









# Pourquoi l'arrivée du DevOps ?

Évolution des pratiques de développement et d'exploitation

Octobre 2025

# Agenda

-  Les défis du développement logiciel traditionnel
-  Les bonnes pratiques issues du développement logiciel
-  L'Infrastructure as Code (IaC)
-  Le développement logiciel moderne (Agilité, 12facteurs, micro-services)
-  La chaîne de production logicielle
-  L'intégration, le déploiement et la livraison continus

# Les défis du développement logiciel traditionnel

## Silos organisationnels

Séparation stricte entre les équipes de développement et d'opérations, entraînant des problèmes de communication et de collaboration.

## ⚡ Cycles de développement longs

Processus séquentiels rigides avec des cycles de livraison pouvant s'étendre sur plusieurs mois.

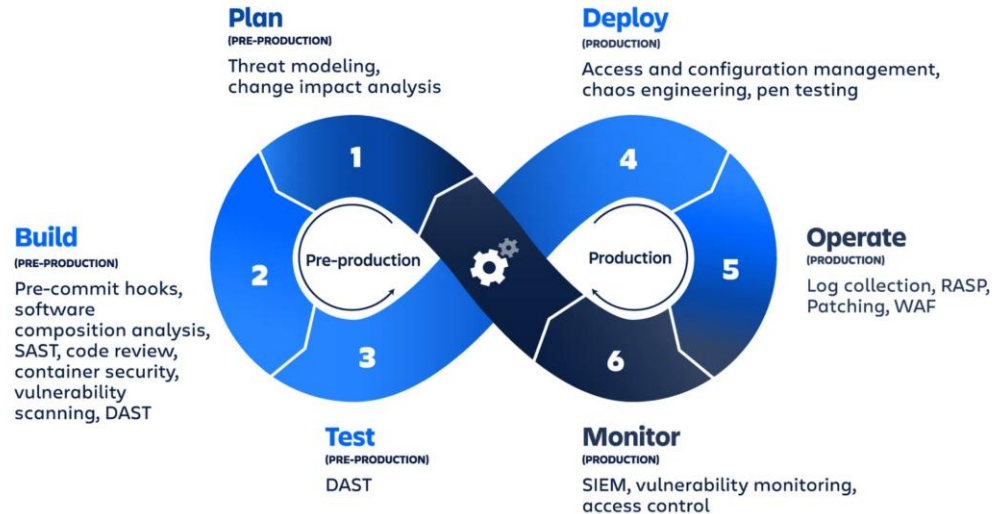
## ⚠️ Déploiements risqués

Déploiements manuels, propices aux erreurs et souvent réalisés pendant les nuits ou week-ends.

## 🤖 Manque d'automatisation

Processus manuels répétitifs consommant du temps et des ressources, sources d'erreurs humaines.

## DevSecOps



# Les bonnes pratiques issues du développement logiciel

## Pratiques Agiles

Livraison incrémentale, collaboration continue, adaptation au changement et amélioration constante.

## Développement piloté par les tests

Écriture des tests avant le code, cycle court : test, code, refactoring.

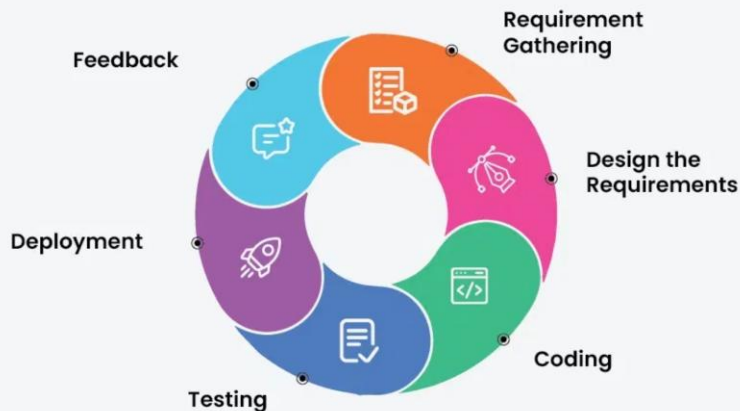
## Revue de code

Vérification par les pairs, partage des connaissances, détection précoce des problèmes.

## Intégration continue

Fusion fréquente du code dans un référentiel partagé, détection rapide des problèmes d'intégration.

### Phases of Agile SDLC



# L'Infrastructure as Code (IaC)

## Définition

Gestion et provisionnement de l'infrastructure à travers du code versionné plutôt que par des processus manuels.

## Principes clés

Reproductibilité des environnements

Idempotence des opérations

Versionnement de l'infrastructure

Testabilité avant déploiement

## Avantages

Cohérence entre environnements, rapidité de déploiement, traçabilité des modifications, réduction des erreurs manuelles.

