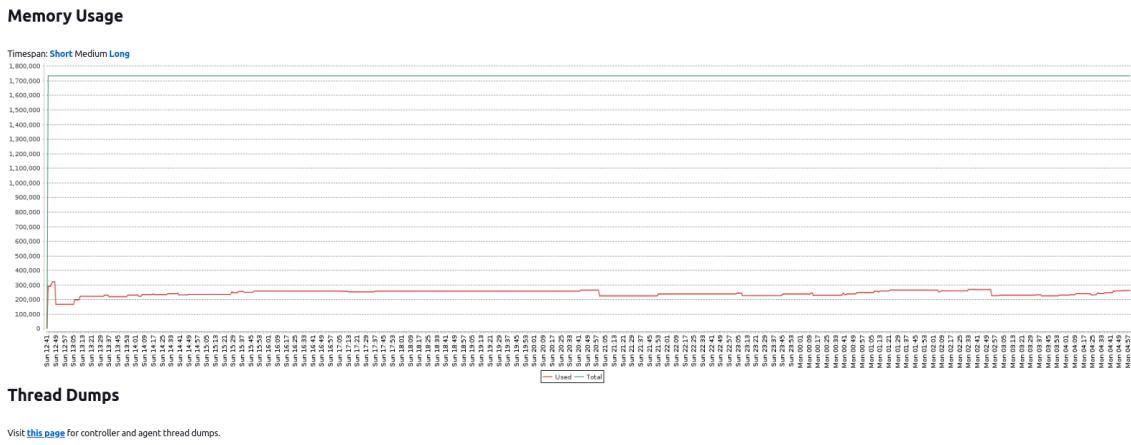


6. Monitoring de la charge CPU.

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 exécuteur 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 aisés. 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.correlator.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

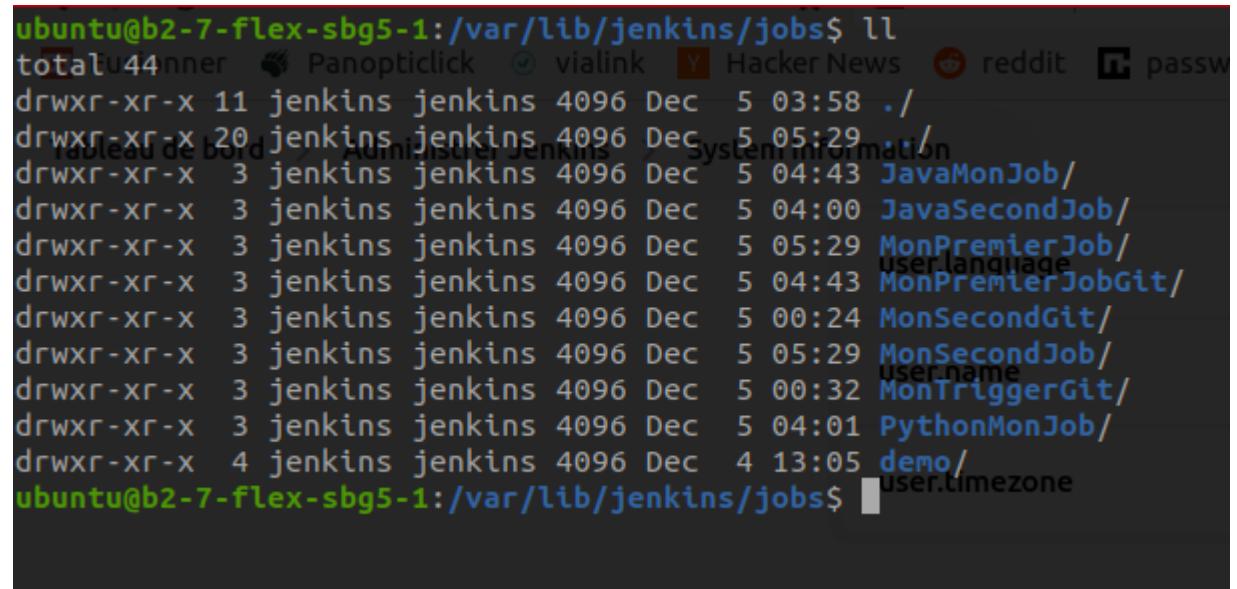
Il contient la configuration de toutes vos tâches de build, les configurations de vos nœuds esclaves et l'historique des builds. La sauvegarde peut se faire pendant que Jenkins tourne. Il n'y a pas besoin de couper votre serveur pendant la sauvegarde.

