DevOPS-CI/CD

Bienvenue

Partie I

Généralités et Bonne pratiques DevOps

Introduction

L’on a le plus souvent entendu parler de DevOps sans vraiment savoir de quoi il s’agit.

Que l’on soit dévéloppeur ou Administrateur Système ou encore un simple curieux, ce cours sera utile.

Il s’agira pour nous de connaitre les fondements du DevOps, de présenter ses avantages et inconvénients et surtout comment bien l’implémenter.

L’objectif pour nous sera d’identifier les méthodes d’implémentation du DevOps en entreprise.

Origines de la méthodes DevOps

- La conference 2008 Agile Toronto conference

- Le mûr de confusion

Caractéristiques de DevOps

Les piliers du DevOps constituent l’acronyme C.A.L.M.S

- Culture
- Automatisation
- Lean
- Mesure
- Share

Culture

Comme nous avons vu précédement le DevOps est de manière générale une affaire de culture et de de collaboration entre les équipes Dev et Ops.

- Tout est basé sur la volonté des equipes de développment et des Opération de travailler ensemble.
- La culture constitue plus de la moitié de la méthodologie
- Il existe quelques petites habitudes à adopter dans un projet(pollinisation croisée)
- La culture des responsabilités partagées

Automatisation

Si la culture est établie, il faudrait mettre en place un ensemble d'outils et de processus afin de favoriser et de luidifier les déploiement ce qui permettra de livrer des produits de qualité, d'où **l'automatisation.**

Quelques avantages

- Déploiement plus fréquent
- Configuration des déploiement automatiquement
- Création des environnement à la demande: Infrastrure as code
- Tests plus fréquents

Lean

Dans une projet il est important d’éviter au maximum le gaspillage. Lean réduit au maximum le gaspillage. Quelsques types de gaspillage:

- surproduction ;
- attente ;
- transports ;
- étapes inutiles ;
- stocks ;
- mouvements inutiles ;
- corrections / retouches.

La chaine VSM(Value Stream Map) outil permettant de détecter les processus prennant du temps et d’apportant aucune valeur ajoutée

Mesure

Comme toute transformation d'entreprise, il est nécessaire d'avoir des **indicateurs de performance clés** (KPI ou Key Performance Indicator) afin de savoir si les efforts de transformation et d'amélioration continue transforment quelque chose.

Quelques indicateurs:

- Combien de temps la nouvelle fonctionnalité a pris pour **passer du développement à la production** ?
- Combien de fois un **bug récurrent** apparaît ?
- Combien de personnes **utilisent le produit** en temps réel ?
- Combien **d'utilisateurs** a-t-on **gagnés ou perdus** en une semaine ?

Sharing(Partage)

Le fameux **mur de la confusion** est largement dû à un manque de savoir commun. Mettre en place une politique de responsabilité et de succès partagés est déjà un premier pas vers le comblement de ce fossé entre les deux mondes. Le DevOps met en exergue le fait que la personne qui **code l'application** doit aussi la **livrer et la maintenir** en condition opérationnelle.

L’un ne doit pas absolument maîtriser le métier de l’autre mais au moins quelques notions de base.

Les développeurs et les opérations doivent se parler, partager les succès et les échec etc. Les relations humaines aussi sont importantes.

Mise en place de DevOps

Nous savons désormais ce qu'est le DevOps ainsi que ses Principaux Piliers. Maintenant il est question de savoir comment l'implémenter. Nous mettrons en exergue les notions de **patterns** et d'**anti-pattern**.

- Un ***pattern*** désigne une **bonne pratique** pour implémenter une méthode.
- Un ***anti-pattern*** est en quelque sorte un contre-exemple, **un piège à éviter** !

Les anti-patterns

Initialement il s’agirait de bonnes idées mais qui deviennent au fur et à mesure contre productives

- **Silo dev et silo ops**: Séparation des équipes Dev des Opérations(le mur de confusion)
- **Silo DevOps**: Créer une autre équipe qui va implémenter la livraison et les déploiement continus.
- **NoOps:** La volonté de laissé toute la maitrise au Dev. Cela deviendra de plus en plus difficile avec la montée en charge.
- Equipe transverse outils
- Ops dans Dev: Créer une petite cellule Ops dans l’équipe dev(En général un responsable qui a déjà d’autres priorités.

