Chapitre 5

L'intégration continue en utilisant Jenkins (Continuous Integration CI)

Enseignante: Dr-Ing. Amina JARRAYA
Email : amina.jarraya@ensi-uma.tn
Niveau: II3- GL

Plan

1. Introduction à l'intégration continue (CI)
2. Présentation de Jenkins
3. Installation de Jenkins
4. Gestion des jobs free-style sous Jenkins
5. Gestion de pipelines sous Jenkins
6. Création des pipelines déclaratives sous Jenkins
7. Github Webhook et Jenkins
8. Mise en place d'un pipeline CI/CD

1. Introduction à l’Intégration Continue (CI)

Limites de l’intégration traditionnelle

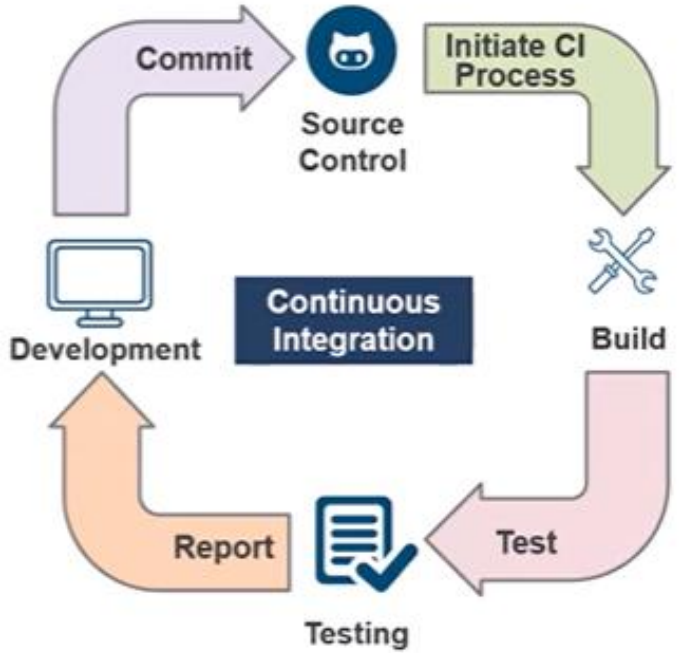
L’intégration traditionnelle se déroule généralement à la fin du développement d'un projet, une fois que les différents modules ont été développés et testés individuellement.

- Implique l'assemblage de tous les modules, suivi de tests, de corrections et de nouveaux tests en boucle jusqu'à ce que tout soit correct.
- Les corrections de bugs en fin de projet peuvent être coûteuses et complexes car elles impliquent souvent des modifications importantes du code.
- Manque souvent de tests exhaustifs, ce qui peut entraîner des versions logicielles avec des bugs persistants.

L’intégration continue (CI)

- L'intégration continue (CI) est un processus qui consiste à intégrer fréquemment les modifications de code dans un référentiel centralisé, suivi de la création et de l'exécution automatisée de tests.

- L'objectif est de détecter rapidement les erreurs et les conflits entre les différentes contributions, favorisant ainsi une collaboration plus efficace et une meilleure qualité du code.



Avantages de l’intégration continue

- **Productivité accrue:** L'automatisation des tâches manuelles telles que la compilation, les tests et le déploiement libère les développeurs pour se concentrer sur des tâches plus complexes et créatives.
- **Détection précoce des erreurs:** Les tests automatisés intégrés à la CI permettent de détecter rapidement les problèmes de code, empêchant ainsi les erreurs de se propager et de devenir plus coûteuses à corriger plus tard.
- **Amélioration de la qualité du code:** Les tests fréquents et les retours d'information constants aident à identifier et à corriger les erreurs plus rapidement, ce qui se traduit par une meilleure qualité du code.
- **Livraisons plus rapides:** En automatisant le processus de construction et de test, la CI permet de livrer des mises à jour plus fréquemment et plus rapidement, ce qui permet aux entreprises de répondre plus efficacement aux besoins du marché.
- **Réduction des coûts:** L'automatisation de la CI permet de réduire le temps et les ressources nécessaires pour les tests et les déploiements, ce qui peut entraîner des économies significatives.

Outils de l’intégration continue

Pourquoi Jenkins ?

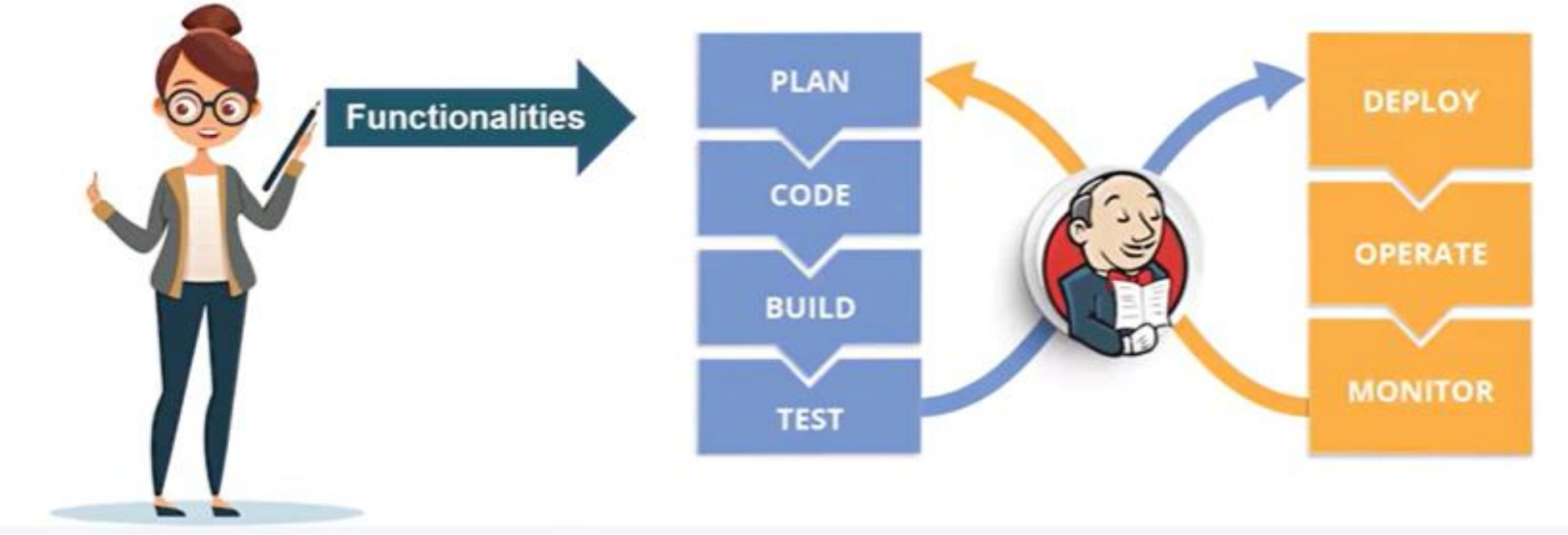
Google Trends: Jenkins VS Travis VS Atlassian Bamboo

Les entreprises utilisant Jenkins

2. Présentation de Jenkins

Jenkins c’est quoi ?

- **Jenkins** est un serveur d'automatisation open source utilisé principalement pour l'intégration et la livraison continues (CI/CD) dans le développement logiciel.
- Il automatise les tâches de **build**, de **test** et de **déploiement**, facilitant ainsi l'intégration continue du code et la livraison rapide des changements.
- Écrit en Java, Jenkins peut être étendu via des plugins pour s'intégrer à divers outils et environnements.



Les cas d’utilisation de Jenkins

Les projets dans Jenkins

- En Jenkins, un "**job**" est une tâche automatisée que vous configurez pour construire, tester et déployer votre projet. C'est une unité fondamentale de travail qui permet d'exécuter une séquence d'actions (telles que la compilation du code, l'exécution de tests, ou le déploiement de l'application).

Les différents types de projet dans Jenkins

Free Style Project

Central feature of Jenkins. Jenkins will build your project with any build system

Pipeline

Suitable for building pipelines or organizing complex activities that do not easily fit in free style

Multi-configuration Project

Suitable for projects that need large number of different configurations

Folder

Creates a container and stores nested items in it. Useful for grouping things together.

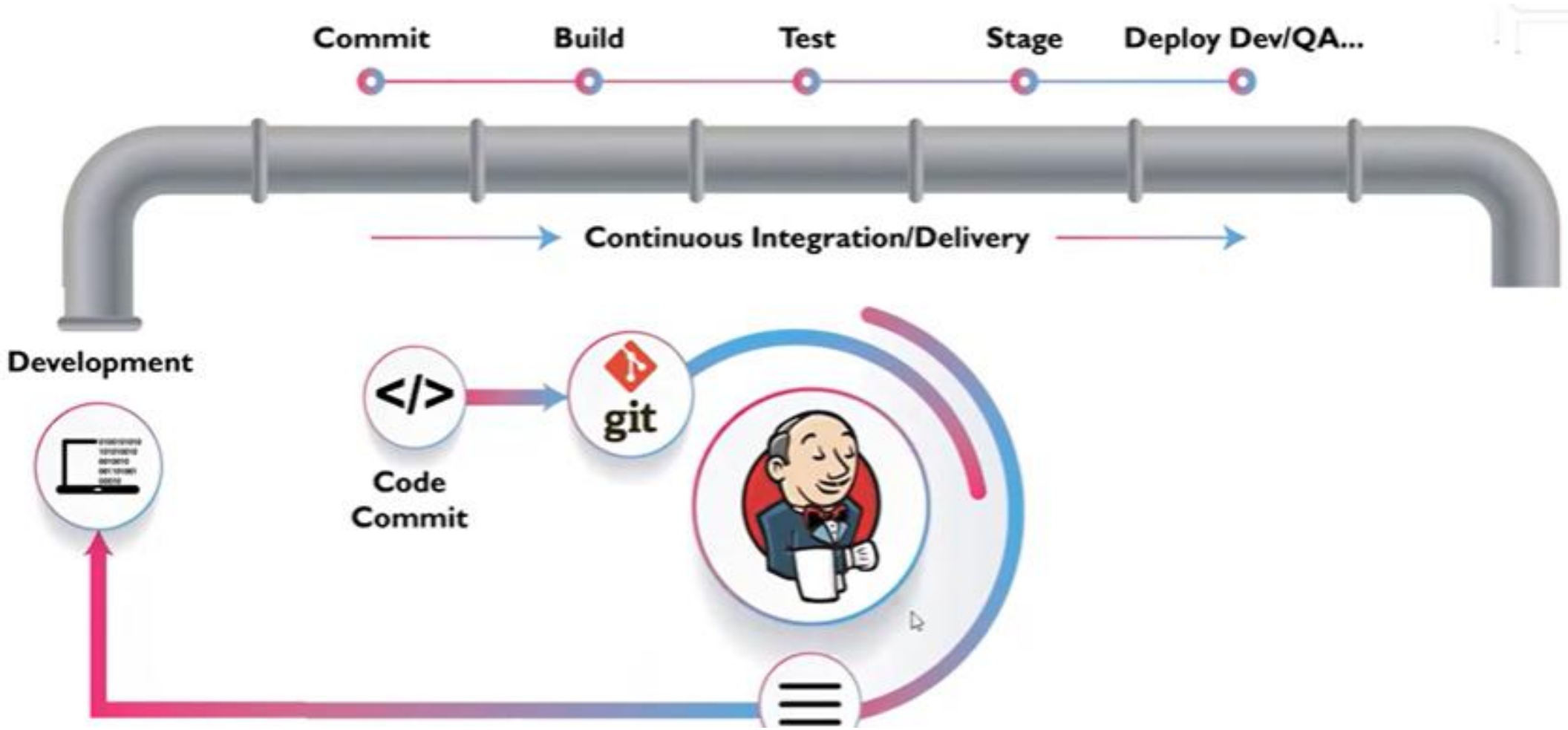
Organization

Scans a Github Organization for all matching repositories..

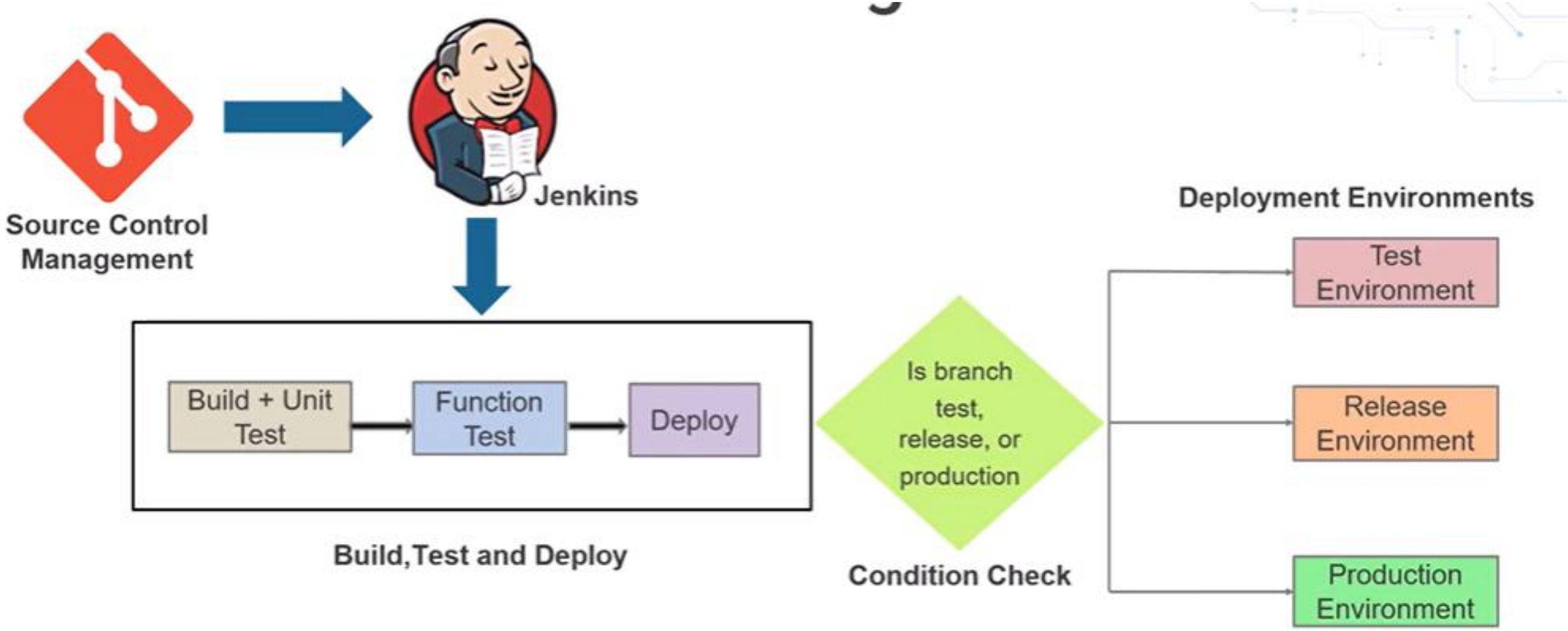
Multibranch Pipeline

Creates a set of pipeline projects according to detected branches in one SCM repositories.

Rôle de Jenkins dans devops



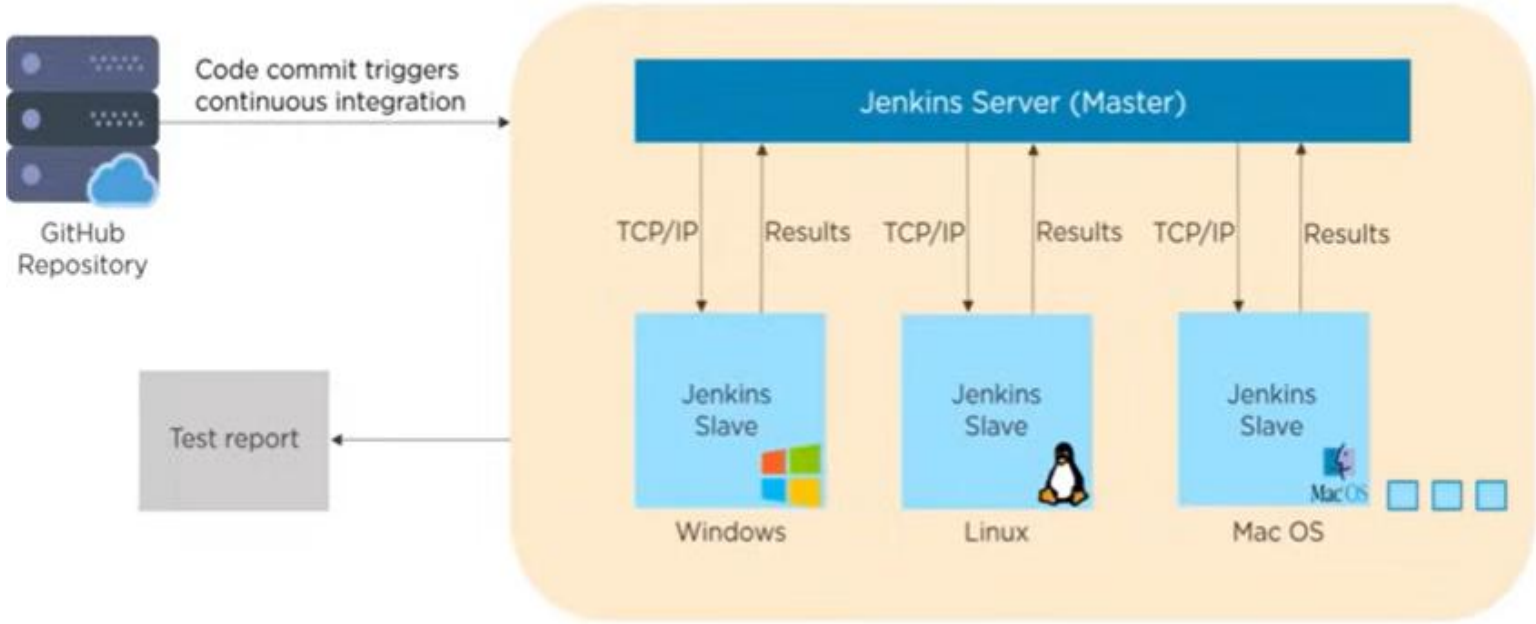
Processus de Jenkins

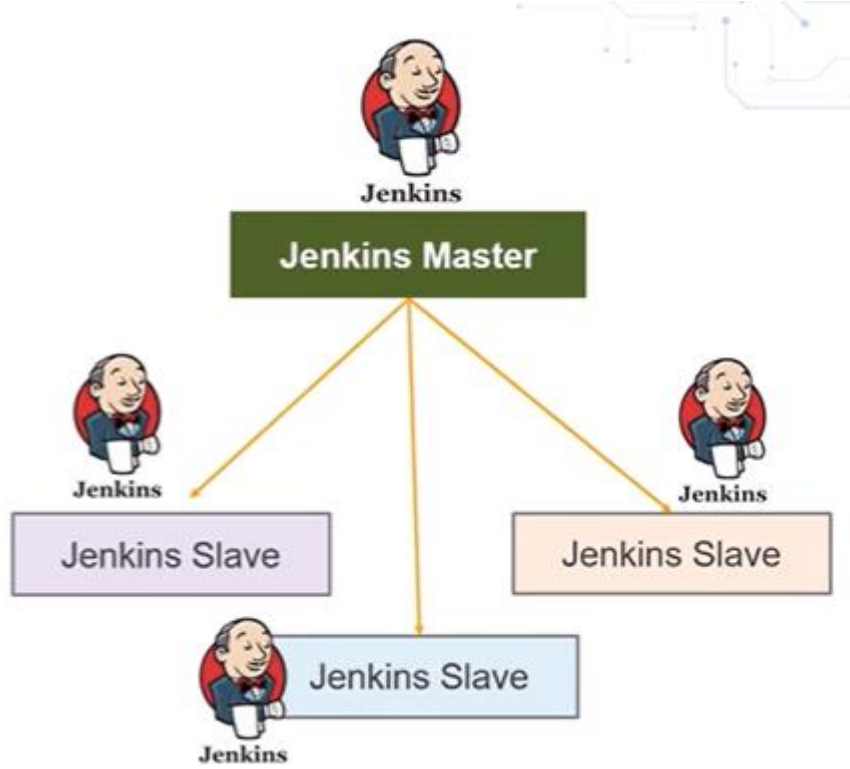


Architecture de Jenkins

- L'architecture de Jenkins est basée sur un modèle maître-esclave (master-slave).