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 /  
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 tâches 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

Annexes

Annexe 1: Installation des outils

1. Sous Linux/Ubuntu/WSL

Mise à jour et installation des package de base

```
> sudo apt update && sudo apt update  
> sudo apt install git git-flow maven nodejs npm yarn curl zip unzip -y
```

Création d'un compte github et Génération et Ajout de la clé ssh:

```
> ssh-keygen -o -t rsa
```

Install basique de java

```
> sudo apt install openjdk-17-jdk -y # Java 17  
> sudo apt install openjdk-21-jdk -y # Java 21  
  
Lister config la version de java par défaut  
  
> update-alternatives --list java  
  
> sudo update-alternatives --config java  
  
> sudo update-alternatives --config java
```

Alternative plus modern d'installation de java via sdkman :

```
> curl -s "https://get.sdkman.io" | bash  
  
> source "$HOME/.sdkman/bin/sdkman-init.sh"  
  
> sdk list java # liste les versions de java  
  
> sdk install java 17.0.10-tem # Java 17  
  
> sdk install java 21.0.2-tem # Java 21
```

```
> sdk use java 17.0.10-tem          # change la version current  
> sdk default java 21.0.2-tem      # change la version par défaut
```

Installation et configuration de docker

```
# supprime les package obsolète  
  
> sudo apt remove $(dpkg --get-selections docker.io docker-compose docker-compose-v2 docker-doc podman-docker containerd runc | cut -f1)
```

```
# Ajoute de la clé officielle GPG:  
  
> sudo apt update  
  
> sudo apt install ca-certificates curl  
  
> sudo install -m 0755 -d /etc/apt/keyrings  
  
> sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o  
/etc/apt/keyrings/docker.asc >sudo chmod a+r /etc/apt/keyrings/docker.asc
```

```
# ajout du repo aux dépots apt:  
  
> sudo tee /etc/apt/sources.list.d/docker.sources <<EOF  
  
Types: deb  
  
URIs: https://download.docker.com/linux/ubuntu  
  
Suites: $(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}")  
Components: stable  
  
Signed-By: /etc/apt/keyrings/docker.asc  
  
EOF  
  
> sudo apt update
```

```
# installation  
  
> sudo apt install docker-ce docker-ce-cli containerd.io docker-buildx-plugin  
docker-compose-plugin
```

```
# verification de l'installation

> sudo systemctl status docker

> sudo systemctl start docker

> sudo docker run hello-world
```

```
# Ajouter son user au group docker

> sudo groupadd docker

> sudo usermod -aG docker $USER

> newgrp docker

> docker run hello-world
```

```
#Activer docker au demurrage tu system

> sudo tee /etc/wsl.conf <<'EOF'

[boot]

systemd=true

EOF
```

```
# Installation de linux sur windows via WSL

> wsl -v # vérifier si wsl est activer sur windows 10/11 sinon installer via
Installer WSL | Microsoft Learn

> wsl --list --online      # liste les distribution disponible

> wsl --install -d <NomDeLaDistribution>      # installation de la distrib

> wsl -d <NomDeLaDistribution>                  # lancer la distrib

> wsl --unregister <NomDeLaDistribution>       # désinstaller la distrib
```