Les patterns

Initialement il s’agirait de bonnes idées mais qui deviennent au fur et à mesure contre productives.

- ***Collaboration entre Dev et Ops***: c’est la clé de la réussite de l’implémentation de la méthode DevOps
- ***Dev et équipe Cloud***: Une équipe qui s'occupe simplement de fournir **l'infrastructure** de déploiement. Ceci peut être une étape **intermédiaire** vers une implémentation plus complète du DevOps.
- ***Service externe***: faire appel à un **fournisseur externe d'expertise DevOps** peut s'avérer une bonne idée mais est souvent plus adapté aux petites équipes.
- ***Equipe DevOps temporaire***: Une équipe DevOps qui n'est présente que le temps de mettre en place les démarches DevOps et **former** les équipes Dev et Ops aux **bonnes pratiques**.
- ***Evangélistes DevOps***: Proche du pattern précédent, c'est une équipe qui s'occupera uniquement de **former les équipes** au DevOps.
- **Collaboration axée sur les conteneurs** : les conteneurs permettent de **fluidifier** le travail entre le développement et la mise en production.

Le métier de SRE

SRE: **Site Reliability Engineer**

Le SRE est apparu **en 2003 chez Google**, lorsque Ben Treynor fut embauché pour diriger une équipe de 7 développeurs, faisant tourner un environnement de production. Le but de cette équipe était de **garantir que les sites de Google fonctionneraient** de manière efficace et stable.

Depuis lors, plusieurs autres grosses structures ont déployé ce rôle de SRE

Le métier de SRE - Rôle

- il passe 50 % de son temps à faire des tâches **"ops"**, comme de l'astreinte, corriger des problèmes en production ou faire des interventions manuelles ;
- il passe les autres 50 % de son temps à **automatiser** tout ou partie de ses tâches, et à développer de nouvelles fonctionnalités.

La force du SRE reside dans le fait que le temps lui soit alloué pour automatiser les tâches répétitives.

SRE - Les piliers

Principalement basés sur l’approche CALMS

- réduire les **silos** organisationnels (**Share**)
- accepter les **pannes** comme normales (**Culture**)
- implémenter des **changements** graduels (**Lean**)
- s’appuyer sur des outils et
  de **l’automatisation** (**Automatisation**)
  tout **mesurer** (**Mesure)**

SRE - Les quatre signaux dorés

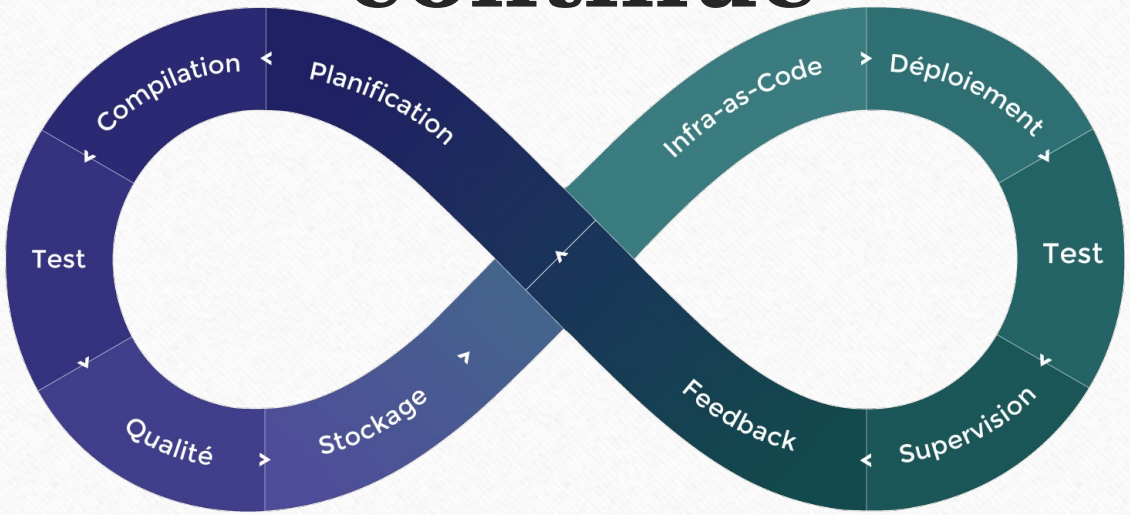
Le SRE définit quatre signaux principaux à observer constamment. Ces quatre signaux sont appelés les quatre **signaux dorés** : **latence**, **trafic**, **erreurs** et **saturation.**

