

# Jenkins Administration



# TABLE DES MATIÈRES

Introduction

Intégration Continue

Installation de Jenkins

Authentification et autorisations

Installation d'agents dans le cluster Jenkins

Stratégies de sauvegarde

Plan de reprise d'activité (PRA) pour Jenkins

Intégration avancée avec GitLab, Bitbucket et Jira

Mutualisation des pipelines

# Introduction

Tour de table

Généralités sur les méthodes agiles

Généralités sur DevOps

Les outils d'intégration continue





# Introduction

[Tour de table]

Présentations et attentes

Connaissances Agile, CI/CD, Docker, Git, ...etc.



# Introduction

[Généralités sur les méthodes agile]

Une approche et non une méthode

Résultats d'une rencontre entre 17 développeurs désireux de partager leur expérience en février 2001 aux USA.



# Introduction

[Généralités sur les méthodes agile]

Toutes les méthodologies Agile reposent sur quatre valeurs fondamentales (les fameux quatre piliers):

- Les individus et les interactions plus que les processus et les outils
- Logiciels (produits, services, etc.) fonctionnels plus qu'une documentation exhaustive
- La collaboration avec les clients plus que la négociation contractuelle
- L'adaptation au changement plus que le suivi d'un plan



# Introduction

[Généralités sur les méthodes agile]

Les douze principes du Manifeste Agile :

1. Livrer de la valeur au client
2. Intégrer les demandes de changement
3. Livrer fréquemment une version opérationnelle
4. Assurer une coopération entre le client et l'équipe
5. Réaliser les projets avec des personnes motivées
6. Privilégier le dialogue en face à face
7. Mesurer l'avancement sur la base d'un produit opérationnel
8. Faire avancer le projet à un rythme soutenable et constant
9. Contrôler l'excellence technique et à la conception
10. Minimiser la quantité de travail inutile
11. Construire le projet avec des équipes auto-organisées
12. Améliorer constamment l'efficacité de l'équipe



# Intégration continue

[Définition]

## Définition de l'intégration continue

L'intégration continue est une méthode de développement logiciel qui consiste à intégrer fréquemment (souvent plusieurs fois par jour) les modifications de code dans un dépôt central partagé.

Chaque intégration déclenche automatiquement une série de tests pour vérifier que le nouveau code n'introduit pas de régressions ou d'erreurs.



# Intégration continue

[principes]



## Principes fondamentaux

- **Intégration fréquente** : Les développeurs fusionnent leur code dans la branche d'intégration plusieurs fois par jour.
  - **Automatisation des tests** : Chaque intégration déclenche une suite de tests automatisés pour détecter rapidement les erreurs.
  - **Détection précoce des bugs** : Les erreurs sont identifiées immédiatement après leur introduction, facilitant leur correction.
  - **Build automatisé** : Le code est compilé automatiquement à chaque modification pour garantir sa validité.
  - **Feedback rapide** : Les développeurs reçoivent des notifications immédiates en cas d'échec des tests ou du build.
- **Réduction de la dette technique** : En corrigéant les erreurs au fur et à mesure, on évite l'accumulation de problèmes complexes à la fin du projet.
- **Amélioration de la qualité du code** : Grâce aux tests et à la validation continue, le code reste propre et fonctionnel



# Intégration continue

[principes]

## Objectifs

- Accélérer le cycle de développement
- Améliorer la collaboration entre les membres de l'équipe
- Livrer des logiciels plus fiables et de meilleure qualité
- Faciliter la mise en œuvre de la livraison continue (CD)



# Intégration continue

[Notions de Génis Logiciel]

*Besoins, Conception, développement, Tests, Déploiement et Maintenance.*

# Intégration continue

[Notions de Génis Logiciel]

## Cycle de vie & Méthodologies

- Phases :  
Besoins → Conception → Développement → Tests → Déploiement → Maintenance
- Approches :
  - Cascade (linéaire)
  - Agile (Scrum, Kanban, XP) → itératif, collaboratif, adaptable

# Intégration continue

[Notions de Génis Logiciel]

## Bonnes pratiques & Qualité

- Outils & pratiques :
  - Git (gestion de versions)
  - Tests automatisés + TDD
  - Revues de code
  - Architecture modulaire (faible couplage)
- Qualité logicielle (ISO 25010) :  
Fonctionnalité • Fiabilité • Utilisabilité • Efficacité • Maintenabilité • Portabilité

« *Le génie logiciel, c'est l'art de transformer des besoins flous en solutions fiables.* »



# Intégration continue

[Chaîne de fabrication logicielle]

**Définition :** Processus structuré et automatisé qui transforme une idée en logiciel opérationnel, fiable et maintenable.

**Étapes clés :**

1. **Expression des besoins** – Recueil des exigences métier
2. **Conception** – Architecture, modélisation, choix technologiques
3. **Développement** – Écriture du code source
4. **Intégration continue (CI)** – Fusion du code + tests automatisés
5. **Tests** – Vérification fonctionnelle, technique et de performance
6. **Livraison continue (CD)** – Déploiement automatisé vers les environnements
7. **Supervision & maintenance** – Suivi en production, corrections et évolutions

**Outils typiques :** Git (GitHub/GitLab), Jenkins, GitLab CI, Docker, Kubernetes, Prometheus, Selenium



# Installation de Jenkins





# Installation de Jenkins

- Voir document Installation\_de\_Jenkins.pdf en annexes



# TP1: Créer et compiler un projet Freestyle

Un Un projet Freestyle (ou Freestyle project) est un type de job Jenkins qui permet de configurer manuellement une séquence d'étapes d'intégration continue. Il est idéal pour les scénarios simples ou personnalisés, comme :

- Compiler du code source
- Exécuter des tests unitaires
- Déployer une application
- Exécuter des scripts shell ou batch
- Envoyer des notifications (email, Slack, etc.)

# TP1: Créer et compiler un projet Freestyle

## ✓ Avantages du Freestyle Project

- Simple à configurer via l'interface web
- Flexible : supporte de nombreux plugins
- Bon pour les débutants en CI/CD
- Supporte les paramètres, les déclencheurs (triggers), les artefacts, etc.

## ⚠ Limites

- Moins adapté aux pipelines complexes ou versionnés
- Configuration non versionable (contrairement aux Pipeline as Code)
- Moins reproduit entre environnements



Pour les projets modernes, utiliser Jenkins Pipeline (défini dans un [Jenkinsfile](#)), car il permet de versionner la CI/CD avec le code source.

# TP1: Crée et compiler un projet Freestyle

The screenshot shows the Jenkins dashboard. At the top left is the Jenkins logo and the word "Jenkins". Below it is a navigation bar with "All" and "New Item". The main area has three buttons: "+ New Item" (highlighted with a red box and labeled 1), "Build History", and "Build Queue" which says "No builds in the queue". Below these are sections for "Build Executor Status" (0/2) and "Start building your software project" with a "Create a job" button.

## Welcome to Jenkins!

This page is where your Jenkins jobs will be displayed or started. You can view build logs or start building a software project.

### Start building your software project

Create a job

### Set up a distributed build

Set up an agent

3 OK

Configure a cloud

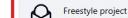
Learn more about distributed builds

## New Item

Enter an item name

Job Freestyle

Select an item type



Freestyle project

Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.



