PRÉPARÉ PAR
THOMAS NOURRY
QUENTIN CHARRIER
JEREMY MARTIN
ILYES BENNAGA

PROFESSEUR
JEREMY TORRES

VEILLE TECHNOLOGIQUE

AVRIL 2021

74 RUE FAIDHERBE, 58000 NEVERS

/cs2i
Bourgogne

PRÉPARÉ PAR
THOMAS NOURRY
QUENTIN CHARRIER
JEREMY MARTIN
ILYES BENNAGA

PROFESSEUR
JEREMY TORRES

SOMMAIRE

PRÉPARÉ PAR
THOMAS NOURRY
QUENTIN CHARRIER
JEREMY MARTIN
ILYES BENNAGA

PROFESSEUR
JEREMY TORRES

LOGICIEL DE GESTION DE VERSION

Un **logiciel de gestion de versions** est un logiciel qui permet de **stocker un ensemble de fichiers en sauvegardant l'ensemble des modifications qui lui ont été apportées**.

Il faut savoir qu'il existe des **logiciels et services de gestion de version centralisé** mais également **distribué**.

Ainsi, la première question à se poser porte sur la mise en place du logiciel de version.

CENTRALISÉ VS DISTRIBUÉ

Logiciel de gestion de versions centralisé

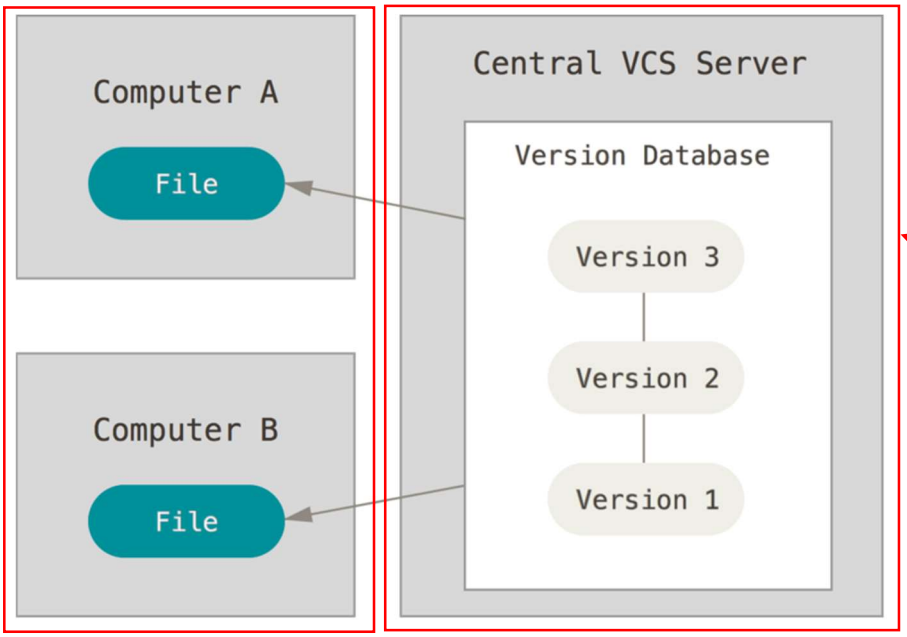
Définition et concept

Les **logiciels de gestion de versions centralisés (CVCS : Centralized Version Control Systems)** offrent une **gestion centralisée du code source**. Ainsi, il n'existe qu'**un dépôt**, c'est-à-dire qu'une copie de l'intégralité des fichiers d'un projet et de leur version, qui est donc située sur **un serveur central**.

Les développeurs se connectent au logiciel de gestion des versions suivant le principe du client/serveur.

Un **schéma** permet d'illustrer le fonctionnement d'un logiciel de gestion de versions centralisé :

Les développeurs n'ont pas accès aux différentes versions mais n'ont que la dernière



Le dépôt est centralisé sur le serveur

Avantages et inconvénients

Les logiciels de gestion de versions centralisés offrent de nombreux **avantages**.

Apparus en premier sur le marché, ils ont rapidement permis aux développeurs de se **détacher du classique « copier/coller »** avec un numéro de version par fichiers au profit d'une **mise à jour systématique des fichiers modifiés**.

Bien que l'**administration** soit **facilitée** par le fait que tout est centralisé sur un seul serveur, ces logiciels ont rapidement montré leurs **limites** dans un contexte où les projets informatiques prennent de plus en plus d'ampleur et dans lesquels le nombre de participant ne fait que croître.

En effet, cette gestion centralisée est aujourd'hui un **frein** quant aux développements d'applications modulaires dans lesquels nous souhaitons ajouter des fonctionnalités au fur et à mesure étant donné **que chaque modification est forcément poussée sur le serveur de versioning**, ce qui implique donc que **tout le monde récupère les versions**.

De plus, lorsqu'on travail à l'aide d'un logiciel de versions centralisé, il est nécessaire de pouvoir **se connecter au logiciel** afin de récupérer la dernière version et il est ainsi **impossible de travailler en mode déconnecté**. Il est d'ailleurs important de noter que la connexion sur un serveur distant étant nécessaire, cela reste **plus lent qu'une connexion au disque dur** de l'ordinateur.

Enfin, la centralisation est, comme nous le savons, un gage de **fiabilité**, mais il est nécessaire **d'opérer des sauvegardes régulières** sous peine d'avoir un problème et de perdre l'ensemble de notre travail.

Interne	**FORCES**	**FAIBLESSES**
	• Mise à jour des fichiers modifiés par l'ensemble des développeurs • Versioning des changements apportés aux fichiers • Fiabilité de la centralisation	• Toutes les modifications sont toujours poussées et on ne peut pas en garder qu'une partie en local
Externe	**OPPORTUNITÉS**	**MENACES**
	• Simple d'utilisation et d'administration	• Il est nécessaire de se connecter au serveur distant ce qui implique des temps de réponse plus conséquents • Obligation d'effectuer de nombreuses sauvegardes en dehors du serveur de versioning

Conclusion

Les **logiciels de versions centralisés** sont des logiciels qui **ont fait leurs preuves** mais qui semblent **aujourd'hui quelque peu dépassés** au vu des besoins grandissants en termes de travail (pouvoir travailler de manière déconnectée, de n'importe où). De plus, ces logiciels ne **permettent pas aux développeurs d'avancer de manière indépendante** sur des modules d'un projet sans se soucier des répercussions futurs (fusions de codes …) , ce qui s'avérer handicapant sur des projets conséquents.

Logiciel de gestion de versions distribués

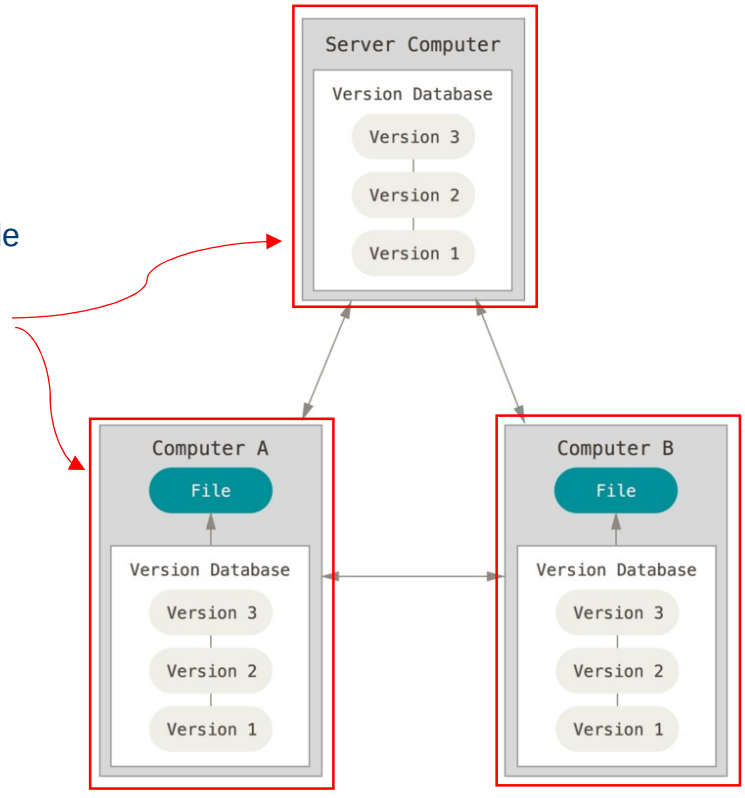
Définition et concept

Les **logiciels de gestion de versions distribués (DVCS : Distributed version control system)** offrent une gestion **décentralisée** (hors ligne) des codes sources. Ainsi, il n'existe plus qu'un dépôt situé sur un serveur central mais **autant de dépôt que de développeurs** sur le projet. Bien évidemment, le logiciel de gestion de versions fournit un **service de synchronisation entre toutes les bases** (fichiers et leurs versions).

