CI/CD : Écosystème et Infrastructure

Emmanuel BRUNO
Université de Toulon, LIS UMR CNRS 7020
emmanuel.bruno@univ-tln.fr

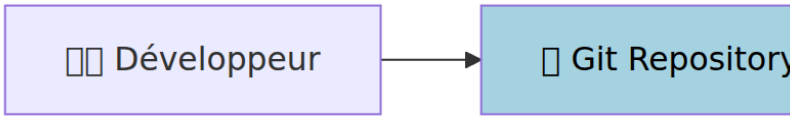
2026-10-17

Table des matières

1 Vue d'ensemble : Les composants d'un système CI/CD 1
2 L'architecture moderne CI/CD 1
3 Les composants essentiels 2
3.1 Serveur de build : Orchestration des pipelines 2
3.2 Environnements de test : Isolation et reproductibilité 3
3.3 Solutions modernes de test 3
3.4 Plateforme de qualité : Gardien du code 3
3.5 Registre d'artefacts : Stockage des binaires 4
3.6 Registre d'images : Hub des conteneurs 4
4 Infrastructure de déploiement : Pourquoi Kubernetes ? 5
5 Orchestrateurs : Kubernetes, Docker Swarm, Docker Compose 5
5.1 Solutions intégrées vs Best-of-breed .6
5.2 Solutions intégrées vs Best-of-breed .6
5.3 Avantages et inconvénients des deux approches 6
5.4 Tout est conteneurisé 7

1 Vue d'ensemble : Les composants d'un système CI/CD

Au-delà du simple pipeline : un écosystème complet



7 composants essentiels

1. **Repository Git** - Source de vérité
2. **Build Server** - Orchestrateur
3. **Test Environments** - Validation
4. **Quality Platform** - Gardien
5. **Artifact Registry** - Binaires
6. **Image Registry** - Conteneurs
7. **Infra Deployment** - Cible

Objectif : Comprendre le rôle et les outils de chaque composant

2 L'architecture moderne CI/CD

La vision complète : Du code à la production

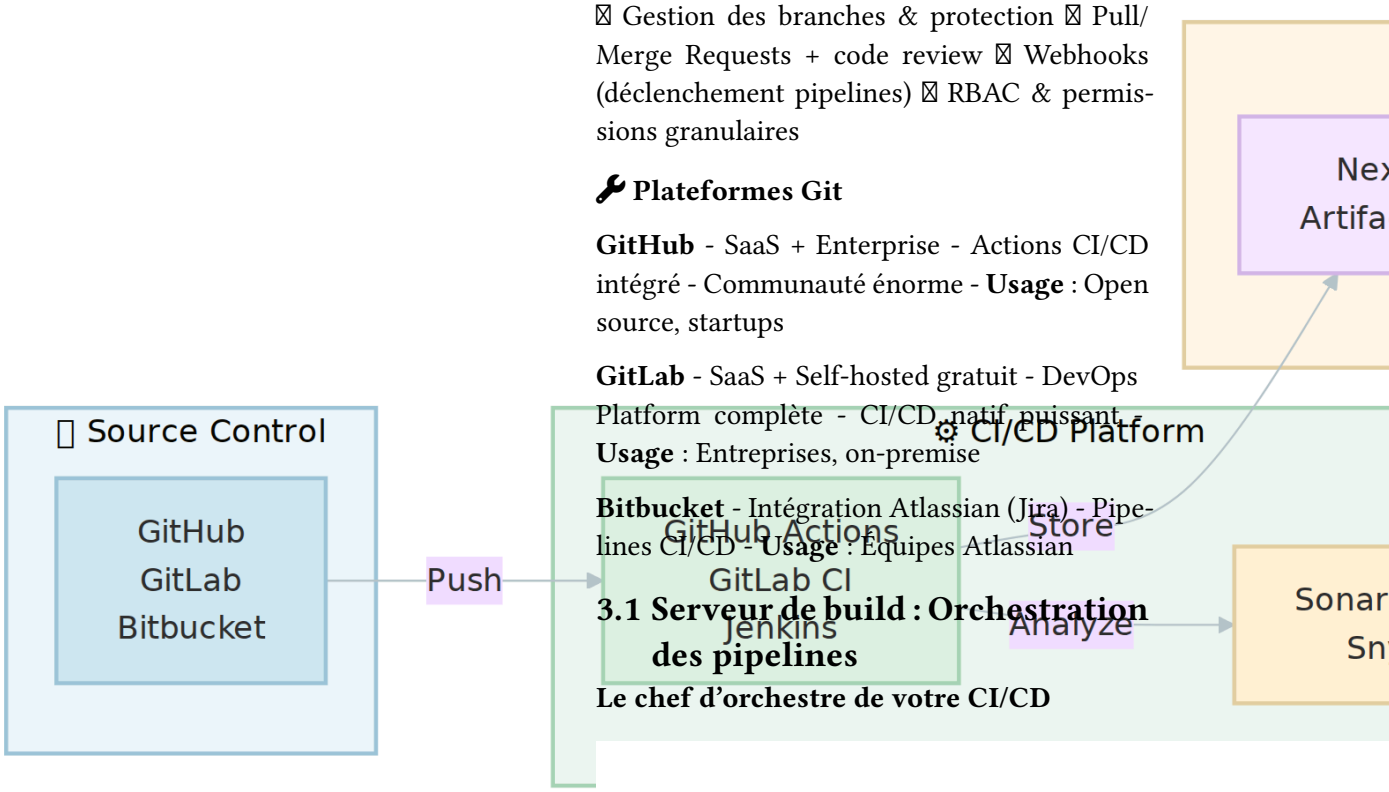
 Gestion des branches & protection Pull/Merge Requests + code review Webhooks (déclenchement pipelines) RBAC & permissions granulaires

