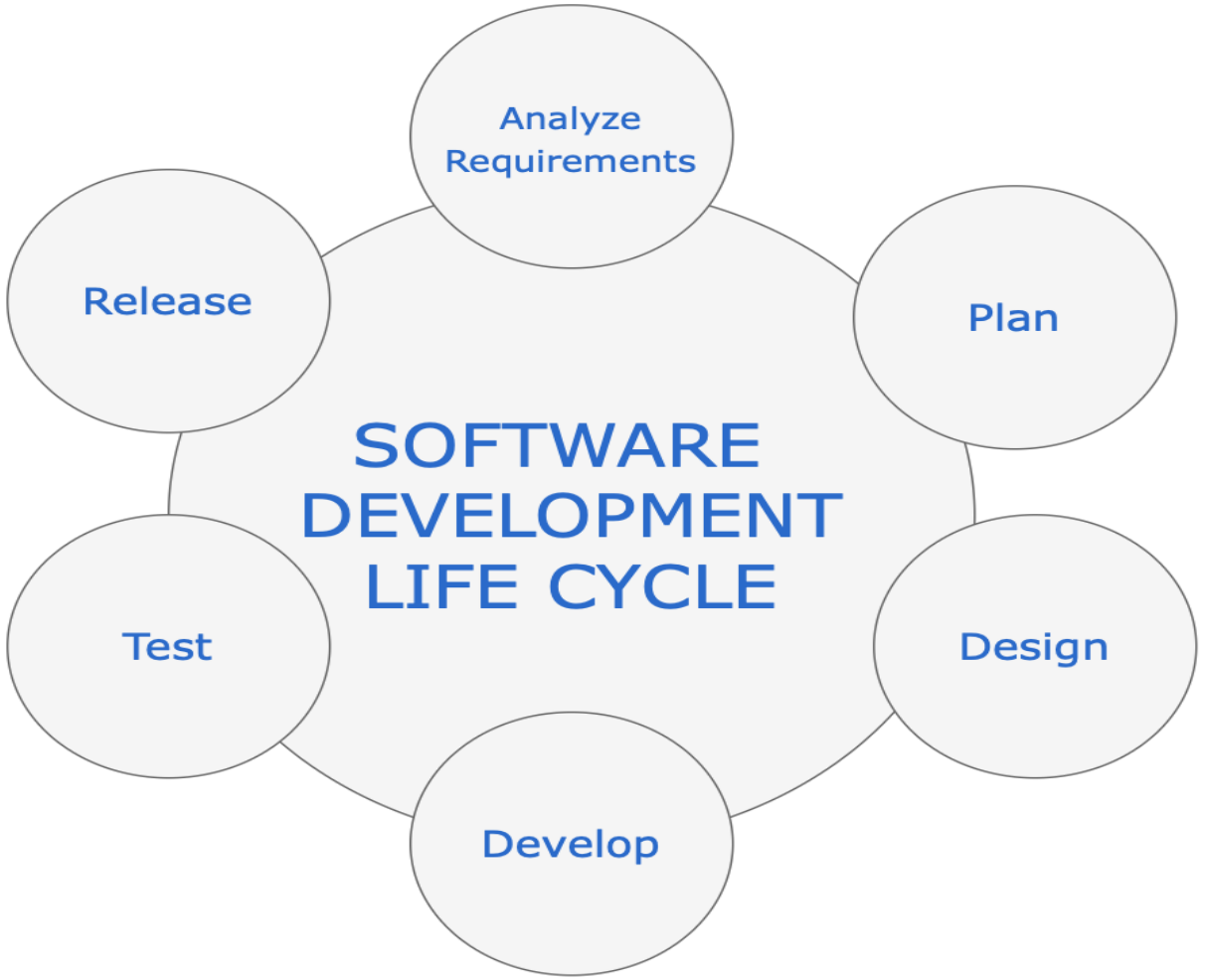
JENKINS 101

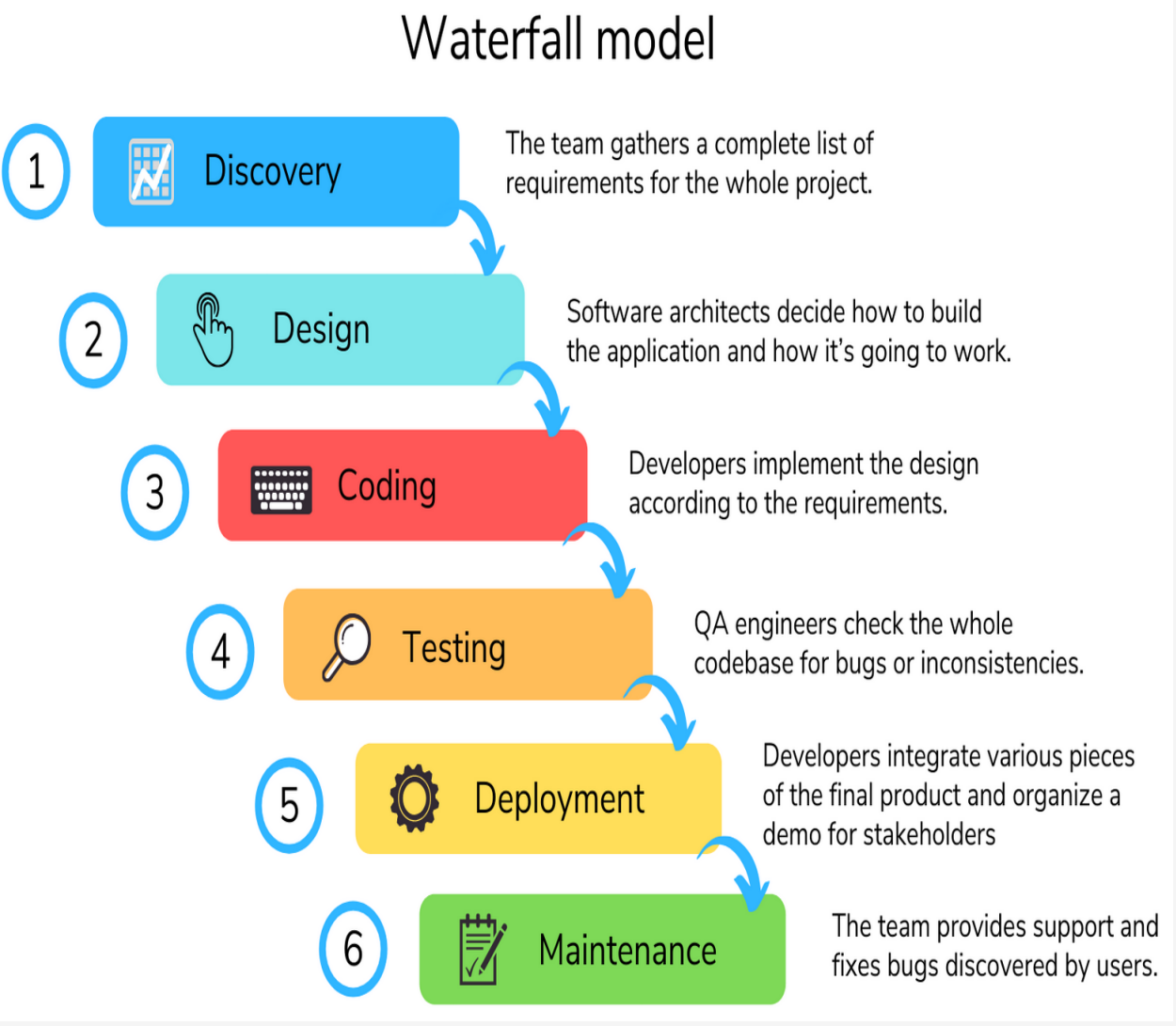
CI/CD

Software Development Life Cycle



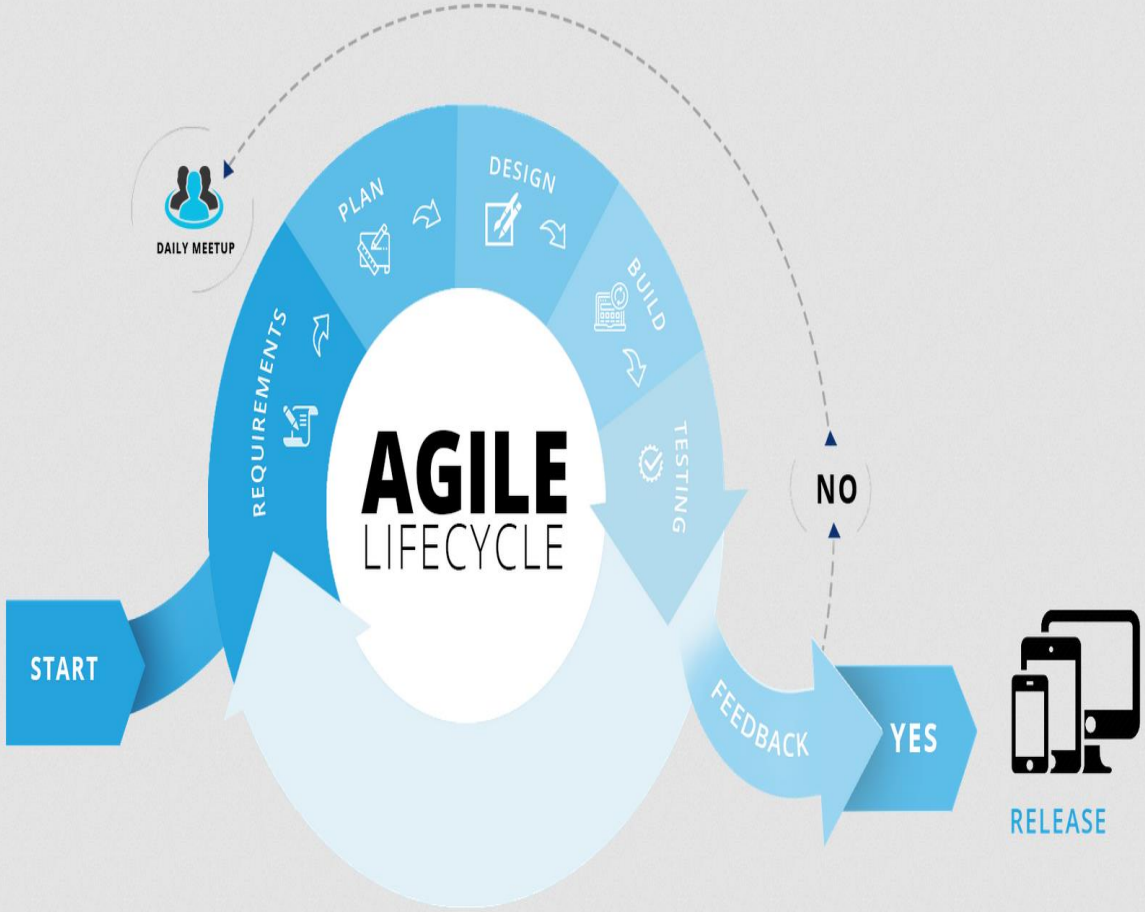
Le développement logiciel suit un flux, en commençant par l'identification de nouvelles fonctionnalités, la planification, le développement réel, la validation des modifications du code source, l'exécution de builds et de tests (unité, intégration, fonctionnel, acceptation, etc.) et le déploiement en production.

Software Development Life Cycle



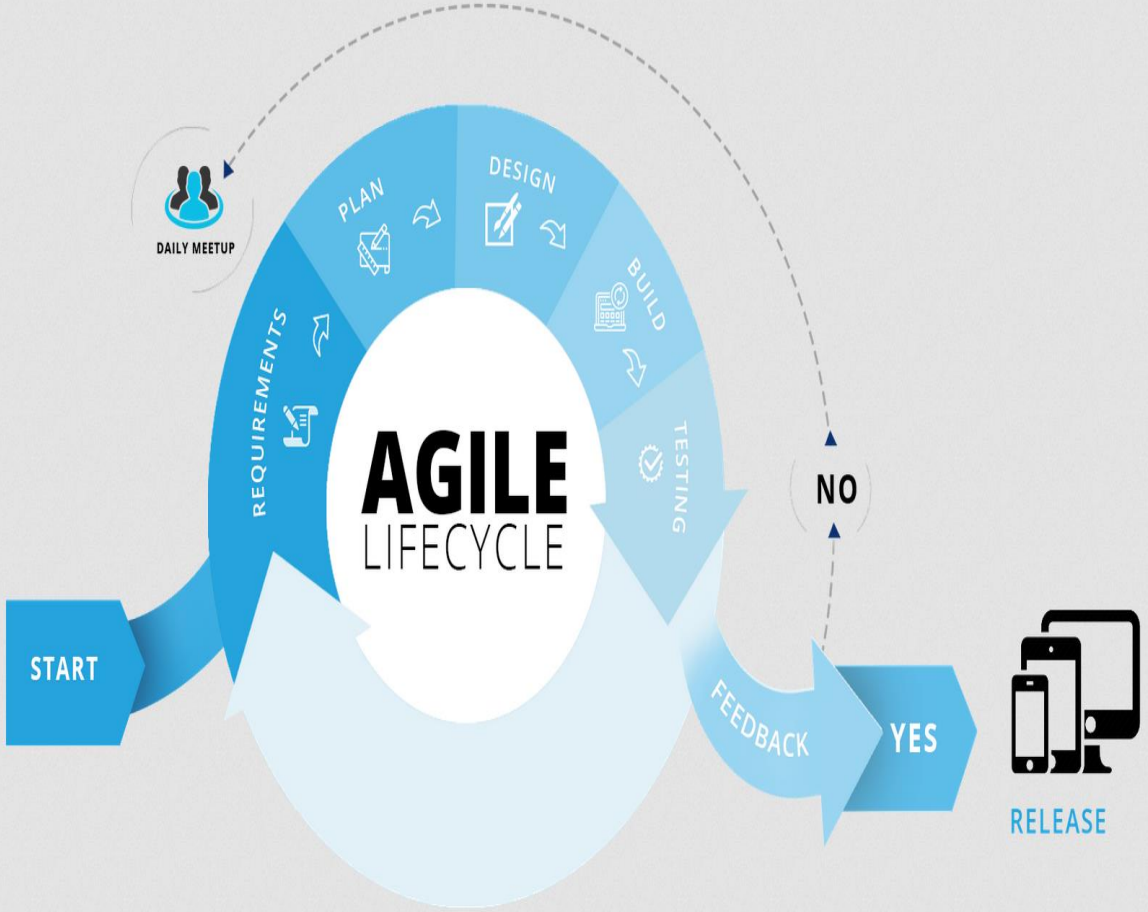
Avec une approche traditionnelle de livraison de logiciels en cascade, les développeurs pourraient travailler de manière indépendante pendant très longtemps. Ils n'auraient aucune idée du nombre de problèmes qu'ils rencontreraient pendant la phase d'intégration.