Cette solution fonctionne suivant le principe du **pair-à-pair**. Cependant, il peut exister un **dépôt de référence** contenant les versions livrées.

Un **schéma** permet d'illustrer le fonctionnement d'un logiciel de gestion de versions distribués :

Un dépôt principal (celui de référence) et un dépôt pour chaque développeur



Avantages et inconvénients

L'**avantage** principale des logiciels de gestion de versions décentralisés est que chaque développeur du projet possède une **copie locale d'un « repository » ou répertoire** de référence ou maître avec lequel elle est synchronisée. Ainsi, cela permet à chaque développeur **d'avancer à son rythme sur une fonctionnalité** et de **demander l'avis à ses collègues** avant de repousser son travail sur le dépôt de référence.

De plus, souvent, la **gestion des « merge »,** c'est-à-dire des fusions de codes est grandement **simplifié** avec les DVCS : le logiciel de gestion des versions effectue **automatiquement les fusions de codes** lorsqu'il n'y a pas de conflit et dans le cas contraire met en avant ces conflits afin qu'on puisse les vérifier et les résoudre.

Encore, il est plus simple pour les développeurs de **développer en mode non connecté** car ceux-ci peuvent tout simplement synchroniser leur répertoire local moins souvent étant donné que la gestion des « merges » est simplifiée et qu'ils n'ont donc pas la crainte de devoir fusionner l'ensemble des codes à la main.

Enfin, le code source et son versioning est sauvegardé sur l'ensemble des repositories locaux et n'est plus centralisé sur une seule machine : cela implique **moins de risque de perdition de données**.

Bien évidemment, les DVCS ne sont pas à la portée de tout le monde et il est bien souvent nécessaire de faire **monter en compétence l'ensemble des développeurs** afin que tout le monde ait les bonnes pratiques sous peine d'avoir des complications par la suite. Cela entraine alors des **coûts indirects** : coûts de formation(s), de migration(s) (si on utilisait déjà un logiciel centralisé) et bien d'autres.

De plus, il est à noter que lorsque les **projets** deviennent **conséquents** (un grand historique de versioning), **télécharger l'entièreté de l'historique peut être long** et prendre **beaucoup de place sur un disque dur**, d'où l'importance de ne pas essayer de toujours resynchroniser des tâches non terminées.

<table>
<tr><td rowspan="2">Interne</td><td>FORCES</td><td>FAIBLESSES</td></tr>
<tr><td>Chaque développeur peut avancer à son rythme
La gestion des merge est grandement simplifiée et peut s'effectuer en local avant d'être repoussé sur le serveur de référence</td><td>Espace disque nécessaire pour des projets conséquents
Temps de récupération sur des projets très conséquents</td></tr>
<tr><td rowspan="2">Externe</td><td>OPPORTUNITÉS</td><td>MENACES</td></tr>
<tr><td>Développer en mode déconnecté aisément</td><td>Difficulté relative de prise en main</td></tr>
</table>

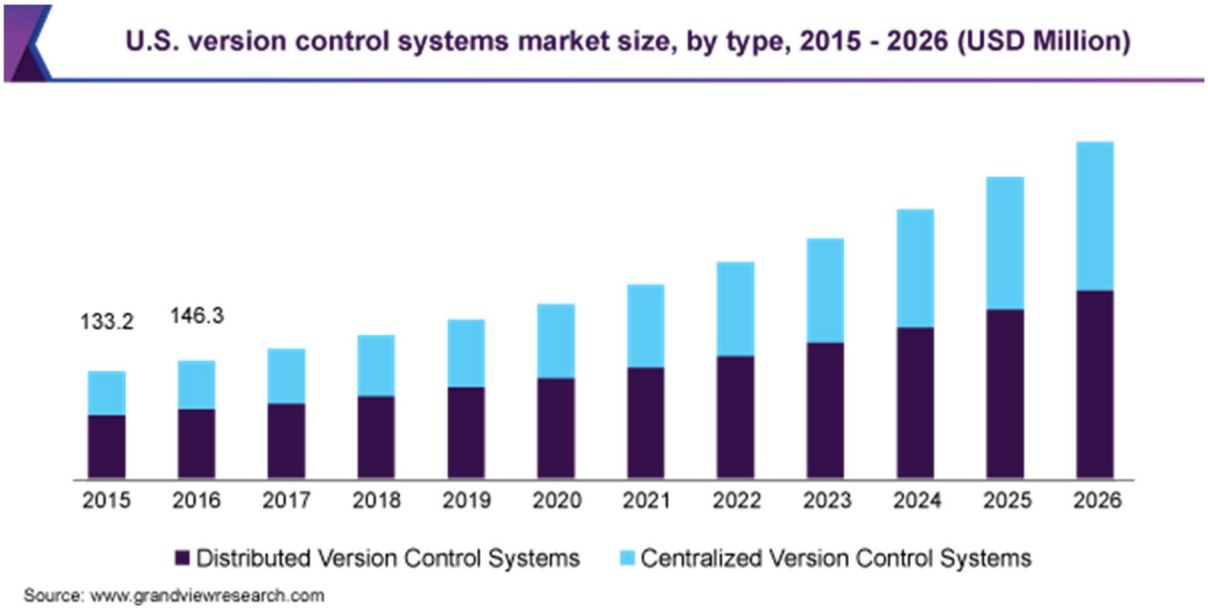
		• Coûts indirects (formation, déploiement…)

Conclusion

Les **logiciels de versions distribués** offrent une toute **nouvelle expérience pour les développeurs**. Bien que **plus complexe à prendre en main**, grâce à ces logiciels, chacun des développeurs du projet peut se permettre **d'avancer à son rythme** et choisir de se **resynchroniser avec le dépôt de référence qu'une fois l'ensemble de son développement terminé sans avoir à se soucier de la fusion des codes et des erreurs pouvant en découler**.

Conclusion

Bien que les logiciels de gestion de versions **centralisés** soient encore **grandement utilisés** (53% d'utilisation en 2019 aux Etats-Unis contre 47% pour les DVCS selon le graphique suivant), il est à noter que les logiciels de gestion de versions **distribués** offrent de **nombreux avantages** en comparaison aux logiciels centralisés. **Possibilité de fusionner les codes efficacement et automatiquement, de se synchroniser quand on le souhaite sans avoir à se soucier des problèmes de versions ou encore de développer en mode 'non connecté' sont les atouts qui nous ont poussé à choisir d'utiliser un logiciel de version distribué**.



LOGICIELS DE GESTION DE VERSIONS DISTRIBUÉS

Deux grands noms ressortent lorsqu'on parle **de logiciel de gestion de versions distribué** : **GIT** et **Mercurial**. C'est pourquoi nous avons décidé d'axer notre étude sur ces deux logiciels de gestion de versions.

Afin de bien **comprendre de quoi on parle**, quelques **définitions** de mots communément **utilisés dans le monde du logiciel de gestion de versions** :

Branche : Les **branches** permettent de **travailler sur différentes versions de code** en même temps pour permettre ensuite **aux développeurs de fusionner les branches sans casser le code existant**

Repository : Un **repository** ou **dépôt** (ou encore référentiel) est un **stockage** sur lequel est sauvegardé **l'ensemble des données de manière centralisée**. Il est important de noter que les données sont stockées de manière centralisées sur le dépôt, mais que dans le cadre d'un logiciel de versions distribué il existe autant de dépôts que de développeurs.

