RÉPUBLIQUE DU BÉNIN
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR
ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ D'ABOMEY-CALAVI

INSTITUT DE FORMATION ET DE RECHERCHE EN INFORMATIQUE

BP 526 Cotonou Tel : +229 21 14 19 88
https://www.ifri-uac.bj/ Courriel : contact@ifri.uac.bj

MÉMOIRE

pour l'obtention du

Diplôme de Licence en Informatique

Option : Sécurité Informatique

Présenté par :
Carmel **ADIKIN**

Implémentation d'une chaine CI/CD avec la méthode DevOps

Sous la supervision :
Dr Arnaud **AHOUANDJINOU**
Enseignant-chercheur à l'Université d'Abomey-Calavi

Membres du jury :

EZIN C. Eugène	Professeur	Université d'Abomey-Calavi	Président
KASSA Célia	Ingénieur	Université d'Abomey-Calavi	Examinateur
AHOUANDJINOU Arnaud	Docteur (MC)	Université d'Abomey-Calavi	Rapporteur

Année Académique : **2022-2023**

Sommaire

Dédicace ii

Remerciements iii

Résumé iv

Abstract v

Table des figures vi

Liste des tableaux viii

Liste des acronymes ix

Introduction 1

1 Etat de l'art 3

2 Analyse et choix techniques 10

3 Solutions et Discussions 16

Conclusion 31

Bibliographie 32

Webographie 33

Table des matières 35

Dédicace

A

Mon Père **Laurent ADIKIN**

Ma Mère **Emeline BOSSA**

Remerciements

Nous exprimons notre vive gratitude aux personnes physiques et institutions qui ont contribué à rendre meilleur notre travail, notamment :

- le Professeur Eugène C. EZIN, Directeur de l'Institut de Formation et de Recherche en Informatique (IFRI) pour ses conseils et la formation qu'il nous a donnés ;
- Dr Arnaud AHOUANDJINOU, notre maître de mémoire, pour sa confiance, ses conseils avisés et sa disponibilité ;
- tous les enseignants de l'IFRI pour nous avoir donné les enseignements nécessaires tout au long de notre formation ;

Que tous ceux qui nous ont aidé, de près ou de loin, trouvent ici l'expression de nos sentiments les meilleurs.

Résumé

Le DevOps est une approche agile de gestion de projets informatiques qui vise à améliorer la collaboration et l'intégration entre les équipes de développement logiciel et d'exploitation informatique. Son objectif principal est de fournir des logiciels de manière rapide, fiable et régulière, tout en améliorant la qualité et la sécurité des applications. Dans le cadre de ce mémoire, une chaîne CI/CD (intégration continue/déploiement continu) a été mise en place pour optimiser le processus de développement et de déploiement logiciel de l'Environnement Numérique de Travail (ENT) d'une institution universitaire. L'implémentation de cette chaîne CI/CD s'est reposée sur plusieurs piliers, dont la mise en place d'une architecture adaptée, le choix des bons outils et technologies, et la définition de processus clairs et efficaces. Les tests réalisés montrent que tous ces efforts contribuent à accroître la productivité des équipes de développement, à réduire les erreurs lors de la rédaction des tests et des déploiements, et à réduire la latence entre le développement et le déploiement.

Mots clés : DevOps, intégration continue, déploiement continu, chaîne CI/CD

Abstract

DevOps is an agile approach to IT project management that aims to improve collaboration and integration between software development and IT operations teams. Its main objective is to deliver software quickly, reliably and consistently, while improving application quality and security. In the context of this thesis, a continuous integration/continuous deployment (CI/CD) pipeline was set up to optimize the software development and deployment process for a university institution's Digital Workplace. The implementation of this CI/CD pipeline was based on several pillars, including the establishment of a suitable architecture, the choice of the right tools and technologies, and the definition of clear and efficient processes. The tests carried out show that all these efforts contribute to increasing the productivity of development teams, reducing errors when writing tests and deployments, and reducing latency between development and deployment.

Key words: DevOps, continuous integration, continuous deployment, CI/CD pipeline

Table des figures

1.1 Collaboration et communication dans un milieu DevOps [17] 4
1.2 Processus d'intégration continue [10] 6
1.3 Processus de déploiement continu [18] 7
1.4 Pipeline CI/CD [14] 8

3.1 Serveurs Amazon Web Services (AWS) 17
3.2 Commandes d'installation de Jenkins [16] 17
3.3 Installation de Jenkins réussie 17
3.4 Ecran Unlock Screen de Jenkins 18
3.5 Plugins par défaut de Jenkins 18
3.6 Formulaire de création d'un administrateur Jenkins 19
3.7 Ecran d'accueil de Jenkins 19
3.8 Plugin SSH Agent 19
3.9 Plugin Blue Ocean 20
3.10 Plugin Global Slack Notifier 20
3.11 Commandes Docker pour mettre à jour l'index des paquets [9] 20
3.12 Ajout de l'adresse de dépôt de paquets Debian [9] 21
3.13 Installation de Docker [9] 21
3.14 Installation de Docker réussie. 21
3.15 Fichier Dusk AdminCreateInstructor.php 23
3.16 Fichier Dusk InstructorLoginTest.php 23
3.17 Fichier Dusk UserRegister.php 24
3.18 Jenkinsfile, partie CI 24
3.19 Jenkinsfile, partie CI 25
3.20 Jenkinsfile, notifications slack pour publication 25
3.21 Slack, notification Docker Hub 25
3.22 Image de l'application sur le Docker Hub 25
3.23 Fichier docker-compose.yml 26
3.24 Fichier docker-compose.yml 26
3.25 Jenkinsfile, étape déploiement en test 26
3.26 Jenkinsfile, étape notifications Slack après déploiement et test en staging 27
3.27 Slack, notification test 27
3.28 Etape de test réussie 27
3.29 Jenkinsfile, étape déploiement en production 27
3.30 Jenkinsfile, étape notifications Slack après déploiement en production 28
3.31 Slack, notification déploiement 28
3.32 Application Ulearn, en ligne 28

3.33 Pipeline, au vert 29
3.34 Résultat pour le test de création d'un instructeur 29
3.35 Résultat pour la connexion d'un instructeur 29
3.36 Résultat pour l'inscription d'un utilisateur 29
3.37 Résultats des tests automatisés 30

Liste des tableaux

2.1 Comparaison de quelques plateformes de CI/CD 11
2.2 Comparaison de quelques technologies de conteneurisation 12
2.3 Comparaison de quelques outils de versionnement 12
2.4 Comparaison de quelques outils de test 13
2.5 Comparaison de quelques outils de communication et de collaboration 13

3.1 Caractéristiques des serveurs 16

Liste des acronymes

AWS :

Amazon Web Services vi, 16, 17

CD :

Continuous Deployment/Continuous Delivery iv–vi, viii, 6, 8, 10, 11, 14, 15, 24, 31, 35, 36

CI :

Continuous Integration iv–vi, viii, 4–6, 8, 10, 11, 14, 15, 24, 25, 31, 35, 36

DevOps :

Development and Operations iv–vi, 3, 4

ENT :

Environnement Numérique de Travail iv, 1, 2

GPG :

GNU Privacy Guard 20

HTTPS :

HyperText Transfer Protocol Secure 20

LMS :

Learning Management System 10, 14, 36

OS :

Operating System 16

PHP :

PHP: Hypertext Preprocessor 13

QA :

Quality Assurance 3

RAM :

Random Access Memory 16

SSH :

Secure Shell vi, 19

VCS :

Version Control System 9

VPS :

Virtual Private Server 16

YAML :

YAML Ain’t Markup Language 25

Introduction Générale

1 Problématique

Dans le cadre d'une approche traditionnelle dans le processus de développement logiciel, on se retrouve avec des équipes distinctes chargées des phases de développement, de test et de déploiement. Les processus de test sont réalisés manuellement, ce qui peut entraîner des erreurs et des incohérences dans le logiciel final. De plus, les déploiements sont effectués de manière manuelle, ce qui peut entraîner des retards, des erreurs de configuration et des interruptions de service.

L'absence d'une approche moderne de développement logiciel engendre également des difficultés pour s'adapter aux besoins changeants des étudiants. Les mises à jour et les correctifs sont souvent lents à déployer, ce qui peut entraîner des problèmes de compatibilité et une baisse de satisfaction des étudiants.

Cependant, en adoptant une approche plus moderne, il est possible d'améliorer considérablement le processus de développement logiciel. En intégrant des pratiques telles que l'automatisation des tests, l'intégration continue et le déploiement continu, il devient possible d'accélérer les livraisons de fonctionnalités, d'améliorer la qualité du logiciel et de mieux répondre aux besoins des utilisateurs.

2 Contexte et justification

Une institution universitaire de premier plan souhaite développer son propre Environnement Numérique de Travail (ENT) afin de fournir une plateforme numérique moderne et complète à sa communauté universitaire. Cette initiative s'inscrit dans la volonté de l'institution de se positionner en tant que référence en matière de technologie éducative et de contribuer à la vision nationale de devenir un acteur majeur dans le domaine technologique.

En tant que leader académique, cette institution reconnaît l'importance de s'armer technologiquement pour améliorer la qualité de son enseignement et offrir aux étudiants, aux enseignants et au personnel administratif des outils numériques performants. En développant son propre ENT, l'institution universitaire vise à créer un environnement numérique intégré pour faciliter les échanges et la collaboration, renforcer les méthodes pédagogiques et administratives, et offrir des services innovants adaptés aux besoins spécifiques de sa communauté universitaire.

3 Objectifs et contributions

L'objectif principal de ce projet de mémoire est de mettre en place une chaîne CI/CD avec la méthode DevOps, adaptée aux besoins spécifiques d'une institution universitaire réputée, fréquentée par un grand nombre d'étudiants. Ce projet vise à automatiser les processus de développement, de tests et de déploiement des applications de son Environnement Numérique de Travail (ENT), en favorisant l'intégration continue et en assurant la qualité et la sécurité des livrables. Pour cela, nous devons atteindre les objectifs spécifiques suivants :

- Effectuer une revue de littérature sur les notions liées au DevOps comme l'intégration continue, le déploiement continu, la collaboration inter-équipes, la conteneurisation.
- Identifier les outils appropriés pour chaque étape de la chaîne CI/CD (gestion de versions, conteneurisation, tests, déploiement).
- Créer un environnement de développement logiciel automatisé, sécurisé et performant, intégrant des processus de tests approfondis, une architecture sécurisée et un déploiement continu des applications.
- Mettre en place des processus d'automatisation pour les tests fonctionnels, d'intégration et de non régression.
- Réaliser des tests fonctionnels, d'intégration et de non régression.
- Documenter et évaluer les bonnes pratiques pour assurer la sécurité des applications déployées.

Les travaux réalisés visent à apporter des contributions significatives à l'institution en améliorant l'efficacité et la rapidité du développement logiciel de son Environnement Numérique de Travail (ENT), tout en augmentant la qualité des livrables. Cette initiative a pour but de permettre de réduire les erreurs et les risques, renforçant ainsi la confiance dans les applications déployées. De plus, la mise en place d'une chaîne CI/CD cherche à favoriser la collaboration et la communication entre les équipes et facilitera l'adaptation aux changements et la gestion des différentes versions du code source de l'ENT. Elle aspire également à la sécurité des applications déployées et à une culture d'amélioration continue au sein de l'institution.

4 Organisation du mémoire

Le présent travail se présente en trois (03) chapitres. Le premier chapitre propose un état de l'art consacré aux divers concepts liés au DevOps, notamment les bases du DevOps, le rôle crucial des tests système dans un pipeline, ainsi que les outils et technologies associés à cette approche. Dans le deuxième chapitre, il s'agit de l'analyse et de la motivation des choix techniques effectués pour la mise en œuvre de notre solution. Le troisième chapitre présente la mise en oeuvre de notre solution ainsi que les tests réalisés. Ce chapitre expose également les résultats obtenus ainsi que les perspectives envisagées.