Software Development Life Cycle
Méthode Agile



Afin de résoudre ce problème, le modèle de développement logiciel Agile a été introduit. Agile a changé la façon dont les équipes de développement logiciel travaillaient. L'un des principes clés de cette méthodologie était de fournir fréquemment des logiciels fonctionnels.
L'accent a été mis sur la publication de logiciels incrémentiels, plutôt que sur les versions en cascade big bang qui ont pris des mois et des années.

Software Development Life Cycle
Méthode Agile



La livraison de logiciels signifiait souvent produire du code stable pour chaque version incrémentielle.
C'était tout un défi d'intégrer les changements de divers développeurs dans les équipes. Cela a conduit les équipes logicielles à rechercher de meilleures approches. L'intégration continue (IC) a offert une lueur d'espoir et a commencé à gagner en popularité.

Software Development Life Cycle
Méthode Agile

L'intégration continue est une pratique d'ingénierie agile.
Elle se concentre principalement sur la construction et le test automatisés pour chaque changement engagé dans le système de contrôle de version par les développeurs.

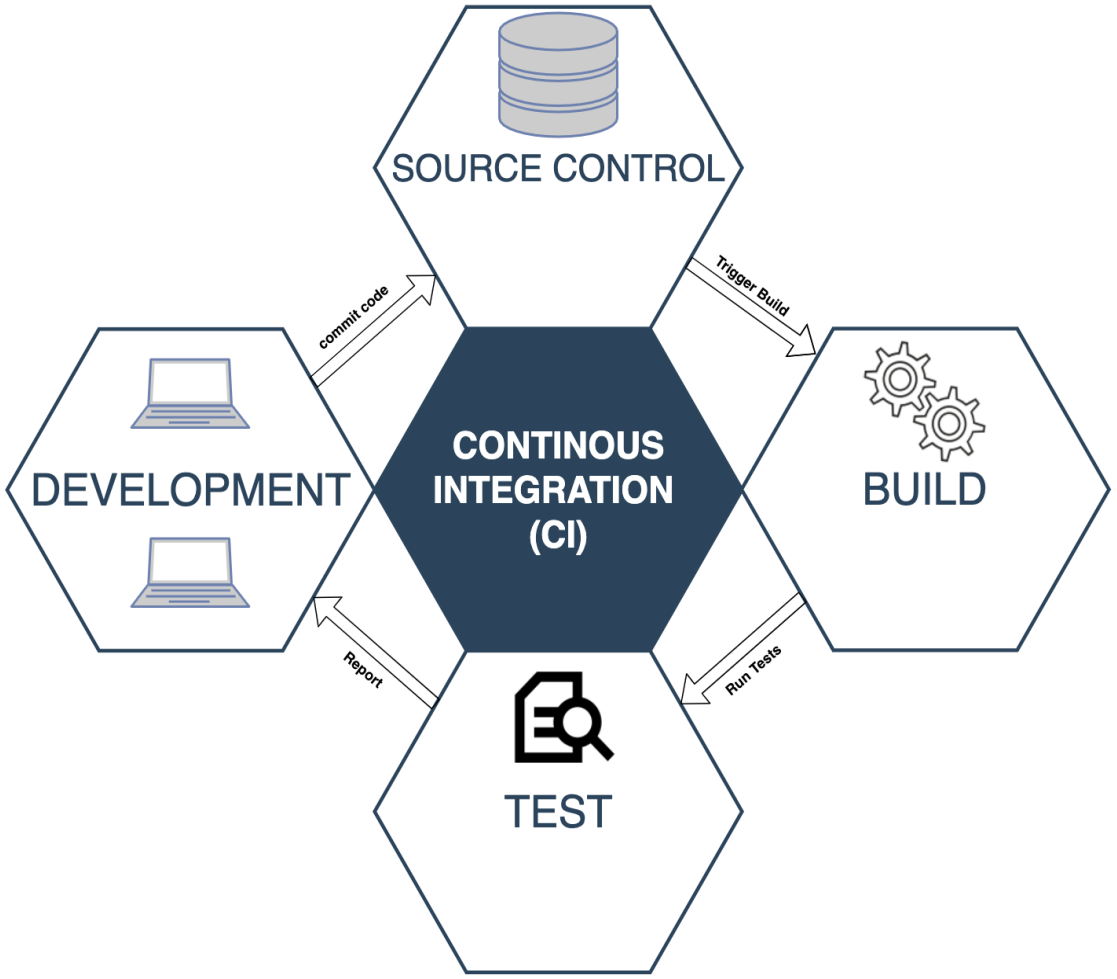
According to Martin Fowler,

"Continuous Integration (CI) is a software development practice where members of a team integrate their work frequently; usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible".

Selon Martin Fowler,

"L'intégration continue (CI) est une pratique de développement de logiciels où les membres d'une équipe intègrent fréquemment leur travail; habituellement, chaque personne intègre au moins quotidiennement, ce qui conduit à plusieurs intégrations par jour. Chaque intégration est vérifiée par un build automatisée (y compris le test) pour détecter les erreurs d'intégration le plus rapidement possible ".

