École Mohammadia d'Ingénieurs

Génie Logiciel / DevOps

TP n°1 : Installation de l'environnement et des outils DevOps

---

Atelier DevOps – Mise en place d'un pipeline CI/CD multi-projets

---

Élèves :
Sami FAOUZI

Encadré par :
Asmaa Retbi

2ème année Génie Informatique

17 novembre 2025

Table des matières

1 Introduction

Ce TP s'inscrit dans le cadre du module **Génie Logiciel / DevOps** et a pour objectif de mettre en place un environnement technique commun pour plusieurs technologies (**Java / Spring Boot**, **Node.js** et **.NET**) et de préparer la création d'un pipeline d'intégration continue (**CI**) avec **Jenkins**.

Le travail demandé porte principalement sur :

— l'identification des outils nécessaires pour chaque technologie ;
— l'installation et la vérification des versions de ces outils ;
— l'installation et la configuration de Jenkins et de ses plugins ;
— la création d'un dossier de travail contenant trois projets (Java, Node.js, .NET) ;
— la vérification de la compatibilité des versions dans un contexte multi-technologies.

2 Outils retenus pour l'atelier

2.1 Tableau récapitulatif des technologies et outils

Technologie	Outil principal	Version recommandée
Java / Spring Boot	Maven	3.9.x
Node.js	npm	2.9.x
.NET	dotnet SDK	8.0.x
Environnement d'intégration	GitLab CI/CD	Dernière version
Intégration continue	Jenkins	2.4.xx / 2.5.xx
Conteneurisation	Docker (mentionné)	(non utilisé dans ce TP)

TABLE 1 – Outils retenus pour l'atelier CI/CD

2.2 Pré requis par technologie (scope du TP)

Élément	Java Spring Boot	Node.js	.NET
Gestionnaire de dépendances	Maven	npm	NuGet (via `dotnet`)
Outils nécessaires / SDK	JDK 17+, Maven, IDE (IntelliJ/Eclipse), Git	Node.js, npm, éditeur (VS Code), Git	.NET SDK 8.x, IDE (Visual Studio / VS Code), Git
Variables d'environnement à vérifier	`JAVA_HOME`, `java` et `mvn` dans le `PATH`	`node` et `npm` dans le `PATH`	`dotnet` dans le `PATH`

TABLE 2 – Pré requis par technologie pour les trois projets du TP

3 Préparation de l'environnement

3.1 Installation des outils

Les consignes (diapo « Préparation de l'environnement ») demandent :

1. d'installer les outils identifiés dans le tableau précédent ;
2. de vérifier les versions installées avec :
   - `git -version`
   - `java -version`
   - `mvn -v`
   - `node -v`
   - `npm -v`
   - `dotnet -version`
3. de prendre des captures d'écran de ces commandes ;
4. de mentionner le système d'exploitation et l'architecture de la machine.

