MGL870
Sujets spéciaux II en génie logiciel – DevOps

Permettre et pratiquer l'intégration continue

The DevOps Handbook

Part III, Chap 11

Objectifs d'apprentissage

- Expliquer les conséquences des branches à longue vie.
- Expliquer les deux classes de stratégies de branchement définie par Jeff Atwood.
- Expliquer en quoi consiste la pratique de développement basée sur le tronc. Quels sont ces avantages?

Sujets

- Introduction
- Branches à longue vie
- Pratique de développement basée sur le tronc
- Conclusion

- **Introduction**
- Branches à longue vie
- Pratique de développement basée sur le tronc
- Conclusion

Introduction

- But de supporter la «création des branches» dans les systèmes de contrôle de versions
  - Permettre aux développeurs de **travailler sur différentes parties du système logiciel en parallèle**,
  - **Éliminer les risques** pour les développeurs **de soumettre des modifications qui pourraient déstabiliser ou introduire des erreurs dans le tronc** (aussi appelé "master" ou ligne principale).

- **Plus les développeurs sont autorisés à travailler longtemps sur une branche, plus il est difficile d'intégrer et de fusionner** les modifications apportées par tous les utilisateurs.
- Les problèmes d'intégration entraînent un nombre considérable de retouches pour revenir à un état déployable.
  - E.g. modifications conflictuelles à fusionner manuellement ou des fusions qui interrompent nos tests automatisés ou manuels.
  - La résolution de ces problèmes nécessitent généralement l'implication de plusieurs développeurs.
- Cela provoque une autre spirale descendante: lorsque fusionner du code est douloureux, nous avons tendance à le faire moins souvent, ce qui rend la fusion future encore pire.

- **L'intégration continue a été conçue pour résoudre ce problème en faisant de la fusion dans le tronc une partie intégrante du travail quotidien de chacun.**

HP LaserJet Firmware

- Division HP LaserJet Firmware
  - Division qui a crée le "firmware" qui gère tous leurs scanners, imprimantes et périphériques multifonctions.
  - Dirigé par Gary Gruver
  - 400 développeurs (États-Unis, Brésil, Inde)

- Malgré la taille de l'équipe, ils progressaient beaucoup trop lentement.
  - Incapables de fournir de nouvelles fonctionnalités aussi rapidement que l'entreprise en avait besoin.

Exemple HP LaserJet Firmware

- Seulement 2 nouvelles versions de "firmware" par an
- Consacraient la majeure partie de leur temps à porter du code pour prendre en charge de nouveaux produits.
- Gruver a estimé que **5% seulement de leur temps était consacré à la création de nouvelles fonctionnalités**.
- Le reste du temps était consacré à des tâches non productives liées à leur dette technique, telles que la gestion de plusieurs branches de code et des tests manuels, comme indiqué ci-dessous:
  - 20% à faire une planification détaillée
    - Le faible débit et les délais d'exécution élevés ont été mal attribués à des erreurs d'estimation, et, dans l'espoir d'améliorer la situation, ils ont été invités à faire une estimation plus détaillée du travail.
  - 25% dépensé à faire le portage de code, tous maintenus dans des branches de code distinctes
  - 10% passé à faire l'intégration de leur code entre les branches de développement
  - 15% passé à faire des tests manuels

Exemple HP LaserJet Firmware

- **Objectif: multiplier par dix le temps consacré à l'innovation et aux nouvelles fonctionnalités**.
- L'équipe espérait que cet objectif pourrait être atteint par:
  - l'**intégration continue** et **développement basé sur le tronc**
  - un investissement important dans l'**automatisation des tests**
  - la **création d'un simulateur matériel** afin que les tests puissent être exécutés sur une plateforme virtuelle
  - la **reproduction des échecs de test sur les postes de travail des développeurs**
  - le **développement d'une nouvelle architecture** pour prendre en charge l'**exécution de toutes les imprimantes à partir d'une version commune**
- Auparavant, chaque ligne de produits nécessitait une nouvelle branche de code, chaque modèle possédant une version de "firmware" unique avec des fonctionnalités définies au moment de la compilation.
- La nouvelle architecture a permit à **tous les développeurs de travailler sur une base de code commune**, avec **une seule version de "firmware"**, prenant en charge tous les modèles LaserJet construits à partir du tronc, **les capacités d'imprimante étant établies au moment de l'exécution dans un fichier de configuration XML**.