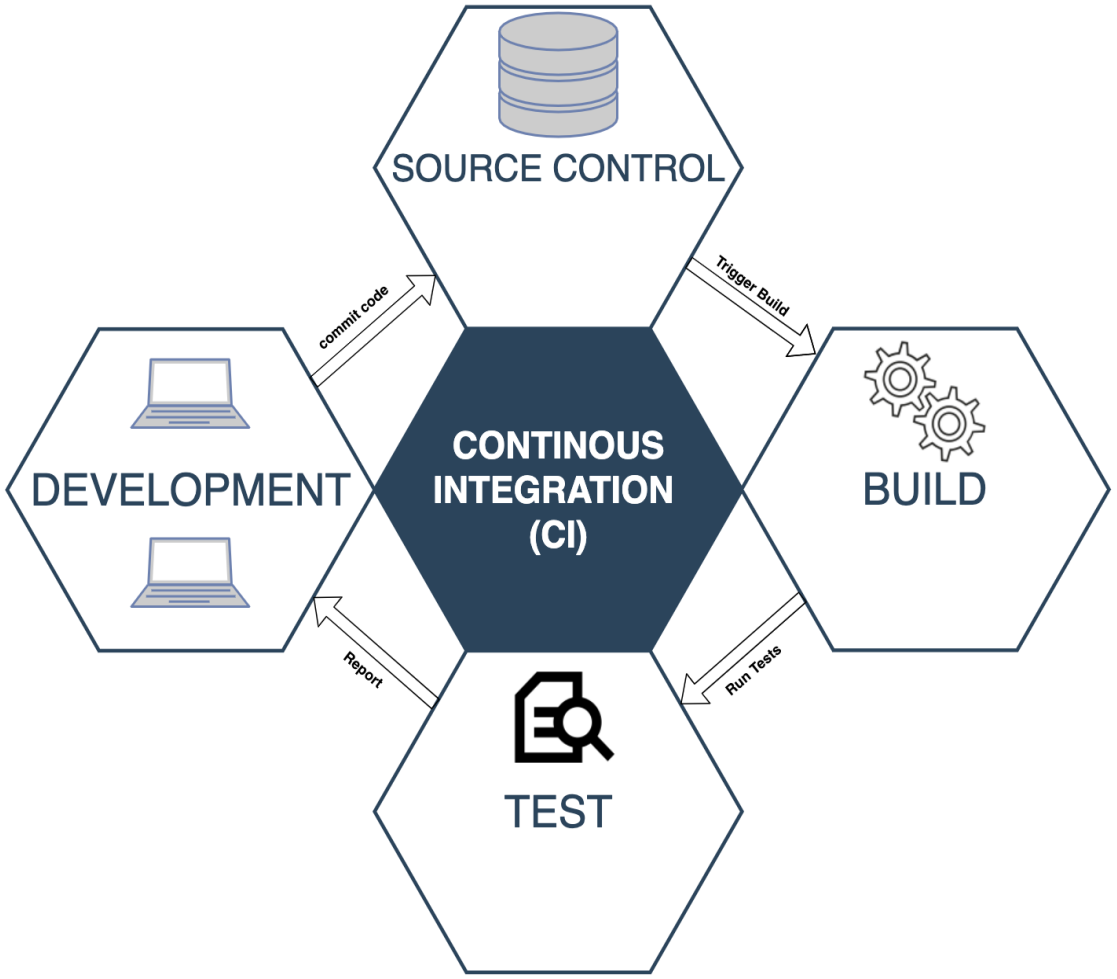
Software Development Life Cycle
Méthode Agile : Continuous Integration



Afin de mettre en œuvre l'intégration continue, vous aurez besoin de:

- Un système de contrôle de version.
  Il stocke tout le code source vérifié par les équipes et agit comme la source unique de vérité.
- Construction automatisée et test unitaire
  Il ne suffit pas que le code écrit par un développeur fonctionne sur sa machine. Chaque commit qui parvient au système de contrôle de version doit être généré et testé par un serveur d'intégration continue indépendant.

Software Development Life Cycle
Méthode Agile : Continuous Integration



- Feedback

  Les développeurs devraient obtenir des Feedback sur leurs commit. Chaque fois qu'une modification d'un développeur interrompt la génération, il peut prendre les mesures nécessaires (par exemple: notifications par e-mail).

- Accord sur les méthodes de travail

  Il est important que tous les membres de l'équipe suivent la pratique consistant à enregistrer les changements progressifs plutôt que d'attendre qu'ils soient complètement développés.

  Leur priorité devrait être de résoudre tous les problèmes de build pouvant survenir avec le code archiver.

Software Development Life Cycle
DEVOPS

L'intégration continue résout principalement la partie développement du flux de travail.

Cependant, la livraison de logiciels ne se limite pas au développement de fonctionnalités et à l'intégration des modifications. Il existe un processus de test (manuel et/ou automatisé) et de publication (déploiement réel chez le client ou en production).

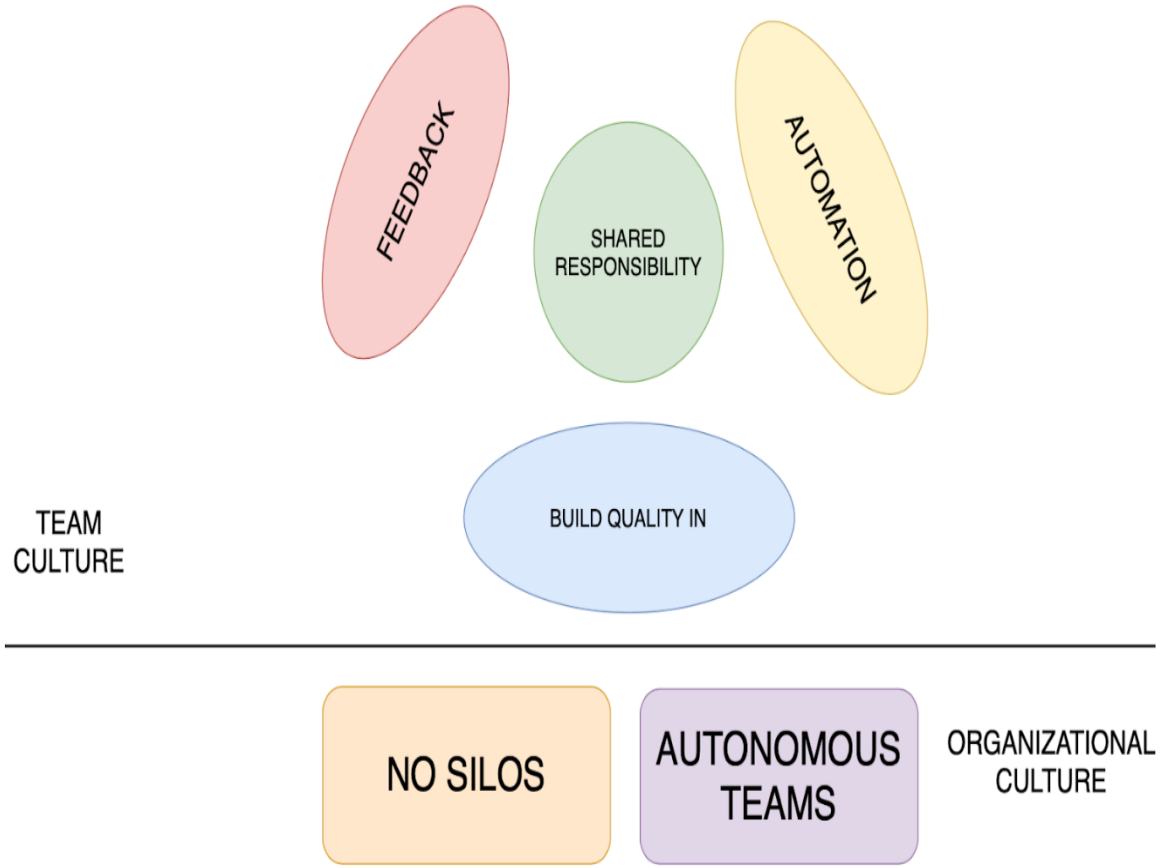
À l'ère pré-DevOps, chaque équipe était responsable de son propre travail. L'équipe de développement s'occuperait du développement des fonctionnalités. C'est là que leur travail s'est terminé. Ils le transmettaient ensuite à l'équipe d'assurance qualité (QA).

