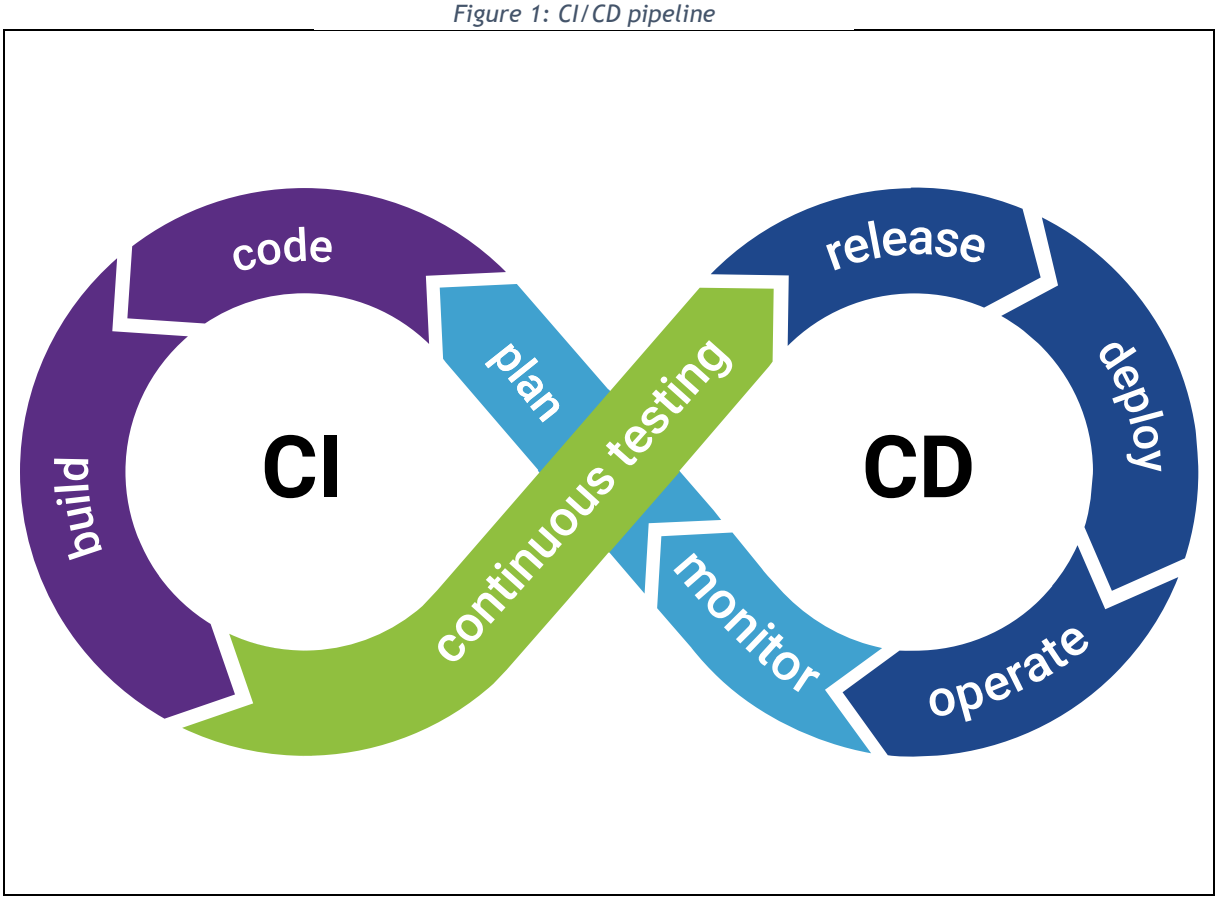
Team Academy - Article réflexif

Comment procéder pour la première fois à une pipeline CI/CD sur un projet concret.

Figure 1: CI/CD pipeline



Etudiant : Dasek Joiakim

Professeur : David Wannier

Table des matières

Introduction

Mettre en place une application web nécessite non seulement des compétences techniques, mais également une planification rigoureuse et une attention minutieuse aux détails. Dans le contexte actuel de complexité accrue et des exigences de livraison rapide, adopter une approche DevOps s'est avéré essentiel pour garantir la réussite d'un projet.

Cet article se propose de refléter sur un projet concret auquel j'ai participé, en mettant particulièrement l'accent sur les défis rencontrés et les solutions adoptées, notamment en matière de mise en place d'une pipeline CI/CD pour le déploiement en production.

