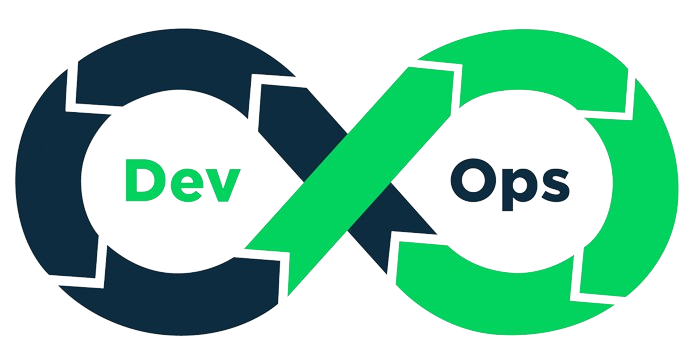
Technologies et Solutions d’un Environnement de Développement Offshore



Ingénierie DevOps & Outils d’Industrialisation

Réalisé par : Encadré par :

ZAZOULI Manal Pr. CHENFOUR Noureddine

MIKH Houssam

MOUSMI Hiba

AMARTI RIFFI El Mehdi

ELGHAYATE Marouane

Table des matières

[Table des matières 2](#_Toc165828158)

[Table des figures 3](#_Toc165828159)

[Chapitre 1 : Introduction 4](#_Toc165828160)

[Chapitre 2 : 4](#_Toc165828161)

[I. Phase du code : 4](#_Toc165828162)

[1. Introduction: 4](#_Toc165828163)

[2. Architecture GIT: 4](#_Toc165828164)

[3. Les commandes Git: 5](#_Toc165828165)

[4. Pratique : 6](#_Toc165828166)

[II. Phase de Build : 9](#_Toc165828167)

[Introduction : 9](#_Toc165828168)

[C’est quoi le build ? 9](#_Toc165828169)

[Outils de Build : 10](#_Toc165828170)

Table des figures

[Figure 1 – Structure générale d’un répertoire Git 5](#_Toc165815441)

Chapitre 1 : Introduction

Chapitre 2 :

1. Phase du code :

# Introduction:

La phase du code dans le cycle **DevOps** représente le moment où le développement logiciel prend vie. Cela commence dès le lancement du projet (après la conception).

Dans cette phase, plusieurs collaborateurs tentent de travailler simultanément sur le même projet. Traditionnellement, cela posait un problème majeur car l'équipe se retrouvait souvent confrontée à des conflits. C'est là qu'un système de contrôle de version est mis en place. Dans ce rapport, nous allons nous concentrer sur Git, qui est un système de contrôle de version distribué, offrant principalement deux fonctionnalités principales : le travail collaboratif et le contrôle des versions.

Pour la première fonctionnalité, le travail collaboratif : Git offre à l'équipe une grande flexibilité pour travailler chacun sur son propre poste ou sa propre machine tout en collaborant simultanément sur le même projet. Il suffit de déposer le projet sur un serveur d'hébergement tel que GitHub, puis les membres peuvent récupérer le projet distant sur leur machine locale, y travailler, enregistrer les modifications localement et les renvoyer vers le projet distant pour qu'elles soient accessibles par les autres membres. Nous explorerons davantage ces détails dans les parties suivantes de ce chapitre.

En plus du travail collaboratif, Git permet également aux développeurs de contrôler les versions de leur projet afin de faciliter le retour en arrière en cas de fausse modification, entre autres avantages...

# Architecture GIT:

L’architecture git est composée de trois stade comme représente la figure suivante :

