GINF4-B, DevOps, Contrôle 2

29.12.2025

Exercice 1

Vous devez développer une application web avec une équipe de 6 développeurs sur une période de 5 mois :

A. Choisissez entre une stratégie Gitflow ou Trunk-based Development, en justifiant votre choix.
La période de 5 mois est courte, et un développement Trunk-based permettra aux changements effectués et fonctionnalités ajoutées d'être livrés aux utilisateurs dès qu'elles sont prêtes.
(Toute réponse avec justification raisonnable a été acceptée)

B. Dans quel type de développement utilise-t-on principalement les Feature Flags ? à quoi servent-ils ?
Utilisées principalement dans le Trunk-based développement, permettant d'activer et désactiver une fonctionnalité en attendant qu'elle soit finalisée et testée, ou si elle cause des bugs.

C. L'équipe devrait-elle travailler sur un seul dépôt partagé ou chaque développeur doit-il créer un fork (copie) du dépôt sur son compte GitHub ? Justifiez votre réponse.
Un seul dépôt partagé, sinon les modifications des différents membres de l'équipes n'apparaitront pas dans le même dépôt.

D. Vous devez développer une nouvelle fonctionnalité. sur quelle branche créez-vous votre code ? sur votre machine locale ou sur le dépôt distant?
Créer une branche 'feature..' à partir de la branche 'main' si vous avez choisi de travailler avec un développement Gitflow, ou à partir de la branche 'develop' si vous avez choisi le Trunk-based. La branche est créée localement puis poussée vers le dépôt distant pour y effectuer un PR.

E. Expliquez le processus de PR et code review avant merge.
Après avoir poussé nos modifications vers la branche feature du dépôt distant, on soumet la PR (sur GitHub) pour demander de fusionner notre branche avec la branche source (main ou develop), ensuite on désigne les membres qui doivent réviser notre code (code review). Après leur révision, on corrige notre code s'ils ont demandé des correction, et finalement ils approuvent la demande et exécutent la fusion.

F. A quel moment devriez-vous faire un git pull ?
Avant de créer une branche, et avant de pousser son code vers le répositoire distant, et avant la fusion (ou de faire un PR), (toute réponse correcte a été acceptée)

G. Décrivez un scénario qui provoquerait un conflit de merge.
Deux développeur travaillent sur le même fichier, et modifient une même ligne. Le premier fusionne son code avec la branche main sur le dépôt distant, le deuxième fait un pull avant de fusionner son code (ou avant de faire le PR). Cela renverra un message de conflit de merge, et Git demandera au deuxième développeur de choisir quel code garder.

Exercice 2

Votre camarade a créé un fichier `.github/workflows/ci.yml` pour lancer les tests lint dans un premier job, puis lancer l'étape 'build and push' dans un deuxième job (si le premier job est effectué avec succès). Modifier ce fichier en respectant les consignes suivantes :

1. Lancer le workflow sur ubuntu et windows.
2. Corriger la séparation des jobs.
3. Ajouter un nom descriptif à chaque étape sans nom.
4. Ajouter un test de qualité de code des fichiers en JavaScript (.js), qui inclut la correction des erreurs identifiées par l'outil lint.
5. Ajouter la condition « `github.ref_name == 'main'` » au job docker.
6. Pousser vers DockerHub au lieu de GHCR.

```
name: CI pipeline

on:
  push:
    branches: ['main','develop']
    paths: ["path/to/app/**"]
  pull_request:
    branches: ['main','develop']
    paths: ["path/to/app/**"]
  schedule:
    - cron: '30 8 1 * *'

jobs:

  job1:
    strategy:
      matrix:
        os: [ubuntu-latest, windows-latest]
    runs-on: ${{ matrix.os }}
    defaults:
      run:
        working-directory: ./path/to/app/

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: 'latest'
          cache: 'npm'
          cache-dependency-path:
./path/to/app/package-lock.json

      - name: Installation des dépendances
        run: npm install
```

```
    - name: Tests lint (qualité de code)
      run: |
        npm run htmlhint **/*.html
        npm run stylelint **/*.css --fix
        npm run eslint **/*.js --fix

    - name: Tests unitaire
      run: npm jest

job2:
  strategy:
    matrix:
      os: [ubuntu-latest, macos-latest]
  runs-on: ${{ matrix.os }}
  needs: job1
  if github.ref_name == 'main'

  steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Connection à DockerHub
      uses: docker/login-action@v3
      with:
        username: ${{ secrets.DOCKER_USERNAME }}
        password: ${{ secrets.DOCKER_PASSWORD }}

    - name: Créer et publier l'image
      uses: docker/build-push-action@v5
      with:
        context: ./path/to/app/
        push: true
        tags: ${{secrets.DOCKER_USERNAME}}/ctrl_app
```

Répondre séparément à ces questions :

A. Est-ce que le workflow va se lancer si on modifie le fichier `'.github/workflows/ci.yml'` depuis la branche `'develop'` directement sur GitHub ?
Les événements déclencheurs concernent les modifications des fichiers se trouvant dans le chemin `path/to/app/**` uniquement, et donc la modification du workflow ne va jamais rien modifier.

