

Université Sultan Moulay Slimane

Ecole Nationale des Sciences Appliquées (ENSA) de Béni-Mellal

**Filière : Transformation Digitale Industrielle (TDI)**

Niveau : 2ème année du cycle Ingénieur

Rapport de l’application dans le cadre du Module : « Génie logiciel »

**GIT/GITHUB**

Présenté le 07/01/2023

Sous l’encadrement de Mr BOUTALLINE

Réalisé par :

Dounia ALKADMIRI

Khadija AZABI

Imane BENJEBBOUR

Chakib BENMHAMED

Oussama BOUYA

Hajar MIQRI

Année Universitaire : 2022-2023

Remerciement

Nous remercions tout d’abord Dieu tout puissant de nous avoir permis de mener à terme ce projet qui présente pour nous le point de départ de notre carrière.

Il nous est agréable d’exprimer notre reconnaissance auprès de toutes les personnes, dont l’intervention au cours de cette formation académique a favorisé son aboutissement.

Il est souvent difficile de remercier les gens qui vous aident à accomplir les tâches qui vous sont données, et pourtant nous nous devons d’exprimer l’entière gratitude que nous ressentons envers eux.

Nous tenons donc à présenter un remerciement bien distingué à notre encadrant Mr BOUTALLINE pour son soutien, son aide, et ses conseils qui nous ont guidés durant l’élaboration de ce travail.

Nous voudrons aussi remercier la direction de l’ENSABM ainsi que tout le corps professoral et administratif de l’école pour les efforts qu’ils fournissent afin de nous garantir une bonne formation.

Finalement, nous espérons que notre travail vous procure une parfaite satisfaction et soit à la hauteur.

Sommaire

Introduction

Les développeurs et les opérateurs rencontrent un ensemble des problèmes qui sont parfois fatiguant et difficiles à résoudre. Mais, avec les technologies on trouve qu’il existe des moyens pour bien s’adapter et mieux gérer les taches à livrer dans une équipe.

La**rivalité “développent VS opérations”**est en train de disparaître, quoique progressivement, car les deux services sont de plus en plus conscients de faire partie de la même équipe et d’œuvrer vers le même objectif. DevOps permet aux sociétés d’être **réactif au Time to market**avec célérité et donc, de **satisfaire les besoins clients.**

L’intérêt et l’adoption de Devops se développent rapidement, car il donne une meilleure gestion et renforcer l’esprit de collaboration, les résultats de l’intégration de concept Devops sont en courageux, et les entreprises évoluent vers **une approche dynamique orientée client**pour le développement et la livraison de leurs applications.

Dans un environnement où les clients se tournent vers des transactions numériques à l’ère du mobile, **le rôle des développeurs**d’applications devient **incontournable dans l’expérience client**. En parallèle, la tendance à l’agilité a été une source d’inspiration pour DevOps dont l’un des points-clés agiles favorise les professionnels et leurs interactions plutôt que les processus et les outils.

Mais le but reste le même : réduire TTM, TTR, et sans oublier le respect du cadre de projet (délai, qualité, budget).

Parmi les outils qui sont utilisés sont les outils de système de version de contrôle, qui renforcent l’efficacité du travail et le rendement.

Alors, l’automatisation et ses aspects est un outil indispensable pour n’importe quel service en raison de satisfaction des besoins des clients.

# Définition Devops

Devops est une culture qui se base sur la collaboration et la disparition des rivalités entre les Développeurs et les Operateurs pour mieux gérer les projets d’une manière efficace.

Conceptualisé en 2009 en Belgique par l'informaticien Patrick Debois, le DevOps est un mouvement qui vise à concilier deux corps de métier : le développeur logiciel (dev) d'une part, l’administrateur en charge des opérations informatiques (ops) d'autre part. Deux profils qui poursuivent historiquement des objectifs antagoniques. Le développeur a pour mission de créer et faire évoluer (ou build) des applications à moindres coûts, et parfois au détriment de leur performance et de leur stabilité. Quant aux ops, ils se chargent de leur mise en production (ou run) en se portant garant ensuite de leur disponibilité et de leur temps de réponse. Une équation difficile à résoudre si chacun travail dans son coin.

En intégrant les devs et les ops à l'équipe de projet, le DevOps vise à casser le mur existant entre les deux mondes. L'enjeu ? Prendre en considération les contraintes de déploiement dès la phase de programmation. Avec pour objectif d'appliquer la logique des méthodes agiles à l'ensemble de l'activité informatique, le DevOps se concrétise par la mise en place de pipelines d'intégration et de livraison continues (CI/CD) courts. Ces derniers ayant pour vocation d'automatiser et de monitorer l'ensemble du cycle de vie des applications, des phases de test jusqu'à la mise en production. Mais pourquoi adapter une approche DEVOPS ?

En prônant la coopération entre les développeurs (dev) et les responsables de la production informatique (ops), le DevOps vise une expérience digitale plus efficiente pour les utilisateurs d'applications, qu'ils soient collaborateurs ou clients. En prenant en compte les contraintes de production informatique dès la phase de programmation, les logiciels sont de facto plus performants. Ils engendrent ensuite moins de bugs et de failles de sécurité.

Une fois les applications déployées, les administrateurs systèmes et équipes de développeurs continuent de collaborer. Objectif : résoudre tout problème qui pourrait survenir autour du logiciel. Des problèmes qui peuvent provenir de l'application en tant que telle ou de ses dépendances avec d'autres couches du système d'information (serveur d'applications, serveur physique, clusters...). Là encore, le travail main dans la main des deux partis prend tout son sens. Les incidents sont solutionnés plus rapidement. L'enjeu du DevOps est ainsi d'améliorer la satisfaction des utilisateurs.



Devops comme approche se base sur un ensemble des principes et piliers, on peut citer :

# La gestion des configurations

Sans automatisation, la construction et la maintenance de systèmes informatiques modernes à grande échelle peuvent être une entreprise gourmande en ressources et entraîner un risque accru en raison d’une erreur manuelle. [La gestion de la configuration et des ressources](https://www.netapp.com/devops-solutions/configuration-management) est une méthode automatisée permettant de maintenir les systèmes informatiques et les logiciels dans un état connu et cohérent.

Il existe plusieurs composants dans un système de gestion de la configuration. Les systèmes gérés peuvent inclure des serveurs, du stockage, des réseaux et des logiciels. Ce sont les cibles du système de gestion de la configuration. L’objectif est de maintenir ces systèmes dans des états connus et déterminés. Un autre aspect d’un système de gestion de la configuration est la description de l’état souhaité pour les systèmes. Le troisième aspect majeur d’un système de gestion de la configuration est le logiciel d’automatisation, qui est responsable de s’assurer que les systèmes et logiciels cibles sont maintenus dans l’état souhaité. Mais pourquoi la gestion des configurations est importante ?

La gestion de la configuration est importante car elle permet de faire évoluer l’infrastructure et les systèmes logiciels sans avoir à faire évoluer le personnel administratif pour gérer ces systèmes. Cela peut permettre d’évoluer là où il n’était pas possible de le faire auparavant.

