Master 1 GIL - Document technique - Fonctionnement de la CI/CD Gitlab d'Agora

Groupe 1 Agora V3-1 : Partie bibliothèque de jeux

03 mai 2024

Version	1
Date	03 mai 2024
Rédigé par	BINGINOT Etienne

Mises à jour du document

Version	Date	Modification réalisée
1	03 mai 2024	Création du document

Table des matières

1 Rôle de la CI/CD

L'objectif de la CI/CD est de pouvoir assurer la qualité du projet au cours du développement, tout en permettant un déploiement accéléré. Agora étant une plateforme comprenant de nombreux jeux avec des règles complexes, le test automatique est une partie très importante du projet. Le test automatique via la pipeline permet d'assurer la non-régression du projet.

Gitlab permet ainsi de mettre en place une CI/CD qui exécutera automatiquement les scripts de votre choix à chaque push sur votre dépôt pour permettre des tests, déploiements ou encore des analyses Sonarqube automatisées.

2 Découpage en différents stages

La pipeline Gitlab est organisée via des stages et des jobs. Un job est en réalité un conteneur Docker qui va exécuter le script fourni. Un stage quant à lui regroupe un ensemble de jobs pour facilement pouvoir les organiser, pour par exemple retarder l'exécution d'un stage en attendant la fin d'un autre.

2.1 Build du projet

Le premier stage va donc être celui du build du projet. L'objectif ici va être de récupérer l'ensemble des dépendances du projet, ainsi que de créer l'image docker qui sera utilisée pour les tests.

Le premier job va donc commencer par créer l'image docker en se basant sur le Dockerfile Agora. Pour cela, le script du job va directement se connecter à la registry Docker du Gitlab du projet, pour ensuite réaliser le build du Dockerfile spécialement conçu pour la CI/CD puis push cette image sur la registry. Celle-ci sera ensuite accessible pour tous les jobs suivants.

Le second job va lui charger cette image Docker pour, à partir de celle-ci, télécharger l'ensemble des dépendances (dépendances composer, tailwind) pour les rendre accessible aux jobs suivants.

2.1.1 Transmission des données entre jobs

La transmission des données entre les différents jobs se fait par un système de cache. Il est en effet possible de définir des fichiers/dossiers de votre job qui seront, en fonction de votre choix, soit envoyés à Gitlab pour que celui-ci les stocke le temps de l'exécution de votre pipeline, soit pour directement demander ces fichiers à Gitlab pour les rendre accessible dans votre conteneur.

Il est ainsi possible de définir un premier job de build qui va push dans le cache les dossiers contenant le code des dépendances (var et vendor) puis, dans chacun des jobs nécessitant ses dépendances, de réaliser un pull du cache.

2.2 Analyse de la qualité

Après cette phase de build, une analyse de la qualité du projet est réalisé. Ainsi, 3 jobs vont s'exécuter en parallèle pour réaliser un scan de la qualité du projet avec phpstan, phpcs et twig-lint.

2.2.1 Utilisation de phpstan

PHPStan est un outil permettant de réaliser des analyses statiques de votre code pour détecter la présence de bugs, de problème de typages (type incompatible ou commentaire incomplet), ou d'erreurs de codes. Son objectif principal est de détecter les erreurs qui feront planter le programme.

2.2.2 Utilisation de PHPCS

PHPCS (PHP Code-Sniffer) est un outil qui va détecter les violations des normages de codage PSR-12. Il va ainsi remonter des warnings pour chaque ligne violant une règle PSR-12, en vous expliquant la raison.

Il est possible d'utiliser en conjonction avec PHPCS l'outil PHPCSFixer, pouvant automatiquement régler une bonne partie des violations des règles PSR-12 de façon automatique.

2.2.3 Utilisation de Twig-Lint

Twig-Lint est un outil qui va analyser vos templates Twig pour y détecter des erreurs de code, afin d'assurer que chaque template est individuellement valide.

2.3 Exécution des différents tests

Une fois l'étape qualité passée, les tests des jeux mais également de la bibliothèque de code seront exécutés. Chaque jeu possède son propre stage, découpé en différents jobs pour les tests unitaires, d'intégrations et d'applications. Cependant, les tests s'exécutent de façon parallèle pour l'ensemble des jeux.

De plus, les tests d'intégration et d'applications possèdent une configuration supplémentaire : il est nécessaire au préalable de l'exécution des tests d'initialiser la BDD et les données de tests.

