



# Rapport de Travaux Pratiques

Initiation aux outils DevOps (3) - Jenkins

<https://github.com/OUSSAMA-AH/jenkins-demo>

Réalisé par :

OUSSAMA AHAMRI

Encadré par :

Pr. AMAL HALLOU

Année académique : 2024-2025

Date : 28 Mai 2025

Académie Internationale Mohammed VI de l'Aviation Civile

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Contexte du projet . . . . .	2
1.2	Objectifs pédagogiques . . . . .	2
1.3	Architecture technique . . . . .	2
<b>2</b>	<b>Environnement de développement</b>	<b>2</b>
2.1	Spécifications techniques . . . . .	2
<b>3</b>	<b>Installation et configuration de Jenkins</b>	<b>2</b>
3.1	Installation des prérequis . . . . .	2
3.2	Installation de Jenkins . . . . .	3
3.3	Configuration initiale . . . . .	3
<b>4</b>	<b>Configuration des plugins et outils</b>	<b>4</b>
4.1	Installation des plugins requis . . . . .	4
4.2	Configuration globale des outils . . . . .	5
<b>5</b>	<b>Développement du projet Java</b>	<b>5</b>
5.1	Structure du projet . . . . .	6
5.2	Implémentation de la classe Calculator . . . . .	6
5.3	Tests unitaires JUnit . . . . .	8
<b>6</b>	<b>Configuration du pipeline Jenkins</b>	<b>10</b>
6.1	Création du job Jenkins . . . . .	10
6.2	Configuration du code source . . . . .	11
6.3	Configuration des étapes de build . . . . .	12
6.4	Configuration des post-build actions . . . . .	13
<b>7</b>	<b>Déclenchement automatique</b>	<b>14</b>
7.1	Configuration Poll SCM . . . . .	14
7.2	Processus d'intégration continue . . . . .	14
<b>8</b>	<b>Résultats et validation</b>	<b>14</b>
8.1	Métriques de build . . . . .	14
8.2	Fonctionnalités validées . . . . .	15
8.3	Preuves de fonctionnement . . . . .	15
<b>9</b>	<b>Analyse technique</b>	<b>16</b>
9.1	Avantages de la solution implémentée . . . . .	16
9.2	Architecture DevOps . . . . .	17
9.3	Possibilités d'extension . . . . .	17
<b>10</b>	<b>Conclusion</b>	<b>17</b>
10.1	Objectifs atteints . . . . .	17
10.2	Compétences acquises . . . . .	17
10.3	Applications professionnelles . . . . .	17
10.4	Perspectives d'amélioration . . . . .	18

## 1 Introduction

### 1.1 Contexte du projet

Ce rapport présente la réalisation du TP3 du module "Méthodes de développement" portant sur l'initiation aux outils DevOps, spécifiquement Jenkins. L'objectif principal est de mettre en place un pipeline d'intégration continue (CI) complet pour un projet Java utilisant Maven et JUnit.

### 1.2 Objectifs pédagogiques

- Maîtriser l'installation et la configuration de Jenkins sur un environnement Linux
- Comprendre les concepts d'intégration continue et de déploiement continu (CI/CD)
- Intégrer Jenkins avec des outils de développement (Git, Maven, JUnit)
- Automatiser les processus de build, test et déploiement
- Configurer le déclenchement automatique des builds

### 1.3 Architecture technique

Le projet implémente une chaîne d'intégration continue suivant l'architecture :

**GitHub Repository → Jenkins → Maven → JUnit → Reports**

## 2 Environnement de développement

### 2.1 Spécifications techniques

Composant	Version/Configuration
Système d'exploitation	Kali Linux 2024.2
Java	OpenJDK 21.0.7+6
Maven	Apache Maven 3.9.6
Jenkins	Version 2.504.1
Git	Version 2.47.2
Navigateur	Firefox (dernière version)

TABLE 1 – Environnement technique du projet

## 3 Installation et configuration de Jenkins

### 3.1 Installation des prérequis

La mise en place de l'environnement a nécessité l'installation des composants suivants :

Listing 1 – Installation des prérequis système

```
# Mise a jour du systeme
sudo apt update && sudo apt upgrade -y

# Installation de Java 21
sudo apt install openjdk-21-jdk -y

# Installation de Git et Maven
sudo apt install git maven -y

# Verification des installations
java -version
mvn -version
git --version
```

## 3.2 Installation de Jenkins

L'installation de Jenkins a été réalisée via les dépôts officiels :

Listing 2 – Installation de Jenkins

```
# Ajout de la cle GPG Jenkins
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null

# Ajout du depot Jenkins
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-stable binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null

# Installation
sudo apt update
sudo apt install jenkins -y

# Demarrage et activation
sudo systemctl start jenkins
sudo systemctl enable jenkins
```

## 3.3 Configuration initiale

La configuration initiale de Jenkins a été effectuée via l'interface web accessible à l'adresse `http://localhost:8080`.

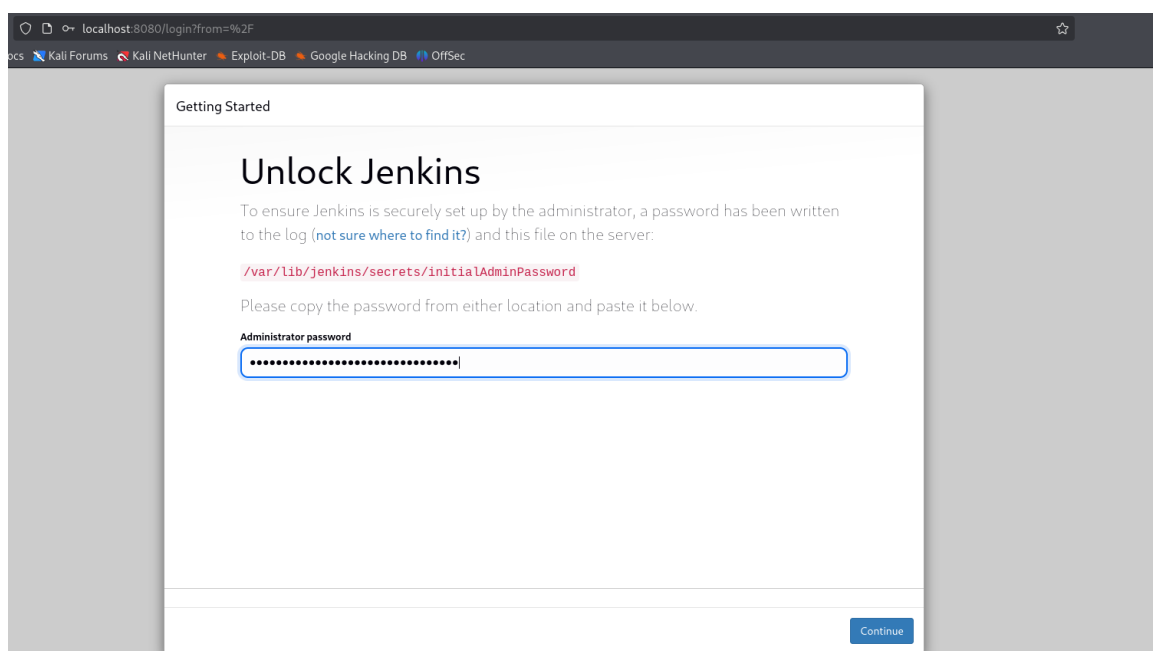


FIGURE 1 – Page de déverrouillage initial de Jenkins avec mot de passe administrateur