On peut dire que le principal avantage de la gestion de la configuration est la cohérence des systèmes et des logiciels. Avec la gestion de la configuration, vous ne devinez plus ou n’espérez plus qu’une configuration est à jour. Il est correct parce que le système de gestion de la configuration garantit qu’il est correct.

Lorsqu’elle est combinée à l’automatisation, la gestion de la configuration peut améliorer [l’efficacité](https://www.netapp.com/hybrid-cloud/it-automation) car les processus de configuration manuels sont remplacés par des processus automatisés. Cela permet également de gérer plus de cibles avec les mêmes ressources, voire moins.

Les outils de Management de configuration :

* [Chapeau rouge Ansible](http://community.netapp.com/t5/Technology/Ansible-by-Red-Hat-Now-with-Native-NetApp-Portfolio-Integration/ba-p/128351)
* [Chef](https://supermarket.chef.io/cookbooks/netapp)
* [Marionnette](https://www.netapp.com/pdf.html?item=/media/12636-tr-4477.pdf)
* La CI/CD (intégration et livraison continues).

CI/CD est une méthode permettant de fournir fréquemment des [applications](https://www.redhat.com/en/topics/cloud-native-apps?percmp=7013a0000034e7YAAQ) à clients en introduisant [l’automatisation](https://www.redhat.com/en/topics/automation?percmp=7013a0000034e7YAAQ) dans le étapes du [développement de l’application](https://www.redhat.com/en/topics/cloud-native-apps/why-choose-red-hat-cloud-native?percmp=7013a0000034e7YAAQ). Les principaux concepts attribués à CI/CD sont continus Intégration, [livraison continue](https://www.redhat.com/en/topics/devops/what-is-continuous-delivery?percmp=7013a0000034e7YAAQ) et déploiement continu. CI/CD est une solution à la Problèmes [d’intégration](https://www.redhat.com/en/topics/integration?percmp=7013a0000034e7YAAQ) de nouveaux code peut causer pour les équipes de développement et d’exploitation (AKA « l’enfer de l’intégration »).

Plus précisément, CI/CD introduit l’automatisation continue et Surveillance continue tout au long [du cycle de vie des applications, des](https://www.redhat.com/en/topics/devops/what-is-application-lifecycle-management-alm?percmp=7013a0000034e7YAAQ) phases d’intégration et de test à la livraison et au [déploiement](https://www.redhat.com/en/topics/automation/what-is-deployment-automation?percmp=7013a0000034e7YAAQ). Prises ensemble, ces pratiques liées sont souvent appelé « [pipeline CI/CD](https://www.redhat.com/en/topics/devops/what-cicd-pipeline?percmp=7013a0000034e7YAAQ) » et soutenu par des équipes de développement et d’exploitation travailler ensemble de manière agile avec une approche [DevOps](https://www.redhat.com/en/topics/devops?percmp=7013a0000034e7YAAQ) ou [d’ingénierie de fiabilité de site (SRE).](https://www.redhat.com/en/topics/devops/what-is-sre?percmp=7013a0000034e7YAAQ)

**Intégration continue**

Dans le développement d’applications modernes, l’objectif est d’avoir plusieurs développeurs travaillant simultanément sur différents fonctionnalités de la même application. Toutefois, si une organisation est Configurer pour fusionner tout le code source de branchement sur Un jour (connu sous le nom de « jour de fusion »), le travail résultant peut Soyez fastidieux, manuel et chronophage. C’est parce que Lorsqu’un développeur travaillant de manière isolée apporte une modification à une application, il y a une chance qu’elle entre en conflit avec différentes modifications étant effectuées simultanément par d’autres Développeurs. Ce problème peut être aggravé si chaque développeur a personnalisé son propre [environnement de développement intégré (IDE)](https://www.redhat.com/en/topics/middleware/what-is-ide?percmp=7013a0000034e7YAAQ) local, plutôt que l’équipe se mette d’accord sur un IDE basé sur le [cloud](https://www.redhat.com/en/topics/cloud?percmp=7013a0000034e7YAAQ).

L’intégration continue (CI) aide les développeurs à fusionner leurs Le code revient à une branche partagée, ou « tronc », plus fréquemment, parfois même tous les jours. Une fois qu’un développeur Les modifications apportées à une application sont fusionnées, ces modifications sont validé en construisant automatiquement l’application et exécution de différents niveaux de tests automatisés, généralement tests unitaires et d’intégration, pour assurer les changements n’ont pas cassé l’application. Cela signifie tout tester des classes et des fonctions aux différents modules qui Composez l’ensemble de l’application. Si les tests automatisés découvrent un conflit entre le nouveau code et le code existant, CI le rend Il est plus facile de corriger ces bugs rapidement et souvent.

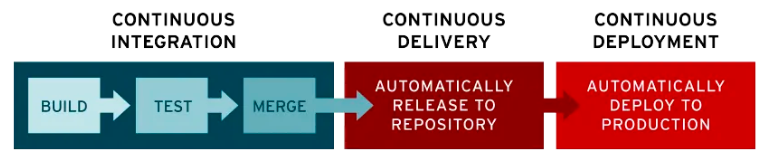
Suite à l’automatisation des constructions et des unités et tests d’intégration en CI, livraison continue automatise la publication de ce code validé dans un référentiel. Ainsi afin d’avoir une livraison continue efficace , il est important que l’IC soit déjà intégré dans votre pipeline de développement. L’objectif de La livraison consiste à avoir une base de code toujours prête pour Déploiement dans un environnement de production.

Dans la livraison continue, chaque étape, de la fusion de Modifications apportées au code pour la livraison de produits prêts pour la production builds : implique l’automatisation des tests et la publication de code automatisation. À la fin de ce processus, les opérations L’équipe est en mesure de déployer une application en production rapidement et facilement.

**Déploiement continu**

L’étape finale d’un pipeline CI/CD mature est continue déploiement. Dans le prolongement de la livraison continue, qui automatise la publication d’une version prête pour la production vers un référentiel de code, le déploiement continu automatise Lancement d’une application en production. Parce qu’il n’y a pas de Portail manuel au stade de la canalisation avant La production et le déploiement continu reposent fortement sur Automatisation des tests bien conçue.

En pratique, le déploiement continu signifie qu’un La modification apportée par un développeur à une application cloud pourrait être mise en ligne dans les minutes qui suivent sa rédaction (en supposant qu’il soit adopté) tests automatisés). Il est ainsi beaucoup plus facile de recevoir et intégrer en permanence les commentaires des utilisateurs. Ensemble, toutes ces pratiques CI/CD connectées rendre le déploiement d’une application moins risqué, ce qui signifie Il est plus facile de publier des modifications apportées aux applications en petits morceaux, plutôt que tout à la fois. Il y a aussi beaucoup d’avance Investissement, cependant, puisque les tests automatisés devront être rédigé pour tenir compte d’une variété de tests et étapes de publication dans le pipeline CI/CD.



**Quels sont les outils CI/CD courants ?**

Les outils CI/CD peuvent aider une équipe à automatiser son développement, déploiement et tests. Certains outils gèrent spécifiquement côté intégration (CI), certains [gèrent](https://www.redhat.com/en/topics/management?percmp=7013a0000034e7YAAQ) le développement et le déploiement (CD), tandis que d’autres se spécialisent dans tests continus ou fonctions connexes.