2.3.1 Configuration des services dépendances

Pour certains types de tests, il est nécessaire de configurer des dépendances, comme par exemple une base de données. Gitlab permet de réaliser ceci via l'utilisation du mot clé services, permettant de sélectionner des images Docker qui seront instanciées en tant que conteneur au début du job et qui seront donc accessible pour le conteneur réalisant les tests.

2.4 Rapport JUnit et Sonarqube

Chaque test réalisé précédemment va produire deux rapports. Un rapport de tests, détaillant l'exécution du test (erreur, temps d'exécution, etc) et un rapport de coverage (les lignes du code qui ont été parcourus). De plus, le coverage est également affiché sur le terminal du job.

2.4.1 Intégration avec GitLab

Il est possible de configurer GitLab avec une regexp dans la CI/CD pour que celui-ci puisse analyser la sortie texte du terminal du job et détecter le coverage de l'application. De plus, il est nécessaire d'utiliser des configurations précises pour chaque jeu, pour pouvoir exclure du coverage tous les autres jeux sauf le jeu testé actuellement.

De plus, il est également possible de fournir les rapports de tests à GitLab via l'utilisation des artifacts pour facilement accéder aux rapports de tests depuis l'application de façon graphique. Les artifacts sont des ressources que gitlab sauvegardera même après la fin de la pipeline pour un temps configurable.

2.4.2 Intégration avec Sonarqube

Il est tout d'abord nécessaire de fusionner les rapports produits par PHPUnit avant de les fournir à Sonarqube pour l'analyse. Un stage report va donc s'occuper de cela, via l'utilisation de dépendances externes réalisant le merge. Ces dépendances sont installées ici afin de ne pas avoir à les installer pour l'ensemble des tests, ce job étant le seul endroit où ceux-ci sont utilisées.

Il est ensuite possible d'utiliser l'image docker Sonar-Scanner pour pouvoir réaliser le scan Sonar de l'application, qui sera ensuite transmis au serveur sonarqube. Il est possible de configurer le scanner via le

fichier sonar-project.properties à la racine du dépôt, et en modifiant les variables CI/CD de gitlab pour l'url et le token d'authentification du serveur sonarqube.

2.5 Déploiement automatique

La dernière étape de la CI/CD va être le déploiement automatique. Il est possible ainsi en configurant les variables de Gitlab de se connecter au serveur distant en SSH pour lancer le serveur à distance, via des commandes Docker. Il est ainsi possible de configurer les variables suivantes :

- SERVER_IP : l'ip du serveur distant
- SERVER_USER : l'utilisateur a utilisé pour se connecter au serveur
- SERVER_LOCATION : l'endroit sur le serveur distant où se trouve votre projet
- DOCKER_COMPOSE_FILE : le fichier docker compose a utilisé pour le déploiement
- GIT_BRANCH_NAME : la branche utilisée pour le déploiement
- ID_RSA : la clé SSH à utiliser pour se connecter en ssh au serveur distant

Il vous suffira donc ensuite d'importer le code du script de déploiement en définissant ses variables pour que votre projet soit automatiquement déployée sur votre serveur.

3 Factorisation du code de la CI/CD

Afin de factoriser le code de la CI/CD, des ancres sont utilisées pour permettre de regrouper le code commun à différents jobs. Ainsi, des ancres ont pû être définir pour réaliser un script de déploiement, pour définir une base de job de test unitaire (permettant ainsi une configuration pour exporter les rapports PHPUnit), une base de test d'intégration (avec un service de BDD) et d'application (avec un service mercure).

3.1 Ancre de base de tests

L'ancre de base de tests unitaires configure l'image Docker créé lors de la phase de build. Elle définit également la localisation des rapports PHPUnit pour pouvoir les exporter. Elle met en place aussi la regexp GitLab pour avoir une indication du coverage, et charge enfin le cache pour récupérer les dépendances nécessaires.

L'ancre de base de tests d'intégrations quand à elle rajoute uniquement la dépendance à la BDD, et celle pour les tests d'applications la dépendance à Mercure.

3.2 Ancres utilitaires

De nombreuses ancres utilitaires sont disponibles pour aider à la réalisation de la CI/CD. Les plus importantes sont :

- cleanReport : permet de nettoyer et de réparer les rapports générés par PHPUnit, pour les rendre compatible avec l'analyseur de Sonarqube et de Gitlab
- publishReport : permet de publier les rapports pour Gitlab
- cache : permet de charger le cache en mode push/pull (pull au début du job et push à la fin)

3.3 Surcharge des ancres

Il est également possible si besoin de redéfinir certaines propriétés des ancres après en avoir hérités pour raffiner leurs comportements. Par exemple, il est possible dans le cas de l'ancre du cache de choisir uniquement le mode pull ou push.