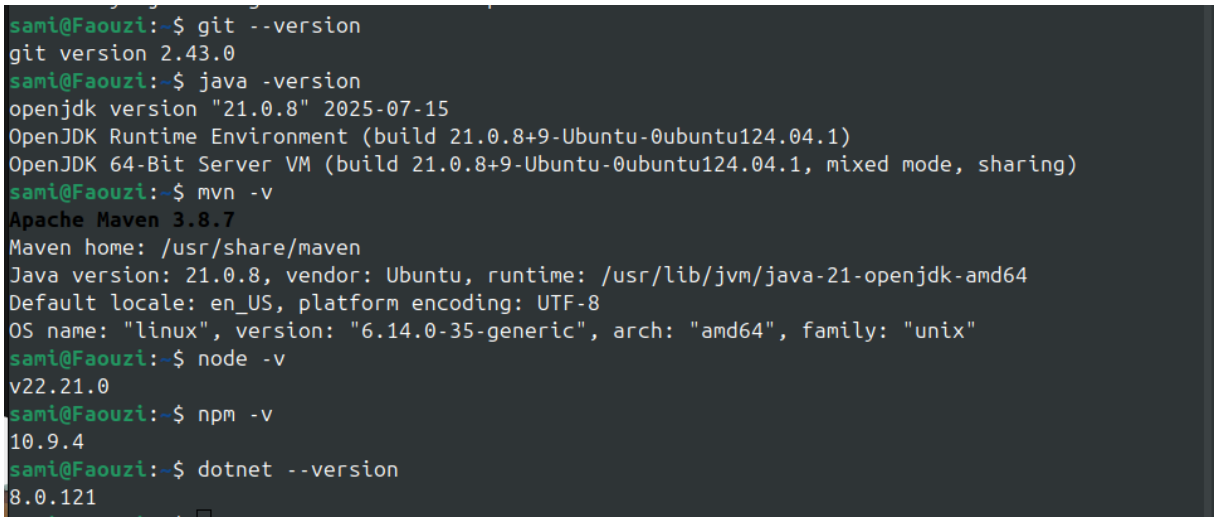

FIGURE 1 – Versions des principaux outils installés (Git, Java, Maven, Node.js, npm, .NET).

4 Installation et configuration de Jenkins

4.1 Installation de Jenkins

Étapes demandées (diapo « Préparation de l'environnement – Jenkins ») :

1. Télécharger Jenkins depuis `https://www.jenkins.io/download/`.
2. Installer la version adaptée au système (Windows / Linux).
3. Choisir l'exécution en tant que « LocalSystem » (ou équivalent).

4. Démarrer Jenkins et accéder à `http://localhost:8080`.
5. Déverrouiller Jenkins avec la clé du fichier `initialAdminPassword`.

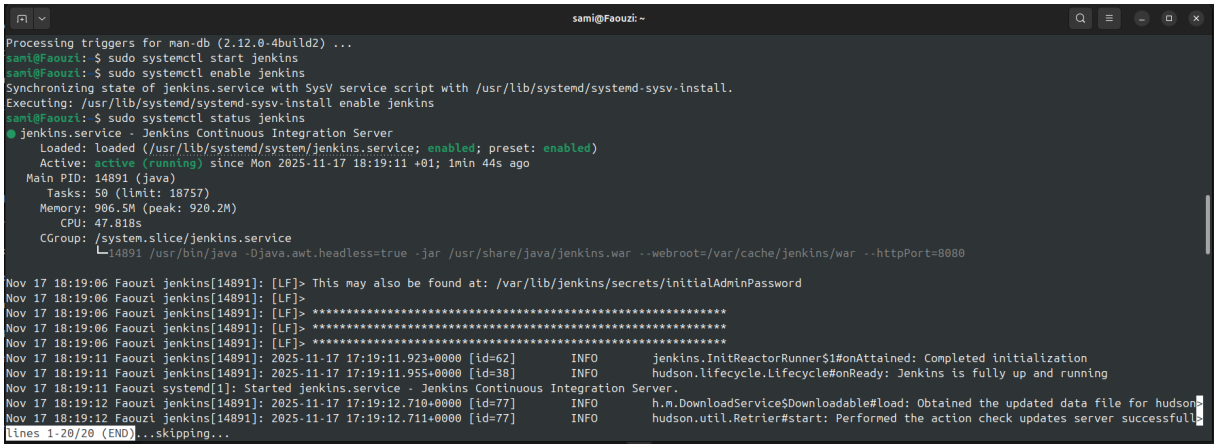

FIGURE 2 – Service Jenkins actif sur la machine (`systemctl status jenkins`).