Chapitre 1

Etat de l'art

Introduction

Dans ce chapitre, nous réalisons en premier lieu, une étude des concepts fondamentaux du DevOps afin de mieux cerner le problème à résoudre. En second lieu, nous étudions les différents types d'outils et technologies utilisés couramment dans le monde DevOps afin d'affiner notre analyse technique par la suite.

1.1 Concepts fondamentaux

1.1.1 DevOps

1.1.1.1 Définition

DevOps est une manière de penser visant à briser les silos entre le développement et l'exploitation logicielle. Il encourage la collaboration et la communication entre les équipes pour améliorer la rapidité, la fiabilité et la sécurité des logiciels [1]. En d'autres termes, DevOps s'attaque aux problèmes qui surviennent lorsque les équipes de développement et d'exploitation travaillent de manière indépendante.
DevOps vise à créer une culture et un environnement dans lesquels la conception, les tests et la diffusion de logiciels peuvent être réalisés rapidement, fréquemment et efficacement.DevOps n'est pas seulement une méthodologie, c'est une véritable philosophie de travail [21].

1.1.1.2 Historique

Le terme "DevOps" a été inventé en 2009 par Patrick Debois et Andrew Clay Shafer lors d'une conférence O'Reilly. Cependant, les racines du mouvement remontent aux années 1990 avec l'avènement du mouvement Agile et des pratiques de développement logiciel Lean. Tout ceci a conduit à la naissance d'un mouvement qui prône le rapprochement des développeurs et des opérations au sein des équipes. Il s'agit de pouvoir apporter plus rapidement une valeur commerciale ajoutée aux utilisateurs et ainsi d'être plus compétitif sur le marché [6].

1.1.1.3 Principes fondamentaux

Le DevOps, plus qu'un mouvement, est une culture qui repose sur des principes et ne peut fonctionner que si toutes les entités collaborent ensemble. Le DevOps affecte alors la plupart des participants aux projets informatiques qui sont : les développeurs, les testeurs QA, le chef de projet, le chef de produit, et les autres (marketing,

ressources humaines, etc.).
Il repose sur plusieurs principes clés tels que la collaboration et la communication, l'automatisation, la culture d'apprentissage et le partage de responsabilité.

Collaboration et communication

Les équipes de développement et d'exploitation doivent travailler ensemble de manière transparente et communiquer ouvertement [12].

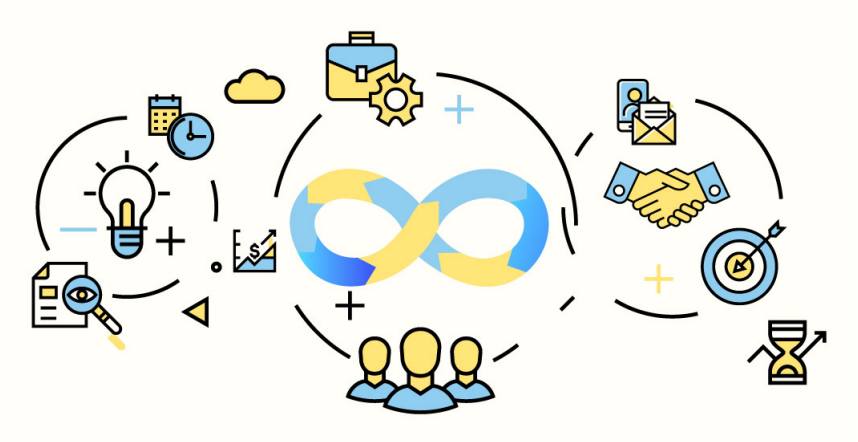


FIGURE 1.1 : Collaboration et communication dans un milieu DevOps [17]

Automatisation

Les processus manuels doivent être automatisés autant que possible pour améliorer la rapidité et la fiabilité.

Culture d'apprentissage

Les équipes doivent adopter une culture d'apprentissage continu et d'amélioration itérative [1].

Partage de responsabilité

Les équipes doivent partager la responsabilité de la qualité et de la performance du logiciel [12].

1.1.2 Intégration continue

1.1.2.1 Définition

L'intégration continue (CI) est une pratique logicielle qui vise à automatiser le processus de développement logiciel en intégrant régulièrement les modifications du code dans une base de code commune. Cette pratique permet d'identifier et de corriger les erreurs plus rapidement, d'améliorer la qualité du code et de garantir une livraison plus fréquente et plus fiable des logiciels.

1.1.2.2 Avantages

Les avantages de l'intégration continue sont l'amélioration de la qualité du code et de la rapidité du développement ainsi que le renforcement de la collaboration et de la fiabilité du travail.

Amélioration de la qualité du code

Grâce aux tests automatisés mis en place dans le cadre de la CI, les développeurs sont en mesure de détecter les erreurs et les anomalies beaucoup plus tôt dans le processus de développement. Cette détection précoce permet de corriger les problèmes au fur et à mesure qu'ils surviennent, évitant ainsi l'accumulation d'erreurs potentielles et garantissant la robustesse et la fiabilité du code produit [24].

Amélioration de la rapidité de développement

En permettant aux développeurs d'intégrer leurs modifications plus rapidement et plus facilement, la CI favorise un rythme de développement plus soutenu. Grâce à des processus automatisés de construction, de test et de déploiement, les équipes peuvent itérer plus rapidement sur leurs codes, réduisant ainsi les cycles de développement et accélérant la mise sur le marché des fonctionnalités [24].

Renforcement de la collaboration
En fournissant une visibilité accrue sur les modifications apportées par les autres membres de l'équipe, la CI encourage la communication et la coordination efficaces entre les développeurs. Cette transparence permet aux membres de l'équipe de mieux comprendre les changements en cours et de collaborer de manière plus étroite pour résoudre les problèmes et atteindre les objectifs du projet de manière collective [24].
Renforcement de la fiabilité
En automatisant les processus de construction et de test, la CI garantit des builds plus fiables et reproductibles. Cela permet de minimiser les risques d'erreurs humaines et de garantir la cohérence et la qualité des versions livrables aux utilisateurs finaux. Ainsi, les équipes peuvent offrir des produits plus stables et plus fiables, renforçant ainsi la confiance des utilisateurs et la réputation de l'entreprise [24].

1.1.2.3 Processus

Le processus d'intégration continue est généralement composé de plusieurs étapes, du développement à la publication d'un artefact.

Etape 1 : Développement
Les développeurs écrivent du code et effectuent des tests locaux sur leurs environnements de développement. Cette étape initiale permet de concevoir et de développer de nouvelles fonctionnalités ou de corriger des problèmes dans le code existant.

Etape 2 : Intégration
Une fois le code développé, il est intégré dans la base de code commune partagée par l'équipe de développement. Cette étape vise à combiner les différentes contributions des développeurs en un seul ensemble de code cohérent et fonctionnel.

Etape 3 : Tests
Après l'intégration du code, des tests automatisés sont exécutés sur l'ensemble du code intégré. Ces tests contribuent à garantir que chaque composant du logiciel fonctionne correctement selon les spécifications définies.

Etape 4 : Construction
Une fois que les tests automatisés sont réussis, le code est compilé pour créer un artefact exécutable, tel qu'un fichier exécutable, une archive ou un conteneur. Cet artefact représente la version stable et testée de l'application prête à être déployée.

Etape 5 : Publication de l'aertefact
L'artefact généré est déployé dans un dépôt distant, prêt à être distribué et déployé dans l'environnement de production et/ou de test.

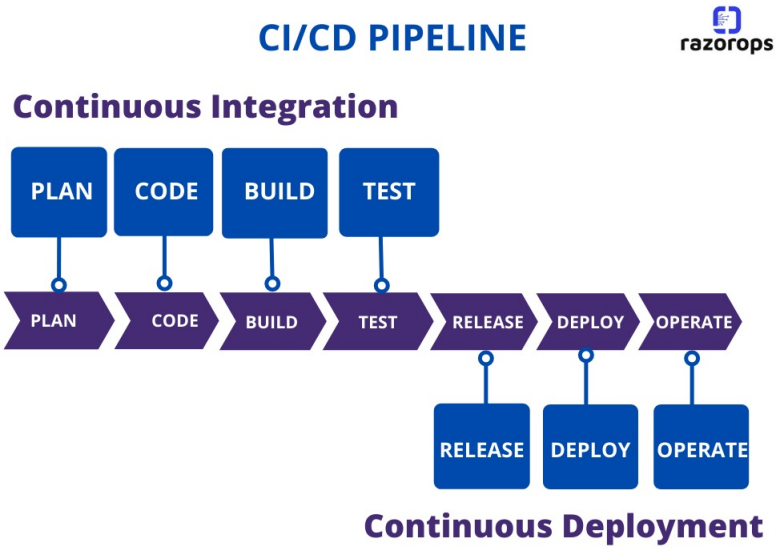


FIGURE 1.2 : Processus d'intégration continue [10]

1.1.3 Déploiement continu

1.1.3.1 Définition

Le déploiement continu (CD) est une pratique logicielle qui vise à automatiser le processus de déploiement de nouvelles versions logicielles en production. Cette pratique permet de réduire le temps et les efforts nécessaires pour déployer des logiciels, d'améliorer la fiabilité du processus de déploiement et de minimiser les risques de pannes en production.

1.1.3.2 Différence avec la livraison continue

Le déploiement continu et la livraison continue sont à peu près similaires mais à une différence près. La livraison étend le CI pour garantir que le code est toujours dans un état publiable, mais le déploiement est manuel tandis que le déploiement continu va encore plus loin en déployant automatiquement le code en production après avoir réussi les tests [19].

1.1.3.3 Avantages

Les avantages du déploiement continu sont l'amélioration de la rapidité de déploiement et de la fiabilité ainsi que le renforcement de la collaboration et l'optimisation de la satisfaction client.

Amélioration de la rapidité de déploiement

Grâce au CD, les équipes sont en mesure de déployer les changements et les mises à jour plus rapidement et plus facilement qu'auparavant. Cette rapidité accrue dans le déploiement permet aux entreprises de réagir plus rapidement aux besoins changeants du marché et de fournir des fonctionnalités plus rapidement à leurs utilisateurs [19].

Amélioration de la fiabilité

En automatisant et en standardisant le processus de déploiement, les entreprises réduisent considérablement le risque d'erreurs humaines et de défaillances techniques lors des mises en production. Les déploiements deviennent ainsi plus fiables, plus cohérents et moins sujets à des erreurs, ce qui garantit une meilleure stabilité et une meilleure disponibilité des applications logicielles [19].

Renforcement de la collaboration

En encourageant une culture de travail plus étroite et une communication transparente entre les différents

départements, le déploiement continu permet aux équipes de surmonter les silos organisationnels et de travailler ensemble de manière plus efficace et harmonieuse. Cette collaboration accrue conduit à une meilleure compréhension des besoins et des défis de chacun, favorisant ainsi une résolution plus rapide des problèmes et une innovation continue [19].

Optimisation de la satisfaction client

En livrant des versions de logiciels de meilleure qualité plus rapidement, les entreprises sont en mesure de répondre de manière plus agile aux demandes et aux retours des clients. Les utilisateurs bénéficient ainsi d'une expérience améliorée avec les produits et services de l'entreprise, ce qui renforce leur fidélité et leur satisfaction globale. En conséquence, l'entreprise peut maintenir sa compétitivité sur le marché et continuer à développer et à fidéliser sa clientèle [19].

1.1.3.4 Processus

Le processus de déploiement continu comprend le déploiement et les tests dans l'environnement de test et le déploiement dans l'environnment de production.

Etape 1 : Déploiement dans un environnement de test

L'artefact compilé à partir du code intégré est automatiquement déployé dans un environnement de test. Cet environnement est une réplique de l'environnement de production, permettant ainsi de tester la nouvelle version du logiciel dans des conditions proches de la réalité. Cette étape vise à identifier tout dysfonctionnement ou anomalie potentielle avant le déploiement en production.