Commit : Un **commit** est un procéder permettant **d'enregistrer dans un dépôt les modifications apportées**.

- GIT
- Présentation

Git est **un logiciel de gestion de versions distribué**, **libre** et créé par **Linus Torvalds**, célèbre créateur du noyau Linux.

Sa première version a été publié le 7 avril 2005. En juillet 2005, Linus Torvalds a décidé de confier la maintenance et les évolutions de Git à **Junio Hamano**.

Distribué selon les termes de la **licence public générale GNU version 2**, en 2016 ce logiciel était **le plus populaire** et utilisé par plus de douze millions de personnes.

On estime qu'aujourd'hui Git possède plus de **80% des parts de marché**.

80% : part de marché estimé

Avantages et inconvénients

Git est un logiciel de gestion de version très **complet** et très **flexible**. Son **système de branches est très avancé** et permet aux développeurs de pouvoir **créer, modifier ou supprimer des branches** comme bon leur semble.

La « **zone de transit** » (staging area en anglais) est très pratique car elle permet aux développeurs de ne pas avoir à tout commit lors de modifications et de pouvoir choisir ce qu'ils veulent repousser sur la base de données.

Git **interagit** très bien **avec des outils** tel que Slack ou encore Visual Studio Code. D'ailleurs, les **extensions** de Git peuvent être **écrites dans tout les langages** possibles, ce qui peut être pratique dans le cadre d'un projet conséquent où l'on pourrait écrire une extension dédiée.

De plus, la **migration de Git vers Mercurial est simple et le risque de perte de données est quasiment inexistant**.

Toutefois, cette flexibilité se paye par une **courbe d'apprentissage assez conséquente** ainsi qu'une **documentation complexe**. Des **novices** de Git pourrait rapidement se voir dans la tourmente s'ils ne faisaient pas les bonnes manipulations, cela pouvant même entrainer des **suppressions irrécupérables**.

La **flexibilité** relative de Git entraine parfois des **questions quant à la bonne pratique à adapter** afin de ne pas se retrouver bloqué plus tard. Il est donc très important de réfléchir en amont à son organisation tout en ayant en tête les fondamentaux.

Enfin, Git fonctionnant sous ligne de commande, il existe des **commandes d'aide** mais qui ne sont **pas assez documentées**. Le fonctionnement en ligne de commande pousse à **devoir choisir un bon GUI** (une interface graphique) afin de réussir à simplifier les processus.

Interne	**FORCES**	**FAIBLESSES**
	• Complet et flexible • Système de branche avancé	• Courbe d'apprentissage conséquente • Documentation complexe • Commandes d'aide pas assez documentées • Importance de choisir un bon GUI
Externe	**OPPORTUNITÉS**	**MENACES**

	• Zone de transit ou staging area • Interagit très bien avec des outils externes • Extensions pouvant être écrites dans n'importe quel langage • Migration vers Mercurial sans perte de données	• Flexibilité pouvant entrainer des questions sur les bonnes pratiques • Flexibilité dangereuse si on ne s'y connait pas assez

Conclusion

Git est aujourd'hui **LA référence**. Bien que **complexe à prendre en main**, sa **flexibilité** est un **atout majeur** pour les développeurs qui veulent toujours pouvoir faire **comme ils veulent et comment ils veulent**. Les **perspectives de travail** avec **des outils** comme Slack ou Visual Studio Code et bien d'autres sont très intéressantes, tout comme **le système de zone de transit** qui permet de ne pas avoir à toujours repousser l'ensemble de ses changements.

Il sera bien évidemment nécessaire de **se documenter tout au long de la mise en place** d'un logiciel tel que Git afin **de comprendre les fondamentaux** tout en **réfléchissant au plan d'action** afin de ne pas être bloqué par le futur.

Mercurial

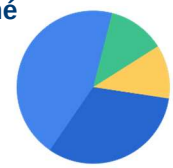
Présentation

Tout comme **Git**, **Mercurial SCM** est un **logiciel de gestion de versions distribué libre** disponible sous la **licence GNU GPL version 2**.

Son développeur principal est **Matt Mackall**.

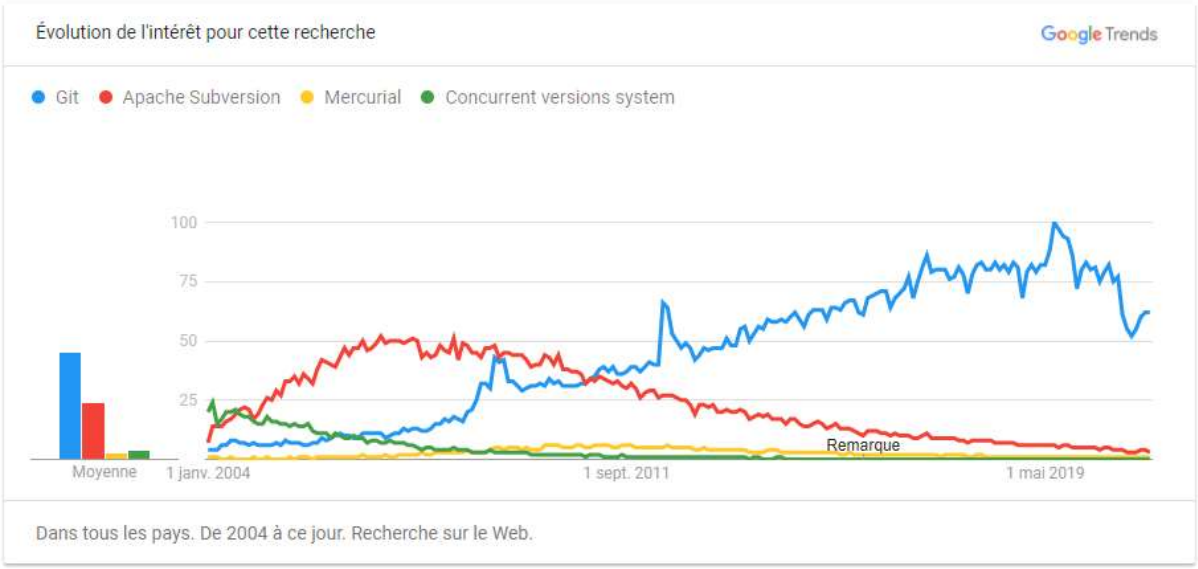
C'est lui-même qui a mis à disposition Mercurial pour la première fois le **19 avril 2005**, seulement **quelques jours après que le développement de Git** ait commencé.

2% : part de marché estimé



En **termes de parts de marché**, **Mercurial** est logiquement **loin derrière Git**, et on estime aujourd'hui que Mercurial ne possède que 2% des parts de marchés (Subversion, un logiciel centralisé, étant largement devant)

On pourra noter une **évolution de l'intérêt** pour mercurial de 2004 à 2019 **très faible** en comparaisons à Git suivant ce diagramme :



Avantages et inconvénients

Contrairement à Git, Mercurial semble **plus simple**. Sa **documentation** est **très complète et très simple** tout comme **la syntaxe de ses lignes de commande**. La **courbe d'apprentissage** étant plus **faible**, nous pouvons plus rapidement nous tourner vers les développements.

Toutefois, cette simplicité se paie par un **manque de flexibilité**, certes **sécurisant pour des novices qui ne peuvent alors pas supprimer tout l'historique de versions**, mais qui peut rapidement se trouver **contraignant sur des projets conséquents** avec beaucoup d'acteurs.

De plus, le **système de branche est complètement différent** de celui de Git, ainsi il n'est **pas possible de créer, modifier, supprimer et fusionner des branches comme bon nous semble** et il sera très **important de bien réfléchir en amont à comment organiser le projet** étant donné que nous ne pouvons pas supprimer les braches (elles font partie de l'historique). Le **retour en arrière** est donc plus **sécurisé**, mais encore une fois peut-être **bloquant sur des projets conséquents**.