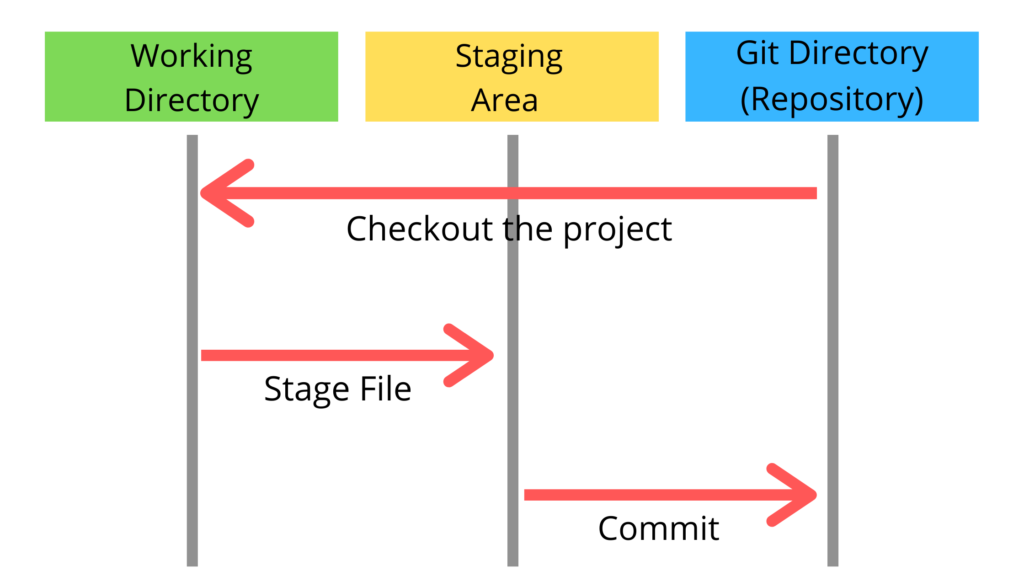
****

Figure 1 – Structure générale d’un répertoire Git

**Working Directory** : C'est le répertoire dans lequel vous travaillez, où vous effectuez des modifications sur votre projet. En effet, il s'agit de votre espace de travail principal.

**Staging Area** : Cette partie est gérée par Git. Chaque modification apportée à votre projet (ajout, suppression, modification du contenu d'un fichier, etc.) est enregistrée dans cette zone. Elle sert de zone de préparation pour les modifications avant leur enregistrement définitif dans le dépôt.

**Repository** : C'est votre dépôt qui contient votre projet ainsi que les modifications apportées, ainsi que des fichiers de journalisation pour suivre les versions antérieures. En d'autres termes, c'est l'endroit où votre code est stocké et versionné.

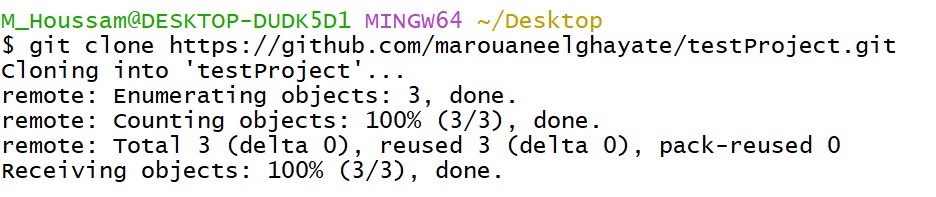
Lorsque vous enregistrez une modification de façon définitive dans le dépôt local, vous effectuez ce qu'on appelle un commit. Dans le langage de Git, un commit consiste à capturer l'état actuel du projet sous forme d'une instance, appelée snapshot. Ces snapshots, regroupés par Git, représentent l'évolution de votre projet et sont organisés en branches. Une branche est un pointeur léger qui indique un commit spécifique, lequel pointe à son tour vers le commit précédent.

# Les commandes Git:

* **Git** **init** : permet d’initialiser un depot locale.
* **Git** **clone** : permet de cloner un dépôt distant dans votre machine locale.
* **Git** **add** : permet d’ajouter les modifications des fichiers dans « staging area » pour préparer le prochain commit .
* **Git** **commit** : enregistre les modifications ajouter à « staging area » dans le dépôt.
* **Git** **push** : envoie les commits locaux vers un référentiel distant.
* **Git** **pull** : récupére les commits d’un référentiel distant.
* **Git** **branch** : pour lister, créer ou supprimer des branches.
* **Git** **merge** : pour fusionner les modifications d’autre branche dans une branche.
* **Git** **checkout** : créer est se dépacer dans la nouvelle branche.
* **Git** **status** : affiche l’état actuel de votre fichier dans le dépôt.