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

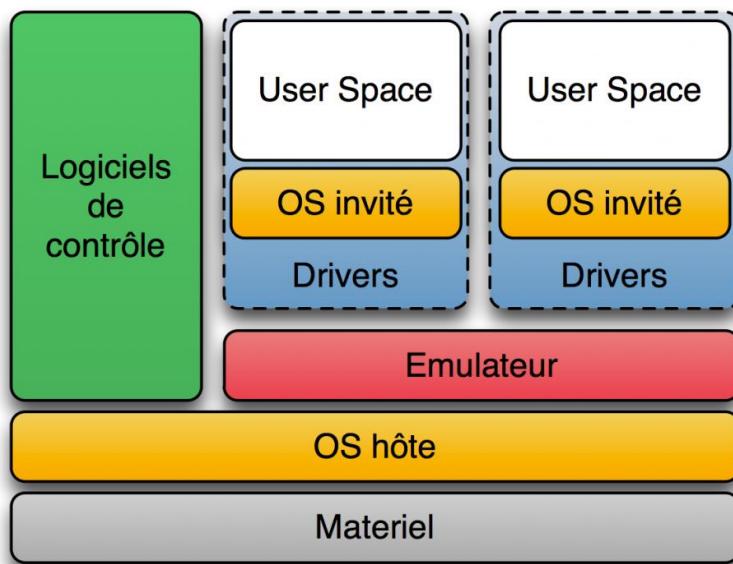
- <https://blog.lecacheur.com/2015/10/14/docker-au-revoir-boot2docker-bonjour-docker-toolbox/>
- Scoop <https://github.com/lukesampson/scoop/wiki/Docker>
- Docker Desktop
 - <https://docs.docker.com/docker-for-windows/install-windows-home>
 - <https://docs.microsoft.com/en-us/windows/wsl/install-win10>
 - <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.



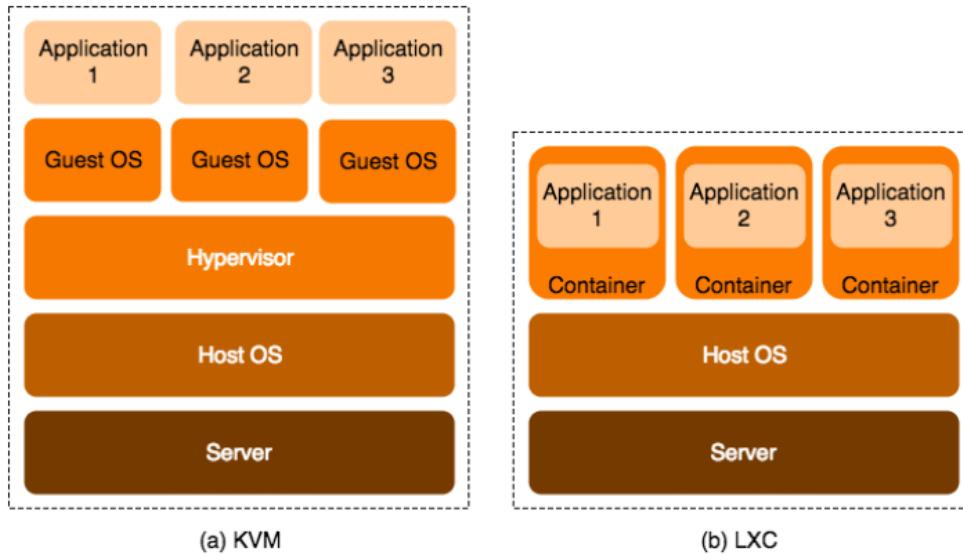
Il ne s'agit pas de **virtualisation**, mais de **conteneurisation**, une forme plus légère qui s'appuie sur certaines parties de la machine hôte pour son fonctionnement.

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**).