## Outils populaires

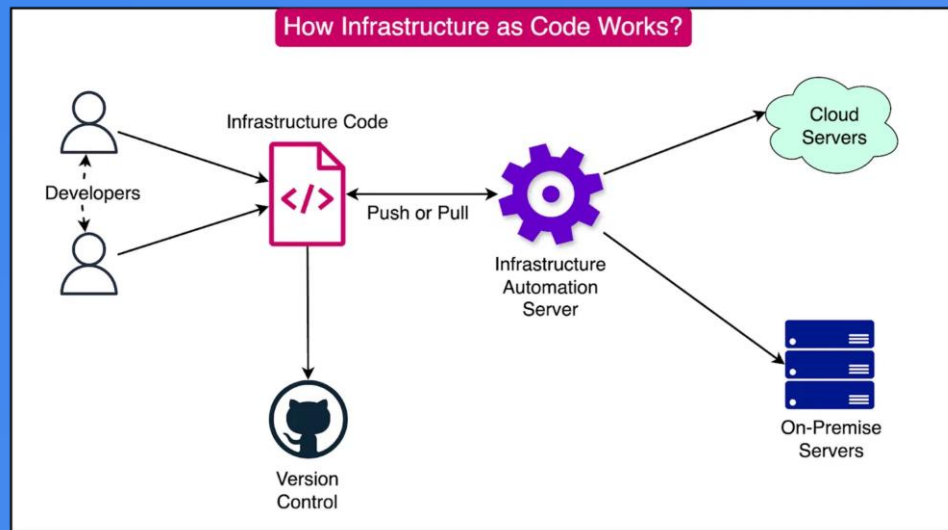
 Terraform

 Ansible

 Chef

 Puppet

 CloudFormation



# Le développement logiciel moderne - Agilité

## Manifeste Agile

Valorise les individus et leurs interactions, les logiciels opérationnels, la collaboration avec les clients et l'adaptation au changement.

## Frameworks

Différentes méthodologies pour mettre en œuvre les principes agiles :

Scrum

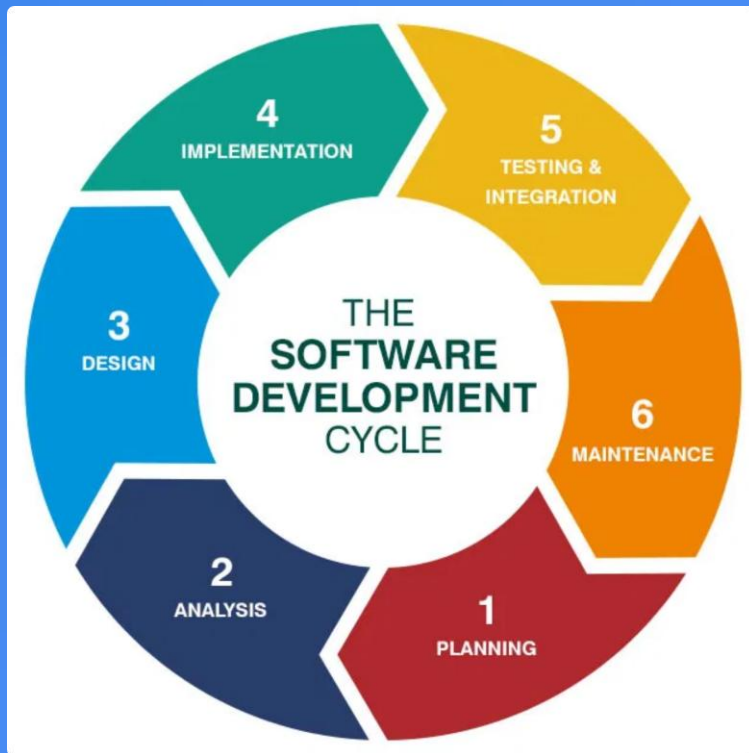
Kanban

XP

SAFe

## Bénéfices pour DevOps

- Livraisons fréquentes et itératives
- Collaboration entre équipes
- Adaptation rapide aux changements
- Amélioration continue des processus



# Le développement logiciel moderne - 12facteurs

Méthodologie pour construire des applications cloud-natives robustes, évolutives et maintenables, particulièrement adaptées aux environnements DevOps.