- Le maître Jenkins est le composant central qui orchestre et coordonne les tâches, tandis que les agents Jenkins (esclaves) exécutent les tâches de construction, de test et de déploiement sur différentes machines.





Le maître et les esclaves en Jenkins

- **Jenkins Master:**
- Le serveur principal qui gère la configuration, la planification et la surveillance des tâches. Il est responsable de la coordination de l'ensemble du processus CI/CD.

- **Jenkins Agents (Esclaves):**
- Des machines distantes qui exécutent les tâches assignées par le maître. Ils peuvent être configurés pour exécuter des tâches spécifiques sur différents systèmes d'exploitation et environnements.

Le rôle des esclaves

•Exécution des tâches :

•Les agents sont les machines sur lesquelles les jobs Jenkins sont réellement exécutés, comme la compilation de code, l'exécution de tests automatisés et le déploiement de nouvelles versions.

•Répartition de la charge :

•Ils permettent de répartir la charge de travail, évitant ainsi que le nœud maître ne soit surchargé. C'est crucial pour les systèmes avec beaucoup d'automatisation et de tâches simultanées.

•Support de configurations multiples :

•Les agents peuvent être configurés sur différentes machines avec des systèmes d'exploitation variés (Windows, Linux, Docker, Kubernetes) pour répondre aux besoins spécifiques de chaque projet ou de chaque étape du processus CI/CD.

•Utilisation optimisée :

•Il est possible de configurer les agents pour qu'ils se lancent uniquement en cas de besoin et se mettent hors ligne lorsqu'ils ne sont pas utilisés, ce qui permet d'économiser les ressources système.

Les types des esclaves dans Jenkins

Esclaves statiques (permanents)

- **Configuration manuelle:**

- Ces agents sont configurés manuellement via des méthodes telles que SSH ou JNLP et sont connectés en permanence au nœud maître de Jenkins.

- **Utilisation courante:**

- Ils sont adaptés pour gérer une charge de travail constante et continue, où des ressources dédiées sont nécessaires.

- **Exemple:** Une machine virtuelle configurée spécifiquement pour exécuter les tâches de build de Jenkins et toujours accessible.

Les types des esclaves dans Jenkins

Esclaves dynamiques:

- **Provisionnement à la demande:**
- Ces agents sont créés et supprimés automatiquement par Jenkins, souvent en réponse à une demande de build ou à des besoins spécifiques de capacité.

- **Utilisation courante:**
- Ils sont idéaux pour gérer des charges de travail variables ou pour utiliser des ressources partagées dans des environnements tels que Kubernetes ou des conteneurs.

- **Exemple:** Un conteneur Docker créé et supprimé sur un cluster Kubernetes lorsqu'un build nécessite plus de ressources ou lorsqu'il est terminé.

Dynamic Slaves

Slaves that are created on-demand and automatically provisioned.

3. Installation de Jenkins

Installer Jenkins sous WSL

- Ouvrez WSL et tapez les commandes suivantes :

```
root@DESKTOP-SKOOCDK:~# apt-get update
```

```
root@DESKTOP-SKOOCDK:~# apt-get install openjdk-21-jdk
```

```
root@DESKTOP-SKOOCDK:~# java -version
openjdk version "21.0.8" 2025-07-15
OpenJDK Runtime Environment (build 21.0.8+9-Ubuntu-0ubuntu124.04.1)
OpenJDK 64-Bit Server VM (build 21.0.8+9-Ubuntu-0ubuntu124.04.1, mixed mode, sharing)
```

- Allez sur : https://www.jenkins.io/doc/book/installing/linux/#debianubuntu

Et exécutez les 4 commandes suivantes (ne pas oublier de supprimer les slachs « \ »)

```
sudo wget -O /etc/apt/keyrings/jenkins-keyring.asc \
  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
echo "deb [signed-by=/etc/apt/keyrings/jenkins-keyring.asc]" \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update
sudo apt-get install jenkins
```

Installer Jenkins sous WSL

- Version installée de Jenkins

```
Setting up jenkins (2.516.1) ...
```

- Démarrer le serveur de Jenkins

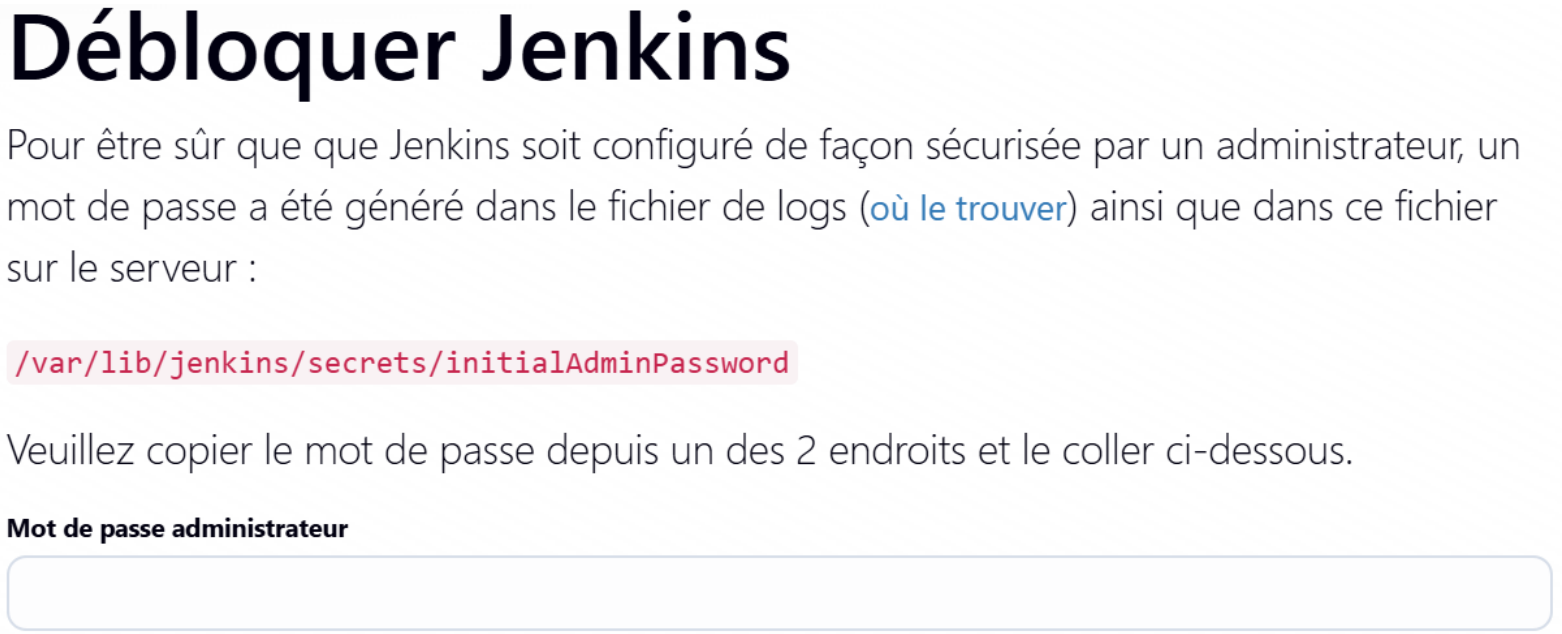
```
root@DESKTOP-SKOOCDK:~# systemctl start jenkins
```

- Vérifier le statut du serveur

```
root@DESKTOP-SKOOCDK:~# systemctl status jenkins
● jenkins.service - Jenkins Continuous Integration Server
     Loaded: loaded (/usr/lib/systemd/system/jenkins.service; enabled; preset: enabled)
     Active: active (running) since Tue 2025-07-29 14:55:14 CEST; 2min 26s ago
   Main PID: 42040 (java)
      Tasks: 53 (limit: 9299)
     Memory: 535.2M (peak: 674.4M)
        CPU: 22.937s
     CGroup: /system.slice/jenkins.service
             └─42040 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cac>

Jul 29 14:55:11 DESKTOP-SKOOCDK jenkins[42040]: 5e2a2f500a4142698da28af2747dd213
Jul 29 14:55:11 DESKTOP-SKOOCDK jenkins[42040]: This may also be found at: /var/lib/jenkins/secrets/initialAdmi>
Jul 29 14:55:11 DESKTOP-SKOOCDK jenkins[42040]: ************************************************************
Jul 29 14:55:11 DESKTOP-SKOOCDK jenkins[42040]: ************************************************************
Jul 29 14:55:11 DESKTOP-SKOOCDK jenkins[42040]: ************************************************************
Jul 29 14:55:14 DESKTOP-SKOOCDK jenkins[42040]: 2025-07-29 12:55:14.471+0000 [id=50]        INFO        jenkins>
Jul 29 14:55:14 DESKTOP-SKOOCDK jenkins[42040]: 2025-07-29 12:55:14.511+0000 [id=40]        INFO        hudson.>
Jul 29 14:55:14 DESKTOP-SKOOCDK systemd[1]: Started jenkins.service - Jenkins Continuous Integration Server.
Jul 29 14:55:15 DESKTOP-SKOOCDK jenkins[42040]: 2025-07-29 12:55:15.572+0000 [id=88]        INFO        h.m.Dow>
Jul 29 14:55:15 DESKTOP-SKOOCDK jenkins[42040]: 2025-07-29 12:55:15.573+0000 [id=88]        INFO        hudson.>
lines 1-20/20 (END)
```

Installer Jenkins sous WSL

- Allez sur : http://localhost:8080/

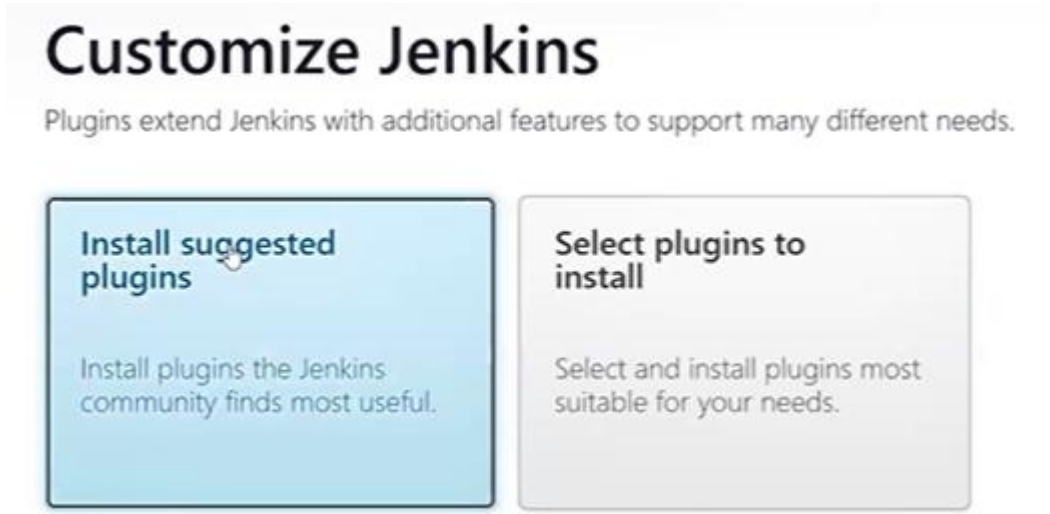


- Extraire le mot de passe depuis le chemin sélectionné en rouge et le coller dans « Mot de passe administrateur » :

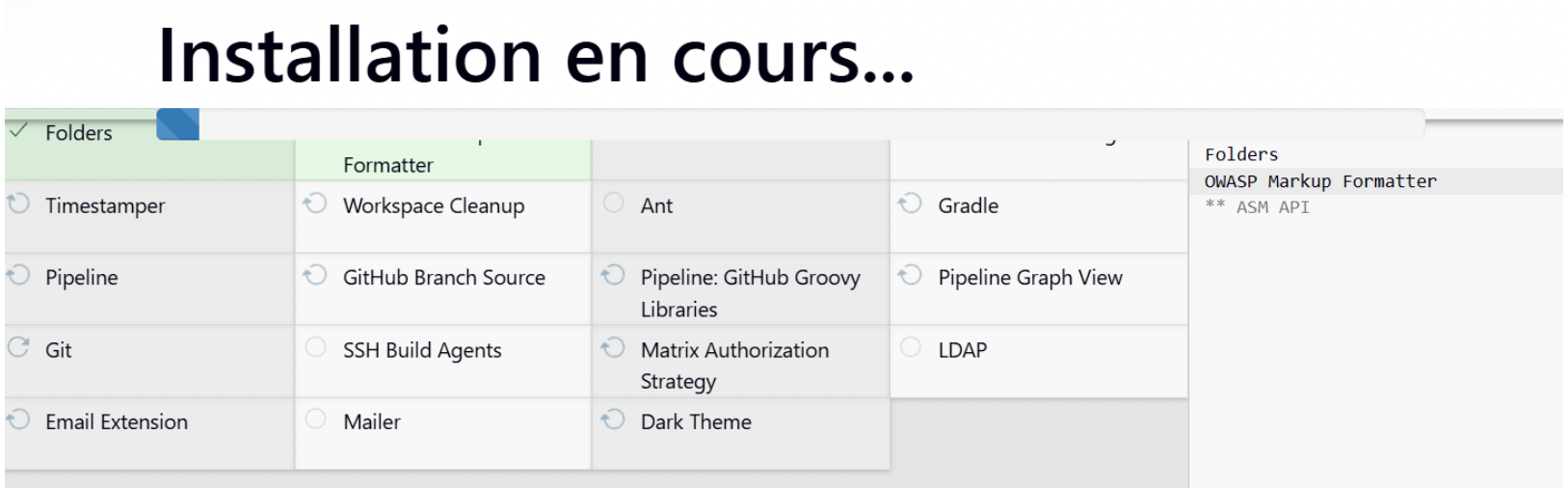
```
root@DESKTOP-SKOOCDK:~# cat /var/lib/jenkins/secrets/initialAdminPassword
5e2a2f500a4142698da28af2747dd213
root@DESKTOP-SKOOCDK:~#
```

Installer Jenkins sous WSL

- Cliquer sur Install suggested plugins



- L'installation des plugins proposée prend du temps….



Installer Jenkins sous WSL

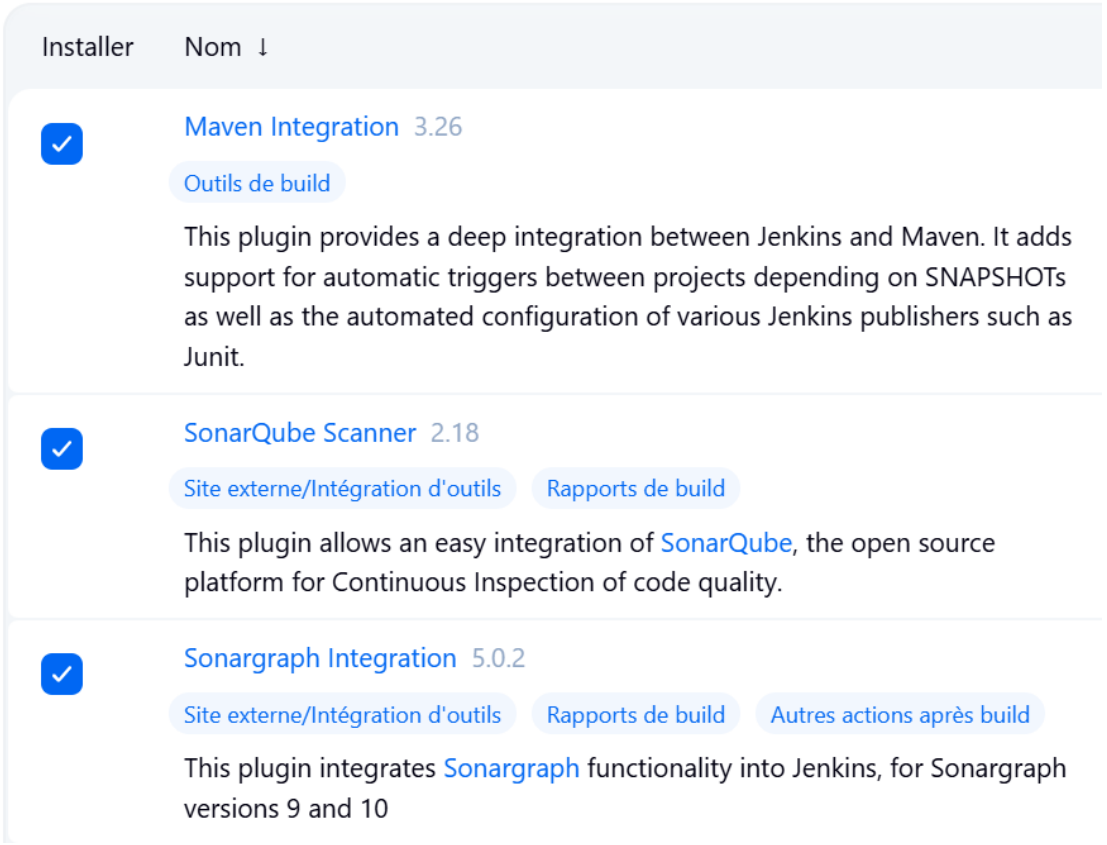
- Compléter le formulaire

- Sauver et terminer l'URL

Bienvenue sur Jenkins

Installer quelques plugins

- Afin de créer une chaîne d'intégration continue, nous allons installer les plugins suivants dans Jenkins (installez ces plugins sans redémarrer, puis redémarrez à la fin)
  - GIT plugin (normalement déjà installé mais vérifiez)
  - Maven integration
  - SonarGraph integration
  - SonarQube scanner



Installer quelques plugins

- Après l'installation des plugins, il faut redémarrer le serveur Jenkins

```
root@DESKTOP-SKOOCDK:~# systemctl restart jenkins
```

Configuration de Jenkins - JDK

- Après installation de jdk sous WSL, il faut modifier la variable d'environnement JAVA_HOME comme suit :

```
root@DESKTOP-SKOOCDK:/usr/lib/jvm# export JAVA_HOME="/usr/lib/jvm/java-21-openjdk-amd64/"
root@DESKTOP-SKOOCDK:/usr/lib/jvm# export PATH="$JAVA_HOME/bin:$PATH"
root@DESKTOP-SKOOCDK:/usr/lib/jvm# systemctl restart jenkins
```

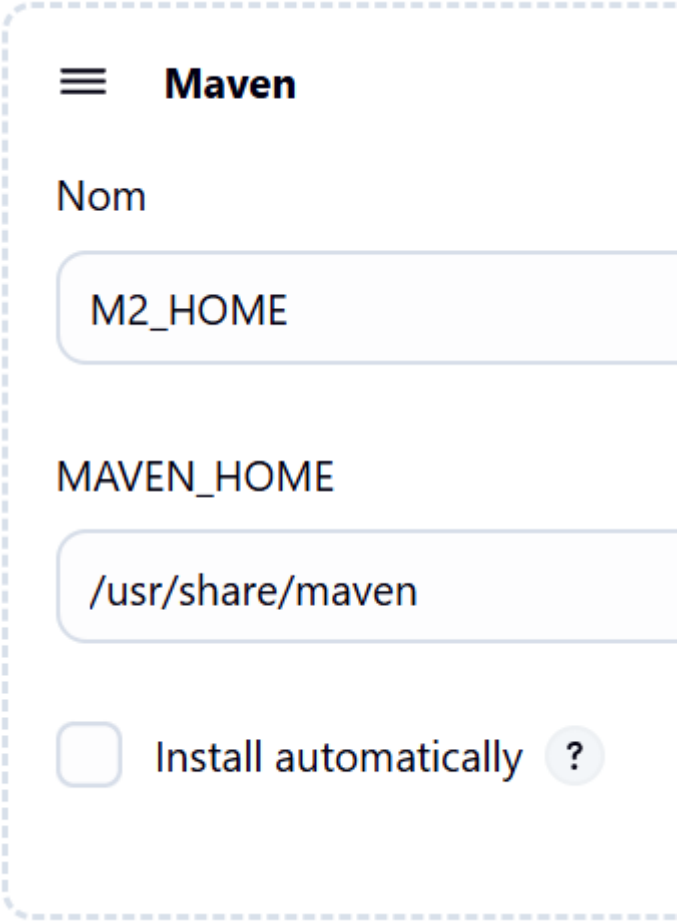
- Puis sous Jenkins, il faut configurer Le JDK sous Tools :

