Etude de l'outil de développement continu
Gitlab CI/CD

BALLEJOS Lilian
ISIMA INP
Année Universitaire 2022/2023

Enseignant référant : Mr HILL David
Date du rendu : 17 avril 2023

ISIMA
1 rue de la Chebarde - TSA 60125 - CS 60026 - 63178 Aubière CEDEX
Tel : 04 73 40 50 00
Site web : isima.fr

Table des matières

Table des figures

1 Introduction

Durant nos cours d'**Architecture Logicielle et Qualité**, nous avons eu le choix d'étudier un outil de développement afin de découvrir son fonctionnement et son intérêt dans le déroulement classique d'un projet informatique.

Mon choix s'est donc porté sur **Gitlab CI/CD**, un outil pour l'intégration continue et le développement continu de projets informatiques.

2 Intérêt de l'intégration continue et du développement continu

Le développement continu et l'intégration continue sont des méthodes de développement logiciel qui visent à améliorer la qualité du code, à réduire le risque de conflits entre développeurs et à accélérer le processus de livraison des applications.

L'intégration continue est le processus d'intégration régulière et automatique des modifications de code apportées par un développeur dans une branche de développement commune, pour une détection précoce des bogues et des correctifs rapides.

Lorsque les développeurs soumettent des modifications de code, l'intégration continue vérifie et détecte automatiquement les conflits avec le code existant. De cette façon, les développeurs peuvent rapidement détecter et résoudre les problèmes avant qu'ils ne se propagent dans le code.

D'autre part, le développement continu implique l'automatisation de l'ensemble du processus de développement, de test et de déploiement d'applications.

L'objectif est de rendre le processus de livraison des applications plus rapide et plus efficace en automatisant les tâches répétitives, telles que la génération, la compilation et les tests automatisés de packages. Cela permet aux développeurs de se concentrer sur la création de nouvelles fonctionnalités au lieu de maintenir l'infrastructure et d'assurer la qualité du code grâce à des tests automatisés.

Les avantages de l'intégration continue et du développement continu sont nombreux :

- Améliorez la qualité du code grâce à des tests automatisés réguliers et à une détection précoce des erreurs
- Réduire le risque de conflits entre développeurs
- Accélérez la livraison des applications
- Meilleure visibilité sur le développement grâce à la qualité du code et aux mesures de progression des tests
- Réduisez les coûts de maintenance du code grâce à une meilleure qualité et fiabilité du code. En un mot, l'intégration continue et le développement continu sont des pratiques clés dans le développement de logiciels modernes qui permettent aux développeurs d'être plus productifs, d'assurer la qualité du code et de fournir des applications plus rapidement.

3 Gitlab CI/CD

3.1 Bref présentation du Gitlab CI/CD

FIGURE 1 – Logo de Gitlab CI/CD

Gitlab CI/CD est un outil de gestion de pipelines d'intégration et de déploiement continu, qui permet aux développeurs de travailler sur des projets en collaboration tout en améliorant la qualité de leur code et en facilitant le déploiement des applications.

Le fonctionnement de Gitlab CI/CD repose sur la définition de pipelines dans un fichier YAML, nommé "gitlab-ci.yml". Ce fichier permet de décrire les différentes étapes du processus de développement, de test et de déploiement de l'application.

Les pipelines sont composés de différentes étapes, appelées "jobs", qui peuvent être configurés pour s'exécuter automatiquement à chaque modification du code source. Les développeurs peuvent ainsi vérifier rapidement la qualité de leur code et détecter les erreurs avant même que le code ne soit fusionné dans la branche principale du projet.

L'intérêt de Gitlab CI/CD réside dans sa capacité à automatiser les tâches répétitives et fastidieuses liées au développement, au test et au déploiement d'applications. En utilisant des pipelines, les développeurs peuvent ainsi se concentrer sur la création de nouvelles fonctionnalités et l'amélioration de leur code, tout en assurant la qualité de leur travail grâce à des tests automatisés.

3.2 Utilisation dans le projet

3.2.1 Implémentation de la pipeline

Ainsi, nous avons fait le choix de faire notre projet de SMA (simulation multi-agent) en C++.

Il me fallait donc créer une pipeline dans le fichier **gitlab-ci.yml** qui permette de gérer à chaque commit (ou lancement à la main de la pipeline) certaines étapes pour vérifier la bonne qualité du projet.

Dans notre cas, étant pressé par le temps, je me suis contenté de mettre en place une pipeline très simple avec une seule étape (un seul "job") qui est la partie **build** (voir figure 2)

```yaml
image: gcc

build:
  stage: build
  # instead of calling g++ directly you can also use some build toolkit like make
  # install the necessary build tools when needed
  before_script:
    - apt update && apt -y install make autoconf
  script:
    - cd code
    - make
```

FIGURE 2 – Mon fichier gitlab-ci.yml

Il est intéressant de remarquer que à chaque passage dans la pipeline on utilise une image docker (à chaque passage dans la pipeline, un nouvel environnement est crée !). Ainsi c'est pour cela qu'avant d'exécuter le script du job "build", je réinstalle le paquet "make" dont j'aurais besoin dans le script.

