Git & GitLab CI/CD : Des piliers pour la gestion de code moderne

Par l'équipe de Crescera Solutions

Introduction

À l'ère du développement agile et de l'intégration continue, la maîtrise des outils de gestion de code et d'automatisation est devenue un impératif pour toute équipe de développement moderne. Parmi ces outils, **Git** s'est imposé comme le système de contrôle de version décentralisé de référence, tandis que **GitLab**, avec sa solution intégrée de **CI/CD (Continuous Integration / Continuous Deployment)**, s'affirme comme un environnement DevOps complet. Leur combinaison forme une fondation robuste pour garantir un code fiable, testable et rapidement livrable.

Ce document propose un tour d'horizon approfondi des principes et des bénéfices de Git et GitLab CI/CD, et de la manière dont ils transforment la gestion du cycle de vie logiciel.

1. Git : Le système de gestion de versions incontournable

1.1 Historique et principes

Créé en 2005 par Linus Torvalds pour gérer le code source du noyau Linux, **Git** repose sur une architecture **décentralisée**. Contrairement aux systèmes centralisés comme Subversion, chaque contributeur dispose d'une copie complète du dépôt. Cela permet des actions hors-ligne, une gestion simplifiée des branches, et une meilleure résilience.

Principes clés de Git :

- **Snapshots, pas de diffs** : Git capture un état complet du projet à chaque commit.
- **Intégrité cryptographique** : Chaque commit est identifié par un hash SHA-1.
- **Branchements légers** : Créer une branche est quasi-instantané.
- **Fusion (merge) et rebase** : Des outils puissants pour intégrer les changements.

1.2 Flux de travail avec Git

Git permet de modéliser plusieurs workflows en fonction de la complexité de l'équipe :

- **Git Flow** : séparation claire entre développement, release, hotfix.
- **Trunk-Based Development** : un tronc principal stable avec des features courtes.
- **Fork & Pull Request** : souvent utilisé dans les projets open source.

1.3 Bonnes pratiques

- **Commits atomiques** et messages clairs.

- Utilisation cohérente des branches (feature/, bugfix/, hotfix/).
- Revue de code via Pull/Merge Requests.
- Utilisation de .gitignore, git stash, git rebase avec précaution.

2. GitLab : Une plateforme DevOps tout-en-un

2.1 Présentation de GitLab

GitLab est une plateforme web de gestion de code source qui va bien au-delà d'un simple dépôt Git. Elle propose des outils de gestion de projet (issues, kanban), des revues de code, une sécurité intégrée, des dashboards DevOps, et surtout une **chaîne CI/CD puissante**.

Contrairement à GitHub qui dépend de GitHub Actions pour la CI/CD, GitLab intègre directement l'automatisation des pipelines, rendant la mise en place de tests et déploiements plus naturelle.

2.2 Interface et fonctionnalités clés

- **Web IDE** pour coder et tester en ligne.
- **Merge Requests** avec approbation, pipelines liés, et policies.
- **Sécurité intégrée** : scans de dépendances, de conteneurs, de secrets.
- **Auto DevOps** : pipelines préconfigurés pour projets standards.

3. GitLab CI/CD : Intégration et déploiement continus

3.1 Concepts fondamentaux

CI/CD regroupe deux piliers majeurs du DevOps moderne :

- **CI (Continuous Integration)** : fusion régulière du code dans une branche principale, avec exécution automatique des tests pour détecter les régressions.
- **CD (Continuous Deployment / Delivery)** : automatisation du déploiement en environnement de test, de préproduction, voire de production.

GitLab CI/CD repose sur un fichier de configuration .gitlab-ci.yml à la racine du projet.

Exemple simple :

```
stages:
  - build
  - test
  - deploy

build:
  stage: build
  script:
    - make build

test:
  stage: test
  script:
    - make test

deploy:
  stage: deploy
  script:
    - ./deploy.sh
  only:
    - main
```

3.2 GitLab Runner

Le **GitLab Runner** est le composant qui exécute les jobs définis. Il peut être partagé, spécifique, tournant sur Docker, Kubernetes ou des machines virtuelles. Les exécutions peuvent être parallélisées, mises en cache, et conditionnées selon l'état du dépôt.

3.3 Avantages concrets

- **Tests automatisés à chaque commit**.
- **Déploiement fréquent et sans erreur**.
- **Moins de régressions en production**.
- **Feedback rapide pour les développeurs**.

4. Cas d'usage : Pipeline CI/CD typique

Prenons un projet web avec React côté client et une API REST en Node.js. Le pipeline CI/CD pourrait inclure les étapes suivantes :

1. **Lint & Build Frontend**
2. **Lint & Test Backend**
3. **Tests E2E avec Cypress**
4. **Build Docker des deux services**
5. **Scan de sécurité SAST/DAST**
6. **Déploiement automatique sur un environnement de staging**
7. **Déploiement manuel (via approval) en production**

GitLab permet d'ajouter des conditions (when: manual, only/except, rules:), des secrets sécurisés (variables d'environnement), et des triggers pour orchestrer le tout.

5. Bonnes pratiques DevOps avec Git et GitLab CI/CD

5.1 Stratégies de branches compatibles CI/CD

- Garder la branche main toujours déployable.
- Faire les merge via Pull/Merge Requests avec pipelines obligatoires.
- Utiliser des environnements de prévisualisation pour chaque branche (Review Apps).

5.2 Sécurité et qualité

- Activer les scans de dépendances (SCA).

- Intégrer les tests de code (lint, unitaires, couverture).
- Stocker les artefacts et images Docker en registre GitLab.

5.3 Observabilité

- Utiliser les **pipelines dashboards** pour visualiser l'état des builds.
- Activer les **logs de déploiement** et **monitoring** des environnements.
- Connecter GitLab à Prometheus ou Grafana si besoin.

Conclusion

L'alliance de **Git** et **GitLab CI/CD** représente aujourd'hui une colonne vertébrale solide pour le développement logiciel moderne. Ces outils favorisent l'agilité, la collaboration, l'automatisation, la qualité du code et la rapidité des livraisons.

Bien utilisés, ils permettent aux développeurs et DevOps de se concentrer sur la création de valeur plutôt que sur la gestion manuelle de l'intégration, des tests ou des déploiements.

Dans un monde où le time-to-market est critique, la maîtrise de Git et de GitLab CI/CD n'est plus un luxe, mais une nécessité stratégique.

Annexes

A. Commandes Git utiles

- git clone, git pull, git push
- git checkout -b feature/xyz
- git merge, git rebase
- git stash, git reset, git log

B. Ressources pour aller plus loin

- https://docs.gitlab.com/ee/ci/

- https://git-scm.com/doc
- https://about.gitlab.com/devops-tools/