Plateformes Git

GitHub - SaaS + Enterprise - Actions CI/CD intégré - Communauté énorme - **Usage** : Open source, startups

GitLab - SaaS + Self-hosted gratuit - DevOps Platform complète - CI/CD natif puissant - **Usage** : Entreprises, on-premise

Bitbucket - Intégration Atlassian (Jira) - Pipelines CI/CD - **Usage** : Équipes Atlassian



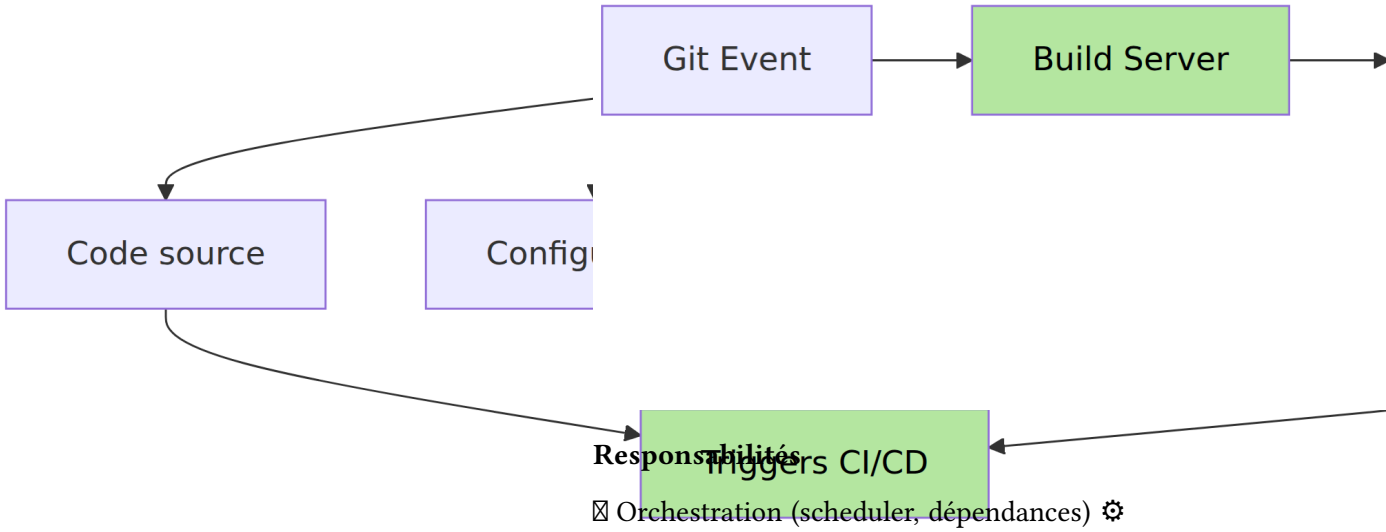
3.1 Serveur de build : Orchestration des pipelines

Le chef d'orchestre de votre CI/CD

3 Les composants essentiels

Repository Git : La source de vérité

Rôle central



Responsabilités

 Orchestration (scheduler, dépendances) ⚙ Exécution (runners, parallélisation) Reporting (logs, métriques, notifications)

Fonctionnalités essentielles

🔧 Plateformes CI/CD

GitHub Actions - Intégré GitHub - YAML simple, marketplace - **Usage** : Projets GitHub