Configuration de Jenkins - Maven

- Rajoutez le chemin d'installation de Maven sous wsl

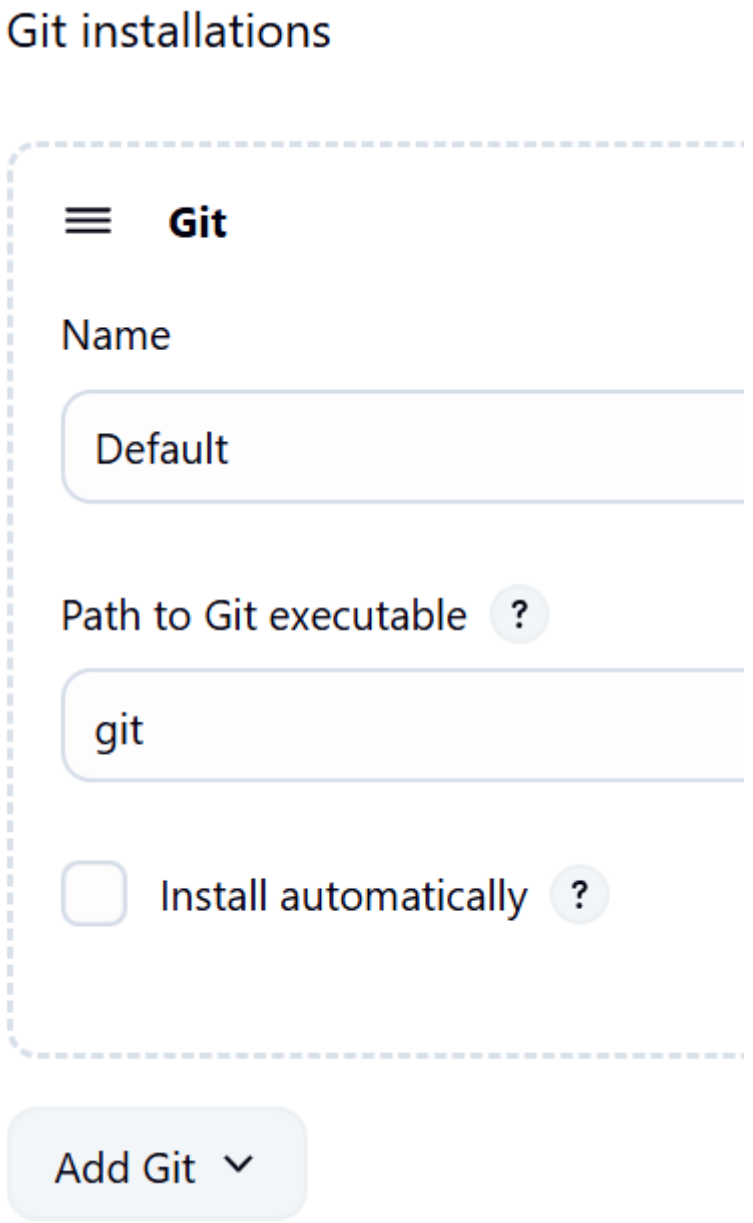
```
root@DESKTOP-SKOOCDK:/# export M2_HOME="/usr/share/maven"
root@DESKTOP-SKOOCDK:/# export PATH="$M2_HOME/bin:$PATH"
```

```
root@DESKTOP-SKOOCDK:/# systemctl restart jenkins
```



Configuration de Jenkins - GIT

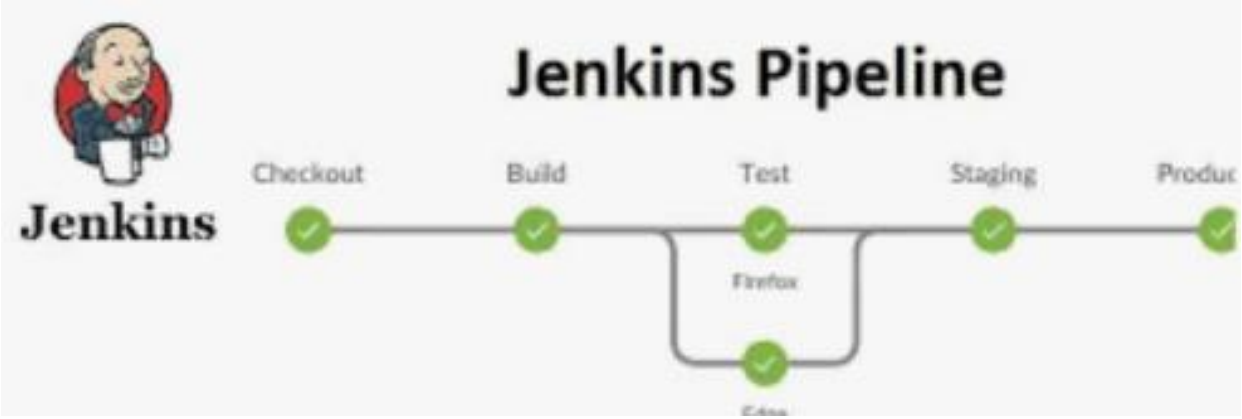
- Le GIT est déjà configuré sous Jenkins



4. Gestion de jobs free-style sous Jenkins

Création de jobs sous Jenkins

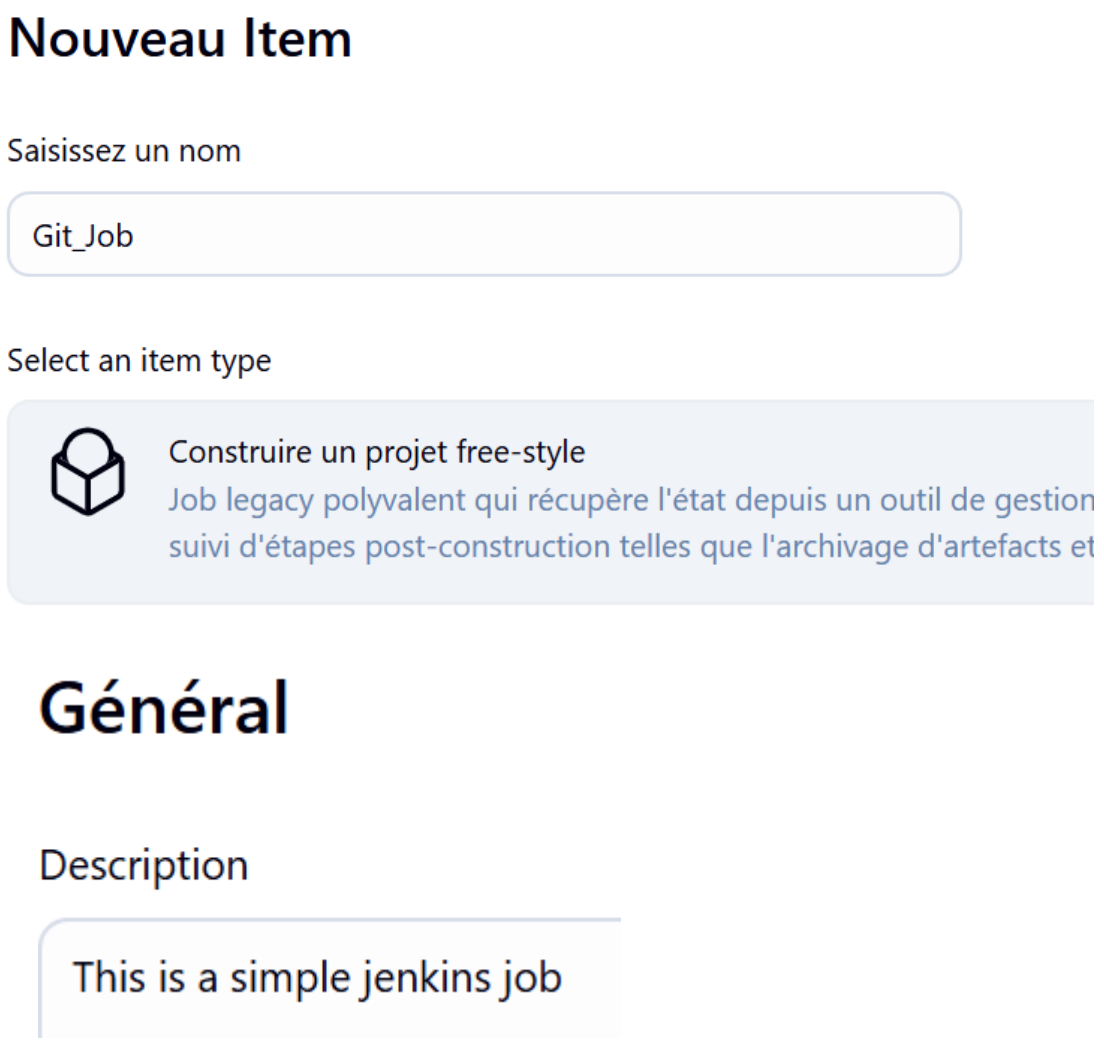
- Il y a deux façons de configurer la compilation d'un projet :
  - **Méthode formulaire (freestyle)** : Vous pouvez configurer le projet en utilisant un formulaire à remplir
  - **Méthode pipeline :** l'alternative est d'utiliser un **pipeline**, qui configure le projet à l'aide de scripts (en utilisant **Groovy**). Cette approche permet également de paralléliser les étapes du projet et offre une interface plus conviviale pour lire les logs.



Créer notre premier job free-style

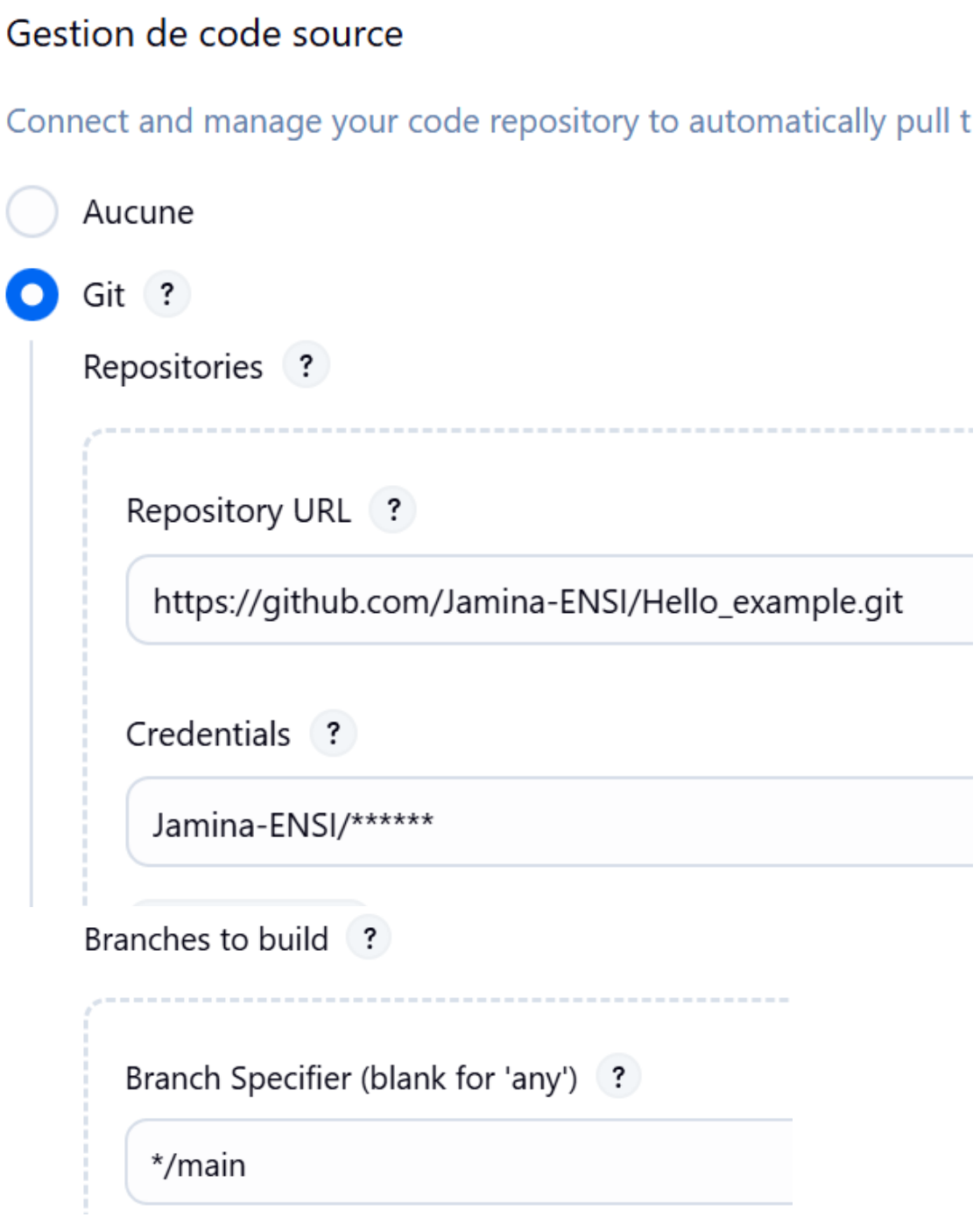
1. Créer un nouveau job de type free-style

2. Créer un nouveau job de type free-style



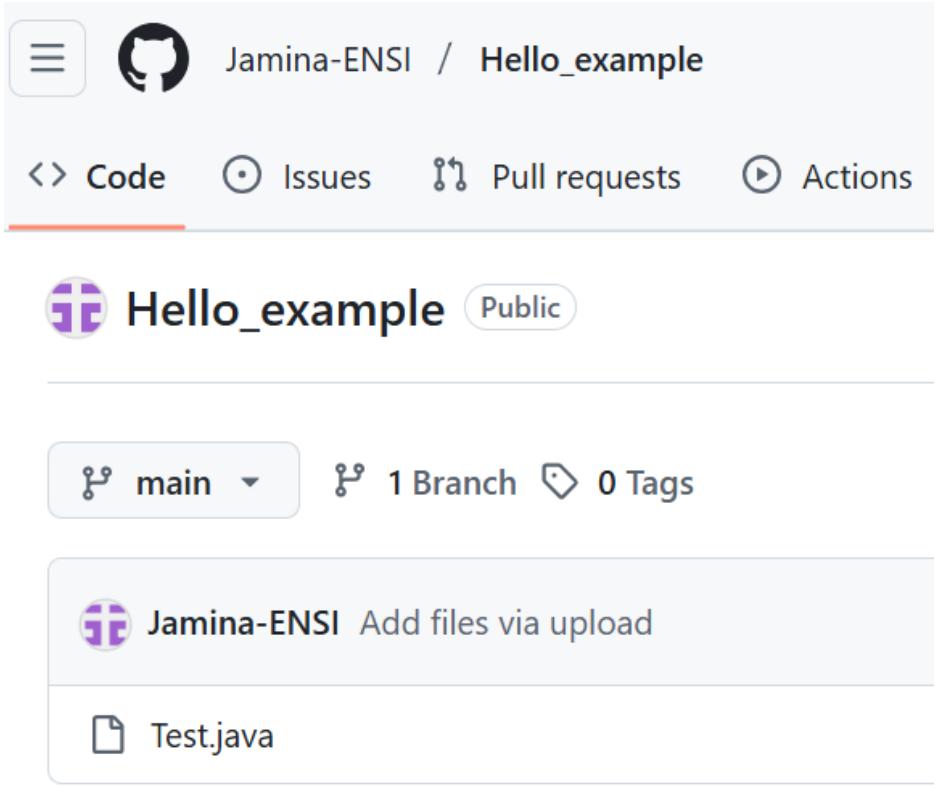
Créer notre premier job

3. Configurer les paramètres de GIT



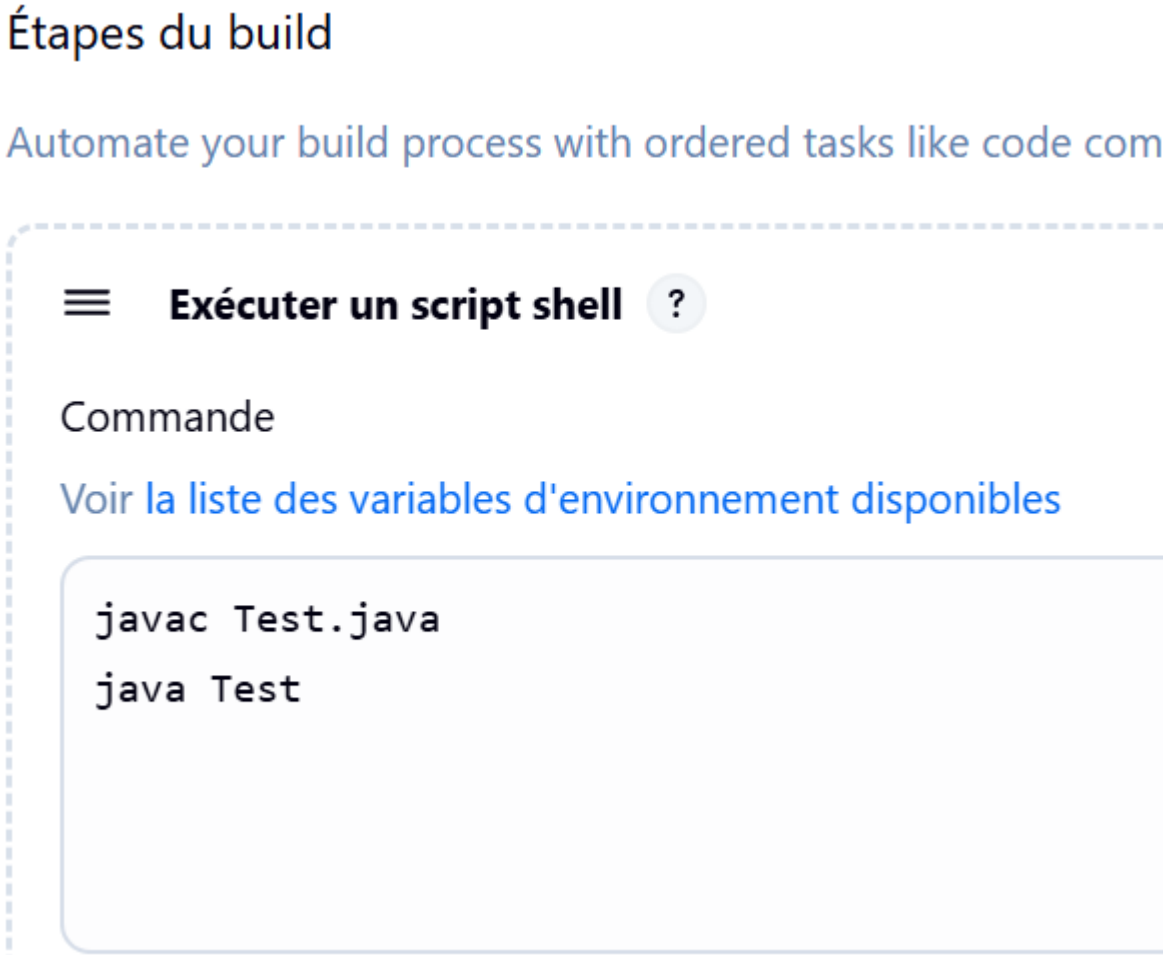
Créer notre premier job

- Le répository Hello_example contient un seul fichier Test.java :

