DAUPHINE UNIVERSITÉ PARIS | PSL

MATHÉMATIQUES ET INFORMATIQUE
DE LA DÉCISION ET DES ORGANISATIONS

afia CFA

MÉMOIRE

Année universitaire 2017 / 2018

NOM et Prénom de l'étudiant : Francisco Javier Martínez Lago

Formation : MIAGE

Entreprise d'accueil : Natixis

Titre du mémoire : Les enjeux de la mise en place de l'intégration continue dans des projets de développement existants

Tuteur enseignant : Florian Sikora

Maître d'apprentissage : Françoisse Boisson

Résumé (10 lignes) :

Ce document est composé de quatre sections. Tout d'abord, je parle sur ma mission et mes motivations. Ensuite, je donne un aperçu général de ce qu'est l'intégration continue et parlerai de certaines des technologies les plus connues dans ce domaine. Troisièmement, j'aborde les différents aspects qui doivent être pris en compte lors de la mise en place de l'IC dans un projet. Finalement, je parle de la façon dont l'IC est gérée dans l'un des projets de l'entreprise où j'ai travaillé.

MASTER 1 MIAGE

LES ENJEUX DE LA MISE EN PLACE DE L'INTÉGRATION CONTINUE DANS DES PROJETS DE DÉVELOPPEMENT EXISTANTS

28 août 2018

Francisco Javier Martínez Lago
Université Paris-Dauphine

Table des matières

Introduction

La maximisation de la productivité est un problème omniprésent dans l'entreprise. La productivité peut permettre à une entreprise de réussir alors que son concurrent fait faillite. Il peut faire la différence entre le respect d'une échéance et la livraison tardive d'un projet. Il n'est pas étonnant qu'il existe de nombreux articles et documents de recherche consacrés à ce sujet.

Dans le développement de logiciels, la productivité est recherchée par l'automatisation. L'élimination des processus répétitifs et manuels minimise les erreurs humaines et permet aux développeurs de concentrer leur énergie sur des tâches plus importantes. L'automatisation est souvent obtenue par la mise en œuvre de l'intégration continue (IC).

Il y a deux cas où nous pouvons nous poser la question de savoir s'il faut mettre en œuvre l'IC dans notre projet. D'abord, quand nous commençons un nouveau projet. Deuxièmement, lorsque nous avons un projet existant que nous souhaitons rendre plus efficace par l'automatisation. Dans le présent document, nous examinerons le deuxième cas, en particulier les difficultés d'une telle entreprise et quelques exemples de mise en œuvre dans le monde réel.

Ce document est composé de quatre chapitres. Tout d'abord, je parlerai sur ma mission et mes motivations. Ensuite, je donnerai un aperçu général de ce qu'est l'IC et parlerai de certaines des technologies les plus connues dans ce domaine. Troisièmement, je vais aborder les différents aspects qui doivent être pris en compte lors de la mise en place de l'IC dans un projet. Finalement, je parlerai de la façon dont l'IC est gérée dans l'un des projets de l'entreprise où j'ai travaillé.

CHAPITRE 1

La Mission

Dans ce chapitre, je donnerai brièvement des détails sur le contexte dans lequel ce mémoire a été écrit et sur mes motivations.

1.1 L'entreprise

Natixis est une banque de financement, de gestion et de services financiers, créée en 2006, filiale du groupe BPCE [1]. Elle est une banque internationale qui compte plus de 21 000 experts présents dans 38 pays [2]. Elle propose des expertises autour des thèmes suivantes :

- Gestion d'actifs et de fortune
- Banque de grande clientèle
- Assurance
- Services financiers spécialisés

Ils développent leurs activités principalement dans trois zones géographiques :

- Amérique
- Asie-Pacifique
- EMOA (Europe, Moyen-Orient, Afrique)

1.2 La mission

J'ai travaillé au sein de la Direction des opérations et des systèmes d'information. Parmi mes responsabilités, il y avait l'intégration d'un outil de reporting open-source avec le serveur basé sur Java et JSP existant de l'entreprise ainsi que la mise en place d'un tableau de bord pour afficher différents graphiques.