Etape 2 : Validation

Une fois déployée dans l'environnement de test, la nouvelle version du logiciel est soumise à une batterie de tests automatisés. Ces tests, comprenant des tests système, des tests de non régression, des tests de performance, des tests de sécurité, etc., sont conçus pour valider le bon fonctionnement de la nouvelle version sous différents aspects. L'objectif est de garantir que la nouvelle version répond aux exigences de qualité et de fiabilité avant d'être déployée en production.

Etape 3 : Déploiement en production

Si la validation est réussie et que la nouvelle version passe avec succès tous les tests automatisés, elle est alors automatiquement déployée en production. Ce déploiement automatique garantit un processus fluide et efficace, réduisant ainsi le temps nécessaire pour mettre à disposition les nouvelles fonctionnalités ou corrections auprès des utilisateurs finaux.

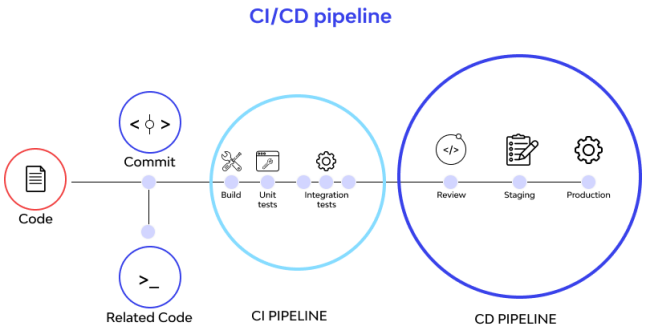


FIGURE 1.3 : Processus de déploiement continu [18]

1.1.4 Pipeline CI/CD

1.1.4.1 Définition

Un pipeline CI/CD ou encore chaine CI/CD est un ensemble d'étapes automatisées qui implémentent les pratiques d'intégration continue (CI) et de livraison continue (CD). Il permet d'automatiser le processus de développement logiciel, depuis l'écriture du code jusqu'au déploiement en production.

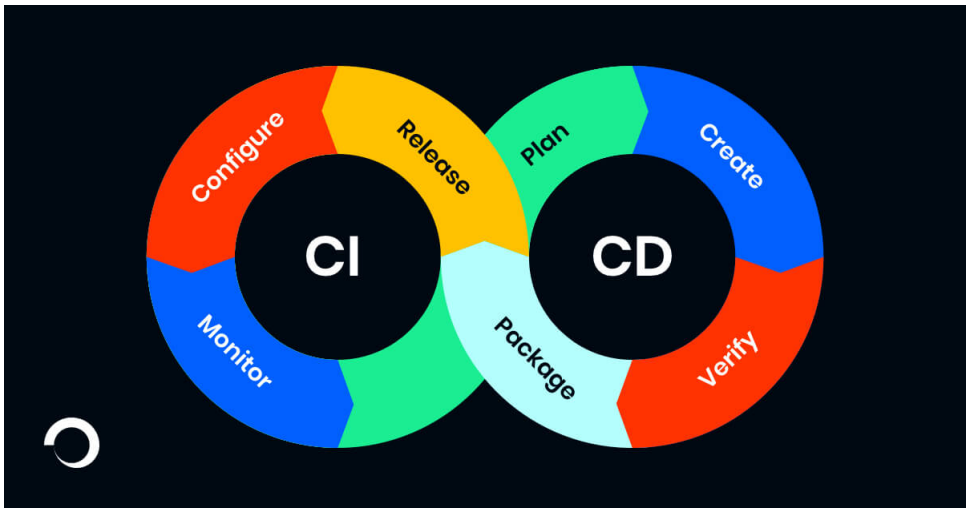


FIGURE 1.4 : Pipeline CI/CD [14]

1.1.4.2 Types de pipeline

Il existe différents types de pipelines CI/CD, en fonction des besoins de l'équipe et de l'organisation.
Les types les plus courants sont le pipeline simple, le pipeline avec plusieurs environnements et le pipeline avec déploiement bleu-vert.

Pipeline simple

Ce type de pipeline est le plus simple et le plus courant. Il est idéal pour les petits projets ou les équipes débutantes mais manque de flexibilité pour les projets plus complexes.

Pipeline avec plusieurs environnements

Ce type de pipeline comprend un ou plusieurs environnements de test. Cela permet de tester le code dans des conditions plus proches de la production avant de le déployer. Il permet des tests dans des environnements distincts, ce qui réduit les risques lors du déploiement en production mais peut être plus complexe à mettre en place et à gérer en raison des multiples environnements.

Pipeline avec déploiement bleu-vert

Ce type de pipeline utilise deux environnements de production, un environnement bleu et un environnement vert. Il réduit les temps d'arrêt car une version stable est toujours disponible pour les utilisateurs mais nécessite une infrastructure supplémentaire pour maintenir deux environnements de production simultanément [4].

1.2 Outils et technologies

1.2.1 Plateformes de CI/CD

Les plateformes de CI/CD (Continuous Integration/Continuous Deployment) sont des outils logiciels qui automatisent les étapes du processus de développement logiciel, de l'intégration continue au déploiement continu. Elles permettent de construire, tester, valider et déployer les applications de manière efficace et cohérente.

Elles fournissent des fonctionnalités telles que l'intégration avec les systèmes de contrôle de version, l'exécution de tests automatisés, la gestion des dépendances, la création d'artefacts et la configuration des pipelines de déploiement. Elles permettent également d'intégrer d'autres outils et services, tels que des services de cloud computing, des outils de test et des outils de gestion des configurations.
Quelques plateformes de CI/CD populaires sont Jenkins, Azure DevOps, GitLab CI et GitHub Actions.

1.2.2 Technologies de conteneurisation

Les technologies de containerisation sont des outils et des frameworks qui permettent d'emballer des applications et leurs dépendances dans des conteneurs isolés. Les conteneurs fournissent un environnement léger et portable, permettant d'exécuter des applications de manière cohérente sur différentes plateformes.
Quelques technologies de conteneurisation populaires sont Docker, Podman et OpenVZ.

1.2.3 Outils de versionnement

Les outils de versionnement, également appelés systèmes de contrôle de version (VCS), sont des logiciels utilisés pour gérer les différentes versions d'un ensemble de fichiers, généralement dans le contexte du développement logiciel. Ils permettent de suivre les modifications apportées aux fichiers au fil du temps, de gérer les branches de développement et de faciliter la collaboration entre les membres d'une équipe de développement.
Quelques outils de versionnement sont Git, Mercurial et Perforce.

1.2.4 Outils de test

Les outils de test sont des outils ou encore des frameworks utilisés pour exécuter des tests sur des logiciels ou des applications informatiques. Ces outils sont conçus pour automatiser le processus de test, facilitant ainsi la création, l'exécution et la gestion des tests.
Quelques outils de test sont Laravel Dusk, PHPUnit, PestPHP et Cypress.

1.2.5 Outils de communication et de collaboration

Les outils de communication et de collaboration sont des logiciels conçus pour faciliter la communication et la coordination entre les membres d'une équipe de travail. Ils jouent un rôle essentiel dans le développement logiciel en permettant aux membres de l'équipe de rester connectés, de partager des informations et de travailler de manière collaborative et efficace. Ces outils offrent un large éventail de fonctionnalités qui favorisent la communication instantanée, la gestion de projet, le partage de fichiers, la coordination des tâches et bien plus encore.
Quelques outils de communication et de collaboration populaires sont Slack, Jira et Microsoft Teams.

Conclusion

Le DevOps s'est imposé comme une approche incontournable dans le monde du développement logiciel, révolutionnant la manière dont les équipes de développement et d'exploitation collaborent. Les outils DevOps constituent un écosystème varié et en constante évolution, englobant des solutions pour la gestion de code source, l'automatisation des tests, le déploiement continu, la surveillance des performances et bien d'autres aspects du cycle de vie du développement logiciel. Le chapitre suivant porte sur l'étude comparative de différents outils susceptibles de nous aider et le choix de ceux qui sont les plus pertinents dans la mise en oeuvre de notre solution.

Chapitre 2

Analyse et choix techniques

Introduction

Dans ce chapitre, nous présentons l'application sur laquelle nous travaillon, etablissons des comparaisons entre différents outils et sélectionnons sur cette base les meilleurs outils pour construire notre chaine CI/CD. Nous étudions également l'architecture de notre chaine. Cela nous sert de base solide pour pouvoir par la suite construire notre solution et réaliser les tests.

2.1 L'environnement logiciel

2.1.1 Présentation de l'application

L'application Ulearn est un système de gestion d'apprentissage (LMS), conçu avec le framework de développement Laravel, qui comprend toutes les fonctionnalités de base nécessaires à l'apprentissage en ligne. Elle comprend un site web responsive frontend, un tableau de bord instructeur/administrateur avec un modèle backend. L'administrateur peut gérer tous les aspects du système, tandis que les instructeurs peuvent créer des cours créatifs à travers un curriculum interactif avec différents types de fichiers de cours tels que l'audio, la vidéo, le document et le texte. Les utilisateurs quant à eux, peuvent s'inscrire et payer pour suivre des cours et bénéficient d'un système de suivi de leurs progrès dans les différents cours auxquels ils participent.

2.1.2 Fonctionnalités de l'application

- Gestion des utilisateurs tels que les étudiants, les instructeurs et les administrateurs
- Création et gestion du contenu de cours par les instructeurs
- Inscription à des cours par les étudiants
- Gestion des inscriptions et des paiements
- Gestion des évaluations ou des certifications
- Suivi détaillé des progrès des étudiants
- Intégration de ressources externes comme les vidéos et les webinaires
- Blog d'actualités et communautaire

2.2 Choix des composants de l'architecture

Pour chaque étape de la chaîne CI/CD, nous effectuons une comparaison des outils spécifiques en évaluant leurs fonctionnalités, leur convivialité, leur capacité à s'intégrer avec d'autres outils, ainsi que leur compatibilité avec les technologies employées dans le développement de notre application.

2.2.1 Plateformes de CI/CD

La table 2.1 compare quatre plateformes populaires de CI/CD : Jenkins, Azure DevOps, GitLab CI et GitHub Actions. Les critères d'évaluation incluent l'intégration avec les systèmes de contrôle de version, la convivialité de l'interface utilisateur, la prise en charge des environnements multiplateformes, l'extensibilité, la documentation, la communauté, l'intégration avec les outils tiers, le déploiement dans le cloud, les fonctionnalités de sécurité, le prix et l'ancienneté [7] [13] [8].

TABLE 2.1 : Comparaison de quelques plateformes de CI/CD

Critères	**Jenkins**	**Azure DevOps**	**GitLab CI**	**GitHub Actions**
Intégration avec les systèmes de contrôle de version	Très Bonne	Très Bonne	Excellente	Excellente
Convivialité de l'interface utilisateur	Moyenne	Bonne	Bonne	Bonne
Prise en charge des environnements multiplateformes	Bonne	Bonne	Excellente	Bonne
Extensibilité	Excellente	Bonne	Bonne	Bonne
Documentation	Bonne	Bonne	Bonne	Bonne
Communauté	Très Grande	Grande	Grande	Grande
Intégration avec les outils tiers	Bonne	Excellente	Bonne	Bonne
Déploiement dans le cloud	Possible	Oui	Possible	Oui
Fonctionnalités de sécurité	Bonnes	Bonnes	Bonnes	Bonnes
Prix	Open source	Tarification basée sur l'usage	Open source	Inclus dans les frais de Github
Ancienneté	2004	2018	2015	2019

Jenkins se distingue comme le choix idéal en tant que plateforme de CI/CD dans le cadre de notre travail. Grâce à une excellente intégration avec les systèmes de contrôle de version et une extensibilité remarquable, Jenkins offre une flexibilité maximale pour les pipelines d'intégration et de déploiement. Jenkins est l'un des outils les plus utilisés pour l'intégration continue en raison de sa flexibilité et de son large écosystème de plugins [2]. Bien que son interface utilisateur puisse être considérée comme moyenne, sa documentation solide et sa très grande communauté en font une ressource précieuse pour résoudre les problèmes et partager les bonnes pratiques.

