



# Plateforme MedHead

Application POC pour le traitement des  
recommandations de lits dans les situations  
d'intervention d'urgence

## **Processus CI/CD**

Version 1.0

---

Auteur : Hervé Prevost - Architecte Logiciel - MedHead

Date : 14/09/2023

Version : 1.0

---

# SOMMAIRE

---

SOMMAIRE	2
OBJET ET CONTEXTE	3
CHAÎNE D'INTÉGRATION CI/CD	3
Principes généraux	3
Outils CI/CD utilisés pour l'application POC	4
Mise en place du workflow	4
Fichier de configuration du workflow	7
Vérification du processus CI/CD	8
Vérification dans JUNIT	8
Vérification dans le rapport de test du Workflow	9

## OBJET ET CONTEXTE

---

Ce document a pour but de présenter l'environnement technique, le pipeline CI/CD et le descriptif des tests effectués pour la Proof Of Concept du traitement des recommandations de lits disponibles en situation d'urgence.

## CHAÎNE D'INTÉGRATION CI/CD

---

### Principes généraux

Le processus CI/CD présente les 3 avantages suivants :

- Tester le bon fonctionnement des composants logiciels au fur et à mesure de leur développement et non uniquement lors des périodes de recette. Ceci améliore la qualité du code en accélérant la détection des erreurs et leur correction.
- Augmente la rentabilité des développements en permettant la mise en production et l'exploitation d'une partie des fonctionnalités sans attendre l'achèvement complet du projet.
- Fluidifie et sécurise le travail en équipe en simplifiant par l'automatisation des tests la livraison et le déploiement du code



Le processus d'intégration et déploiement continu mis en place pour la POC va permettre à une l'équipe de développement de :

- partager un code commun à distance et le faire évoluer rapidement au fur et à mesure des développements
- vérifier son fonctionnement sans affecter la version en production
- automatiser les tests d'intégration et de déploiement dès la mise en place d'une nouvelle version

## Outils CI/CD utilisés pour l'application POC

Le code Java a été déposé dans le repository **github** suivant :

<https://github.com/HPrevost/OCProjet11>

La publication du code sur le dépôt distant s'effectue avec **GitBash**.

Tous les tests ont été écrits en Spring Boot test et sont concentrés dans un fichier situé à la racine du dossier test du projet. Chaque modification de code déclenche le script de test et le rapport de test est consultable dans le dépôt Github.

Les tests ont été réalisés avec **JUnit**.

Le workflow d'intégration est décrit dans le fichier **yaml** suivant :

**p11api/github/workflows/maven.yml**, il a également été placé à la racine du projet afin de faciliter sa publication sur le repo github.

## Mise en place du workflow

Le processus de publication d'une version du code et le lancement des tests automatisés est nommé **workflow** sur github

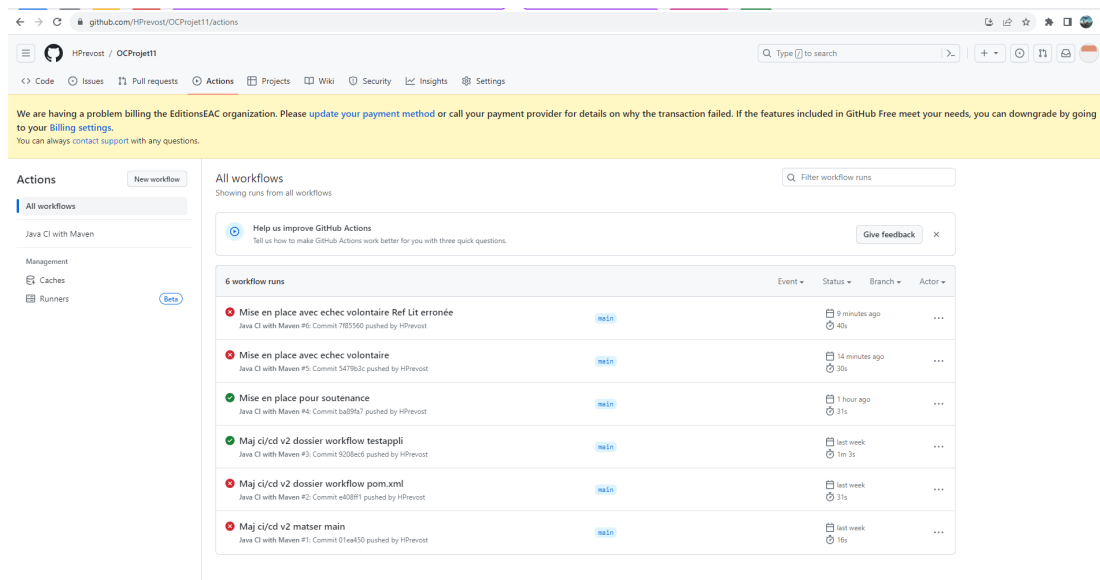
Pour le projet POC il s'effectue de façon suivante.