Pipeline

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.



Multi-configuration project

Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.



Folder

Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.



Multibranch Pipeline

Creates a set of Pipeline projects according to detected branches in one SCM repository.



Organization Folder

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



# TP1: Créeer et compiler un projet Freestyle

Jenkins / Job FreeStyle / Configuration

Configurer

Général

Description

Job free style.

1

Texte brut Prévisualisation

Ce build a des paramètres ?

GitHub project

Permission to Copy Artifact

Supprimer les anciens builds ?

Throttle builds ?

Exécuter des builds simultanément si nécessaire ?

Avancé ▾

Gestion de code source

Connect and manage your code repository to automatically pull the latest code for your builds.

Aucune

Git ?

Triggers

Enal

## Tips

- Faire passer la souris sur les points d'interrogation
- Prendre le temps de lire les informations affichées



# TP1: Créer et compiler un projet Freestyle

## Étapes du build

Automate your build process with ordered tasks like code compilation, testing, and

+ Ajouter une étape au build 1

Filter

- Appeler Ant
- Exécuter un script shell 2**
- Exécuter une ligne de commande batch Windows
- Invoke Gradle script
- Invoquer les cibles Maven de haut niveau
- Run with timeout
- Set build status to "pending" on GitHub commit

≡ Exécuter un script shell ?

Commande

Voir la liste des variables d'environnement disponibles

```
echo bonjour  
echo "Execution du job $JOB_NAME"  
echo "JENKINS_HOME = $JENKINS_HOME"
```

3

Avancé ▾

+ Ajouter une étape au build

Actions à la suite du build

Save 4 Appliquer



# TP1: Créer et compiler un projet Freestyle

- </> Modifications
- Répertoire de travail
- Lancer un build **1**
- Configurer
- Supprimer Projet
- Renommer
- Identifiants

Job free style.

Jenkins / Job FreeStyle / #289 / Sortie de la console

Liens permanents

2

Sortie de la console

Informations de la construction

Supprimer le build "#289"

Timings

Build précédent

Timestamps View as plain text ▾

System clock time

Use browser timezone

Elapsed time

None

Sortie de la console

22:40:27 Démarré par l'utilisateur Jean DUPOND

22:40:27 Exécution en tant que SYSTEM

22:40:27 En cours de construction dans le répertoire de travail C:\Jenkins\data\.jenkins\workspace\Job FreeStyle

22:40:27 [Job FreeStyle] \$ cmd /c call C:\Users\admin\AppData\Local\Temp\jenkins10298895722871927196.

22:40:27

22:40:27 C:\Jenkins\data\.jenkins\workspace\Job FreeStyle>echo Bonjour,

22:40:27 Bonjour,

22:40:27

22:40:27 C:\Jenkins\data\.jenkins\workspace\Job FreeStyle>echo Exécution du job Job FreeStyle

22:40:27 Exécution du job Job FreeStyle

22:40:27

22:40:27 C:\Jenkins\data\.jenkins\workspace\Job FreeStyle>echo JENKINS\_HOME = C:\Jenkins\data\.jenkins\workspace\Job FreeStyle

22:40:27 JENKINS\_HOME = C:\Jenkins\data\.jenkins\workspace\Job FreeStyle

22:40:27

22:40:27 C:\Jenkins\data\.jenkins\workspace\Job FreeStyle>exit 0

22:40:27 Finished: SUCCESS



# TP1: Créer et compiler un projet Freestyle

[Analyse du résultat du build]

## Dernier build stable

- Définition : Le dernier build qui a **réussi et qui n'a pas de tests instables** (pas de tests échoués ou ignorés).
- Exemple : #9 est stable car il a réussi **et** tous les tests sont passés.

## Dernier build en échec

- Définition : Le dernier build qui a **échoué** (erreur de compilation, tests échoués, problème de déploiement, etc.).
- Exemple : #8 est le dernier à avoir échoué.

## Dernier build complété

- Définition : Le dernier build qui s'est **terminé**, quelle que soit l'issue (succès, échec, instable, annulé).
- Exemple : #9 est le dernier à s'être terminé.

## Dernier build avec succès

- Définition : Le dernier build qui s'est **terminé avec succès, même s'il contient des tests instables**.

- Différence avec "stable" : Un build peut être "avec succès" mais **pas stable** s'il a des tests qui échouent de manière intermittente.

- Exemple : #9 est aussi "avec succès", donc il n'a pas échoué.

Build	Etat	Début
#9	Terminé	12:09
#8	Échec	12:09
#7	Échec	11:45
#6	Terminé	11:33

**Liens permanents**

- Dernier build (#9), il y a 7.8 s
- Dernier build stable (#9), il y a 7.8 s
- Dernier build avec succès (#9), il y a 7.8 s
- Dernier build en échec (#8), il y a 26 s
- Dernier build non réussi (#8), il y a 26 s
- Dernier build complété (#9), il y a 7.8 s



# TP1: Créer et compiler un projet Freestyle

[Analyse du résultat du build]

Statut	Inclut quels builds ?	Exemple
Dernier build stable	Réussi <b>et</b> sans tests instables	#9
Dernier build avec succès	Réussi, même avec tests instables	#9
Dernier build en échec	Échec pur	#8
Dernier build non réussi	Échec, instable ou annulé	#8
Dernier build complété	Tout build terminé	#9



# TP1: Créer et compiler un projet Freestyle

[les triggers]



- Aller dans la configuration du job
- Configurer l'exécution du job toutes les 2 minutes

## Configurer

### Général

Ce build a des paramètres ?

GitHub project

Permission to Copy Artifact

Supprimer les anciens builds ?

Throttle builds ?

Exécuter des builds simultanément si nécessaire ?

Avancé ▾

### Gestion de code source

### Gestion de code source

Connect and manage your code repository to automatically pull the latest code for your builds.

Aucune

Git ?

### Triggers

### Triggers

Set up automated actions that start your build based on specific events, like code changes or scheduled times.

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

Construire après le build sur d'autres projets ?

Construire périodiquement ?

GitHub hook trigger for GITScm polling ?

Scrutation de l'outil de gestion de version ?

Save

Appliquer



# TP2: Cr ation d'un projet maven

- Cr er un job freestyle
- Ajouter le d pot git  
<https://github.com/jpenekusu/helloworld-java-maven-app.git>

Jenkins / Job FreeStyle / Configuration

### Configurer

Aucune  
 Git ?

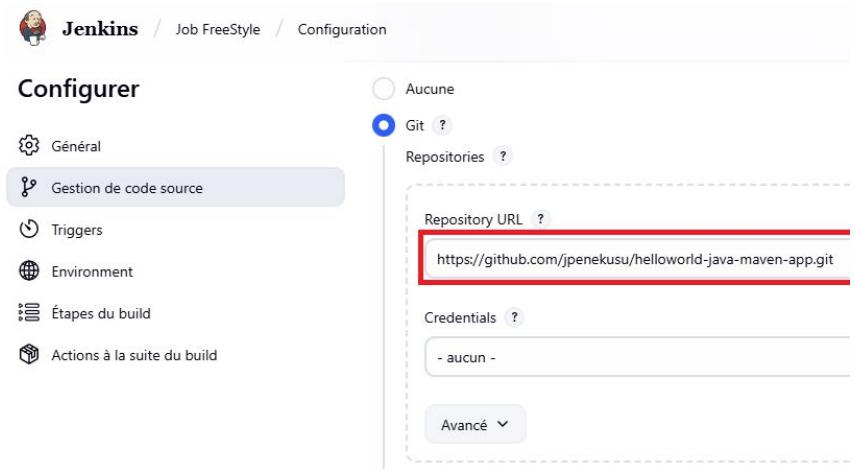
**Gestion de code source**

Repositories ?