B. à quoi servent les noms des jobs et des étapes ?
Ils apparaissent dans les logs et fenêtre Actions sur GitHub, ce qui permet d'identifier les jobs et étapes, suivre le progrès du worflow, et comprendre à quelle étape le workflow a éventuellement échoué (débuguer).

C. à quoi sert la condition ajoutée dans la question 5, en pratique ?
Cette condition permet au worfklow de ne générer et pousser l'image docker que si les modifications indiqués dans la partie 'on' (événement déclencheurs) sont effectuées sur la branche principale. On empêche la création de l'image lorsqu'on modifie la branche 'develop'.

D. Où doit-on définir la variable `'secrets.GITHUB_TOKEN'` ?
`'secrets.GITHUB_TOKEN'` est générée automatiquement par GitHub.
(`'secrets.DOCKER_USERNAME'` et `'secrets.DOCKER_PASSWORD'` doivent être définies dans les paramètres du dépôt (settings > secrets and variables > actions ..))

E. Quelle est la valeur de chacune des variables [`github.workflow, github.event_name`] dans ce workflow ?
`github.workflow : CI pipeline`
`github.job : job1 ou job2`
`github.event_name : push ou pull_request ou schedule`

F. Citer 2 fichiers de configurations nécessaires pour lancer les tests (lint ou unitaires). Spécifier leur chemin.
`path/to/app/.htmlhintrc`
`path/to/app/.stylelintrc.json`
`path/to/app/eslint.config.js`
`path/to/app/jest.setup.js`
`path/to/app/package.json`
`path/to/app/package-lock.json`

G. `'ctrl_app'` sera poussé sous quelle version ?
Par défaut `'latest'`, puisqu'on n'a pas spécifié une version.

Exercice 3

Vous avez créé une image Docker nommée `'myapp'` et lancé un conteneur `'myapp_c'` à partir de cette image avec les caractéristiques suivantes :

- L'application à l'intérieur du conteneur écoute sur le port 3000.
- Ce port doit être accessible via le port 5050 de votre machine locale
- Le dossier `'/data'` du conteneur est monté sur le volume `'data-volume'`,
- et le dossier `'/app'` est monté sur le dossier `'$PWD/app'` de votre machine locale.

A. Ecrire les lignes de commande de création de `'myapp'` et `'myapp_c'`.

```
docker build -t myapp .
docker run -d \
        --name myapp_c \
        -p 5050:3000 \
        -v data-volume:/data \
        -v "$PWD/app":/app \
        Myapp
```

B. Lequel doit-on pousser à un répositoire distant ?
On pousse l'image, donc `'myapp'`

C. Citer un avantage de Docker Hub et un autre de GHCR.
Docker : universel, populaire, registre par défaut
GHCR : intégré dans GitHub et GitHub Actions, plus de flexibilité sur les image privées

D. Ecrire toutes les commandes nécessaires pour pousser vers DockerHub.

```
docker login
docker tag myapp username/myapp:v1
docker push username/myapp:v1
```

E. Expliquer ces commandes utilisées dans les Dockerfile : `RUN, ARG, FROM, WORKDIR.`
FROM : l'image de base
ARG : Définit des variables d'environnement qui seront disponibles uniquement pendant la construction de l'image.
RUN : Exécute des commandes pendant la construction de l'image
WORKDIR : Définit le répertoire de travail et s'applique à toutes les instructions suivantes

F. Décrire deux caractéristiques de chacune des stratégies de déploiement suivantes : Canary, Blue/Green.
Canary : mise à jour progressive, par pourcentage de population, ou régions spécifique. En cas de bugs, seuls ces utilisateurs sont affectés. Nécessite un système sophistiqué pour définir les utilisateurs exposés à la nouvelle version.
Bue/Green : utilise deux environnement de production identiques, tout le trafic est basculé vers la dernière version (green), en cas de bugs le rollback est rapide, la mise en place de deux environnements est coûteuse.