2.2.2 Technologies de conteneurisation

La table 2.2 compare trois technologies de conteneurisation populaires : Docker, Podman et OpenVZ. Les critères d'évaluation incluent l'architecture, la prise en charge des systèmes d'exploitation, les images officielles et la communauté, la sécurité, l'intégration avec Kubernetes et le prix [25] [15].

TABLE 2.2 : Comparaison de quelques technologies de conteneurisation

Critères	**Docker**	**Podman**	**Open VZ**
Architecture	Client-serveur	Client-serveur	Hyperviseur
Prise en charge des systèmes d'exploitation	Multi-plateforme	Multi-plateforme	Linux
Images officielles et communauté	Vaste	Grandissante	Limitée
Sécurité	Bonne	Bonne	Bonne
Intégration avec Kubernetes	Excellente	Bonne	Non
Prix	Version community gratuite	Open source	Open source

Docker se démarque comme le choix évident en tant que technologie de conteneurisation dans le cadre de notre travail. Avec son architecture client-serveur bien établie, il offre une prise en charge multiplateforme, des images officielles déjà construite et une communauté vaste.

2.2.3 Outils de versionnement

La table 2.3 compare trois outils de versionnement populaires : Git, Mercurial et Perforce. Les critères d'évaluation incluent le type de système de versionnement, la popularité, la performance, la gestion des branches, la fusion de branches, le support multiplateforme, la prise en charge de grands projets et l'intégration avec des outils tiers [11] [3].

TABLE 2.3 : Comparaison de quelques outils de versionnement

Critères	**Git**	**Mercurial**	**Perforce**
Type de système de versionnement	Décentralisé	Décentralisé	Centralisé
Popularité	Très populaire	Moins populaire	Populaire
Performance	Rapide	Moins rapide	Rapide
Fusion de branches	Puissante	Moins puissante	Moins puissante
Support multiplateforme	Oui	Oui	Oui
Prise en charge de grands projets	Oui	Oui	Oui
Intégration avec des outils tiers	Excellente	Bonne	Bonne

Git se démarque comme le choix incontesté pour le versionnement dans notre environnement. En tant que système décentralisé extrêmement populaire, il offre une performance plus rapide et une fusion de branches

plus puissante que les autres outils. Avec un support multiplateforme et une intégration exceptionnelle avec des outils tiers, Git garantit une collaboration fluide et efficace, et convient parfaitement à notre projet.

2.2.4 Outils de test

La table '2.4 compare quatre outils de test populaires pour le développement PHP : Laravel Dusk, PHPUnit, PestPHP et Cypress. Les critères d'évaluation incluent le type de test, la facilité d'utilisation, l'intégration avec Laravel, les fonctionnalités avancées, et l'environnement DevOps [23] [20] [22].

TABLE 2.4 : Comparaison de quelques outils de test

Critères	**Laravel Dusk**	**PHPUnit**	**PestPHP**	**Cypress**
Type de test	Intégration, bout en bout	Unitaire, intégration	Unitaire, intégration	Intégration, bout en bout
Facilité d'utilisation	Excellente	Bonne	Excellente	Excellente
Intégration avec Laravel	Excellente	Bonne	Bonne	Moyenne
Fonctionnalités avancées	Limitées	Excellentes	Bonnes	Excellentes
Environnement DevOps	Bonne	Excellente	Bonne	Bonne

Laravel Dusk se distingue comme le choix optimal pour les tests dans le cadre de notre application. Avec une facilité d'utilisation remarquable et une intégration parfaite avec Laravel, Dusk offre des fonctionnalités avancées pour des tests d'intégration et de non régression robustes, répondant ainsi précisément à nos besoins en matière de tests.

2.2.5 Outils de communication et de collaboration

La table 2.5 compare trois outils populaires de communication et de collaboration : Jira, Slack et Trello. Les critères d'évaluation incluent les fonctionnalités de communication, les intégrations avec les outils DevOps, la convivialité de l'interface et la présence de la messagerie en temps réel [26] [5].

TABLE 2.5 : Comparaison de quelques outils de communication et de collaboration

Critères	**Jira**	**Slack**	**Microsoft Teams**
Fonctionnalités de communication	Moins axé	Centré	Centré
Intégrations avec les outils DevOps	Bonnes	Excellentes	Bonnes
Convivialité de l'interface	Complexe	Intuitive	Intuitive
Messagerie en temps réel	Non	Oui	Oui

Slack se démarque comme le choix idéal pour la communication dans le cadre de notre travail. Son focus sur la messagerie en temps réel et ses excellentes intégrations avec les outils DevOps en font un outil central pour la collaboration instantanée et efficace. De plus, son interface intuitive garantit une adoption rapide par les équipes, favorisant ainsi une productivité accrue.

2.3 Architecture de la chaîne CI/CD

2.3.1 Détails des étapes du pipeline

Pour mettre en place notre chaine CI/CD, il est important de suivre une séquence chronologique dans l'utilisation des différents outils précédemment choisis. Nous avons fait le choix d'un pipeline avec plusieurs environnements.
Voici une description étape par étape de la manière dont ces outils peuvent être orchestrés pour réussir à implémenter notre solution.

1. **Gestion de version** : La première étape consiste à mettre en place un système de contrôle de version tel que Git. Il permet de gérer les modifications du code source et de collaborer efficacement sur le projet. Les développeurs travaillent sur des branches distinctes pour ajouter de nouvelles fonctionnalités ou corriger des bogues.
2. **Intégration continue** : Une fois que les développeurs ont effectué leurs modifications, celles-ci sont intégrées dans la branche de développement principale. Notre plateforme de CI/CD, Jenkins, est utilisé pour automatiser la compilation, les tests unitaires, de non régression et l'intégration du code.
3. **Construction de l'artefact** : Ici, nous allons utiliser Docker pour construire l'image de base de l'application.
4. **Publication de l'image** : Nous allons publier l'image ainsi construite sur un dépot distant, Dockerhub.
5. **Déploiement en staging** : Nous allons déployer l'application dans un environnement de test similaire à l'environnement de production.
6. **Tests automatisés** : Une fois l'application déployée, nous allons utiliser Laravel Dusk pour effectuer des tests système, de non régression et d'interface utilisateur sur notre application.
7. **Déploiement en production** : Nous allons utiliser Docker pour déployer l'application de manière automatisée. Cet outil permet des conteneurs à partir de l'image de l'application précédemment publiée.
8. **Communication et collaboration** : Tout au long de ce processus, nous allons utiliser Slack pour faciliter la communication entre les membres de l'équipe, partager des informations sur les tâches en cours, signaler les problèmes ou demander des modifications.

2.3.2 Intégration avec le LMS ulearn

2.3.2.1 Déclenchement des pipelines

- Les pipelines sont déclenchés automatiquement lors de la fusion de code dans la branche principale de développement.
- Dans le cas de tests ou de certains correctifs, le pipeline peut etre déclenché manuellement.

2.3.2.2 Déploiement des nouvelles versions

- Déploiement automatisé des nouvelles versions vers l'environnement de test après la réussite des tests.
- Déploiement automatisé des nouvelles versions vers l'environnement de production après validation en environnement de test.

Conclusion

Ce chapitre nous a permis de clairement définir les limites de notre application, d'élaborer des comparaisons et de motiver les choix des meilleurs outils dans le cadre de notre travail. Cela nous a permis d'établir la liste des outils à utiliser et d'élaborer le plan de construction de la pipeline. Le chapitre suivant porte sur la mise en oeuvre proprement dite.

Chapitre 3

Solutions et Discussions

Introduction

Dans ce chapitre, nous présentons notre infrastructure ainsi que les diverses phases de la mise en œuvre de notre solution. Nous montrons également les tests réalisés pour garantir le bon fonctionnement de la solution. Enfin, nous abordons en détail les résultats obtenus à la suite de cette implémentation.

3.1 Mise en place des outils

3.1.1 Présentation de l'infrastructure

Dans le cadre de notre travail, nous utilisons deux serveurs, des serveurs virtuels privés (VPS) proposés par AWS. Il s'agit d'un serveur pour notre environnement de test où sera également exécuté Jenkins et un deuxième pour notre environnement de production.
La table 3.1 présente les caractéristiques des serveurs que nous utilisons pour notre travail. Il détaille le système d'exploitation, le nombre de processeurs, la quantité de mémoire vive (RAM) et la capacité de stockage de chaque serveur.

TABLE 3.1 : Caractéristiques des serveurs

Serveurs	OS	Processeurs	RAM	Capacité disque
Serveur de test/Jenkins	Ubuntu 22.04	2 vCPU	4 Go	24 Go
Serveur de production	Ubuntu 22.04	1 vCPU	1 Go	12 Go

Le serveur de test/Jenkins dispose, comme exposé dans la table 3.1 de ressources plus importantes que le serveur de production. C'est parce que le serveur de test est utilisé pour des tâches plus gourmandes en ressources, telles que l'exécution de builds et de tests ainsi que l'exécution permanente de Jenkins.
Les deux serveurs utilisent Ubuntu 22.04 comme système d'exploitation. Il s'agit d'une distribution Linux stable et largement utilisée pour les serveurs.

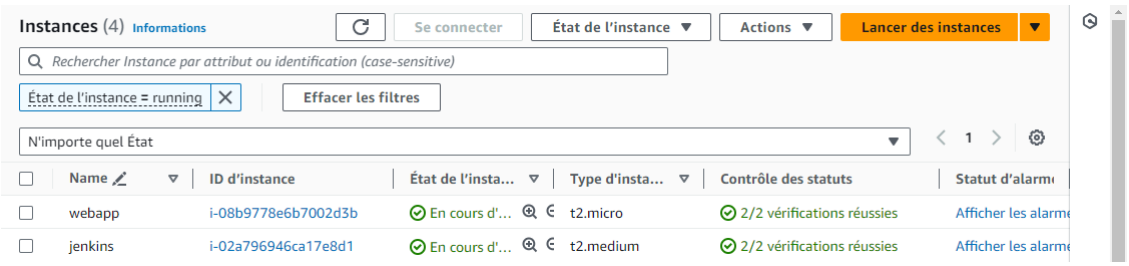


FIGURE 3.1 : Serveurs AWS

3.1.2 Jenkins

3.1.2.1 Installation de Jenkins

Pour l'installation de Jenkins sur le serveur de test, nous avons, tout d'abord, ajouté la clé du dépôt à votre système. Ensuite, nous avons ajouté l'adresse du dépôt de paquets Debian à la liste des sources du serveur et exécuté **apt update** pour qu'**apt** utilise le nouveau dépôt. Enfin, nous avons lancé l'installation de Jenkins et ses dépendances.

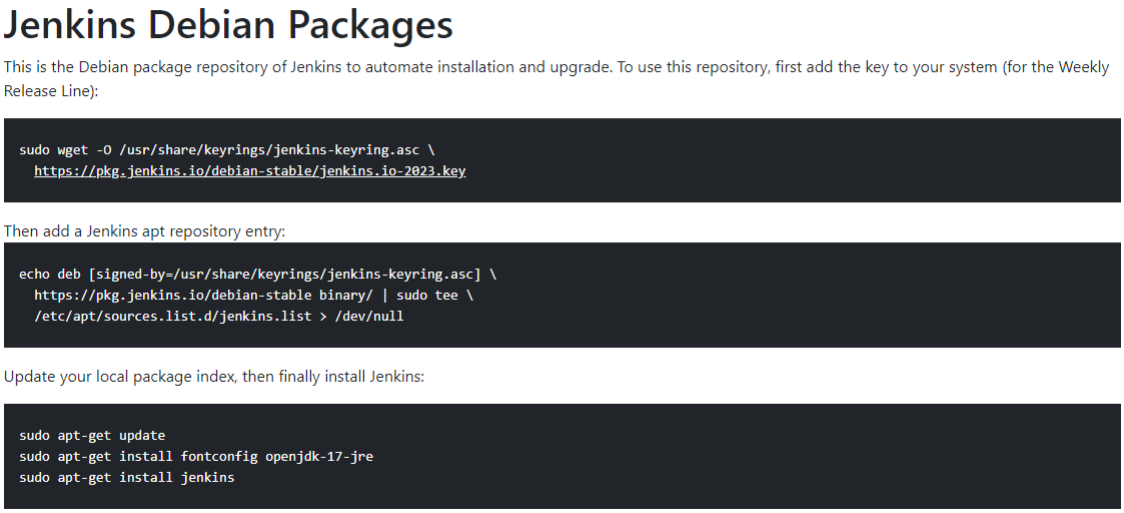


FIGURE 3.2 : Commandes d'installation de Jenkins [16]

Une fois l'installation terminée, nous l'avons démarré et vérifié s'il tournait effectivement bien avec **systemctl**.