Exemple HP LaserJet Firmware

- Quatre ans plus tard, ils possédaient une base de code prenant en charge les vingt-quatre gammes de produits HP LaserJet en cours de développement sur le tronc.
- Gruver admet que **le développement basé sur le tronc nécessite un grand changement de mentalité**.
  - Les ingénieurs pensaient que le développement basé sur le tronc ne fonctionnerait jamais, mais une fois qu'ils ont commencé, ils ne pouvaient plus imaginer revenir en arrière.
  - Au fil des années, plusieurs ingénieurs ont quitté HP.
  - Ils m'appelaient pour me dire à quel point le développement de leurs nouvelles entreprises était rétrograde, soulignant à quel point il était difficile d'être efficace et de publier un code correct sans le feedback de l'intégration continue.
- Cependant, **le développement basé sur le tronc les obligeait à mettre au point des tests automatisés plus efficaces**.
- Gruver a observé: **«Sans les tests automatisés, l'intégration continue est le moyen le plus rapide d'obtenir un gros tas de déchets qui ne sont jamais compilés ou qui ne fonctionnent pas correctement.»**
  - Au début, un cycle complet de tests manuels nécessitait six semaines.

Exemple HP LaserJet Firmware

- Afin de pouvoir tester automatiquement toutes les versions du firmware, ils ont **investi massivement dans le développement de simulateurs d'imprimantes** et ont **créé une batterie de tests** en six semaines
  - En quelques années, **2000 simulateurs d'imprimante** fonctionnaient sur six racks de serveurs chargés de charger les versions de microprogrammes à partir de leur pipeline de déploiement.
  - Le système d'intégration continue (CI) exécutait l'ensemble des tests automatisés d'unité, d'acceptation et d'intégration sur des versions construites à partir du tronc, comme décrit dans le chapitre précédent.
  - Ils ont **créé une culture qui arrêtait tout travail à chaque fois qu'un développeur cassait le pipeline de déploiement**, garantissant que ces derniers ramenaient rapidement le système à un état vert.
- Les tests automatisés ont créé un retour rapide qui a permis aux développeurs de confirmer rapidement que leur code engagé fonctionnait réellement.
  - Les tests unitaires s'exécutaient sur leurs stations de travail en quelques minutes.
  - **Trois niveaux de tests automatisés étaient exécutés à chaque commit**, ainsi que toutes les deux et quatre heures.
  - Le **test de régression complet final se déroulerait toutes les vingt-quatre heures**.

Exemple HP LaserJet Firmware

- Au cours de ce processus, ils ont:
  - réduit le nombre de builds à un build par jour, éventuellement de **10 à 15 builds par jour**
  - **passé d'une vingtaine de commits par jour** effectués par un «**chef de projet**» à **plus de cent commits par jour par** des **développeurs individuels**
  - permis aux développeurs de **modifier** ou d'**ajouter 75,000 à 100,000 lignes de code chaque jour**
  - **réduit le temps de test de régression** de **six semaines à un jour**

- Ce niveau de productivité n'aurait jamais pu être supporté avant l'adoption de l'intégration continue, lorsque la simple création d'une construction écologique a nécessitait plusieurs jours d'héroïsme.

Exemple HP LaserJet Firmware

- Les avantages commerciaux en résultant ont été étonnants:
  - Le **temps consacré à l'innovation et à la rédaction de nouvelles fonctionnalités** est passé **de 5% du temps de développement à 40%.**
  - Les **coûts de développement globaux** ont été **réduits d'environ 40%.**
  - Les **programmes en développement** ont été **augmentés d'environ 140%**.
  - Les **coûts de développement par programme** ont été **réduits de 78%.**

- L'expérience de Gruver montre que, **après l'utilisation complète du contrôle de version**, **l'intégration continue est l'une des pratiques les plus critiques permettant la fluidité du travail dans notre flux de valeur**, permettant à de nombreuses équipes de développement de développer, de tester et de générer de la valeur.
  - Néanmoins, l'intégration continue **reste une pratique controversée**.
  - La suite de ce chapitre décrit les pratiques nécessaires à la mise en œuvre de l'intégration continue, ainsi que les moyens de surmonter les objections courantes.