Afin de profiter d'une « **zone de transit** » tel que celle de Git, il est nécessaire de passer par des **extensions supplémentaires**, extensions ne pouvant être écrites qu'en **Python**, cela limitant donc le nombre d'extensions et les possibilités de création pour des équipes qui ne connaissent pas ce langage.

Enfin, et cela est très probablement un point central, il n'est pas possible de passer de **Mercurial à Git** sans **perdre des données**, ce qui est très contraignant pour des équipes qui pourrait vouloir gagner en flexibilité après avoir gagné en confiance sur un outil simple.

Interne	**FORCES**	**FAIBLESSES**
	• Simple d'utilisation • Facile à comprendre • Documentation complète et simplifié • Lignes de commandes simple et intuitives	• Système de branche particulier (impossible d'en supprimer et modifier)
Externe	**OPPORTUNITÉS**	**MENACES**
	• Sécurisé pour des novices	• Installation d'extensions pour activer une « zone de transit » • Extensions en Python • Migration vers Git entraine des pertes de données

Conclusion

Mercurial, bien que **peu connu** semble être un **logiciel de gestion de versions simple**. Dans le cadre de **projets peu conséquents**, cet outil semble être un **outil adéquat**, mais dans le cadre de **projets d'ampleur** avec de nombreux développeurs, cet outil pourrait rapidement **manquer en flexibilité**. De plus, le **manque d'extensions** découlant de la **contrainte « Python »** est certain et pourrait sur le **long terme porter préjudice aux équipes de développement**.

Conclusion

Aujourd'hui, **Git** s'est aujourd'hui très clairement **imposé sur le marché** au point que **Mercurial** n'est presque **pas connu**, cela s'expliquant certainement par sa **flexibilité** mais également par ses **possibilités d'intégrations** dans de nombreux environnements en couplage **avec de nombreux autres logiciels**. Malgré une **courbe d'apprentissage complexe**, il semble très important de prendre en compte la possibilité de **migrer aisément de Git vers Mercurial**, ce qui permet d'avoir une **issue de secours**, issue non disponible quand on est sur Mercurial.

C'est pourquoi, l'équipe du projet a décidé de se tourner vers **Git** : malgré une documentation officielle complexe, il existe de **nombreux forums** et **sujets** où l'on peut trouver des **informations** et des **explications claires**. De plus, nous sommes déjà à l'aise avec Git, ayant déjà eu des cours d'initiation à ce logiciel de gestion de version.

COMPARAISON DES SERVICES D'HÉBERGEMENT DE GIT

Afin d'utiliser **Git** de manière **simplifiée**, sans avoir à tout faire en ligne de commande et afin de pouvoir **partager le travail entre tout les membres de projet**, il est nécessaire de choisir un **bon service d'hébergement**.

Nous comparerons ici les **trois services d'hébergement** les plus connus et en vogue : Github, GitLab et Bitbucket.

GitHub

Présentation

GitHub est une entreprise de développement logiciel et de service dont le siège est situé aux Etats-Unis. Elle est notamment à l'origine de la **plateforme GitHub**, de **l'éditeur de texte Atom** ou encore du **Framework Electron**.

La **plateforme GitHub**, racheté en 2018 par Microsoft pour **7,5 milliards de dollars**, est un **service d'hébergement de référentiel Git** basé sur le Cloud. En simplifié, GitHub est une plateforme permettant **d'utiliser Git à travers diverses interfaces** à des fins de **simplification** pour les utilisateurs novices du logiciel gestion de versions.

Aujourd'hui, GitHub compte près de **85 millions de référentiels** (de projets hébergés sur GitHub) et elle se revendique de **50 millions d'utilisateurs**.

50M d'utilisateurs

56.16% : part de marché estimé

Selon le site Slintel, on estime aujourd'hui que la plateforme possède **56.16% des parts du marché** de la gestion de version à travers le monde.

Avantages et inconvénients

GitHub n'a plus besoin de faire ses preuves. Doté d'une **communauté de près de 50millions d'utilisateurs**, sa **documentation est complète**, ou du moins il est très **facile de trouver des informations en ligne** du fait de la **communauté très active**. GitHub est un service d'hébergement en ligne pour Git, ce qui signifie qu'il permet à l'ensemble des développeurs d'une entreprise par exemple de pouvoir **accéder au projet de n'importe où** sans avoir à se connecter au VPN de celle-ci.

GitHub met à disposition **GitHub desktop**, permettant de simplifier l'ensemble des actions Git à travers une **interface graphique intuitive** sans avoir à passer par des lignes de commande sur notre ordinateur.

Ce service permet également, depuis peu, de faire de **l'intégration continue** par le biais de **GitHub Actions** (**2000 minutes d'intégration continuent par mois**), mais également de pouvoir mettre en place des **systèmes de workflow de validation de codes** entre les branches (**pull requests**), permettant ainsi aux développeurs de relire le code des autres développeurs avant de les passer en production par exemple.

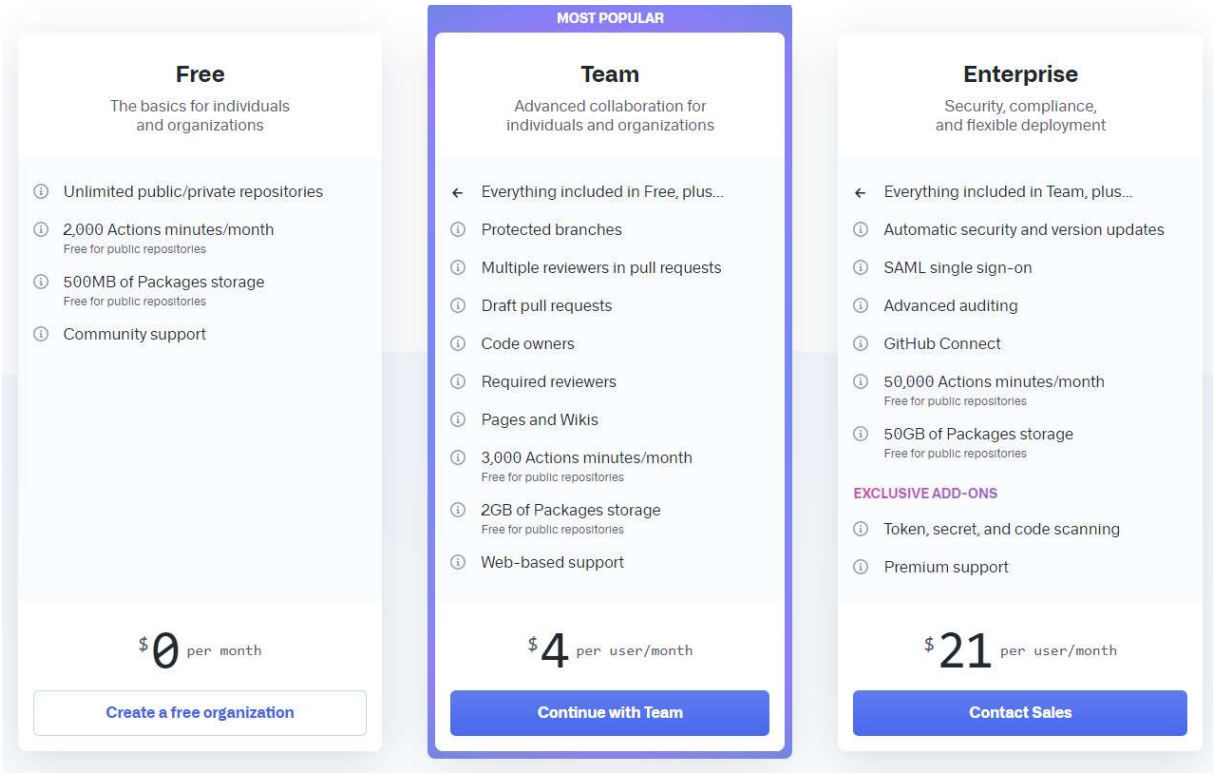
Bien que GitHub propose de nombreuses fonctionnalités gratuites, il sera nécessaire d'opter pour une **version payante** afin de débloquer **l'ensemble des fonctionnalités** notamment si l'on souhaite rendre son **repository privé** (avec un repository public il n'y a pas de problème). Il faudra compter de **4$ par utilisateurs par mois à 21$ par utilisateurs par mois**.