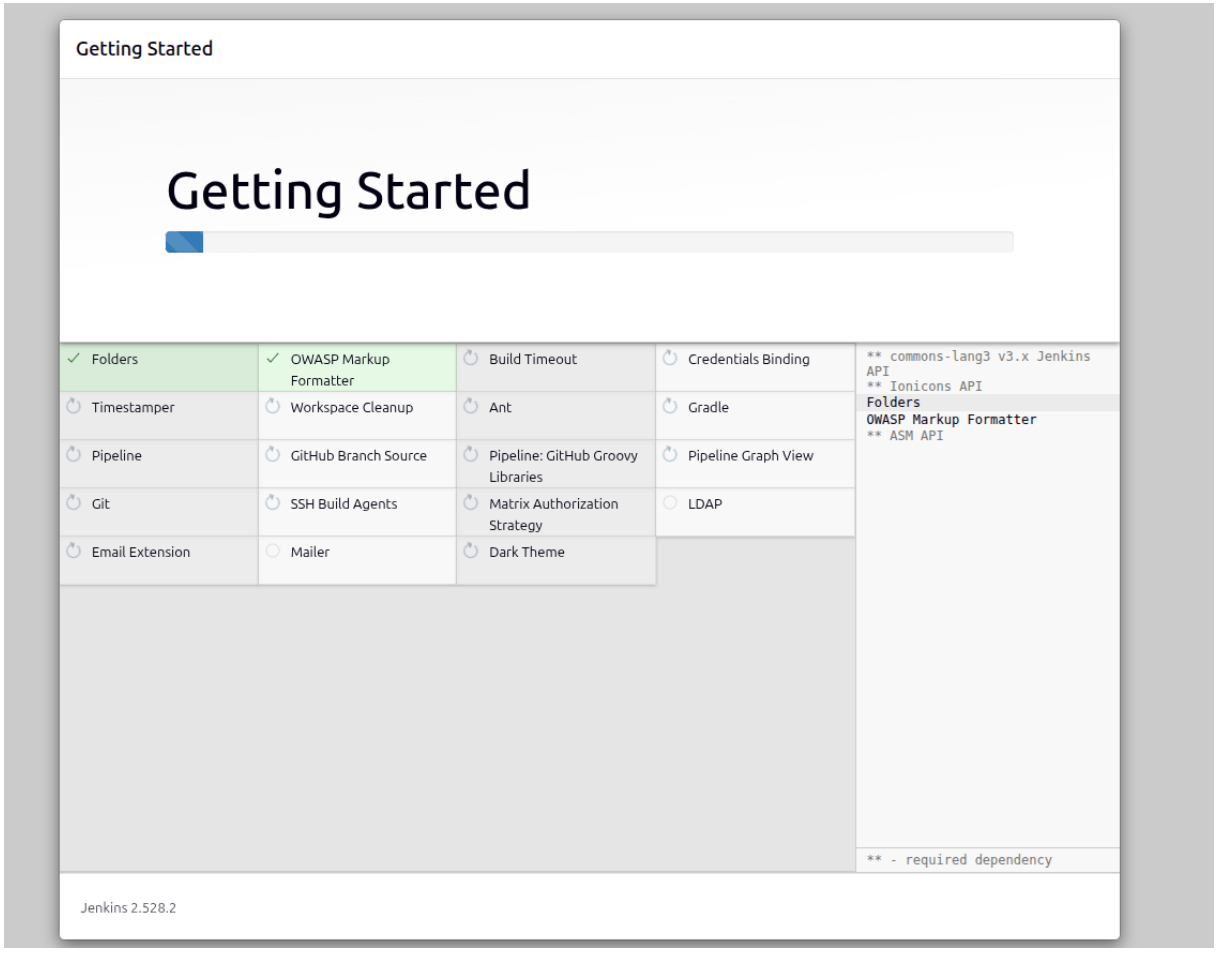

FIGURE 3 – Installation des plugins .