L'équipe d'assurance qualité exécuterait toutes les suites de tests étendues (manuelles et automatisées). Si les choses fonctionnaient bien, ils confieraient les fonctionnalités à l'équipe d'exploitation, qui finalement déploierait les nouvelles fonctionnalités en production et les gérerait.

Chacune de ces équipes travaillait en silos. Les processus de publication étaient pour la plupart manuels et sujets aux erreurs, entraînant des cycles de publication plus longs et douloureux.

Chaque fois que quelque chose tournait mal, un temps considérable était passé à essayer de trouver la cause première du problème, sans parler du jeu du blâme. Tout cela a entraîné une perte de temps considérable à différents niveaux de l'organisation.

Software Development Life Cycle
DEVOPS

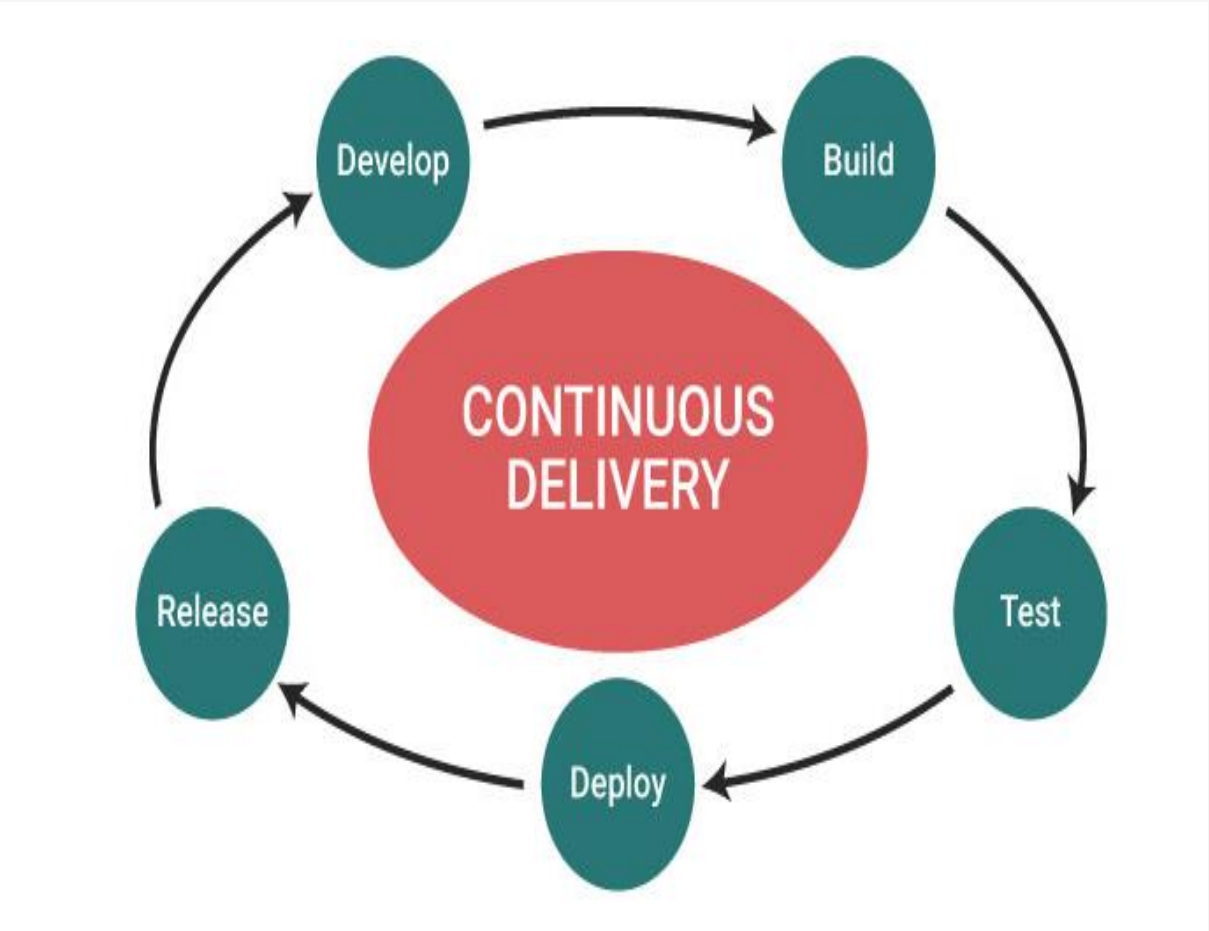


En réalité, chaque équipe devrait être également responsable de la sortie du nouveau logiciel, ce qui signifie que toutes ces équipes doivent travailler en étroite collaboration les unes avec les autres.

Le mouvement DevOps, issu du développement de logiciels Agile, insiste fortement sur la nécessité d'une collaboration entre les différentes équipes impliquées dans le processus de livraison de logiciels.

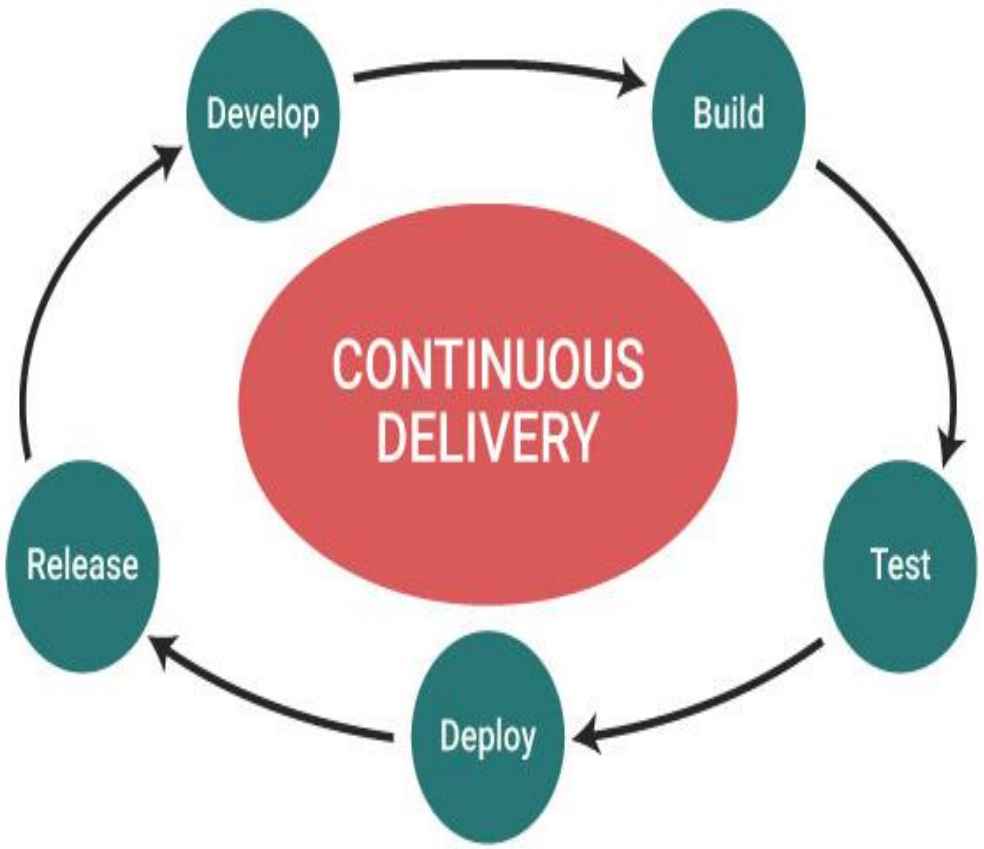
En plus d'une collaboration étroite, l'automatisation de chacune des étapes du processus de livraison du logiciel et les cycles de rétroaction constants sont également considérés comme extrêmement importants. La livraison continue fournit un cadre pour atteindre les objectifs de DevOps grâce à l'automatisation et aux boucles de Feedback continues.