Nous plongeons dans le projet Koloka, une plateforme de mise en relation pour futurs colocataires basée sur le matching de personnalités. Durant un an et demi de développement, nous avons affronté différentes problématiques et appris des leçons précieuses qui ont enrichi notre compréhension de la gestion de projets informatiques.

Cet article vise à partager ces expériences et à fournir une vision structurée sur les meilleures pratiques pour la mise en place d'une intégration et livraison continues dans le cadre d'une application web.

Expérience Concrète

Je vais tout d'abord introduire le projet Koloka, c'est une plateforme que nous développons qui permet de mettre en relation des futurs colocataires basé sur un système de matching de personnalités. Le stack technique est la suivante :

Frontend :

- Next.js, un meta-framework basé sur React prêt pour la production.

Backend :

- Strapi, un headless CMS qui permet de gérer le contenu pour le client du projet mais aussi pour les développeurs afin de générer une API.

Service tiers :

- Pusher, service de streaming de données (WebSockets) pour la fonctionnalité de messagerie entre futurs colocataires.

Après avoir lu plusieurs ressources, fait des lectures individuelles sur la philosophie « DevOps », ainsi que « Kubernetes » et des petits « labs ». J'ai appris et exercer pour avoir une vue d'ensemble sur les différentes manières d'implémenter une pipeline « CI/CD ».

Nous sommes à une étape cruciale du projet où il va falloir déployer en production. Je vais donc dans ce document proposer une solution que je vais mettre en pratique prochainement pour cette étape du projet.

Défis Initiaux

Au commencement du projet, nous étions portés par l'enthousiasme et la passion. Cependant, rapidement, nous nous sommes heurtés à des défis de planification et

d'optimisation du code et des délais. En visant la livraison fréquente et rapide, nous n'avons pas toujours accordé la priorité appropriée à certaines étapes critiques :

- **La factorisation du code** : Nous avons réalisé tardivement l'importance de créer du code réutilisable. Cela nous a amené à revoir certains modules, entraînant des pertes de temps significatives.

- **La sécurité** : Initialement sous-estimée, la sécurisation des données utilisateurs est devenue une urgence. Nous avons dû réévaluer nos priorités pour renforcer les mesures de sécurité.

- **L'UI/UX** : Bien que la création des maquettes ait commencé tard dans le projet, elles se sont révélées essentielles pour améliorer notre compréhension des besoins des utilisateurs et pour obtenir des retours constructifs.

Observation Réfléchie

Durant le projet et arrivant à la phase finale, je me préoccupais de plus en plus sur comment pouvoir mettre en place une stratégie qui me permettrait d'automatiser tout le processus « CI/CD » et que les clients finaux ne remarquent aucune différence sur le confort de consommation du service mais aussi pour que le côté « développeur & opérationnel » ait la possibilité de rollback une version en cas de problème ou que l'on puisse de manière simple pousser des nouvelles features sur la production.

Je prenais conscience de la nécessité d'avoir une bonne implémentation « CI/CD » avant même d'arriver à cette étape du projet et je m'inquiétais de la manière d'implémenter parce que sur l'ensemble des ressources à ce sujet, les implémentation gardaient dans l'ensemble le même principe mais elles n'étaient pas appliquées de la même manière. Cela m'a mené à piocher différents principes important « CI/CD » et les lister dans l'étapes suivantes qui sera la conceptualisation abstraite.

Conceptualisation Abstraite

Dans cette étape je vais vulgariser les éléments théoriques sur lesquels je vais me baser pour l'expérimentation active.

Je me suis immergé dans le monde de l'informatique et du développement logiciel, et l'un des concepts qui m'a le plus marqué est la philosophie DevOps. DevOps, contraction de « Development » et « Operations », est une méthodologie qui vise à amalgamer les équipes de développement logiciel et celles des opérations informatiques.

Cette approche vise à améliorer la collaboration entre ces deux fonctions traditionnellement cloisonnées. L'idée est simple : en favorisant la synergie entre le développement et les opérations, on accélère la livraison de logiciels tout en améliorant leur qualité.

À la base, DevOps naît de la reconnaissance que la culture, les processus et les outils doivent collaborer efficacement afin de répondre aux exigences de développement, de test et de mise en production de logiciels modernes.