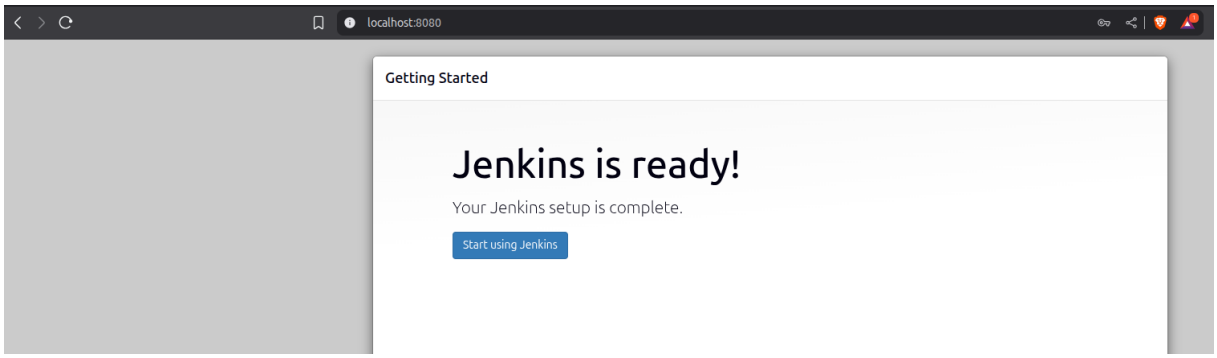

FIGURE 4 – Jenkins Ready.

Captures attendues

— écran de déverrouillage (champ « Administrator password ») ;
— écran d'installation des plugins ;
— écran final « Jenkins is ready ! ».

4.2 Installation des plugins Jenkins

Plugins à installer (diapo « Jenkins – Installation des plugins complémentaires ») :
— Maven Integration ;
— NodeJS ;
— .NET SDK / MSBuild ;
— Docker Pipeline (pour les prochaines séances).
Étapes :

1. Ouvrir **Manage Jenkins → Manage Plugins**.
2. Rechercher et installer les plugins.
3. Redémarrer Jenkins si nécessaire.

5 Création du dossier de travail et des projets

5.1 Création du dossier local

On crée un dossier de travail centralisé, comme indiqué dans la diapo « Création des projets – Création du Dossier de Travail (en local) » :

```
devops-workspace/
    springboot-app/
    nodejs-app/
    dotnet-app/
```

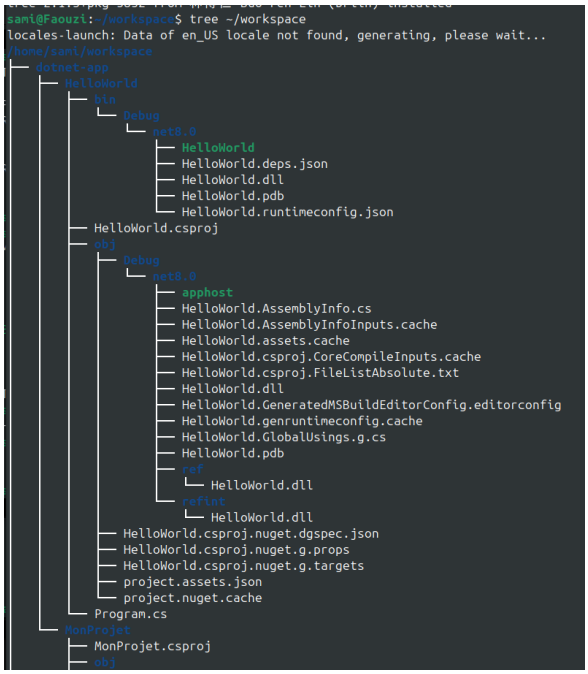

FIGURE 5 – Structure du dossier workspace (image 1).

