Compte rendu - DevOps

CI/CD : GitHub Actions & Render

Auteur : Jean Jireh | Fortune Alvarez
Date : 4 février 2026

1 Objectifs et Contexte

Objectifs du Lab

Ce laboratoire visait à automatiser le cycle de vie de notre application (Node.js + Redis) en mettant en place un pipeline complet :

- **Intégration Continue (CI) :** Exécution automatique des tests unitaires à chaque modification du code via **GitHub Actions**.
- **Déploiement Continu (CD) :** Mise en production automatique de l'application sur la plateforme **Heroku** si les tests passent.

2 Application dans le monde réel

L'automatisation CI/CD est la colonne vertébrale des entreprises tech modernes (Netflix, Spotify, etc.).

Rapidité	Les développeurs ne perdent plus de temps à tester et déployer manuellement.
Stabilité	Le code cassé est détecté immédiatement (avant la mise en prod) grâce aux tests automatiques.
Feedback Loop	Le développeur reçoit un retour immédiat sur la qualité de son code via les notifications GitHub.

3 Étape dans le cycle DevOps

Ce Lab couvre les étapes centrales de la boucle infinie DevOps :

— **Test (CI) :** Validation automatique.

— **Release & Deploy (CD) :** Livraison du logiciel sur l'environnement de production.

4 Problèmes rencontrés et Résolutions

Connexion Redis impossible dans la CI

Erreur "Connection Refused"

Symptôme : Lors du push sur GitHub, le workflow échoue avec l'erreur :
`Error: Redis connection to 127.0.0.1:6379 failed.`
Analyse : Les environnements virtuels de GitHub Actions (`ubuntu-latest`) sont vides. Ils n'ont pas de serveur Redis installé par défaut, contrairement à nos machines locales.

Résolution

Nous avons utilisé la fonctionnalité des **Service Containers** de GitHub Actions.

— Modification du fichier `.github/workflows/main.yml`.

— Ajout de la section `services: redis` utilisant l'image Docker officielle `redis`.

— Cela lance un conteneur Redis temporaire accessible sur le port 6379 pendant la durée des tests.

Choix technique : Déploiement sur Render

Alternative à Heroku

Contexte : Heroku ne proposant plus de plan gratuit sans carte bancaire, nous avons opté pour la plateforme **Render**, qui offre une solution moderne et gratuite pour l'hébergement d'applications Node.js.

Résolution

J'ai configuré le pipeline de Déploiement Continu (CD) ainsi :

— **Configuration Render :** Création d'un Web Service pointant vers le dépôt GitHub, avec le paramètre `Root Directory` réglé sur `lab`.

— **Deploy Hook :** Utilisation d'un webhook secret fourni par Render.

— **GitHub Actions :** Au lieu d'utiliser une action complexe, le job `deploy` exécute une simple requête `curl` vers ce webhook uniquement si les tests passent.

5 🚀 Finalité du Lab

Objectif atteint ? OUI.

Nous avons réussi à transformer un processus manuel en un pipeline automatisé :

1. **Push** du code → GitHub détecte le changement.
2. **Build & Test** → GitHub Actions lance Redis et `npm test`.
3. **Deploy** → Si (et seulement si) les tests sont verts, l'application est envoyée sur Render.