Labo 2 : DevOps et CI/CD pour MobilitySoft

PAR

MASSY HADDAD, HADM81090107
TAHA BENIFFOU, BENT80060007
NILAXSAN THARMALINGAM, THAN63370001

RAPPORT DE LABORATOIRE PRÉSENTÉ À MONSIEUR MOHAMMED SAYAGH DANS LE CADRE DU COURS INTRODUCTION À L'APPROCHE DEVOPS (LOG680-01)

MONTRÉAL, LE 06 NOVEMBRE 2025

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

1. Introduction

Le second laboratoire avait pour objectif d'établir un processus d'intégration continue (CI) et de conteneuriser les applications MobilitySoft et Metrics en respectant les pratiques DevOps. MobilitySoft sert à prédire le trafic routier, tandis que Metrics facilite le suivi des indicateurs de performance, en particulier pour le CI. Le concept central consistait à simplifier et garantir la fiabilité du déploiement et de la gestion des applications, tout en garantissant leur stabilité.

Pour cela, nous avons configuré des pipelines CI qui automatisent les étapes de build, de test et de déploiement vers DockerHub. En même temps, la conteneurisation avec Docker nous a permis d'encapsuler chaque application avec ses dépendances. Des métriques CI ont également été ajoutées pour surveiller la performance du pipeline et améliorer les processus.

Ce laboratoire nous a permis d'approfondir notre connaissance des méthodes DevOps et de mettre en œuvre des techniques concrètes pour administrer le cycle de vie des applications dans un contexte collaboratif et optimisé.

2. Répartition du travail

Dans un premier temps, Nilaxsan Tharmalingam a été chargé de conteneuriser les applications MobilitySoft et Metrics en créant les images Docker et en les publiant sur DockerHub. Il a également configuré les pipelines d'intégration continue pour ces applications avec GitHub Actions, incluant les étapes de build, de test et de déploiement sur DockerHub.

Massy Haddad s'est ensuite occupé de la configuration des variables d'environnement et de la mise en place des hooks pré-commit (pre-commit Git Hooks) pour préserver la qualité du code. Il a également travaillé sur les modifications du code pour garantir la persistance des données en configurant leur sauvegarde dans la base de données. Massy Haddad et Taha Beniffou ont collaboré ensemble pour mettre en place les métriques CI permettant d'évaluer les performances du pipeline.

Taha Beniffou a pris en charge l'implémentation des tests automatiques, incluant la séparation automatique des tests unitaires et d'intégration avec les markers pytest. Il a également contribué à l'amélioration de la qualité du code et à la configuration des différents aspects du pipeline CI/CD.

Nilaxsan Tharmalingam a rédigé une bonne partie de la documentation et du rapport avec l'aide de Massy Haddad et Taha Beniffou. Les trois membres ont collaboré à la rédaction finale et ont réalisé les tests pour s'assurer de la qualité et de la fiabilité de l'application.

Au cours de ce projet, nous avons collaboré pour surmonter les obstacles techniques et assurer que chaque phase se déroulait en accord avec les critères du laboratoire 2.

3. Description et justifications des étapes d'implémentation du CI

Notre pipeline d'intégration continue a été structuré en trois étapes séquentielles distinctes, chacune ayant un rôle précis dans le processus de déploiement. Cette organisation permet de garantir la qualité du code à chaque niveau avant de passer à l'étape suivante, réduisant ainsi les risques d'erreurs en production.

La première étape consiste en l'exécution des tests d'intégration automatiques. Ce choix a été fait pour valider le bon fonctionnement de l'application avant toute tentative de build ou de déploiement. Les tests d'intégration vérifient que les différents composants de l'application (API FastAPI, base de données PostgreSQL, modèle de prédiction) communiquent correctement entre eux. Cette étape s'exécute sur toutes les branches pour permettre aux développeurs de détecter rapidement les problèmes avant de merger leurs modifications. Nous avons utilisé pytest avec des markers pour séparer automatiquement les tests unitaires des tests d'intégration, ce qui permet une exécution ciblée et plus rapide du pipeline. Si cette étape échoue, le pipeline s'arrête immédiatement, empêchant ainsi la construction d'images Docker défectueuses.