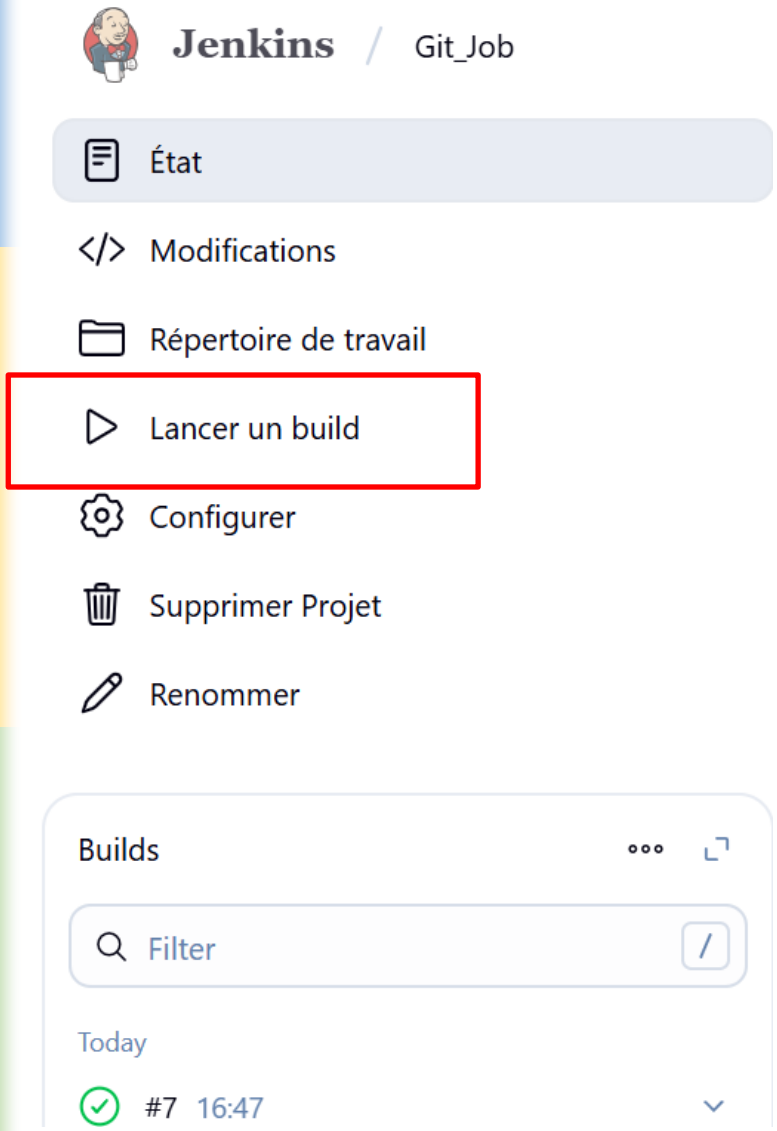


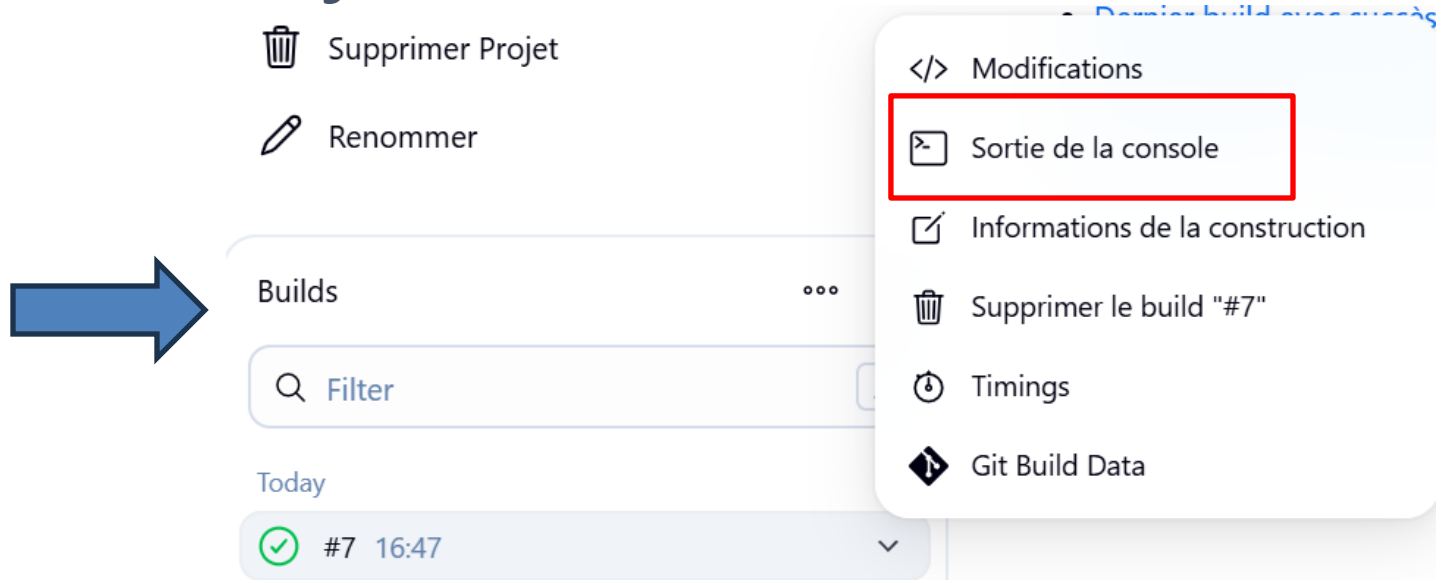
Créer notre premier job

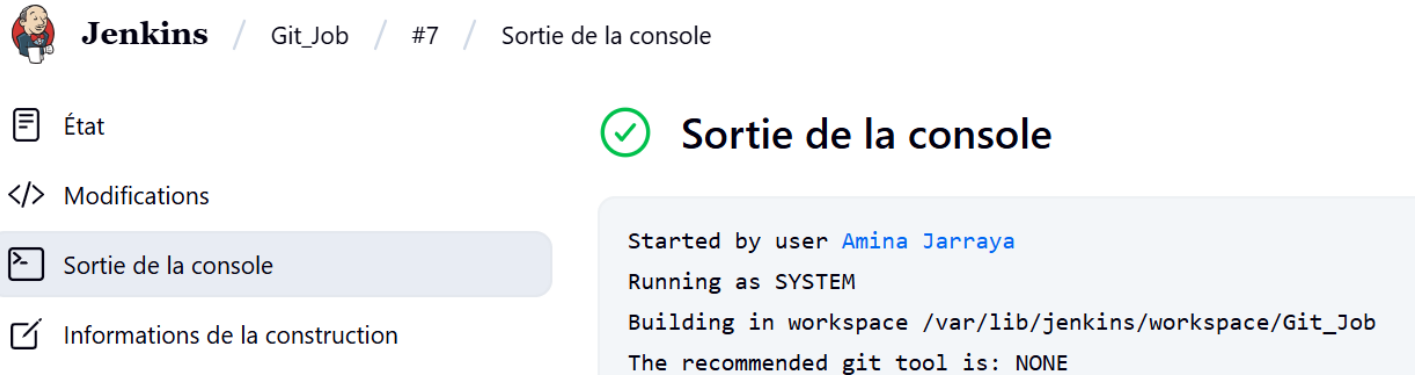
4. Configurer les étapes du build et appliquer



Exécuter notre premier job





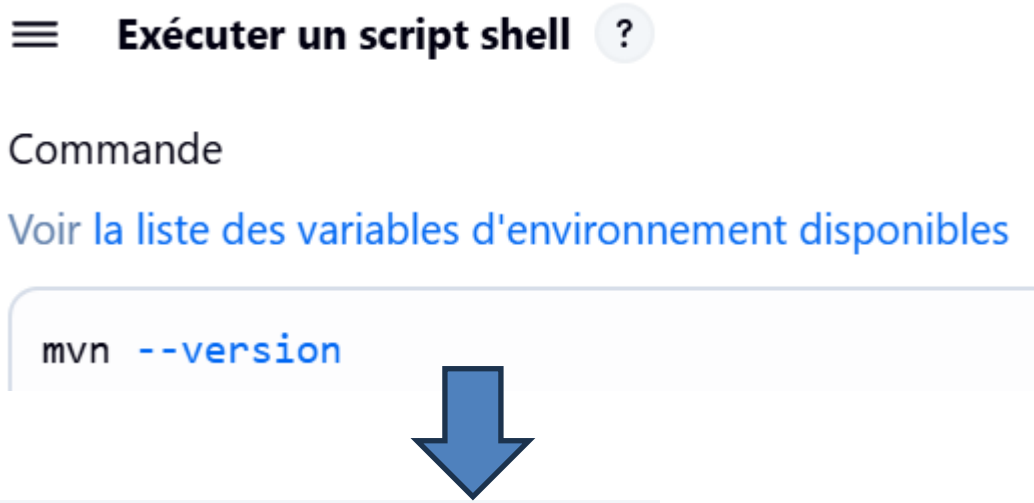


Analyser la sortie du job

```
Started by user Amina Jarraya
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/Git_Job
The recommended git tool is: NONE
using credential 1fb93111-3a24-405e-915f-ffeeb1f80b68
 > git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/Git_Job/.git # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url https://github.com/Jamina-ENSI/Hello_example.git # timeout=10
Fetching upstream changes from https://github.com/Jamina-ENSI/Hello_example.git
 > git --version # timeout=10
 > git --version # 'git version 2.43.0'
using GIT_ASKPASS to set credentials
 > git fetch --tags --force --progress -- https://github.com/Jamina-ENSI/Hello_example.git +refs/heads/*:refs/remotes/origin/* # timeout=10
 > git rev-parse refs/remotes/origin/main^{commit} # timeout=10
Checking out Revision 8cac4f30783c4e47b58f3ffe730466f332f74226 (refs/remotes/origin/main)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f 8cac4f30783c4e47b58f3ffe730466f332f74226 # timeout=10
Commit message: "Add files via upload"
 > git rev-list --no-walk 8cac4f30783c4e47b58f3ffe730466f332f74226 # timeout=10
[Git_Job] $ /bin/sh -xe /tmp/jenkins6571700839735225603.sh
+ javac Test.java
+ java Test
Hello
Finished: SUCCESS
```

Rajouter un autre script

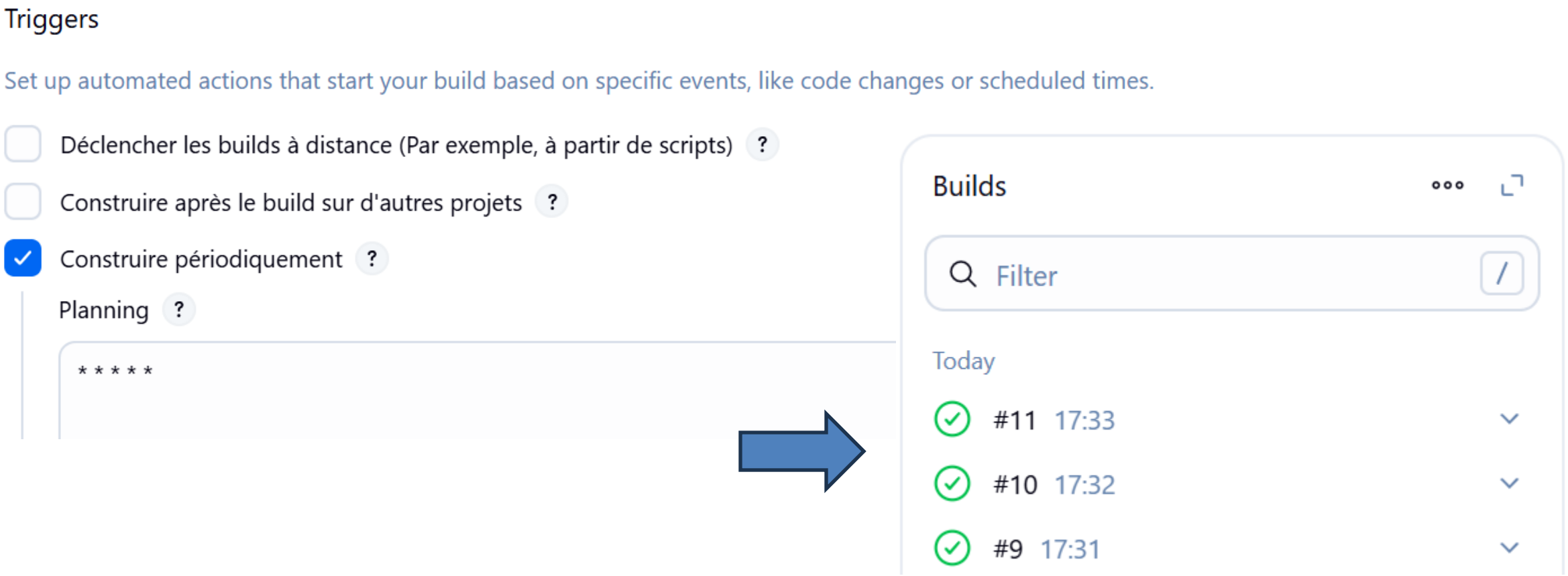
- Vous pouvez rajoutez d'autres scripts dans votre job :




```
+ javac Test.java
+ java Test
Hello
[Git_Job] $ /bin/sh -xe /tmp/jenkins7850525868335514129.sh
+ mvn --version
[1mApache Maven 3.8.7[m
Maven home: /usr/share/maven
Java version: 21.0.8, vendor: Ubuntu, runtime: /usr/lib/jvm/java-21-openjdk-amd64
Default locale: en, platform encoding: UTF-8
OS name: "linux", version: "6.6.87.2-microsoft-standard-wsl2", arch: "amd64", family: "unix"
Finished: SUCCESS
```

La construction périodique

- Sélectionnez l'action pour déclencher périodiquement le flux d'intégration continue.
- Cela peut être un procesus qui s'exécute régulièrement, par exemple chaque minute, ou qui vérifie si une nouvelle version a été poussée sur GIT



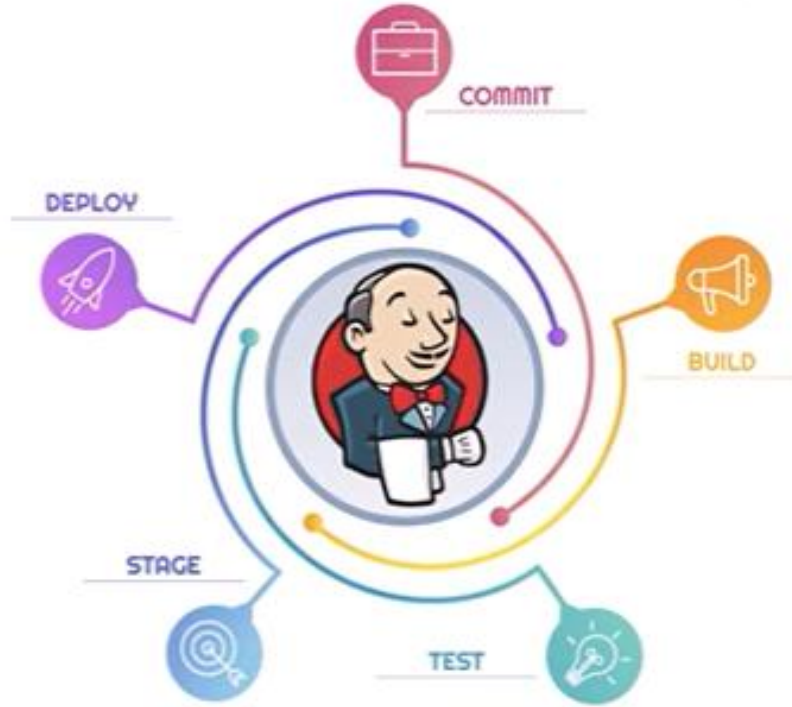
5. Gestion de pipelines sous Jenkins

C’est quoi un delivery pipeline ?

- Un **pipeline Jenkins**, dans le contexte de l'intégration continue et de la livraison continue (CI/CD), est une série d'étapes automatisées qui permettent de construire, tester et déployer un logiciel de manière efficace et répétable.

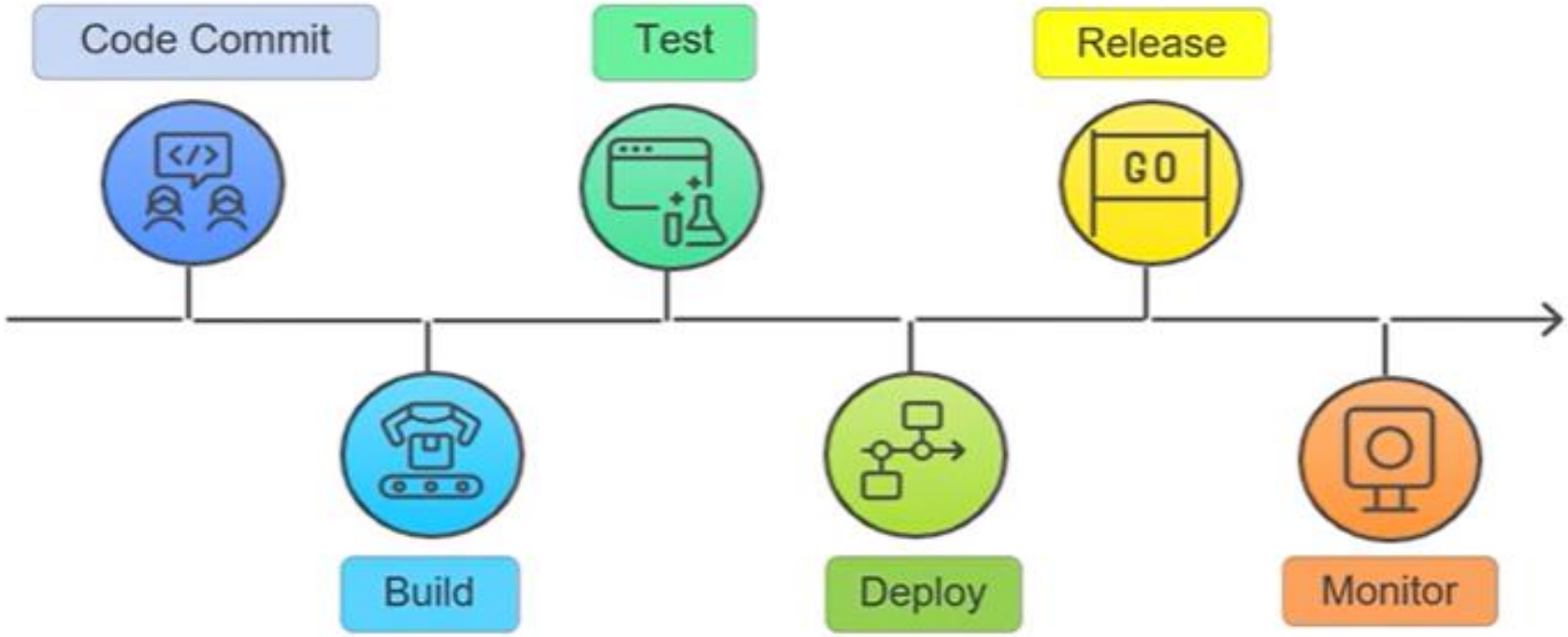
- Il s'agit d'un ensemble de tâches séquentielles ou parallèles, définies dans un script (généralement **Groovy**) et exécutées par Jenkins pour automatiser le processus de développement logiciel.

- En d'autres termes, un pipeline Jenkins est une représentation de votre workflow de développement, de la création du code à sa mise en production, avec des étapes clairement définies et automatisées.