Repository URL ?

Credentials ?  
- aucun -

Avanc 





# TP2: Crédit d'un projet maven

- Créer un job freestyle
- Ajouter le dépôt git  
<https://github.com/jpenekusu/helloworld-java-maven-app.git>
- Ajouter une étape de build maven

Jenkins / Job FreeStyle / Configuration

## Configurer

Général    Gestion de code source    Triggers    **Environment**    Étapes du build    Actions à la suite du build

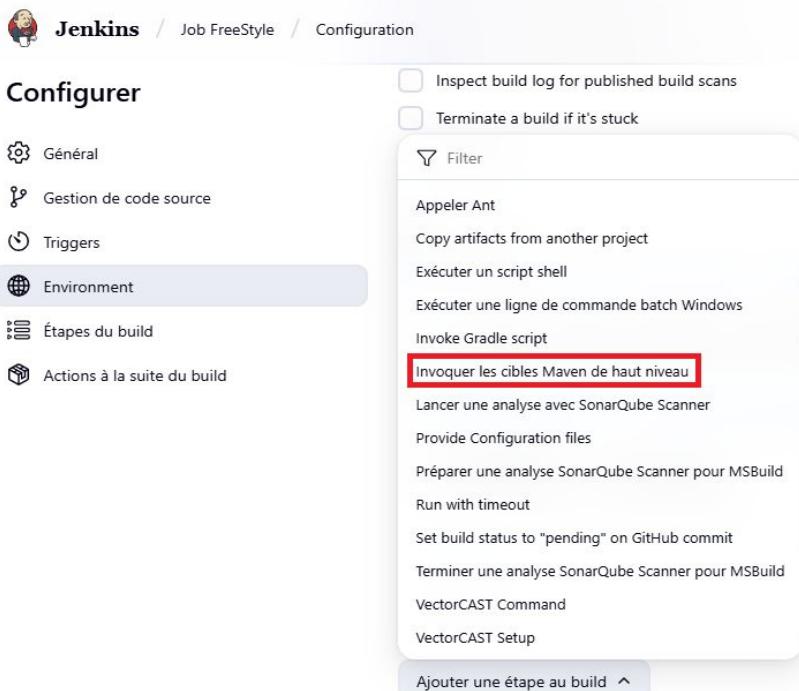
Inspect build log for published build scans  
 Terminate a build if it's stuck

Filter

Appeler Ant  
Copy artifacts from another project  
Exécuter un script shell  
Exécuter une ligne de commande batch Windows  
Invoke Gradle script  
**Invoquer les cibles Maven de haut niveau**  

Lancer une analyse avec SonarQube Scanner  
Provide Configuration files  
Préparer une analyse SonarQube Scanner pour MSBuild  
Run with timeout  
Set build status to "pending" on GitHub commit  
Terminer une analyse SonarQube Scanner pour MSBuild  
VectorCAST Command  
VectorCAST Setup

Ajouter une étape au build ^



# TP2: Crédit d'un projet maven

- Créer un job freestyle
- Ajouter le dépôt git  
<https://github.com/jpenekusu/helloworld-java-maven-app.git>
- Ajouter une étape de build maven
- Ajouter les goals maven install
- Sauvegarder et exécuter le build



# TP2: Cr ation d'un projet maven

- Le job  choue car maven n'est pas install 



Jenkins / Job freestyle / #1 / Console Output

```
First time build. Skipping changelog.  
[freestyle] $ mvn install  
FATAL: command execution failed  
java.io.IOException: error=2, No such file or directory  
        at java.base/java.lang.ProcessImpl.forkAndExec(Native Method)  
        at java.base/java.lang.ProcessImpl.<init>(Unknown Source)  
        at java.base/java.lang.ProcessImpl.start(Unknown Source)  
Caused: java.io.IOException: Cannot run program "mvn" (in directory  
"/var/jenkins_home/workspace/freestyle"): error=2, No such file or directory  
        at java.base/java.lang.ProcessBuilder.start(Unknown Source)  
        at java.base/java.lang.ProcessBuilder.start(Unknown Source)  
        at hudson.Proc$LocalProc.<init>(Proc.java:252)
```

# TP2: Crédit d'un projet maven

## [Installation de maven]

- Installation de maven

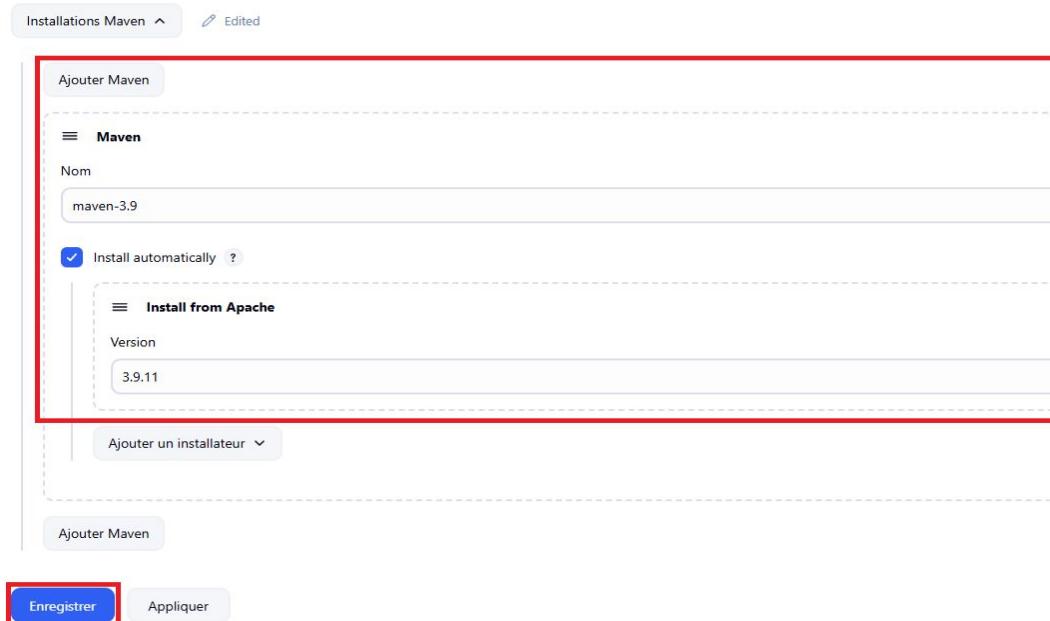
- Aller dans le menu Administration Jenkins  
=> Tools

The screenshot shows the Jenkins Administration interface. At the top right, there is a search bar with a magnifying glass icon (labeled 1) and a refresh button. Below the header, the page title is "Administrer Jenkins". The main content area is titled "Configuration du système". A red box highlights the "Tools" section, which contains a sub-section titled "Configure les outils, leur localisation et les installateurs automatiques." (Configure tools, their location and automatic installers). Other sections visible include "System" (Configure general parameters and file paths), "Clouds" (Add, remove and configure cloud instances to provision agents on demand), "Apparence générale" (Configure general appearance), "Security" (Secure Jenkins by defining who is authorized to access the system), "Credentials" (Configure credentials), "Users" (Create or modify users who can log in to this Jenkins server), "Informations sur le système" (Display various system information to help resolve problems), "Logs système" (The log system captures the java.util.logging output relative to Jenkins), "Statistiques" (Statistics), and "À propos de Jenkins" (About Jenkins). At the bottom left, there is a "Dépannage" (Debugging) section.