La **version gratuite** permet de créer un nombre de **repository illimité** et de **collaborer avec autant d'utilisateur que l'on souhaite**, que le repository soit public ou privé.

Toutefois, on notera la **limitation de taille des repository à 1gb** ce qui peut être un frein pour des projets conséquents, ainsi **qu'une limitation de taille de fichier à 100mb**.

GitHub n'étant **pas open source**, il n'est pas possible de modifier le code source afin de rajouter des fonctionnalités utiles pour l'entreprise.

Enfin, il est parfois avancé que **des comportements étranges** surviennent lors des **fusions de codes** (**merges**), comportements qui sont très probablement causés par une mauvaise utilisation plus que par la plateforme elle-même.



Interne	**FORCES**	**FAIBLESSES**
	• Communauté très active • Documentation complète • Interface graphique intuitive avec GitHub Desktop • Pull requests • Repository illimités avec utilisateurs illimités	• Repository limité à 1GB • Version payante pour l'accès aux fonctionnalités si repository privé (4$ à 21$ par utilisateur par mois) • Pas open source
Externe	**OPPORTUNITÉS**	**MENACES**
	• Hébergement en ligne → accès depuis n'importe où • Intégration continue avec GitHub Actions	• Comportements étranges lors des merges

Conclusion

GitHub est un outil très **réputé** et très **intuitif**. Sa **communauté très active** lui permet de **prospérer** et permets à tout novice de pouvoir **prendre en main ce service rapidement**.

Le principal défaut restera la nécessité de passer par une **version payante** afin de **débloquer l'ensemble des fonctionnalités pour un repository privé**.

GitLab

Présentation

GitLab est **un logiciel libre de forge** basé sur **Git** proposant les fonctionnalités de **wiki**, un **système de suivi des bugs**, **l'intégration continue** et la **livraison continue**.

Développé par **GitLab Inc** et créé par **Dmitriy Zaporozhets** et par **Valery Sizov**, le logiciel est utilisé par plusieurs grandes entreprises informatiques incluant **IBM**, **Sony**, ou encore la **Nasa.**

GitLab est scindé en **deux versions**, l'une libre sous **licence MIT** nommée **GitLab CE** et l'autre contenant des modifications propriétaires sous licence GitLab EE et nommé **GitLab EE**.

En 2020, **GitLab** revendiquait plus de **100 000 organisations** utilisant sa plateforme à travers le monde, ce qui représenterai un total de plus de **30 millions d'utilisateurs** enregistrés. Toutefois, les parts de marchés sur le site Slintel sont plus faibles que celles de GitHub et équivaudraient à seulement **39.20%**.

39.20% : part de marché estimé

30M d'utilisateurs

Avantages et inconvénients

GitLab est un service très puissant qui a su se démarquer au départ par son offre permettant de faire de **l'intégration continue** sur ses projets.

La **communauté** de GitLab, certes moins conséquente que celle de GitHub, est toutefois **très active**, ce qui permet d'avoir de trouver de nombreuses informations en ligne. La **documentation** proposée par GitLab est très **complète** et **simple de compréhension**.

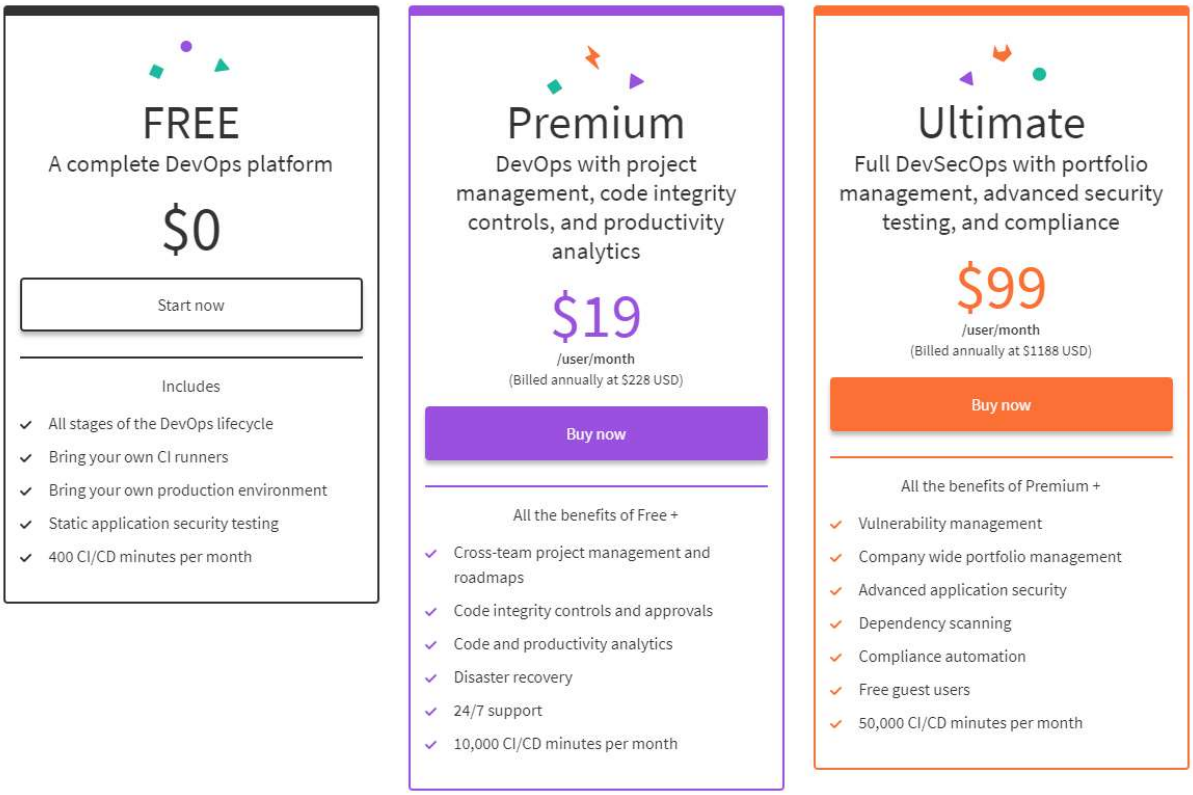
La version **open-source** de GitLab est **entièrement téléchargeable et modifiable**, ce qui peut s'avérer être un avantage dans la cadre où nous aurions besoin de fonctionnalités supplémentaires.

Tout comme GitHub, GitLab propose un système de **workflow de validation de codes (pull requests)** et un **système de repository public ou privé**. Toutefois, ces **repository** peuvent être plus conséquents et atteindre les **10gb** sans problème.

Bien évidemment, GitLab étant un **service d'hébergement en ligne**, cela permet encore une fois **d'accéder depuis n'importe où aux codes et documentations**.

Le principal défaut de GitLab pourrait aujourd'hui être son **interface** qui reste quelque peu **plus complexe que celui de GitHub**, mais également le **prix** des versions payantes pouvant aller de **19$ par utilisateur par mois à 99$ par utilisateur et par mois**.

On notera également la **limitation en termes de temps d'intégration continue en version gratuite qui est de 400 minutes**, ce qui est **4 fois moins que celui proposé par GitHub.**



Interne	**FORCES**	**FAIBLESSES**
	• Communauté active • Documentation complète et simple de compréhension • Open-source • Pulls requests • Repository public et privé • Repository de 10GB	• Interface complexe • Version payante pour l'accès aux fonctionnalités si repository privé (19$ à 99$ par utilisateur par mois)
Externe	**OPPORTUNITÉS**	**MENACES**
	• Intégration continue • Hébergement en ligne → accès depuis n'importe où	• Intégration continue limitée à 400 minutes par mois en version gratuite

Conclusion

GitLab est un **outil performant** qui est avant tout utilisé pour son **système d'intégration continue** basé sur Jenkins. Malgré une **communauté plus faible** que celle de GitHub, il existe de **nombreux forums** afin de trouver des informations et une **documentation officielle très complète**.