- **Latence**: temps entre une demande et une réponse
- **Le trafic**: Nombre de demandes qui circulent sur le réseau
- **Les erreurs**: Des erreurs de configuration, des bugs de codes ou des dépendances non résolues
- **La saturation**: Il s'agit de la charge sur le réseau et les ressources du serveur

Google donne plus d'informations sur ce métier ici

PARTIE II(CI/CD)

Intégration continue et déploiement continue



Intégration continue: Définition

- **L’intégration continue** est un ensemble de pratiques utilisées en génie logiciel, consistant à vérifier, à chaque modification de code source que le résultat des modifications ne produit pas de régression dans l'application développée.

Intégration continue – Les étapes

Elle se fait sur 5 étapes

- **Planifier** le développement.
- **Compiler** et intégrer le code.
- **Tester** le code.
- Mesurez la **qualité** du code.
- Gérer les **livrables** de l’application

Etape 1: Planifier le développement

- Il est primordial de définir les tâche du projet afin d’attribuer à chaque membre sa charge de travail.
- Tous les membres absolument être impliqués.
- L’on peur se servir des Outils Comme Jira, Assana, Trello … qui implémentent pour la plupart la méthodes Agile.
- Ces outils permettent une meilleur collaboration entre collaborateurs

Etape2: Compiler et intégrer le code

Il est question ici de contrôler le code source et d'automatiser les étapes grâce à un Orchestrateur

- Contrôle de code source

Le code source doit être disponible à chaque moment sur un dépôt central. Chaque développement fera l'objet d'un suivi de revison. Bien que les branches existent il est mieux de travailler sur une seule branche pour éviter d'en suivre plusieurs à la fois. On peut utiliser des outils comme Git, GitLab,BitBucket …

- Orchestrateur: les étapes automatisées seront reproduites et la gestion des dépendances entre elles est facile. Cela permet de rendre tout disponible à chaque moment et avoir tout le temps la santé de ces dernières. Il servira aussi dans le déploiement contnue. Outils: GitLab CI, Azure DevOps …

Etape3: Tester le code

- Les tests unitaires: Permettent de vérifier le bon fonctionnement d’une partie précise de l’application.
- Après la compilation l’orchestrateur lance les tests unitaires.
- Ils doivent s’exécuter le plus rapidement possible afin d’avoir des feedback
- Ils servent à: trouver facilement les erreurs, sécuriser la maintenance et documenter le code
- On peut les implémenter avec des outils comme Junit, phpUnit ou encore Xunit
- Ils doivent être maintenus au fur et à mesure et s’exécuter à chaque modification

Etape 4: Mesurez la qualité de votre code

- Elle permet de s’assurer que le code sera maintenable et évolutif pendant son cycle de vie.
- Dans cette étape nous recherchons la plus petite **dette technique** *(date nécessaire à la correction d’un bug ou à l’ajout d’une fonctionnalité)* possible.
- Elle permet aussi de mesurer d’autres métriques: La vulnérabilité, la couverture des tests, code smell (mauvaises pratiques), la complexité cyclomatique, la duplication, etc.
- Si le code n’atteint pas la qualité requise avant déploiement l’on peut implémenter l’arrêt du pipeline CI.

La gestion des livrables

- L’objectif est d’obtenir des artefacts prêts à être déployer sur l’environnement de test ou en production
- Ils doivent aussi être disponibles
- Cela peut faire avec des outils comme GitLab repository, Docker Hub, Nexus ou autre

L’intégration continue

- Pour la suite nous allons prendre un projet Open source pour réaliser le pipeline
- Pas besoin de maîtriser le développement
- Nous choisirons l’environnement GitLab pour implémenter ces étapes.