# TP2: Cr ation d'un projet maven

## [Installation de maven]

- Aller dans “Installations Maven”
  - Saisir un nom
  - Ajouter une installation automatique de maven



# TP2: Création d'un projet maven

[Installation de maven]

- Revenir dans la configuration du job freestyle
- Sélectionner la nouvelle version de maven dans le champ Maven Version
- Enregistrer le job

## Build Steps

Automate your build process with ordered tasks like code compilation, testing, and deployment.

Invoke top-level Maven targets ?

Maven Version

maven-3.9

Goals

install

Advanced ▾

+ Add build step

Save Apply



# TP2: Crédit d'un projet maven

## [Installation de maven]

- Exécuter le job
- Le projet maven est compilé avec succès

The screenshot shows a Jenkins job named "Job freestyle" with build number "#3". The "Console Output" tab is selected. The log output shows the Maven build process:

```
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/resolver/maven-resolver-api/1.9.18/maven-resolver-api-1.9.18.jar (157 kB at 1.6 MB/s)
Progress (1): 86/93 kB
Progress (1): 93 kB

Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-xml/3.0.0/plexus-xml-3.0.0.jar (93 kB at 861 kB/s)
[INFO] Installing /var/jenkins_home/workspace/Job freestyle/pom.xml to
/var/jenkins_home/.m2/repository/com/mycompany/app/my-app/1.0-SNAPSHOT/my-app-1.0-SNAPSHOT.pom
[INFO] Installing /var/jenkins_home/workspace/Job freestyle/target/my-app-1.0-SNAPSHOT.jar to
/var/jenkins_home/.m2/repository/com/mycompany/app/my-app/1.0-SNAPSHOT/my-app-1.0-SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time:  54.615 s
[INFO] Finished at: 2025-10-19T18:26:50Z
[INFO] -----
Finished: SUCCESS
```



# Intégration avec github

[ Pourquoi intégrer Jenkins à GitHub]

## Objectif : Automatiser le cycle de développement

- Déclenchement automatique des builds à chaque `push` ou `pull request`
- Feedback immédiat sur la qualité du code (tests, analyse statique)
- Traçabilité : chaque build est lié à un commit spécifique
- Culture DevOps : collaboration fluide entre dev et ops

 « Un bon pipeline commence par un bon déclenchement. »

# TP3: Intégration avec github

[Création d'un identifiant -un token- github]

- Aller sur <http://github.com>
- Se connecter à son compte github
- Aller dans le menu Settings

The screenshot shows a GitHub profile page for a user named 'jpenekusu'. The user menu is open, with two specific items highlighted with red boxes and numbers: 'Settings' (labeled 2) and 'Copilot settings'.

**User Profile:** jpenekusu (1 follower)

**Profile Options:**

- Watch 18
- Fork 32.1k
- Add file
- Code
- About
- Profile
- Repositories
- Stars
- Gists
- Organizations
- Enterprises
- Sponsors
- Settings (2)
- Copilot settings
- Feature preview
- Appearance
- Accessibility
- Try Enterprise
- Sign out

**Recent Activity:**

- org.junit... · last week · 104 Commits
- Configuration with prebuilds and repro... · 3 weeks ago
- using a Docker image anymore. · 7 months ago
- · 4 months ago
- · 8 years ago
- 17 onwards · 3 years ago
- ... · 3 weeks ago
- ... · last year
- · 3 years ago
- into dependabot/maven/org.junit/jup... · last week



# TP3: Intégration avec github

[Création d'un identifiant -un token- github]

Code, planning, and automation

- Repositories
- Codespaces
- Models
- Packages
- Copilot
- Pages
- Saved replies

Security

- Code security

Integrations

- Applications
- Scheduled reminders

Archives

- Security log
- Sponsorship log

Developer settings

Social accounts

- Link to social profile 1
- Link to social profile 2
- Link to social profile 3
- Link to social profile 4

Company

You can @mention your company's GitHub organization to link it.

Location

Display current local time  
Other users will see the time difference from their local time.

ORCID ID

ORCID provides a persistent identifier - an ORCID ID - that distinguishes you from other researchers. Learn more at [ORCID.org](#).

Connect your ORCID ID

All of the fields on this page are optional and can be deleted at any time, and by filling them out, you're giving us consent to share this data wherever your user profile appears. Please see our [privacy statement](#) to learn more about how we use this information.

Update profile

# TP3: Intégration avec github

[Création d'un identifiant -un token- github]

The screenshot shows the GitHub Developer Settings page. At the top, there are links for "Settings" and "Developer Settings". A search bar on the right has the placeholder "Type". Below the header, there are three main categories: "GitHub Apps", "OAuth Apps", and "Personal access tokens". The "Personal access tokens" category is expanded, showing "Fine-grained tokens" and a sub-section for "Tokens (classic)". A red box labeled "1" highlights the "Tokens (classic)" section. To the right, under "Personal access tokens (classic)", it says "No personal access token created" and provides instructions to "Generate a personal access token GitHub API". A "Generate new token" button is visible. A dropdown menu for "Generate new token" is open, showing two options: "Generate new token" (Fine-grained, repo-scoped) and "Generate new token (classic)" (For general use). A red box labeled "2" highlights the "Generate new token (classic)" option. A note at the bottom explains that "Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of" and points to "authenticate to the API over Basic Authentication".

# TP3: Intégration avec github

[Création d'un identifiant -un token- github]

<input type="checkbox"/> read:network_configurations	Read org hosted compute network configurations
<input type="checkbox"/> project	Full control of projects
<input type="checkbox"/> read:project	Read access of projects
<input type="checkbox"/> admin:gpg_key	Full control of public user GPG keys
<input type="checkbox"/> write:gpg_key	Write public user GPG keys
<input type="checkbox"/> read:gpg_key	Read public user GPG keys
<input type="checkbox"/> admin:ssh_signing_key	Full control of public user SSH signing keys
<input type="checkbox"/> write:ssh_signing_key	Write public user SSH signing keys
<input type="checkbox"/> 4 read:ssh_signing_key	Read public user SSH signing keys

**4**

Settings / Developer Settings  

GitHub Apps  
 OAuth Apps  
 Personal access tokens  
  Fine-grained tokens  
  Tokens (classic)

### New personal access token (classic)

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

**Note** 1   
What's this token for?

**Expiration** 2   
The token will expire on the selected date

**Select scopes**  
Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

repo 3  repo:status Full control of private repositories  
 repo:deployment Access commit status  
 public\_repo Access deployment status  
 repo:invite Access public repositories  
 security\_events Access repository invitations  
 Read and write security events

workflow Update GitHub Action workflows

write:packages Upload packages to GitHub Package Registry  
 read:packages Download packages from GitHub Package Registry



# TP3: Intégration avec github

[Création d'un identifiant -un token- github]

- Copier le token

The screenshot shows the GitHub Developer Settings page under Personal access tokens (classic). A red box highlights the token 'ghp\_U1DQF2r...'. A tooltip says: 'Make sure to copy your personal access token now. You won't be able to see it again!'. Below the token, a note states: 'Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.'