A première vue, GitLab semble toutefois **moins intuitif en termes d'interface**, notamment pour des personnes ayant déjà utilisés GitHub tel que les membres du projet.

Bitbucket

Présentation

ATLASSIAN Bitbucket

Bitbucket est **un service web d'hébergement** et de **gestion de développement logiciel** utilisant le **logiciel de gestion de versions Git** (et par le passé également le logiciel **Mercurial**).

6M d'utilisateurs

Il s'agit d'un **service freemium** dont la version gratuite permet déjà de créer jusqu'à un **nombre illimité de dépôts privés**, accessibles par **cinq utilisateurs au maximum**.

2.04% : part de marché estimé

Selon le site Slintel, **Bitbucket** représenterai aujourd'hui environ **2.04% des parts du marché**. Bitbucket revendiquerai **6 millions d'utilisateurs**.

Avantages et inconvénients

Bitbucket est très peu connu mais propose de **solides fondations**, notamment dans son **intégration avec Jira**, le logiciel de suivi de tickets. Bitbuket propose également son outil **d'intégration continue**, mais qui semble bien **moins performant que celui proposé par GitLab ou encore GitHub** et qui, en version gratuite, est limité à **50 minutes par mois**.

Sa **documentation est relativement faible** et le **manque de communauté** se fait très vite ressentir. L'interface nécessite une **courbe d'apprentissage** mais reste **assez simple**.

Les **repository** sont limités à **1gb** tout comme pour GitHub, mais il est impossible de **les archiver par le biais de l'interface**. Il sera donc nécessaire de **connaitre quelques lignes de commande de Git**.

En termes de prix, BitBucket propose une **version gratuite** mais également deux **versions payantes de 3$ et 6 $ par utilisateurs et par mois**. Il est a noté que la **version gratuite empêche la collaboration de plus de 5 utilisateurs sur un repository**.

	Free Get started $0 Forever	Standard Try it $3 / user / month starting price	Premium Try it $6 / user / month starting price
User limit	Up to 5 users	Unlimited	Unlimited
Build minutes	50 min / mo	2500 min / mo	3500 min / mo
Git Large File Storage	1 GB	5 GB	10 GB
Unlimited private repositories	✓	✓	✓
Jira Software integration	✓	✓	✓
Trello integration	✓	✓	✓
CI/CD	✓	✓	✓
Unlimited pull request reviewers	✓	✓	✓
Code Insights	3 integrations	Unlimited	Unlimited
Merge checks	✓	✓	✓
Enforced merge checks			✓
Deployment permissions			✓
IP Whitelisting			✓
Required two-step verification			✓
Standard support	✓	✓	✓

Conclusion

BitBucket semble être **très limité** par son **manque de communauté** et donc de **documentation**. C'est un outil **simple et pratique** pour des **projets peu complexe** et **non conséquents**. BitBucket ne se **démarque pas de ses concurrents** et **propose moins de fonctionnalités**, ce qui semble expliqué sa **faible popularité**.

Conclusion

GitHub, **GitLab** semblent très clairement être **en avance sur Bidbucket**. Proposant de **meilleurs fonctionnalités** mais également une c**ommunauté beaucoup plus grande et active**, **ces solutions sont complètes** et proposent même de **l'intégration continue**, un point crucial pour notre projet.

La principale **divergence** entre **GitLab** et **GitHub** semble porté sur la **mise en place de l'intégration continue qui**, pour **GitHub**, nous oblige à **mettre en place un repository public ou bien a payé**, mais également sur les tarifs appliqués par ses solutions (**GitLab est beaucoup plus chère que GitHub**). Il est d'ailleurs à noter que le **nombre de minutes d'intégration continue de la version gratuite de GitLab** semble être vraiment **faible en comparaisons à l'offre proposé par GitHub.**

CONCLUSION

L'analyse des différents **logiciels de gestion de versions** et des **services d'hébergement** proposés en complément nous permets aujourd'hui de porter notre choix sur **GitHub** étant donné que nous **connaissons déjà cet outil** et qu'il possède **l'ensemble des fonctionnalités utiles à une bonne gestion des versions d'un projet.**

Bien que **GitHub** intègre un **outil d'intégration continue**, il sera nécessaire d'étudier **la pertinence d'utiliser leur outil plutôt qu'un autre**.

OUTIL D'INTÉGRATION CONTINUE

L'intégration continue est un ensemble de pratiques consistant à vérifier à chaque modification de code source que le résultat des modifications ne produit pas de régression dans l'application développée.

Le **principale but** de cette pratique est donc de **détecter au plus tôt les problèmes d'intégration lors du développement**. De plus, cette pratique permet **d'automatiser l'exécution des tests**.

Cette pratique a donc pour finalité **l'amélioration de la qualité du code mais également du produit final**.

Afin d'appliquer cette technique, il est nécessaire que :

- Le **code source soit partagé** entre les développeurs par le biais d'un **logiciel de gestion de versions**
- Les **développeurs intègrent régulièrement leurs modifications**
- Des **tests d'intégration** soient développés pour **valider l'application.**

Dans notre cas de figure, l'ensemble des développeurs du projet passent par **GitHub** afin de **gérer les différentes versions de codes et documentations** et **intègrent régulièrement leurs modifications**.

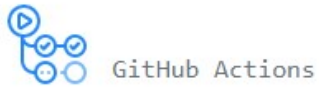
De plus, **des tests d'intégration et unitaires** seront développés **pour chaque nouvelles fonctionnalités.**

Comme mentionné précédemment, **GitHub** propose leur **propre outil d'intégration continue** : **GitHub Actions**. Toutefois, il est important de savoir qu'il n'est **en aucun cas obligatoire d'utiliser cet outil** si nous utilisons **GitHub comme un simple outil de gestion de versions**. En effet, il existe d'autres outils que nous pourrons coupler à GitHub afin de faire notre intégration continue tel que **GitLab CI** ou encore **Travis CI**.

COMPARAISON DES SOLUTIONS

GitHub Actions

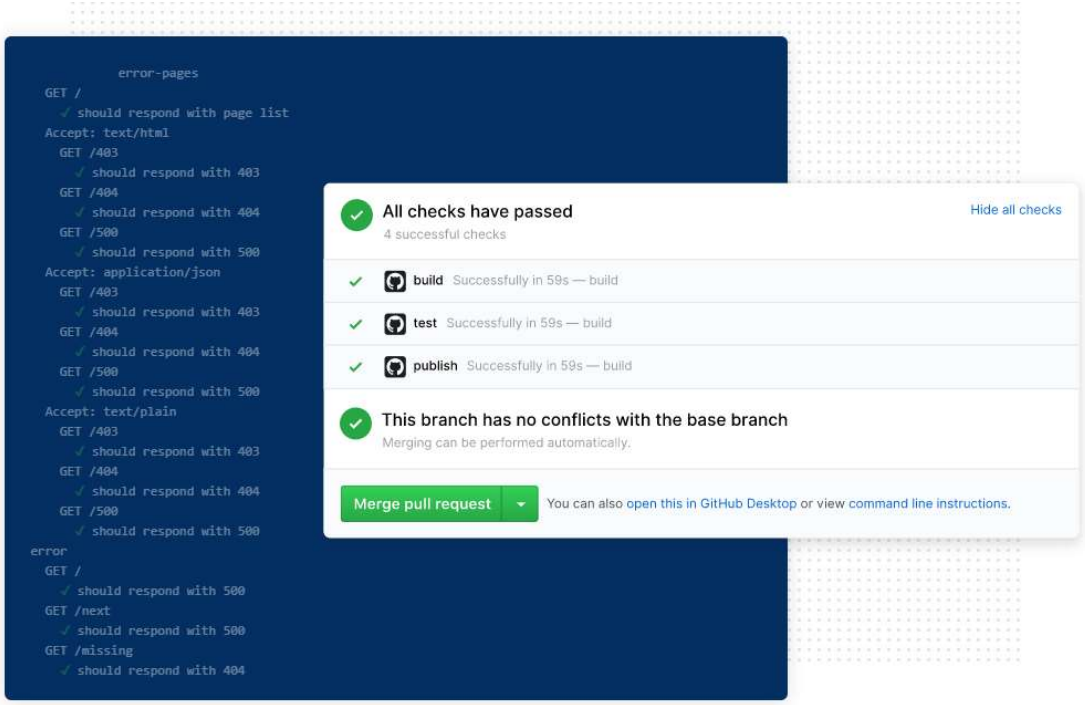
Présentation



GitHub Actions est la **solution propulsée par GitHub** afin de permettre de faire de **l'intégration continue**. Cette solution **sortie en 2018** pour la première fois a très vite grandit afin de pouvoir **concurrencer la solution proposée par GitLab, GitLab CI**.

