

CI/CD – Principes & Mise en place

Automatiser l'intégration, les tests et le déploiement pour livrer du code de qualité en continu

01

Comprendre la CI/CD

Les concepts fondamentaux de l'intégration et du déploiement continus

02

Identifier les étapes d'un pipeline

De l'installation des dépendances au déploiement en production

03

Préparer la mise en place

Les outils et bonnes pratiques pour démarrer efficacement

Qu'est-ce que la CI/CD ?

CI

Continuous Integration

Intégration fréquente du code dans la branche principale avec validation automatique

CD

Continuous Delivery

Le code est toujours prêt à être déployé en production après validation

CD

Continuous Deployment

Déploiement automatique en production après passage des tests

L'automatisation complète des tests, de la qualité du code et du déploiement pour une livraison rapide et fiable

Pourquoi adopter la CI/CD ?



Réduire les erreurs humaines

Automatisation des tâches répétitives et sujettes aux erreurs



Feedback rapide

Détection immédiate des problèmes à chaque commit



Garantir la qualité

Tests automatiques et analyses de code systématiques



Déployer avec confiance

Processus reproductible et traçable à chaque livraison

CI

Continuous Integration

L'intégration continue en pratique

git

Intégration fréquente

Les développeurs mergent leur code plusieurs fois par jour dans la branche principale



Déclenchement automatique

Le pipeline démarre automatiquement à chaque push ou pull request



Validation systématique

Tests unitaires, analyses de qualité et build exécutés automatiquement

Étapes typiques d'un pipeline CI

1

Installation des dépendances

npm install, pip install, maven download...

2

Lint & Formatage

Vérification du style et des conventions de code

3

Tests unitaires

Exécution de la suite de tests avec couverture

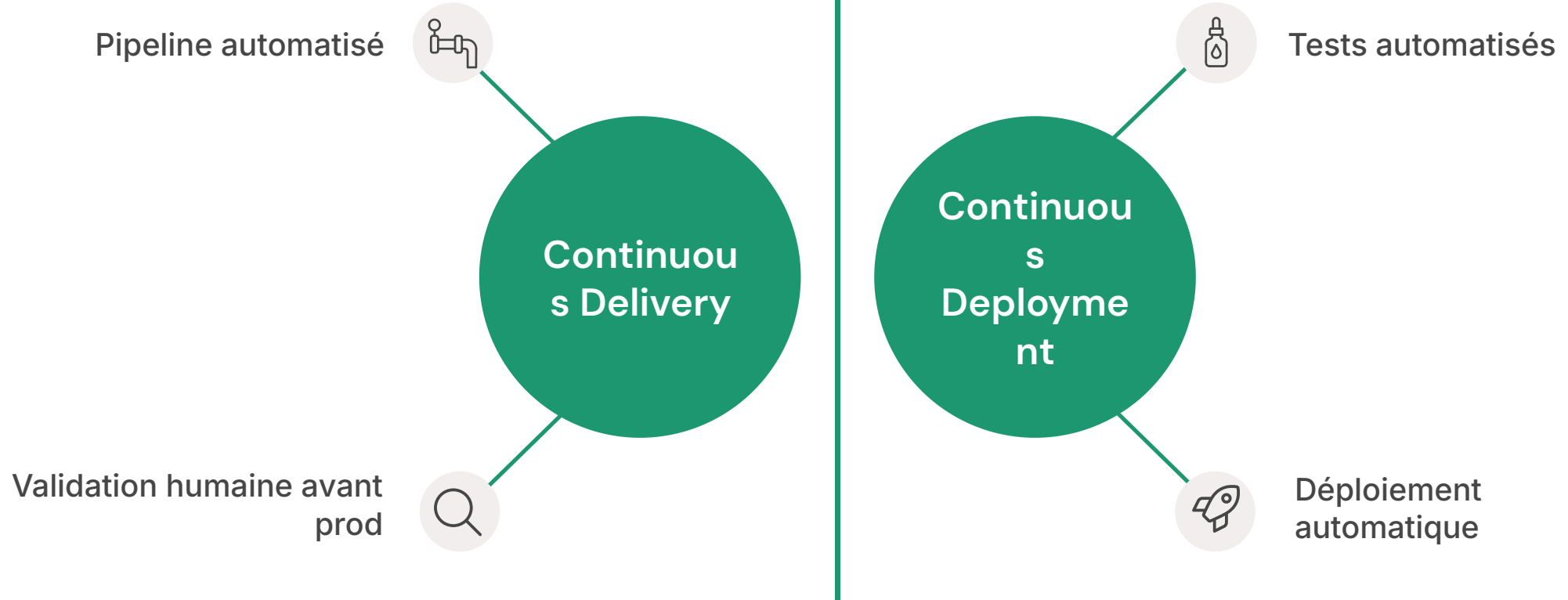
4

Build

Compilation et génération des artefacts

📌 **Principe clé :** Chaque étape doit réussir avant de passer à la suivante. Un échec stoppe le pipeline.