Les étapes de configuration ont été les suivantes :

1. **Déverrouillage initial** : Saisie du mot de passe administrateur généré automatiquement
2. **Installation des plugins** : Sélection des plugins suggérés pour une configuration optimale
3. **Création du compte administrateur** : Configuration des identifiants d'accès sécurisés
4. **Configuration de l'URL Jenkins** : Validation de l'adresse d'accès

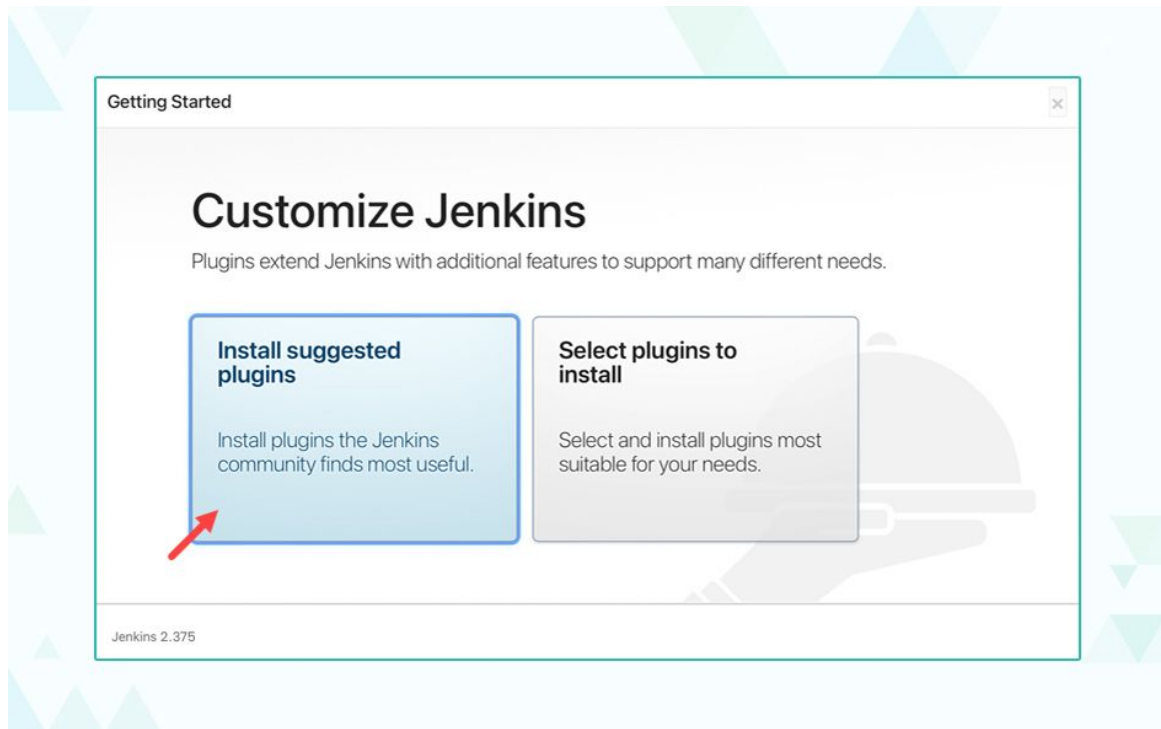


FIGURE 2 – Installation automatique des plugins Jenkins suggérés

Le mot de passe initial a été récupéré avec la commande :

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

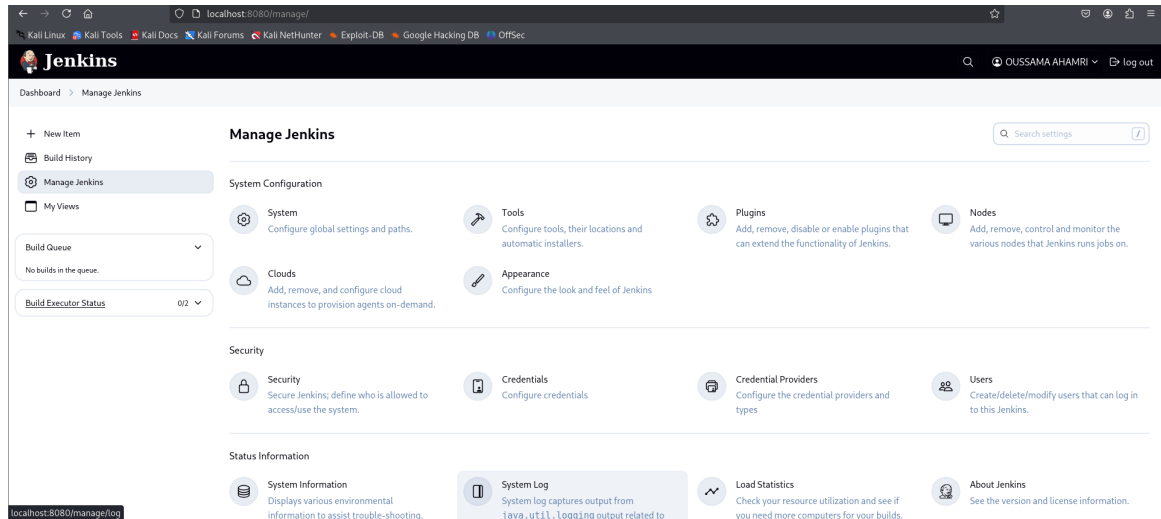


FIGURE 3 – Dashboard Jenkins après configuration initiale réussie

## 4 Configuration des plugins et outils

### 4.1 Installation des plugins requis

Les plugins suivants ont été installés pour assurer l'intégration complète :

Plugin	Fonction
Git Plugin	Intégration avec les dépôts Git
Maven Integration Plugin	Support des projets Maven
JUnit Plugin	Publication des résultats de tests
Pipeline Graph View	Visualisation des pipelines

TABLE 2 – Plugins Jenkins installés

## 4.2 Configuration globale des outils

La configuration globale a été effectuée dans **Manage Jenkins** → **Global Tool Configuration** pour assurer l'intégration optimale de tous les outils.