- Introduction
- **Branches à longue vie**
- Pratique de développement basée sur le tronc
- Conclusion

Conséquences des branches à longue vie

- Que se passe-t-il lorsque nous commettons rarement du code dans trunk
  - Chaque fois que des modifications sont apportées au contrôle de version, entraînant l'échec de notre pipeline de déploiement, nous essaimons rapidement le problème pour le résoudre, ramenant ainsi notre pipeline de déploiement à un état vert.
  - Cependant, des problèmes importants résultent du fait que les développeurs travaillent dans des branches privées à vie longue (également appelées «branches de fonctions»), ne fusionnant que de manière sporadique dans le tronc.
    - Entraîne un nombre important de modifications par lots.
    - Résulte un chaos important et des retouches afin de placer le code dans un état pouvant être livré.

Conséquences des branches à longue vie

- Il existe de nombreuses stratégies de branchement, elles peuvent toutes être classées dans le spectre suivant:
  - **Optimiser pour la productivité individuelle**:
    - Chaque personne participant au projet travaille dans sa propre branche privée.
    - Chacun travaille de manière indépendante et personne ne peut perturber le travail de quelqu'un d'autre; cependant, la fusion devient un cauchemar.
    - La collaboration devient très difficile - le travail de chaque personne doit être minutieusement fusionné avec le travail de tous les autres, même la plus petite partie du système.
  - **Optimiser pour la productivité des équipes**:
    - Tout le monde travaille dans le même espace commun.
    - Il n'y a pas de branches, juste une longue ligne droite de développement ininterrompue.
    - Les commits sont simples, mais chaque commit peut créer des problèmes dans l'ensemble du projet et interrompre tout progrès.

Conséquences des branches à longue vie

- L'observation d'Atwood est tout à fait correcte - plus précisément, **l'effort requis pour réussir la fusion des branches s'accroît de manière exponentielle à mesure que le nombre de branches augmente.**
  - Le **problème** ne réside pas seulement dans le remaniement créé par cette **«fusion d'enfer»**, mais également dans les **rétroactions (feedbacks) tardifs** que nous recevons de notre pipeline de déploiement.
  - À mesure que nous augmentons le taux de production de code et le nombre de nouveaux développeurs, nous augmentons la probabilité qu'un changement donné affecte une autre personne et que nous augmentions le nombre de développeurs qui seront touchés lorsque quelqu'un casse le pipeline de déploiement.
- Autre effet secondaire troublant lié à la fusion de lots de grande taille:
  - Lorsque la fusion est difficile, nous sommes moins en mesure et moins motivés d'améliorer et refactoriser notre code, car les refactorisations sont plus susceptibles de provoquer des problèmes pour tout le monde.
  - Lorsque cela se produit, nous sommes plus réticents à modifier du code comportant des dépendances dans toute la base de code, qui est (tragiquement) l'endroit où les gains peuvent être les plus élevés.

- Introduction
- Branches à longue vie
- **Pratique de développement basée sur le tronc**
- Conclusion

Pratique de développement basée sur le tronc

- **Solution** : **mettre en place des pratiques d'intégration continue et de développement basées sur le tronc**
  - **Tous les développeurs enregistrent leur code dans le tronc au moins une fois par jour.**
  - **Permet de réduire la taille des lots de travail.**
- **Permet d'exécuter tous les tests automatisés** sur notre système logiciel dans son ensemble et recevoir des alertes lorsqu'un changement crée un problème.
  - La détection des problèmes de fusion lorsqu'ils sont petits, permet de les corriger plus rapidement.
- Configurer notre pipeline de déploiement pour rejeter les validations qui nous font sortir d'un état déployable.
  - "gated validation" -- le pipeline de déploiement confirme d'abord que la modification soumise fusionnera et passera tous les tests automatisés avant d'être fusionnée dans le tronc.
  - Dans le cas contraire, le développeur en sera informé, ce qui permettra d'apporter des corrections sans affecter les autres utilisateurs du flux de valeur.
- La discipline des codes quotidiens **nous oblige à diviser notre travail en petits morceaux tout en maintenant le tronc dans un état fonctionnel et libérable.**
  - Le contrôle de versions devient un mécanisme intégral de la façon dont l'équipe communique.