Au cours de la mission, j'ai dû communiquer avec des fournisseurs externes, écrire de la documentation, coder en Java et JavaScript et créer des requêtes SQL pour une base de données Oracle.

1.3 Les motivations et la démarche

Même si ma mission n'impliquait pas directement de processus d'IC, j'ai eu l'idée d'étudier ce sujet à partir de mon expérience précédente dans l'entreprise où j'ai travaillé l'année dernière. Là-bas, j'ai été directement impliqué dans la mise en place d'un workflow d'intégration continue. Maintenant que j'étais dans une entreprise de plus grande taille, j'étais curieux de savoir comment on traitait ici l'IC du point de vue de logiciels, de processus et de mise à l'échelle.

J'ai pris contact avec les employés d'autres secteurs de l'entreprise et j'ai parlé à des personnes chargées de créer des workflows d'intégration continue que d'autres départements pourraient ensuite mettre en œuvre. Je les ai interviewés avec une série de questions qui m'ont donné une meilleure perspective de l'architecture en usage. À partir de ces entretiens ainsi que de quelques informations librement disponibles en ligne, j'ai pu créer une image que je partagerai dans les chapitres suivants.

CHAPITRE 2

L'intégration continue

Dans ce chapitre, je vais parler de ce qu'est l'IC et de comment elle a évoluée depuis sa conception. Ensuite, je vais montrer des plateformes populaires qui permettent de mettre en place l'IC.

2.1 Principe

Dans les premiers jours du développement de logiciels, le travail des développeurs était basé sur le monde du hardware où il était important de corriger les bogues avant de sortir un produit [3]. En effet, si vous deviez produire quelque chose en masse, vous feriez mieux de vous assurer que vous avez obtenu le bon produit *avant* production. Ce modèle de développement logiciel centré sur le hardware a été incarné dans un article de Royce (1970), dans lequel il propose une approche séquentielle dans la création de programmes qui comprend plusieurs phases. Chaque phase doit être complétée dans son intégralité avant de passer à la phase suivante, un peu comme si l'eau coule sans effort le long d'une chute d'eau (modèle Waterfall). Bien sûr, tout comme une chute d'eau, il est difficile et coûteux d'essayer de remonter. Par conséquent, le modèle est basé sur l'idée que chaque phase doit être bien faite pour qu'il n'y ait pas besoin de revenir aux phases précédentes. Pour cette raison, de longues périodes sont consacrées à la création de la documentation et des spécifications. Lorsque le produit final arrive, il correspond parfaitement aux exigences originales, mais il se peut qu'il ne soit plus pertinent.

Dans un monde où les besoins évoluent, l'approche rigide par phases du modèle Waterfall ne pouvait pas rester pertinente. De nouvelles méthodologies de développement sont apparues pour présenter leur solution au problème d'essayer d'être plus dynamique. Le monde a vu l'essor du Rapid Application Development, du Dynamic Systems Development, du Scrum et d'Extreme Programming, entre autres. L'air du temps de ces années était un désir de vitesse et une meilleure capacité d'évolution et d'adaptation. Un exemple des pratiques mises en avant à cette époque était l'accent mis sur les courtes boucles de rétroaction. Plus un programmeur reçoit rapidement de la rétroaction sur un code qu'il écrit, plus il a de chances de faire la bonne chose.

Lorsque les principaux experts de ces méthodologies se sont réunis en Utah en 2001, ils ont créé le manifeste Agile. Il s'agissait d'une nouvelle approche audacieuse en matière de développement de logiciels. Il a encouragé les développeurs à se concentrer sur le software qui marche et à réagir au changement plutôt que de s'appuyer sur une documentation complète et de suivre un plan. Il se trouve que l'IC est naturellement liée à cette nouvelle façon de faire les choses.

