Travaux Pratiques Principes et outils pour le DevOps

Enoncé 1 – Intégration continue avec Github Actions

Les dépôts initiaux se trouvent à l'adresse https://github.com/CERI-M1-DevOps

Ce premier TP correspond au dépôt TPIntegrationContinue, que vous allez forker (dans classroom) en cliquant sur le lien proposé dans le README. Vous travaillez ensuite localement dans un clone sur votre poste de travail.

Exercice 1 – Débuter…

Essayez l'exemple présenté en cours pour voir comment utiliser GitHub Actions.
Pour cela :

- sur la branche main, créez deux répertoires : .github et .github/workflows
- dans ce répertoire, créez un fichier salut.yaml dont le contenu sera :

```
name: Bonjour

on: [push] # Ce workflow se déclenche sur un push dans le dépôt

jobs:
  Salutations:  # Nom du job : Salutations
    name: Salutations
    runs-on: ubuntu-latest  # OS où s'exécute le job
    steps:
      - name: Hello #une tâche pour saluer
        env:
          personneASaluer: 'Mon voisin Totoro'
        run: echo "Bonjour ${personneASaluer}"
      - name: L'heure de salutation # affiche l'heure de la précédente tâche
        run: echo "L'heure était $(date)."
```

- committez et poussez sur votre fork
- observez dans Actions (fonctionnalité proposée sur la plateforme Github) le workflow Bonjour qui s'exécute. Consultez les 4 étapes, et comparez avec le contenu du fichier YAML.
- désactivez ce workflow, ce n'était qu'une simple prise en main de l'environnement…

Exercice 2 – Build

Travaillez maintenant sur la branche **`dev`**, dans laquelle vous pouvez constater que du code vous est fourni. Prenez un peu de temps pour le lire, ainsi que les tests qui vont avec.

1. Créer un fichier `buildJava.yml` qui doit respecter les informations suivantes :

   Le nom du workflow sera "Java CI with Maven - Test and package", il doit se déclencher lorsqu'on émet une pull request sur la branche main. Il ne contient qu'un seul job, qui s'exécute sur la dernière version d'ubuntu.

   Il y aura 3 étapes, que vous nommerez de façon appropriée :

   - utilisez l'action `actions/checkout@v4` pour se placer dans le dépôt courant (vous pouvez consulter https://github.com/actions/checkout)
   - utilisez ensuite l'action `actions/setup-java@v4` (vous pouvez consulter https://github.com/actions/setup-java). Choisissez la version 21 et la distribution corretto
   - enfin faire construire votre package avec maven : pour cela un fichier pom.xml vous est fourni, qui définit les dépendances impliquées dans ce projet. La commande pour construire est `mvn -B package`, elle lancera les tests avec JUnit.

2. Sur la plateforme github, ouvrez maintenant une pull-request de la branche dev vers la branche main. Constatez que le workflow s'exécute et observez les étapes. Des tests ont été exécutés, retrouvez les résultats dans le rapport. Remarque : vous n'avez pas à fusionner cette pull-request, laissez-la toujours ouverte.

Exercice 3 – Qualité du code

Vous allez maintenant observer la qualité de votre code en utilisant la plateforme https://sonarcloud.io sur laquelle vous vous connectez via votre compte github.

1. Pour pouvoir lancer l'analyse de la qualité, il faut changer la commande maven pour :

```
mvn -B verify org.sonarsource.scanner.maven:sonar-maven-plugin:sonar
-Dsonar.projectKey=$(echo ${{ github.repository }} | sed 's-/-_-')
```

Cette commande maven nécessite une authentification sur la plateforme sonarqube. Celle-ci a été définie pour toute l'organisation CERI-M1-DevOps (à laquelle appartiennent vos dépôts) sous la forme d'un jeton nommé `SONAR_TOKEN`. La valeur de ce jeton est définie comme un "secret", que l'on obtient en évaluant `${{ secrets.SONAR_TOKEN }}`.
Pour que la commande maven fonctionne correctement, il faut donc qu'elle connaisse une variable d'environnement avec les nom et valeur spécifiés ci-dessus.

2. Constatez les défauts dans le code proposé et corrigez-les (tous !).

3. Faites en sorte que la couverture de code soit à 100%.

Exercice 4 – Mise en cache

Vous constatez que les téléchargements des dépendances maven se font systématiquement à chaque build. Il peut être intéressant de configurer votre pipeline pour que la recherche des dépendances se fasse en cache, avant d'éventuellement télécharger.
Pour cela, vous pouvez vous documenter sur

https://docs.github.com/fr/actions/writing-workflows/choosing-what-your-workflow-does/caching-dependencies-to-speed-up-workflows

et ensuite utiliser l'action `actions/cache@v4` et son implémentation pour maven.

Faites une simple modification dans un des fichiers source (pas exemple l'ajout d'un saut de ligne) et constatez que la recherche se fait maintenant en cache.

Vous pouvez ensuite ajouter une utilisation du cache pour les packages de SonerQube, inspirez-vous de ce qui est proposé ici :

https://docs.sonarsource.com/sonarqube/latest/devops-platform-integration/github-integration/adding-analysis-to-github-actions-workflow/

Exercice 5 – Génération (livraison) de documentation

Vous pouvez maintenant désactiver le pipeline correspondant au build (`buildJava.yml`)

A défaut de pouvoir livrer une application finalisée, vous allez livrer la documentation du module réalisé.

A - Dans un premier temps, documentez quelques fonctions du fichier `ListeSimple.java`. Pour cela, ajoutez au-dessus de n'importe quelle méthode (y compris un constructeur) :

```
/**
 * Description de la méthode
 * @param var1 description du premier paramètre
 * @param var2 description du second paramètre
 * @return Description de la valeur retournée
 */
```

Ce texte sera utilisé pour générer la documentation.

B - Créez maintenant un deuxième pipeline `documentation.xml` qui ne doit se déclencher que lorsque la pull-request sera **fusionnée** (merged).
Pour cela consultez :

https://docs.github.com/en/actions/writing-workflows/choosing-when-your-workflow-runs/events-that-trigger-workflows#pull_request

Vous utiliserez la commande maven : `mvn -B javadoc::javadoc`, qui génère la documentation java et ensuite l'action peaceiris/actions-gh-pages@v4 qui permet de déployer la documentation. Consultez

https://github.com/peaceiris/actions-gh-pages#readme

C - Pour que les pages apparaissent sur github, il faut configurer votre dépôt : onglet Settings puis Pages (menu latéral gauche). Dans la section Source, choisssez Deploy from a branch et dans la section Branch, choisissez gh-pages/root, puis Save.
La documentation devient accessible à l'adresse :

https://ceri-m1-devops.github.io/tpintegrationcontinue-votreIdentifiantGithub/