1. Mise en place du fichier maven.yml contenant la configuration du build de l'exécutable dans le sous dossier local du projet  
.github/workflows/
2. Enregistrement des fichiers modifiés lors du développement du projet à l'aide de l'IDE Eclipse
3. Publication des fichiers modifiés sur le repository github avec l'outil CLI GitBash. La publication devant être libellée pour indiquer le

contenu de la mise à jour et qui correspondra au nom de la tâche de build sur le dépôt github.

4. Dans l'onglet action du repository GitHub, on constate que le push effectué à l'étape précédente se retrouve automatiquement en tête de l'historique des workflows . Un spinner orange indique que le processus est en cours. A l'issue du traitement l'icône devient verte ou rouge en cas de réussite ou d'échec. Pour consulter le détail des logs d'erreur il suffit de cliquer sur le nom du processus.
5. On peut tester le bon fonctionnement de la tâche de build en créant une erreur volontaire dans le code (Error JUnit) ou failure en indiquant une valeur erronée attendue sur un assertEquals (Failure au sens JUnit). Le build conséquent doit échouer.

Voici quelques captures d'écran pour illustrer le travail de build automatisé dans github :



← Java CI with Maven

✖

Mise en place avec echec volontaire Ref Lit erronée #6

Summary

Jobs

✖ build

Run details

Usage

Workflow file

Triggered via push 12 minutes ago

Status

Total duration

Artifacts

HPrevost pushed · 7f85560

main

Failure

40s

–

maven.yml

on: push

✖ build

31s

Annotations

1 error

✖ build

Process completed with exit code 1.

Summary

Jobs

✖ build

Run details

Usage

Workflow file

build

failed 13 minutes ago in 31s

✖ Build patient\_service with Maven

125 at org.apache.maven.surefire.junitplatform.LazyLauncher.execute(LazyLauncher.java:50)

126 at org.apache.maven.surefire.junitplatform.JUnit4Provider.execute(JUnit4Provider.java:184)

127 at org.apache.maven.surefire.junitplatform.JUnit4Provider.invokeAllTests(JUnit4Provider.java:148)

128 at org.apache.maven.surefire.junitplatform.JUnit4Provider.invoke(JUnit4Provider.java:122)

129 at org.apache.maven.surefire.booter.ForkedBooter.runSuitesInProcess(ForkedBooter.java:385)

130 at org.apache.maven.surefire.booter.ForkedBooter.execute(ForkedBooter.java:162)

131 at org.apache.maven.surefire.booter.ForkedBooter.run(ForkedBooter.java:587)

132 at org.apache.maven.surefire.booter.ForkedBooter.main(ForkedBooter.java:495)

133

134 [INFO]

135 [INFO] Results:

136 [INFO]

137 Error: Failures:

138 Error: PillapiApplicationTests.testfindLit:69 expected: <Lit Erroné> but was: <Clinique Beauregard>

139 [INFO]

140 Error: Tests run: 2, Failures: 1, Errors: 0, Skipped: 0

141 [INFO]

142 [INFO] -----

143 [INFO] BUILD FAILURE

144 [INFO] -----

145 [INFO] Total time: 12.581 s

146 [INFO] Finished at: 2023-09-15T15:39:29Z

147 [INFO] -----

148 Error: Failed to execute goal org.apache.maven.plugins:maven-surefire-plugin:3.0.0:test (default-test) on project pillapi: There are test failures.

149 Error:

150 Error: Please refer to /home/runner/work/OCProjet11/OCProjet11/target/surefire-reports for the individual test results.