Settings / Developer Settings

Type to search

Some of the scopes you've selected are included in other scopes. Only the minimum set of necessary scopes has been saved.

GitHub Apps

OAuth Apps

Personal access tokens

Fine-grained tokens

Tokens (classic)

Personal access tokens (classic)

Generate new token

ghp\_U1DQF2r...  
Delete

Make sure to copy your personal access token now. You won't be able to see it again!

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.



# TP3: Intégration avec github

[Création de l'identifiant github dans Jenkins]

- Aller dans le menu administration Jenkins

The screenshot shows the Jenkins Administration interface. At the top, there's a navigation bar with the Jenkins logo and the text "Administrer Jenkins". Below it, a sidebar lists various configuration sections: "Configuration du système", "Sécurité", "Information du statut", and "Dépannage". The "Sécurité" section contains a link to "Credentials Configure credentials", which is highlighted with a red box. This link leads to a page where you can manage Jenkins' credentials, including adding GitHub authentication.



# TP3: Intégration avec github

[Création de l'identifiant github dans Jenkins]

 Jenkins / Administrer Jenkins / Identifiants

## Credentials

T	P	Store ↓	Domain	ID
		System	(global)	github
		System	(global)	github_token

## Stores scoped to Jenkins

P	Store ↓	Domains
	System	(global)

Icône: S L

 Jenkins / Administrer Jenkins / Identifiants / System

## System

Domaine ↓

 Identifiants globaux (illimité)

Icône: S M L

2



# TP3: Intégration avec github

[Création de l'identifiant github dans Jenkins]



Jenkins

/ Administrer Jenkins

/ Identifiants

/ System

/ Identifiants globaux (illimité)

## Identifiants globaux (illimité)

+ Add Credentials

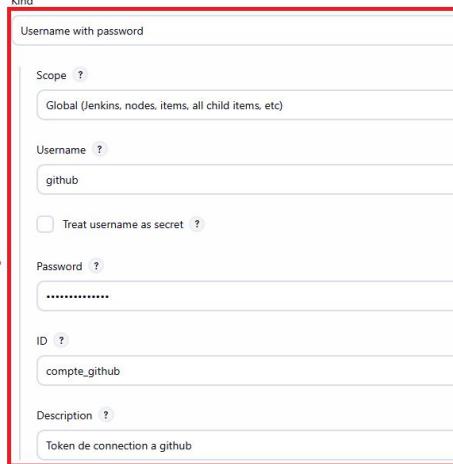
Credentials that should be available irrespective of domain specification to requirements matching.

ID	Nom	Type	Description
Ce domaine d'identifiants est vide. Que diriez-vous d'ajouter des identifiants ?			

Icône: S M L

# TP3: Intégration avec github

[Création de l'identifiant github dans Jenkins]



Jenkins / Manage Jenkins / Credentials / System / Global credentials (unrestricted)

New credentials

Kind: Username with password

Scope: Global (Jenkins, nodes, items, all child items, etc)

Username: github

Treat username as secret

Password:

ID: compte.github

Description: Token de connection a github

Create

1

- Créer un credential de type Nom d'utilisateur et mot de passe
- Renseigner le champ Username par un terme parlant (exemple github)
- Coller le token dans le champ mot de passe
- Renseigner l'ID du token (identifiant technique) et le commentaire

Jenkins / Administrer Jenkins / Identifiants / System / Identifiants globaux (illimité)

## Identifiants globaux (illimité)

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Nom
	github_token
	compte.github

2

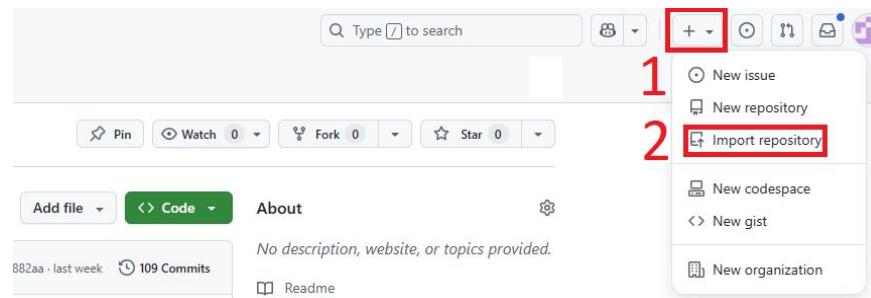
jenkins/\*\*\*\*\* (Token de connection a github)



# TP3: Intégration avec github

[Création d'un dépôt github privé par import du dépôt public du TP2]

- Créer un dépôt github privé en important le dépôt git du TP2





# TP3: Intégration avec github

[Création d'un dépôt github privé par import d'un dépôt public]

- Reprendre le projet du TP2
- Créer un dépôt github privé en important le dépôt git du TP2

## Your source repository details

The URL for your source repository \*

`https://github.com/jpenekusu/helloworld-java-maven-app.git`

Learn more about [importing git repositories](#).

1

Please enter your credentials if required for cloning your remote repository.

Your username for your source repository

Your access token or password for your source repository

## Your new repository details

Owner \*

jpenekusu

Repository name \*

helloworld-java-maven-ap

2

Public

Anyone on the internet can see this repository. You choose who can commit.

Private

You choose who can see and commit to this repository.

3

(i) You are creating a private repository in your personal account.

Cancel

Begin import

# TP3: Intégration avec github

[ Configuration de l'identifiant github dans un job]

- Remplacer le dépôt github du TP2 par le dépôt git privé nouvellement créé
- Constater l'échec de connection au dépôt github avant la saisie du credential





# TP3: Intégration avec github

[ Configuration de l'identifiant github dans un job]

- Ajouter le credential compte\_github au projet du TP2
- Compiler le projet et observer le résultat





# TP4: Installation des plugins

Aller dans “Manage Jenkins” => Plugins => Available plugins puis:

- Rechercher puis sélectionner les plugins Local et Maven Integration
- Cliquer sur Installer
- Cocher la case redémarrer Jenkins à la fin de l’installation



# TP4: Installation des plugins

[Configuration de la langue avec le plugin Local]

- Aller dans “Manage Jenkins” => Appearance
- Changer la langue dans la section “Default Language”.

Jenkins / Manage Jenkins / Appearance

## Appearance

---

### Theme

Select a theme to change how Jenkins appears.

Light

Dark

Dark (System)

Do not allow users to select a different theme

Default Language ?