L’un des outils open source les plus connus pour CI/CD est le serveur d’automatisation Jenkins. Jenkins est conçu pour gérer n’importe quoi, d’un simple serveur CI à un concentrateur de CD complet.

[Déploiement de Jenkins sur Red Hat OpenShift](https://cloud.redhat.com/blog/deploying-jenkins-on-openshift-part-1?percmp=7013a0000034e7YAAQ&extIdCarryOver=true&intcmp=7013a000002wBnmAAE&sc_cid=7013a000002DgC5AAK%27%5d%5d)

Tekton Pipelines est un framework CI/CD pour Kubernetes plates-formes qui fournissent un CI/CD cloud natif standard expérience avec les conteneurs.

[Déploiement de Jenkins sur Red Hat OpenShift](https://cloud.redhat.com/learn/topics/ci-cd?percmp=7013a0000034e7YAAQ&extIdCarryOver=true&intcmp=7013a000002wBnmAAE&sc_cid=7013a000002DgC5AAK%27%5d%5d)

Au-delà de Jenkins et Tekton Pipelines, d’autres open source Les outils CI/CD que vous voudrez peut-être étudier comprennent :

* [Spinnaker](https://spinnaker.io/), un CD Plate-forme conçue pour les environnements [multicloud](https://www.redhat.com/en/topics/cloud-computing/what-is-multicloud?percmp=7013a0000034e7YAAQ).
* [GoCD,](https://www.gocd.org/) un CI/CD serveur en mettant l’accent sur la modélisation et visualisation.
* [Hall](https://concourse-ci.org/), « Un faiseur de choses continu open-source. »
* [Tournevis](https://screwdriver.cd/), un plate-forme de construction conçue pour le CD.

Les équipes peuvent également envisager des outils de CI/CD gérés, qui sont disponibles auprès d’une variété de fournisseurs. Les principaux [fournisseurs de cloud public](https://www.redhat.com/en/topics/cloud-computing/what-are-cloud-providers?percmp=7013a0000034e7YAAQ) proposent tous des solutions CI/CD, ainsi que [GitLab](https://about.gitlab.com/), [CircleCI](https://circleci.com/), [Travis CI,](https://www.travis-ci.com/) [Atlassian Bamboo](https://www.atlassian.com/software/bamboo) et bien d’autres.

De plus, tout outil fondamental pour DevOps est susceptibles de faire partie d’un processus CI/DC. Outils [d’automatisation de la configuration](https://www.redhat.com/en/topics/automation/what-is-configuration-management?percmp=7013a0000034e7YAAQ) (tels que [Ansible](https://www.redhat.com/en/technologies/management/ansible?percmp=7013a0000034e7YAAQ), [Chef](https://www.chef.io/) et [Puppet](https://puppet.com/)), conteneur Les runtimes (tels que [Docker](https://www.redhat.com/en/topics/containers/what-is-docker?percmp=7013a0000034e7YAAQ), [rkt](https://cloud.redhat.com/learn/topics/rkt?percmp=7013a0000034e7YAAQ) et [cri-o](https://cri-o.io/)) et l’orchestration de [conteneurs](https://www.redhat.com/en/topics/containers/what-is-container-orchestration?percmp=7013a0000034e7YAAQ) ([Kubernetes](https://www.redhat.com/en/topics/containers/what-is-kubernetes?percmp=7013a0000034e7YAAQ)) ne sont pas strictement des outils CI/CD, mais ils apparaîtront dans de nombreux flux de travail CI/CD.

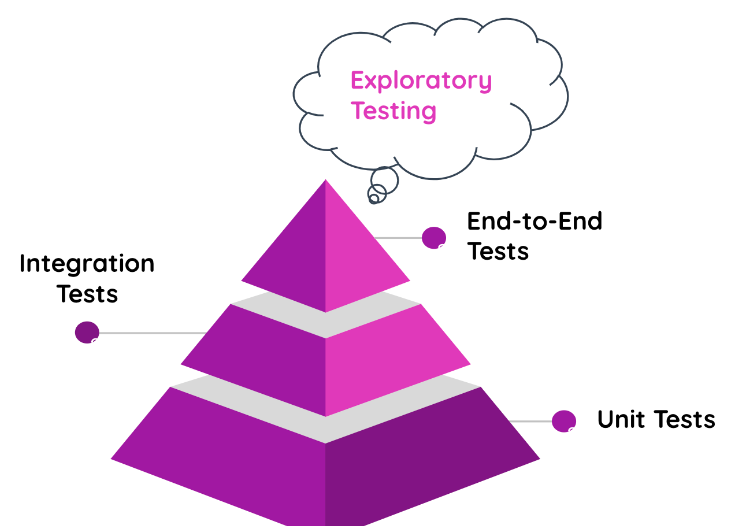
* L'automatisation des tests.

L’automatisation des tests est la pratique consistant à examiner et à valider automatiquement un produit logiciel, tel qu’une application Web, pour s’assurer qu’il répond aux normes de qualité prédéfinies pour le style de code, les fonctionnalités (logique métier) et l’expérience utilisateur.

Les pratiques de test impliquent généralement les étapes suivantes :

* **Test unitaire** : valide des unités de code individuelles, telles qu’une fonction, afin qu’elle fonctionne comme prévu
* **Test d’intégration** : garantit que plusieurs morceaux de code peuvent fonctionner ensemble sans conséquences imprévues
* Test de bout **en bout** : valide que l’application répond aux attentes de l’utilisateur
* **Tests exploratoires** : adopte une approche non structurée pour examiner de nombreux domaines d’une application du point de vue de l’utilisateur, afin de découvrir des problèmes fonctionnels ou visuels

Les différents types de tests sont souvent visualisés sous forme de pyramide. Au fur et à mesure que vous montez dans la pyramide, le nombre de tests dans chaque type diminue et le coût de création et d’exécution des tests augmente.



Une bonne pratique DevOps consiste à exécuter des tests automatisés aussi tôt et aussi souvent que possible dans le pipeline CI/CD. Cela inclut l’exécution de tests automatisés de l’interface utilisateur en production pour surveiller de manière proactive les problèmes d’expérience utilisateur. Étant donné que les applications d’aujourd’hui reposent sur de nombreux services avec plusieurs pièces mobiles, l’exécution d’une [surveillance synthétique des transactions](https://www.mabl.com/blog/using-synthetic-transaction-monitoring-for-an-outside-in-view-of-app-health) en exécutant des tests en production peut détecter les problèmes avec les services tiers avant vos utilisateurs.