Pratique de développement basée sur le tronc

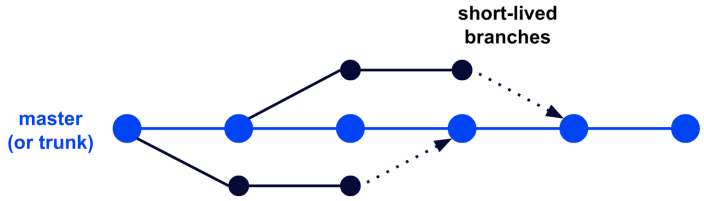
- Ayant ces pratiques en place, nous pouvons à nouveau modifier notre définition de «tâche complété (done)» (ajout en caractères gras):

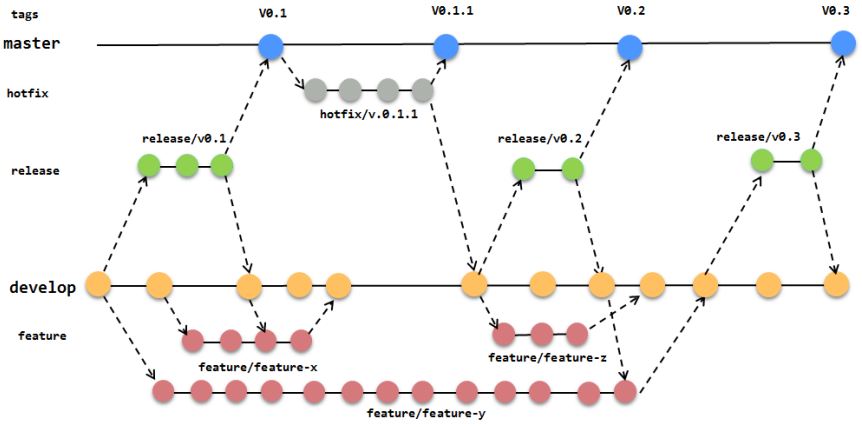
  «À la fin de chaque intervalle de développement, nous devons avoir un code intégré, testé, fonctionnel et potentiellement livrable, démontré dans un environnement de type production, **créé à partir du tronc en utilisant un processus en un clic (one-clic process), et validé avec des tests automatisés**. ″

- Adhérer à cette définition révisée de «tâche complété (done)» nous **aide à garantir la testabilité et la déployabilité continues du code** que nous produisons.
- En maintenant que notre code est dans un état déployable, nous sommes en mesure d'éliminer la pratique courante consistant à avoir une phase de test et une phase de stabilisation séparées à la fin du projet.

Pratique de développement basée sur le tronc

- **Trunk-based development**






bazaarvoice™

Exemple Bazaarvoice

- Bazaarvoice
  - Bazaarvoice fournit des contenus générés par les clients (revues, évaluations, par exemple) à des milliers de détaillants, tels que Best Buy, Nike, etc. et Walmart.
  - Ernest Mueller
    - A contribué à la transformation de DevOps chez National Instruments
    - A par la suite transformé les processus de développement et de publication chez Bazaarvoice en 2012.

- En 2012, Bazaarvoice affichait alors un chiffre d'affaires de 120 millions de dollars et se préparait à une introduction en bourse (IPO)
  - L'activité reposait principalement sur l'application Bazaarvoice Conversations, une application monolithique en Java composée de près de cinq millions de lignes de code datant de 2006 et couvrant quinze mille fichiers.
  - Le service fonctionnait sur 1,200 serveurs répartis dans quatre centres de données et plusieurs fournisseurs de services cloud.

Exemple Bazaarvoice

- En partie à cause du passage à un processus de développement Agile et à des intervalles de développement de deux semaines, il y avait un désir énorme d'augmenter la fréquence de publication par rapport au calendrier de production actuel de dix semaines.
  - Ils avaient également commencé à découpler des parties de leur application monolithique, la décomposant en microservices.
- Leur première tentative de calendrier de sortie de deux semaines a eu lieu en janvier 2012.
  - Selon Mueller:

    «Ça n'a pas bien marché. Cela a provoqué un chaos massif, avec quarante-quatre incidents de production enregistrés par nos clients.

    La principale réaction de la direction a été fondamentalement la suivante: «Ne refaisons plus jamais ça».