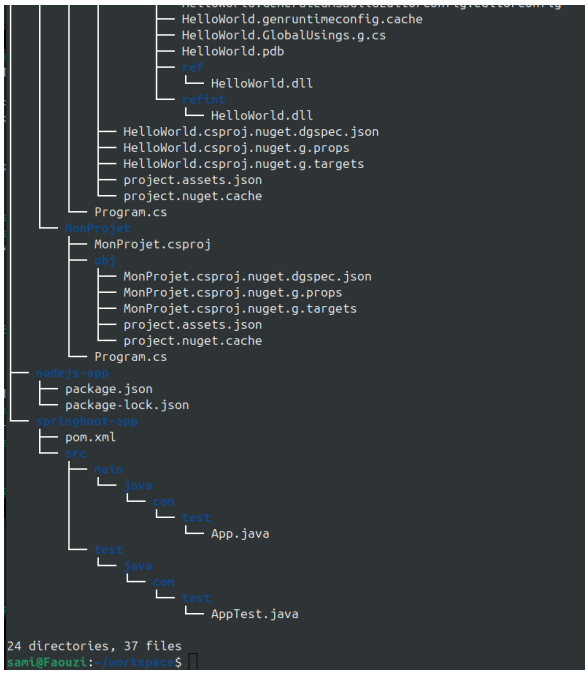

FIGURE 6 – Structure du dossier workspace (image 2).

Capture attendue

— capture de l'explorateur ou du terminal montrant cette structure.

5.2 Commandes de base par technologie

Java / Spring Boot avec Maven

Structure : `mvn <objectif> [options]`

Action	Commande Maven
Créer un projet	`mvn archetype:generate`
Compiler	`mvn compile`
Tester	`mvn test`
Packager (JAR)	`mvn package`
Nettoyer	`mvn clean`
Build complet	`mvn clean package -DskipTests`