Avantages et inconvénients

GitHub Actions est une **solution récente**. Sortie en 2018, elle a toutefois su séduire l'ensemble des utilisateurs de GitHub de part sa **simplicité d'utilisation** mais également de part son **interface très intuitif**.



Malgré sa récente sortie, GitHub Actions possède déjà une **documentation assez fournie**, cela notamment grâce à la **communauté très active.**

D'un point de vue technique, GitHub Actions est une solution **SaaS** mais qui pourra être rapidement utilisé de manière auto-hébergé.

Une **marketplace** est disponible sur lequel la communauté peut publier des **pipelines pré-fait** permettant ainsi aux développeurs de gagner du temps dans leur démarche d'intégration continue.

De manière plus technique, **GitHub Actions** offre des possibilités d'**intégrations avec Docker**, un logiciel libre permettant de lancer des applications dans des conteneurs logiciels. Il est possible de créer de **nombreux pipelines**, c'est-à-dire de nombreuses tâches à exécuter les unes après les autres, le tout en **parallèle**.

La **configuration est simple** et passe par la création d'un fichier YAML dans un dossier .github/workflows. Bien évidemment, c'est **simplicité est relative à l'expérience des développeurs**.

En termes de prix, GitHub Actions étant propulsé par GitHub, les **prix sont ceux mentionnés précédemment** à savoir de 4$ à 21$ par mois et par utilisateur.

La **version gratuite** permet encore une fois de mettre en place **l'intégration continue pour des repository public à hauteur de 2000 minutes par mois**.

Evidemment, la récente mise en place de GitHub Actions signifie que certaines **fonctionnalités** que l'on pourrait retrouver dans d'autres logiciels ne sont **pas disponibles**. De plus, GitHub Actions n'est disponible **que pour les projets qui sont hébergés sous GitHub**, ce qui pourrait être un frein dans la cadre où notre projet ne serait pas hébergé chez eux.

Interne	**FORCES**	**FAIBLESSES**
	• Documentation fournie • Forte communauté • Version gratuite permettant 2000 minutes d'intégration continue • Configuration simple • Interface intuitive	• Intégration continue disponible seulement pour les repository public • Simplicité relative à l'expérience des développeurs
Externe	**OPPORTUNITÉS**	**MENACES**
	• SaaS • Marketplace avec des pipelines préconstruits • Intégration avec Docker	• Certaines fonctionnalités ne sont pas disponibles par rapport à des logiciels implantés depuis plus longtemps • Disponible seulement pour les projets hébergés sous GitHub

Conclusion

GitHub Actions est un nouveau produit très **prometteur**. **Propulsé par la communauté de fidèle de GitHub**, cette solution connait une **très forte avancé** et permet aujourd'hui de **réaliser les fondamentaux de l'intégration continue**.

Tout comme mentionné précédemment, il est nécessaire de **mettre son dépôt public** afin de pouvoir **profiter pleinement de GitHub Actions**, ce qui pourrait être un **frein pour certaines entreprises**, mais qui dans notre cas ne semble pas l'être étant donné notre choix de nous tourner vers GitHub pour la gestion de versions.

GitLab CI

Présentation

GitLab CI/CD est un **outil d'intégration et de déploiement continue propulsé par GitLab**.

Intégré en **septembre 2015** lors de la sortie de **GitLab 8.0**, cet outil est aujourd'hui un **incontournable** dans le domaine des outils d'intégration continue **SaaS** (software as a service).

GitLab CI/CD aide les développeurs à **transformer leurs idées en production** en trouvant des **améliorations potentielles à leurs processus de développement**. **Ses pipelines créent, testent, déploient et surveillent le code dans le cadre d'un flux de travail unique et intégré**. Les développeurs **partagent chaque nouveau morceau de code** dans **une demande de fusion**, ce qui déclenche le **pipeline** qui effectue toutes **les tâches de validation** avant de **fusionner les modifications** dans le **référentiel de code source**.

Avantages et inconvénients

GitLab CI/CD **est implémenté depuis maintenant 6 ans** sur le marché de l'intégration continue, ce qui lui permets de se prévaloir d'une **forte documentation officielle** et de **nombreuses extensions**.

Facile d'utilisation pour des utilisateurs expérimentés, GitLab CI/CD peu toutefois resté **déroutant pour les nouveaux utilisateurs** qui pourront tout de même s'appuyer sur la forte documentation.

GitLab CI/CD faisant partie de GitLab, il est ainsi possible de **combiner la gestion de versions avec l'intégration continue**, ce qui peut faire gagner du temps (solution tout en un)

De manière plus technique, **GitLab CI/CD** offre une **des meilleures intégrations avec Docker**, un logiciel libre permettant de lancer des applications dans des conteneurs logiciels. Il est possible de créer de **nombreux pipelines**, c'est-à-dire de nombreuses tâches à exécuter les unes après les autres, le tout en parallèle avec des phases différentes.

De plus, il est possible de voir en temps réels les **performances relatives** sur les **déploiements et l'intégration** de nos projets.

Une **API** est également disponible pour permettre de faire **des intégrations plus profondes** sur des projets en dehors de l'environnement GitLab.

Enfin, GitLab CI propose une fonctionnalité permettant de **scruter la qualité du code fournit**.

Etant intégré directement dans GitLab, il sera nécessaire de s'appuyer sur une **version payante** afin **d'augmenter le temps disponible** pour effectuer de l'intégration continue qui est de **400 minutes par mois** en version gratuite.

Bien que la **communauté** soit très présente autour du logiciel de versioning, elle semble l'être **moins sur toute la partie intégration continue**, il est donc nécessaire de se tourner vers la **documentation officielle** qui, même en étant bien complète, ne fournit pas forcément l'ensemble des réponses à nos questions.

Interne	**FORCES**	**FAIBLESSES**
	• Documentation officielle complète • Facile d'utilisation pour des utilisateurs expérimentés • Création de pipelines • Possibilité d'exécuter les pipelines en parallèle • SaaS, pas besoin de l'héberger soit-même	• Communauté moins active sur l'intégration continue • Interface peu intuitive pour des novices
Externe	**OPPORTUNITÉS**	**MENACES**
	• Capacité de combiner la gestion de version et l'intégration continue • Intégration avec Docker • API disponible pour des intégrations approfondies • Fonctionnalités de vérification du code	• Intégration continue limitée à 400 minutes par mois en version gratuite

Conclusion

GitLab CI/CD est une **solution très complète** et **assez simple de mise en place**. L'intérêt véritable de sa mise en place résiderai dans son **couplage avec GitLab** pour la **gestion des versions**.

Jenkins

Présentation

Jenkins est un outil **logiciel d'intégration continu** sorti en **2011**. Il s'agit d'un **logiciel open source**, développé à l'aide du langage de programmation **Java**. Il permet de **tester et de rapporter les changements effectués sur une large base de code en temps réel**. En utilisant ce logiciel, les développeurs peuvent **détecter** et **résoudre** les **problèmes dans une base de code** et rapidement.

Ainsi les **tests** de nouveaux builds peuvent être **automatisés**, ce qui permet **d'intégrer plus facilement des changements à un projet**, de façon continue.

L'objectif de **Jenkins** est en effet d'accélérer le développement de logiciels par le biais de **l'automatisation**. **Jenkins** permet **l'intégration de toutes les étapes du cycle de développement**.

Avantages et inconvénients

Jenkins est une solution implémentée depuis maintenant **10 ans,** ce qui lui permet de se prévaloir de la **plus grande bibliothèque d'extension** dans le domaine, d'une **forte documentation** et d'une **grande communauté**.

