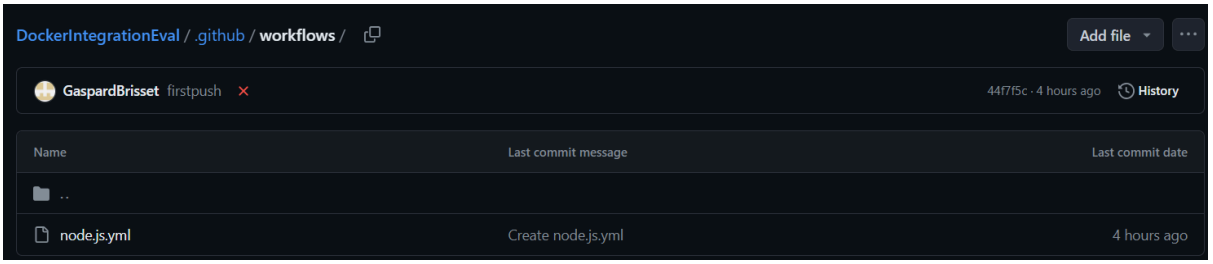
Docker intégration évaluation

Partie 1 :Configuration CI/CD avec GitHub

Création d’un GitHub

“https://github.com/GaspardBrisset/DockerIntegrationEval.git”

Création d’un workflows



Configuration du pipeline

```yaml
name: Node.js CI/CD Pipeline

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v2

      - name: Install Docker CLI
        run: |
```

```
          sudo apt-get update
          sudo apt-get install -y docker.io

      - name: Build Docker image
        run: |
          docker build -t my-node-app .
          # Assurez-vous d'ajuster le tag et le nom de l'image selon
votre besoin

  test:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v2

      - name: Install Dependencies
        run: npm install

      - name: Run Unit Tests
        run: npm test

  e2e-test:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v2

      - name: Install Dependencies
        run: npm install

      - name: Run End-to-End Tests
        run: npm run e2e
        # Assurez-vous d'ajuster la commande pour les tests bout en
bout

  deploy:
    runs-on: ubuntu-latest
    needs: [build, test, e2e-test]

    steps:
      - name: Install Docker CLI
        run: |
          sudo apt-get update
```

```
        sudo apt-get install -y docker.io

    - name: Deploy Docker image
      run: |
        # Écrire votre commande de déploiement d'image Docker ici
        # Exemple : docker push my-registry/my-node-app:latest

    - name: SSH Remote Commands
      uses: appleboy/ssh-action@v1.0.3
      with:
        host: ${{ secrets.SSH_HOST }}
        username: ${{ secrets.SSH_USERNAME }}
        password: ${{ secrets.SSH_PASSWORD }}
        port: ${{ secrets.SSH_PORT }}
        script: |
          # Écrire vos commandes SSH à exécuter à distance ici
          # Exemple : echo "Hello, world!"
```

Documentation de 2 problemes potentiel et leurs résolutions

Problème : Déploiement échoué en raison de dépendances manquantes ou incorrectes :

Lorsque vous déployez votre application, il peut arriver que certaines dépendances requises ne soient pas correctement installées dans l'environnement de déploiement, ou qu'elles soient de versions incorrectes. Cela peut entraîner des erreurs d'exécution ou des dysfonctionnements de votre application dans l'environnement de production ou de pré-production.

Solution :

Utilisez des outils de gestion des dépendances cohérents : Assurez-vous que votre pipeline CI/CD utilise les mêmes outils de gestion des dépendances que votre environnement de production. Par exemple, si vous utilisez npm pour gérer les dépendances de votre application Node.js, assurez-vous que votre pipeline CI/CD utilise également npm pour installer les dépendances.

Fournissez des fichiers de spécifications de dépendances cohérents : Utilisez des fichiers tels que package.json pour Node.js ou requirements.txt pour Python afin de spécifier les dépendances de votre application de manière cohérente. Assurez-vous de geler les versions des dépendances pour éviter les mises à jour non intentionnelles.

Problème : Erreurs de déploiement en raison de configurations incorrectes :

Il peut arriver que les erreurs de déploiement se produisent en raison de configurations incorrectes dans votre pipeline CI/CD ou dans l'environnement de déploiement lui-même. Cela peut inclure des erreurs dans les scripts de déploiement, des erreurs de configuration réseau, ou des erreurs de sécurité.

Solution :

Testez les scripts de déploiement localement : Avant de les inclure dans votre pipeline CI/CD, testez vos scripts de déploiement localement pour vous assurer qu'ils fonctionnent comme prévu. Cela vous permettra de détecter et de résoudre les erreurs plus tôt dans le processus de développement.

Utilisez des variables d'environnement sécurisées : Pour les informations sensibles telles que les noms d'utilisateur, les mots de passe et les clés API, utilisez des variables d'environnement sécurisées dans votre pipeline CI/CD. Assurez-vous de stocker ces variables dans un endroit sécurisé, tel que les secrets de votre dépôt GitHub, et de ne jamais les inclure dans le code source.

Partie 2 : Conteneurisation avec Docker

Création d’un Dockerfile

```
# Utiliser une image de base légère de Node.js
FROM node:14-alpine
# Définir le répertoire de travail dans le conteneur
WORKDIR /app
# Copier le package.json et package-lock.json (si disponible) dans le répertoire de travail
COPY package*.json ./
# Installer les dépendances
RUN npm install
# Copier le reste des fichiers de l'application dans le répertoire de travail
```

```
COPY . .
# Exposer le port sur lequel votre application s'exécute
EXPOSE 3000
# Commande pour démarrer l'application
CMD [ "npm", "start" ]
```

Gestion des conteneurs

```
# Utiliser une image de base légère de Node.js
FROM node:14-alpine
# Définir le répertoire de travail dans le conteneur
WORKDIR /app
# Copier les fichiers de l'application dans le répertoire de travail du conteneur
COPY . .
# Installer les dépendances de l'application
RUN npm install
# Exposer le port sur lequel votre application s'exécute
EXPOSE 3000
# Commande pour démarrer l'application
CMD [ "npm", "start" ]
```

Docker pour les bases de données de test

```
# Utiliser une image de base pour MySQL
FROM mysql:5.7
# Définir les variables d'environnement pour la base de données
ENV MYSQL_ROOT_PASSWORD=root
ENV MYSQL_DATABASE=test_db
ENV MYSQL_USER=test_user
ENV MYSQL_PASSWORD=test_password
# Copier le script SQL de peuplement de la base de données dans le répertoire de travail du conteneur
COPY init.sql /docker-entrypoint-initdb.d/
```