Laboratoire 2

LOG680

Été 2022

Table des matières

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

La compagnie Oxygène Software développe une solution logicielle, appelée Oxygène CS, qui permet le contrôle d'un système CVC/HVAC[1](heating, ventilation, air-conditioning). Tel qu'illustré dans la Figure 1, le contrôleur Oxygène CS se connecte à deux types de composants externes : un ensemble de senseurs qui fournissent diverses données, par exemple la température ambiante, et un système HVAC qui contient les équipements mécaniques permettant de chauffer, climatiser, ou ventiler une pièce. Le contrôleur reçoit la valeur des températures ambiantes (des senseurs) et envoie des commandes de contrôle au système HVAC pour maintenir la température de la pièce à une température désirée.

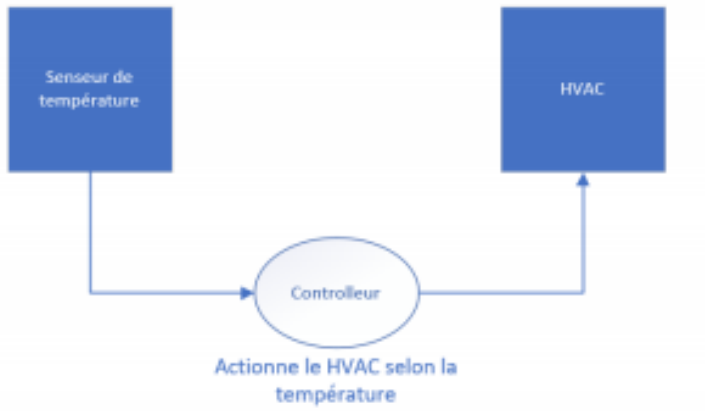

Figure 1. Architecture de la solution Oxygène CS

L'analyse du processus de développement existant d'Oxygène Software 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).

[1] Un HVAC est un système de chauffage, ventilation et d'air climatisé.

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 développement de l'application Oxygène CS. Vous devrez aussi faire des modifications à l'application (Oxygène CS), incluant l'ajout de test unitaire, 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, Oxygène Software vous demande d'implémenter le pipeline CI suivant (qui sera détaillé dans les Sections suivantes).

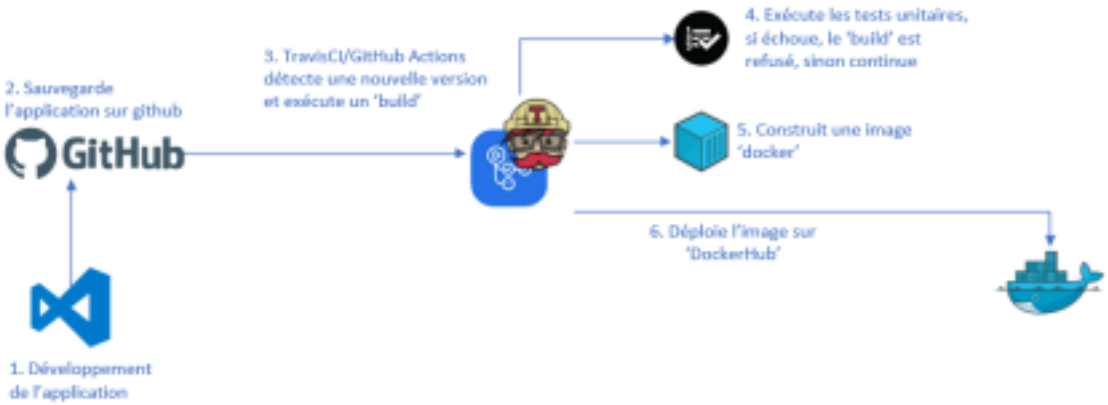

Figure 2. Pipeline CI

Pour tester l'application Oxygène CS, Oxygène Software a mis en ligne une application web qui permet le contrôle manuel des unités HVAC. De plus, elle a rendu disponible l'API qui doit être utilisée pour contrôler les unités et recevoir les données de température :

http://159.203.50.71

Pré-requis

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

Contenu

Création d'un répertoire GitHub

Le code de base en Python est fourni pour ce TP au lien suivant : *Python Template*.