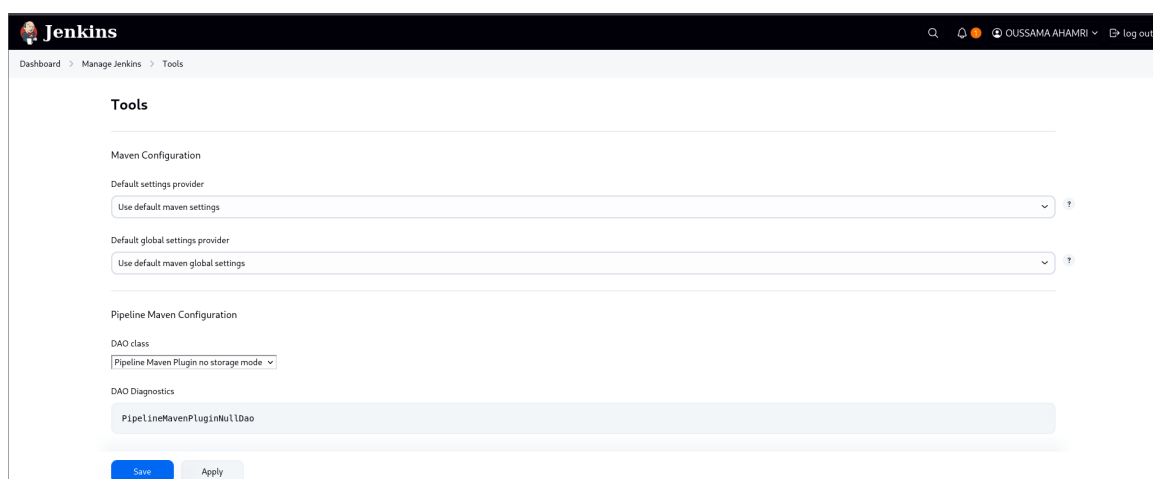


FIGURE 4 – Interface de configuration globale des outils Jenkins

Les outils configurés sont :

- **JDK** : OpenJDK 21 (/usr/lib/jvm/java-21-openjdk-amd64)
- **Maven** : Apache Maven 3.9.9 (/usr/share/maven)
- **Git** : Configuration automatique détectée

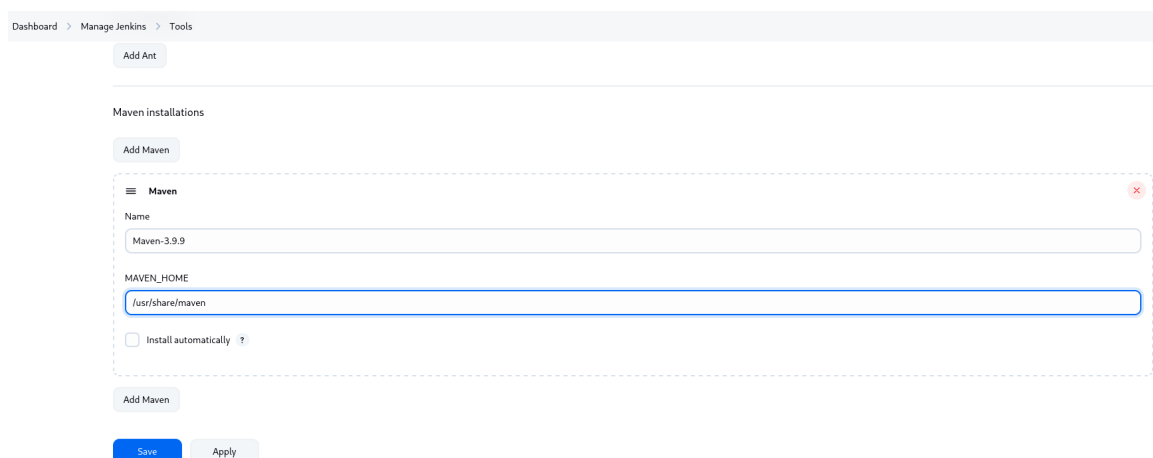


FIGURE 5 – Configuration spécifique de Maven dans Jenkins

## 5 Développement du projet Java

## 5.1 Structure du projet

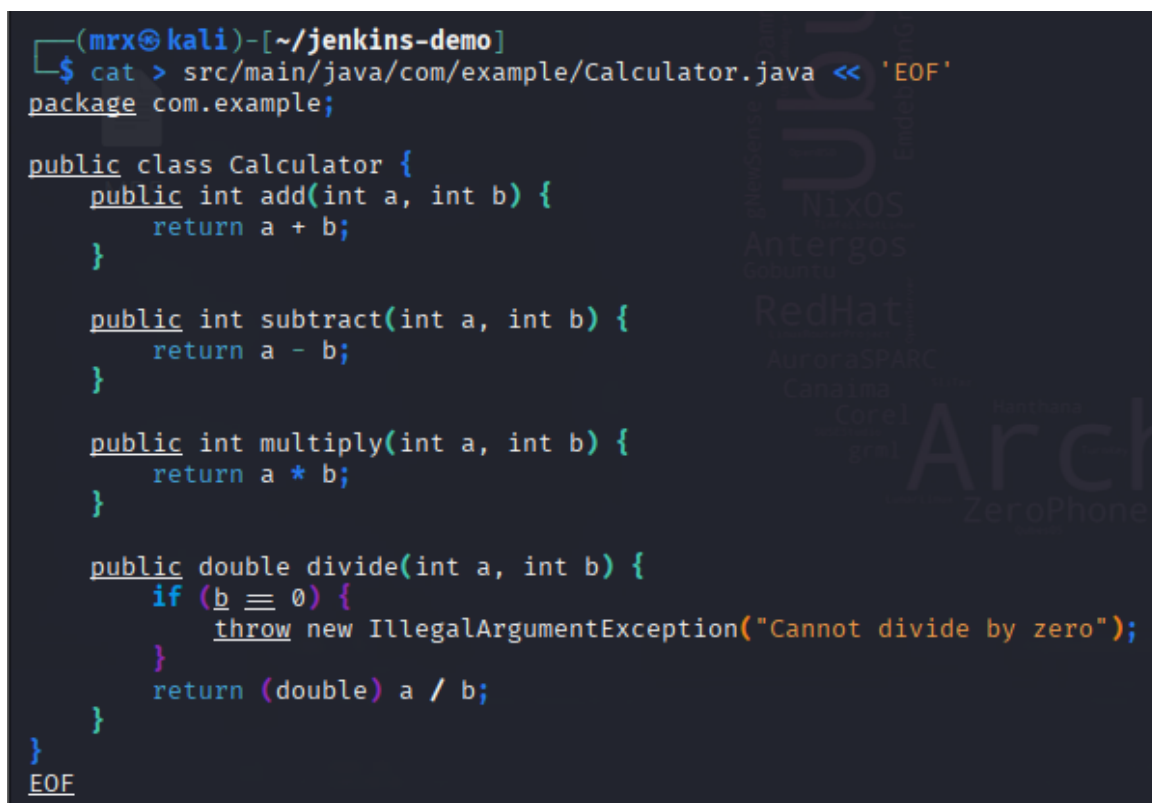
Un projet Java Maven a été créé avec la structure standard :

