Laboratoire 2

LOG680 – Introduction à l'approche DevOps

Automne 2025

Intégration continue et conteneurisation

Ce laboratoire a pour objectif de mettre en place un pipeline d'intégration continue et la conteneurisation d'une application.

Mise en scène

MTL-MobilitySoft est une jeune startup de 20 personnes, composée majoritairement de spécialistes en statistiques et en mathématiques, avec un peu d'expertise en DevOps. La compagnie est spécialisée dans le développement de solutions logicielles innovantes pour la prévision du trafic urbain à Montréal.

Pour élargir leur marché, dans le futur, MTL-MobilitySoft prévoit d'étendre son offre en développant une application mobile, en intégrant la prédiction d'itinéraires optimisés et en fournissant aux utilisateurs des explications sur les décisions prises par les modèles d'apprentissage automatique. Ces perspectives visent à rendre la solution plus accessible, transparente et utile pour la mobilité urbaine.

Cependant, la startup a identifié un défi majeur : l'intégration et la validation continue de ses modèles d'apprentissage automatique constituent une contrainte importante qui freine l'amélioration du flux de valeur dans son processus de développement. Ce défi provient principalement du profil des développeurs de MTL-MobilitySoft, qui sont majoritairement des spécialistes en apprentissage automatique, en algorithmique et en statistique, mais qui possèdent peu de connaissances des approches modernes de développement telles que DevOps et les méthodes agiles (p. ex. : Kanban, Scrum, Extreme Programming). De plus, l'équipe souhaite accélérer le cycle d'intégration des nouvelles fonctionnalités dans son application et disposer d'un processus de déploiement qui permette, dès qu'une fonctionnalité est ajoutée, que ses clients finaux puissent immédiatement en bénéficier.

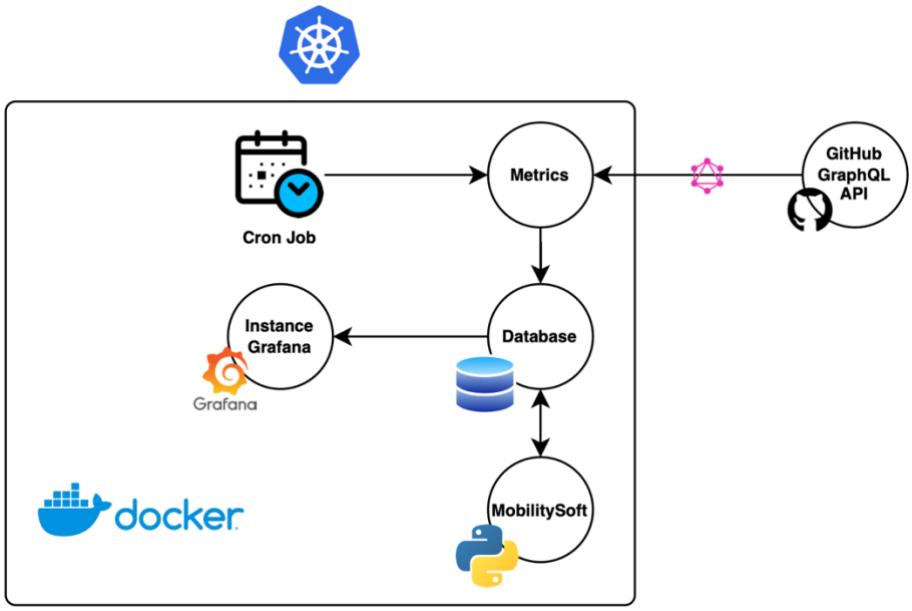

Figure 1. Architecture de la solution MobilitySoft

L’analyse du processus de développement existant d’MTL-MobilitySoft a permis d’identifier la phase d’intégration du logiciel comme étant la contrainte principale à l’amélioration du flux de valeur de développement du logiciel (*First way*).

Dans cette deuxième phase de votre projet, vous devez mettre en place le pipeline d'intégration continue (CI), incluant l’automatisation des tests et la conteneurisation (Docker) du logiciel, qui permettra de supporter le déploiement continu de l’application MTL-MobilitySoft. Vous devrez aussi faire des modifications à l’application MobilitySoft, **incluant l’ajout de tests unitaires ET d’intégration**, et exécuter le pipeline CI à la suite des modifications pour le tester/valider. L’image Docker produite par votre pipeline CI devra être disponible sur DockerHub.

En résumé, MTL-MobilitySoft vous demande d’implémenter un pipeline d’intégration continue qui sera détaillé dans les sections suivantes.

Figure 2. Pipeline CI

Prérequis

- Le laboratoire demande d’avoir un compte GitHub par étudiant
- Le laboratoire demande d’avoir un compte DockerHub par équipe

Contenu

Création d'un deuxième répertoire GitHub

Le code de base en Python est fourni pour ce TP dans un fichier zip accessible sur Moodle ainsi que dans un répertoire GitHub.
Vous devrez **créer un deuxième repository sur Github pour votre équipe avec ce code** et le nommer selon l'appellation suivante : **mobilitysoft-eq{x}.** Les détails d'exécution du projet se trouvent dans le README.

Vous devrez également utiliser les modèles d'issues et de pull requests sur ce repository. Pour cela vous pouvez directement faire un copier-coller du dossier *.github* de votre repository Metrics.