GitLab CI - Intégré GitLab - Puissant, flexible - **Usage** : Workflow DevOps complet

Jenkins - Open source, extensible - Plugins pour tout - **Usage** : Legacy, on-premise complexe

CircleCI / Travis CI - SaaS, config simple - **Usage** : Startups, projets open source

3.2 Environnements de test : Isolation et reproductibilité

Tester dans les bonnes conditions

Types d'environnements

```
Environnements de test:

  Unit Tests:
    - JVM isolée
    - Mocks/Stubs
    - Pas de dépendances

  Integration Tests:
    - Base de données
    - Message broker
    - Services externes

  E2E Tests:
    - Application complète
    - Frontend + Backend
    - Données réalistes
```

3.3 Solutions modernes de test

Solutions modernes

☐ TestContainers

- DB dans Docker
- Éphémère et propre
- Même version que prod

☸ Preview Environments

- K8s namespace par PR
- URL unique
- Auto-cleanup

🌐 Service Virtualization

- Mock APIs externes
- Contrôle des scénarios
- Pas de dépendances

3.4 Plateforme de qualité : Gardien du code

La qualité comme prérequis, pas comme option



Analyses effectuées

☐ Code Quality (bugs, smells, complexité) 🔨 Test Coverage (lignes, branches) 🔒 Security

(CVE, OWASP, secrets) ⊠ Trends (dette technique)

Outils de qualité

SonarQube - Leader du marché - Self-hosted ou Cloud - Quality Gates - **Usage** : Toutes tailles d'équipe

Snyk - Focus sécurité (deps, code, containers) - Auto-fix PRs - **Usage** : Security-first teams

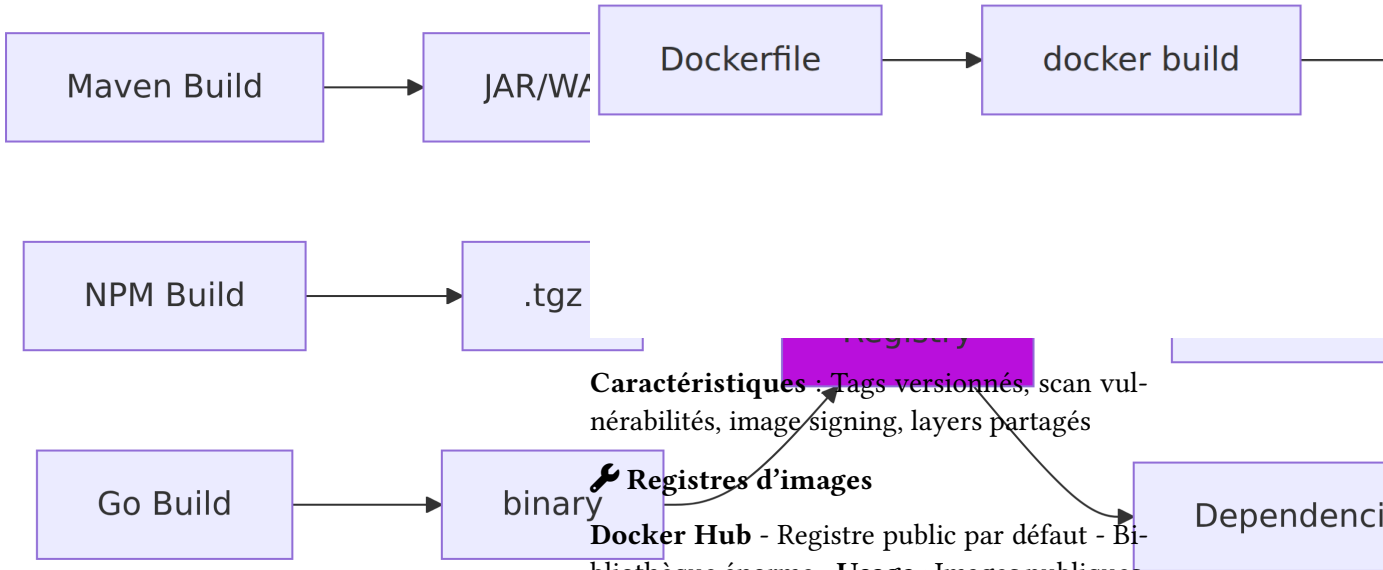
CodeClimate - SaaS simple - Maintenabilité focus - **Usage** : Startups, open source

Trivy - Open source - Scan containers + IaC - **Usage** : CI/CD security scans

3.5 Registre d'artefacts : Stockage des binaires

Stocker les résultats du build de manière persistante

Qu'est-ce qu'un artefact ?



