Documentation du Pipeline CI/CD

1. Présentation Générale

Ce pipeline CI/CD est défini avec GitHub Actions et a pour objectif d'automatiser l'exécution des tests et le déploiement de notre application. Il assure qu'à chaque modification du code, des tests sont lancés pour valider la qualité avant d'intégrer les changements en production. Le pipeline couvre à la fois le backend et le frontend, avec des jobs dédiés pour chacun, ainsi qu'un job de déploiement déclenché uniquement sur la branche de production (main).

---

2. Déclencheurs du Pipeline et Commandes Associées

Le pipeline se déclenche à partir de commandes et événements suivants :

- **Push :**
    - **Branche dev :** Branche d'intégration principale où chaque push déclenche l'exécution des tests.
    - **Branches dev-backend/** et dev-frontend/** :** Les push sur ces branches spécifiques au développement backend ou frontend déclenchent aussi les tests correspondants.
    - **Branche main :** Les push sur cette branche déclenchent le job de déploiement en production.
- **Pull Request (PR) :**
    - Lorsqu'une PR est créée vers les branches dev, dev-backend/** ou dev-frontend, le pipeline se déclenche pour exécuter les tests et s'assurer que le code proposé est fiable avant la fusion.

Commandes Clés dans les Étapes des Jobs

- **npm install :**
  Cette commande installe toutes les dépendances nécessaires à l'exécution des tests et au build de l'application. Elle est utilisée dans :
    - Les jobs **Test Backend** et **Test Frontend** avant l'exécution des tests.
    - Le job **Deploy** pour préparer le build des applications.
- **npm run test :**
  Cette commande lance les tests définis dans le fichier package.json. Elle est utilisée dans les jobs **Test Backend** et **Test Frontend** pour vérifier que le code fonctionne correctement.

- **npm run build :**
  Cette commande compile l'application en vue de sa mise en production. Elle est utilisée dans le job **Deploy** :
    - Pour le frontend : Compilation de l'application React en mode production.
    - Pour le backend (si nécessaire) : Exécution du build pour préparer l'application à être déployée.

Ces commandes sont exécutées automatiquement par GitHub Actions sur les branches concernées, garantissant ainsi que le pipeline s'exécute dans le bon contexte et sur la bonne base de code.

---

3. Structure et Fonctionnement des Jobs

Le pipeline est organisé en trois jobs principaux :

A. Test Backend

- **Objectif :** Vérifier la qualité du code côté backend.
- **Branches concernées :** S'exécute sur la branche dev et sur toute branche commençant par dev-backend.
- **Étapes clés :**
    - **Récupération du code :** Utilisation de l'action actions/checkout@v2.
    - **Configuration de Node.js :** Installation de Node.js version 16 via actions/setup-node@v3.
    - **Installation et test :**
        - **Commande :** npm install pour installer les dépendances.
        - **Commande :** npm run test pour exécuter les tests du backend, dans le répertoire backend.

B. Test Frontend

- **Objectif :** Assurer la qualité du code côté frontend.
- **Branches concernées :** S'exécute sur la branche dev et sur toute branche commençant par dev-frontend.
- **Étapes clés :**
    - **Récupération du code :** Utilisation de l'action actions/checkout@v2.
    - **Configuration de Node.js :** Installation de Node.js version 16.
    - **Installation et test :**
        - **Commande :** npm install pour installer les dépendances.

        - **Commande :** npm run test pour lancer les tests, dans le répertoire frontend/edifis-pro.

C. Déploiement en Production

- **Objectif :** Construire les artefacts de l'application et préparer le déploiement.
- **Branche concernée :** S'exécute uniquement sur la branche main.
- **Dépendances :** Ce job ne démarre qu'après la réussite des tests du backend et du frontend (via needs: [test-backend, test-frontend]).
- **Étapes clés :**
    - **Récupération du code :** Utilisation de actions/checkout@v2.
    - **Configuration de Node.js :** Installation de Node.js version 16.
    - **Build du Frontend :**
        - **Commande :** npm install pour installer les dépendances.
        - **Commande :** npm run build pour compiler l'application React, dans le répertoire frontend/edifis-pro.
    - **Build du Backend (si nécessaire) :**
        - **Commande :** npm install et npm run build dans le répertoire backend.
    - **Déploiement simulé :** Messages d'information pour indiquer le déploiement et l'archivage des artefacts.
    - **Archivage des Artefacts :**
        - Utilisation de actions/upload-artifact@v2 pour archiver le build du frontend et le code/artefact du backend.

---

4. Pourquoi ce Pipeline ?

Séparation des Responsabilités

- **Modularité :** Les tests sont distincts pour le backend et le frontend, facilitant la localisation des erreurs et optimisant le temps d'exécution.
- **Clarté :** Chaque job a une fonction bien définie, ce qui simplifie la maintenance et l'évolution du pipeline.

Sécurité et Fiabilité

- **Exécution conditionnelle :** L'utilisation de conditions (if) garantit que les jobs s'exécutent uniquement sur les branches appropriées, réduisant ainsi les risques d'exécutions inappropriées.
- **Validation avant déploiement :** Le job de déploiement est conditionné à la réussite des tests, assurant ainsi que seules les versions validées passent en production.

Optimisation des Ressources

- **Parallélisation :** Les jobs de tests peuvent s'exécuter en parallèle, ce qui accélère le processus d'intégration continue.
- **Archivage des Artefacts :** Conserve une trace des builds, facilitant la traçabilité et le rollback en cas de besoin.

---

5. Enjeux et Points d’Intérêt

Qualité et Automatisation

- **Tests Automatisés :** Permettent de détecter rapidement les erreurs et d'améliorer la qualité du code.
- **Feedback Immédiat :** Les développeurs bénéficient d'un retour rapide sur leurs modifications, ce qui permet une correction précoce des erreurs.

Déploiement Continu

- **Contrôle du Déploiement :** Le déploiement est réservé à la branche main, garantissant que seules les versions validées et stables sont mises en production.
- **Traçabilité :** L'archivage des builds offre une vue historique sur les déploiements, facilitant la gestion des versions et les retours arrière en cas d'incident.

Adaptabilité et Scalabilité

- **Gestion des Branches :** L'utilisation de branches spécifiques pour le backend et le frontend permet de gérer efficacement différents aspects du développement.
- **Évolution du Pipeline :** Le pipeline peut être étendu avec de nouvelles étapes (tests de sécurité, audits de performance, etc.) selon les besoins futurs.

Choix Technologiques

- **Node.js :** Choisi pour ses performances et sa compatibilité avec des outils modernes de test et de build.
- **GitHub Actions :** Offre une solution intégrée et simplifiée pour la gestion des workflows CI/CD sans nécessiter d'infrastructure supplémentaire.

---

6. Conclusion

Ce pipeline CI/CD constitue une solution robuste et efficace pour automatiser l'intégration, les tests et le déploiement de notre application. En s'assurant que le code est systématiquement validé par des tests avant d'être déployé, le pipeline permet de maintenir une production stable et de haute qualité. La séparation entre les tests backend et frontend, combinée à l'exécution conditionnelle des jobs, en fait un outil flexible, sécurisé et évolutif pour accompagner le développement du projet.