Mon étape (job) consiste juste à compiler le code reçu.

Il y a alors deux états possibles :
— La compilation réussie : Le seul job de la pipeline est validé et donc la pipeline entière est validée (un tick vert apparait dans l'état du projet)
— La compilation échoue : Le seul job de la pipeline a échoué et donc la pipeline indique que il y a eu un soucis (une croix rouge apparait)

On peut voir sur la figure 3 les deux états possibles de la pipeline

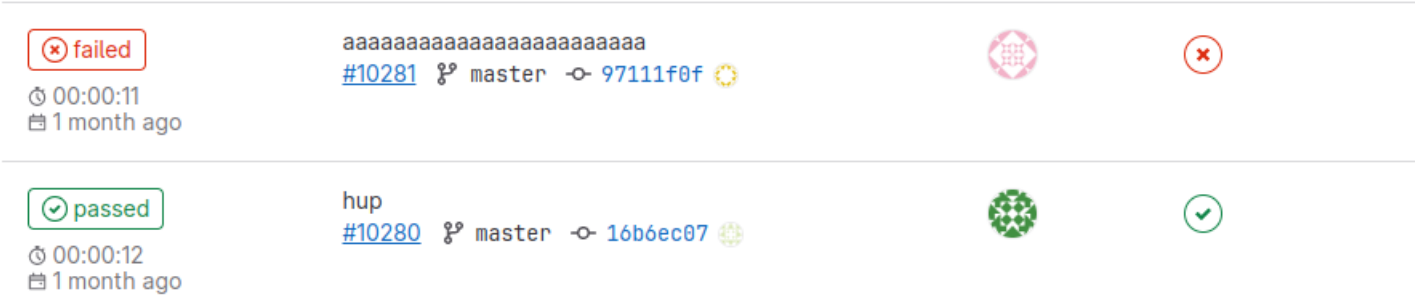


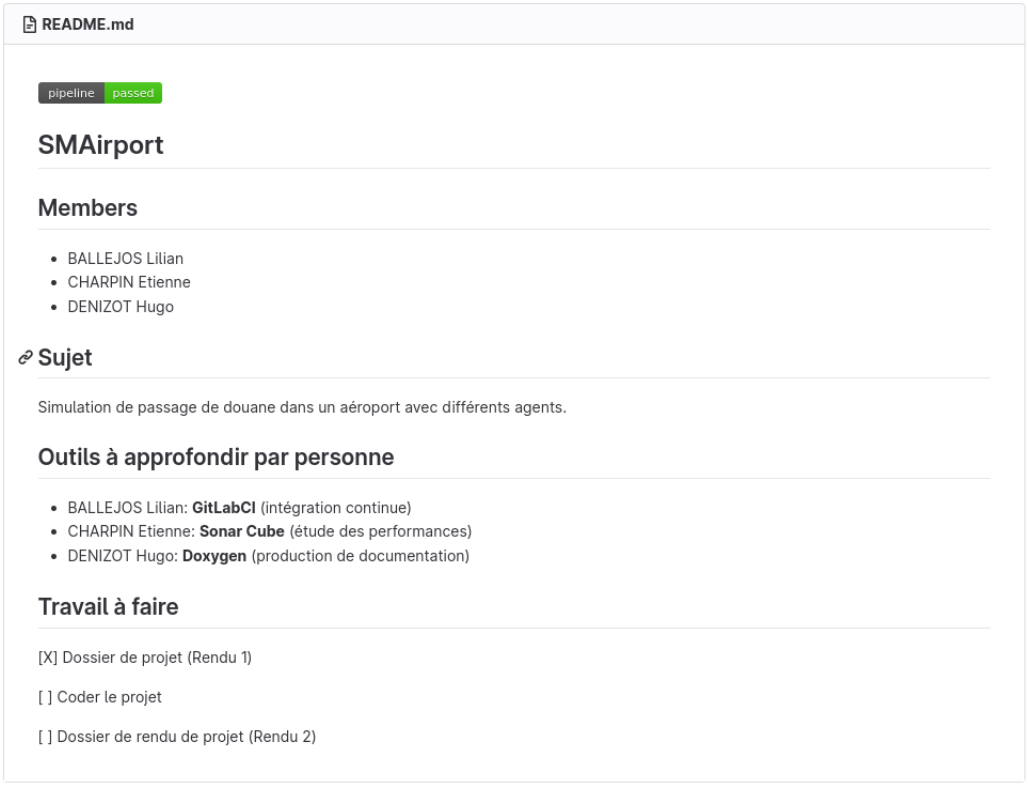
FIGURE 3 – Différents retours de la pipeline

On remarque aussi sur la figure 3 qu'il y a bien un seul job dans la pipeline car une seule croix rouge ou un seul tick vert !

3.2.2 Visualisation de l'état de la dernière pipeline

Il est possible d'intégrer une visualisation du dernier état de la pipeline directement sur la page d'accueil d'un projet GitLab ou encore sur le **README.md** du projet.
Cela permet d'informer les possibles développeurs ou encore utilisateurs d'un projet informatique de l'état actuel de ce dit projet.