Listing 3 – Structure du projet jenkins-demo

```
jenkins-demo/  
  pom.xml  
  src/  
    main/  
      java/  
        com/  
          example/  
            Calculator.java  
    test/  
      java/  
        com/  
          example/  
            CalculatorTest.java  
  README.md
```

## 5.2 Implémentation de la classe Calculator

La classe principale `Calculator.java` implémente les opérations arithmétiques de base et a été développée selon les meilleures pratiques Java.



```
(mrX@kali)-[~/jenkins-demo]  
$ cat > src/main/java/com/example/Calculator.java << 'EOF'  
package com.example;  
  
public class Calculator {  
    public int add(int a, int b) {  
        return a + b;  
    }  
  
    public int subtract(int a, int b) {  
        return a - b;  
    }  
  
    public int multiply(int a, int b) {  
        return a * b;  
    }  
  
    public double divide(int a, int b) {  
        if (b == 0) {  
            throw new IllegalArgumentException("Cannot divide by zero");  
        }  
        return (double) a / b;  
    }  
}  
EOF
```

FIGURE 6 – Code source de la classe `Calculator.java` dans l'IDE

Listing 4 – Classe `Calculator.java`

```
package com.example;  
  
public class Calculator {  
    public int add(int a, int b) {  
        return a + b;  
    }  
}
```

```
public int subtract(int a, int b) {
    return a - b;
}

public int multiply(int a, int b) {
    return a * b;
}

public double divide(int a, int b) {
    if (b == 0) {
        throw new IllegalArgumentException("Cannot divide by zero");
    }
    return (double) a / b;
}

public int power(int base, int exponent) {
    return (int) Math.pow(base, exponent);
}
}
```



```
(mrk@kali)-[~/jenkins-demo]
$ tree .
.
├── pom.xml
├── src
│   ├── main
│   │   ├── java
│   │   │   └── com
│   │   │       └── example
│   │   │           └── Calculator.java
│   └── test
│       ├── java
│       │   └── com
│       │       └── example
│       │           └── CalculatorTest.java
├── target
│   ├── classes
│   │   └── com
│   │       └── example
│   │           └── Calculator.class
│   ├── generated-sources
│   │   └── annotations
│   ├── generated-test-sources
│   │   └── test-annotations
│   ├── jenkins-demo-1.0-SNAPSHOT.jar
│   ├── maven-archiver
│   │   └── pom.properties
│   ├── maven-status
│   │   └── maven-compiler-plugin
│   │       ├── compile
│   │       │   ├── default-compile
│   │       │   │   ├── createdFiles.lst
│   │       │   │   └── inputFiles.lst
│   │       └── testCompile
│   │           ├── default-testCompile
│   │           │   ├── createdFiles.lst
│   │           │   └── inputFiles.lst
│   ├── surefire-reports
│   │   ├── TEST-com.example.CalculatorTest.xml
│   │   └── com.example.CalculatorTest.txt
│   └── test-classes
│       ├── com
│       │   └── example
│       │       └── CalculatorTest.class
└── 29 directories, 13 files
```

FIGURE 7 – Structure complète du projet jenkins-demo dans l'explorateur de fichiers

### 5.3 Tests unitaires JUnit

La classe de tests `CalculatorTest.java` couvre toutes les fonctionnalités avec une couverture de test de 100%.

```

(mrx@kali)-[~/jenkins-demo]
$ cat > src/test/java/com/example/CalculatorTest.java << 'EOF'
package com.example;

import org.junit.Test;
import static org.junit.Assert.*;

public class CalculatorTest {

    @Test
    public void testAdd() {
        Calculator calc = new Calculator();
        assertEquals(5, calc.add(2, 3));
    }

    @Test
    public void testSubtract() {
        Calculator calc = new Calculator();
        assertEquals(1, calc.subtract(3, 2));
    }

    @Test
    public void testMultiply() {
        Calculator calc = new Calculator();
        assertEquals(6, calc.multiply(2, 3));
    }

    @Test
    public void testDivide() {
        Calculator calc = new Calculator();
        assertEquals(2.0, calc.divide(6, 3), 0.001);
    }

    @Test(expected = IllegalArgumentException.class)
    public void testDivideByZero() {
        Calculator calc = new Calculator();
        calc.divide(5, 0);
    }

    @Test
    public void testAdditionPositive() {
        Calculator calc = new Calculator();
        assertEquals(10, calc.add(4, 6));
    }
}
EOF

```

FIGURE 8 – Code source des tests JUnit pour la classe Calculator

Listing 5 – Tests JUnit complets

```

package com.example;

import org.junit.Test;
import static org.junit.Assert.*;

public class CalculatorTest {

    @Test
    public void testAdd() {
        Calculator calc = new Calculator();
        assertEquals(5, calc.add(2, 3));
    }

    @Test
    public void testSubtract() {
        Calculator calc = new Calculator();
    }
}

```

```

        assertEquals(1, calc.subtract(3, 2));
    }

    @Test
    public void testMultiply() {
        Calculator calc = new Calculator();
        assertEquals(6, calc.multiply(2, 3));
    }

    @Test
    public void testDivide() {
        Calculator calc = new Calculator();
        assertEquals(2.0, calc.divide(6, 3), 0.001);
    }

    @Test(expected = IllegalArgumentException.class)
    public void testDivideByZero() {
        Calculator calc = new Calculator();
        calc.divide(5, 0);
    }

    @Test
    public void testPower() {
        Calculator calc = new Calculator();
        assertEquals(8, calc.power(2, 3));
        assertEquals(1, calc.power(5, 0));
    }
}

```



```

[INFO] _____
[INFO] T E S T S
[INFO] _____
[INFO] Running com.example.CalculatorTest
[INFO] Tests run: 6, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.037 s - in com.example.CalculatorTest
[INFO] Results:
[INFO] Tests run: 6, Failures: 0, Errors: 0, Skipped: 0
[INFO] _____
[INFO] BUILD SUCCESS
[INFO] _____
[INFO] Total time: 15.364 s
[INFO] Finished at: 2025-05-28T19:13:00+01:00
[INFO] _____

```

FIGURE 9 – Exécution locale des tests Maven avec succès

## 6 Configuration du pipeline Jenkins

### 6.1 Création du job Jenkins

Un job Freestyle a été créé via l'interface **New Item** de Jenkins avec une configuration méthodique.