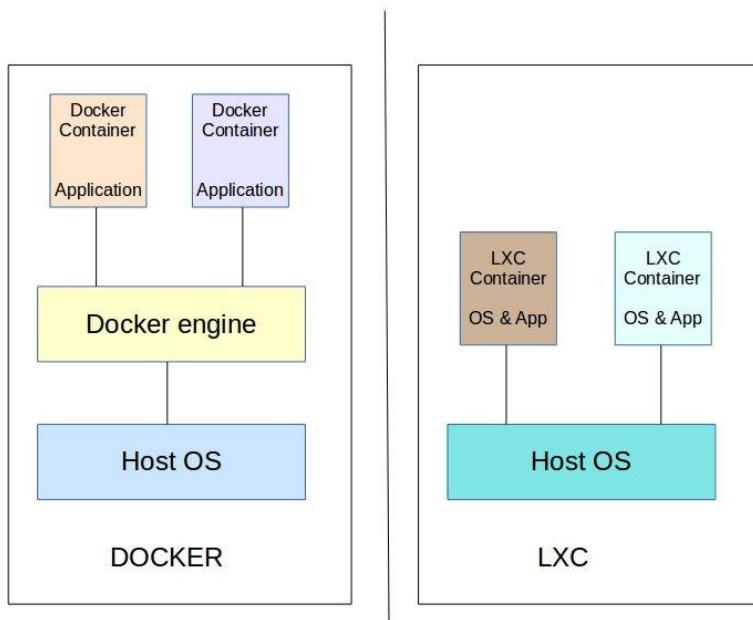
Il est utilisé pour faire fonctionner **des environnements Linux isolés les uns des autres dans des conteneurs, partageant le même noyau** et une plus ou moins grande partie du système hôte.

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

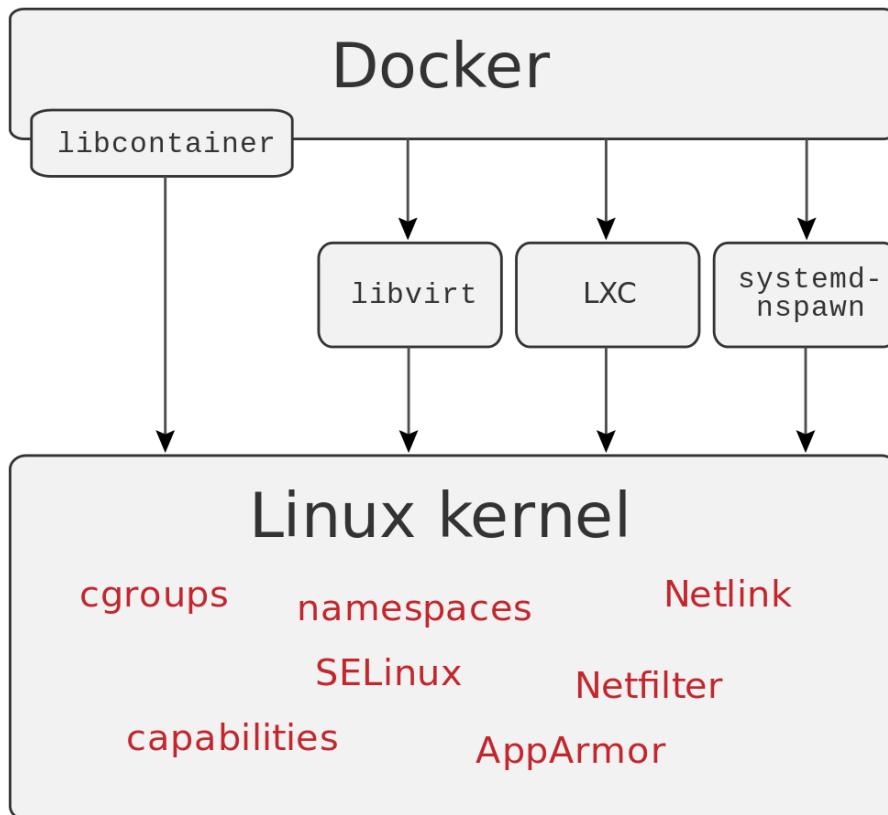


Pour cette raison, on parle de « **conteneur** » et non de « **machine virtuelle** ». au niveau du **système d'exploitation**: contrairement à la **virtualisation** qui émule par logiciel différentes **machines** sur une machine physique, la **conteneurisation** émule **differents OS sur un seul OS**.