French - fr



# Authentification et autorisations

Sécuriser votre instance Jenkins avec des stratégies d'authentification et de contrôle d'accès



# Authentification et autorisations

[Objectifs pédagogiques]

- Configurer l'authentification dans Jenkins
- Choisir et implémenter une stratégie d'autorisation adaptée
- Intégrer Jenkins avec des systèmes externes (LDAP, OAuth, etc.)
- Appliquer le principe du moindre privilège
- Auditer les accès et permissions





# Authentification et autorisations

[Pourquoi sécuriser Jenkins]

- Accès non contrôlé au pipeline CI/CD
- Exécution de code malveillant via des jobs
- Fuite de secrets (credentials, tokens, etc.)
- Modification non autorisée de la configuration

⚠ Jenkins n'est pas sécurisé par défaut !



# Authentification et autorisations

[Les bases]

- Accéder à [Manage Jenkins > Configure Global Security](#)
- Activer *Security Realm* (authentification)
- Choisir *Authorization Strategy* (autorisation)
- Sauvegarder et tester





# Authentification et autorisations

[Security Realm – Authentification]

- Jenkins's own user database → Utilisateurs locaux
- LDAP / Active Directory → Annuaire d'entreprise
- GitHub / GitLab OAuth → Auth via compte Git
- SAML → SSO d'entreprise

 Plugin requis pour LDAP/AD/SAML/OAuth.





# Authentification et autorisations

[Authorization Strategy Autorisations]

- Anyone can do anything → À éviter
- Logged-in users can do anything → Trop permissif
- Matrix-based security → Contrôle fin (petites équipes)
- Role-Based Strategy → Recommandé (plugin)



# Authentification et autorisations

[Matrix-Based Security]

- Permissions définies **par utilisateur ou groupe**
- Granularité : Job, Run, Overall, Agent...
- Bon pour les petites équipes
- Devient **complexe à grande échelle**



# Authentification et autorisations

[Le plugin Role-Based Strategy]

- Plugin : Role-based Authorization Strategy
- 3 types de rôles :
  - a. Global roles – Accès global
  - b. Item roles – Accès aux jobs (regex)
  - c. Agent roles – Accès aux nœuds
- Centralisé, réutilisable, scalable





# Authentification et autorisations

[Intégration avec LDAP]

## Objectifs pédagogiques

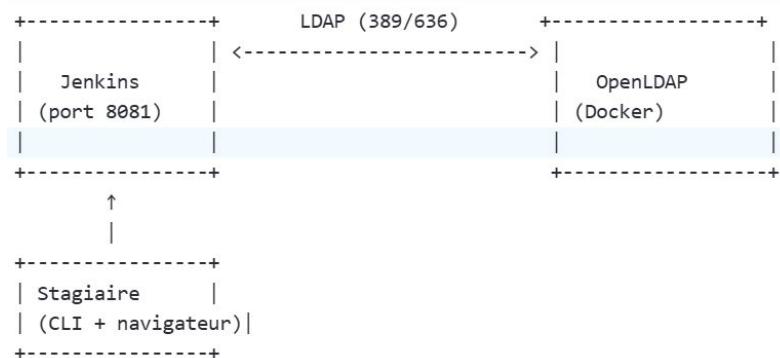
- Déployer un serveur LDAP avec Docker
- Créer des utilisateurs et des groupes (`admins`, `developers`, `testers`)
- Configurer Jenkins pour l'authentification LDAP
- Gérer les autorisations par groupe (Matrix-based security)
- Sécuriser la connexion avec LDAPS (optionnel avancé)



# Authentification et autorisations

[Intégration avec LDAP]

Architecture du TP (Architecture technique):





# Authentification et autorisations

[TPs]

Cfr. fichier TPs TPs\_Jenkins\_Authentification\_et\_autorisations.pdf en annexe

N.B: pour éviter de rester bloqué à l'extérieur de Jenkins, avant de changer ou configurer des nouveaux droits d'un administrateur Jenkins:

- garder la session de l'administrateur courant active (ex. avec le navigateur chrome)
- ouvrir une nouvelle session avec un autre navigateur (ex. Edge)
- configurer les nouveaux droits.





# Authentification et autorisations

[Bonnes pratiques]

- Ne jamais utiliser le compte admin par défaut en production
- Utiliser des groupes (LDAP/AD) plutôt que des utilisateurs individuels
- Appliquer le principe du moindre privilège
- Auditer régulièrement les permissions
- Tester les accès avec un compte non-admin



# Authentification et autorisations

[Résumé & Prochaines étapes]

- Authentification ≠ Autorisation
- Choisissez le bon *Security Realm* et *Authorization Strategy*
- Privilégiez les rôles pour une gestion scalable





# Installation d'agents dans le cluster Jenkins

[Scalabilité, sécurité et partage de ressources]



# Installation d'agents dans le cluster Jenkins

[Pourquoi un cluster d'agents]

**Le master ne fait pas les builds !**

- Master Jenkins = cerveau
  - Orchestration, UI, planification
- Agents (nœuds) = force de travail
  - Exécution des jobs, tests, déploiements





# Installation d'agents dans le cluster Jenkins

[Pourquoi un cluster d'agents]

**Le master ne fait pas les builds !**

✓ Avantages :

- Parallélisation des builds
- Isolation des environnements
- Sécurité renforcée

⚠ Bonne pratique : Désactiver les exécuteurs sur le master en production





# Installation d'agents dans le cluster Jenkins

[Connexion des agents – Clés SSH]

## Méthode standard : SSH avec clé privée

1. Générer une paire de clés sur le master

```
$ ssh-keygen -t rsa -b 4096 -C "jenkins"
```

2. Déposer la clé publique sur l'agent

→ [~jenkins/.ssh/authorized\\_keys](~jenkins/.ssh/authorized_keys)

3. Dans Jenkins :

- Nouveau nœud → Permanent Agent
- Lancement : via SSH
- Spécifier IP, utilisateur, clé privée



Utilisez un utilisateur dédié ([jenkins](#)) avec droits minimaux (least privilege)



# Installation d'agents dans le cluster Jenkins

[TPs]

Voir doc de TPs ConnexionDunAgentJenkinsViaSSH\_CentOS9.pdf en annexes





# Installation d'agents dans le cluster Jenkins

[Répartition des outils entre agents]

**Évitez la surcharge inutile !**

JDK, Maven	Sur agents Java	Dépendances de build
Node.js	Sur agents frontend	Builds JS/TS
Docker CLI	Sur agents Docker	Builds conteneurisés
.NET, Python	Sur agents spécialisés	Éviter la pollution globale





# Installation d'agents dans le cluster Jenkins

[Dimensionnement du cluster]

## Combien d'agents ? De quelle puissance ?

Facteurs clés :

- Nombre de jobs simultanés
- Type de workload (CPU, RAM, I/O)
- Temps de réponse attendu (SLA)

📌 Règles empiriques :

- 1 agent = 2 à 4 exécuteurs
- Agents CPU-lourds →  $\geq 4$  vCPU, 8 Go RAM
- Agents légers → 2 vCPU, 4 Go RAM



Outil recommandé : Prometheus + Grafana pour le monitoring

# Installation d'agents dans le cluster Jenkins

[Espace disque partagé]

## Pourquoi partager le stockage ?

Problème :