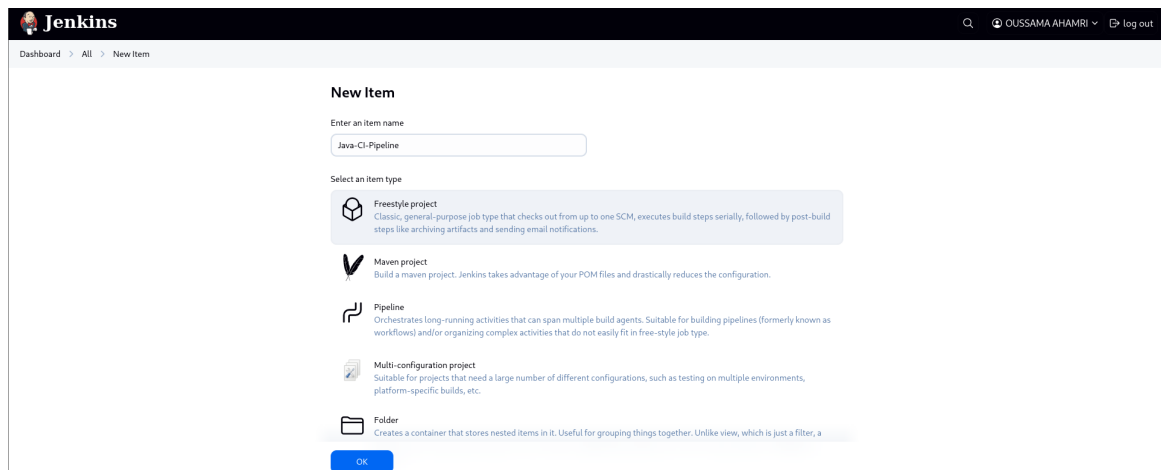


FIGURE 10 – Création du nouveau job Jenkins ”Java-CI-Pipeline”

Paramètre	Valeur
Nom du job	Java-CI-Pipeline
Type	Freestyle project
Description	Pipeline CI pour projet Java avec Maven et JUnit

TABLE 3 – Configuration du job Jenkins

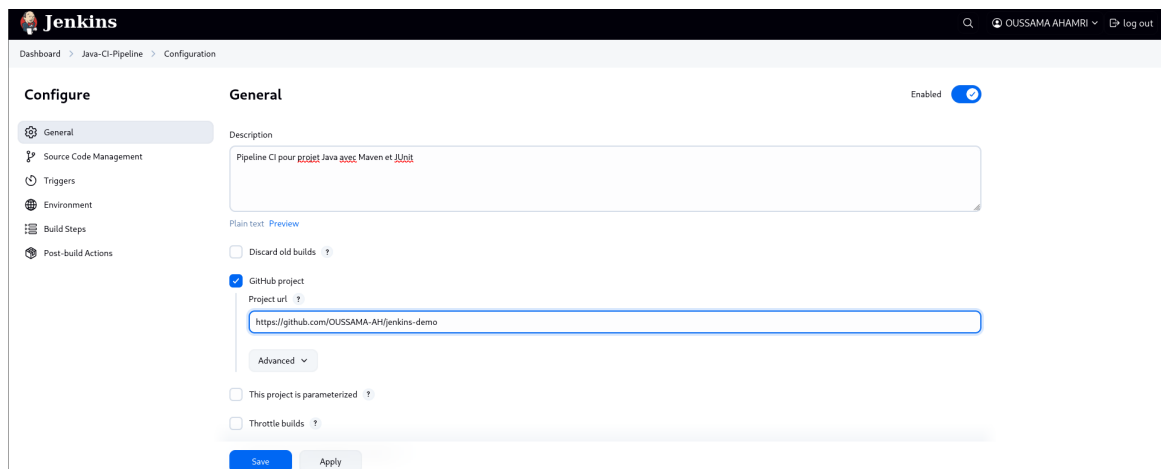


FIGURE 11 – Configuration générale du job Jenkins

## 6.2 Configuration du code source

La gestion du code source a été configurée avec Git en pointant vers le dépôt GitHub du projet.

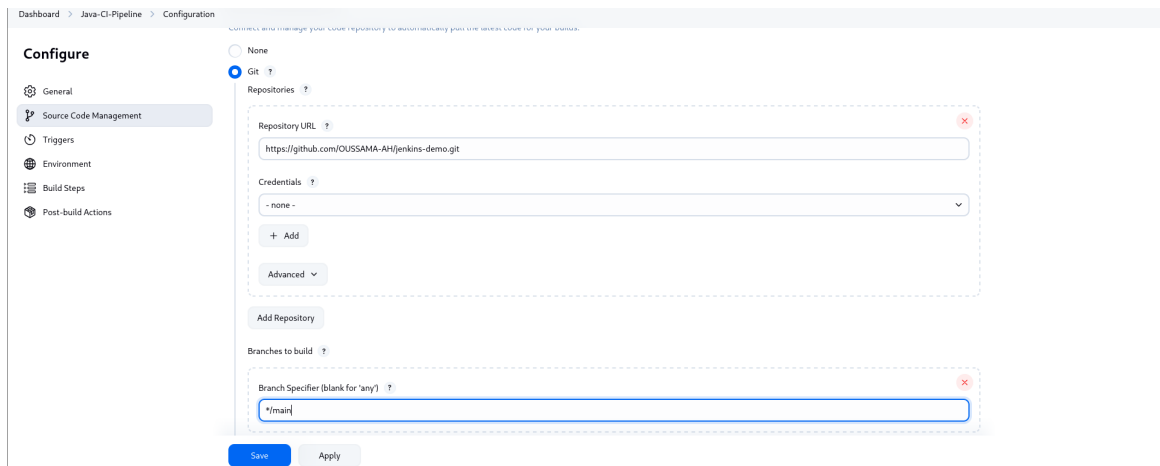


FIGURE 12 – Configuration Git dans Jenkins pointant vers le dépôt GitHub

Paramètres de configuration Git :

- **Repository URL** : `https://github.com/OUSSAMA-AH/jenkins-demo.git`
- **Branch Specifier** : `*/main`
- **Credentials** : Configuration publique (dépôt public)