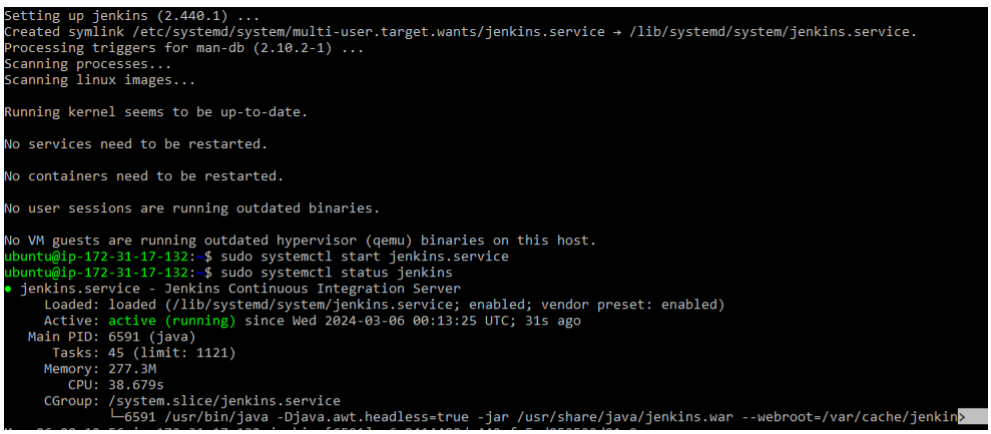


FIGURE 3.3 : Installation de Jenkins réussie

3.1.2.2 Configuration initiale de Jenkins

Pour configurer Jenkins, nous avons visité Jenkins sur son port par défaut, **8080**, en utilisant l'adresse du serveur de Jenkins. Nous avons vu l'écran Unlock Jenkins, qui affiche l'emplacement du mot de passe initial et avons entré le mot de passe pour continuer.



FIGURE 3.4 : Ecran Unlock Screen de Jenkins

Nous avons installé les plugins par défaut et créé un nouveau compte administrateur.



FIGURE 3.5 : Plugins par défaut de Jenkins



FIGURE 3.6 : Formulaire de création d'un administrateur Jenkins

Enfin, nous avons eu l'interface d'accueil de Jenkins.

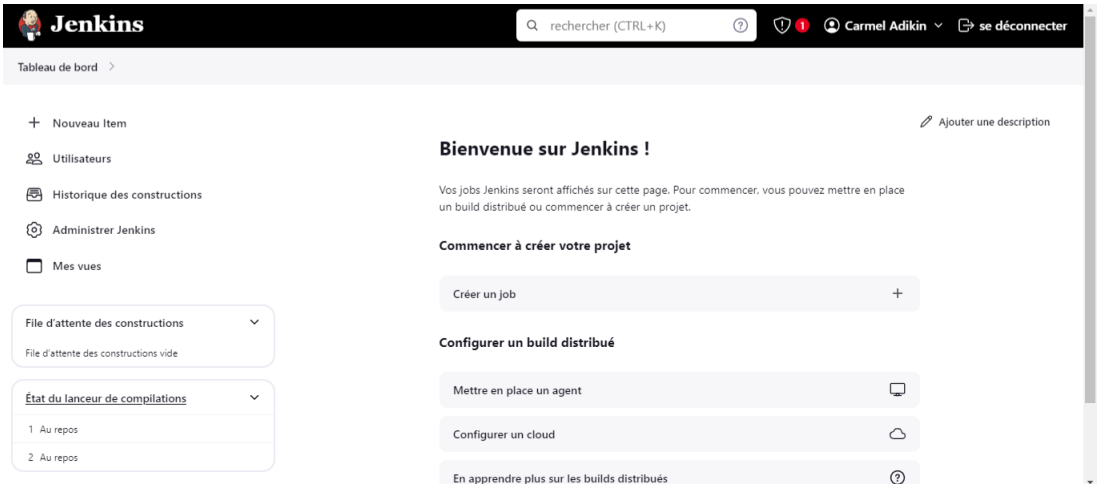


FIGURE 3.7 : Ecran d'accueil de Jenkins

3.1.2.3 Plugins de Jenkins

Nous avons eu besoin de quelques plugins supplémentaires de Jenkins pour mener à bien notre travail.
SSH Agent : nous avons utilisé ce plugin pour pouvoir gérer les informations de connexion par SSH de Jenkins à d'autres serveurs en toute confidentialité. Cela nous a permis de manière sécurisée de pouvoir se connecter au serveur de production et y effectuer des tâches.

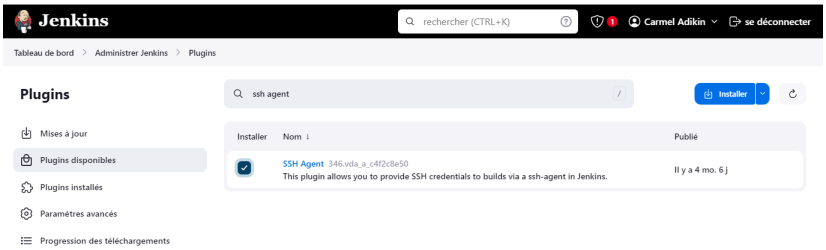


FIGURE 3.8 : Plugin SSH Agent

Blue Ocean : Ce plugin nous a permis d'avoir une interface utilisateur plus conviviale et plus intuitive pour la gestion de notre pipeline. Notre travail s'en est trouvé facilité.

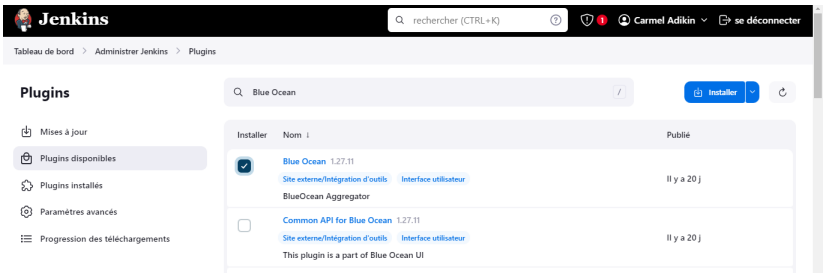


FIGURE 3.9 : Plugin Blue Ocean

Global Slack Notifier : Ce plugin nous a permis de pouvoir envoyer des notifications dans la chaine Slack du projet à certaines étapes clés de notre pipeline.

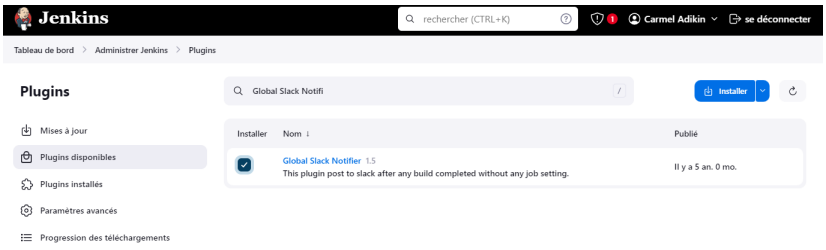


FIGURE 3.10 : Plugin Global Slack Notifier

3.1.3 Docker

Nous avons d'abord opéré un nettoyage pour supprimer des anciennes versions éventuelles de Docker déjà installées sur nos deux serveurs. Ensuite, nous avons mis à jour l'index des paquets apt et installé des paquets pour permettre à apt d'utiliser un dépôt via HTTPS.



FIGURE 3.11 : Commandes Docker pour mettre à jour l'index des paquets [9]

Par la suite, nous avons ajouté la clé GPG officielle de Docker et ajouté l'adresse du dépôt de paquets Debian à la liste des sources du serveur.



FIGURE 3.12 : Ajout de l'adresse de dépôt de paquets Debian [9]

Nous avons, ensuite, mis à jour l'index des paquets. Enfin, nous avons installé Docker et ses dépendances.



FIGURE 3.13 : Installation de Docker [9]

Une fois l'installation terminée, nous avons vérifié si elle a été un succès en exécutant l'image Docker **hello-world**.

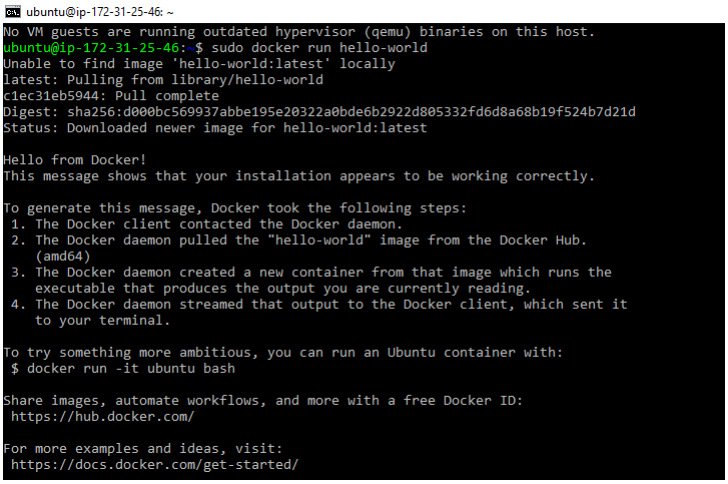


FIGURE 3.14 : Installation de Docker réussie.

3.2 Ecriture des tests

Nous allons écrire nos tests et les exécuter à l'aide de **Laravel Dusk**. Laravel Dusk est un outil de test d'intégration et d'automatisation de navigateur pour les applications Laravel. Il permet d'effectuer des tests de bout en bout de manière simple et expressive.
Nous l'avons utilisé pour effectuer des tests de non regression sur notre application afin de pouvoir déployer une application totalement fiable. Nous avons testé les fonctionnalités suivantes :

- Créer un instructeur en tant qu'admin.
- Se connecter en tant qu'instructeur
- S'inscrire en tant qu'étudiant

3.2.1 Exigence des tests

3.2.1.1 Critères d'acceptation

- Les fonctionnalités existantes doivent fonctionner sans erreur.
- Les nouvelles fonctionnalités doivent fonctionner sans causer de régressions.
- Les performances doivent rester dans les limites acceptables (temps de réponse < 2 secondes).

3.2.1.2 Procédures de Test

1. Initialiser la base de données de test.
2. Lancer l'application dans l'environnement de test.
3. Exécuter les scénarios de test en utilisant Laravel Dusk.

3.2.1.3 Suivi des Résultats

- Les résultats des tests sont affichés en console pour un suivi en temps réel.
- Les résultats peuvent être consultés dans les logs des différentes exécutions du pipeline.
- Pour les tests automatisés, une notification Slack est envoyée par le pipeline indiquant le succès ou l'échec des tests.

3.2.2 Scénarios de test

3.2.2.1 Créer un instructeur en tant qu'administrateur

Ici, nous simulons d'abord la connexion d'un administrateur en utilisant les identifiants de l'administrateur de test. Une fois connecté, nous allons à la page d'ajout d'un nouvel utilisateur, entrons les informations nécessaires à la création et soumettons le formulaire. Une fois que c'est fait, nous visitons la page qui liste les utilisateurs de notre site pour vérifier si l'utilisateur précédemment créé est dans la liste.