TABLE 3 – Commandes Maven utilisées dans le TP

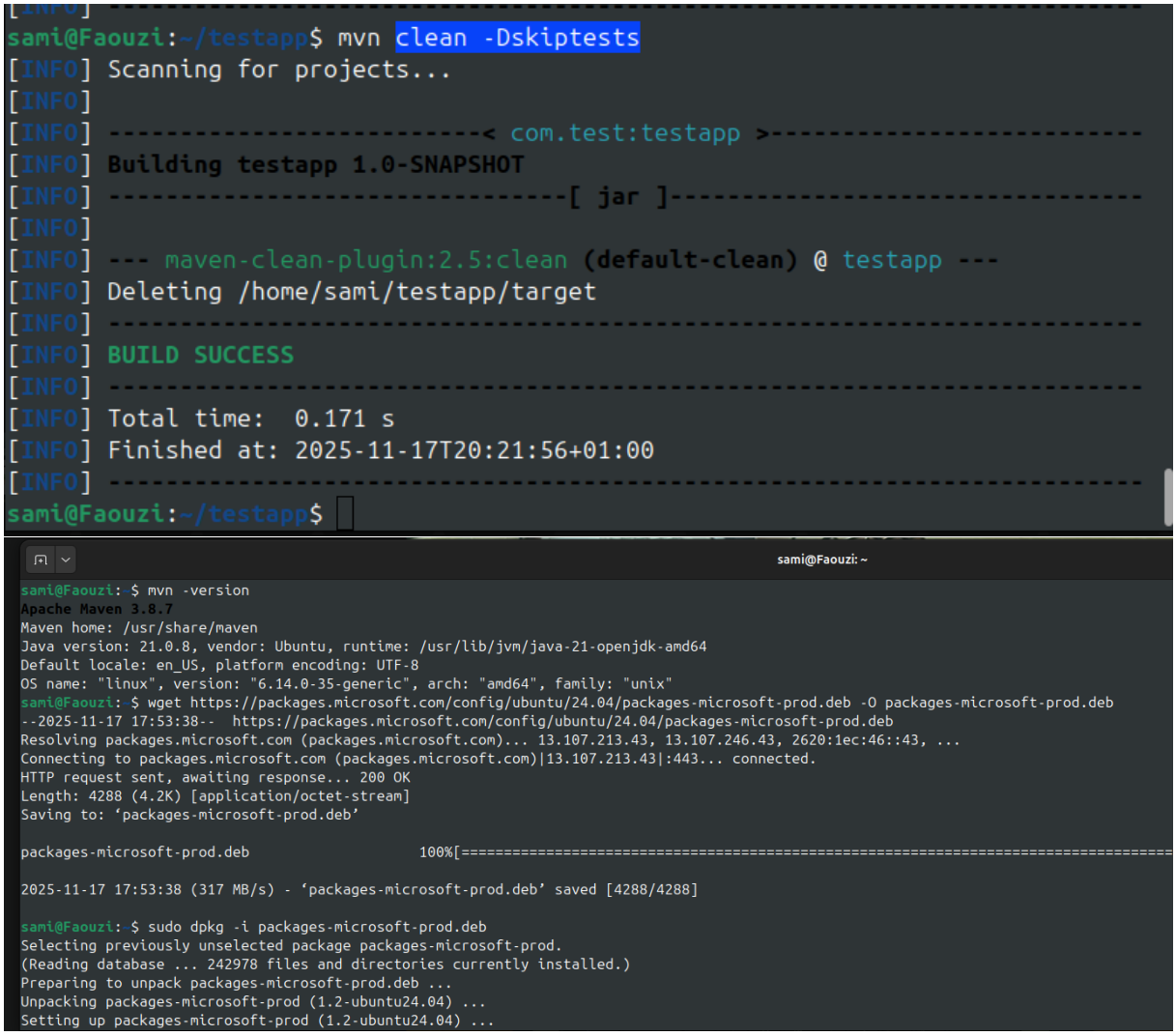

FIGURE 7 – Création du projet Maven (`mvn archetype:generate`) puis `mvn clean compile` et `mvn clean -DskipTests`.

.NET

Structure : `dotnet <commande> [options]`

Action	Commande dotnet
Créer un projet console	`dotnet new console -n MonProjet`
Compiler	`dotnet build`
Exécuter	`dotnet run`
Tester	`dotnet test`

TABLE 4 – Commandes .NET utilisées dans le TP

```
sami@Faouzi:~/testapp$ dotnet new console -n HelloWorld
The template "Console App" was created successfully.

Processing post-creation actions...
Restoring /home/sami/testapp/HelloWorld/HelloWorld.csproj:
  Determining projects to restore...
  Restored /home/sami/testapp/HelloWorld/HelloWorld.csproj (in 177 ms).
Restore succeeded.


sami@Faouzi:~/testapp$ cd HelloWorld
sami@Faouzi:~/testapp/HelloWorld$ dotnet build
MSBuild version 17.8.43+f0cbb1397 for .NET
  Determining projects to restore...
  All projects are up-to-date for restore.
  HelloWorld -> /home/sami/testapp/HelloWorld/bin/Debug/net8.0/HelloWorld.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:03.12
sami@Faouzi:~/testapp/HelloWorld$ dotnet run
Hello, World!
sami@Faouzi:~/testapp/HelloWorld$ dotnet test
  Determining projects to restore...
  All projects are up-to-date for restore.
sami@Faouzi:~/testapp/HelloWorld$
```

FIGURE 8 – Exécution des commandes `dotnet new`, `dotnet build`, `dotnet run` et `dotnet test` sur le projet `HelloWorld`.

Node.js avec npm

Structure : `npm <commande> [arguments]`

Action	Commande npm
Initialiser un projet	`npm init`
Installer les dépendances	`npm install` ou `npm ci`
Exécuter le script de build	`npm run build`
Exécuter le projet	`npm start`

TABLE 5 – Commandes npm utilisées dans le TP

```
sami@Faouzi:~/testapp$ npm run build

> testapp@1.0.0 build
> echo Building...

Building...
```

FIGURE 9 – Exemple d'exécution de `npm run build` sur le projet Node.js.

```
sami@Faouzi:~/testapp$ echo "console.log('Hello from Node.js!');" > index.js
bash: !': event not found
sami@Faouzi:~/testapp$ nano package.json
sami@Faouzi:~/testapp$ npm start

> testapp@1.0.0 start
> node index.js

node:internal/modules/cjs/loader:1386
  throw err;
  ^

Error: Cannot find module '/home/sami/testapp/index.js'
    at Function._resolveFilename (node:internal/modules/cjs/loader:1383:15)
    at defaultResolveImpl (node:internal/modules/cjs/loader:1025:19)
    at resolveForCJSWithHooks (node:internal/modules/cjs/loader:1030:22)
    at Function._load (node:internal/modules/cjs/loader:1192:37)
    at TracingChannel.traceSync (node:diagnostics_channel:328:14)
    at wrapModuleLoad (node:internal/modules/cjs/loader:237:24)
    at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_
in:171:5)
    at node:internal/main/run_main_module:36:49 {
  code: 'MODULE_NOT_FOUND',
  requireStack: []
}

Node.js v22.21.0
```