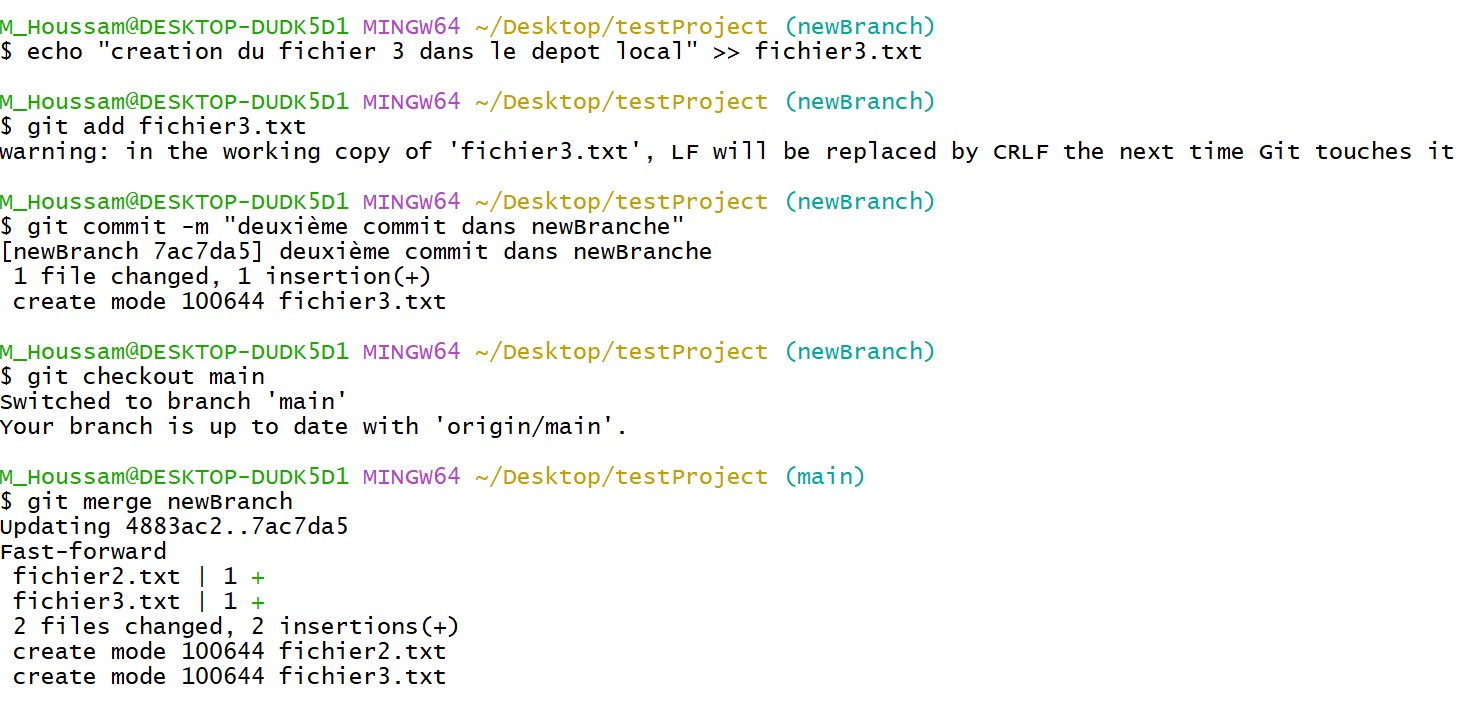
# Pratique :

Dans cette pratique, nous allons travailler sur un projet en collaboration avec d'autres membres de l'équipe. Ce projet est déposé sur GitHub, alors la première étape consiste à cloner ce projet à partir de GitHub.