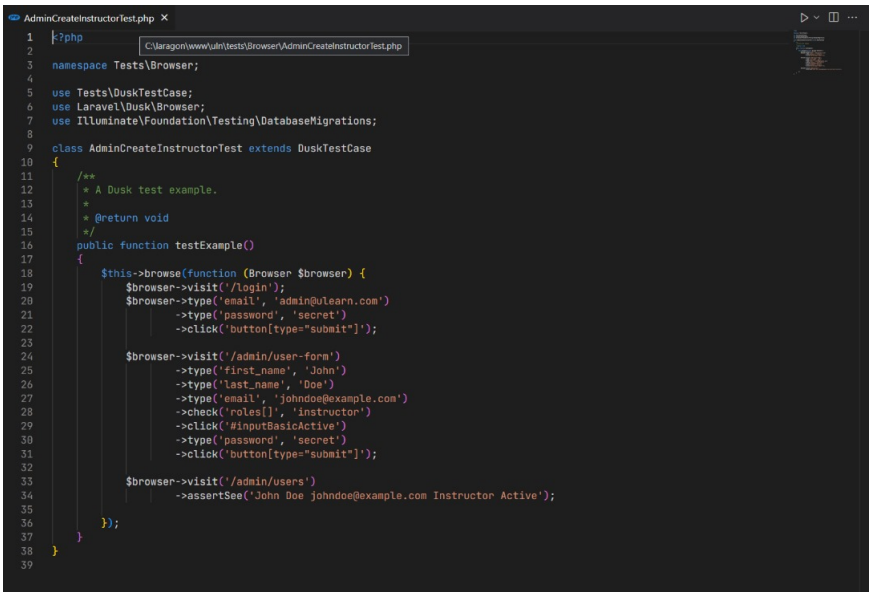


FIGURE 3.15 : Fichier Dusk AdminCreateInstructor.php

3.2.2.2 Se connecter en tant qu'instructeur

Nous ouvrons d'abord la page de connexion, puis nous saisissons les identifiants de l'instructeur de test et soumettons le formulaire de connexion. Enfin, nous vérifions que l'utilisateur est redirigé vers le tableau de bord prévu pour les comptes instructeur.

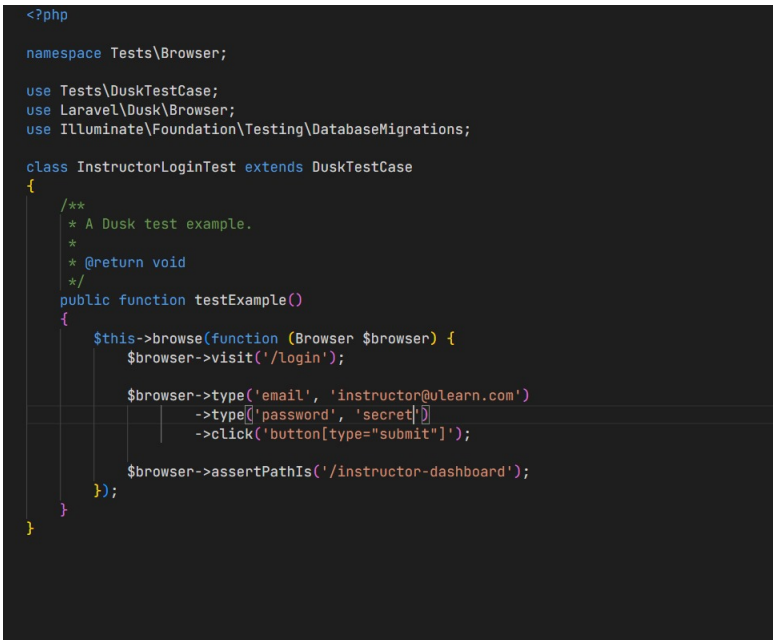


FIGURE 3.16 : Fichier Dusk InstructorLoginTest.php

3.2.2.3 S'inscrire en tant qu'étudiant

Nous ouvrons d'abord la page d'inscription, puis nous remplissons le formulaire d'inscription avec les informations de l'utilisateur (prénom, nom, adresse e-mail, mot de passe). Ensuite, nous soumettons le formulaire d'inscription. Après cela, nous ouvrons la page d'accueil et vérifions que l'utilisateur est connecté en vérifiant la présence de son prénom dans le coin supérieur droit, comme prévu lors du développement de l'application.

```php
<?php

namespace Tests\Browser;

use Tests\DuskTestCase;
use Laravel\Dusk\Browser;
use Illuminate\Foundation\Testing\DatabaseMigrations;

class UserRegister extends DuskTestCase
{
    /**
     * A Dusk test example.
     *
     * @return void
     */
    public function testExample()
    {
        $this->browse(function (Browser $browser) {
            $browser->visit('/register')
                    ->type('first_name', 'Jane')
                    ->type('last_name', 'Doe')
                    ->type('email', 'janedoe@example.com')
                    ->type('password', 'secret')
                    ->type('password_confirmation', 'secret')
                    ->click('button[type="submit"]');

            $browser->visit('/')
                    ->assertSee('Jane');


        });
    }
}
```

FIGURE 3.17 : Fichier Dusk UserRegister.php

3.3 Construction du pipeline

Pour mettre en place notre pipeline avec Jenkins et le faire s'éxécuter de manière complètement automatisée, nous avons écrit un **Jenkinsfile**.
Un **Jenkinsfile** est un fichier texte écrit dans le langage Groovy qui définit un pipeline de livraison continue (CI/CD) dans Jenkins. Il décrit les étapes à suivre pour construire, tester, déployer et versionner notre application.

3.3.1 Intégration continue

Nous récupérons le code de l'application Ulearn depuis le dépot distant Github à l'aide de git. Nous construisons notre image Docker de base de l'application.

Script ?

```
pipeline {
    agent any

    stages {
        stage('Checkout Code') {
            steps {
                git 'https://github.com/CarmelAdk/ulearn'
                echo 'Git Checkout Completed'
            }
        }

        stage('Build Docker Image') {
            steps {
                sh 'docker build --file .docker/app/Dockerfile -t carmeladikin/lms:1.$BUILD_NUMBER .'
                echo 'Build Image Completed'
            }
        }
```

FIGURE 3.18 : Jenkinsfile, partie CI

L'image construite est sur la base d'un **Dockerfile** que nous avons écrit.
Un **Dockerfile** est un fichier texte qui contient des instructions pour construire une image Docker. Il s'agit d'un ensemble de commandes qui définissent l'environnement d'exécution de votre application, y compris les logiciels à installer, les configurations à effectuer et les actions à exécuter au démarrage du conteneur.

Nous nous connectons au Dockerhub et y publions l'image versionnée ainsi construite.



FIGURE 3.19 : Jenkinsfile, partie CI

Nous avons également envoyé des notifications Slack informant du succès ou de l'échec de la publication de l'image sur le Docker Hub.



FIGURE 3.20 : Jenkinsfile, notifications slack pour publication

On peut constater que les notifications sont bien envoyées.

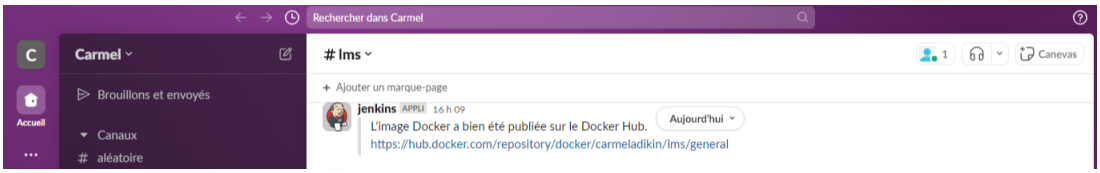


FIGURE 3.21 : Slack, notification Docker Hub

Et que l'image a bien été publiée sur le Docker Hub.

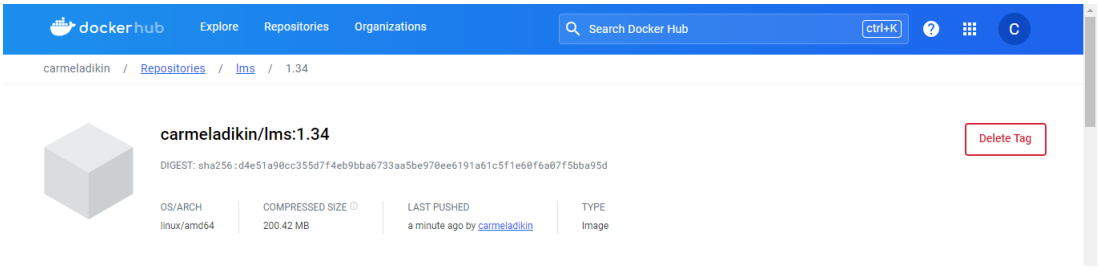
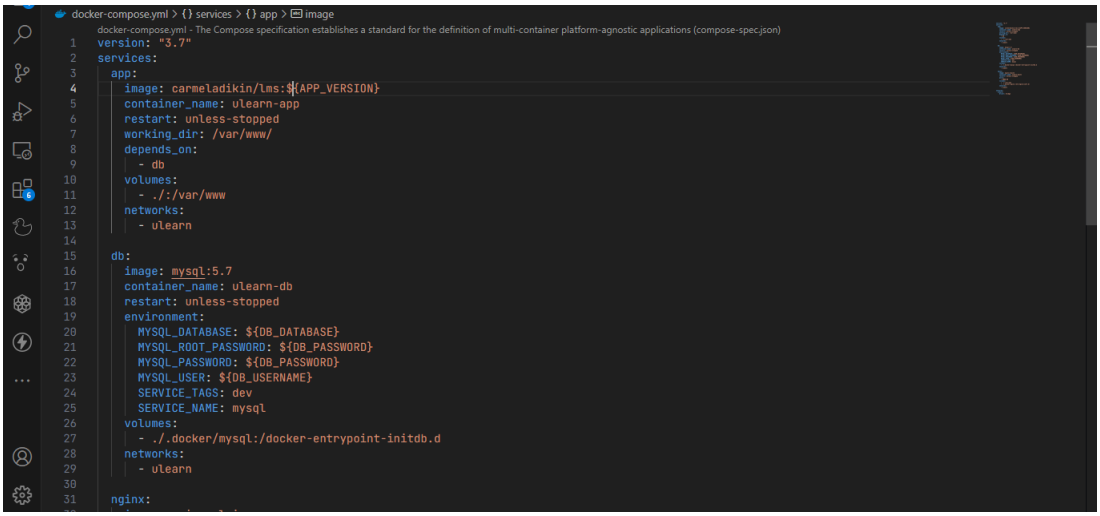


FIGURE 3.22 : Image de l'application sur le Docker Hub

3.3.2 Déploiement continu

Une fois l'image construite, nous déployons l'application dans un environnement de test où les tests écrits précédemment doivent être exécutés. Nous utilisons un fichier **docker-compose.yml** comme base pour le déploiement.
Un fichier **docker-compose.yml** est un fichier de configuration, écrit en YAML, qui permet de définir et de gérer plusieurs conteneurs Docker en tant qu'application unique. Il décrit les services qui composent notre application, leurs dépendances et les configurations réseau et de volume.