→ Les workspaces et artefacts consomment beaucoup d'espace

Solutions :

- NFS (Linux) – simple et efficace
- SMB/CIFS – pour Windows
- Stockage objet (S3/MinIO) – pour artefacts
- Volumes Docker – en environnement conteneurisé





# Installation d'agents dans le cluster Jenkins

[Espace disque partagé]

## ✓ Avantages :

- Centralisation
- Nettoyage facilité
- Accès multi-jobs



Attention : Latence réseau & permissions ([chmod](#), [chown](#))



# Installation d'agents dans le cluster Jenkins

[Bonnes pratiques d'administration]

## Checklist admin Jenkins

- 0 exécuteur sur le master
- Utiliser des labels (`linux`, `docker`, `maven`)
- Nettoyer les workspaces automatiquement
- Surveiller l'espace disque (`df -h`)
- Sauvegarder `$JENKINS_HOME/nodes/`
- Préférer les agents éphémères (Docker, Kubernetes)



Bonus : Déclarez vos agents avec Jenkins Configuration as Code (JCasC)



# Installation d'agents dans le cluster Jenkins

[Résumé & perspectives]

## Points clés à retenir

- Agents = exécuteurs, master = orchestrateur
- SSH + clés = connexion sécurisée standard
- Outils répartis selon les besoins métier
- Dimensionnement basé sur la charge réelle
- Stockage partagé = utile, mais à gérer avec soin



Bonus : Déclarez vos agents avec Jenkins Configuration as Code (JCasC)

# Stratégies de sauvegarde

[Anatomie du répertoire `$JENKINS_HOME`]

**Objectif : Comprendre ce qui doit être sauvegardé.**

Le répertoire `$JENKINS_HOME` contient toute la configuration et l'état de votre instance Jenkins :

- `config.xml` : configuration globale de Jenkins
- `jobs/` : définitions, builds et historiques des jobs
- `plugins/` : plugins installés (`.jpi` ou `.hpi`)
- `users/` : comptes utilisateurs et préférences
- `nodes/` : configuration des agents (nœuds)
- `secrets/` : clés de chiffrement, credentials chiffrés
- `workspace/ (optionnel)` : espaces de travail des builds (souvent exclus de la sauvegarde)

💡 Bon à savoir : Sauvegarder `$JENKINS_HOME` = sauvegarder l'instance Jenkins.



# Stratégies de sauvegarde

[Choix de la stratégie de sauvegarde]

**Objectif :** Définir une stratégie adaptée au contexte métier.

**Options possibles :**

- Sauvegarde manuelle (copie du dossier `JENKINS_HOME`) → simple mais risquée
- Sauvegarde automatisée (scripts + tâches planifiées) → plus fiable
- Plugins dédiés :
  - *ThinBackup* (legacy, non maintenu)
  - *Periodic Backup Plugin* (à évaluer avec prudence)
- Intégration dans une solution d'entreprise (Veeam, Bacula, etc.)





# Stratégies de sauvegarde

[Choix de la stratégie de sauvegarde]

## Bonnes pratiques :

- Exclure les dossiers temporaires ([workspace/](#), [builds/\\*/archive/](#) si volumineux)
- Sauvegarder hors du serveur Jenkins (stockage distant, cloud, NAS...)
- Vérifier régulièrement l'intégrité des sauvegardes

⚠ Attention : Jenkins doit être arrêté ou mis en mode *quiet* pour une sauvegarde cohérente.





# Plan de reprise d'activité (PRA) pour Jenkins

[Minimiser le temps d'indisponibilité en cas de sinistre.]

## Étapes clés du PRA :

1. Restauration du système (OS, Java, Jenkins, permissions)
2. Restauration de `$JENKINS_HOME` depuis la dernière sauvegarde valide
3. Redémarrage de Jenkins avec le même utilisateur/service
4. Valider : accès web, jobs, credentials, agents

⚠️ Attention : Jenkins doit être arrêté ou mis en mode *quiet* pour une sauvegarde cohérente.





# Plan de reprise d'activité (PRA) pour Jenkins

[Minimiser le temps d'indisponibilité en cas de sinistre.]

## Recommandations:

- Documenter la procédure de restauration pas à pas
- Tester le PRA régulièrement (ex. : environnement de staging)
- Prévoir un plan de basculement (ex. : instance Jenkins de secours)
- Définir des objectifs clairs :
  - a. RPO (Recovery Point Objective) : ex. 24h max de données perdues
  - b. RTO (Recovery Time Objective) : ex. remise en service en  $\leq$  2h





# Intégration avancée avec GitLab, Bitbucket et Jira

[Intégration Jenkins avec les hébergeurs Git.]

**Objectif : Automatiser les pipelines CI/CD à partir de GitLab ou Bitbucket.**

Plugins clés :

- GitLab Plugin + GitLab Branch Source Plugin
- Bitbucket Branch Source Plugin
- Jira Plugin (pour l'intégration des tickets)





# Intégration avancée avec GitLab, Bitbucket et Jira

[Intégration Jenkins avec les hébergeurs Git.]

**Objectif : Automatiser les pipelines CI/CD à partir de GitLab ou Bitbucket.**

Ces plugins permettent :

- La découverte automatique des dépôts, branches et pull/merge requests
- Le déclenchement de builds à chaque push ou MR/PR
- La mise à jour automatique des tickets Jira



Ces fonctionnalités s'appuient sur les Webhooks et les API tokens.



# Intégration avancée avec GitLab, Bitbucket et Jira

[Découverte automatique de l'arborescence.]

## Fonctionnement :

Jenkins scanne régulièrement un projet (ou groupe/projet) sur GitLab ou Bitbucket pour :

- Déetecter de nouveaux dépôts
- Créer automatiquement des jobs Multibranch Pipeline
- Gérer les branches, tags et merge/pull requests comme des pipelines séparés





# Intégration avancée avec GitLab, Bitbucket et Jira

[Découverte automatique de l'arborescence.]

## Configuration requise :

- Un compte technique avec accès API (token)
- Une organisation ou groupe/projet configuré dans Jenkins
- Le plugin Branch Source correspondant activé

Idéal pour les équipes avec de nombreux dépôts (« GitOps at scale »).





# Intégration avancée avec GitLab, Bitbucket et Jira

[Build-on-push avec GitLab ou Bitbucket.]

**Principe : Déclencher un build dès qu'un commit est poussé.**

Étapes de configuration :

1. Dans Jenkins : créer un **Multibranch Pipeline** avec source GitLab/Bitbucket
2. Dans GitLab/Bitbucket : configurer un **Webhook** pointant vers  
[`https://<jenkins-url>/project/<nom-du-job>`](https://<jenkins-url>/project/<nom-du-job>)
3. Ajouter un **secret token** pour sécuriser le webhook
4. Donner les droits nécessaires au token Jenkins (« `read_repository` », « `webhook` »...)





# Intégration avancée avec GitLab, Bitbucket et Jira

[Build-on-push avec GitLab ou Bitbucket.]

**Principe : Déclencher un build dès qu'un commit est poussé.**

Avantages :

- Builds immédiats sans polling
- Réduction de la latence CI
- Meilleure réactivité aux contributions



Vérifiez la connectivité réseau (firewall, proxy, certificats).