L’adoption des tests automatisés permet de débloquer les avantages DevOps suivants :

* **Vitesse sans sacrifier la qualité** : gagnez en vélocité de produit élevée qui rend les développeurs heureux et leur permet d’offrir plus de valeur aux utilisateurs, plus rapidement
* Amélioration de la collaboration au sein de l’équipe : le partage de la responsabilité de la qualité permet une meilleure collaboration entre les membres **de l’équipe**
* **Fiabilité** : améliorez la fiabilité des versions en augmentant la couverture grâce à l’automatisation des tests. Les problèmes de production devraient être rares plutôt que la norme
* **Échelle** : produire des résultats de qualité constante avec un risque réduit en répartissant le développement entre plusieurs petites équipes qui fonctionnent de manière autonome
* **Sécurité** : agissez rapidement sans compromettre la sécurité et la conformité en tirant parti des politiques de conformité automatisées, des contrôles précis et des techniques de gestion de la configuration
* Augmentation de la satisfaction des clients : une fiabilité accrue et des réponses rapides aux commentaires des utilisateurs augmentent **la satisfaction des** utilisateurs et conduisent à plus de références de produits

VERSION DE SYSTEME DE CONTROLE :

La **gestion de versions** (en [anglais](https://fr.wikipedia.org/wiki/Anglais) : *version control* ou *revision control*) consiste à gérer l'ensemble des versions d'un ou plusieurs fichiers (généralement [en texte](https://fr.wikipedia.org/wiki/Fichier_texte)). Essentiellement utilisée dans le domaine de la [création de logiciels](https://fr.wikipedia.org/wiki/Programmation_informatique), elle concerne surtout la gestion des [codes source](https://fr.wikipedia.org/wiki/Codes_source). Autrement, système de contrôle de révision ou de code source, un système de contrôle de version (ou VCS, pour Version Control System) est un utilitaire logiciel qui suit et gère les changements apportés à un système de fichiers. Un VCS offre également des utilitaires collaboratifs pour partager et intégrer les changements apportés au système de fichiers avec d'autres VCS. Lorsqu'il est utilisé au niveau du système de fichiers, un VCS suit les actions d'ajout, de suppression et de modification appliquées aux fichiers et répertoires. Un dépôt est un terme propre aux VCS qui décrit le suivi d'un système de fichiers par un VCS. Dans le cadre de fichiers de code source individuels, un VCS suit les ajouts, suppressions et modifications des lignes de texte dans ce fichier. Parmi les VCS populaires dans le secteur du développement figurent Git, Mercurial, SVN et Perforce.

Pourquoi ai-je besoin d'un logiciel de contrôle de version ?

Un VCS est un outil inestimable qui procure de nombreux avantages dans le workflow collaboratif d'une équipe de développement. Tout projet logiciel impliquant plusieurs développeurs pour gérer les fichiers de code source devrait absolument avoir recours à un VCS. Par ailleurs, un VCS procurera également de nombreux avantages pour les projets à mainteneur unique. Aucune raison valide ne justifie de renoncer à utiliser un VCS dans un projet de développement moderne.

Les avantages de VSC sont :

Résolution des conflits

Durant le cycle de vie d'un projet logiciel en équipe, il y a fort à parier que plusieurs membres de l'équipe devront apporter des changements au même fichier de code source en même temps. Un VCS suit ces changements et s'avère utile en cas de conflit entre plusieurs développeurs. Ces opérations de résolution de conflits génèrent une trace d'audit qui offre des informations sur l'historique d'un projet

Restauration et annulation des changements apportés au code source

Une fois qu'un VCS a commencé à suivre un système de fichiers de code source, il conserve un historique des changements et de l'état du code source tout au long du projet. Il est ainsi possible d'« annuler » un projet de code source ou de le restaurer à son dernier état connu. Si un bug est identifié dans une application active, il est possible de restaurer rapidement le code à une version connue pour être stable.

Sauvegarde du code source hors site

Lorsque vous utilisez un VCS de façon collaborative, il est nécessaire de créer une instance distante de celui-ci pour partager les changements entre les développeurs. Cette instance distante du VCS peut être hébergée hors site sur un service tiers de confiance, comme Bitbucket. Elle devient alors une sauvegarde hors site sécurisée. Dans le pire des cas, comme le vol d'un ordinateur portable, l'instance distante du VCS conserve une copie du code source.

Avantages des outils de contrôle de version

L'intégration d'un VCS à un projet de développement logiciel procure divers avantages en termes d'organisation et de gestion. Par défaut, un VCS seul offre les avantages techniques abordés plus haut (résolution des conflits au sein d'une équipe et aides à la collaboration). Un service de VCS hébergé encapsule un VCS par défaut et procure des fonctionnalités améliorées. Il est incroyablement puissant et offre une vision transparente du processus de développement logiciel, un effort de création traditionnellement opaque. Voici quelques avantages généraux d'un VCS hébergé.

Intégrations tierces étendues

Un VCS peut être amélioré davantage lorsqu'il est intégré à une solution tierce hébergée et sécurisée, comme Bitbucket. Un VCS hébergé offre ses propres extensions pour d'autres apps et services grâce à des intégrations comme l'Atlassian Marketplace. Par exemple, un VCS hébergé sur Bitbucket sera en mesure de partager les données du VCS avec le logiciel de suivi des tâches Jira et avec la plateforme de documentation et de base de connaissances Confluence. Ce ne sont là que deux exemples parmi toutes les intégrations que propose un VCS hébergé.

Communication entre équipes

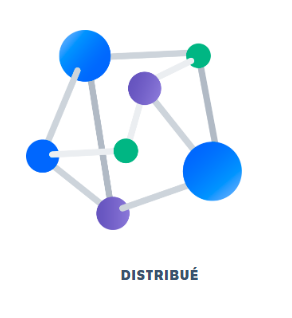
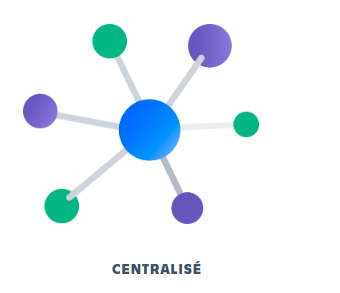
Un VCS hébergé offre des outils de messagerie et de communication (forum) étendus, qui peuvent s'avérer inestimables pour la collaboration en équipe. Ces outils de messagerie permettent des discussions à thème claires sur le développement de code source, et ce, ligne par ligne. Ils ne se limitent pas à la discussion sur le code source et peuvent s'avérer très utiles pour favoriser la discussion transverse sur des objectifs métier généraux, qui sont liés au développement de code source.

Informations, métriques et responsabilité

Une organisation spécialisée dans l'ingénierie peut utiliser un VCS hébergé pour suivre et mesurer des indicateurs clés de performance (KPI). Un VCS hébergé vous fournira des informations sur la vitesse et l'efficacité d'un projet de développement logiciel. Il procure des outils pour capturer des métriques, comme les nouvelles fonctionnalités déployées, la vélocité de la création de code, le temps écoulé durant le développement, les bugs créés et corrigés, et bien plus.

Automatisation du pipeline de CI/CD

Il est possible de concevoir des workflows de livraison continue à l'aide d'intégrations externes avec un VCS hébergé. Un développeur initie une transaction pour intégrer ses modifications du code source à la base de code plus large. Durant ce processus, le VCS hébergé peut être configuré avec de nombreuses extensions de sorte à simplifier la revue de code et le processus de livraison. Les builds automatisés, les tests ou encore les déploiements peuvent être orchestrés dans un fichier YAML (config-as-code) simple.

Les outils VCS sont basés sur deux principaux types d'architecture distante : centralisé et distribué.