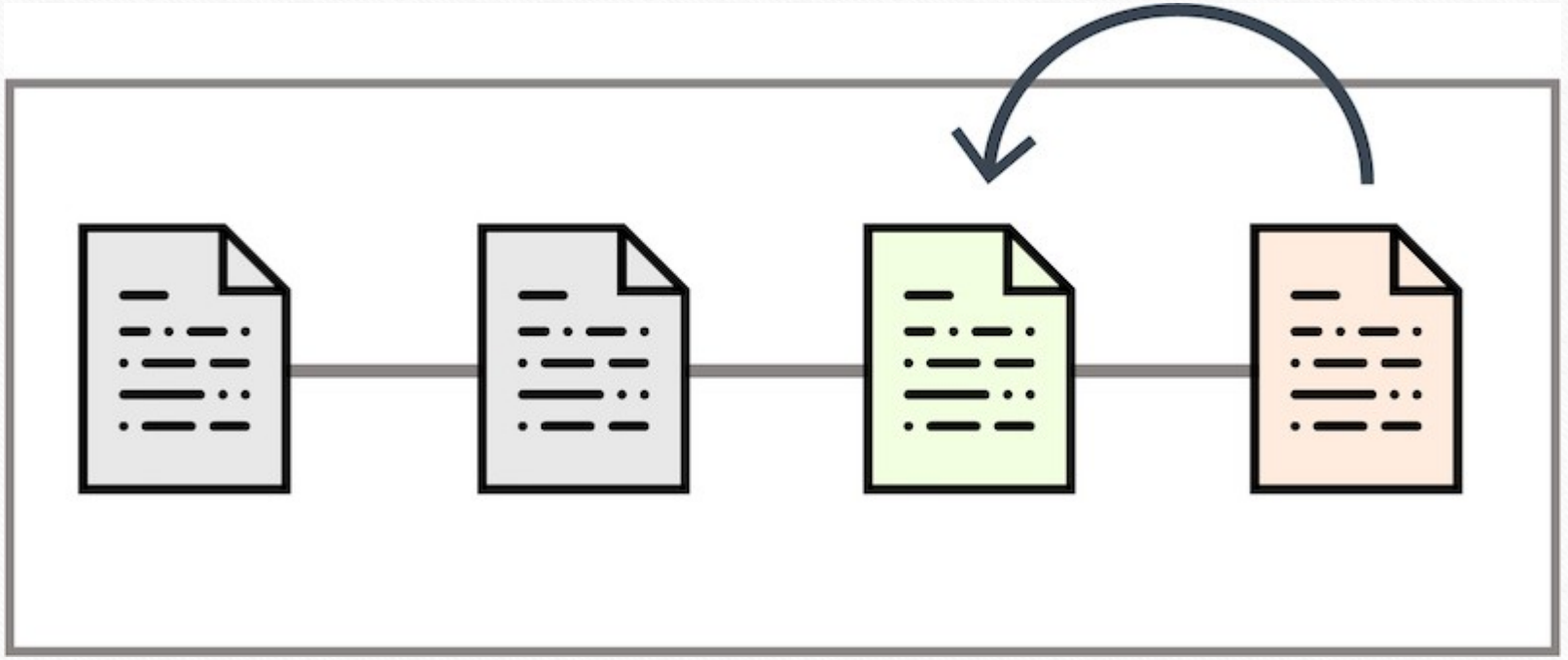
Les outils

- Pour gérer notre intégration continue nous allons nous servir de la plateforme Gitlab CI/CD
- Avant tout faisons un révision de git et ses commandes.

Git: Gestionnaire de version

- **Un gestionnaire de versions** est un programme qui permet aux développeurs de conserver un historique des modifications et des versions de tous leurs fichiers.
- L’action de contrôler les versions est aussi appelée "versioning" en anglais
- Il permet de garder en mémoire chaque modification des fichiers, pourquoi elle a eu lieu et par qui.
- Git est le système de versioning le plus utilisé aujourd'hui.