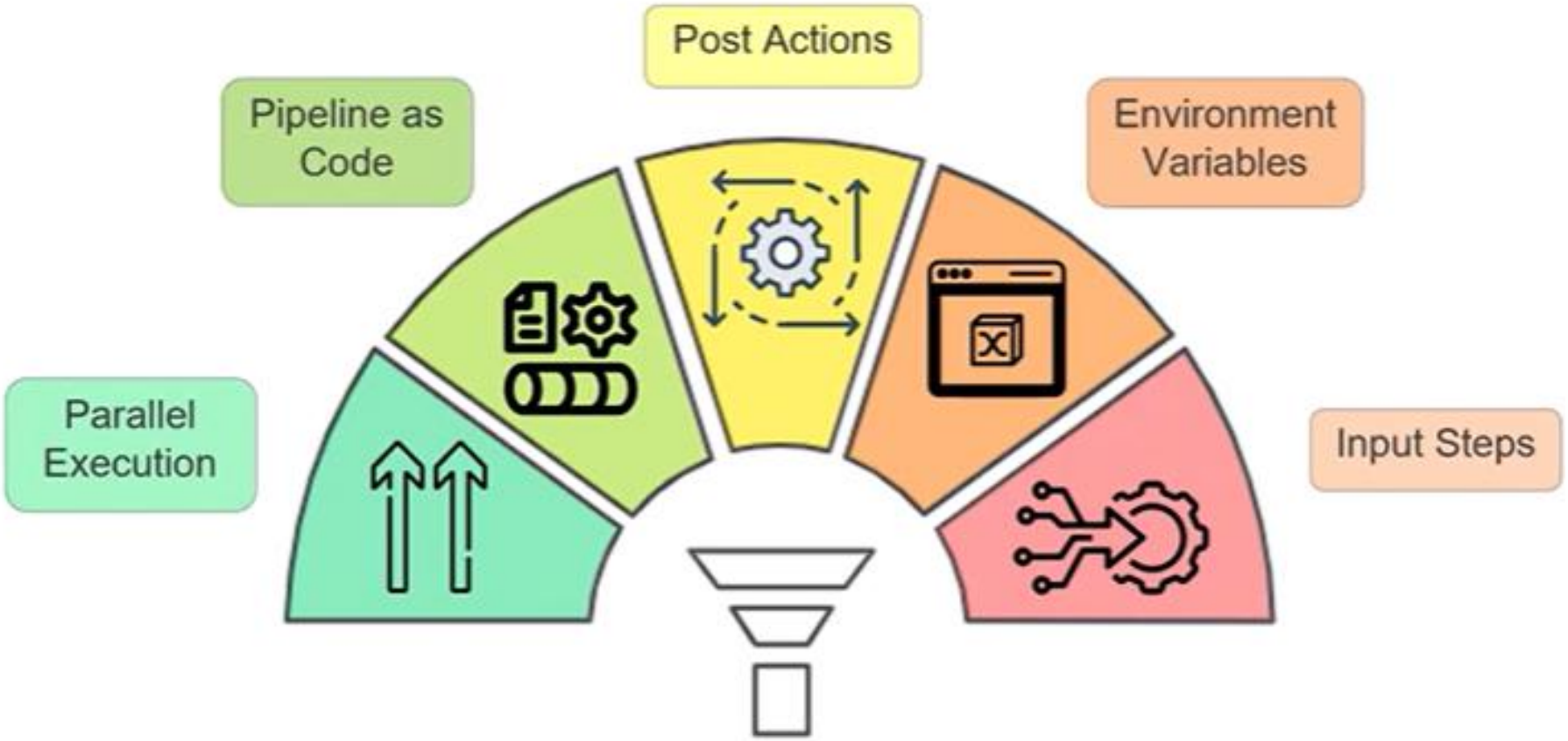


Le pipeline de la livraison continue

- Un ***pipeline de livraison continue*** est une série de processus automatisés destinés à livrer de nouveaux logiciels. Il s'agit d'une implémentation du paradigme « Continuous Everything », où les builds, les tests et les déploiements automatisés sont orchestrés pour former un workflow de livraison unique



Jenkins pipeline features



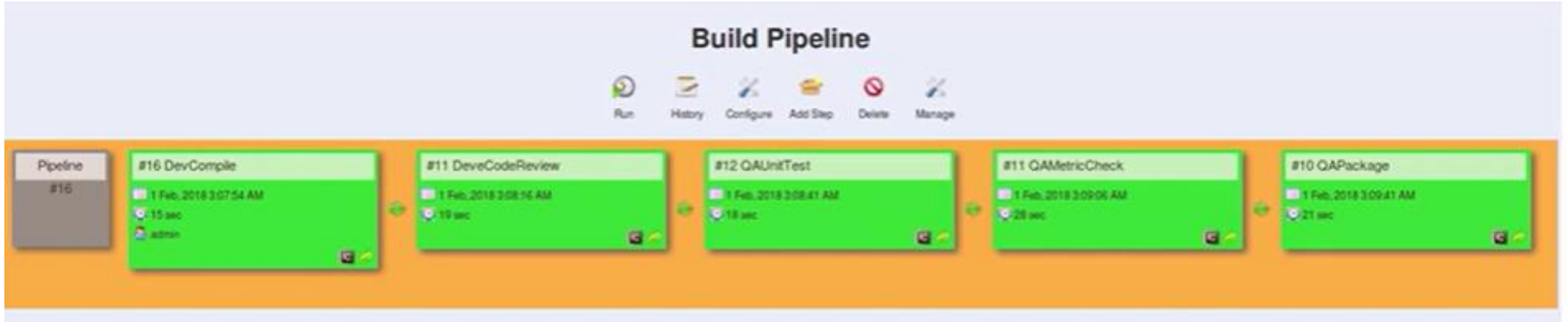
Une vue pipeline

Yellow blocks represent the currently executing job

Blue blocks represent the jobs waiting for execution

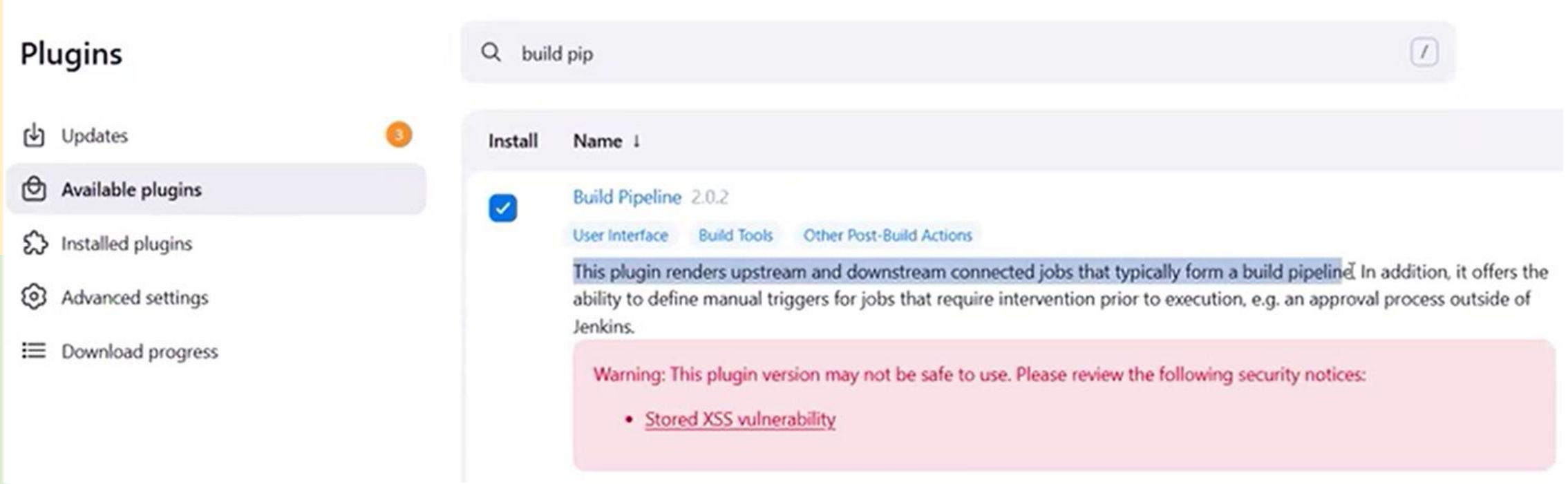
Green blocks represent the successful build of job

Red blocks represent failure while building a job



Installer plugin – build pipeline

- Installer le plugin build pipeline

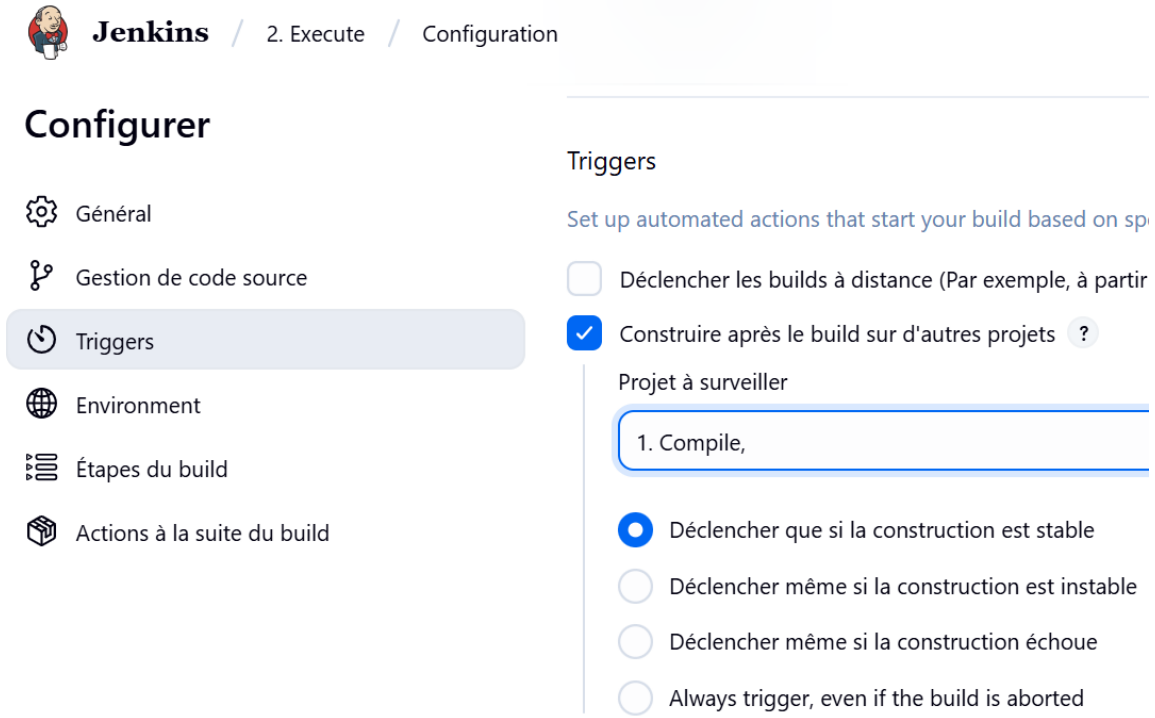


Connecter deux jobs sous Jenkins

- Créez deux jobs : « 1.Compile » et « 2. Execute ». Le premier exécute le shell « echo « compiler le code» et le 2ème exécute le shell «echo « executer le code »

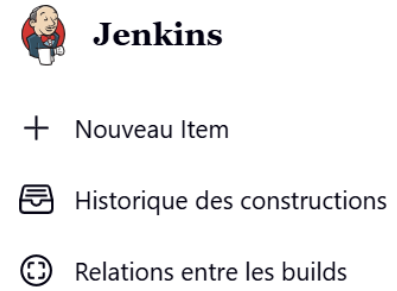
S	M	Nom du projet ↓	Dernier succès	Dernier échec	Dernière durée
		1. Compile	s. o.	s. o.	ND
		2. Execute	s. o.	s. o.	ND

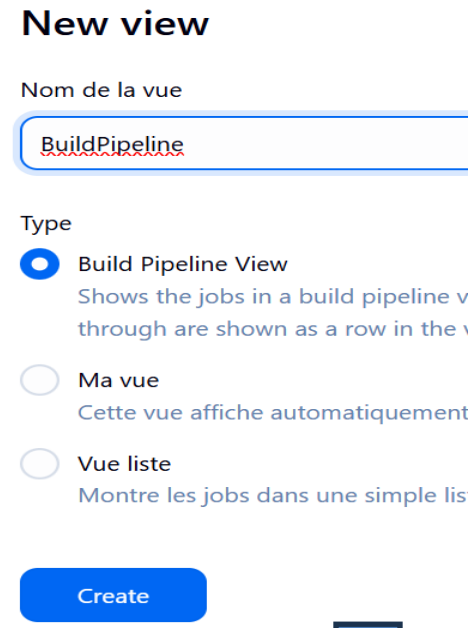
- Pour le job « 2. Execute », il faut cocher l'option « Construire après le build sur d'autres projets » en spécifiant le 1er job.

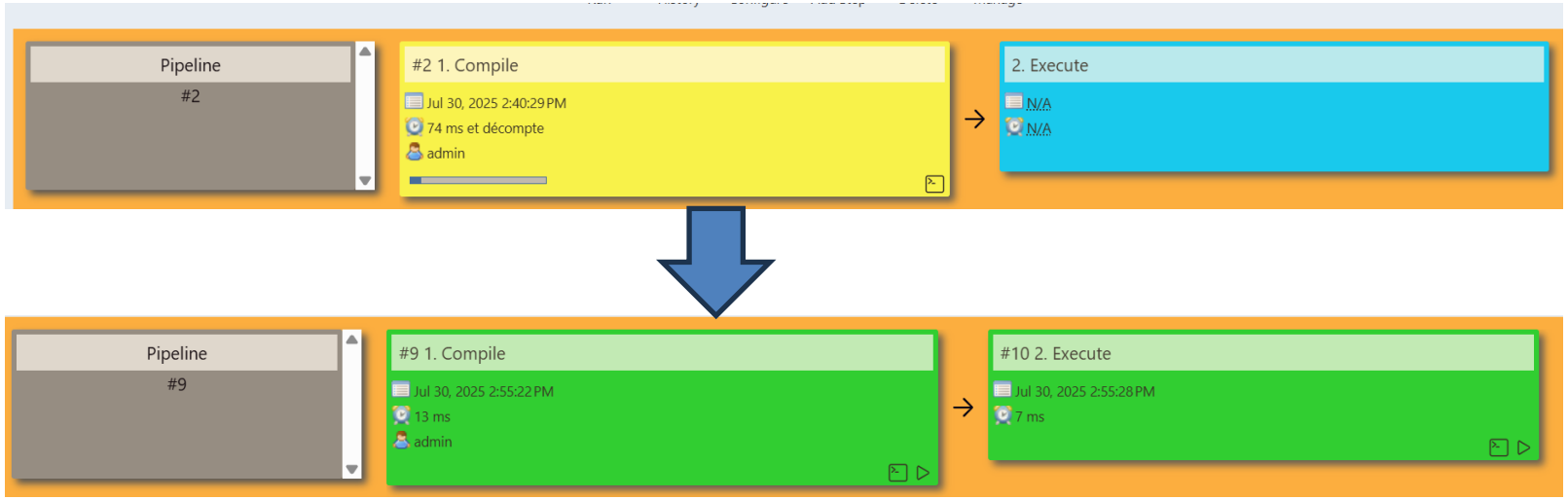


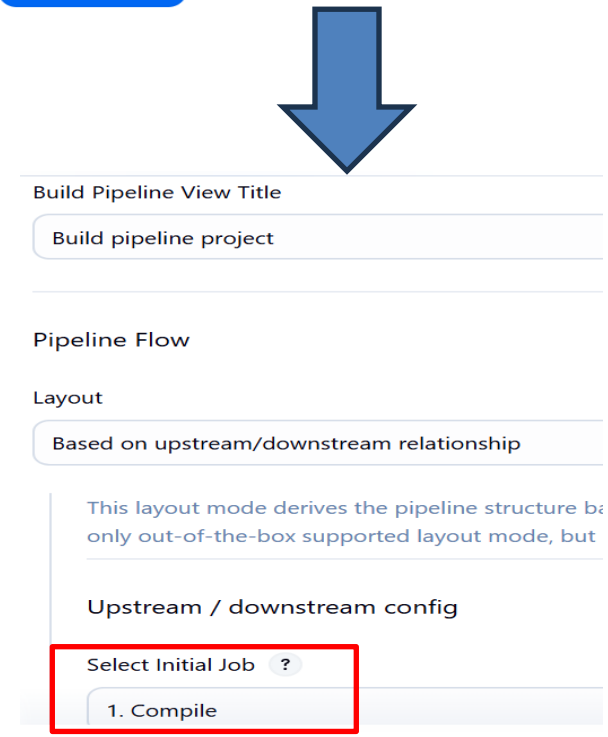
Création d'une vue pipeline

- Créez une vue pipeline, sélectionner le job initial puis exécutez :



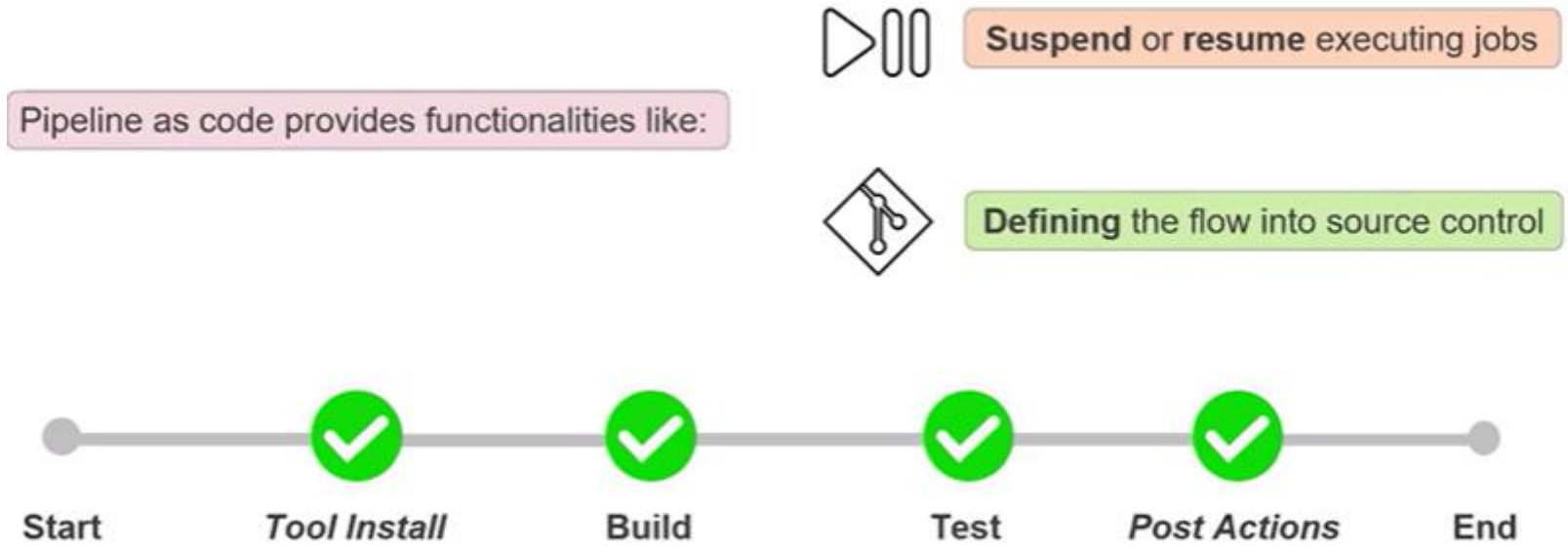






Pipeline as code sous Jenkins

- En Jenkins, "Pipeline as Code" (Pipeline en tant que code) signifie définir votre processus de livraison continue (CI/CD) dans un fichier texte, appelé **Jenkinsfile**, qui est ensuite stocké et versionné avec votre code source.
  - Cela permet de traiter le pipeline comme n'importe quel autre code, le rendant ainsi plus facile à gérer, partager et réviser.
  - **Jenkinsfile**: C'est un fichier texte qui contient la logique de votre pipeline, généralement écrit dans le langage **Groovy** avec une syntaxe spécifique à Jenkins.