Software Development Life Cycle
DEVOPS: Continuous Delivery



La livraison continue est une extension logique de l'intégration continue.
Alors que l'intégration continue vous permet d'automatiser le processus de création et de test du logiciel, la livraison continue automatise le processus complet de livraison des applications en prenant en compte toute les modifications du code (nouvelles fonctionnalités, corrections de bogues, etc.) du développement (validation du code) au déploiement (aux environnements tels que la mise en scène et la production).
Il garantit que vous êtes en mesure de publier rapidement de nouvelles modifications à vos clients de manière fiable et reproductible.

Software Development Life Cycle
DEVOPS: Continuous Delivery



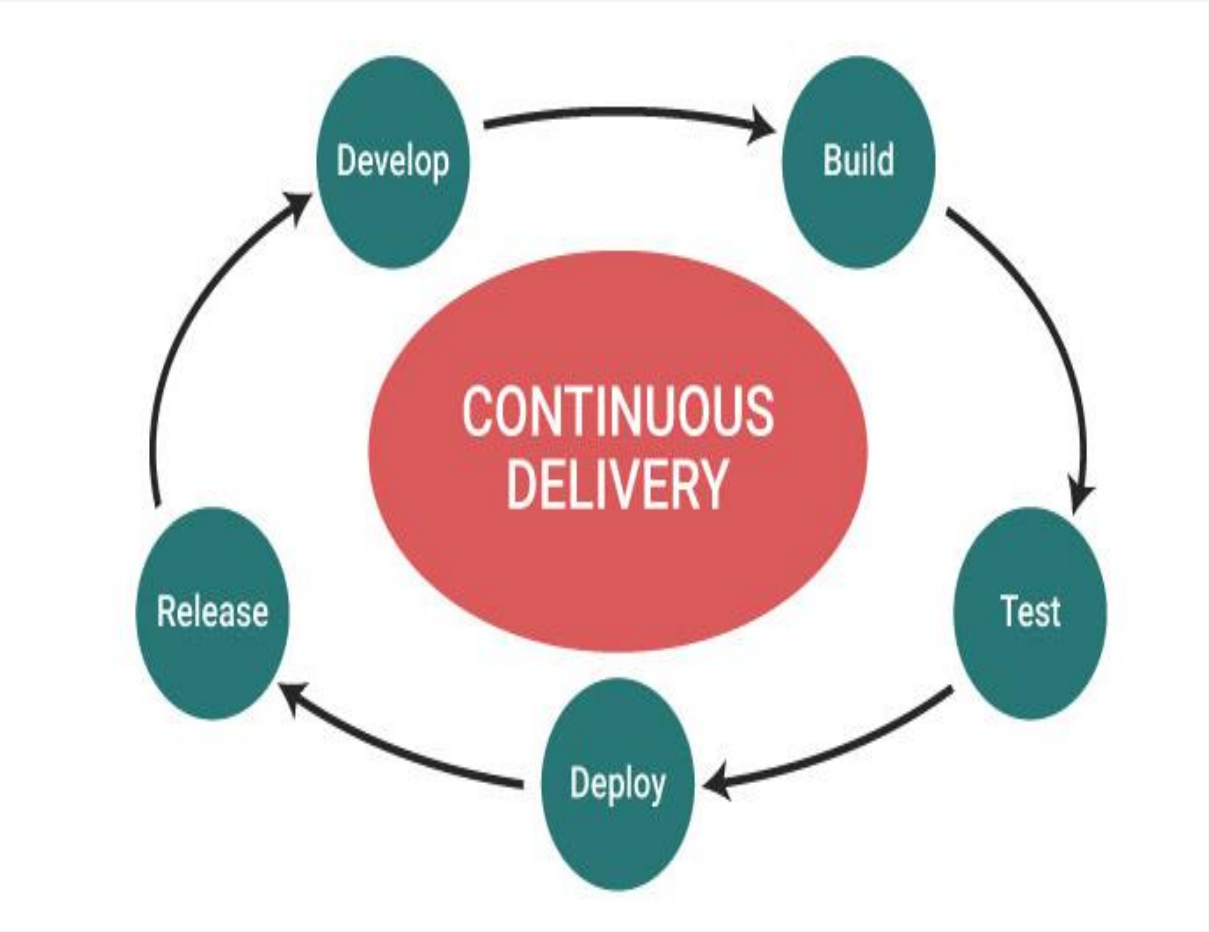
According to Martin Fowler, who coined the term Continuous Delivery,

"Continuous Delivery is a software development discipline where you build software in such a way that the software can be released to production at any time. "

Selon Martin Fowler, qui a inventé le terme livraison continue,

" La livraison continue est une discipline de développement logiciel où vous construisez des logiciels de manière à ce que ces derniers puissent être mis en production à tout moment. "