* **git clone url**

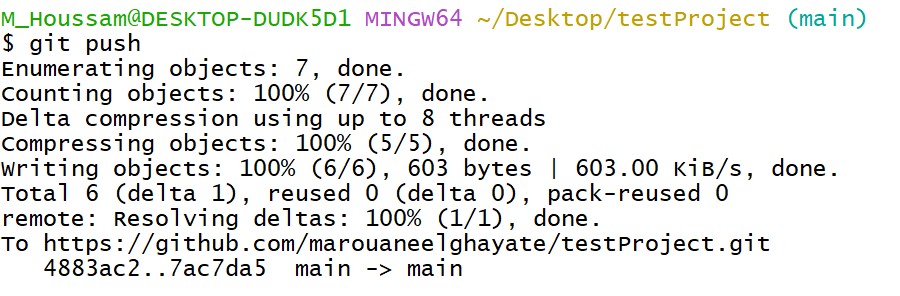
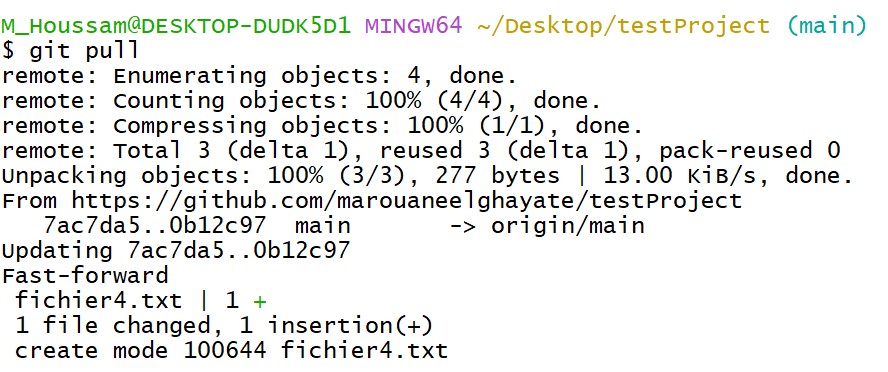
La deuxième étape consiste à créer une autre branche, y apporter mes modifications localement, puis à enregistrer ces modifications dans la "staging area" avant de prendre un "snapshot" en effectuant un commit:

* **git checkout –b newBranch**
* **git add**
* **git commit –m ^message de commit ^**

 Après avoir déplacé vers la branche principale "main", nous avons fusionné les commits de la branche "newBranch" avec la branche principale "main" :

* **git checkout main**
* **git merge newBranch**

Pour envoyer ces modifications vers le dépôt distant, nous effectuons un "git push", puis chaque membre de l'équipe est invité à effectuer un "git pull" pour rapporter les nouvelles modifications.