FIGURE 10 – Exemple d'exécution de `npm start` sur le projet Node.js.

6 Analyse des structures des projets

Cette partie répond à la consigne de la diapo « Analyse des structures ». Elle vise à identifier les fichiers de configuration, expliquer leur rôle et préciser pourquoi ils sont

essentiels dans un pipeline CI/CD.

Projet	Fichier clé	Rôle principal	Exemples d'informations contenues	Importance dans le pipeline CI/CD
Spring Boot (Java)	`pom.xml`	Définir les dépendances et plugins Maven	Dependencies, version Java, plugins, packaging, phases de build	Permet à Jenkins d'exécuter automatiquement `mvn package`, gérer tests, build et artefacts.
Node.js	`package.json`	Définir le projet Node et ses scripts	Nom, version, scripts, dépendances, commandes `npm run`	Permet à Jenkins de lancer `npm install`, `npm test`, `npm run build` sans configuration manuelle.
.NET	`.csproj`	Définir le projet .NET, framework et dépendances	Framework cible, références NuGet, configuration de build	Permet à Jenkins d'exécuter automatiquement `dotnet restore`, `dotnet build`, `dotnet test`.

TABLE 6 – Analyse des fichiers de configuration pour les trois projets

Pourquoi ces fichiers sont essentiels en CI/CD ?

— Ils définissent **toutes les dépendances et versions nécessaires** à l'exécution du projet.
— Ils permettent au pipeline de **reproduire le build automatiquement et de façon déterministe**.
— Ils servent de **source de vérité** : si le fichier est versionné dans Git, le pipeline peut reconstruire l'application sans manipulation manuelle.
— Ils permettent d'automatiser les étapes clés : installation, compilation, tests, packaging, analyse de qualité.

7 Vérification des compatibilités

7.1 Tableau de synthèse

Outil	Commande utilisée	Version installée	Compatible avec ?	Observation / Remarque
Java	`java -version`	21.0.8	Maven, Jenkins (plugin Maven)	Version LTS recommandée par la doc Jenkins
Maven	`mvn -v`	Apache Maven 3.8.7	Java, Jenkins	Vérifier la compatibilité avec le JDK installé
Node.js	`node -v`	v22.21.0	npm, plugin NodeJS Jenkins	S'assurer que la version est supportée par le plugin NodeJS
npm	`npm -v`	10.9.4	Node.js	Version fournie avec Node.js
.NET SDK	`dotnet -version`	8.0.121	plugin .NET / MSBuild	Version 8.x comme recommandée dans le TP

TABLE 7 – Vérification de la compatibilité des versions des outils

Dans les observations, on commente par exemple :

— si la version est LTS ou non ;

— si elle correspond bien à la version recommandée dans les diapositives ;

— si des ajustements ont été nécessaires (mise à jour, changement de plugin, etc.).

8 Conclusion

Ce TP a permis de :

— configurer un environnement de développement multi-technologies (Java, Node.js, .NET) ;

— installer et vérifier les principaux outils de build (Maven, npm, dotnet) ;

— installer et configurer Jenkins ainsi que les plugins nécessaires ;

— structurer un dossier de travail contenant trois projets prêts à être intégrés dans un pipeline CI/CD ;

— vérifier la compatibilité des versions des outils dans une perspective DevOps.

Ce travail constitue la base pour les séances suivantes, qui auront pour objectif la mise en place concrète d'un pipeline d'intégration continue et de déploiement continu pour ces trois projets.