Types : JAR/WAR, npm, wheels, binaries, NuGet

Bénéfices : Traçabilité, reproductibilité, partage, cache

Registres d'artefacts

Sonatype Nexus - Repository manager mature - Multi-format (Maven, npm, PyPI...) - Open source disponible - **Usage** : Entreprises, on-premise

JFrog Artifactory - Universal artifact repository - Cloud-native - 30+ formats - **Usage** : Cloud, scaling

GitHub Packages - Intégré GitHub - Zéro configuration - **Usage** : Projets GitHub

3.6 Registre d'images : Hub des conteneurs

Stocker et distribuer les images Docker

Caractéristiques : Tags versionnés, scan vulnérabilités, image signing, layers partagés

Registres d'images

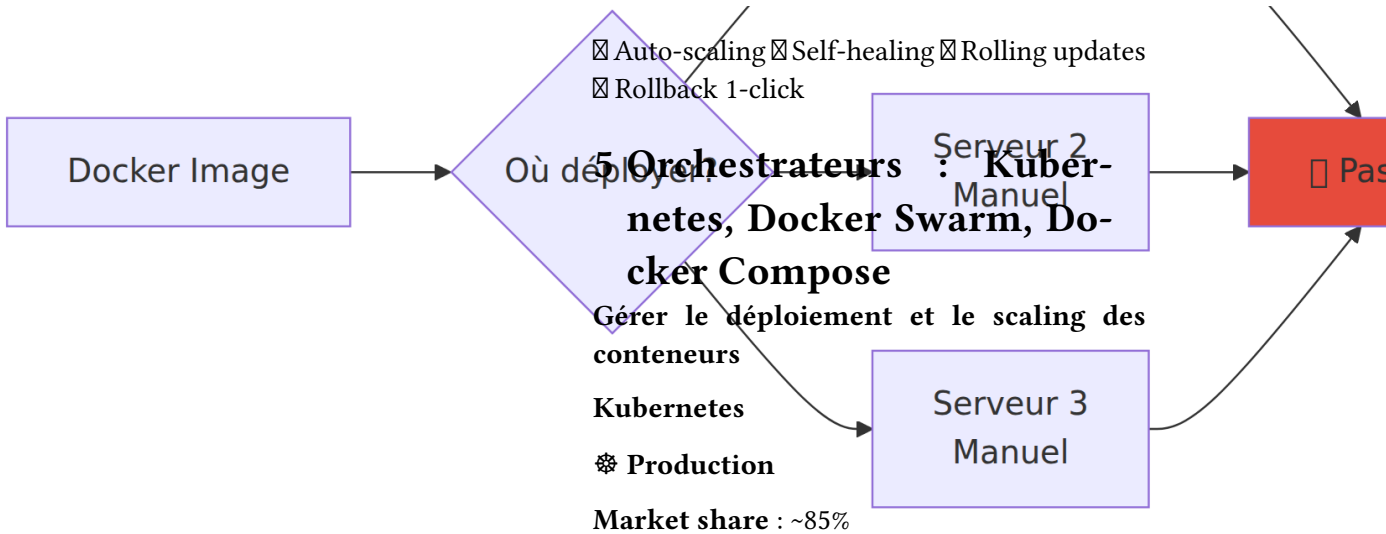
Docker Hub - Registre public par défaut - Bibliothèque énorme - **Usage** : Images publiques, dev **Harbor** - Open source self-hosted - Vulnerability scan intégré - RBAC avancé - **Usage** : Entreprises, on-premise **AWS ECR / GCP Artifact Registry** - Managed cloud - Intégration

native IAM - Scan automatique - **Usage** : Apps cloud (AWS/GCP)

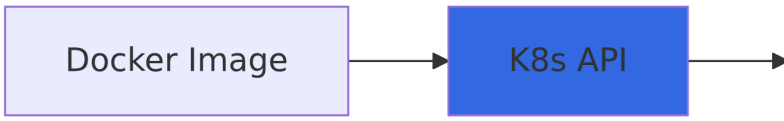
4 Infrastructure de déploiement : Pourquoi Kubernetes ?