* **git push**
* **git pull**

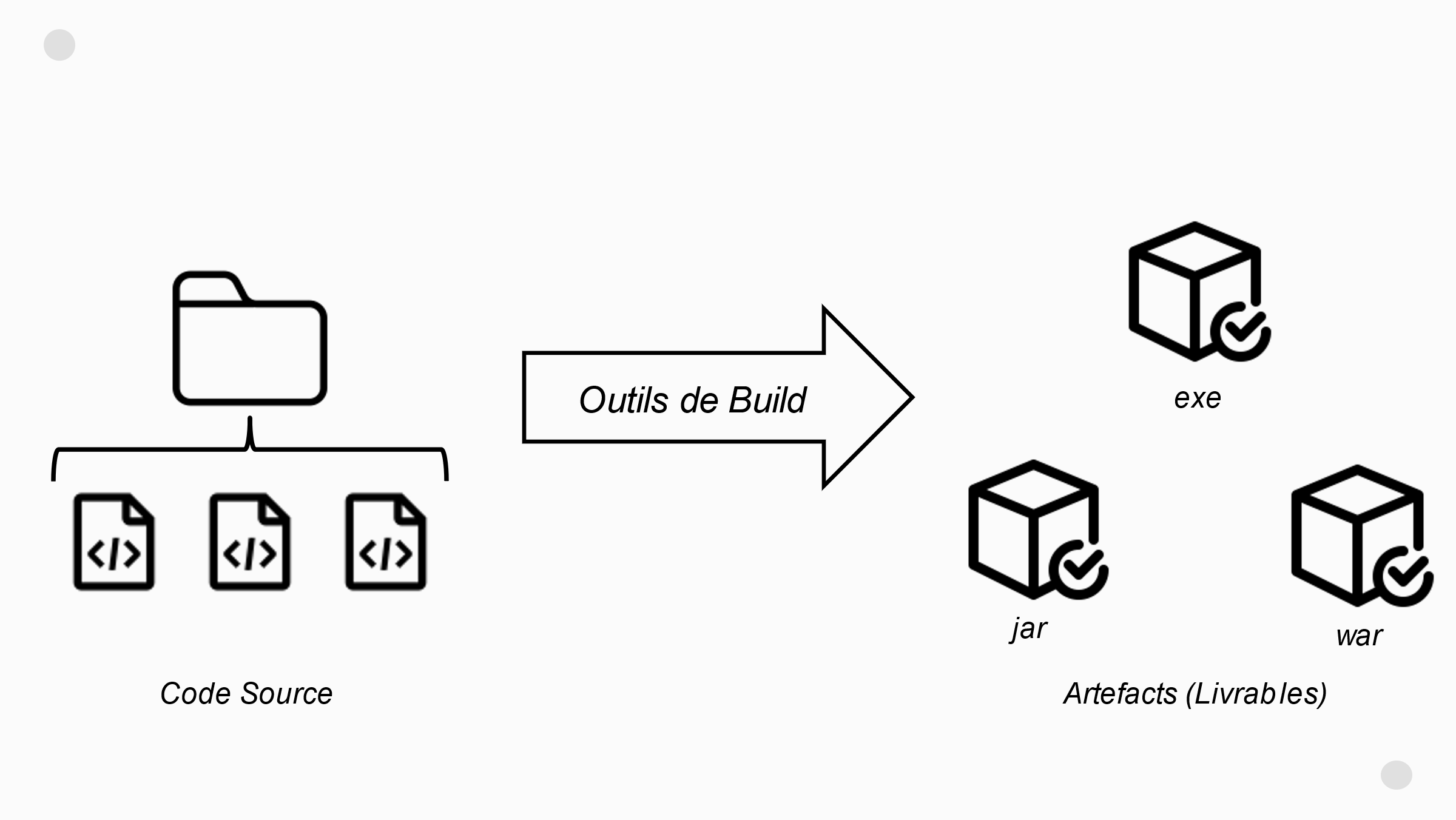
1. Phase de Build :

Introduction :

Au cœur de la méthodologie **DevOps**, le processus de **"build"** incarne l'essence même de l'automatisation et de l'efficacité dans le développement logiciel. Il représente un élément crucial de la chaîne d'outils DevOps, catalysant la transformation du code source en artefacts déployables de manière cohérente et fiable. Cette introduction vise à plonger dans les rouages du "build" dans le contexte spécifique de DevOps, mettant en lumière son rôle pivot dans la réalisation d'une intégration et d'un déploiement continus (CI/CD).

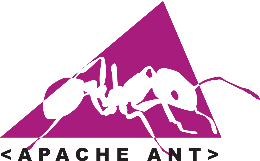
C’est quoi le build ?

La phase de build en DevOps est une étape cruciale du processus de développement logiciel qui vise à transformer le code source en artefacts exécutables et prêts à être déployés. Cette phase comprend généralement plusieurs sous-étapes telles que la compilation du code source, l'exécution des tests automatisés, la gestion des dépendances et la création de packages d'installation.



Outils de Build :

Les outils de build jouent un rôle essentiel dans l'orchestration du processus de développement logiciel. Ils permettent d'automatiser les tâches de construction et de déploiement, contribuant ainsi à accélérer le cycle de développement et à garantir la cohérence des livrables. Parmi les outils les plus populaires, on retrouve Apache Maven, Gradle, Ant,CMake et bien d'autres encore. Chacun de ces outils présente des fonctionnalités spécifiques et des avantages uniques, adaptés à différents besoins et contextes de développement.





Dans le cadre de ce livrable, nous avons décidé d'opter pour Apache Maven comme outil de build privilégié pour notre pratique. Maven est largement reconnu pour sa simplicité d'utilisation, sa gestion efficace des dépendances, et sa capacité à standardiser les cycles de développement. Son approche basée sur des conventions et sa vaste bibliothèque de plugins en font un choix idéal pour automatiser et rationaliser les workflows de construction dans un environnement DevOps.