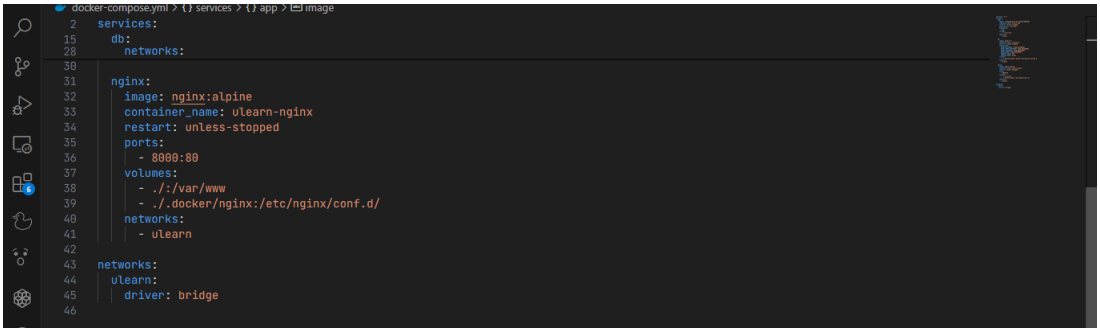


```yaml
version: "3.7"
services:
  app:
    image: carmeladikin/lms:${APP_VERSION}
    container_name: ulearn-app
    restart: unless-stopped
    working_dir: /var/www/
    depends_on:
      - db
    volumes:
      - ./:/var/www
    networks:
      - ulearn

  db:
    image: mysql:5.7
    container_name: ulearn-db
    restart: unless-stopped
    environment:
      MYSQL_DATABASE: ${DB_DATABASE}
      MYSQL_ROOT_PASSWORD: ${DB_PASSWORD}
      MYSQL_PASSWORD: ${DB_PASSWORD}
      MYSQL_USER: ${DB_USERNAME}
      SERVICE_TAGS: dev
      SERVICE_NAME: mysql
    volumes:
      - ./.docker/mysql:/docker-entrypoint-initdb.d
    networks:
      - ulearn

  nginx:
```

FIGURE 3.23 : Fichier docker-compose.yml



```yaml
services:
  db:
    networks:

  nginx:
    image: nginx:alpine
    container_name: ulearn-nginx
    restart: unless-stopped
    ports:
      - 8000:80
    volumes:
      - ./:/var/www
      - ./.docker/nginx:/etc/nginx/conf.d/
    networks:
      - ulearn

networks:
  ulearn:
    driver: bridge
```

FIGURE 3.24 : Fichier docker-compose.yml

Nous déployons notre application dans notre environnement de test et effectuons les tests de non régression que nous avons écrits.



Script

```
    stage('Deploy to test environment') {
        steps {
            sh "sed -i -E 's|(APP_URL=).*|APP_URL=http://localhost:8000|' .env"
            sh "composer update"

            sshagent(['test-server']) {
                sh '''
                    ssh -tt -o StrictHostKeyChecking=no ubuntu@localhost "cd ulearn; git pull; sed -i -E 's|(APP_VERSION=).*|APP_VER
                '''
            }
            echo 'Deployed to test environment'

            sh 'php artisan dusk:install; rm -f tests/Browser/ExampleTest.php'
            sh 'vendor/bin/phpunit tests/Browser'

            echo 'App tested !'
```

FIGURE 3.25 : Jenkinsfile, étape déploiement en test

Ensuite, nous configurons également notre pipeline pour que des notifications Slack soient envoyés informant du succès ou de l'échec du déploiement.

Script ?

```
    }

    success {
        slackSend channel: 'lms', message: 'Les tests ont tous réussi !', tokenCredentialId: 'slack'
    }
    failure {
        slackSend channel: 'lms', message: 'Il y a un problème lors du déploiement en staging ou lors des tests.', tokenCredentialId:
    }
}
```

FIGURE 3.26 : Jenkinsfile, étape notifications Slack après déploiement et test en staging

On peut constater que la notification a bien été envoyée.

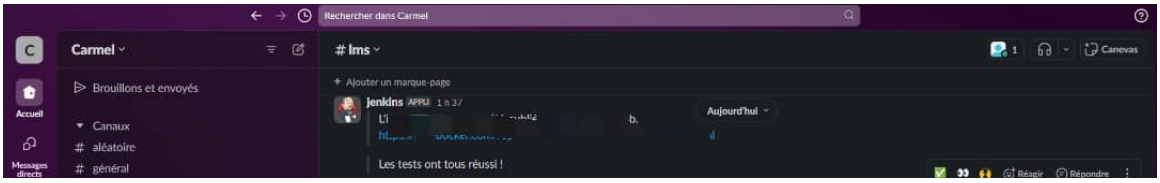


FIGURE 3.27 : Slack, notification test

Et que l'application a bien été déployée et testée.

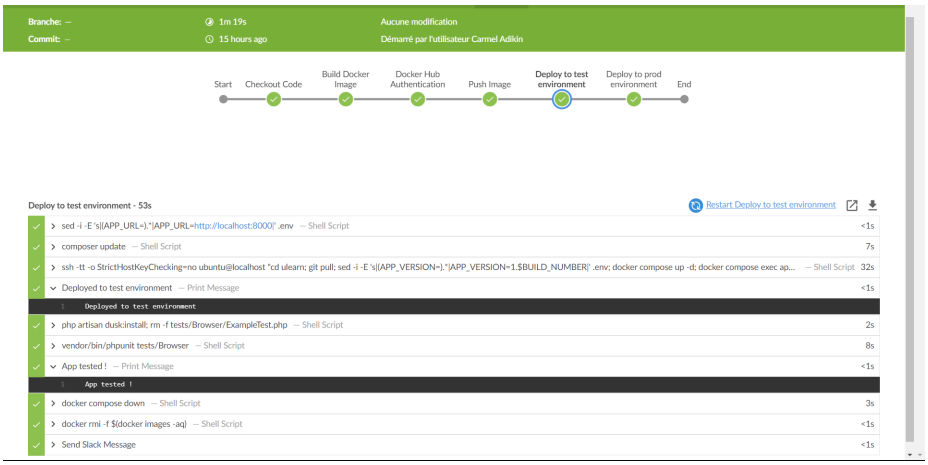


FIGURE 3.28 : Etape de test réussie

Nous allons déployer notre application en production. Jenkins va se connecter par SSH à notre serveur de production pour y effectuer les différentes actions.

Script ?

```
    }

    stage('Deploy to prod environment') {
        steps {
            sshagent(['prod-server']) {
                sh '''
                    ssh -tt -o StrictHostKeyChecking=no ubuntu@ec2-18-217-182-176.us-east-2.compute.amazonaws.com
                    "cd ulearn; git pull;
                    sed -i -E 's|(APP_VERSION=).*|APP_VERSION=1.$BUILD_NUMBER|' .env; docker compose up -d;
                    docker compose exec app composer update;
                    docker compose exec app php artisan migrate; docker compose exec app php artisan db:seed"
                '''
            }
        }
    }
}
```

FIGURE 3.29 : Jenkinsfile, étape déploiement en production

Ensuite, nous configurons également notre pipeline pour que des notifications Slack soient envoyés informant du succès ou de l'échec du déploiement comme pour l'étape de test.



FIGURE 3.30 : Jenkinsfile, étape notifications Slack après déploiement en production

On peut constater que les notifications sont bien envoyées.

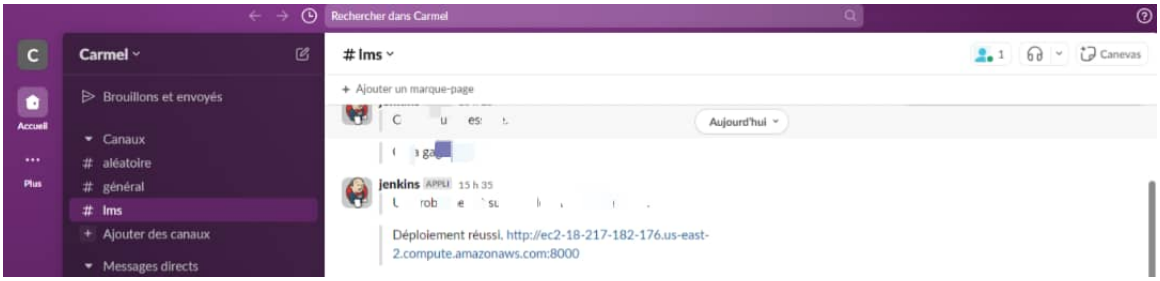


FIGURE 3.31 : Slack, notification déploiement

Et que l'application est bel et bien déployée.

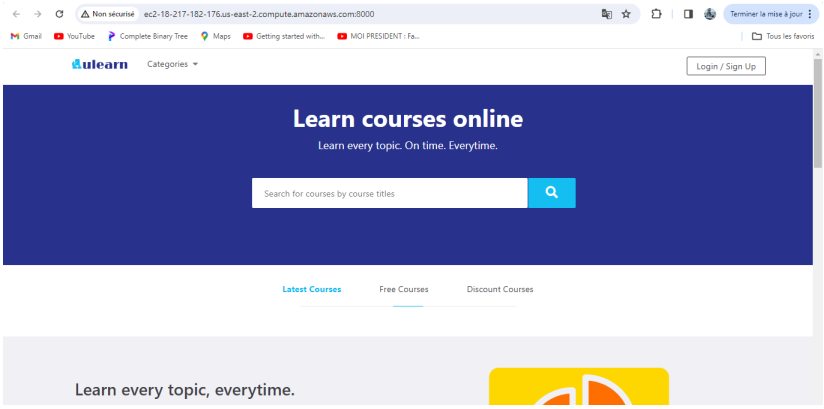


FIGURE 3.32 : Application Ulearn, en ligne

Nous pouvons d'ailleurs voir via l'UI fournie par le plugin Blue Ocean que toutes les étapes de notre pipeline se sont bien déroulées.

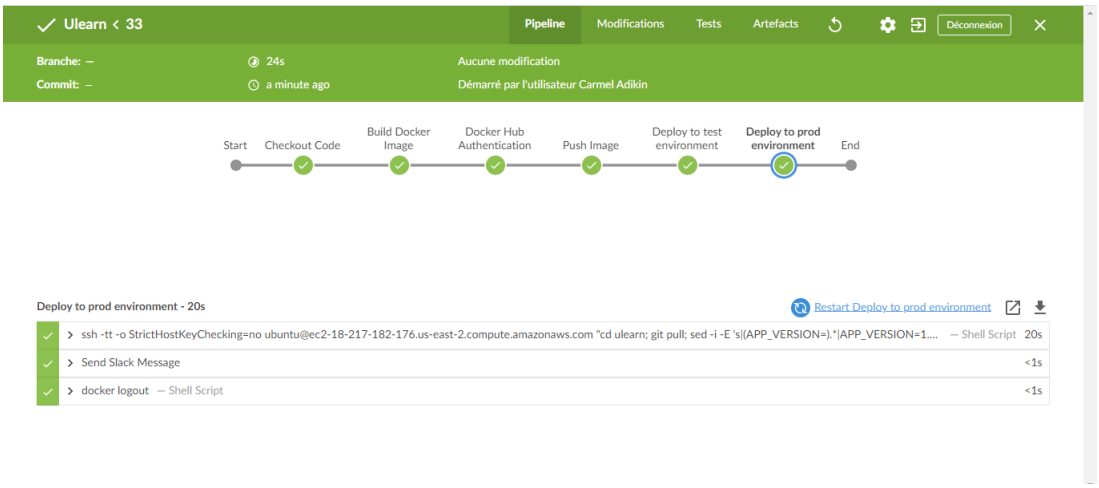


FIGURE 3.33 : Pipeline, au vert

3.4 Résultats des tests

3.4.1 Tests manuels

Avant d'intégrer cette étape dans la chaine CI/CD, nous avons exécuté nos tests manuellement tour à tour chacun. Eux tous ont été un succès.

```
ubuntu@ip-172-31-25-46:~/ulearn$ vendor/bin/phpunit tests/Browser/AdminCreateInstructor.php
PHPUnit 7.5.20 by Sebastian Bergmann and contributors.

.                                                                   1 / 1 (100%)

Time: 3.17 seconds, Memory: 18.00 MB

OK (1 test, 1 assertion)
```

FIGURE 3.34 : Résultat pour le test de création d'un instructeur

```
ubuntu@ip-172-31-25-46:~/ulearn$ vendor/bin/phpunit tests/Browser/InstructorLoginTest.php
PHPUnit 7.5.20 by Sebastian Bergmann and contributors.

.                                                                   1 / 1 (100%)

Time: 1.97 seconds, Memory: 18.00 MB

OK (1 test, 1 assertion)
```

FIGURE 3.35 : Résultat pour la connexion d'un instructeur

```
ubuntu@ip-172-31-25-46:~/ulearn$ vendor/bin/phpunit tests/Browser/UserRegister.php
PHPUnit 7.5.20 by Sebastian Bergmann and contributors.

.                                                                   1 / 1 (100%)

Time: 2.23 seconds, Memory: 18.00 MB

OK (1 test, 1 assertion)
```

FIGURE 3.36 : Résultat pour l'inscription d'un utilisateur

Fort de ce succès, nous avons donc pu intégrer ces tests dans notre chaine CI/CD afin d'automatiser leur exécution.