Lorsqu'il est question des avantages et inconvénients de chaque architecture, la fonctionnalité de sauvegarde hors site est le principal sujet de discussion. Un VCS centralisé possède un point de défaillance unique, l'instance centrale distante du VCS. La perte de celle-ci peut entraîner une chute de la productivité et une perte de données. Elle devra être remplacée par une autre copie du code source. En cas d'indisponibilité temporaire, les développeurs ne pourront plus faire de push, merger ou restaurer le code. Une architecture distribuée évite ces pièges, puisqu'elle conserve une copie complète du code source sur chaque instance du VCS. Si l'un des scénarios de défaillance centralisée mentionnés plus haut venait à se produire dans un modèle distribué, une nouvelle instance du VCS pourrait être mise en place pour piloter le développement, ce qui empêcherait une chute marquée de la productivité.

**Gestion de versions centralisée**[[modifier](https://fr.wikipedia.org/w/index.php?title=Gestion_de_versions&veaction=edit&section=7) | [modifier le code](https://fr.wikipedia.org/w/index.php?title=Gestion_de_versions&action=edit&section=7)]

Avec les logiciels de gestion de versions centralisée, comme [CVS](https://fr.wikipedia.org/wiki/Concurrent_versions_system) et [Subversion](https://fr.wikipedia.org/wiki/Subversion_(logiciel)) (SVN), il n'existe qu'un seul dépôt des versions qui fait référence.

Cela simplifie la gestion des versions mais est contraignant pour certains usages comme le travail sans connexion au réseau, ou tout simplement lorsque l'on travaille sur des branches expérimentales ou contestées.

**Gestion de versions décentralisée**[[modifier](https://fr.wikipedia.org/w/index.php?title=Gestion_de_versions&veaction=edit&section=8) | [modifier le code](https://fr.wikipedia.org/w/index.php?title=Gestion_de_versions&action=edit&section=8)]

La gestion de versions décentralisée consiste à voir l'outil de gestion de versions comme un outil permettant à chacun de travailler à son rythme, de façon désynchronisée des autres, puis d'offrir un moyen à ces développeurs de s'échanger leur travaux respectifs. De fait, il existe plusieurs dépôts pour un même logiciel. Ce système est très utilisé par les [logiciels libres](https://fr.wikipedia.org/wiki/Logiciels_libres).

Par exemple, [GNU Arch](https://fr.wikipedia.org/wiki/GNU_Arch), [Git](https://fr.wikipedia.org/wiki/Git) et [Mercurial](https://fr.wikipedia.org/wiki/Mercurial) sont des logiciels de gestion de versions décentralisée.

Avantages de la gestion décentralisée :

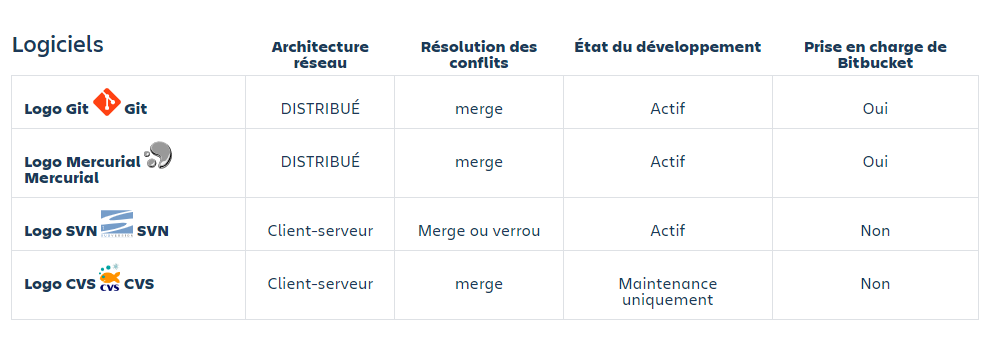
* permet de ne pas être dépendant d'une seule machine comme point de défaillance ;
* permet aux contributeurs de travailler sans être connectés au gestionnaire de version ;
* permet la participation à un projet sans nécessiter les permissions par un responsable du projet (les droits de commit/soumission peuvent donc être donnés après avoir démontré son travail et non pas avant) ;
* la plupart des opérations sont plus rapides car réalisées en local (sans accès réseau) ;
* permet le travail privé pour réaliser des brouillons sans devoir publier ses modifications et gêner les autres contributeurs ;
* permet toutefois de garder un dépôt de référence contenant les versions livrées d'un projet.

Inconvénients :

* cloner un dépôt est plus long que récupérer une version pour une gestion de version décentralisée car tout l'historique est copié (ce qui est toutefois un avantage par la suite) ;
* il n'y a pas de système de lock (ce qui peut poser des problèmes pour des données binaires qui ne se fusionnent pas).

L'auteur de développement logiciel [Joel Spolsky](https://fr.wikipedia.org/wiki/Joel_Spolsky) décrit la gestion de version décentralisée comme « probablement la plus grande avancée dans les technologies de développement logiciel dans les 10 [dernières] années. »[3](https://fr.wikipedia.org/wiki/Gestion_de_versions#cite_note-3).

Une comparaison des outils :



# GIT:

Git est de loin le système de contrôle de version le plus largement utilisé aujourd'hui. Git est un projet open source avancé, qui est activement maintenu. À l'origine, il a été développé en 2005 par Linus Torvalds, le créateur bien connu du noyau du système d'exploitation Linux. De plus en plus de projets logiciels reposent sur Git pour le contrôle de version, y compris des projets commerciaux et en open source. Les développeurs qui travaillent avec Git sont bien représentés dans le pool de talents disponible, et la solution fonctionne bien sur une vaste gamme de systèmes d'exploitation et d'environnements de développement intégrés (IDE).

Par sa structure décentralisée, Git illustre parfaitement ce qu'est un système de contrôle de version décentralisé (DVCS). Plutôt que de consacrer un seul emplacement pour l'historique complet des versions du logiciel comme c'était souvent le cas dans les systèmes de contrôle de version ayant fait leur temps, comme CVS et Subversion (également connu sous le nom de SVN), dans Git, chaque copie de travail du code est également un dépôt qui contient l'historique complet de tous les changements.

En plus d'être décentralisé, Git a été conçu pour répondre à trois objectifs : performances, sécurité et flexibilit Les performances brutes de Git sont très élevées par rapport à de nombreuses alternatives. Commiter de nouveaux changements, créer des branches, faire des merges et comparer les anciennes versions… Toutes ces actions sont optimisées pour les performances. Les algorithmes implémentés dans Git tirent parti de connaissances approfondies sur les attributs courants des arborescences de fichiers de code source réel, la manière dont ils sont modifiés au fil du temps et les schémas d'accès.

Git est la solution idéale pour la plupart des équipes de développement actuelles. Si chaque équipe est différente et doit faire sa propre analyse, voici les principales raisons pour lesquelles le contrôle de version avec Git est l'option à privilégier :

Git est un outil de qualité

Git offre les fonctionnalités, les performances, le niveau de sécurité et la flexibilité dont la plupart des équipes et des développeurs ont besoin. Ces avantages de Git sont repris en détail ci-dessus. Lorsqu'elles les comparent à d'autres solutions, beaucoup d'équipes préfèrent Git.

