ESIREM - 4A - ILC/SQR

MODULE DE CI/CD

EXAMEN PRATIQUE : TP PROJET

Janvier 2023

Le projet noté du module de CI/CD évaluera compétences et bonnes pratiques de développement vues en cours. La notation tiendra compte des fonctionnalités déployées dans l'API, de la bonne mise en place des points d'exigences projets ainsi que de la collaboration entre les membres du groupe.

1 Exigences générales projet

Le projet est à rendre pour le *01 février 2023 à 23h59*
À partir de cette date, **aucune modification** du dépôt ou de code ne sera prise en compte.

1.1 GitHub

- Un dépôt GitHub dédié au projet, auquel vous m'aurez ajouté en tant que collaborateur.
- L'historique des changements sur le dépôt devra montrer la collaboration entre les membres du groupe (changement de sources différentes).
- Automatiser le build d'image dans des GitHub Actions.
- Le dépôt devra être documenté via README.

1.2 Docker

- Pousser au moins trois images différentes sur le Registre d'image. (GCR)
- L'image générée par le Dockerfile peut-être exécutée via docker run.

1.3 Documentation

- Un README principal contenant au minimum : **sujet du projet**, **membres du groupe**, les **technologies utilisées**, des **badges** des résultats des builds des dernières CI exécutées et la **procédure** pour exécuter l'API.
- Un README supplémentaire pour chaque dossier décrivant succinctement son contenu.
- Un fichier Swagger valide au sens de l'éditeur Swagger. (cf. *https://editor.swagger.io/*)

Toutes informations supplémentaires sur le fonctionnement des endpoints et de l'API sera la bienvenue.

2 Sujets

2.1 Sujet guidé : Un chemin tout tracé

Objectif : Créer une API Flask pour de la gestion CRUD d'un système de transaction.
Langage : *Python* (à préciser dans le README) vous pouvez choisir un autre langage.

Définissons une transaction comme étant un tuplet (P1, P2, t, s), où **s** est égal à la somme d'argent transférée de la personne **P1** à la personne **P2** à l'instant **t**.

2.1.1 Réaliser une première version de l'API REST

En utilisant Flask, réaliser une première version de l'API. Voici une liste des actions (aussi appelées routes ou endpoints) qui doivent être mises à la disposition via un appel HTTP sur API:

E1 - Enregistrer une transaction.

E2 - Afficher une liste de toutes les transactions dans l'ordre chronologique.

E3 - Afficher une liste des transactions dans l'ordre chronologique liées à une personne.

E4 - Afficher le solde du compte de la personne.

E5 - Importer des données depuis un fichier csv. (à documenter)

2.1.2 Documenter l'API via des READMEs et un fichier Swagger

- Documenter et justifier votre choix de sujet dans le fichier README.md.
- Ajouter le détail pour une procédure de chargement de données dans l'API à partir d'un fichier .cvs dans le dépôt et du endpoint E5.
- Détailler les endpoints dans un YAML valide suivant l'éditeur Swagger.
  cf. *https://editor.swagger.io/*

2.1.3 Préparer l'intégration continue (CI)

Créer trois github actions :

- Une déclenchée à chaque changement pour builder l'application.
- Une déclenchée manuellement pour builder et dockeriser et pousser l'image de l'API.
- Une déclenchée pour chaque tag semver pour builder et dockeriser et pousser l'image de l'API avec en tag la version semver spécifiée.

2.1.4 Anticiper le déploiement continue (CD)

Vous allez maintenant automatiquement publier les nouvelles versions dans un registre de conteneur Google (GCR).

Ajoutez au workflow déclenché pour chaque nouveau tag, le **job** et la **variable** d'environement disponible dans l'action à l'adresse suivante pour pousser l'image créé sur le registre :
https://github.com/JeromeMSD/module_ci-cd/blob/main/.github/workflows/Docker_push_GCR.yaml

Modifiez le paramètre **tags** de l'étape "Build and push Docker images" pour y mettre la valeur :

gcr.io/esirem/4A_[ILC ou SQR]/[NOM1_NOM2]/[nom_du_projet]/:{github.ref_name}

Modifiez les paramètres **file** et **context** de l'étape "Build and push Docker images" pour y mettre les bonnes valeurs suivant votre projet.