# Intégration avancée avec GitLab, Bitbucket et Jira

[Mise à jour automatique des tickets Jira.]

**Objectif : Lier les builds Jenkins aux tickets Jira pour la traçabilité.**

Fonctionnalités offertes par le plugin Jira :

- Détection automatique des clés Jira dans les messages de commit (ex. PROJ-123)
- Mise à jour du statut du ticket (ex. « In Progress », « Tested »)
- Ajout de commentaires avec lien vers le build, logs, artefacts
- Échec du build → notification dans le ticket





# Intégration avancée avec GitLab, Bitbucket et Jira

[Mise à jour automatique des tickets Jira.]

**Objectif : Lier les builds Jenkins aux tickets Jira pour la traçabilité.**

Prérequis :

- Plugin Jira installé et configuré dans Jenkins
- Compte technique Jira avec droits d'écriture
- Jenkins autorisé à accéder à l'API Jira (token ou login/mot de passe)



Cette intégration renforce la collaboration Dev + Ops + QA.





# Intégration avancée avec GitLab, Bitbucket et Jira

[Conclusions.]

## En résumé.

- ✓ Jenkins s'intègre nativement avec GitLab et Bitbucket grâce aux Branch Source Plugins.
- ✓ La découverte automatique des dépôts simplifie la gestion à grande échelle.
- ✓ Le build-on-push via webhooks rend la CI réactive et efficace.
- ✓ L'intégration Jira assure traçabilité, transparence et collaboration.





# Intégration avancée avec GitLab, Bitbucket et Jira

[Voir doc de TPs en annexes]

- TP1\_CréationDunGroupeGitLabPrivéAvecTroisProjetsCI
- TP2\_CréationDunPersonalAccessTokenGitLabEtIntégrationDan Jenkins
- TP3\_ConfigurationDeLauthentificationSSHEntreGitLabEtJenkinsPourGitLabFolderOrganization
- TP4\_ConfigurerUnWebhookGitLabJenkinsPourLindexationEnTempsRéal
- TP5\_IntégrationJenkinsAvecJiraMiseAJourAutomatiqueDesTickets





# Intégration avancée avec GitLab, Bitbucket et Jira

[Conclusions.]

## Bonnes pratiques :

- Utiliser des comptes techniques avec droits minimaux
- Sécuriser les webhooks avec des tokens secrets
- Tester les flux d'intégration dans un environnement isolé



Une intégration bien configurée = pipelines fluides, équipes alignées, livraisons fiables.



# Projet Freestyle vs Pipeline

Critère	Projet freestyle	Pipeline
Configuration	Via UI	Dans un fichier texte ( <a href="#">Jenkinsfile</a> )
Versionnable	✗ Non	✓ Oui
Complexité	Simple à modéré	Adapté aux workflows complexes
Représentation visuelle	Basique	Stage View, Blue Ocean
Maintenance	Manuelle	Automatisée avec le code



# Mutualisation des pipelines - les Shared Libraries

[Pourquoi mutualiser les pipelines ?]

**Objectif : Éviter la duplication, garantir la cohérence et simplifier la maintenance.**

Avantages des librairies partagées (Shared Libraries) :

- Centralisation de la logique CI/CD (ex. : déploiement, tests, scans)
- Réutilisation par tous les pipelines de l'organisation
- Mise à jour centralisée → impact immédiat sur tous les jobs
- Meilleure gouvernance (sécurité, conformité, bonnes pratiques)



Idéal pour les équipes DevOps ou les plateformes internes (Internal Developer Platform).

# Mutualisation des pipelines - les Shared Libraries

[Structure d'une librairie partagée.]

Une librairie Jenkins est un dépôt Git avec une arborescence standard :

```
(root)
├── src/           ← Classes Groovy réutilisables
│   └── org/
│       └── entreprise/
│           └── Deploy.groovy
├── vars/          ← Fonctions globales (appelables dans le pipeline)
│   ├── buildApp.groovy
│   └── notifySlack.groovy
└── resources/     ← Fichiers statiques (templates, scripts...)
    └── k8s/deployment.yaml
```

- ✓ La librairie est chargée automatiquement ou explicitement dans les pipelines.

# Mutualisation des pipelines - les Shared Libraries

[Syntaxe Groovy dans les pipelines.]

Les Shared Libraries s'appuient sur Groovy, un langage dynamique compatible Java.

Exemple dans `vars/buildApp.groovy` :

groovy

```
1 def call(String appVersion = 'latest') {  
2     sh "docker build -t myapp:${appVersion} ."  
3     sh "docker push myapp:${appVersion}"  
4 }
```

- ✓ La librairie est chargée automatiquement ou explicitement dans les pipelines.

# Mutualisation des pipelines - les Shared Libraries

[Syntaxe Groovy dans les pipelines.]

Les Shared Libraries s'appuient sur Groovy, un langage dynamique compatible Java.

Exemple dans `vars/buildApp.groovy` :

groovy

```
1 def call(String appVersion = 'latest') {  
2     sh "docker build -t myapp:${appVersion} ."  
3     sh "docker push myapp:${appVersion}"  
4 }
```

- ✓ La librairie est chargée automatiquement ou explicitement dans les pipelines.



# Mutualisation des pipelines - les Shared Libraries

[Syntaxe Groovy dans les pipelines.]

## Utilisation dans un pipeline :

```
groovy
1 @Library('mon-librairie') _
2 pipeline {
3     agent any
4     stages {
5         stage('Build') {
6             steps {
7                 buildApp('1.2.3')
8             }
9         }
10    }
```

⚠ Respecter les conventions Groovy (visibilité, typage, gestion d'erreurs).



# Mutualisation des pipelines - les Shared Libraries

[Tests unitaires des librairies.]

**Objectif : Valider le comportement des fonctions avant déploiement.**

Outils recommandés :

- Jenkins Pipeline Unit Test Framework  
([com.lesfurets:jenkins-pipeline-unit](#))
- Framework de test Groovy/Spock ou JUnit





# Mutualisation des pipelines - les Shared Libraries

[Tests unitaires des librairies.]

## Bonnes pratiques :

- Tester les `vars/` comme des fonctions (mock des étapes Jenkins : `sh`, `echo`, etc.)
- Automatiser les tests via un pipeline dédié ou un CI externe
- Couvrir les cas d'erreur (ex. : échec de `docker push`)





# Mutualisation des pipelines - les Shared Libraries

[Conclusion.]

En résumé:

- ✓ Les Shared Libraries permettent de mutualiser, standardiser et sécuriser la CI/CD.
- ✓ Structure claire (`src/`, `vars/`, `resources/`) → facilité de maintenance.
- ✓ Syntaxe Groovy puissante mais exigeante → documentation et revue de code essentielles.
- ✓ Les tests unitaires garantissent la robustesse des librairies.





# Mutualisation des pipelines - les Shared Libraries

[Conclusion.]

## Prochaines étapes :

- Créer une librairie d'entreprise versionnée
- Documenter les fonctions disponibles
- Intégrer les tests dans le pipeline de la librairie elle-même

✖ « Un bon pipeline commence par une bonne librairie. »





---

Merci !

