Grille de correction détaillée – EC06

Informations générales

- **Épreuve :** EC06 – CI/CD et versionning
- **Durée :** 4h – Individuelle
- **Projet fil rouge :** RebootCamp – Mise en place d'une chaîne DevOps complète
- **Nature :** Épreuve pratique (dépôt + scripts + doc)

Critères d'évaluation et attentes

Critère	Intitulé officiel	Attentes côté correcteur	Points de vigilance
C7.3	Structuration du dépôt Git	Branches organisées (`main` , `dev` , `feature/...`), commits clairs, historique propre.	Vérifier conventions de nommage, présence éventuelle de README au niveau repo.
C9.4	Pipeline CI/CD	Fichier de pipeline fonctionnel (GitHub Actions, GitLab CI…), étapes install → lint → test → build → deploy.	Pipeline doit tourner sans erreur, pas juste théorique. Logs ou captures attendus.
C9.5	Déploiement sécurisé via conteneurisation	Dockerfile multi-stage optimisé, scripts de déploiement, gestion sécurisée des variables d'environnement.	Vérifier absence de secrets en clair, image finale légère, reproductibilité assurée.

Livrables à corriger

- **Dépôt Git complet** (structure, branches, historique)
- **Fichier pipeline CI/CD** (`.gitlab-ci.yml` , `.github/workflows/ci.yml` , etc.)
- **Dockerfile** optimisé et fonctionnel
- **Scripts de déploiement** (`deploy.sh` , `run.sh` , etc.)
- **README** détaillant les étapes, l'utilisation et les bonnes pratiques mises en place

Barème indicatif (sur 20 points)

Axe évalué	Points
Structuration Git (C7.3)	/6
Pipeline CI/CD complet (C9.4)	/7
Déploiement sécurisé + Docker (C9.5)	/7

> **Tolérance :** ±2 points selon la qualité de la documentation et la robustesse technique.

Points positifs attendus

- Dépôt Git clair : branches, tags, commits lisibles
- Pipeline CI/CD automatisé incluant lint, tests, build et déploiement
- Dockerfile multi-stage avec image finale optimisée et légère
- Gestion sécurisée des secrets (variables d'environnement, vault, GitHub secrets…)
- Documentation claire dans le README, incluant instructions locales et CI/CD
- Mise en avant de **bonnes pratiques éco-responsables** (jobs réduits, images sobres, optimisation du temps de build)
- Captures ou logs de builds réussis (preuve d'exécution)

Erreurs fréquentes à surveiller

- Dépôt Git désorganisé (commits "WIP", branches manquantes ou confuses)
- Pipeline incomplet ou non fonctionnel (jobs non exécutés, étapes manquantes)
- Dockerfile non optimisé (image trop lourde, absence de multi-stage)

- Secrets (mots de passe, tokens) laissés en clair dans le code ou pipeline
- README trop superficiel ou absent
- Absence de prise en compte de l'éco-conception (aucune réflexion sur sobriété ou optimisation)

Rappel pédagogique

Cette épreuve valide la capacité de l'apprenant à :

- Mettre en place une **stratégie Git claire et collaborative**
- Automatiser les processus de test, build et déploiement via **CI/CD**
- Conteneuriser une application de manière **optimisée et sécurisée**
- Comprendre et appliquer les enjeux de **sécurité et durabilité** dans une chaîne DevOps moderne
- Produire une documentation claire permettant la **reproductibilité**

Elle développe une posture de **développeur DevOps** attentif à la qualité, à la sécurité et à l'impact environnemental de ses choix.