Très **complète** et **facile d'installation**, cette solution est **facile de prise en main** pour les utilisateurs.

Le **débogage est très simple** et le **code** est **facile à déployer**.

Disponible dans **plusieurs langues**, Jenkins est totalement **gratuit** et est **très flexible**. Il propose également une **API**.

Open-source, il est nécessaire **d'auto-hébergé** cette solution, ce qui implique donc d'avoir à disposition le matériel et l'ensemble des configuration requises afin de pouvoir faire tourner le logiciel, ce qui engendrera donc **des coûts indirects** (machines, formation si nécessaire).

Bien que Jenkins puisse se prévaloir de la plus grande bibliothèque d'extensions disponible sur le marché de l'intégration continue, ces **extensions** sont souvent **complexes à intégrer** et il faut **veiller à ne pas créer d'anomalies sur le fonctionnement globale du logiciel**.

Enfin, le **manque d'analyse et de rapports sur le suivi global des pipelines** se fait ressentir.

Interne	**FORCES**	**FAIBLESSES**
	• Documentation complète • Grande communauté • Facile d'installation et de paramétrage • Facile à prendre en main • Débogage simple • Gratuit	• Doit être auto-hébergé (coûts indirects) • Extensions complexes à intégrer
Externe	**OPPORTUNITÉS**	**MENACES**
	• Bibliothèque d'extension la plus grande sur le marché • Disponible en plusieurs langues • Flexible • Open-source	• Manque d'analyse et de rapports sur le suivi global des pipelines

Conclusion

Jenkins est une solution qui **a fait ses preuves** et a **su se démarquer** au fil des années. **Gratuite** à première vue, il est toutefois **nécessaire de prendre en compte les coûts de déploiement et d'installation de l'outil** sur des serveurs personnels.

Le **manque d'analyse** relatif à **l'exécution globale des pipelines** pourrait être un **frein dans le processus d'amélioration continue** des équipes et des développements

CONCLUSION

Les outils **d'intégration continue** mentionnés précédemment sont des **outils complets, documentés et intuitifs**. Tandis que **Jenkis** s'impose comme étant une **référence** de part sa date de parution, **GitLab CI/CD** et **GitHub Actions** semblent avoir réussis à **s'imposer comme des normes également**.

Etant donné le caractère du projet, il ne semblerait **pas pertinent** de se tourner vers **Jenkins** qui est un outil certes très complet mais qui doit être **hébergé**.

Bien que **GitLab CI/CD** soit une **solution qui pourrait convenir à nos besoins** étant donné notre étude, cette solution ne **serait pas la plus pertinente au vu de notre choix concernant le logiciel de gestion de versions.**

En effet, **GitHub Actions** étant propulsé par **GitHub**, il semble intéressant de **centraliser l'ensemble de nos outils, logiciels et méthodes autour d'un seul et**

même service d'hébergement afin de ne pas avoir à naviguer entre différents services et/ou logiciels.

BIBLIOGRAPHIE & WEBOGRAPHIE

TABLEAU DE BORD

Nous noterons l'utilisation d'une seule méthode de recherche, plus simple pour ce sujet et dans les délais donnés, **la méthode pull. Ainsi, nous sommes allés chercher l'information nous-même afin d'obtenir un ensemble de résultats pertinents le plus rapidement possible par rapport au sujet donné.**

Nom de la source	Lien	Commentaires	Mode de suivi	Auteur	Catégorie	Type de source	Fréquence de MAJ
bpesquet.gitbooks.io	https://bpesquet.gitbooks.io/genie-logiciel/content/chapters/06-gestion-versions.html	Mise en avant des avantages et inconvénients des logiciels de gestion de versions centralisés et distribués	Pull	-	Logiciels de gestion de versions	Site Web	-
Overblog	http://amine-benkirane.over-blog.com/2012/02/les-avantages-et-inconv%C3%A9nients-des-outils-de-gestion-de-version-distribu%C3%A9s-dvcs	Mise en avant des avantages des outils de gestion de version distribués	Pull	Benkirane	Logiciels de gestion de versions – distribués	Site web	Article publié le 21 Février 2012
Atlassian	https://www.atlassian.com/blog/software-teams/version-control-centralized-dvcs	Comparaison des logiciels de gestion de versions centralisés et distribués	Pull	Giancarlo Lionetti	Logiciels de gestion de versions	Site web	Article publié le 14 février 2012

Grand View Research	https://www.grandviewresearch.com/industry-analysis/version-control-system-market	Parts de marché aux Etats-Unis des logiciels de gestion de version centralisés et décentralisés	Pull	-	Logiciels de gestion de versions - Parts de marché	Site web	Article publié en aout 2020
PERFORCE	https://www.perforce.com/blog/vcs/git-vs-mercurial-how-are-they-different#:~:text=Git%20has%20more%20than%2080%25%20of%20market%20share.	Comparaison de Git et Mercurial	Pull	Chuck Gehman	Logiciels de gestion de versions – GIT & Mercurial	Site web	Article publié le 9 janvier 2019
CodeBuilder.fr	https://www.codebuilder.fr/blog/developpement/developpement-collaboratif-logiciels-gestion-versions/	Comparaison de Git et Mercurial	Pull	Steven Buttarazzi	Logiciels de gestion de versions distribués – GIT & Mercurial	Site web	Article publié le 27 février 2017
Intland Software	https://content.intland.com/blog/sdlc/why-is-git-better-than-mercurial	Comparaison de Git et Mercurial	Pull	Eva Johnson	Logiciels de gestion de versions distribués – GIT & Mercurial	Site web	Mis à jour le 29 janvier 2021

Trustradius	https://www.trustradius.com/products/git/reviews?qs=pros-and-cons	Avantages et inconvénients de Git	Pull	-	Logiciels de gestion de versions distribués – GIT	Site web	Dernier avis le 23 janvier 2020
CodeGiant	https://blog.codegiant.io/gitlab-vs-github-which-one-is-better-2020-d8ec7fb9542c	Avantages et inconvénients de GitHub et GitLab	Pull	Team CodeGiant	Service d'hébergement de referentiel Git – GitHub & GitLab	Site web	10 Avril 2020
Trustadius	https://www.trustradius.com/products/bitbucket/reviews?qs=pros-and-cons	Avantages et inconvénients de BitBucket	Pull	-	Service d'hébergement de referentiel Git – BitBucket	Site web	-
Slintel	https://www.slintel.com/tech/source-code-management/github-market-share#Github-faqs	Parts de marché des logiciels de gestion de versions distribués - GitHub	Pull	-	Service d'hébergement de referentiel Git - GitHub	Site web	Mise à jour en temps réel

Kinsta	https://kinsta.com/fr/base-de-connaissances/base-de-connaissances-github/	Description de GitHub	Pull	-	Service d'hébergement de referentiel Git - GitHub	Site web	Mise à jour le 25 août 2020
Wikipédia	https://fr.wikipedia.org/wiki/GitLab	Description de GitLab	Pull	-	Service d'hébergement de referentiel Git - GitLab	Site web	-
Wikipédia	https://fr.wikipedia.org/wiki/Bitbucket	Description de BitBucket	Pull	-	Service d'hébergement de referentiel Git - BitBucket	Site web	-
GitHub	https://fr.github.com/features/actions	Description de GitHub Actions	Pull	-	Outil d'intégration continue – GitHub Actions	Site web	-

Le Big Data	https://www.lebigdata.fr/jenkins-definition-avantages#:~:text=Jenkins%20est%20un%20outil%20logiciel,similaires%2C%20ainsi%20que%20son%20fonctionnement.	Description de Jenkins	Pull	Bastien L	Outil d'intégration continue – Jenkins	Site web	Article publié le 11 décembre 2017
DZone	https://dzone.com/articles/jenkins-vs-gitlab-ci-battle-of-cicd-tools	Jenkins VS GitLab CI/CD	Pull	Rahul Jain	Outil d'intégration continue – Jenkins & GitLab CI/CD	Site web	Article publié le 8 Octobre 2020