151 Error: Please refer to dump files (if any exist) [date].dump, [date]-jvmRun[N].dump and [date].dumpstream.

152 Error: -> [Help 1]

153 Error:

154 Error: To see the full stack trace of the errors, re-run Maven with the -e switch.

155 Error: Re-run Maven using the -X switch to enable full debug logging.

156 Error:

157 Error: For more information about the errors and possible solutions, please read the following articles:

158 Error: [Help 1] <http://cwiki.apache.org/confluence/display/MAVEN/MojoFailureException>

159 Error: Process completed with exit code 1.

## Fichier de configuration du workflow

Voici le contenu du fichier de configuration de la génération de l'exécutable du projet (build), les lignes précédés de # sont des commentaires expliquant la fonction du bloc de paramètre :

### **# Nom du workflow**

name: Java CI with Maven

### **# Branches du dépôt concernées**

on:

push:

branches: [ "main" ]

pull\_request:

branches: [ "main" ]

### **# Description de la tâche de build**

jobs:

build:

### **# plateforme système cible pour le build**

runs-on: ubuntu-latest

### **# Étapes du build**

steps:

#### **# Récupération du code du projet**

- uses: actions/checkout@v3

#### **# Récupération de la version et de la distribution java pour le build**

- name: Set up JDK 17

uses: actions/setup-java@v3

with:

java-version: '17'

distribution: 'temurin'

cache: maven

#### **# Nom de la tâche de build avec Maven**

- name: Build patient\_service with Maven

**# Build effectif en précisant l'emplacement et le nom du fichier de configuration du projet**

run: mvn -B package --file pom.xml

## Vérification du processus CI/CD

### Vérification dans JUNIT

Dans une étape préalable nous avons constaté la réussite des tests dans JUNIT et la bonne exécution du build lors du “push” du code de l'application sur github.

← Java CI with Maven

✓ Mise en place pour soutenance #4

Summary

Jobs

✓ build

Run details

Usage

Workflow file

Triggered via push last week	Status	Total duration	Artifacts
HPrevost pushed · ba89fa7 · main	Success	31s	—

maven.yml

on: push

✓ build 23s

Afin de valider le bon fonctionnement du processus nous allons à présent volontairement introduire une erreur dans une instruction de test `assertEquals` laquelle doit théoriquement produire une erreur de type “Failures” dans JUNIT. Nous allons par exemple saisir une autre valeur que celle attendue pour une des variables retournées par la fonction de recherche de lits.

Cette affectation est effectuée à la ligne 69 de la classe de test `P11apiApplication.java`.

Le nom de l'établissement retourné par la fonction principale de l'API lors d'une recherche de lit avec comme paramètre spécialité 2 et position GPS 20 est théoriquement “Clinique Beauregard”.



```
nomEtabExpected = "Clinique Beauregard";
```

Nous allons modifier cette valeur et indiquer par exemple

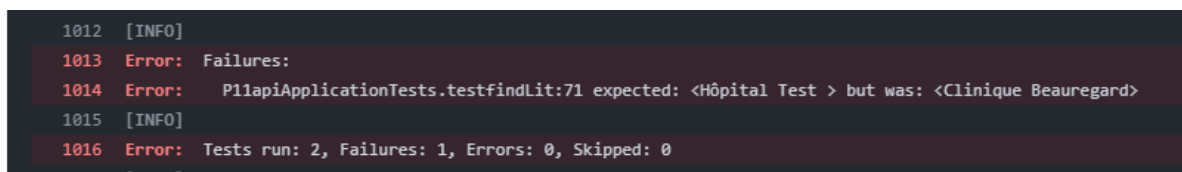
```
nomEtabExpected = "Hôpital Test";
```

L'instruction `assertEquals` suivante doit échouer :

```
nomEtabResult = ps.findLit((long) 2, (long) 7).getNomEtab();  
assertEquals(nomEtabExpected, nomEtabResult);
```

## Vérification dans le rapport de test du Workflow

Nous constatons bien dans le rapport de test du build la présence de l'erreur de type "Failure" comme l'illustre la capture d'écran suivante :



```
1012 [INFO]  
1013 Error: Failures:  
1014 Error: P11apiApplicationTests.testfindLit:71 expected: <Hôpital Test > but was: <Clinique Beauregard>  
1015 [INFO]  
1016 Error: Tests run: 2, Failures: 1, Errors: 0, Skipped: 0  
1017 [INFO]
```