Git s'est imposé comme la norme de facto

Parmi les outils du genre, Git est le plus adopté. Git séduit pour plusieurs raisons. Chez Atlassian, le code source de nos projets est essentiellement géré dans Git.

Un grand nombre de développeurs ont déjà travaillé avec Git, et il est fort probable qu'une proportion importante de jeunes diplômés ne connaissent que cet outil. Si certaines entreprises ont beaucoup de choses à apprendre au moment de passer d'un autre système de contrôle de version à Git, bon nombre de leurs développeurs actuels et à venir n'ont pas besoin de formation.

Outre le large pool de talents, la prédominance de Git implique aussi que de nombreux logiciels et services tiers sont déjà intégrés à Git y compris les IDE et nos propres outils comme le client de bureau DVCS, [Sourcetree](http://www.sourcetreeapp.com/fr/), le logiciel de suivi des tickets et des projets, [Jira](https://www.atlassian.com/fr/software/jira), ainsi que le service d'hébergement de code, [Bitbucket](https://bitbucket.org/product/fr/).

Si vous êtes un développeur inexpérimenté et que vous souhaitez acquérir des compétences utiles dans les outils de développement, en particulier le contrôle de version, Git doit figurer sur votre liste.

Contrairement à certains logiciels de contrôle de version, Git n'est pas dupe des noms des fichiers au moment de déterminer le type de stockage et l'historique de version de l'arborescence des fichiers. Au lieu de cela, Git se concentre sur le contenu du fichier lui-même. Après tout, les fichiers de code source sont fréquemment renommés, partagés et réorganisés. Le format d'objet des fichiers du dépôt Git associe le codage différentiel (stockage de différences de contenu) et la compression ; en clair, il stocke explicitement les contenus du répertoire et les objets de métadonnées de version.

La décentralisation a également de grands avantages sur le plan des performances.

Par exemple, imaginons qu'Alice, une développeuse, apporte des changements au code source en ajoutant une fonctionnalité pour la livraison 2.0 à venir, puis qu'elle crée des commits de ces changements en y associant des messages descriptifs. Ensuite, elle travaille sur une deuxième fonctionnalité et réalise à nouveau des commits de ces changements. Bien sûr, ces changements sont stockés comme des tâches isolées dans l'historique de version. Par la suite, Alice bascule sur la branche de la version 1.3 du même logiciel pour corriger un bug qui n'affecte que cette version plus ancienne. Le but est de permettre à l'équipe d'Alice de livrer un correctif de bug, la version 1.3.1, avant que la version 2.0 ne soit prête. Alice peut ensuite retourner sur la branche 2.0 pour continuer à travailler sur de nouvelles fonctionnalités pour cette version ; le tout sans accéder au réseau, ce qui lui permet de travailler rapidement et efficacement. Elle pourrait même effectuer ces tâches à bord d'un avion. Une fois qu'elle est prête à envoyer tous les changements commités individuellement vers le dépôt distant, Alice peut les « pusher » en une fois.é.

L'intégrité du code source géré a été la priorité absolue lors de la conception de Git. Le contenu des fichiers, les liens entre les fichiers et les répertoires, les versions, les tags, les commits : tous ces éléments du dépôt Git sont sécurisés à l'aide d'un algorithme de hachage sécurisé cryptographiquement, appelé SHA1. Celui-ci protège le code et l'historique des changements contre toute modification accidentelle ou malveillante, tout en assurant une traçabilité complète de l'historique.

Avec Git, vous êtes sûr de disposer d'un historique authentique de votre code source.

D'autres systèmes de contrôle de version ne prévoient pas de protection contre des modifications secrètes à une date ultérieure. Toute entreprise qui dépend du développement logiciel peut ainsi faire face à d'importantes failles de sécurité.

Lors de la conception de Git, l'un des principaux objectifs a été la flexibilité. Git est flexible à plusieurs niveaux : que ce soit pour la prise en charge de divers workflows de développement non linéaires, pour son efficacité dans l'élaboration de projets de différente envergure ou pour sa compatibilité avec de nombreux systèmes et protocoles existants.

Git a été conçu pour prendre en charge le branching et le tagging en priorité (contrairement à SVN), et les opérations qui concernent les branches et les tags (comme les merges et les reverts) sont également stockées dans l'historique des changements. Peu de systèmes de contrôle de version offrent un tel niveau de suivi.

Et comme tous les outils, Git possèdent des avantages et inconvénients, on peut les résumer sous la forme suivante :

Avantages du contrôle de version Git

* Architecture réseau à tolérance de panne distribuée
* Optimisée pour les commits et les check-outs rapides
* Mises à jour non destructrices grâce à un journal immuable

Inconvénients du contrôle de version Git

* Faible prise en charge pour les fichiers binaires volumineux
* Les dépôts très imposants, à l'historique long, peuvent ralentir les interactions
* Courbe d'apprentissage et commandes non intuitives

**Commandes GIT de base**

**Git config**

* L’une des commandes git les plus utilisées est **git config.** On l’utilise pour configurer les préférences de l’utilisateur : son mail, l’algorithme utilisé pour diff, le nom d’utilisateur et le format de fichier etc. Par exemple, la commande suivante peut être utilisée pour définir le mail d’un utilisateur:

git config --global user.email sam@google.com

**Git init**

* Cette commande est utilisée pour créer un nouveau dépôt [**GIT**](https://www.hostinger.fr/tutoriels/comment-utiliser-une-branche-git) :

git init

**Git add**

* La **commande git add** peut être utilisée pour ajouter des fichiers à l’index. Par exemple, la commande suivante ajoutera un fichier nommé temp.txt dans le répertoire local de l’index:

git add temp.txt

**Clone git**

* La **commande git clone** est utilisée pour la vérification des dépôts. Si le dépôt se trouve sur un serveur distant, utilisez:

git clone alex@93.188.160.58:/chemin/vers/dépôt

* Inversement, si une copie de travail d’un dépôt local doit être créée, utilisez:

git clone /chemin/vers/dépôt

**Git commit**

* La **commande git commit** permet de **valider les modifications apportées** au HEAD. Notez que tout commit ne se fera pas dans le dépôt distant.

git commit –m “Description du commit”