Software Development Life Cycle
DEVOPS: Continuous Delivery



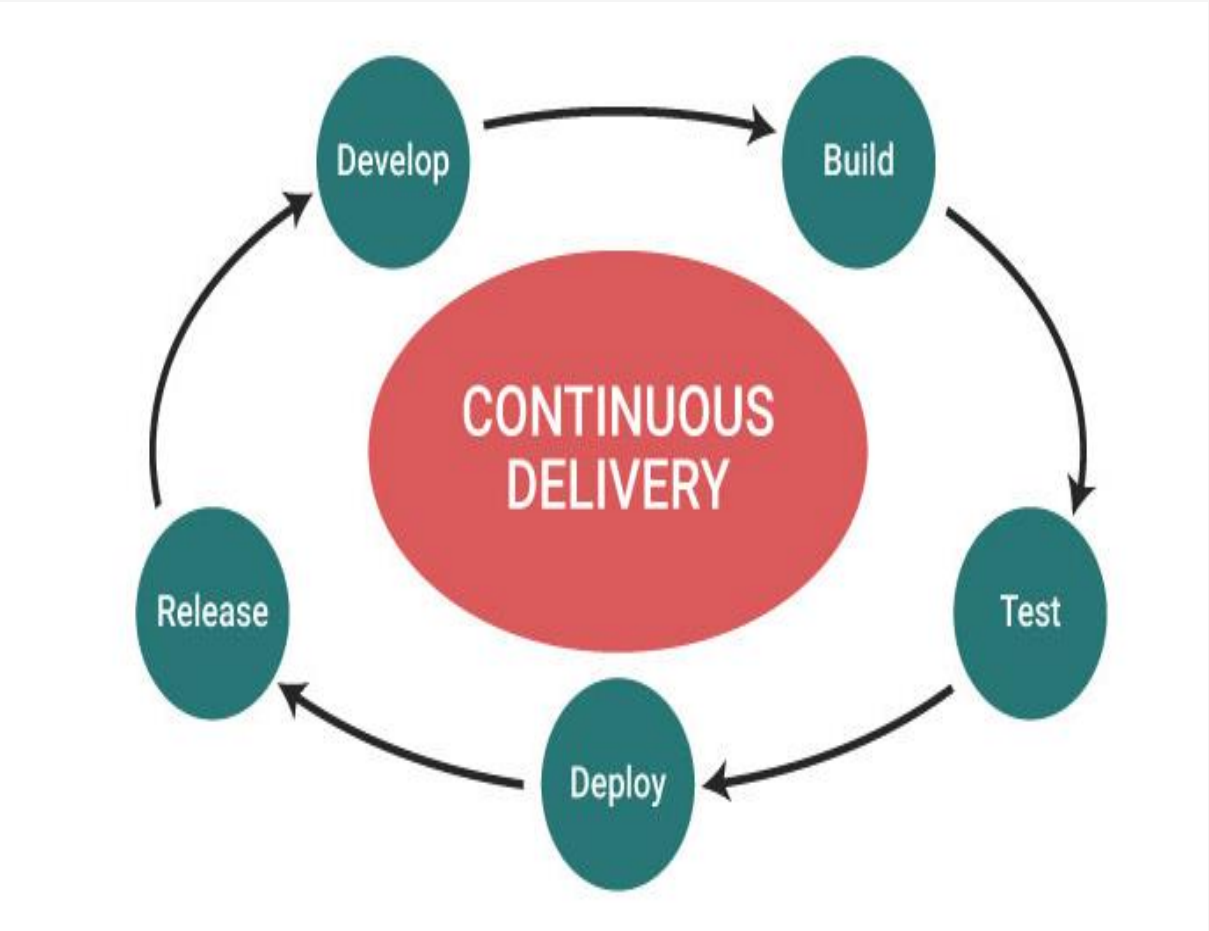
You're doing continuous delivery when:

a. Your software is deployable throughout its lifecycle.
b. Your team prioritizes keeping the software deployable over working on new features.
c. Anybody can get fast, automated feedback on the production readiness of their systems any time somebody makes a change to them.
d. You can perform push-button deployments of any version of the software to any environment on demand.

Vous effectuez une livraison continue lorsque : un.

- Votre logiciel est déployable tout au long de son cycle de vie.
- Votre équipe donne la priorité au maintien du logiciel déployable plutôt qu'au travail sur de nouvelles fonctionnalités.
- N'importe qui peut obtenir un Feedback rapide et automatisé sur l'état de préparation de la production de ses systèmes chaque fois que quelqu'un y apporte une modification.
- Vous pouvez effectuer des déploiements par bouton-poussoir de n'importe quelle version du logiciel dans n'importe quel environnement à la demande.

Software Development Life Cycle
DEVOPS: Continuous Delivery

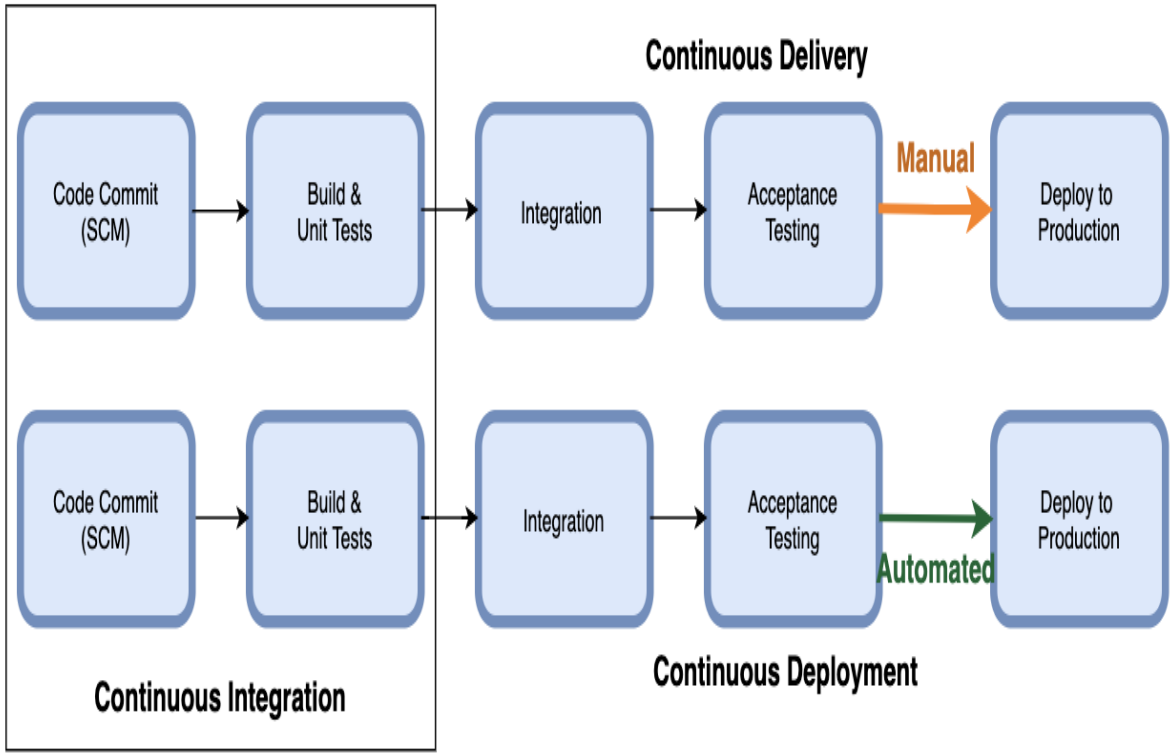


Pour obtenir une livraison continue, vous avez besoin de:

• Une relation de travail étroite et collaborative entre toutes les personnes impliquées dans la livraison (souvent appelée culture DevOps)

• Automatisation étendue de toutes les parties possibles du processus de livraison, généralement à l'aide d'un pipeline de déploiement* ».