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 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
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)

```
Intallation du Jhispter Generator
```

```
sudo npm install -g generator-jhipster
```

```
> 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

JHIPSTER

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 utiles

- > 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
```

Annexe 5 Mise à jour Jenkins 2025

1. Évolutions Jenkins (2023–2025)

- **Version LTS** : Jenkins est désormais en 2.479+ (sept. 2025).
- **Java** : le contrôleur et les agents fonctionnent officiellement avec **Java 21**. Java 11 est déprécié.
- **Pipeline as Code** : les *freestyle jobs* sont de moins en moins utilisés.
 - Recommandation : utiliser des **Jenkinsfile** (déclaratif ou scripté).
 - Support des **Multibranch Pipelines** (un job par branche automatiquement).
- **Shared Libraries** : permettent de factoriser des étapes communes entre projets.
- **Agents Docker** : trois patterns sont aujourd’hui utilisés :
 - **Docker-outside-of-Docker (DoD)** : montage du socket Docker de l’hôte (simple mais attention à la sécurité).
 - **Docker-in-Docker (DiND)** : un démon Docker dans un conteneur (souvent en mode privilégié).
 - **Remote builder (BuildKit)** : délégation des builds d’images à un serveur distant spécialisé

2. Qualité & Analyses modernes

- **FindBugs** a été remplacé par **SpotBugs**.
- Le plugin **Violations** a disparu → remplacé par **Warnings Next Generation (Warnings NG)**.
 - Il centralise les rapports de Checkstyle, PMD, SpotBugs, ESLint, Pylint, ShellCheck, etc.
- **Couverture de code** : JaCoCo reste la référence, avec export en **XML**.
- Exemple pipeline :

```
recordIssues tools: [  
  
    checkStyle(pattern: '**/target/checkstyle-result.xml'),  
  
    pmdParser(pattern: '**/target/pmd.xml'),  
  
    spotBugs(pattern: '**/target/spotbugsXml.xml')  
]
```

```
recordCoverage tools: [jacoco(pattern: '**/target/site/jacoco/jacoco.xml')]
```

3. Intégration Sonar

- Deux options principales :
 - **SonarQube Community Edition** : on-premise, gratuit, nécessite un serveur.
 - **SonarCloud** : hébergé, gratuit pour projets publics.
- Exemple pipeline :

```
withSonarQubeEnv('sonarqube') {  
    sh './mvnw sonar:sonar'  
}
```

Possibilité d'ajouter une étape **Quality Gate** : le pipeline échoue si la gate est rouge.

4. Artefacts & Registries

- Alternatives modernes aux gestionnaires d'artefacts :
 - **Nexus Repository OSS** (gratuit, on-premise).
 - **GitHub Packages** (hébergé, pratique si vos sources sont déjà sur GitHub).
 - **JFrog Artifactory** reste très répandu en entreprise.
- Maven (mvn deploy) et npm (npm publish) savent publier directement sur ces registries.

5. Déploiement Cloud

- **Heroku** n'est plus recommandé (fin du gratuit depuis 2022).
- Options simples et现实istes en 2025 :
 - **VM Cloud (GCP, AWS, OVH)** : déployer avec scp et systemd (backend) + Nginx (frontend).
 - **Cloud Run (GCP) ou AWS ECS/Fargate** pour déploiement Docker managé.
- Bonus : **Jib** (plugin Maven/Gradle) permet de construire une image Docker **sans Docker installé** :

```
mvn jib:build -Dimage=gcr.io/my-project/my-service:1.0.0
```

6. Administration moderne

- **Jenkins Configuration as Code (JCasC)** : configuration de Jenkins via un fichier **YAML** (sécurité, agents, credentials, outils, plugins).
- **Gestion des plugins** : pré-installer via `jenkins-plugin-cli --plugin-file plugins.txt`.
- **Sécurité** :
 - 0 exécuteur sur le contrôleur,
 - exécution des builds uniquement sur des agents,
 - credentials gérés par le Credential Store.