**Git status**

* La **commande git status** affiche la liste des fichiers modifiés ainsi que les fichiers qui doivent encore être ajoutés ou validés. Usage:

git status

**Git push**

* **Git push** est une autre commandes [**GIT**](https://www.hostinger.fr/tutoriels/meilleurs-clients-git-gui/) de base. Un simple push envoie les modifications locales apportées à la branche principale associée :

git push origin master

**Git checkout**

* La **commande git checkout** peut être utilisée pour créer des branches ou pour basculer entre elles. Par exemple nous allons créer une branche:

command git checkout -b <nom-branche>

* Pour passer simplement d’une branche à une autre, utilisez:

git checkout <nom-branche>

**Git remote**

* Cette commande**remote** permet à un utilisateur de se connecter à un dépôt distant. La commande suivante répertorie les dépôts distants actuellement configurés:

git remote –v

* Cette commande permet à l’utilisateur de connecter le dépôt local à un serveur distant:

git remote add origin <93.188.160.58>

**Branche git**

* La **commande git branch** peut être utilisée pour répertorier, créer ou supprimer des branches. Pour répertorier toutes les branches présentes dans le dépôt, utilisez:

git branch

* Pour supprimer une branche:

git branch –d <nom-branche>

**Git pull**

* Pour fusionner toutes les modifications présentes sur le dépôt distant dans le répertoire de travail local, la commande pull est utilisée. Usage:

git pull

**Git merge**

* La **commande git merge** est utilisée pour fusionner une branche dans la branche active. Usage:

git merge <nom-branche>

**Git diff**

* La **commande git diff** permet de lister les conflits. Pour visualiser les conflits d’un fichier, utilisez

git diff --base <nom-fichier>

* La commande suivante est utilisée pour afficher les conflits entre les branches à fusionner avant de les fusionner:

git diff <branche-source> <branche-cible>

* Pour simplement énumérer tous les conflits actuels, utilisez:

git diff

**Git tag**

* Le marquage est utilisé pour marquer des commits spécifiques avec des poignées simples. Un exemple peut être:

git tag 1.1.0 <insert-commitID-here>

**Git log**

* L’ **exécution de** cette commande génère le log d’une branche. Un exemple de sortie :
* commit 15f4b6c44b3c8344caasdac9e4be13246e21sadw
* Author: Alex Hunter <alexh@gmail.com>

Date: Mon Oct 1 12:56:29 2016 -0600

**Git reset**

* Pour réinitialiser l’index et le répertoire de travail à l’état du dernier commit, la **commande git reset** est utilisée :

git reset --hard HEAD

**Git rm**

* **Git rm** peut être utilisé pour supprimer des fichiers de l’index et du répertoire de travail. Usage:

git rm nomfichier.txt

**Git stash**

* L’une des moins connues, **git stash** aide à enregistrer les changements qui ne doivent pas être commit immédiatement. C’est un commit temporaire. Usage:

git stash

**Git show**

* Pour afficher des informations sur tout fichier git, utilisez la **commande git show** . Par exemple:

git show

**Git fetch**

* **Git fetch** permet à un utilisateur d’extraire tous les fichiers du dépôt distant qui ne sont pas actuellement dans le répertoire de travail local. Exemple d’utilisation:

git fetch origin

**Git ls-tree**

* Pour afficher un fichier arborescent avec le nom et le mode de chaque élément, et la valeur SHA-1 du blob, utilisez la **commande git ls-tree** . Par exemple:

git ls-tree HEAD

**Git cat-file**

* À l’aide de la valeur SHA-1, affichez le type d’un fichier à l’aide de la **commande git cat-file** . Par exemple:

git cat-file –p d670460b4b4aece5915caf5c68d12f560a9fe3e4

**Git grep**

* **Git grep** permet à un **utilisateur de rechercher** dans les arbres de contenu des expressions et / ou des mots. Par exemple, pour *rechercher www.hostinger.com* dans tous les fichiers, utilisez:

git grep "www.hostinger.com"

**Gitk**

* **Gitk** est l’interface graphique du dépôt local. Vous pouvez l’appeler en exécutant:

gitk

**Git instaweb**

* Avec la **commande git instaweb** , un serveur Web peut être exécuté par interface avec le dépôt local. Qui redirige directement vers un serveur web. Par exemple:

git instaweb –httpd=webrick

**Git gc**

* Pour optimiser le dépôt en supprimant les fichiers inutiles et les optimiser, utilisez:

git gc

**git archive**

* La **commande git archive** permet à un utilisateur de créer un fichier zip ou tar contenant les composants d’un arbre du dépôt. Par exemple:

git archive --format=tar master

**Git prune**

* Via la **commande git prune** , les fichiers qui n’ont pas de pointeurs entrants seront supprimés. Usage:

git prune

GITLAB :

GitLab est un outil qui peut être utilisé par une seule personne comme par une équipe, voire une organisation ou une grande entreprise, pour **gérer des projets de développement**.

Concurrent direct du connu et reconnu **GitHub**, GitLab lui est souvent préféré ; on verra pourquoi par la suite.

Parmi les fonctionnalités qu’il propose et sur lesquelles on reviendra plus loin dans l’article, en voici une liste :

* Système de **suivi de bugs** (ou issues) ;
* Une **gestion d’équipe** (membres, rôles) ;
* Un **tableau de bord interactif** ;
* De **l’intégration et livraison continue** ;
* Un **hébergement de pages web statiques** (HTML/CSS/JS) ;
* Un **Wiki**.

Historique

GitLab a été créé par deux développeurs Ukrainiens, Valery Sizov et Dmitriy Zaporozhets, et **officiellement lancé en 2014**. En 2015, l’entreprise rachète Gitorious, une autre société hébergeant des repositories Git, raflant au passage plus de 820 000 utilisateurs.

Dans les années qui suivirent, GitLab fit plusieurs levées de fonds, racheta plusieurs entreprises et étendit son nombre de clients, affirmant sa position de logiciel libre de forge Git.

Qui utilise GitLab ?

Le logiciel, qui n’a plus a faire ses preuves, est utilisé aujourd’hui par beaucoup de développeurs, entreprises et grands groupes.

Comme on le verra par la suite, le fait que GitLab rassemble au même endroit un ensemble de fonctionnalités intéressantes, attire beaucoup d’organisations ayant des **stacks techniques poussées** et des **besoins de déploiement** spécifiques.

Parmi ses plus prestigieux clients, on retrouve par exemple **IBM**, **Oracle** ou **SpaceX**.

GitLab vs GitHub

Comme leurs noms le laisse entendre, GitLab et GitHub utilisent tous les deux la forge Git. Mais **il existe évidemment des différences entre les deux logiciels**.

Déjà, il faut savoir que **GitHub a été racheté par Microsoft** en 2018. Cela a fait fuir certains développeurs, qui ne voulaient pas que leur code soit hébergé par une grande entreprise comme Microsoft. Mais ce rachat a au moins pour mérite de garantir une évolution (favorable) et un maintien du logiciel. Rappelons que Microsoft s’investit de plus en plus dans le développement ouvert et propose de plus en plus d’outils gratuits, parmi lesquels on peut citer Visual Studio Code.

**Passons maintenant à la comparaison.**

En terme du nombre d’utilisateurs (et donc de repositories publics), GitHub l’emporte haut la main. Ce dernier est aussi reconnu pour être plus fiable que son concurrent, au niveau de la stabilité.

En revanche, GitLab possède plus de fonctionnalités. Par exemple, contrairement à son concurrent, il possède un service de **continuous integration** et **continuous delivery** **intégré** ; avec GitHub, il faudra utiliser des services externes. Aussi, dans sa version gratuite, GitLab offre plus de fonctionnalités, on peut citer par exemple **l’installation sur un serveur privé**.

L’utilisation de GitLab

Comme on l’a vu précédemment, GitLab propose tout un [panel de fonctionnalités](https://about.gitlab.com/stages-devops-lifecycle/) pour assister le développement de projets informatiques. Il en propose même plus que GitHub.

Mais alors quelles sont ces fonctionnalités ? Faisons un tour des principales.