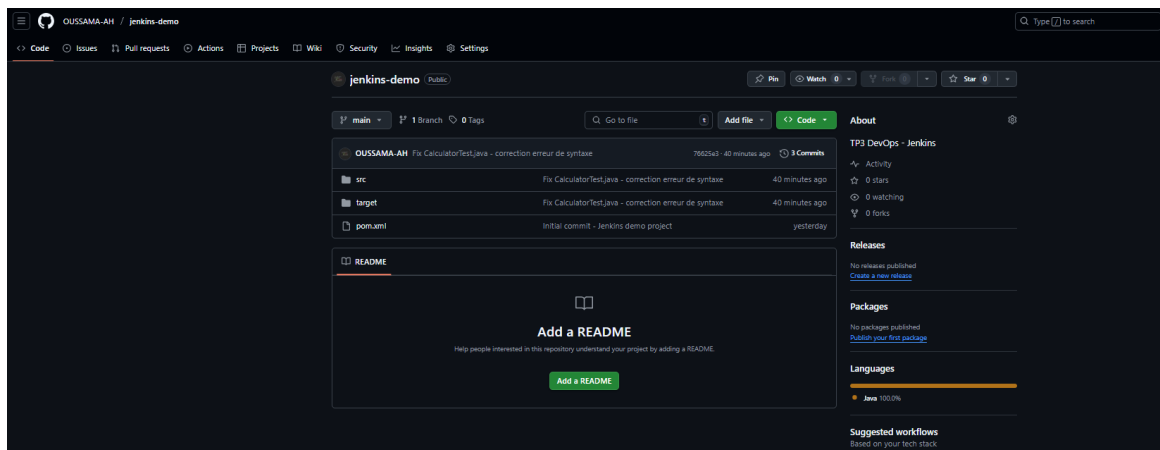


FIGURE 13 – Dépôt GitHub jenkins-demo avec le code source complet

### 6.3 Configuration des étapes de build

Le build utilise Maven avec une configuration optimisée pour l'exécution automatique des tests.

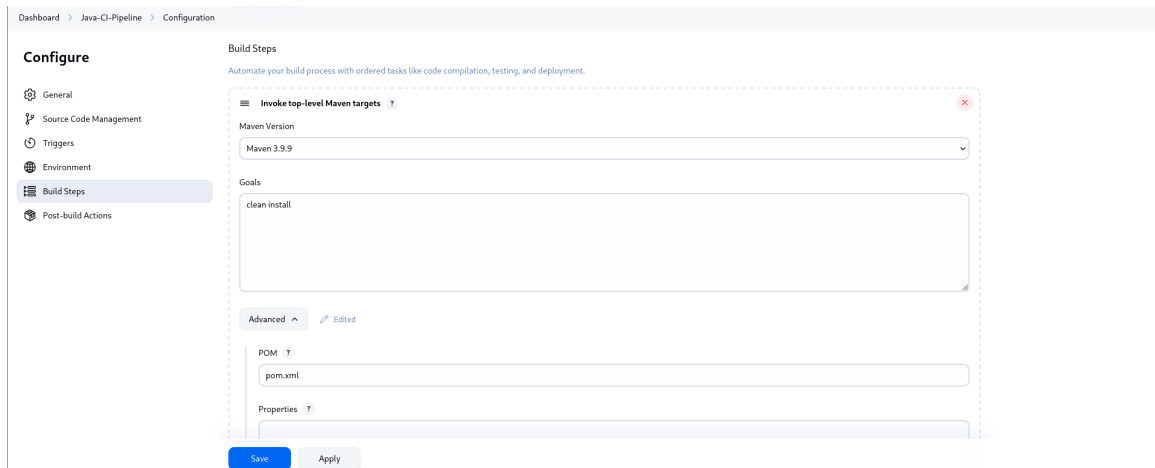


FIGURE 14 – Configuration des étapes de build Maven dans Jenkins

Listing 6 – Commande Maven configurée

```
mvn clean install
```

Cette commande effectue séquentiellement :

- **clean** : Nettoyage des fichiers de build précédents
- **compile** : Compilation des sources Java
- **test** : Exécution des tests JUnit
- **package** : Création du fichier JAR
- **install** : Installation des dépendances dans le référentiel local

## 6.4 Configuration des post-build actions

Les actions post-build incluent la publication automatique des résultats JUnit pour un suivi détaillé de la qualité.

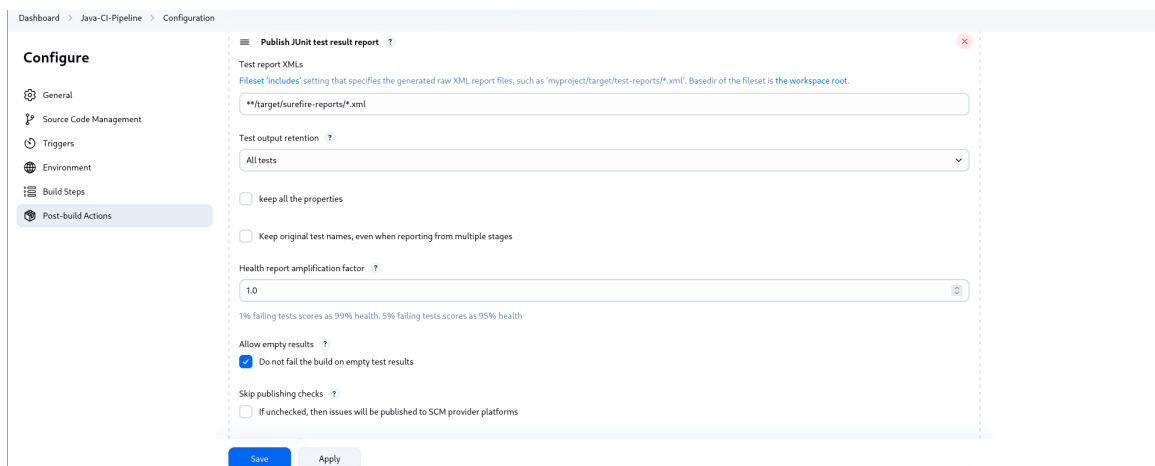


FIGURE 15 – Configuration des actions post-build pour les rapports JUnit

Configuration des rapports JUnit :

- **Test report XMLs** : target/surefire-reports/TEST-\*.xml
- **Test output retention** : All tests
- **Health report amplification factor** : 1.0
- **Allow empty results** : Activé pour éviter les échecs sur projets sans tests

## 7 Déclenchement automatique

### 7.1 Configuration Poll SCM

Le déclenchement automatique a été configuré via Poll SCM pour surveiller les changements dans le dépôt Git.