Git: Les fonctionnalités

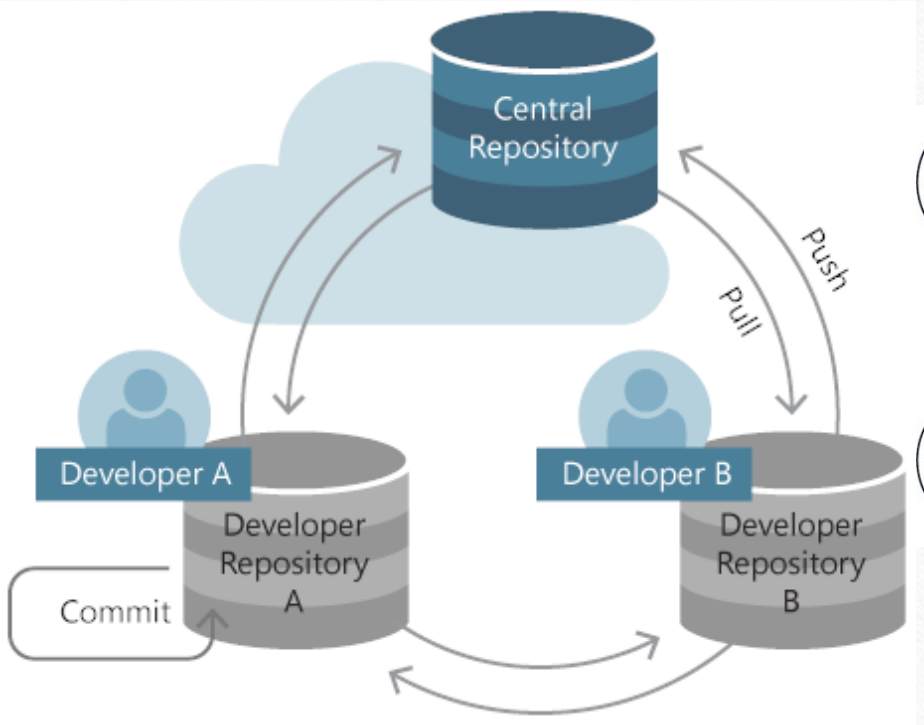


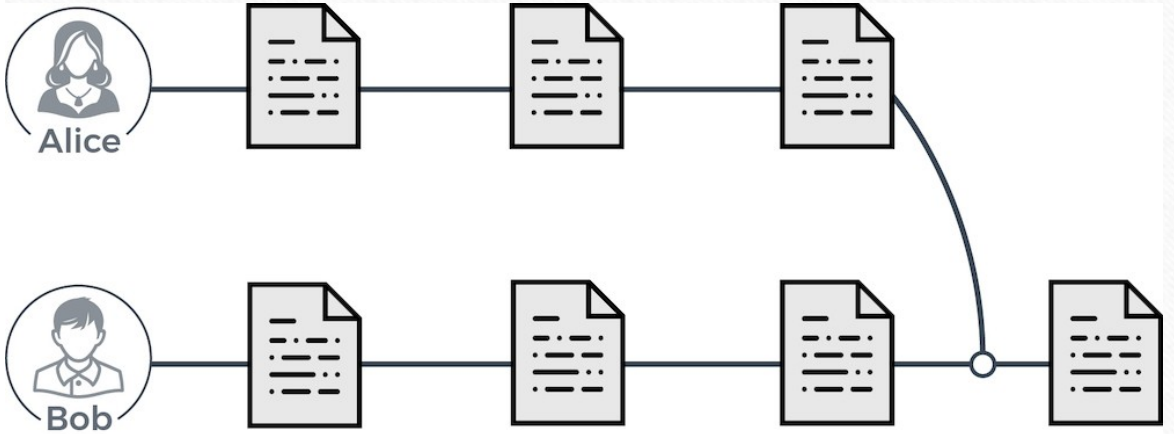
Git: Les fonctionnalités

On peut soit utiliser git seul pour suivre ses modifications soit en équipe pour suivre toutes les modifications de chaque membre et les fusionner sans risque de perte. Il a pour principale fonctionnalités:

- Revenir à une version précédente de votre code en cas de problème.
- Suivre l’évolution de votre code étape par étape.
- Travailler à plusieurs sans risquer de supprimer les modifications des autres collaborateurs.