Les fonctionnalités basiques

Évidemment, **GitLab comporte ce que j’appellerai les fonctionnalités basiques qu’une forge Git doit proposer**.

Par là, j’inclus la création, modification et suppression de différents projets (ou repositories). **La gestion des fonctionnalités de Git**, telles que les commit, push, revert, merge requests, etc.

Mais GitLab **met aussi à disposition un Wiki**, pour pouvoir écrire de la documentation.

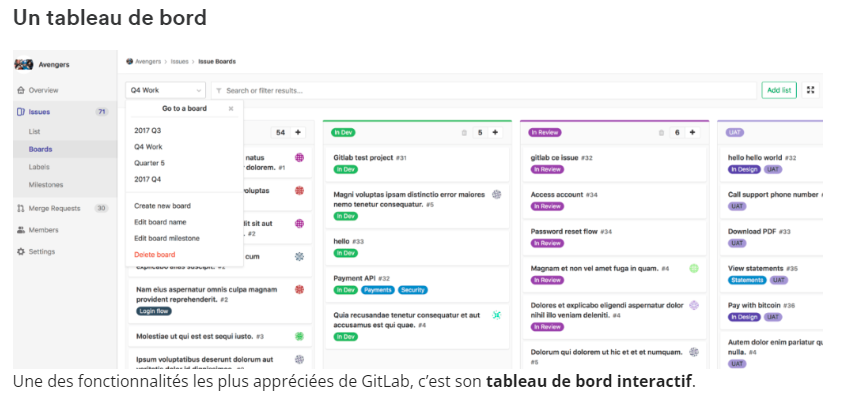
une gestion d’équipe (rôles)

On l’a déjà dit, si GitLab permet à un développeur solo d’héberger son projet Git – qu’il travaille pour des clients ou pour ses propres projets – c’est avec une utilisation en équipe que le logiciel révèle toute sa puissance. Et qui dit équipe dit membre, avec éventuellement des [droits, ou rôles](https://docs.gitlab.com/ee/user/permissions.html), différents.

GItLab **permet très facilement d’ajouter ou retirer des utilisateurs** d’un projet, et de leur confier des rôles différents.

GitLab met à disposition cinq rôles qu’on peut affecter un à utilisateur pour qu’il participe au projet : owner, maintener, developer, reporter et guest. Ils ont évidemment tous des droits différents.

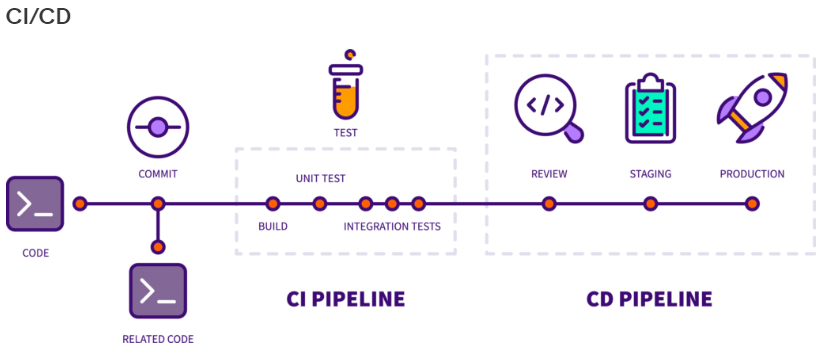
Par exemple, seul le owner a le droit de modifier le nom d’un projet. Un developer peut créer une nouvelle branche (tout comme le owner et le maintener), alors qu’un reporter n’en a pas le droit.



Ce board rassemble ce que GitLab appelle les **issues**. Bien que issue signifie problème en français, il peut s’agir aussi bien de **bugs** que de **demande d’évolutions** ou de **features déjà prévues** par exemple.

Affichant les issues crées sous formes de cartes dans des listes personnalisables, à la manière de **Trello**, cette fonctionnalité est très utile pour rassembler de manière visuelle l’avancement du projet.

Ce tableau interactif, de part son format et son affichage, peut parfaitement **s’adapter aux méthodes agiles** (par exemple en copiant un tableau de type **Kanban**).



omme je l’ai déjà dit, **GitLab inclut une fonctionnalité de CI/CD**.

Si vous ne savez pas exactement ce que c’est le CI/CD (**Contiunous Integration, Continuous Deployment**), c’est, pour faire court, le fait de déployer automatiquement (sous diverses conditions), votre code. Typiquement, un commit, via une certaine **pipeline**, va déclencher automatiquement les actions suivantes : build, exécution des tests, déploiement en staging et/ou en prod ; et ce **sans action de la part d’un humain**.

Ce type de fonctionnalités a pris de l’essor depuis l’apparition et la montée en puissance du **mouvement DevOps**, et est aujourd’hui utilisé dans beaucoup d’organisations.

GitLab inclut donc nativement ce process. Relativement facile à mettre en place, c’est une fonctionnalité évidemment très appréciée du logiciel.

Démarrer avec GitLab

Démarrage

**L’utilisation de GitLab est plutôt intuitive**. L’interface est différente de celle de GitHub, pour ceux qui viennent d’abord de ce logiciel, mais elle est réputée plus claire, plus facile à comprendre.

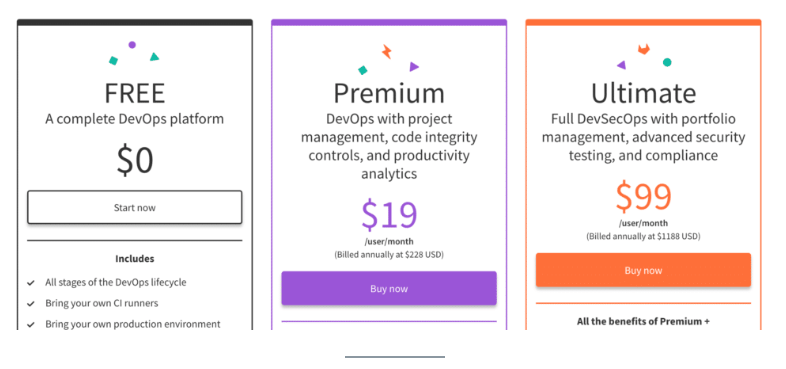
Il existe évidemment de la [documentation en ligne](https://docs.gitlab.com/ee/README.html) pour vous aider tout au long de l’utilisation de GitLab.

Abonnements

Si GitLab, en version gratuite, offre plus de fonctionnalités que son cousin GitHub, il y a aussi une [version payante](https://about.gitlab.com/pricing/), proposant plus de features.

Si la version gratuite est largement suffisante pour un développeur solo, les versions **Premium** ou **Ultimate** sont fortement recommandées pour les entreprises.

Voici les prix proposés par GitLab :



On l’a vu, **GitLab est un logiciel libre puissant, complet**, permettant à un développeur ou à une organisation de gérer ses projets, son code, de manière efficace.

Que vous soyez utilisateur de GitHub et pensez à passer sous GitLab, ou tout simplement que vous hésitiez entre les deux, je vous conseille de choisir GitLab si vous avez besoin des fonctionnalités propres à cet outil dont on a parlé tout au long de cet article. Dans le cas contraire, GitHub, sa communauté et sa stabilité feront aussi bien l’affaire.