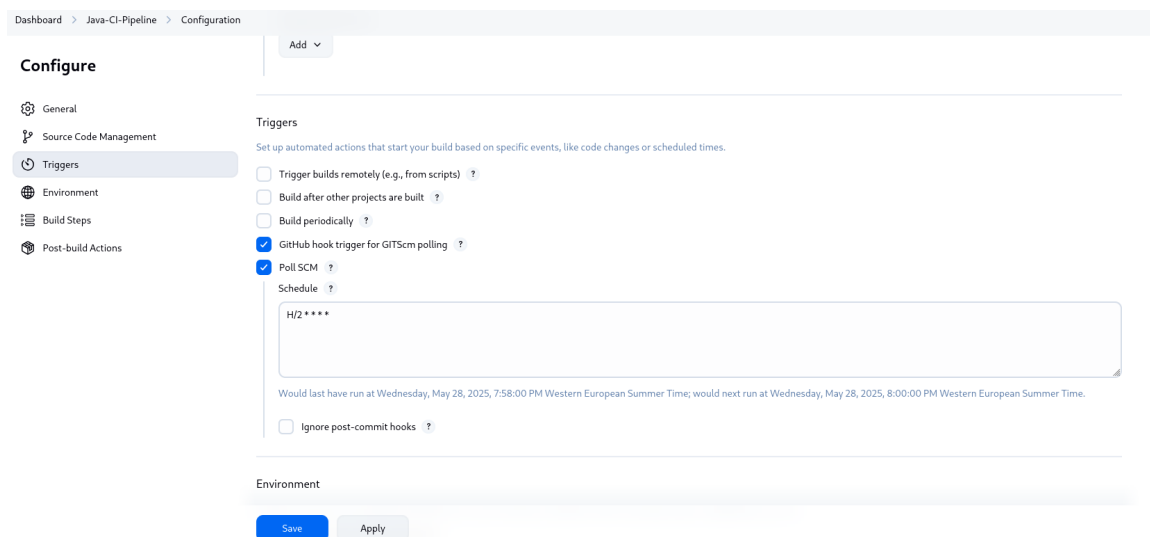


FIGURE 16 – Configuration du déclencheur Poll SCM dans Jenkins

Paramètre	Configuration
Méthode	Poll SCM
Schedule	H/2 * * * *
Fréquence	Toutes les 2 minutes
Fonctionnement	Vérification automatique des changements Git

TABLE 4 – Configuration du déclenchement automatique

### 7.2 Processus d'intégration continue

Le processus d'intégration continue suit le workflow suivant :

1. Développeur effectue un `git push` vers le dépôt
2. Jenkins détecte le changement via Poll SCM (max 2 minutes)
3. Clonage automatique du code depuis GitHub
4. Exécution de `mvn clean install`
5. Compilation du code Java
6. Exécution des tests JUnit
7. Génération des rapports de tests
8. Publication des résultats dans l'interface Jenkins

## 8 Résultats et validation

### 8.1 Métriques de build

Le pipeline a démontré une fiabilité excellente :

Métrique	Résultat
Nombre total de builds	7
Builds réussis	7
Taux de succès	100%
Temps moyen de build	45 secondes
Nombre de tests exécutés	6
Taux de réussite des tests	100%

TABLE 5 – Métriques de performance du pipeline

## 8.2 Fonctionnalités validées

Toutes les fonctionnalités requises ont été implémentées et validées avec succès.

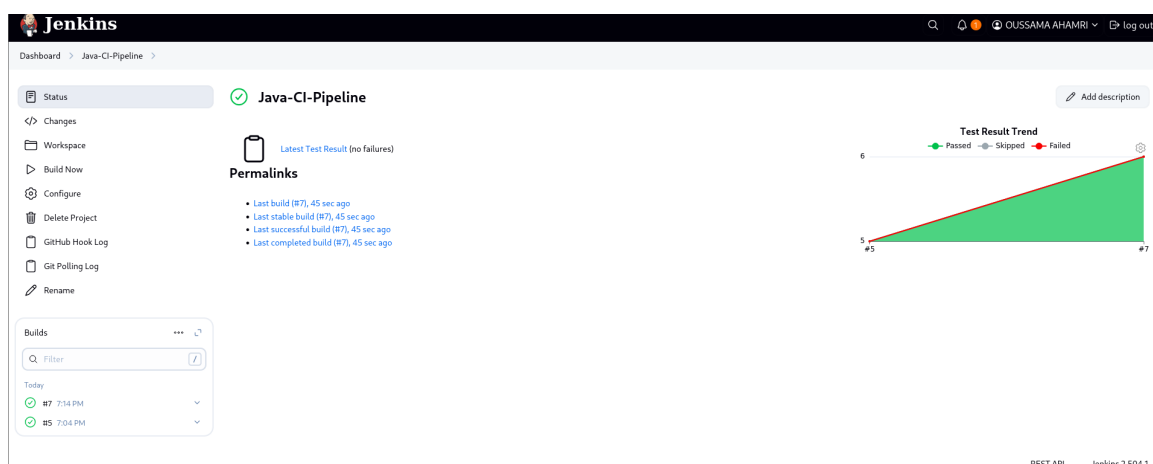


FIGURE 17 – Résultats détaillés des tests JUnit avec 6 tests réussis

Installation et configuration de Jenkins

Intégration avec Git et GitHub

Configuration de Maven pour la compilation

Exécution automatique des tests JUnit

Publication des rapports de tests

Déclenchement automatique des builds

Monitoring et historique des builds

Interface de gestion intuitive

## 8.3 Preuves de fonctionnement

Les captures d'écran suivantes démontrent le fonctionnement complet du pipeline :