**1** Base de code unique  
Maintenir une seule base de code pour le projet.

**3** Configuration dans l'environnement  
Stocker la configuration dans l'environnement.

**5** Build, release, run séparés  
Séparer les étapes de construction, de publication et d'exécution.

**7** Binding de port  
Utiliser le binding de port pour les services.

**9** Jetabilité (démarrage rapide)  
Concevoir pour un démarrage rapide.

**11** Logs comme flux d'événements  
Concevoir des processus. Traiter les logs comme des flux d'événements sans état.

**2** Dépendances explicites  
Déclarer explicitement toutes les dépendances.

**4** Services externes comme ressources  
Traiter les services externes comme des ressources.

**6** Processus sans état  
Concevoir des processus sans état.

**8** Concurrence via processus  
Gérer la concurrence via des processus.

**10** Parité dev/prod  
Assurer une parité entre les environnements de développement et de production.

**12** Processus d'administration  
Concevoir des processus. Gérer les processus d'administration sans état.

## 12 Factor App Methodology

1. Codebase

2. Dependencies

3. Config

4. Backing services

5. Build, release, run

6. Processes

7. Port binding

8. Concurrency

9. Disposability

10. Dev/prod parity

11. Logs

12. Admin processes

# Le développement logiciel moderne - Microservices

## 🔗 Architecture microservices

Approche architecturale où une application est composée de petits services indépendants, chacun exécutant un processus unique et communiquant via API.

## ❖ Caractéristiques clés

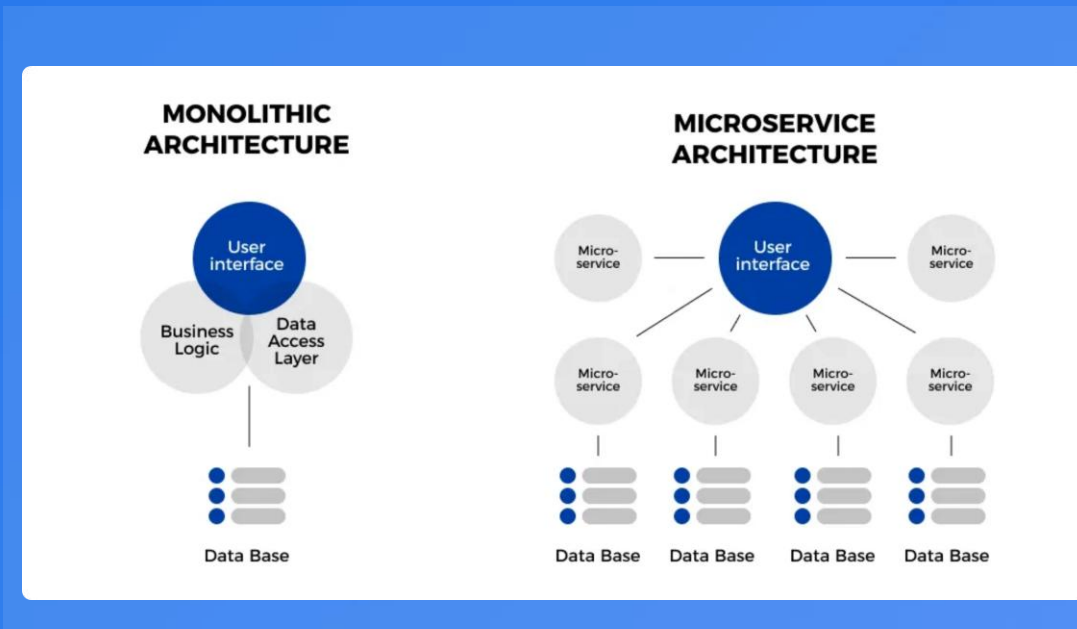
Services autonomes, spécialisés, déployables indépendamment, avec leur propre base de données et communiquant via API bien définies.

## ⊕ Avantages

Scalabilité fine, résilience accrue, flexibilité technologique, déploiements indépendants et équipes autonomes.

## ⚠ Défis

Complexité distribuée, tests d'intégration, monitoring, cohérence des données et communication inter-services.





# La chaîne de production logicielle

## Composants clés

💎 Gestion de code source

🚀 Déploiement

📦 Gestion des dépendances

🔧 Build automation

✍️ Tests automatisés

📋 Analyse de qualité

📁 Gestion des artefacts

## Flux de travail typique

- 1 Développement local
- 2 Commit et push vers le dépôt central
- 3 Déclenchement du pipeline CI/CD
- 4 Build et tests
- 5 Analyse de qualité
- 6 Création d'artefacts
- 7 Déploiement dans les environnements
- 8 Monitoring et feedback



# L'intégration, le déploiement et la livraison continus

## 🔗 Intégration continue (CI)

Fusion et test automatisés du code à chaque modification, permettant une détection rapide des problèmes d'intégration.

## 🚚 Livraison continue (CD)

Automatisation jusqu'à la préparation du déploiement, avec validation manuelle finale pour la mise en production.