Continuous Delivery vs Deployment



Continuous Delivery

Le code est **toujours prêt** à être déployé en production

- Nécessite une validation humaine
- Déploiement manuel sur décision
- Contrôle total du timing

Continuous Deployment

Déploiement **automatique** en production

- Aucune intervention humaine
- Livraison instantanée des features
- Confiance totale dans les tests

Anatomie d'un Pipeline CI/CD

1

Build

Compilation du code, installation des dépendances

2

Test

Tests unitaires, tests d'intégration, couverture de code

3

Quality

Analyse statique, linting, vérification de sécurité

4

Deploy

Déploiement vers staging puis production

Un pipeline efficace transforme chaque commit en un candidat potentiel pour la production

Variables & Secrets

Gérer la configuration de manière sécurisée

Variables d'environnement

Configuration spécifique à chaque environnement (dev, staging, prod)

- URLs des API
- Noms de bases de données
- Configurations de feature flags

Secrets chiffrés

Informations sensibles stockées de manière sécurisée

- Clés API et tokens d'accès
- Mots de passe et certificats
- Credentials cloud (AWS, Azure, GCP)

Règle d'or : zéro hardcoding

Jamais de mots de passe ou secrets dans le code source ou les fichiers de configuration versionnés

Outils CI/CD populaires



GitHub Actions

Intégré directement dans GitHub

- Configuration en YAML
- Marketplace d'actions réutilisables
- Gratuit pour projets publics



GitLab CI

Solution complète DevOps intégrée

- Runners auto-hébergés ou cloud
- Pipeline as code (.gitlab-ci.yml)
- Visualisation avancée des pipelines

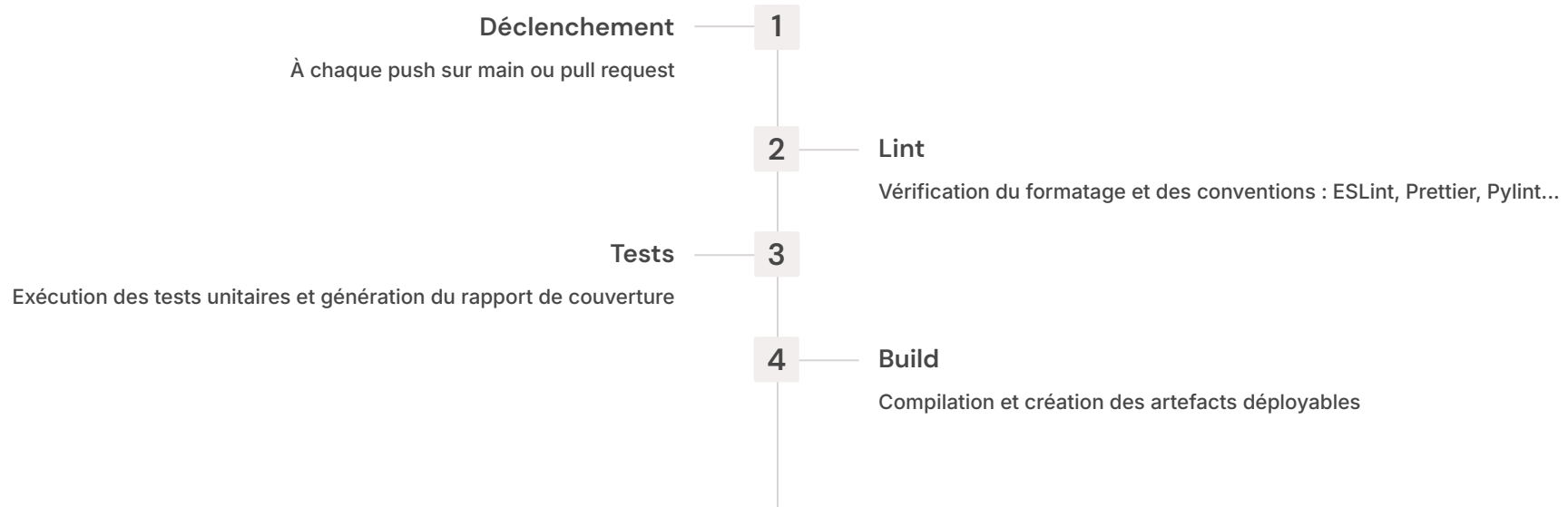


Jenkins

Solution open source historique

- Très flexible et extensible
- Écosystème de plugins riche
- Nécessite maintenance serveur

Exemple : Pipeline minimal



```
# Exemple GitHub Actions
on: [push, pull_request]
jobs:
  ci:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - run: npm install
      - run: npm run lint
      - run: npm test
      - run: npm run build
```

Bonnes pratiques CI/CD

Étapes courtes

Chaque job doit s'exécuter en quelques minutes maximum

Pipeline rapide = feedback rapide

Logs clairs

Messages d'erreur explicites et contextuels

Facilite le debugging en cas d'échec

Fail fast

Un échec stoppe immédiatement le pipeline

Pas de déploiement de code défectueux

-
- Tester le pipeline localement avant de push
 - Utiliser le cache pour accélérer les builds
 - Versionner la configuration du pipeline
 - Monitorer les temps d'exécution et optimiser

Lien avec Docker & Kubernetes



📌 **GitOps** : Le déploiement Kubernetes peut lui-même être géré via Git, avec des outils comme ArgoCD ou Flux qui synchronisent automatiquement l'état du cluster avec le repository.

Si ce n'est pas automatisé, ce n'est pas fiable.

L'automatisation n'est pas une option

Dans un monde où la vitesse de livraison et la qualité sont essentielles, la CI/CD devient le fondement d'une pratique DevOps mature et efficace.

✓ TESTEZ SOUVENT

🔑 DÉPLOYEZ EN CONFIANCE

🔄 ITÉREZ RAPIDEMENT