Description des différents principes

La philosophie DevOps repose sur plusieurs principes fondamentaux :

Collaboration et Communication

La clé du succès de DevOps réside dans l'amélioration de la communication entre les équipes. Les silos organisationnels sont brisés pour encourager le dialogue ouvert et continu entre les développeurs, les administrateurs systèmes, et d'autres parties prenantes. Cela réduit les malentendus et les frictions qui résultent des priorités différentes et des objectifs opposés.

Automatisation

L'une des pierres angulaires de DevOps est l'automatisation des processus. L'automatisation s'applique au déploiement du code, aux tests, à la migration de bases de données et même à la surveillance des systèmes en production. L'objectif est de réduire l'intervention humaine, ainsi que les erreurs associées, tout en accélérant le processus de livraison.

Intégration Continue (CI)

Pratique dans laquelle le code développé par les différents membres de l'équipe est intégré fréquemment (au moins quotidiennement). Chaque intégration est vérifiée par une construction automatisée, permettant ainsi de détecter rapidement les erreurs.

Livraison Continue (CD)

Étendre l'intégration continue pour s'assurer que le code est toujours à un stade prêt pour la mise en production. Cela inclut des tests automatisés plus approfondis, des livraisons fréquentes et de petites mises à jour de code.

Surveillance et analyse continue

Il est crucial de surveiller et d'analyser constamment les systèmes en production pour identifier les problématiques potentielles et prendre des décisions éclairées. Les outils de

monitoring et de logging jouent un rôle clé dans cette démarche, permettant de collecter des métriques de performance et d'identifier les défaillances.

Infrastructure as a Code (IaaC)

La gestion d'infrastructure avec du code et des techniques de développement logiciel permet de traiter l'infrastructure de manière modulaire et évolutive. Cela réduit les risques d'erreur humaine et accroît la reproductibilité des environnements.

Points de vigilance sur la mise en pratique de la philosophie DevOps

Voici quelques points cruciaux que j'ai pu noter :

Culture et mentalité

La mise en place d'une culture DevOps implique un changement de mentalité qui peut susciter des résistances. Encourager une culture de confiance, d'apprentissage continu et de responsabilité partagée demande du temps et de l'effort.

Investissement initial

Intégrer DevOps nécessite souvent un investissement initial significatif en temps, en formation et en outils. Le retour sur investissement peut ne pas être immédiat, ce qui pourrait décourager les dirigeants sans une compréhension claire des bénéfices à long terme.

Sécurité DevSecOps

La sécurité doit être intégrée dès le début du cycle de développement et ne doit pas être une réflexion après coup. DevSecOps est une extension de la DevOps qui incorpore la sécurité comme une partie intégrante du processus.

Choix des outils

Le marché regorge de nombreux outils DevOps, souvent avec des fonctionnalités qui se chevauchent. Il est essentiel de faire des choix réfléchis et de ne pas céder à l'effet de mode, sinon ces outils peuvent devenir contre-productifs.

Introduction au CI/CD

Maintenant que nous avons vu les bases de la philosophie DevOps, concentrons-nous sur un de ses éléments essentiels : CI/CD. L'Intégration Continue (CI) et la Livraison Continue (CD) forment ensemble une chaîne de processus automatisés permettant de livrer des logiciels de manière plus efficace et fiable.

Une pipeline CI/CD est une série d'étapes que le code traverse pour être construit, testé et déployé. Chaque étape joue un rôle crucial pour garantir que seuls les changements de haute qualité atteignent la production. Voici une description détaillée des étapes typiques d'une pipeline CI/CD et leurs justifications :

Déclencheur (Trigger)

Une pipeline CI/CD démarre généralement par un déclencheur. Celui-ci peut être une action comme un commit dans le système de versionnement, l'ouverture d'une pull request, ou un événement planifié. Le but est de s’assurer que la pipeline s’exécute dès que du code nouveau ou modifié est disponible.

Compilation (Build)

Première étape après le déclencheur, la compilation transforme le code source en un binaire exécutable. Pour des langages interprétés, il peut s'agir de l'étape d'installation des dépendances. L’objectif est de s’assurer que le code se compile ou s'installe correctement avant toute autre étape.

Tests unitaires

Ces tests vérifient individuellement chaque composant ou module du code pour s'assurer qu'ils fonctionnent correctement. Exécuter les tests unitaires à ce stade identifie rapidement les régressions ou les bugs introduits par les changements récents.

Tests de qualité de code

Inclure des analyses statiques et des vérifications de style de code permet de maintenir une base de code propre et conforme aux normes de l'équipe. Les outils de linters ou d'analyse statique du code sont couramment utilisés ici.