2.1.5 Top départ

Déployez une première release publique de l'API à ce stade via GitHub avec un tag équivalent à la bonne version sémantique.

2.1.6 Améliorer l'API

Pour chacune des fonctionnalités suivantes, faites une release de votre API avec le numéro de version adapté.

Ajoutez maintenant le hash d'une transaction dans son modèle: (P1, P2, t, s, h),où **s** est égal à la somme d'argent transférée de la personne **P1** à la personne **P2** à l'instant **t** et **h** correspond au hash de P1, de P2, et s. Votre choix de la fonction de hachage doit également être documenté dans le fichier README.md.

> Déployez une release publique de l'API à ce stade via GitHub avec un tag équivalent à la bonne version sémantique.

Ajouter l'action suivante disponible en API HTTP : Vérifier l'intégrité des données envoyées en recalculant les hashs à partir des données envoyées et en les comparant avec les hashs stockés précédemment.

> Déployez une release publique de l'API à ce stade via GitHub avec un tag équivalent à la bonne version sémantique.

Corriger le calcul de hash en prenant en compte le paramètre t : la date de transfert.

> Déployez une release publique de l'API à ce stade via GitHub avec un tag équivalent à la bonne version sémantique.

2.2 Sujet non-guidé Flask : Un brin de décision

Objectif : Tout en respectant les exigences projets en page 1. Créer une API Flask pour de la gestion CRUD sur le sujet de votre choix (ex: boulangerie, parking, film, ...)

Langage : Python est recommandé, vous pouvez choisir un autre langage.

2.2.1 API REST

Définir au moins un modèle de données, que vous détaillerez dans votre README. Déclarer une API REST permettant de jouer avec ce ou ces modèle de données, vous devrez implémenter toutes les actions CRUD (Create, Read, Update, Delete).

2.2.2 Documenter l'API via des READMEs et un fichier Swagger

- Documenter et justifier votre choix de sujet dans le fichier README.md.
- Ajouter le détail pour une procédure de chargement de données dans l'API à partir d'un fichier .cvs dans le dépôt et du endpoint E5.
- Détailler les endpoints dans un yaml valide suivant l'éditeur Swagger.
  cf. *https://editor.swagger.io/*

2.2.3 Intégration continue

Créer trois github actions :

- Une déclenchée à chaque changement pour builder l'application.
- Une déclenchée manuellement pour builder et dockeriser et pousser l'image de l'API.
- Une déclenchée pour chaque tag semver pour builder et dockeriser et pousser l'image de l'API avec en tag la version semver spécifié.

2.2.4 Déploiement continue (CD)

- Ajoutez au workflow déclenché pour chaque nouveau tag le **job** et la **variable** environement disponible dans l'action à l'adresse suivante pour pousser l'image créé sur le registre : *https://github.com/JeromeMSD/module_ci-cd/blob/main/.github/workflows/Docker_push_GCR.yaml*
- Modifiez le paramètre **tags** de "Build and push Docker images" pour y mettre la valeur :

gcr.io/esirem/4A_[ILC ou SQR]/[NOM1_NOM2]/[nom_du_projet]/:${{ github.ref_name }}

- Modifiez les paramètres **file** et **context** de "Build and push Docker images" pour y mettre les bonnes valeurs suivant votre projet.

2.2.5 Top départ

Déployez une première release publique de l'API à ce stade via GitHub avec un tag équivalent à la bonne version sémantique.

2.2.6 Améliorer l'API

Pour chacune des fonctionnalités suivantes, faites une release de votre API avec le numéro de version adapté.

1. Ajoutez et documentez un endpoint de chargement de données depuis un fichier.
2. Ajoutez et stockez maintenant le hash d'une instance du modèle. Votre choix de la fonction de hachage doit également être documenté dans le fichier README.md.
3. Implémenter et documentez des modifications **non-rétrocompatibles**, **rétrocompatibles** et des **correctifs** et faites les releases SemVer associées.