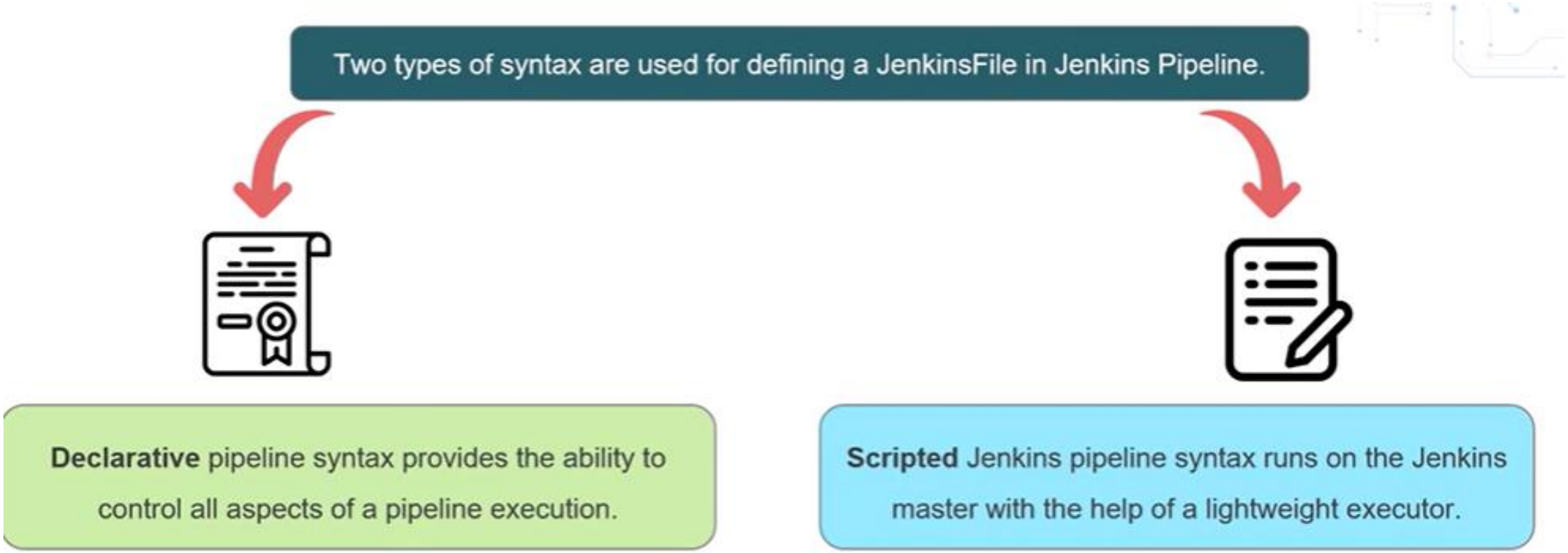


Types de pipeline sous Jenkins

- Jenkins offre deux types de pipelines pour orchestrer des workflows d'intégration continue et de livraison continue (CI/CD) : les pipelines déclaratifs et les pipelines scriptés.

➢ **Les pipelines déclaratifs** offrent une syntaxe plus simple et plus lisible

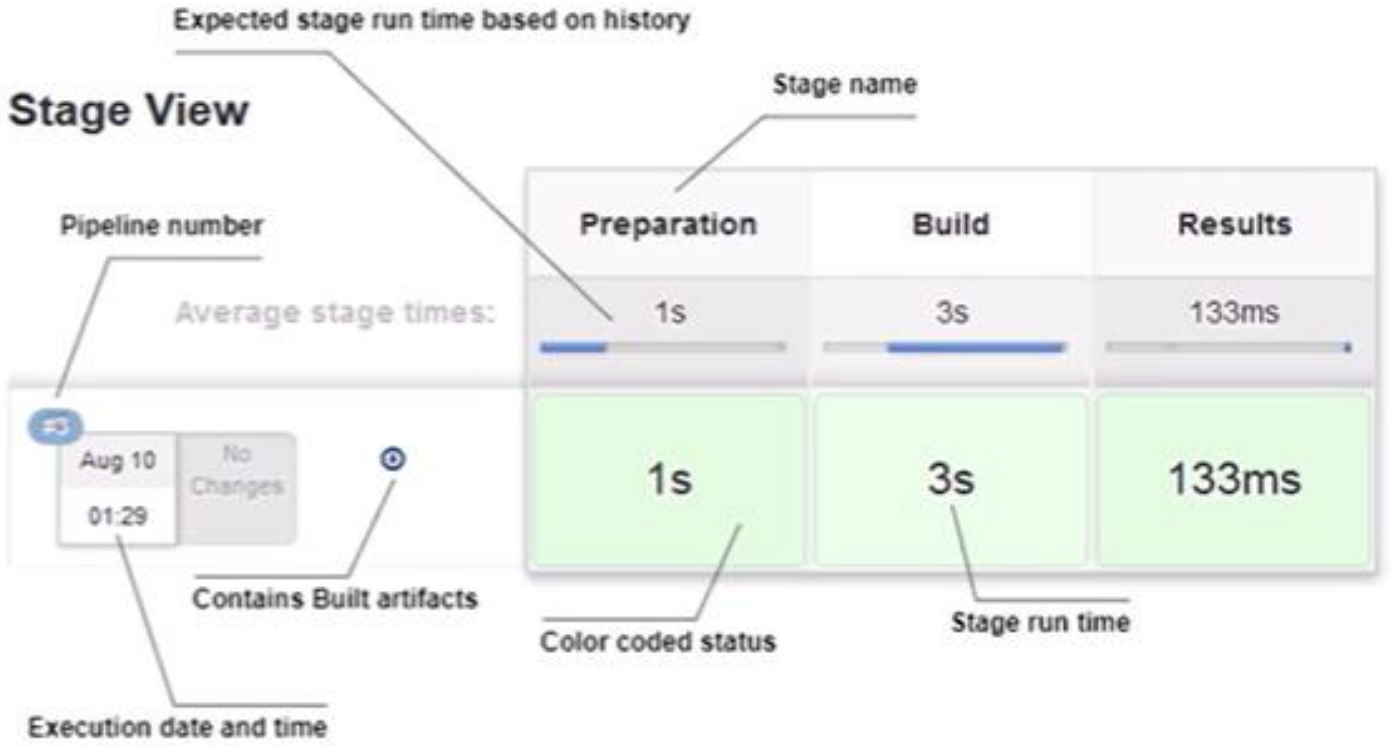
➢ **Les pipelines scriptés** offrent plus de flexibilité grâce à l'utilisation du langage Groovy.



Jenkins pipeline Stage view

Jenkins Stage View helps to visualize the **progress** of various stages of the Pipeline in **real time** .

The **Stage View** page will look something like the following screenshot:



Le pipeline déclarative

- **Syntaxe simplifiée:** Les pipelines déclaratifs utilisent une syntaxe plus concise et structurée, ce qui les rend plus faciles à lire et **à maintenir.**
- **Structure définie:** Ils suivent une structure prédéfinie avec des blocs tels que pipeline, agent, stages, steps.
- **Facilité d'utilisation:** Ils sont généralement plus conviviaux pour les utilisateurs qui ne sont pas familiers avec Groovy.
- **Limitations:** Moins flexibles pour des besoins de personnalisation complexes.

Jenkinsfile (Declarative Pipeline)

```
pipeline {
    agent any ❶
    stages {
        stage('Build') { ❷
            steps {
                // ❸
            }
        }
        stage('Test') { ❹
            steps {
                // ❺
            }
        }
        stage('Deploy') { ❻
            steps {
                // ❼
            }
        }
    }
}
```

Le pipeline scripté

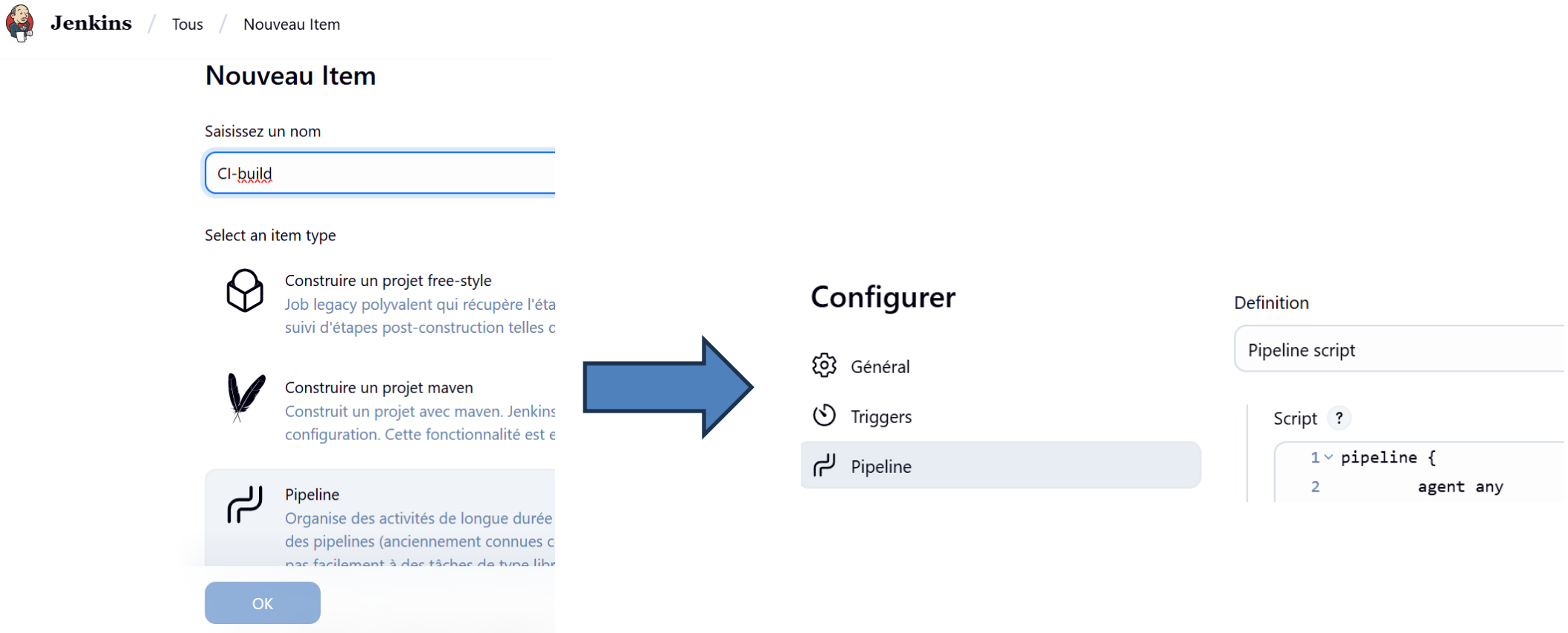
- **Flexibilité**: Ils sont écrits en Groovy, offrant une plus grande flexibilité pour des tâches complexes et des personnalisations.
- **Contrôle total**: Permettent un contrôle précis sur chaque étape du pipeline.
- **Complexité**: Nécessitent une connaissance de Groovy et peuvent être plus difficiles à maintenir pour les utilisateurs moins expérimentés.

Jenkinsfile (Scripted Pipeline)

```
node {  ❶
    stage('Build') { ❷
        // ❸
    }
    stage('Test') { ❹
        // ❺
    }
    stage('Deploy') { ❻
        // ❼
    }
}
```

6. Création des pipelines déclaratives sous Jenkins

Exemple de création d’un pipeline déclarative

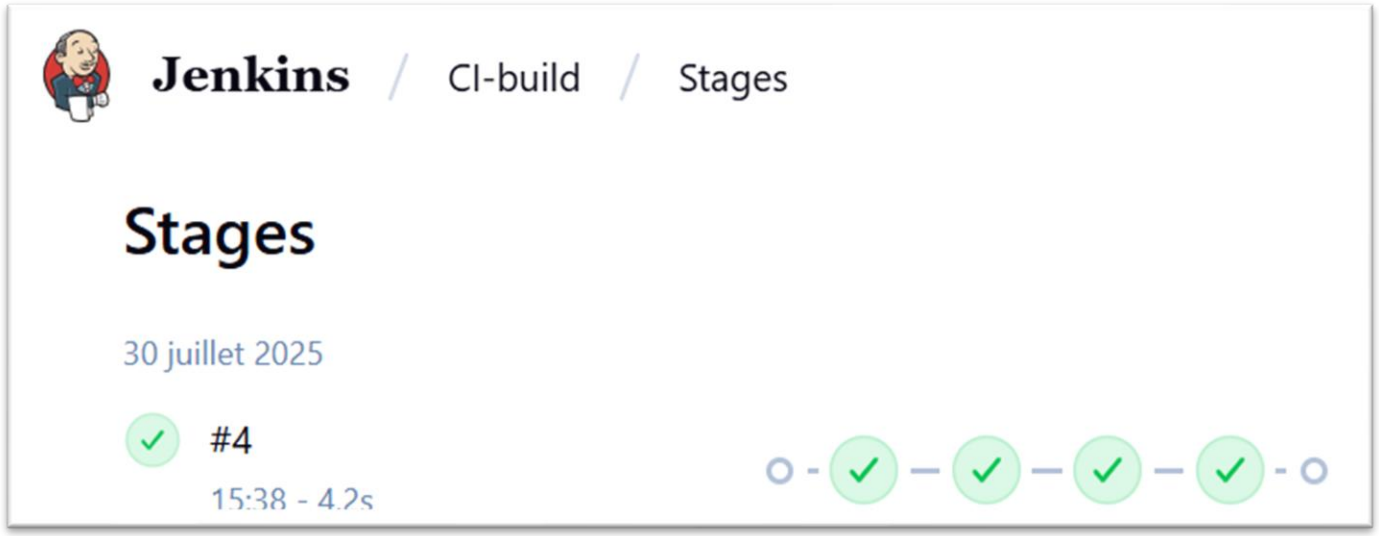


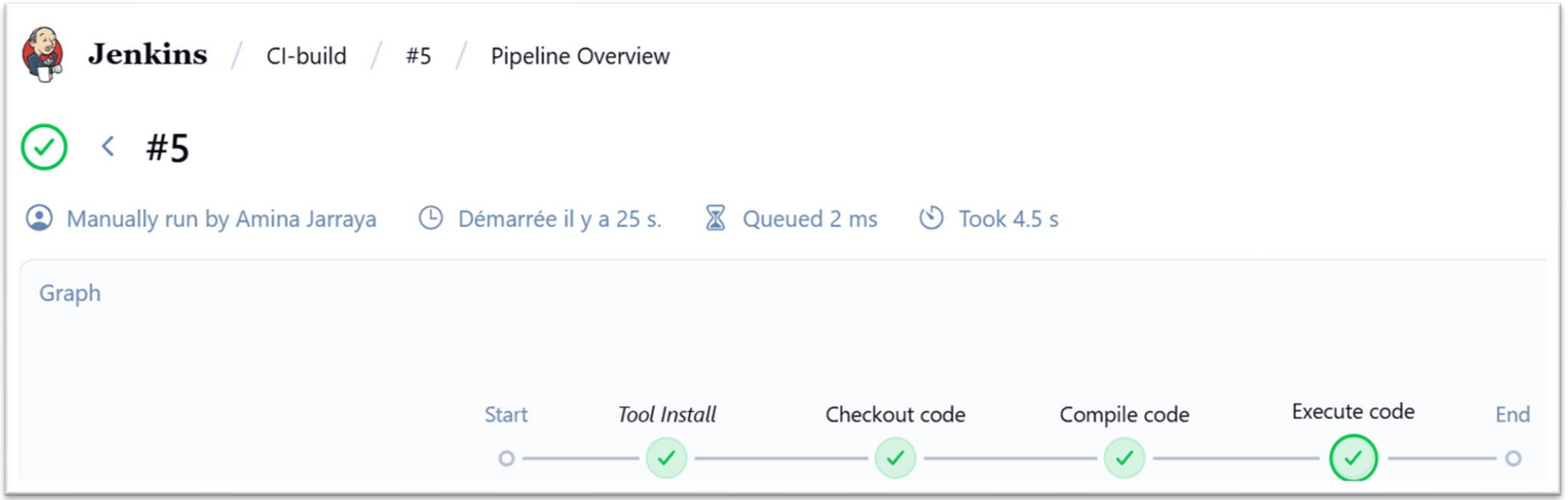
Le script déclarative

Script ?

```
pipeline {
        agent any
        tools {
            jdk 'JDK21'
        }
        stages {
            stage('Checkout code')
            {
                steps {
                    git branch: 'main', url: 'https://github.com/Jamina-ENSI/Hello_example.git'
                }
            }
            stage('Compile code')
           {
               steps {
                   sh 'javac Test.java'
               }
           }
           stage('Execute code')
           {
               steps {
                   sh 'java Test '
               }
           }
       }
```

Exécution du pipeline





Utilisation de jenkinsfile

- Créer jenkins file sous votre projet

```
root@DESKTOP-SKOOCDK:/mnt/c/mydevops/Hello# git init
```

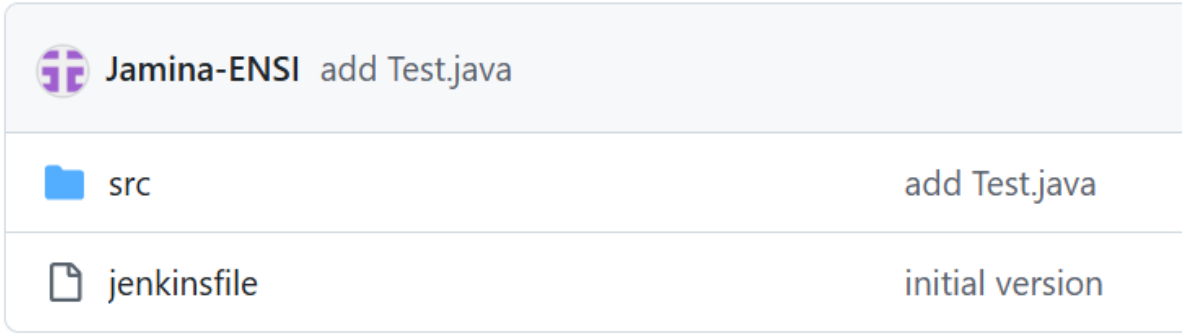
```
root@DESKTOP-SKOOCDK:/mnt/c/mydevops/Hello# vi jenkinsfile
```

- Tapez vi mon_fichier.txt.
- Pour entrer en mode édition, appuyez sur la touche i.
- Tapez votre texte.
- Pour enregistrer et quitter, appuyez sur la touche Échap, puis tapez :wq et appuyez sur Entrée.