Tests d'intégration

Ensuite, le code passe par des tests d'intégration qui vérifient si les différents modules fonctionnent bien ensemble. Cela est essentiel pour identifier les problèmes d'interaction entre les composants.

Déploiement en environnement de test

Une fois les tests passés, le code est déployé dans un environnement de test ou de staging qui imite la production. Ceci permet de valider les comportements dans un environnement quasi-réel.

Tests de régression et tests de performance

À ce stade, des tests automatisés plus exhaustifs comme les tests de régression, de performance ou de sécurité sont exécutés pour garantir que le code fonctionne sous différentes conditions de charge et qu'il n'a pas introduit de vulnérabilité.

Validation humaine

Parfois, une vérification manuelle est nécessaire. C'est l'opportunité pour les testeurs ou les parties prenantes de valider fonctionnellement et visuellement les changements.

Déploiement en production

Si tout est validé, le code est alors déployé dans l'environnement de production. Ce déploiement peut être automatisé pour minimiser les interruptions et éviter les erreurs humaines.

Surveillance en production

Après le déploiement en production, la surveillance continue est mise en place pour détecter les anomalies ou les problèmes de performance. Des outils de monitoring et de

journalisation sont essentiels pour fournir des retours immédiats et prendre des mesures correctives rapidement.

Points de vigilance sur la mise en pratique d'une pipeline CI/CD

Lors de la mise en œuvre d'une pipeline CI/CD, il est important de garder les points suivants à l'esprit :

Complexité de la pipeline

Il est tentant d'ajouter de nombreuses étapes dans la pipeline, mais une trop grande complexité peut ralentir les déploiements et rendre la maintenance plus difficile. Il est crucial de trouver un équilibre entre exhaustivité des tests et efficacité des processus.

Feedback rapide

Une des valeurs clés d'une pipeline CI/CD est un feedback rapide. Assurez-vous que les tests les plus critiques sont exécutés en premier pour offrir un retour d'information rapide aux développeurs.

Isolation des environnements

Les environnements de test et de production doivent être correctement isolés pour éviter que les tests aient des impacts involontaires sur la production. Utiliser des conteneurs ou des machines virtuelles est une bonne pratique.

Gestion des secrets

Il est important de gérer les secrets (comme les clés API, les certificats, etc.) de manière sécurisée. Les pipelines CI/CD doivent éviter d'exposer ces informations sensibles.

Scalabilité et résilience

La pipeline doit pouvoir gérer une croissance du volume de code ou du nombre d'équipes tout en restant résiliente aux pannes ou aux erreurs. Utiliser des solutions cloud-native peut aider à atteindre ces objectifs.

Pourquoi les Tests d'intégration sont dans la phase de déploiement continu ?

J’avais besoin d’éclaircissement sur ce point parce que je ne comprenais pas pourquoi les tests d’intégrations ne se retrouvent pas dans la phase d’intégration continu dû au fait que le terme « intégration » se retrouve dans les deux éléments.

Les **tests d'intégration** nécessitent généralement un environnement plus complet, où plusieurs composants de l'application peuvent interagir. Ils vont au-delà des tests unitaires en vérifiant comment différentes parties du système fonctionnent ensemble, souvent nécessitant des bases de données, des services externes, et d'autres dépendances. Il est évident que les tests d’intégrations se retrouvent dans un environnement où l’application est en cours d’exécution.

Dans la phase CI :

- Les tests sont souvent isolés et rapides pour fournir un feedback immédiat aux développeurs.
- Les tests d'intégration, plus complexes et plus lents, ne sont pas idéaux à ce stade, car ils ralentiraient le cycle de feedback rapide.

Dans la phase CD :

- Les tests d'intégration sont exécutés après que le code a été validé par des tests unitaires et autres vérifications rapides.
- L'environnement de staging peut être configuré pour refléter fidèlement l'environnement de production, permettant des tests d'intégration plus réalistes.

Synthèse des Phases CI/CD

CI (Intégration Continue)

1. Linting
2. Analyse Statique
3. Tests Unitaires
4. Build Léger (optionnel)

CD (Déploiement Continu)

1. Build Final de l'Image Docker
2. Déploiement en Staging
3. Tests d'Intégration
4. Tests de Charge
5. Déploiement en Production

Expérimentation Active

Je vais décrire dans cette étape la pipeline qui vise à automatiser les processus de développement, d'intégration, de test et de déploiement, tout en minimisant les risques d'erreurs humaines et en assurant une haute disponibilité de l'application.