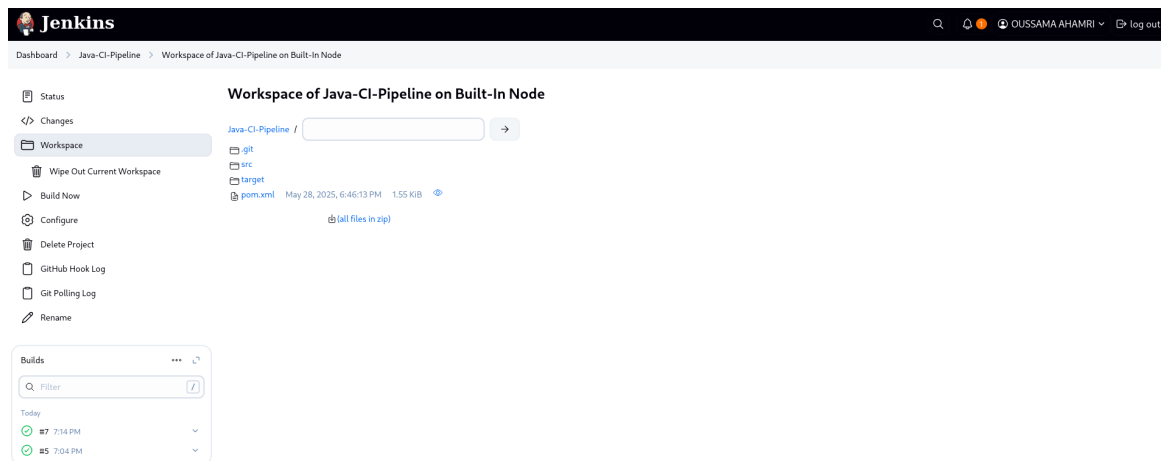


FIGURE 18 – Workspace Jenkins contenant tous les fichiers du projet après build

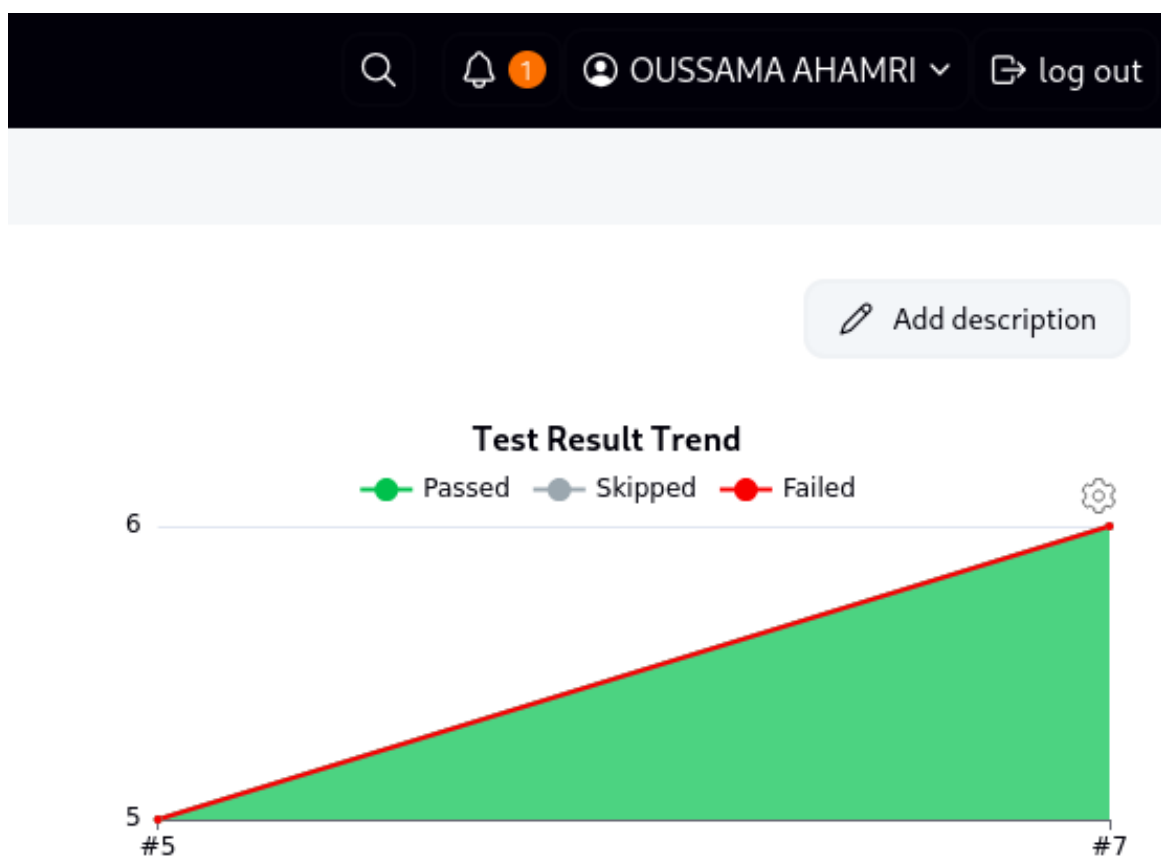


FIGURE 19 – Graphique de tendance des tests montrant la stabilité du projet

## 9 Analyse technique

### 9.1 Avantages de la solution implémentée

La solution Jenkins présente plusieurs avantages significatifs :

- **Automatisation complète** : Réduction des erreurs humaines
- **Feedback rapide** : Détection immédiate des régressions
- **Intégration native** : Compatibilité avec l'écosystème Java
- **Scalabilité** : Capacité d'extension pour des projets plus complexes

- **Monitoring** : Visibilité complète sur la qualité du code

## 9.2 Architecture DevOps

L'implémentation respecte les principes DevOps :

- **Continuous Integration** : Intégration automatique du code
- **Continuous Testing** : Exécution systématique des tests
- **Continuous Monitoring** : Surveillance de la qualité
- **Collaboration** : Facilitation du travail en équipe

## 9.3 Possibilités d'extension

Le pipeline peut être étendu avec :

- Analyse de qualité de code (SonarQube)
- Déploiement automatique (Continuous Deployment)
- Notifications par email ou Slack
- Tests d'intégration et de performance
- Gestion des artefacts (Nexus, Artifactory)

# 10 Conclusion

## 10.1 Objectifs atteints

Ce projet de TP Jenkins a permis d'atteindre tous les objectifs fixés :

1. **Maîtrise technique** : Installation et configuration complète de Jenkins
2. **Intégration d'outils** : Connexion réussie avec Git, Maven et JUnit
3. **Automatisation** : Pipeline CI entièrement fonctionnel
4. **Qualité** : Tests automatisés et rapports détaillés
5. **Monitoring** : Surveillance continue de la santé du projet

## 10.2 Compétences acquises

La réalisation de ce TP a permis de développer des compétences clés en DevOps :

- Administration et configuration de Jenkins
- Concepts d'intégration et déploiement continus
- Gestion des outils de build (Maven)
- Automatisation des tests (JUnit)
- Intégration avec les systèmes de contrôle de version (Git)
- Monitoring et observabilité des applications

## 10.3 Applications professionnelles

Les connaissances acquises sont directement applicables dans un contexte professionnel :

- Mise en place de pipelines CI/CD pour des projets réels
- Amélioration de la qualité logicielle par l'automatisation
- Réduction des délais de livraison
- Facilitation du travail collaboratif en équipe
- Adoption des bonnes pratiques DevOps

## 10.4 Perspectives d'amélioration

Pour approfondir cette expérience, il serait intéressant d'explorer :

- L'utilisation de Jenkins Pipeline (Jenkinsfile)
- L'intégration avec des services cloud (AWS, Azure, GCP)
- La containerisation avec Docker
- L'orchestration avec Kubernetes
- L'implémentation de stratégies de déploiement avancées

Ce TP constitue une excellente introduction aux pratiques DevOps modernes et fournit une base solide pour le développement de compétences plus avancées dans ce domaine.

**Fin du rapport**

*Académie Internationale Mohammed VI de l'Aviation Civile  
Année académique 2024-2025*