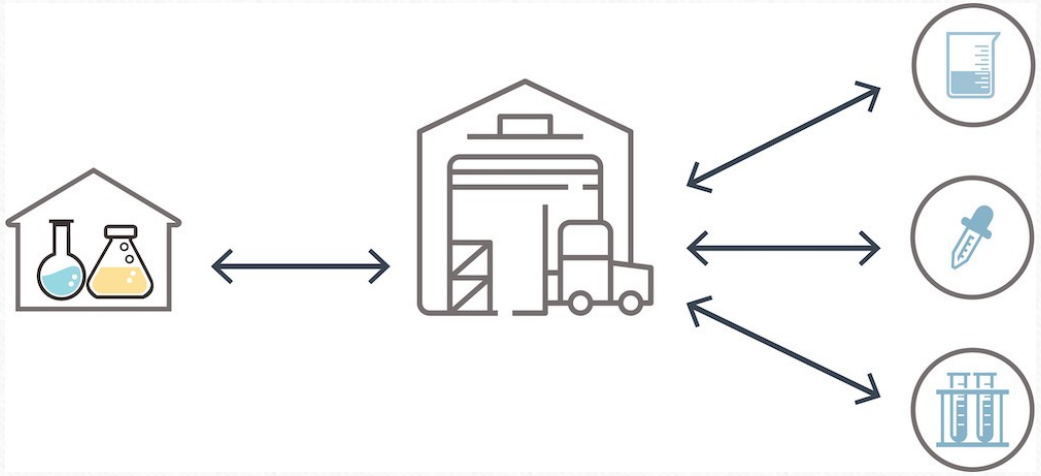
Git: Fonctionnement

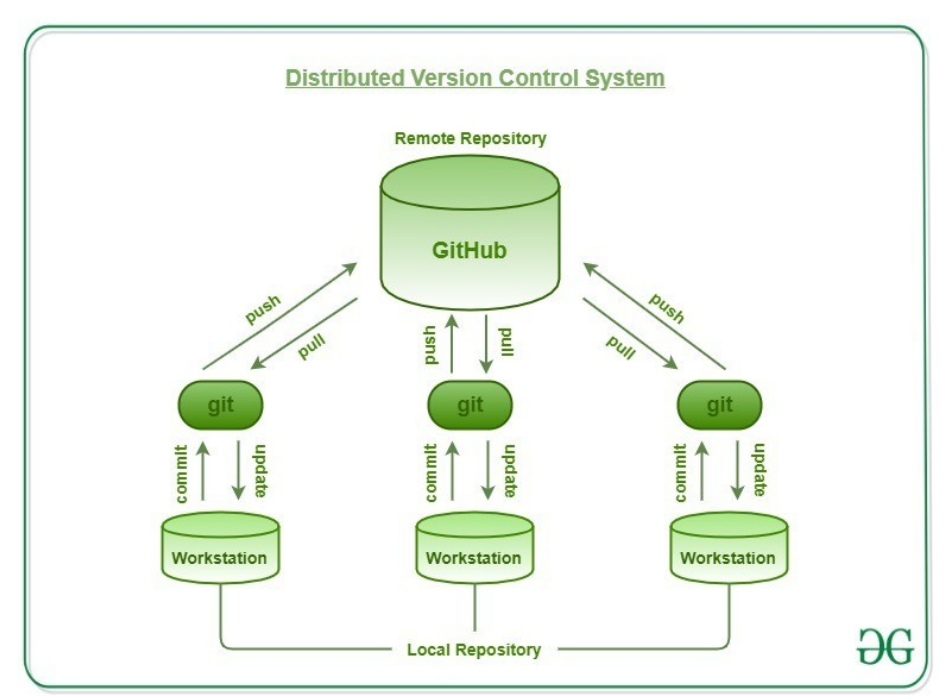




Git et dépôts distants

- Git fonctionne en local
- Un dépôt distant permet de stoker





Git: Installation

- Sur le site de git(git-scm.com) dans la rubrique download, téléchargez la version qui corresrpond la mieux à votre sy

Quelques commandes Git

git init

git add .

git commit

git branch

git checkout

git merge

TP: Git

Créer un dossier dans votre machine

Activer le suivi des modifications locales des fichiers dans ce dossier

Créez deux branches dev et stable

Dans la branche dev créer un fichier dev.txt qui contient un text de votre choix

Dans la branch stable créer un fichier stable.txt

Faites des commit de chaque branche avec des messages significatifs

A partir d’une commande git, rendez disponibles toutes les modifications des deux branches dans la branche master.

Git et GitLab

Après avoir fait toutes les action nécessaires en local, il est important de stocker le projet sur le serveur GitLab.
Quelques commandes

- Git clone
- Git remote
- Git push
- Git pull

TP: Liaison Git et GitLab

- Créer un projet sur git lab nommé « test-ligne »
- Lier le dossier local du TP1 au projet en ligne
- Faites les push nécessaires de toutes les branches
- Vérifiez sur gitLab les modifications

Pratique de l’intégration continue

- Nous allons ensemble implémenter l’intégration continue sur un projet différent. Cela se fera étape par étape
- Nous allons nous servir d’un projet OpenSource développé sur Java avec Sprint Boot

TP: Intégration continue