Les détails d'exécution du projet sont détaillés dans le *README*. Si jamais vous voulez le faire dans un autre langage, vous pouvez le faire. Toutefois, vous devrez trouver une bibliothèque client de *SignalR* dans ce langage et faire votre propre implémentation.

Conteneurisation de l'application

Pour faciliter le flux DevOps de l'application, vous devez utiliser *Docker* pour conteneuriser votre application. Vous allez devoir créer une image. Une fois le conteneur créé, vous allez devoir mettre en ligne l'image de votre application sur votre compte *DockerHub*.

Pour réussir cette tâche de conteneurisation, vous devrez apprendre les bases de *Docker* sur le web (*Docker Hub Quickstart | Docker Documentation*).

Un principe fondamental du DevOps est l'amélioration continue. Améliorez l'image *Docker* que vous avez créée en sorte qu'elle consomme moins d'espace.

Intégration continue

Pre-commit git hook

Vous devez définir un *pre-commit git hook* pour vous assurer de la qualité du code à travers votre équipe. Voici la documentation sur les *git hooks*. Puisque le projet est en *python*, vous pouvez définir des étapes de *linting*, analyse statique de code (e.g. SonarQube) et de formatage. Vous pourriez aussi ajouter vos tests unitaires au *git hook*. Voici de la documentation sur les *linters* et les formateurs pour python :

- User Guide — Pylint 2.10.3-dev0 documentation (*linter)*
- The uncompromising code formatter — Black 21.9b0 documentation (formateur)

Construction et déploiement de l'image *Docker*

Pour vous assurer que chaque nouvelle version de l'application soit fonctionnelle, vous devez mettre en place un pipeline d'intégration continue à l'aide de la technologie de votre choix (*GitHub Actions*, *TravisCI*, *CircleCI*, etc). Voici la documentation pour ces technologies :

- *GitHub Actions Documentation*
- *Welcome to CircleCI Documentation*
- *Travis CI User Documentation*

Votre *CI* doit accomplir quatre tâches :

1. Lancer les tests unitaires de votre application. Si les tests échouent, le *build* s'arrête automatiquement.
2. Lancer les tests unitaires sur toutes les branches (*events* comme *push* ou *pull-request* sur une branche).
3. Lancer la construction de l'image *Docker* seulement lorsque les changements sont sur la branche *main* (*events* comme *push*). Vous devez ***tag*** 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.
4. Lancer le déploiement sur *DockerHub* seulement lorsqu'il y a des changements sur la branche *main* (*events* comme *push*).

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 du pipeline de build pour un build donné, temps moyen pour l'ensemble des builds pour une période donnée, quantité de builds et quantité de tests automatisés réussis et échoués) à votre application du TP1. Les nouvelles métriques doivent être récupérables pour le pipeline d'intégration continue du TP2 à partir de l'application développée au TP1.

Modification du code source

Pour bien tester le *pipeline* d'intégration continue créer à l'étape 3, il vous est demandé d'apporter la modification suivante à votre projet : paramétrer l'application à l'aide de variables d'environnement. Dans l'application de base, il y a 4 variables qui pourraient être paramétrées pour faciliter la configuration selon les besoins et l'environnement :

- La limite de froid
- La limite de chaud
- Le *token* de l'unité HVAC
- Le nombre de *tick* lors de l'activation du HVAC (un *tick* correspond à un appel d'activation de la climatisation ou du chauffage)

Si la variable d'environnement n'est pas définie, vous devez avoir une valeur par défaut. Si le *token* n'est pas défini en variable d'environnement, l'application doit retourner une erreur. Pour toute modification du code, vous devez ajouter un ensemble des tests unitaires en lien.

Consignes additionnelles

Vous devez continuer d'utiliser les bonnes pratiques que vous avez développées au TP1, 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 TP1.

L'effort additionnel que vous mettez pour améliorer l'image *Docker* de votre application se reflète dans la cohérence et la pertinence du projet.

Rapport

Pour le rapport, vous devez soumettre un court document qui explique le fonctionnement de votre intégration continue. Vous pouvez faire référence à votre documentation dans votre repo GitHub dans le rapport. Vous n’êtes pas obligé de vous répéter.