La deuxième étape du pipeline est la construction de l'image Docker pour l'application MobilitySoft. Cette étape ne s'exécute que sur la branche principale (main/master) et uniquement si les tests d'intégration ont réussi. Le choix de Docker comme technologie de conteneurisation a été motivé par plusieurs facteurs : la portabilité de l'application entre différents environnements, l'isolation des dépendances, et la facilité de déploiement. Le Dockerfile a été optimisé en utilisant une image de base Python légère (python:3.11-slim) et en organisant les instructions dans un ordre qui maximise l'utilisation du cache Docker. Les dépendances sont installées avant de copier le code source, ce qui évite de reconstruire entièrement l'image à chaque modification de code. Cette approche réduit significativement le temps de build et la consommation de ressources dans le pipeline CI.

La troisième et dernière étape consiste à publier l'image Docker sur DockerHub. Cette étape ne s'exécute qu'après le succès des deux étapes précédentes et uniquement sur la branche principale. Le choix de DockerHub comme registry a été dicté par sa gratuité pour les repositories publics, sa large adoption dans l'industrie, et son intégration native avec GitHub Actions. Chaque image est taguée avec plusieurs identifiants : le tag `latest` pour toujours pointer vers la version la plus récente, le numéro de build GitHub Actions pour tracer précisément quelle version a été déployée, et le SHA du commit Git pour permettre un rollback rapide en cas de problème. Cette stratégie de tagging multiple facilite à la fois le déploiement automatique (avec `latest`) et le débogage ou le rollback (avec les tags spécifiques). Les credentials DockerHub sont stockés de manière sécurisée dans les secrets GitHub Actions, évitant ainsi toute exposition dans le code source.

L'ensemble de ces trois étapes forme un pipeline robuste qui automatise complètement le processus de validation, construction et publication de l'application. Cette approche DevOps garantit que seul du code testé et validé atteint l'environnement de production, tout en permettant une itération rapide grâce à l'automatisation complète du processus.

---

4. Choix des métriques CI

Pour évaluer et optimiser notre pipeline d'intégration continue, nous avons implémenté quatre métriques clés qui nous permettent de surveiller différents aspects de la performance et de la fiabilité du système. Ces métriques ont été choisies pour leur capacité à fournir des insights actionnables sur le processus DevOps et à identifier rapidement les problèmes potentiels.

La première métrique mesure le temps d'exécution individuel de chaque build, calculé comme la différence entre l'heure de complétion et l'heure de démarrage du workflow (completed_at - started_at). Cette métrique est stockée dans le champ duration_seconds de la table WorkflowRun. Son avantage principal réside dans sa granularité : elle permet d'identifier précisément quels builds sont anormalement lents et nécessitent une optimisation. Par exemple, si un build particulier prend 15 minutes alors que la moyenne est de 5 minutes, nous pouvons investiguer les changements de code introduits dans ce build spécifique. Cette métrique a été choisie car elle permet une action immédiate et ciblée pour améliorer la performance du pipeline, en identifiant les goulots d'étranglement au niveau de builds individuels plutôt que de se fier uniquement à des moyennes qui peuvent masquer des problèmes ponctuels.

La deuxième métrique calcule le temps moyen d'exécution sur différentes périodes (7, 14 ou 30 jours), en appliquant une moyenne arithmétique sur toutes les durées de builds pendant la période sélectionnée. Cette valeur est stockée dans le champ avg_duration_seconds de la table CIMetrics. L'avantage de cette métrique est qu'elle révèle les tendances à long terme et permet de détecter une dégradation progressive de la performance qui serait difficile à percevoir en observant uniquement les builds individuels. Par exemple, si la moyenne sur 30 jours augmente graduellement de 5 à 8 minutes, cela indique une accumulation de complexité dans le projet (plus de tests, dépendances plus lourdes, etc.). Nous avons choisi cette métrique car elle complète la première en fournissant une vue macroscopique de l'évolution du pipeline, permettant ainsi une planification proactive de l'optimisation avant que les temps de build ne deviennent problématiques.