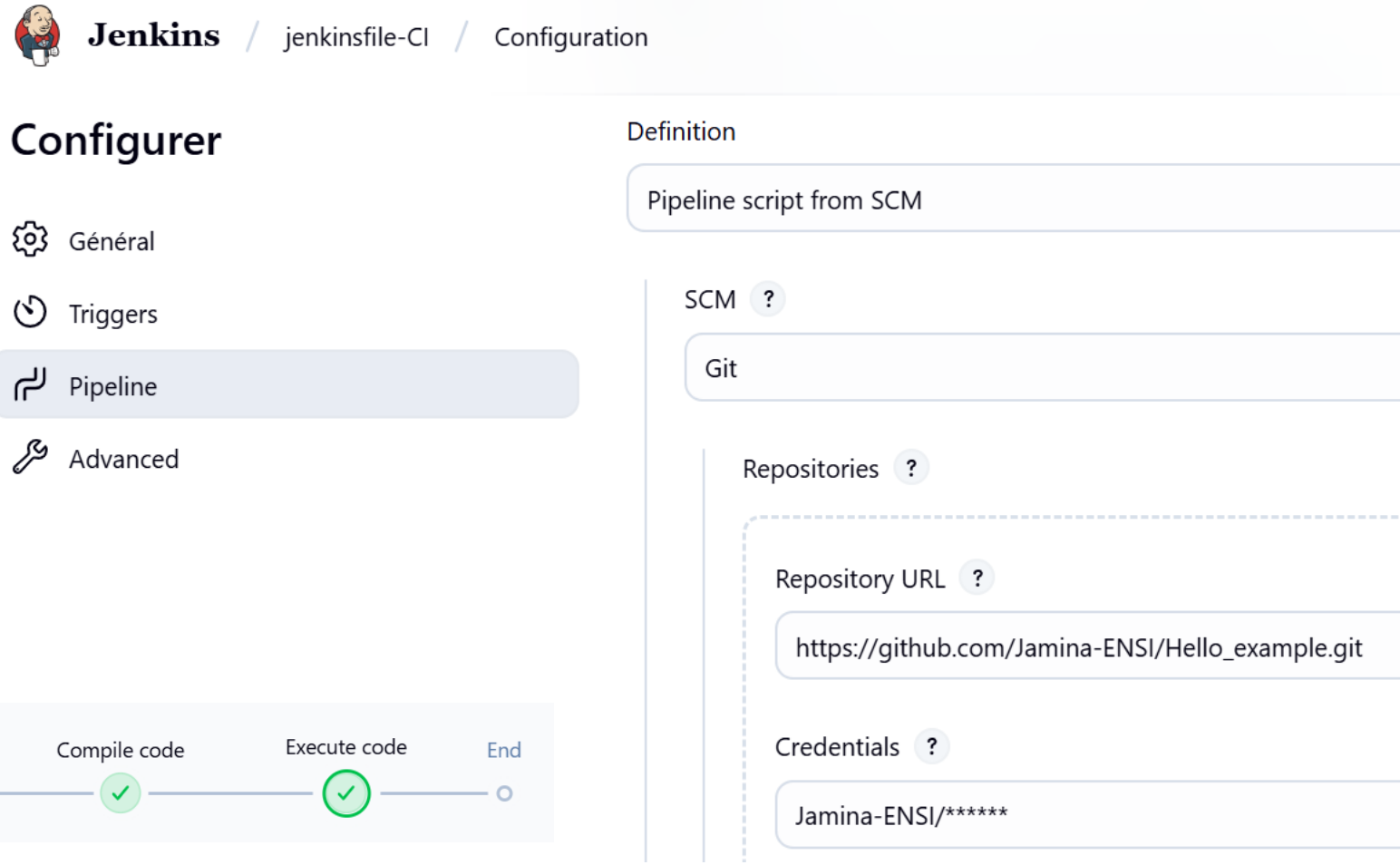
```
pipeline {
        agent any
        tools {
            jdk 'JDK21'
        }
        stages {
            stage('Checkout code')
            {
                steps {
                    git branch: 'main', url: 'https://github.com/Jamina-ENSI/Hello_example.git'
                }
            }
            stage('Compile code')
            {
                steps {
                    sh 'javac Test.java'
                }
            }
            stage('Execute code')
            {
                steps {
                    sh 'java Test '
                }
            }
        }
    }
```

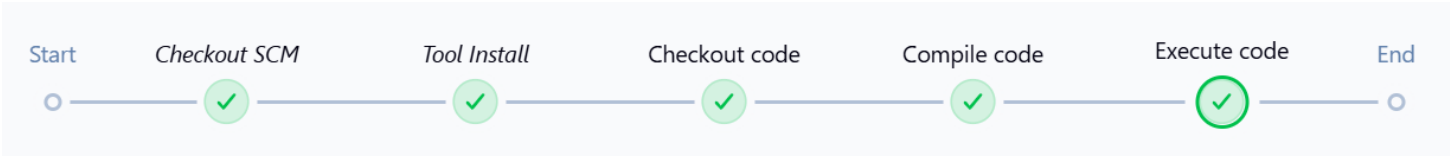
Utilisation de jenkinsfile

- Push le fichier Jenkinsfile dans le repository



- Créer un nouveau pipeline en spécifiant que le pipeline est à partir de GIT





7. Github Webhook et Jenkins

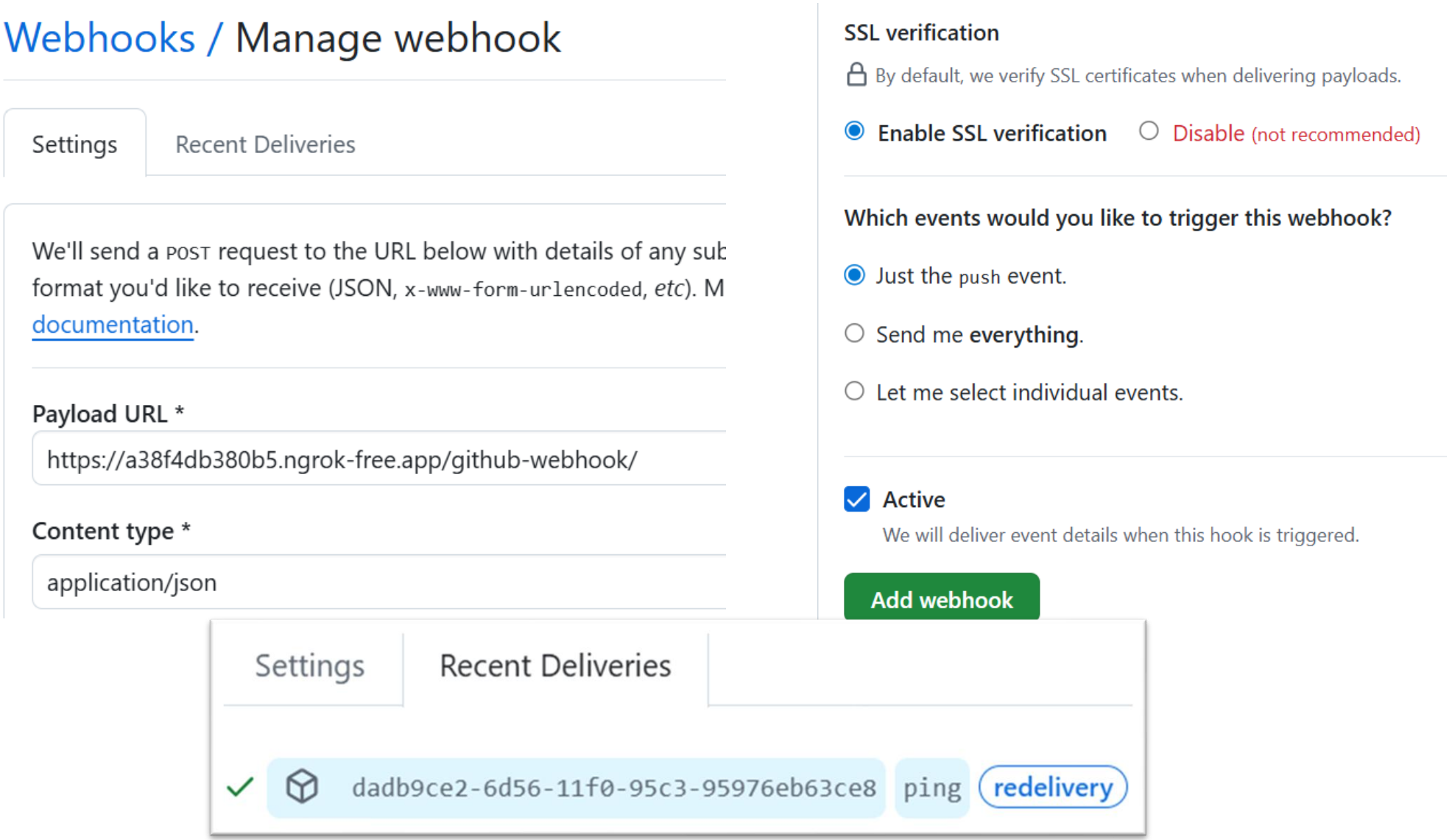
Github Webhook

- Un **webhook GitHub** est un mécanisme qui permet à GitHub de notifier un serveur externe (Jenkins) lorsqu'un événement spécifique se produit sur un dépôt ou sur un compte.

- Lorsque vous créez un nouveau pipeline « GithubWebhook », cochez l'option suivante dans la rubrique Triggers

Configurer Webhook sous Github

- Dans votre repository, allez sur **Settings –> Webhooks –> add webhook**



Utiliser ngrok pour créer une adresse publique

- Install ngrok sur windows (il se trouve dans microsoft app store)
- Ensuite tapez ngrok http 8080

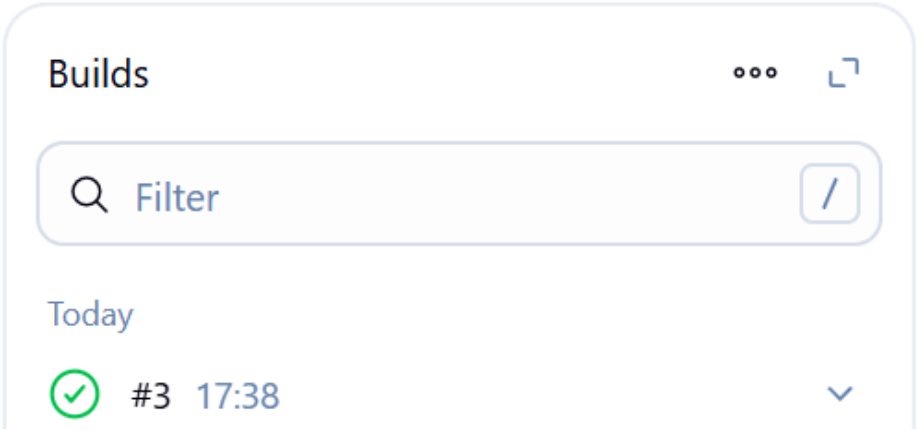
```
Session Status                online
Account                       Jamina-ENSI (Plan: Free)
Version                       3.24.0-msix
Region                        Europe (eu)
Latency                       63ms
Web Interface                 http://127.0.0.1:4040
Forwarding                    https://a38f4db380b5.ngrok-free.app -> http://loc

Connections                   ttl     opn     rt1     rt5     p50     p90
                              14      0       0.00    0.02    0.41    38.88
```

Notifier Jenkins par GitHub

- En créant avec succès le webhook sur github et si vous essayez de modifier le fichier jenkins sur github, un build se crée automatiquement dans le pipeline GithubWebhook :

- Rajoutez ce stage dans jenkinsfile sous github et sauvegarder == ça va créer automatiquement un build dans votre pipeline sous Jenkins.

```
25          stage('display message')
26             {
27                 steps {
28                     echo 'Hello from github '
29                 }
30             }
```



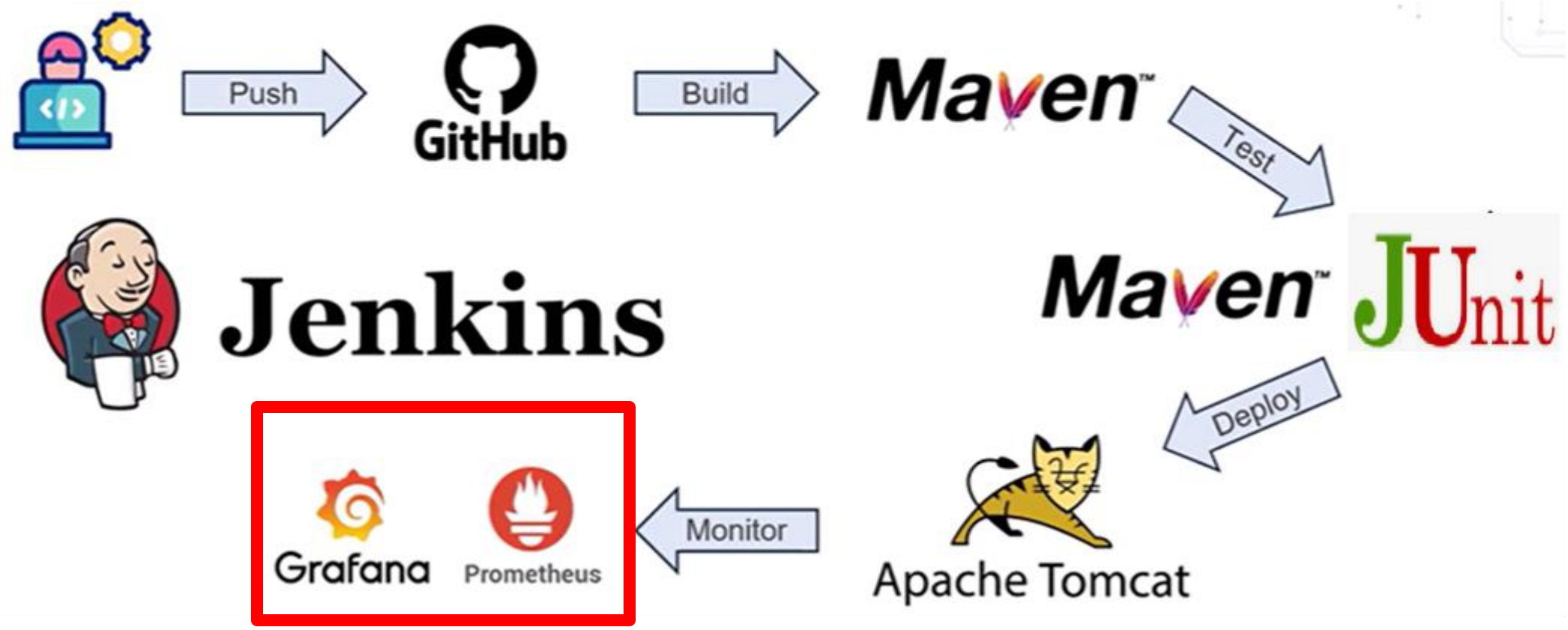
8. Mise en place d’un pipeline CI/CD (Github, maven, Junit, Apache Tomcat)

Pipeline CI/CD pour un micro-service

- Réalisez le pipeline **CI/CD (Continuous Integration/Continuous Delivery)** pour le micro-service **country-service** comme suit :

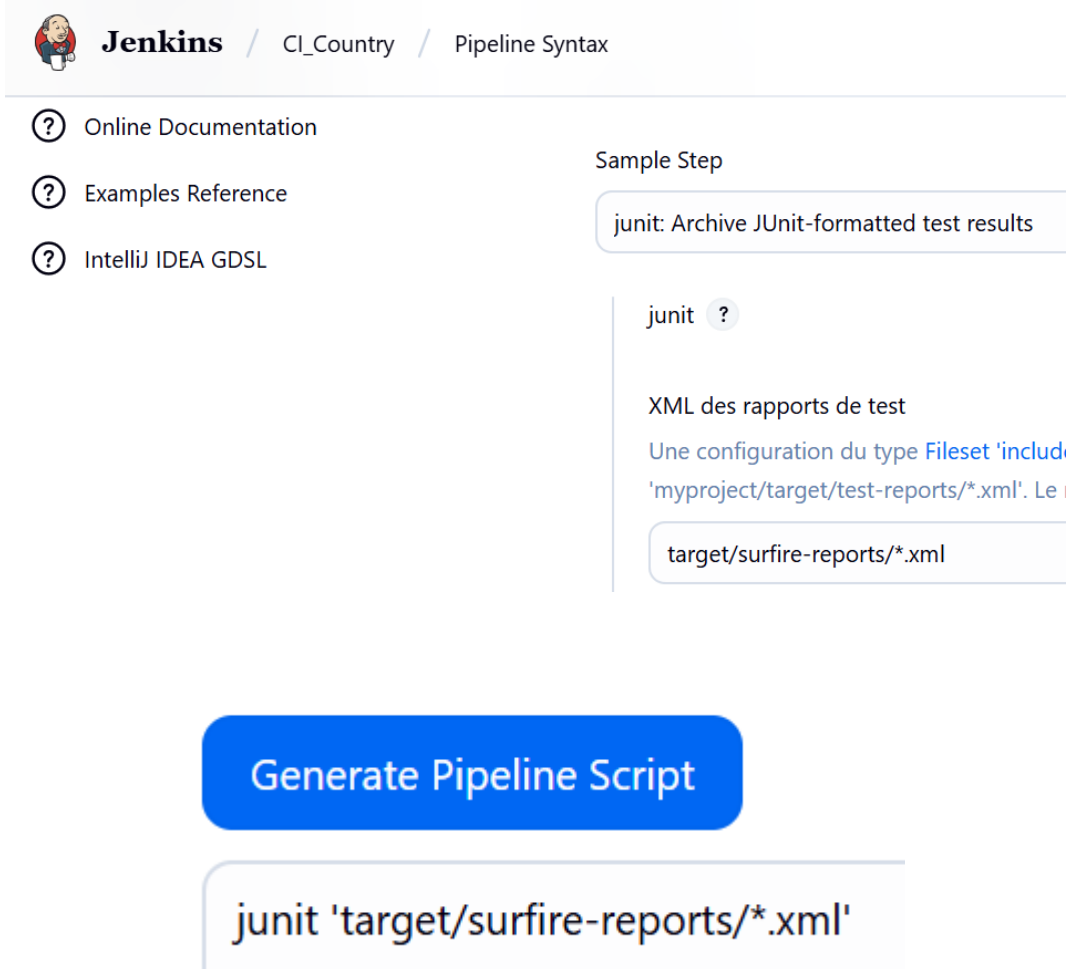
1. Faites le **push** du micro-service dans votre **github**
2. Utilisez un **github webhook** pour lancer le pipeline à chaque modification au niveau du github
3. Créez un **Jenkinsfile** dans votre github. Ce pipeline décrit les étapes de compilation, de tests et de déploiement (Livraison continue) avec le serveur Apache Tomcat.
4. Créez un job de type pipeline sous Jenkins qui automatise l'intégration des différents outils

Pipeline CI/CD pour un micro-service



On va le voir dans le dernier chapitre

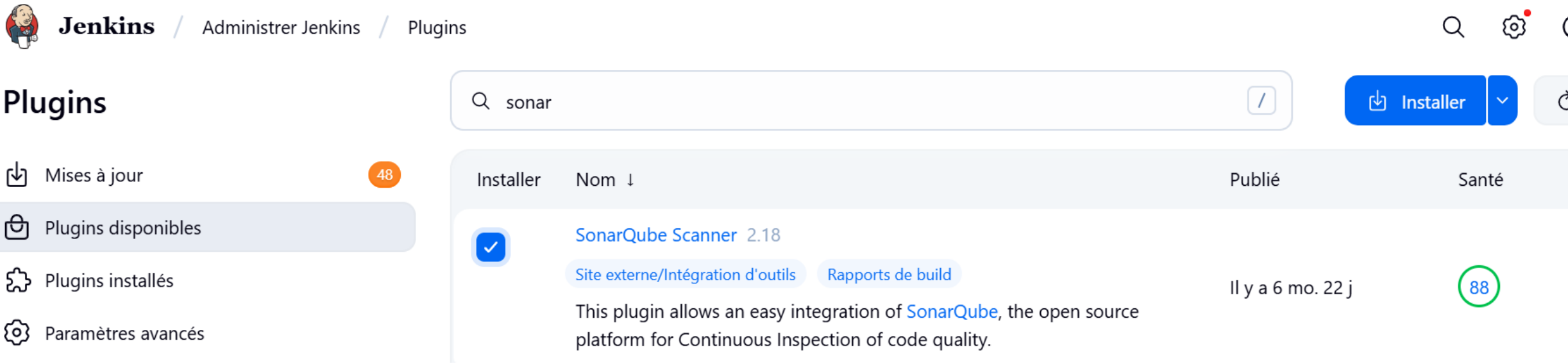
Jenkins file



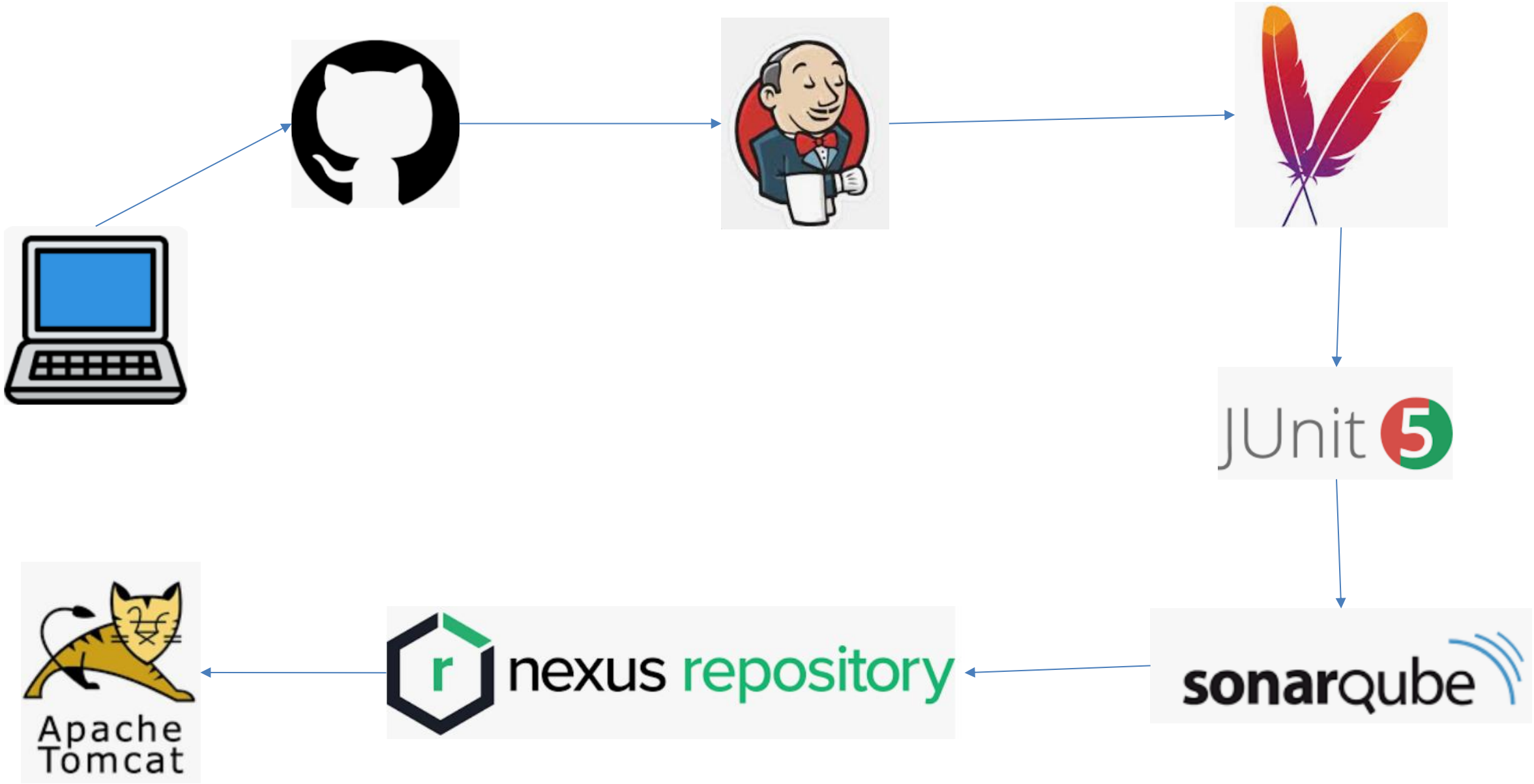

```
pipeline {
    agent any
    tools {
      maven 'mymaven'
    }
    stages {
      stage('Checkout code')
      {
        steps {
          git branch: 'master', url: 'https://github.com/Jamina-ENSI/Country-service.git'
        }
      }
      stage('Compile code')
      {
        steps {
          sh 'mvn compile'
        }
      }
      stage('Test code')
      {
        steps {
          sh 'mvn test '
        }
        post{
          success {
            junit allowEmptyResults: true, testResults: '**/target/surfire-reports/*.xml'
          }
        }
      }
      stage('Package code')
      {
        steps {
          sh 'mvn package '
        }
      }
    }
  }
```