Modification du code source – Sauvegarde des données dans la base de données

Vous devrez modifier l'application MobilitySoft afin de pouvoir sauvegarder les informations suivantes dans la base de données du laboratoire (une table différente de celle du laboratoire 1) :

- Les points de départ et d'arrivée, la vitesse prévue et l'heure à laquelle ils ont été reçus. Il faut sauvegarder le plus de données possible afin de les visualiser dans une interface graphique à l'avenir.
- Créer une API permettant de lire les données déjà stockées dans la base de données.

Vous devrez initialiser la connexion à la base de données lors du démarrage de l'application et non à chaque envoi de données (afin d'éviter de créer un grand nombre de connexions).

Conteneurisation des applications

Pour faciliter le flux DevOps de l'application, vous devez utiliser Docker pour conteneuriser vos applications (Metrics et MobilitySoft). Vous allez devoir créer deux images, une pour chaque projet. Une fois les images créées, vous devrez plus tard mettre en ligne les images de vos applications sur votre compte DockerHub afin qu'elles soient accessibles pour un déploiement sur un cloud public.

Pour réussir cette tâche de conteneurisation, vous devrez apprendre les bases de Docker sur le web à travers les différentes documentations que vous trouverez en ligne.

Un principe fondamental du DevOps est l'amélioration continue, il vous est donc demandé d'améliorer les images Docker que vous avez créées pour faire en sorte qu'elles **consomment le moins d'espace** possible. Pour vous donner un ordre d'idée, une image Docker de MobilitySoft bien optimisée a une taille d'une trentaine de Mo sur DockerHub (une image sur DockerHub est automatiquement compressée sur la plateforme).

Intégration continue

Pre-commit git hook

Il vous est demandé ici de définir un pre-commit git hook pour vous assurer de la qualité du code à travers votre équipe.
Une librairie existe afin d'automatiser et de faciliter la mise en place d'un pre-commit, nous vous invitons à l'utiliser (documentation).
Votre pre-commit git hook devra inclure :

1. Linting
2. Formatage
3. Exécution de vos **tests unitaires seulement** (vous devrez donc créer également des tests unitaires pour MobilitySoft, à l'image de ce qui est demandé pour Metrics)

Voici des suggestions de linters et de formateurs pour Python que vous pouvez utiliser :

- Pylint (linter)
- Black (formateur)

Construction et déploiement des images Docker

Pour vous assurer que chaque nouvelle version des applications soit fonctionnelle, vous devez mettre en place un pipeline d'intégration continue sur **chacun de vos deux repositories** à l'aide de la technologie de votre choix (GitHub Actions est recommandé).

Votre CI doit accomplir trois tâches :

1. Lancer **les tests d'intégration (pas les tests unitaires)** de votre application. Si les tests échouent, le build s'arrête automatiquement.
2. Lancer la construction de l'image Docker **seulement lorsque les changements sont sur la branche main**. Vous devez tagger votre image avec « latest » et un autre identifiant de votre choix (la version de l'application, le build id, etc.) pour pouvoir réutiliser cette image même après la création d'une nouvelle image.
3. Lancer le déploiement sur DockerHub seulement lorsqu'il y a des changements sur la branche main.

Pour le déploiement sur DockerHub, vous aurez besoin d'un compte Docker et de fournir à Github des informations d'authentification pour ce compte. Ces informations (comme un token par exemple) ainsi que d'autres variables nécessaires (pour vos tests par exemple) pourront être mises dans des *secrets Github* (documentation).

N.B : Ainsi, même dans une autre branche que la branche main, les tests d'intégration doivent être effectués.

Figure 3. DockerHub

Métriques d'intégration continue

Vous devez ajouter un minimum de **4 métriques** de votre choix reliées à l'intégration continue (ex : temps d'exécution pour un build donné, temps moyen pour l'ensemble des builds pour une période donnée, quantité de builds et tests automatisés réussis et échoués) à votre application « Metrics » du Laboratoire 1. Elles aussi devront être insérées dans la base de données déjà créée au Laboratoire 1. C'est également l'occasion pour les équipes qui ne l'ont pas encore fait de mettre en place la sauvegarde de leurs métriques du Laboratoire 1 dans la base de données.

Consignes additionnelles

Vous devez continuer d'utiliser les **bonnes pratiques que vous avez développées au Labo 1**, notamment l'utilisation des pull-requests, de la structure de branche et du Kanban pour suivre l'évolution de vos tâches. Vous allez être évalué sur cela comme dans le Labo 1. Vous devez utiliser le même Kanban (project) pour vos deux repositories (il suffit de lier le repo MobilitySoft au projet également). Vous pourrez donc créer les issues liées à MobilitySoft dans le bon repo et elles apparaitront également dans le Kanban.

Pensez également à recréer les mêmes milestones dans le repo MobilitySoft afin de pouvoir filtrer les taches facilement.

Rapport

Comme au précédent laboratoire, vous devez soumettre un document PDF concis, comprenant une introduction, une description de la manière dont vous vous êtes réparti le travail, la description et les justifications des étapes d'implémentation de votre CI, les choix de métriques CI, et enfin une conclusion.

Le rapport peut faire référence à votre wiki / documentation dans GitHub. Cependant, il ne doit pas simplement être une page simple avec le lien de votre wiki sur GitHub.