Software Development Life Cycle
DEVOPS: Continuous Deployment

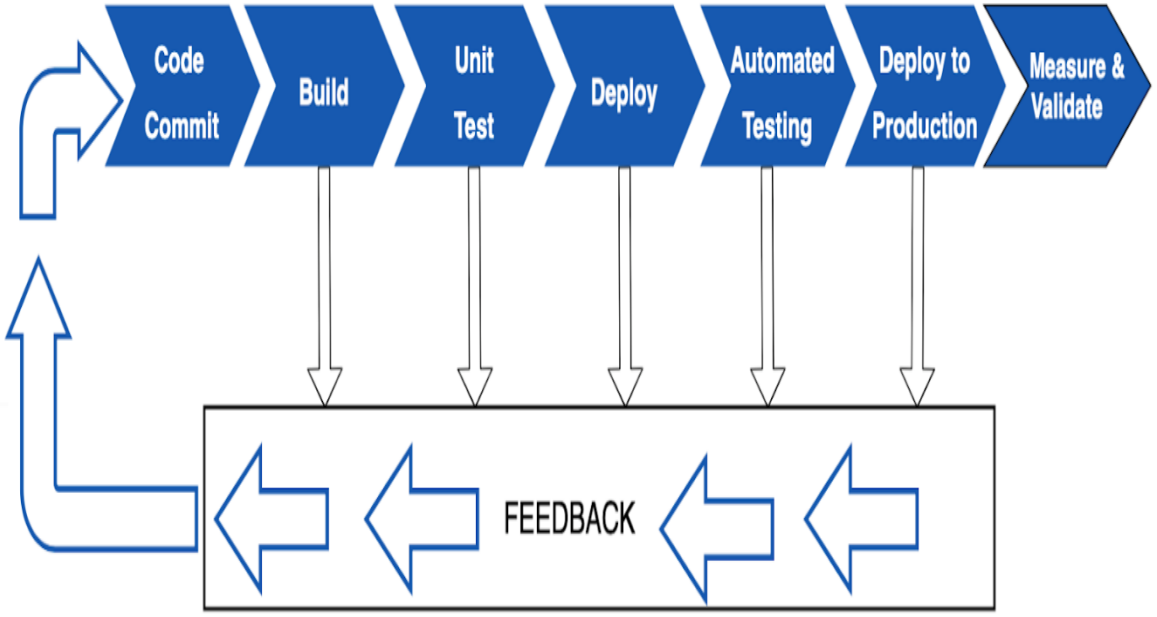


Souvent, les termes livraison continue et déploiement continu sont utilisés comme synonymes par beaucoup, mais en réalité, ces concepts ont une signification différente.

Bien que la livraison continue vous donne la possibilité de déployer fréquemment en production, cela ne signifie pas nécessairement que vous automatisez le déploiement.
Vous aurez peut-être besoin d'une approbation manuelle avant le déploiement en production, ou votre entreprise ne voudra peut-être pas déployer fréquemment. Le déploiement continu, cependant, est un moyen automatisé de déployer vos versions en production.

Vous devez effectuer une livraison continue afin de pouvoir effectuer un déploiement automatisé. Des entreprises comme Netflix, Amazon, Google et Facebook se déploient automatiquement en production plusieurs fois par jour.

CI/CD
Deployment Pipelines



Les pipelines de déploiement (ou pipelines de livraison continue) sont le noyeau de la livraison continue car ils automatisent toutes les étapes (génération, test, publication, etc.) de votre processus de livraison de logiciels.
L'utilisation de pipelines de déploiement continu présente de nombreux avantages.

Un pipeline automatisé permet à toutes les parties prenantes de surveiller la progression, élimine les frais généraux de tout le travail manuel, fournit un retour d'information rapide et, plus important encore, renforce la confiance dans la qualité du code.

Outils pour le pipeline de déploiement

Afin d'automatiser les différentes étapes de votre pipeline de déploiement, vous aurez besoin de plusieurs outils. Par exemple:

- Un système de contrôle de version tel que Git pour stocker votre code source
- Un outil d'intégration continue (CI) tel que Jenkins pour exécuter des builds automatisés
- Frameworks de test tels que xUnit, Selenium, etc., pour exécuter diverses suites de tests
- Un référentiel binaire tel qu'Artifactory pour stocker les artefacts de construction
- Outils de gestion de configuration tels que Ansible
- Un tableau de bord unique pour rendre la progression visible à tous
- Commentaires fréquents sous forme d'e-mails ou de notifications Slack.

Et ce n'est pas tout. Vous aurez également besoin d'un seul outil capable de rassembler tous ces outils pour atteindre les objectifs CI/CD qui est d'automatiser la livraison de logiciels.

Outils CI/CD

Il existe un grand nombre d'outils CI/CD open source (gratuits) et d'entreprise (payants) disponibles sur le marché.

Voici une matrice qui compare certains des outils CI/CD les plus populaires en fonction du type d'offre (payante ou gratuite), des licences (open source ou closed source), de l'hébergement (sur site ou cloud) et des extensions/plugins.

CI/CD Tools	Feature: Pricing	Feature: Licensing	Feature: Hosting	Feature: Extensions/ Plugins
Atlassian Bamboo	Payant	Closed	Sur site	Moyen
Bitbucket Pipelines	Payant	Closed	Cloud/ Sur site	Moyen
AWS CodePipeline	Payant	Closed	Cloud	Faible
Azure Pipelines	Payant	Closed	Cloud	Moyen
Circle CI	Payant	Closed	Sur site/ Cloud	Moyen
CloudBees Core	Payant	Closed	Sur site/ Cloud	Fort
GitHub Actions	Payant	Closed	Sur site/ Cloud	Moyen
GitLab CI/CD	Gratuit/Payant	OSS/Closed	Sur site/ Cloud	Faible
Google Cloud Build	Payant	Closed	Cloud	Moyen
Jenkins	Gratuit	OSS	Sur site/ Cloud	Fort
Travis CI	Payant	Closed	Sur site/ Cloud	Moyen

Outils CI/CD
GitLab CI/CD vs Jenkins

Comme vous pouvez le voir, il n'y a que deux options gratuites, GitLab CI / CD et Jenkins.

Cependant, GitLab community edition ne vous donne pas la flexibilité d'utiliser un référentiel de contrôle de version de votre choix et n'offre pas beaucoup d'intégrations via les plugins. D'autre part.

Jenkins est un outil gratuit et open source qui a résisté à l'épreuve du temps. Il existe depuis plus de 15 ans. Il dispose d'une communauté open source importante et active qui déploie constamment de nouvelles versions, des corrections de bogues, etc.
Il existe de nombreuses sources en ligne pour obtenir de l'aide.

Jenkins peut être installé sur n'importe quel système d'exploitation sur site ou dans le cloud. Sa solide intégration avec de nombreux autres outils tiers en fait un excellent choix.