Processus de la pipeline CI/CD

La pipeline CI/CD est composée de plusieurs branches et étapes qui assurent un flux de travail structuré et sécurisé. Voici un aperçu des branches utilisées et des étapes clés du processus :

Branching strategies :

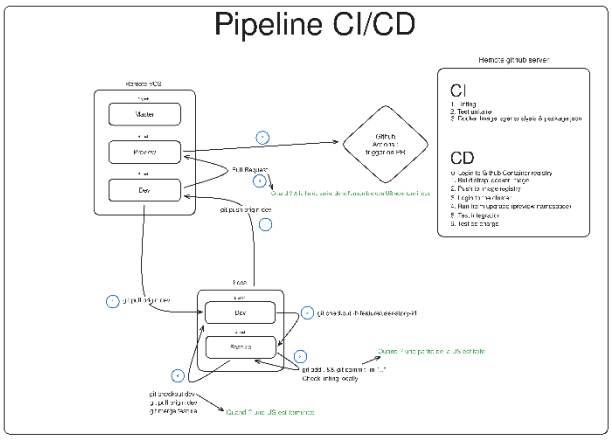
- feature/<feature-name> : Pour le développement de nouvelles fonctionnalités en local, lorsqu'un développeur commence la réalisation de sa User Story.
- dev : Pour l'intégration des nouvelles fonctionnalités.
- preview : Pour les tests de préproduction, incluant les tests d'intégration et de charge.
- master : Contient le code prêt pour la production.

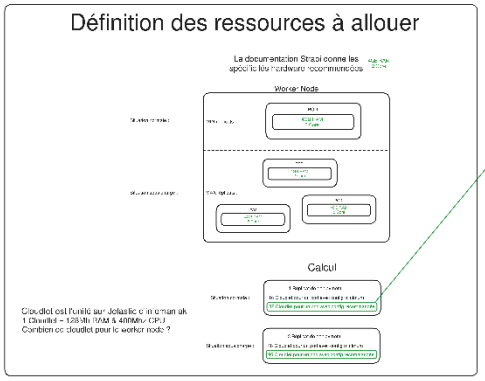
Étapes clés lorsqu'un développeur a une user story affectée :

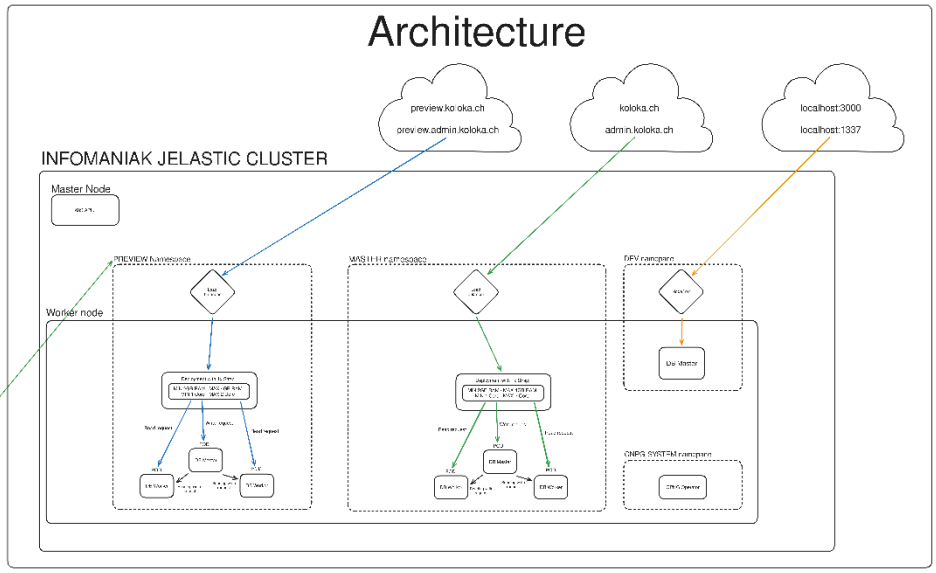
1. Git pull de la branche dev.
2. Création de la branche feature en local.

3. Développement et commits, idéalement plusieurs commits lors du développement de la user story.
4. Linting en local même si automatisé plus tard.
5. Merge de la branche feature dans Dev local et push des changements sur l'origine.
6. Lorsque l'ensemble des user stories (de l'équipe dev) est terminée pull request de dev à la branche preview.
7. Déclenchement de la pipeline CI/CD
8. Intégration continue sur la branche preview
9. Build et déploiement dans le namespace preview et continuité des étapes de CD sur preview
10. Pour cet article réflexif je m'arrête à cette étape, pas de pipeline de production pour l'instant.