Exemple Bazaarvoice

- Mueller a repris les processus de publication peu de temps après, dans le but de réaliser des publications toutes les deux semaines sans causer de temps mort aux clients.
- Les objectifs commerciaux de publication plus fréquents incluaient notamment des tests A/B plus rapides (chapitre 17) et une augmentation du flux de fonctionnalités dans la production.
- Mueller a identifié **trois problèmes fondamentaux**:
  - **L'absence d'automatisation des tests** rendait tout niveau de test insuffisant pendant les deux semaines pour prévenir les défaillances à grande échelle.
  - La **stratégie de branchement du contrôle de versions** permettait aux développeurs d'archiver le nouveau code jusqu'au moment de la production du release.
  - **Les équipes qui développaient des microservices produisait également des versions ("release") indépendantes**, qui posaient souvent des problèmes lors de la publication de monolithes ou inversement.

Exemple Bazaarvoice

- Mueller a conclu que le processus de déploiement de l'application monolithique Conversations devait être stabilisé, ce qui nécessitait une intégration continue.
- Au cours des six semaines qui ont suivi, **les développeurs ont cessé de travailler sur les fonctionnalités pour se consacrer plutôt à l'écriture de suites de tests automatisées**.
  - Notamment des tests unitaires dans JUnit, des tests de régression dans Selenium et à l'exécution d'un pipeline de déploiement dans TeamCity.

- « **En effectuant ces tests tout le temps, nous pensions pouvoir apporter des modifications avec un certain niveau de sécurité.**

  Et plus important encore, nous pourrions immédiatement détecter le cas où une personne aurait cassé quelque chose, au lieu de le découvrir seulement après sa production. »

Exemple Bazaarvoice

- Ils ont également changé pour un modèle de versions basé sur le tronc/branche, où toutes les deux semaines, ils créaient une nouvelle branche de publication dédiée, aucun nouveau commit n'étant autorisé pour cette branche sauf en cas d'urgence.
  - Toutes les modifications étaient traitées via un processus de validation, par ticket ou par équipe via leur wiki interne.
  - Cette branche passait par un processus d'assurance qualité, qui était ensuite promu dans la production.
- Les améliorations apportées à la prévisibilité et à la qualité des versions ont été surprenantes:
  - Version de janvier 2012: quarante-quatre incidents clients (le processus d'intégration continue commence)
  - Version du 6 mars 2012: cinq jours de retard, cinq incidents clients
  - Version du 22 mars 2012: à l'heure, un incident client
  - Version du 5 avril 2012: à l'heure, zéro incident client

Exemple Bazaarvoice

- Mueller a ensuite décrit le succès de cet effort:

  Nous avons eu un tel succès avec des versions toutes les deux semaines.

  Nous **sommes passés à des versions hebdomadaires**, ce qui n'a nécessité pratiquement aucun changement de la part des équipes techniques.

  Étant donné que les publications ("releases") sont devenues si routinières, il s'agissait simplement de doubler le nombre de publications du calendrier et de les publier lorsque le calendrier nous le demandait.

  Sérieusement, c'était presque un non-événement.

  La majorité des changements nécessaires ont été apportés à nos équipes de service à la clientèle et de marketing, qui ont dû modifier leurs processus, par exemple en modifiant le calendrier de leurs courriels hebdomadaires afin de s'assurer que les clients étaient au courant des changements de fonctionnalités.

- Après cela, nous avons commencé à travailler vers nos prochains objectifs,

  ce qui a finalement conduit à accélérer nos tests de moins de trois heures à moins d'une heure,

  réduisant ainsi le nombre d'environnements de quatre à trois (développement, test, production, en supprimant l'environnement de "staging"), et

  passer à un modèle de publication continue complet dans lequel nous autorisons des déploiements rapides en un clic.

- Introduction
- Branches à longue vie
- Pratique de développement basée sur le tronc
- **Conclusion**

Conclusion

- Le développement basé sur le tronc est probablement la pratique la plus controversée discutée dans ce livre.
- De nombreux ingénieurs ne croient pas que cela est possible, même ceux qui préfèrent travailler sans interruption dans une branche privée sans avoir à traiter avec d'autres développeurs.
- Les données du **Puppet Labs 2015 State of DevOps Report** sont claires: **le développement basé sur le tronc prédit un débit plus élevé et une meilleure stabilité**, et **même une plus grande satisfaction au travail et des taux de surmenage plus faibles**.
- Bien que convaincre les développeurs puisse être difficile au début, une fois qu'ils auront perçu des avantages extraordinaires, ils deviendront probablement des convertisseurs à vie, exemples HP LaserJet et Bazaarvoice.
- Les pratiques d'intégration continue préparent la prochaine étape, à savoir l'automatisation du processus de déploiement et l'activation de versions à faible risque.