La troisième métrique mesure le taux de réussite et d'échec des builds en calculant le ratio entre les builds réussis et le total des builds, multiplié par 100 pour obtenir un pourcentage. Les valeurs sont stockées dans trois champs de la table CIMetrics : success_rate (le pourcentage), successful_runs (nombre de succès) et failed_runs (nombre d'échecs). Cette métrique offre l'avantage d'être un indicateur direct de la stabilité du code et de la qualité du pipeline. Un taux de réussite élevé (>90%) indique que le code mergé dans la branche principale est généralement de bonne qualité, tandis qu'un taux faible suggère soit des problèmes de qualité de code, soit des tests instables (flaky tests), soit une configuration CI inadéquate. Nous avons choisi cette métrique car elle est un indicateur clé de performance (KPI) DevOps reconnu qui permet de mesurer objectivement l'impact des pratiques de développement sur la fiabilité du système et de justifier les investissements dans l'amélioration de la qualité du code.

La quatrième métrique analyse la distribution temporelle des exécutions du pipeline en calculant plusieurs agrégations : le nombre moyen de runs par jour, l'heure la plus active et le jour de la semaine le plus actif. Ces informations sont stockées dans les champs runs_per_day_avg, most_active_hour et most_active_day de la table CIMetrics. Cette métrique présente l'avantage de révéler les patterns d'activité de l'équipe de développement et d'utilisation du pipeline, ce qui permet une meilleure planification de la maintenance et de l'allocation des ressources. Par exemple, si nous observons que 80% des builds s'exécutent entre 14h et 16h, nous pouvons éviter de planifier des maintenances du serveur CI pendant cette période. De plus, comprendre les jours les plus actifs aide à anticiper la charge sur l'infrastructure. Nous avons choisi cette métrique car elle apporte une dimension organisationnelle aux données techniques, permettant d'aligner les opérations DevOps avec les réalités du workflow de l'équipe et d'optimiser l'utilisation des ressources d'infrastructure en fonction des besoins réels.

Ces quatre métriques combinées offrent une vision complète et multidimensionnelle de la performance du pipeline CI/CD, permettant à la fois une optimisation tactique (builds individuels lents) et stratégique (tendances à long terme, patterns organisationnels).

5. Conclusion

Ce deuxième laboratoire nous a permis de mettre en pratique les concepts fondamentaux du DevOps en implémentant un pipeline d'intégration continue complet pour les applications MobilitySoft et Metrics. L'automatisation des processus de test, build et déploiement a transformé un workflow manuel susceptible aux erreurs en un système fiable et reproductible. Nous avons réussi à conteneuriser les applications avec Docker, à configurer un pipeline CI/CD en trois étapes séquentielles, et à implémenter un système de métriques permettant de surveiller la performance du pipeline.

Au-delà des aspects techniques, ce laboratoire nous a enseigné l'importance de la collaboration dans un contexte DevOps. La répartition claire des responsabilités entre les membres de l'équipe, combinée à l'utilisation d'outils modernes comme GitHub Actions et Docker, nous a permis de livrer un système robuste. Les défis rencontrés, notamment la gestion des secrets et l'optimisation des temps de build, ont renforcé notre compréhension des enjeux du déploiement continu.

Les quatre métriques CI/CD implémentées fournissent une visibilité précieuse sur la santé du pipeline et permettent d'identifier proactivement les opportunités d'amélioration. Cette approche data-driven illustre la philosophie DevOps qui place la mesure et l'amélioration continue au cœur du processus de développement.

En conclusion, les compétences acquises en matière d'automatisation, de conteneurisation et de monitoring sont directement applicables dans un contexte professionnel et constituent une base solide pour nos futurs projets. L'infrastructure CI/CD mise en place démontre qu'il est possible de créer un système de déploiement automatisé, fiable et mesurable pour des applications complexes.

Références

> **Liens de documentation**
>
> - Repository GitHub
> - Documentation Wiki complète
> - API Documentation (disponible quand l'API est démarrée)