Veuillez suivre ce lien pour que vous visualisiez la pipeline CI/CD, ou voici un snapshot :







Description du schéma DevOps et pipeline CI/CD

Pipeline CI/CD

Description Générale : La section supérieure gauche du schéma représente le pipeline CI/CD (Intégration Continue et Livraison Continue). Ce pipeline est essentiel pour automatiser et orchestrer les processus de développement, de test et de déploiement.

Éléments et Explications :

1. **Dépôt de Code Source :**
    - **Remote VCS (Version Control System)** : Le code source est maintenu dans un dépôt Git, hébergé sur un serveur GitHub distant. Les branches principales sont **master**, **preview**, et **dev**.
    - **Local Repository** : Les développeurs travaillent sur des copies locales du code, où ils créent des branches pour chaque fonctionnalité ou correctif (**feature/story-id**).
2. **Processus de Pull Request (PR) :**
    - Lorsqu'une fonctionnalité est terminée, une pull request est créée depuis la branche **dev** vers la branche **preview**. Ceci déclenche des workflows automatisés sur GitHub.
3. **Étapes de l'Intégration Continue (CI) :**
    - **Linting** : Vérification du code pour s'assurer qu'il respecte les normes de style définies.
    - **Tests Unitaires** : Exécution de tests automatisés pour vérifier le bon fonctionnement des composants individuels.

- **Analyse Docker et package.json** : Analyse de l'image Docker (Dockerfile) et des dépendances définies dans le fichier **package.json**.

4. **Étapes de la Livraison Continue (CD) :**

    - **Login to Docker Container Registry** : Connexion au registre de conteneurs Docker.
    - **Build and Push Image** : Construction de l'image Docker et envoi au registre.
    - **Helm upgrade** : Mise à jour des services à l'aide d'Helm, un gestionnaire de packages pour Kubernetes.
    - **Tests fonctionnels et de charge** : Exécution de tests pour vérifier le bon fonctionnement et la performance de l'application.

Conclusion

En adoptant une philosophie DevOps, nous allons pouvoir voir l'impact de la collaboration systématique entre les équipes de développement et celles des opérations en l'occurrence nous faisant les deux parties dans ce projet. Cette synergie devrait pouvoir conduire à des itérations plus rapides et une meilleure qualité de code, tout en maintenant un équilibre entre des processus automatisés et des interventions humaines pour la validation finale. La diversité des outils et la mise en place stratégique des processus d'automatisation, comme l'Intégration Continue (CI) et la Livraison Continue (CD), devraient renforcer non seulement l'efficacité mais aussi la sécurité du déploiement.

Les stratégies de branchement et de validation mises en place ont joué un rôle crucial, assurant une gestion structurée des versions et une minimisation des risques lors des déploiements. La stricte séparation entre les environnements de développement, de préproduction et de production, accompagnée par des tests exhaustifs à chaque étape de la pipeline CI/CD, a garanti que seuls les changements de haute qualité atteignent les utilisateurs finaux.

Enfin, l'application de ces pratiques n'a pas été sans défis, notamment en termes de complexité des pipelines et de gestion des ressources. Cependant, en reconnaissant et en adressant ces défis, j'ai établi un cadre robuste et scalable pour le développement, le test et le déploiement continus, assurant une haute disponibilité et des déploiements sans interruption notable pour les utilisateurs finaux.

Cette expérience a enrichi ma compréhension et me pousse à continuer d'adopter et d'améliorer nos pratiques DevOps pour répondre aux attentes de plus en plus élevées en matière de développement logiciel dans un environnement de plus en plus compétitif et rapide.

Bibliographie

Agarwal, G. (2021). *Modern DevOps Practices*. Packt Publishing.

Alliel, M. (s.d.). *ci-cd-pipelines-for-kubernetes-best-practices-and-tools*. Récupéré sur komodor.com: https://komodor.com/blog/ci-cd-pipelines-for-kubernetes-best-practices-and-tools/

Gitlab. (s.d.). *ci-cd*. Récupéré sur about.gitlab.com: https://about.gitlab.com/topics/ci-cd/

Red Hat. (s.d.). *what-is-ci-cd*. Récupéré sur redhat: https://www.redhat.com/fr/topics/devops/what-is-ci-cd