J'ai implémenté cette fonctionnalité dans notre projet comme on peut le voir sur la figure 4 et 5



README.md

pipeline passed

SMAirport

Members

- BALLEJOS Lilian
- CHARPIN Etienne
- DENIZOT Hugo

Sujet

Simulation de passage de douane dans un aéroport avec différents agents.

Outils à approfondir par personne

- BALLEJOS Lilian: **GitLabCI** (intégration continue)
- CHARPIN Etienne: **Sonar Cube** (étude des performances)
- DENIZOT Hugo: **Doxygen** (production de documentation)

Travail à faire

[X] Dossier de projet (Rendu 1)

[ ] Coder le projet

[ ] Dossier de rendu de projet (Rendu 2)

FIGURE 4 – Information de l'état de la pipeline dans le README.md

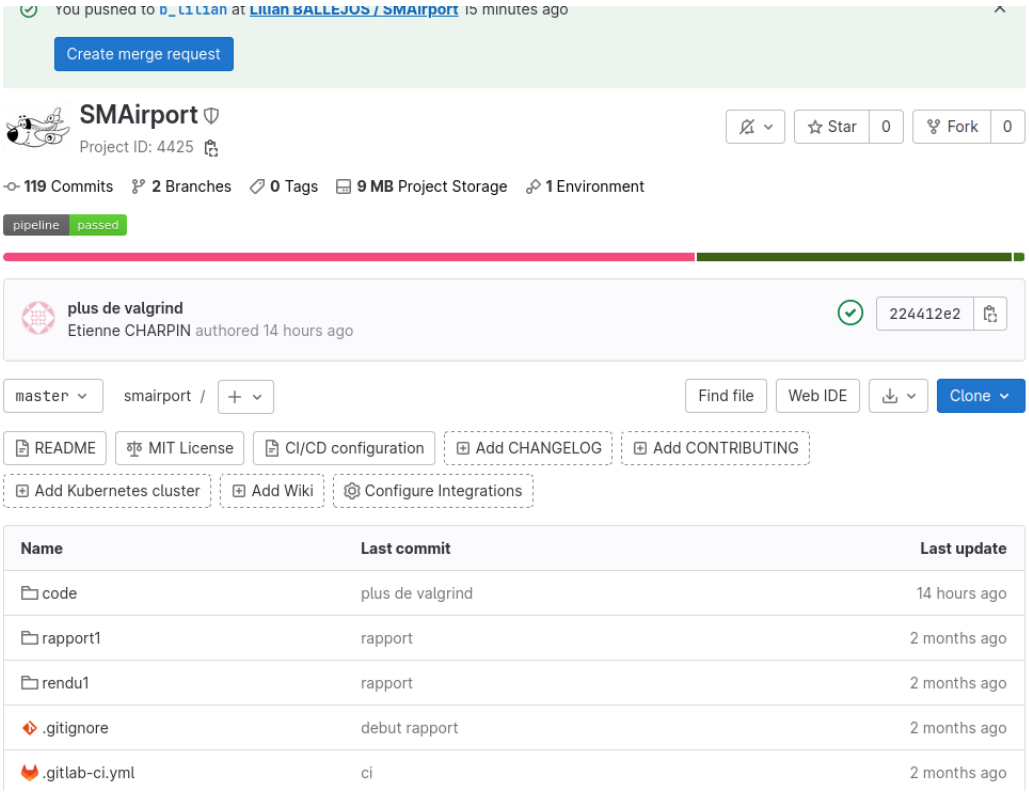


FIGURE 5 – Information de l'état de la pipeline directement sur la page du projet

3.3 Améliorations et ajouts possibles

Dans notre cas, étant pressé par le temps, nous n'avons pas eu le temps d'ajouter un élément clé à traiter dans chaque pipeline : des tests.

Il serait intéressant d'ajouter localement pour chaque classe de notre projet des tests unitaires qui vérifient le bon fonctionnement des méthodes implémentées.

Il serait aussi judicieux en théorie d'intégrer des tests fonctionnels qui couvrent le fonctionnement global de notre SMA. Le souci étant quel test doit-on essayer de vérifier sur une SMA en général ? sur notre SMA ?

Le but d'implémenter ces tests va permettre ensuite de pouvoir ajouter des jobs dans notre pipeline.

Je pense par exemple à cette architecture globale qu'il aurait possiblement fallu implémenter.

```
1    build:
2        stage: build
3        (action du build)
4
5    test_unitaire:
6        stage: tests unitaires
7        (effectuer tout les tests unitaires)
8
9    test_fonctionnel:
10       stage: tests fonctionnels
11       (effectuer tout les tests fonctionnels)
12
```

Ainsi, en séparant en plusieurs jobs notre pipeline, on pourrait savoir exactement où un problème a eu lieu et le fixer au plus vite ! Ensuite, après avoir repéré le job qui a échoué, on peut directement vérifier les logs de la pipeline échouée pour savoir quel test en particulier a échoué et ainsi régler le problème !

C'est là l'intérêt du développement continu et de l'intégration continue. Elles permettent de détecter et régler des problèmes directement lors de la création et du développement d'un projet !