3. Mise en place d’un pipeline CI/CD (Github, maven, Junit,SonarQube, Nexus Repository, Apache Tomcat)

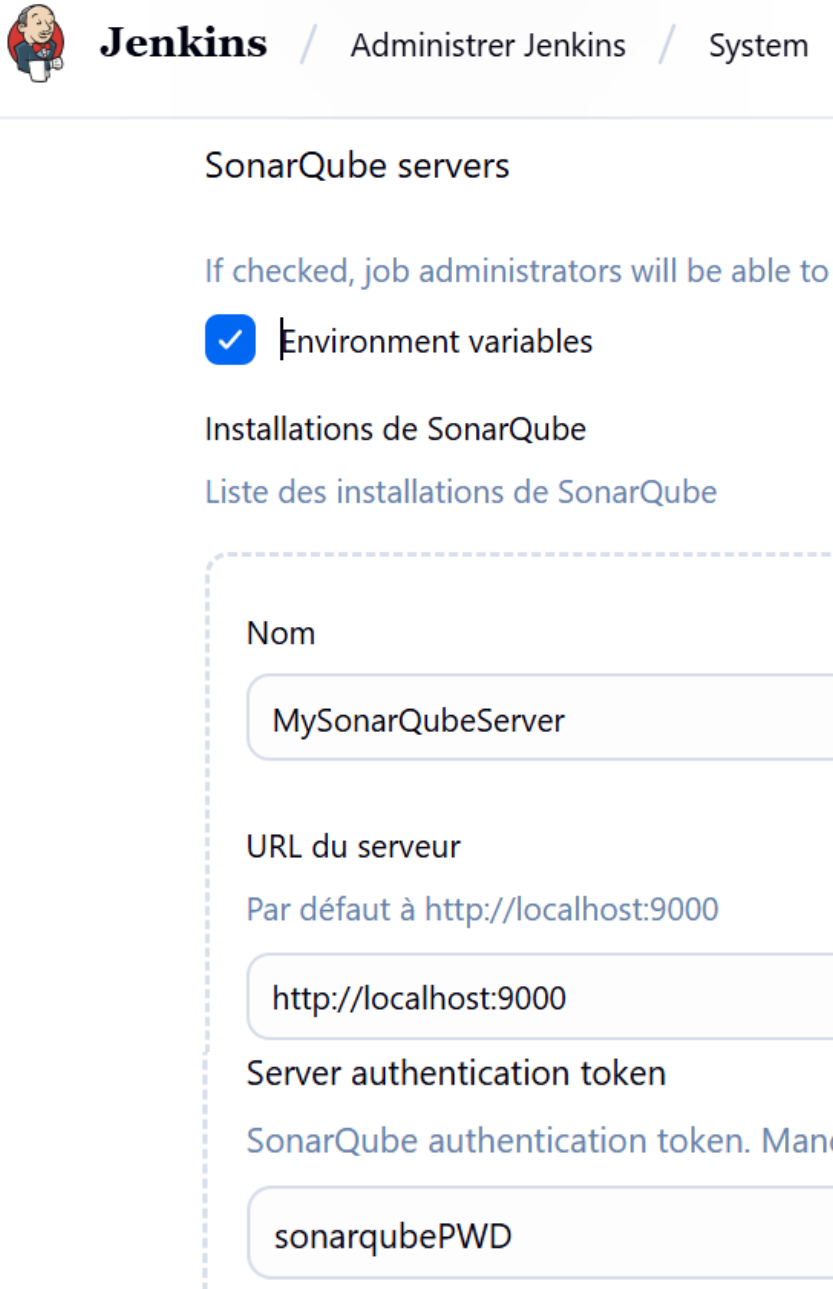
Installer le plugin SonarQube scanner



Pipeline CI/CD à mettre en place

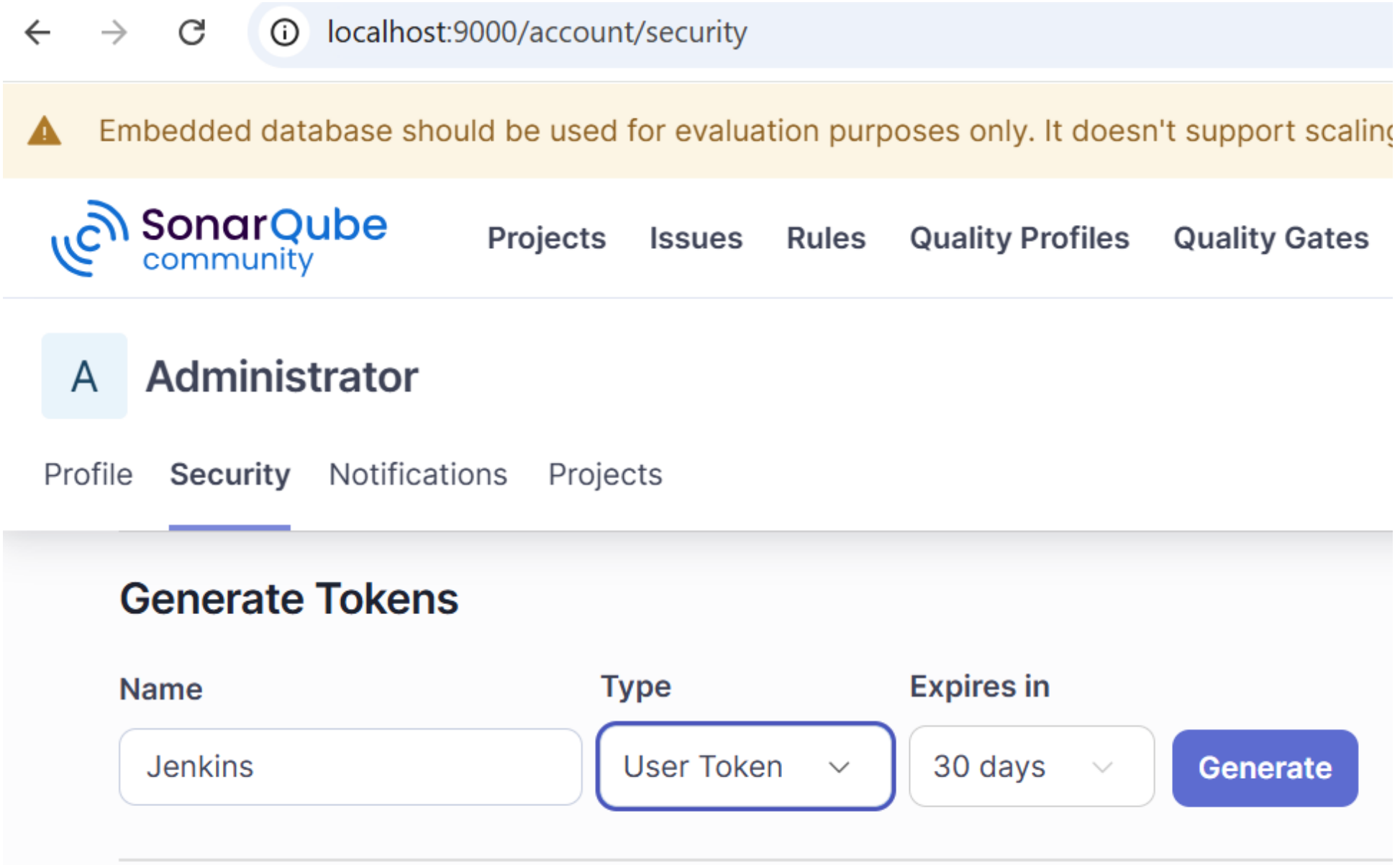


Configurer Sonarqube Server dans Jenkins



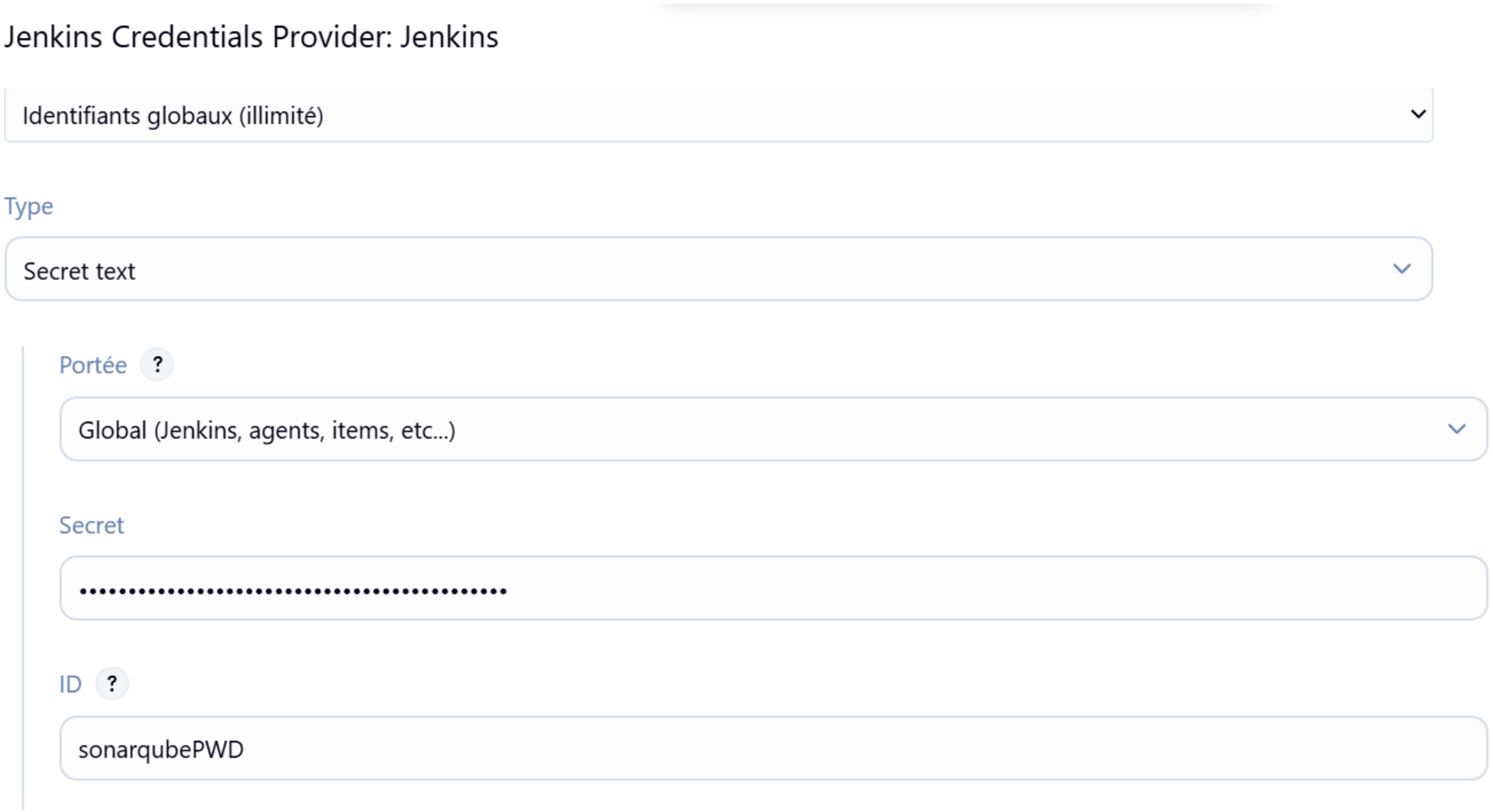
Configurer Sonarqube Server dans Jenkins

- Générer un token utilisateur

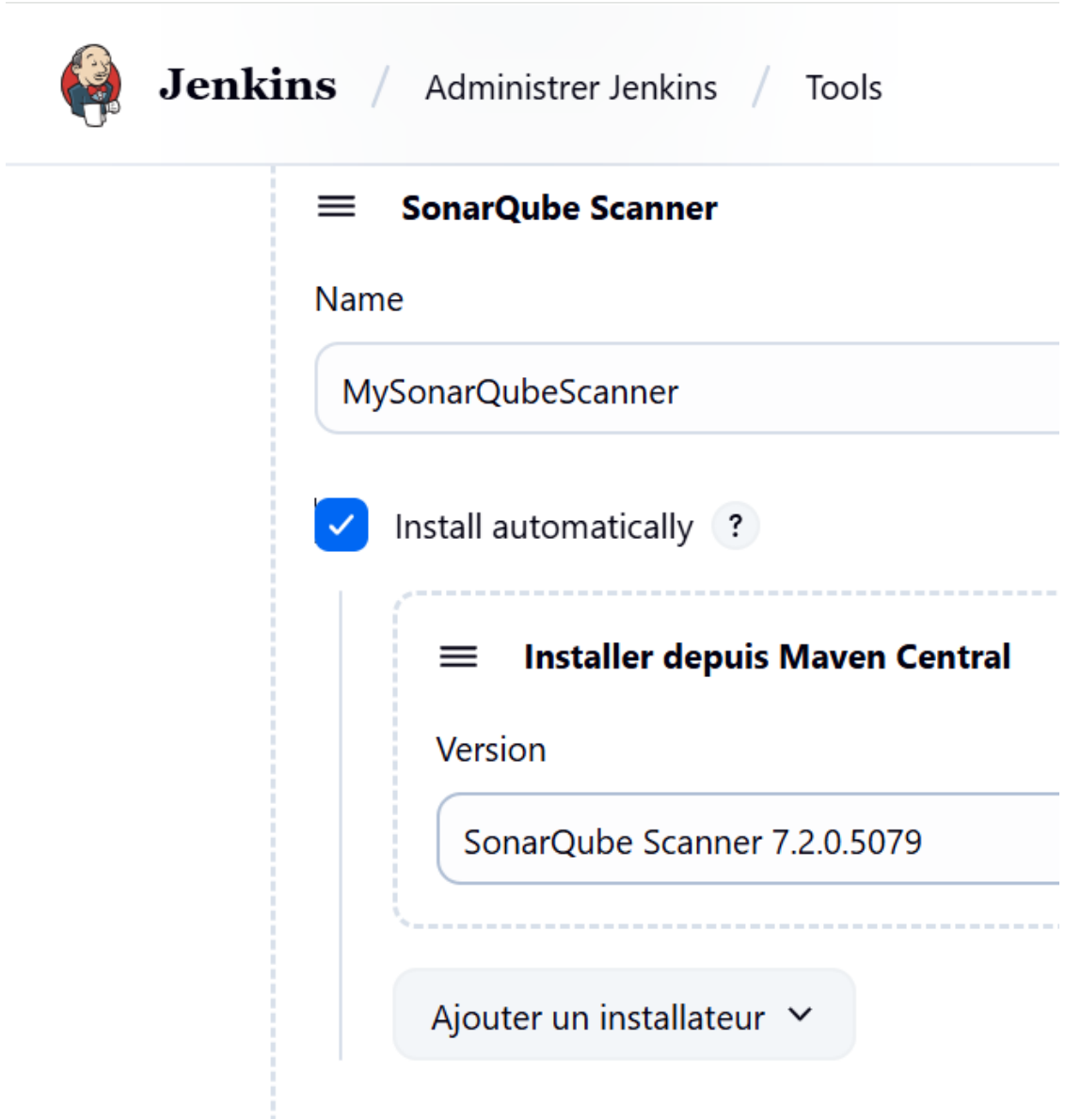


Configurer Sonarqube Server dans Jenkins

- Générer un identifiant sonarqube



Configurer SonarqubeScanner



Le script Groovy

Script ?

```
pipeline {
        agent any
        tools {
            maven 'mymaven'
        }
        stages {
            stage('Checkout code')
            {
                steps {
                    git branch: 'master', url: 'https://github.com/Jamina-ENSI/Country-service.
                }
            }
            stage('Compile, test code, package in war file and store it in maven repo')
            {
                steps {
                    sh 'mvn clean install'
                }
                post{
                    success {
                        junit allowEmptyResults: true, testResults: '**/target/surfire-reports
                    }
                }
            }
```

Le script Groovy

```
24              stage('SonarQube Analysis')
25              {
26                  steps {
27                    withSonarQubeEnv(installationName: 'MySonarQubeServer', credentialsId: 'sonar
28                          sh "mvn sonar:sonar -Dsonar.projectKey=country-service -Dsonar.projectN
29                      }
30                  }
31              }
32          }
33      }
```

Résultat

Jenkins / CI_Sonarqube

- Status
- Changes
- Lancer un build
- Configurer
- Supprimer Pipeline
- SonarQube
- Stages
- Renommer
- Pipeline Syntax

CI_Sonarqube

This is an example of a CI pipeline using Sonarqube.

SonarQube Quality Gate

country-service Passed

server-side processing: Success

Liens permanents

- Dernier build (#4), il y a 3 mn 56 s
- Dernier build stable (#4), il y a 3 mn 56 s
- Dernier build avec succès (#4), il y a 3 mn 56 s
- Dernier build en échec (#2), il y a 11 mn

Travail à faire

- Vous allez compléter le pipeline pour :
  - Stocker les fichiers .war ou .jar générés par Maven dans un dépôt **nexus**. Vous pouvez même stocker les répports générés par Sonarqube (cnes report).
  - Déployer le fichier .war dans le serveur **Tomcat**.
- **Nexus Repository** : une plateforme qui permet de stocker, gérer et distribuer des artefacts logiciels, tels que des fichiers JAR, WAR, des images Docker, des paquets NPM, etc. https://help.sonatype.com/en/download.html

- *"Apprendre par le projet, c'est découvrir par l'action, créer par la compréhension, et réussir par la persévérance."*

amina.jarraya@ensi-uma.tn

ENSI Manouba