2021

CI/CD

Karl Menino

Version : 1.0

12/10/2021

OPENCLASSROOMS

Table des matières

Table des matières

1. HISTORIQUE DES REVISIONS

Version	Description	Date	Validateur	
1.0		12/10/2021		Menino karl

Ce document décrit le pipeline d'intégration et de développement continue.

Karl menino,

Architecte Logiciel,

2. CI/CD

Voici un Screenshot du pipeline intégré à GITLAB :

```yaml
image: maven:latest

stages:
  - build
  - test
  - package

cache:
  paths:
    - .m2/repository/

variables:
  MAVEN_CLI_OPTS: "--batch-mode"
  MAVEN_OPTS: "-Dmaven.repo.local=.m2/repository"


build:
  stage: build
  script:
    - echo "Maven build started"
    - mvn $MAVEN_CLI_OPTS compile
  artifacts:
    expire_in: 10 min
    paths:
      - target/
      - "*/target"

test:
  stage: test
  script:
    - mvn surefire-report:report

  artifacts:
    when: always
    paths:
      - target/
      - "*/target"

package:
  stage: package
  script:
    - mvn package
  artifacts:
    name: "app-snapshot"
    paths:
      - target/*.jar
```

Nous allons rentrer plus en détail et décrire ligne par ligne à quoi correspond le code :

```
image: maven:latest
```
: Cela correspond à l'importation de la dernière image Maven. Cela permet de de rendre compatible notre Pipeline avec Spring.

```
stages:
  - build
  - test
  - package
```
: Ce sont les étapes que notre pipeline va faire. Nous demandons là de compiler notre projet, de le tester et enfin de générer le .JAR qui pourra être déployé en prod.

```
cache:
  paths:
    - .m2/repository/
```
: Nous indiquons le chemin du cache.

```
variables:
  MAVEN_CLI_OPTS: "--batch-mode"
  MAVEN_OPTS: "-Dmaven.repo.local=.m2/repository"
```
: Maven_OPT permet d'indiquer à Maven où se trouve le répertoire de cache. Il en a besoin pour son exécution. MAVEN_CLI_OPT permet de conserver les options que l'on utilisera avec la commande MVN.

```
build:
  stage: build
  script:
    - echo "Maven build started"
    - mvn $MAVEN_CLI_OPTS compile
  artifacts:
    expire_in: 10 min
    paths:
      - target/
      - "*/target"
```
: C'est la configuration de la compilation. Nous utilisons la commande MVN COMPILE avec les options déterminées dans variables. Nous récupérons ensuite un artefact. Vous pouvez voir que pour l'artefact, j'ai prévu large et récupère l'ensemble du dossier TARGET.

```
test:
  stage: test
  script:
    - mvn surefire-report:report

  artifacts:
    when: always
    paths:
      - target/
      - "*/target"
```

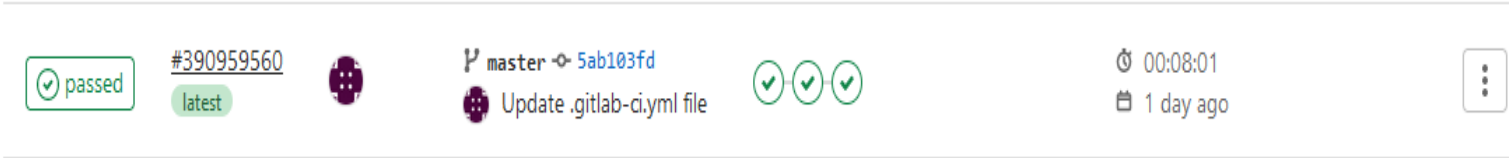
: C'est la configuration du test. Vous remarquer que je suis la même logique que le build. J'utilise la commande MVN surefire-report :report qui permet de faire les test et de générer un rapport surefire.

```
package:
  stage: package
  script:
    - mvn package
  artifacts:
    name: "app-snapshot"
    paths:
      - target/*.jar
```

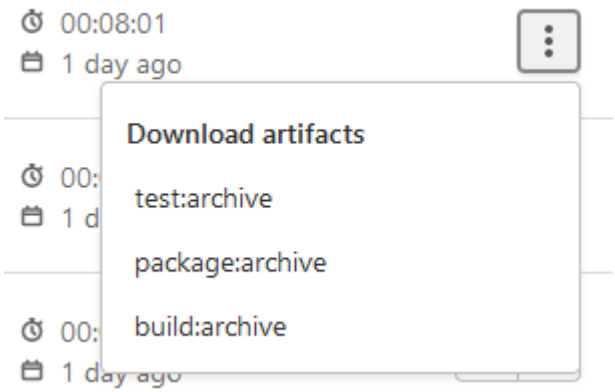
: Enfin c'est la création du .JAR. Nous utilisons la commande MVN package pour cela. Nous récupérons ici grâce a l'artefact que ce qui nous intéresse c'est à dire le JAR.

3. Résultat

Comme vous pouvez le voir dans notre Screenshot ci-dessous l'ensemble de notre pipeline fonctionne :



Nous récupérons bien nos artefacts :



Et nous générons bien un rapport SUREFIRE :

Surefire Report

Summary

Summary] [Package List] [Test Cases]

Tests	Errors	Failures	Skipped	Success Rate	Time
43	0	0	0	100%	139.145

Note: failures are anticipated and checked for with assertions while errors are unanticipated.

Package List

Summary] [Package List] [Test Cases]

Package	Tests	Errors	Failures	Skipped	Success Rate	Time
com.openclassroom.p11.model	5	0	0	0	100%	24.641
com.openclassroom.p11.dao	14	0	0	0	100%	4.816
com.openclassroom.p11	1	0	0	0	100%	0.04
com.openclassroom.p11.controller	3	0	0	0	100%	84.494