## 🚀 Déploiement continu

Automatisation complète jusqu'à la production, chaque modification validée étant automatiquement déployée.

## 🔧 Outils populaires

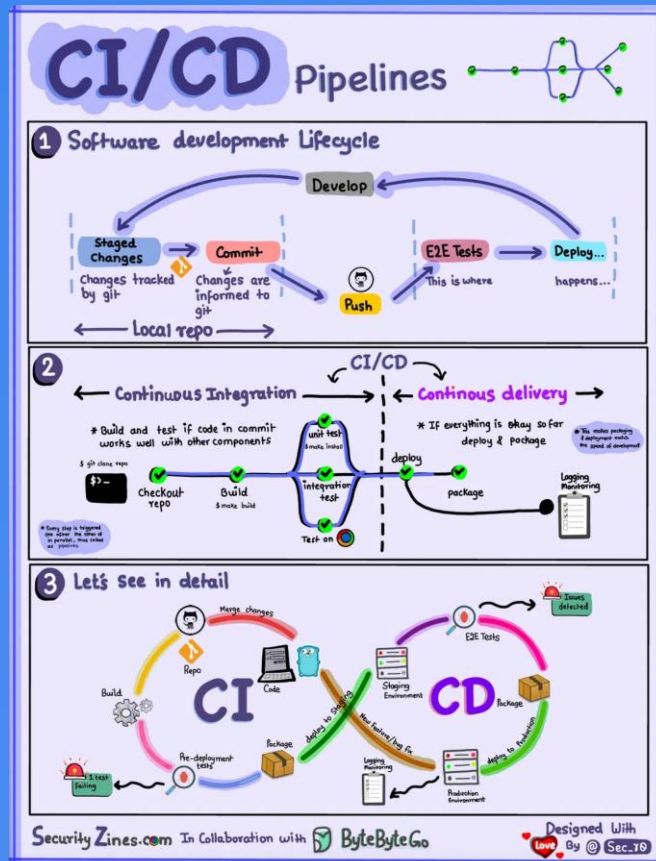
📦 Jenkins

🏠 GitLab CI/CD

🔄 GitHub Actions

🔄 CircleCI

🏠 Azure DevOps



# Synthèse : Pourquoi DevOps ?

DevOps répond aux défis du développement logiciel traditionnel en combinant les meilleures pratiques de développement et d'opérations pour créer une approche unifiée et collaborative.



## Collaboration accrue

Élimination des silos organisationnels et création d'une culture de responsabilité partagée entre développement et opérations.



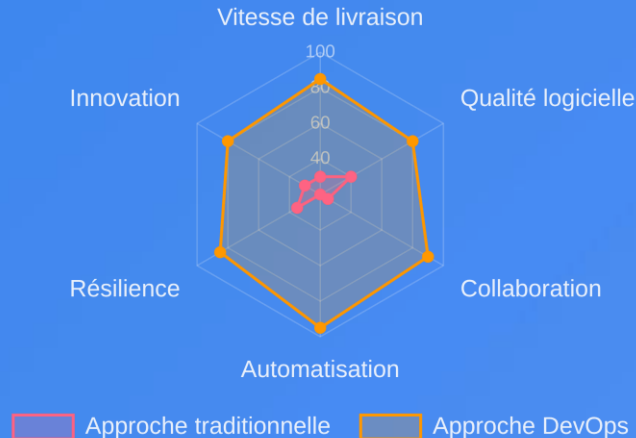
## Livraison accélérée

Cycles de développement plus courts, déploiements plus fréquents et fiables, et time-to-market réduit grâce à l'automatisation.



## Qualité améliorée

Détection précoce des problèmes, tests automatisés et feedback continu conduisant à des produits plus stables et fiables.



# Conclusion et perspectives

## DevOps : une évolution naturelle

DevOps représente l'évolution naturelle des pratiques de développement et d'exploitation, répondant aux défis du développement logiciel traditionnel. En combinant les meilleures pratiques, outils et cultures, DevOps permet d'accélérer la livraison de valeur tout en maintenant la qualité et la stabilité.



### DevSecOps

Intégration de la sécurité dès le début du cycle de développement, transformant DevOps en DevSecOps pour répondre aux enjeux croissants de cybersécurité.



### GitOps

Utilisation de Git comme source unique de vérité pour la gestion de l'infrastructure et des déploiements, renforçant l'automatisation et la traçabilité.



### FinOps

Optimisation des coûts cloud par la responsabilisation des équipes et la visibilité des dépenses, complétant l'approche DevOps avec une dimension financière.

*"Le DevOps n'est pas une destination, mais un voyage d'amélioration continue."*