L'orchestrateur de conteneurs devenu standard

Avant Kubernetes



Avec Kubernetes



✅ Auto-scaling ✅ Self-healing ✅ Rolling updates ✅ Rollback 1-click

5 Orchestrateurs : Kubernetes, Docker Swarm, Docker Compose

Gérer le déploiement et le scaling des conteneurs

Kubernetes

☸ **Production**

Market share : ~85%

Managed : EKS (AWS) · GKE (Google) · AKS (Azure)

Usage : Production, toutes échelles

Docker Swarm

✅ **Petites équipes**

Market share : ~5%

Usage : petites équipes · migration Compose · quand K8s est overkill

Docker Compose

⌨️ **Dev / Prototypage**

Usage : dev local · prototypage · tests IT CI/CD · démo

⚠ **Non recommandé en production**

Kubernetes

✅ Feature-rich ✅ Écosystème énorme ✅ Multi-cloud ✅ Auto-scaling ✅ Self-healing ✅ GitOps ready ❌ Complexité ❌ Courbe apprentissage

Docker Swarm

✅ Simple setup ✅ Intégré Docker ✅ Courbe apprentissage douce ✅ Suffisant PME

❌ Moins de features ❌ Écosystème limité ❌ Pas de GitOps natif

Docker Compose

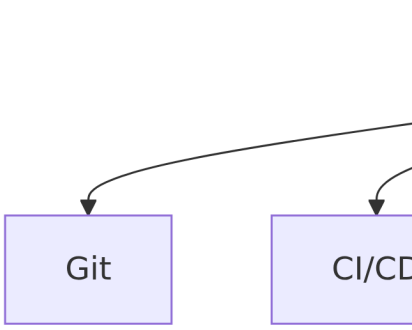
✅ Fichier YAML simple ✅ Démarrage instantané ✅ Idéal environnement local ✅ Reproductibilité dev

❌ Single-host uniquement ❌ Pas de HA ❌ Pas pour production

5.1 Solutions intégrées vs Best-of-breed

Deux philosophies d'architecture CI/CD

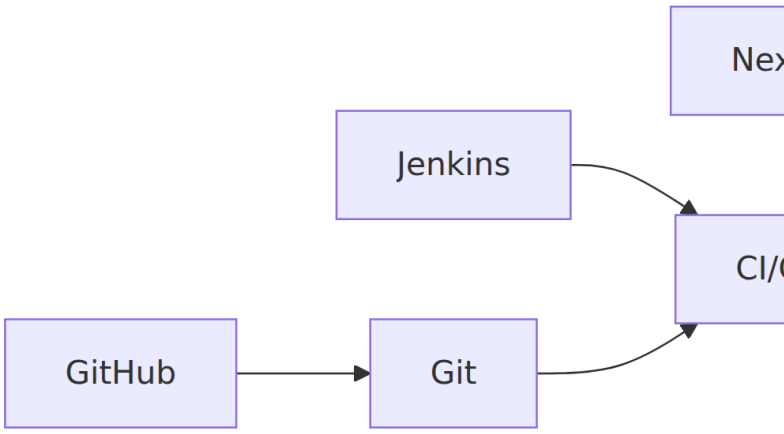
🏢 **All-in-One (Integrated)**



Exemples : GitLab DevOps · Azure DevOps · JFrog · Atlassian

5.2 Solutions intégrées vs Best-of-breed

🔧 **Best-of-Breed (Composable)**



Approche : Meilleur outil pour chaque fonction

5.3 Avantages et inconvénients des deux approches

🏢 **All-in-One (Integrated)**

Avantages : ✅ Intégration native ✅ Single UI/UX ✅ Un seul vendor ✅ Setup rapide ✅ Support unifié

Inconvénients : ❌ Vendor lock-in ❌ Moins flexible ❌ Pas toujours « best tool »

🔧 **Best-of-Breed (Composable)**

Avantages : ✅ Meilleur de chaque catégorie ✅ Flexibilité maximale ✅ Portable (multi-cloud) ✅ Éviter vendor lock-in

Inconvénients : ✕ Intégration complexe ✕ Multiple vendors ✕ Maintenance ++ ✕ Courbe apprentissage

5.4 Tout est conteneurisé

Runners / agents CI (conteneurs)

- Scalables et éphémères
- Isolation des builds
- Démarrage rapide

Environnements de test éphémères

- TestContainers, namespaces K8s par PR
- Reproductibilité des scénarios
- Nettoyage automatique

Outils de qualité conteneurisés

- SonarQube, Snyk, Trivy
- Scans CI intégrés (SAST/SCA)
- Rapports et Quality Gates

Applications déployées en conteneurs

- Images versionnées et signées
- Tags par environnement (dev/pr/staging/prod)
- Scan vulnérabilités avant push

Infrastructure as Code (IaC) + conteneurs

- Terraform / Helm / Kustomize
- Déploiement déclaratif et reproductible
- Intégration avec GitOps pour la synchronisation