3.4.2 Tests automatisés

Nous pouvons remarquer que les tests se sont bien déroulés au niveau du pipeline.



FIGURE 3.37 : Résultats des tests automatisés

De nouveaux tests peuvent être écrits au fur et à mesure de l'intégration de nouvelles fonctionnalités. Les anciens tests continueront à automatiquement s'effectuer à chaque exécution de notre pipeline afin d'assurer la non régression de l'application.

3.5 Discussion

En usant des bonnes pratiques DevOps et des outils les plus adaptés, nous avons pu proposer une solution fiable pour accélérer accélérer les livraisons de fonctionnalités, améliorer la qualité du logiciel et mieux répondre aux besoins des utilisateurs. Ce projet nous a permis de consolider nos connaissances et d'améliorer notre maitrise du mouvement DevOps et notre capacité à choisir les outils plus adaptés à un contexte précis. Nous avons également pu implémenter des tests de non régression qui ont donné des résultats positifs. Conscients des limites actuelles du projet, nous sommes déterminés à le perfectionner en améliorant l'automatisation de nos tâches et en adoptant des pratiques DevOps plus matures.

Conclusion

Ce chapitre nous a permis de mettre en place notre chaine CI/CD en installant et configurant les outils précédemment choisis et en écrivant le Jenkinsfile qui nous a permis d'implémenter les différentes étapes de notre chaine.Nous avons écrit et exécuté des tests de non régression sur des fonctionnalités critiques de l'application pour nous assurer de son bon fonctionnement.

Conclusion Générale et Perspectives

Ce mémoire a abouti à la mise en place réussie d'une chaîne CI/CD pour l'application pédagogique Ulearn. Cette chaîne automatise efficacement les processus de test, d'intégration et de déploiement, contribuant ainsi à l'amélioration de la qualité du code, à l'accélération du développement et à la fiabilité accrue de l'application. Les principaux résultats obtenus incluent la création d'une architecture CI/CD utilisant Jenkins et Docker, l'automatisation des tests de non régression, ainsi que le déploiement automatisé de l'application en production. Ce travail a également permis une réduction du temps de développement grâce à l'automatisation des tâches répétitives et une meilleure fiabilité de l'application grâce à des tests et des déploiements plus rigoureux. Nous avons également mis en place un environnement de communication et de collaboration conforme à l'esprit DevOps en intégrant un outil de collaboration dans notre chaine. En outre, ce mémoire a permis d'approfondir les connaissances sur les concepts fondamentaux de DevOps, d'intégration continue et de déploiement continu, ainsi que sur les outils et technologies utilisés pour mettre en place une chaîne CI/CD.
Pour les perspectives futures, il est envisagé d'intégrer des tests de performance dans la chaîne CI/CD et de mettre en place une infrastructure de monitoring pour surveiller l'application en production. Ainsi, les problèmes survenus en production seront plus rapidement remontés aux développeurs permettant une accélération plus accrue du développement.

Bibliographie

[1] G. Kim, P. Debois, J. Willis, J. Humble, and J. Allspaw. The devops handbook : How to create world-class agility, reliability, and security in technology organizations, 2016.

[2] B. Smith. Devops for the desperate : A hands-on survival guide, 2022.

Webographie

[3] Assembla. Perforce vs git : A comparison. https://get.assembla.com/blog/perforce-vs-git/. Consulté le : 04 juillet 2024.

[4] AWS. Blue/green deployments. https://docs.aws.amazon.com/whitepapers/latest/overview-deployment-options/bluegreen-deployments.html. Consulté le : 25 Octobre 2023.

[5] ByteScout. Trello vs jira vs slack : A comparison. https://bytescout.com/blog/trello-vs-jira-vs-slack.html. Consulté le : 03 juillet 2024.

[6] Combell. Devops, pourquoi c'est si important. https://www.combell.com/fr/blog/devops-pourquoi-si-important/. Consulté le : 18 Août 2023.

[7] DataScientest. Azure devops vs github actions : Which is the best ci/cd tool ? https://datascientest.com/en/azure-devops-vs-github-actions-which-is-the-best-ci-cd-tool. Consulté le : 30 Juin 2024.

[8] Deviniti. Devops tools comparison : Gitlab vs azure devops. https://deviniti.com/blog/devops/devops-tools-comparison-gitlab-vs-azure-devops/. Consulté le : 30 Juin 2024.

[9] Docker. Install docker engine on ubuntu. https://docs.docker.com/engine/install/ubuntu/. Consulté le : 25 Avril 2023.

[10] ElasticWeb. Wat is ci/cd ? continuous integration en continuous delivery verder uitgelegd. https://www.elasticweb.nl/kennisbank/continuous-integration-en-continuous-delivery-verder-uitgelegd. Consulté le : 30 Août 2023.

[11] FlareCompare. Git vs. mercurial : Which distributed version control system is better for agile development ? https://flarecompare.com/Agile%20Development/Git%20vs.%20Mercurial%20%20Which%20distributed%20version%20control%20system%20is%20better%20for%20Agile%20development/. Consulté le : 04 juillet 2024.

[12] Gartner. The state of devops 2022. https://www.gartner.com/en/information-technology/glossary/devops. Consulté le : 20 Août 2023.

[13] Geekflare. Gitlab ci vs jenkins : Key differences. https://geekflare.com/gitlab-ci-vs-jenkins/. Consulté le : 30 Juin 2024.

[14] F. Group. Why you should implement ci/cd pipelines in development. https://www.fabricgroup.com.au/blog/why-you-should-implement-ci-cd-pipelines-in-development. Consulté le : 25 Septembre 2023.

[15] C. P. IT. Understanding openvz and docker. https://www.cyberplanetit.com/2023/06/Understanding%20OpenVZ%20and%20Dock.html. Consulté le : 01 juillet 2024.

[16] Jenkins. Jenkins debian packages. https://pkg.origin.jenkins.io/debian-stable. Consulté le : 25 Mai 2023.

[17] I. Learning. Top 9 devops collaboration tools. https://www.invensislearning.com/blog/top-devops-collaboration-tools. Consulté le : 20 Septembre 2023.

[18] OpenReplay. What is a ci/cd pipeline ? https://blog.openreplay.com/what-is-a-ci-cd-pipeline/. Consulté le : 02 Septembre 2023.

[19] OxyProxy. Intégration continue et déploiement continu. https://oxyproxy.pro/fr/wiki/continuous-integration-and-continuous-deployment/. Consulté le : 25 Août 2023.

[20] K. Pro. Difference between laravel dusk and phpunit. https://knapsackpro.com/testing_frameworks/difference_between/dusk/vs/phpunit. Consulté le : 02 juillet 2024.

[21] Qrpinternational. Devops c'est quoi ? https://www.qrpinternational.fr/blog/gestion-des-services-informatiques/devops-cest-quoi-definition-devops/. Consulté le : 18 Août 2023.

[22] Reddit. Does anyone here prefer phpunit to pest ? https://www.reddit.com/r/laravel/comments/155cw2e/does_anyone_here_prefer_phpunit_to_pest/. Consulté le : 01 juillet 2024.

[23] Reddit. How does cypress compare to laravel dusk ? https://www.reddit.com/r/laravel/comments/lmn6v1/how_does_cypress_compare_to_laravel_dusk/. Consulté le : 01 juillet 2024.

[24] Scafe. Les 10 avantages de l'intégration continue. https://www.scafe.io/lintegration-continue-en-10-avantages/. Consulté le : 22 Août 2023.

[25] P. Storage. Docker vs. podman : A developer's guide. https://blog.purestorage.com/purely-educational/docker-vs-podman. Consulté le : 01 juillet 2024.

[26] Zapier. Slack vs microsoft teams : Which is better ? https://zapier.com/blog/slack-vs-microsoft-teams/. Consulté le : 02 juillet 2024.

Table des matières

Dédicace **ii**
ii

Remerciements **iii**
iii

Résumé **iv**
iv

Abstract **v**
v

Table des figures **vi**

Liste des tableaux **viii**

Liste des acronymes **ix**

Introduction **1**
1
1 Problématique 1
2 Contexte et justification 1
3 Objectifs et contributions 2
4 Organisation du mémoire 2

1 Etat de l'art **3**
Introduction 3
1.1 Concepts fondamentaux 3
1.1.1 DevOps 3
1.1.1.1 Définition 3
1.1.1.2 Historique 3
1.1.1.3 Principes fondamentaux 3
1.1.2 Intégration continue 4
1.1.2.1 Définition 4
1.1.2.2 Avantages 4
1.1.2.3 Processus 5
1.1.3 Déploiement continu 6
1.1.3.1 Définition 6
1.1.3.2 Différence avec la livraison continue 6
1.1.3.3 Avantages 6
1.1.3.4 Processus 7
1.1.4 Pipeline CI/CD 8
1.1.4.1 Définition 8

1.1.4.2 Types de pipeline . . . 8
1.2 Outils et technologies . . . 8
1.2.1 Plateformes de CI/CD . . . 8
1.2.2 Technologies de conteneurisation . . . 9
1.2.3 Outils de versionnement . . . 9
1.2.4 Outils de test . . . 9
1.2.5 Outils de communication et de collaboration . . . 9
Conclusion . . . 9

2 Analyse et choix techniques **10**
Introduction . . . 10
2.1 L'environnement logiciel . . . 10
2.1.1 Présentation de l'application . . . 10
2.1.2 Fonctionnalités de l'application . . . 10
2.2 Choix des composants de l'architecture . . . 11
2.2.1 Plateformes de CI/CD . . . 11
2.2.2 Technologies de conteneurisation . . . 12
2.2.3 Outils de versionnement . . . 12
2.2.4 Outils de test . . . 13
2.2.5 Outils de communication et de collaboration . . . 13
2.3 Architecture de la chaîne CI/CD . . . 14
2.3.1 Détails des étapes du pipeline . . . 14
2.3.2 Intégration avec le LMS ulearn . . . 14
2.3.2.1 Déclenchement des pipelines . . . 14
2.3.2.2 Déploiement des nouvelles versions . . . 14
Conclusion . . . 14

3 Solutions et Discussions **16**
Introduction . . . 16
3.1 Mise en place des outils . . . 16
3.1.1 Présentation de l'infrastructure . . . 16
3.1.2 Jenkins . . . 17
3.1.2.1 Installation de Jenkins . . . 17
3.1.2.2 Configuration initiale de Jenkins . . . 18
3.1.2.3 Plugins de Jenkins . . . 19
3.1.3 Docker . . . 20
3.2 Ecriture des tests . . . 22
3.2.1 Exigence des tests . . . 22
3.2.1.1 Critères d'acceptation . . . 22
3.2.1.2 Procédures de Test . . . 22
3.2.1.3 Suivi des Résultats . . . 22
3.2.2 Scénarios de test . . . 22
3.2.2.1 Créer un instructeur en tant qu'administrateur . . . 22
3.2.2.2 Se connecter en tant qu'instructeur . . . 23
3.2.2.3 S'inscrire en tant qu'étudiant . . . 23
3.3 Construction du pipeline . . . 24
3.3.1 Intégration continue . . . 24
3.3.2 Déploiement continu . . . 25
3.4 Résultats des tests . . . 29
3.4.1 Tests manuels . . . 29
3.4.2 Tests automatisés . . . 30

3.5 Discussion 30
Conclusion 30

Conclusion **31**

Bibliographie **32**

Webographie **33**

Table des matières **35**