2.2 Qu'est-ce que l'intégration continue?

Le terme "Intégration Continue" a été utilisé à l'origine par Grady Booch (1991) pour décrire une technique de développement de logiciels orientés objet qui impliquait que chaque développeur fasse des intégrations régulières à une branche principale et intègre leur code avec celui des autres développeurs plutôt qu'il n'y ait qu'un seul événement d'intégration "big-bang"[4]. Cette stratégie avait pour but de prévenir les problèmes d'intégration et d'aider à trouver les problèmes au début du processus de développement. Le terme a ensuite été développé dans la méthodologie Extreme Programming qui a introduit le concept d'intégration de plusieurs jours par jour [5]. Plus tard, Martin Fowler a publié un article sur son site web [6] qui décrivait l'approche en détail et suggérait les meilleures pratiques telles que la maintenance d'un référentiel unique, l'automatisation de la compilation et la réparation immédiate des compilations cassées.

Si nous considérons tous ces changements au concept original d'intégration continue, nous pouvons dire que c'est maintenant compris comme étant le processus d'intégration régulière dans une branche principale et le build automatique, le test et la validation du code à chaque fois que cela se produit.

L'IC est la solution au problème des développeurs travaillant de manière isolée.. Quand ils prennent trop de temps pour intégrer leurs changements dans la branche principale, il peut y avoir des problèmes tels que des conflits de fusion de code, des stratégies de code divergentes, des efforts en double et des bogues. Comme l'IC exige que l'équipe intègre continuellement ses changements à une branche partagée, cela permet d'atténuer ces problèmes.[7].

2.3 L'intégration continue dans l'entreprise aujourd'hui

Il y a des défis associés au passage à des pratiques continues dans l'entreprise [8]. Parmi les défis les plus courants dans l'adoption de pratiques d'intégration continue, il y a le manque d'outils et de technologies appropriés, la résistance au changement, le scepticisme quant à la valeur ajoutée des pratiques continues, l'absence d'une stratégie de test appropriée, la mauvaise qualité des tests et la résolution des conflits dans le code[9].

Cependant, l'IC a bénéficié de taux d'adoption élevés. Une enquête menée par Ambysoft [10] auprès de 337 développeurs agiles a posé la question de savoir dans quelle mesure la pratique de l'intégration continue était courante dans leur projet de développement logiciel. 58,6% d'entre eux ont dit très fréquemment ou fréquemment.

En plus, les résultats du sondage de recherche Evans des professionnels du développement logiciel [11] montrent que 65% des développeurs de logiciels, des managers et des cadres rapportent que leurs organisations ont commencé à adopter l'IC.

Les grandes entreprises comme Google [12] ou Netflix [13] utilisent également l'intégration continue.

En résumé, l'IC continue reste populaire parmi les entreprises malgré les défis potentiels liés à son adoption.

2.4 Les plateformes d'intégration continue

Parce qu'une seule solution ne convient pas à tous, de multiples plates-formes d'intégration continue ont été créées pour répondre aux différents besoins des entreprises. Dans cette section, nous verrons quelques-unes des options les plus populaires.

Jenkins

Jenkins est une application d'IC open-source qui permet de construire et de tester en continu des projets logiciels.[14]. C'est l'un des outils d'intégration continue le plus populaire [15] et il y a plus d'un millier de plugins qui étendent ses fonctionnalités de base. Jenkins fonctionne comme un serveur Java autonome et nécessite donc l'environnement d'exécution Java. Parce qu'il est écrit en Java, il est capable de fonctionner sur n'importe quel système d'exploitation où Java fonctionne. En outre, il prend en charge plusieurs systèmes de contrôle de version prêts à l'emploi, des projets Maven et Ant ainsi que des scripts shell, des commandes batch de Windows et bien d'autres. En outre, il prend en charge l'envoi de notifications par courrier électronique.

Jenkins a été créé par Kohsuke Kawaguchi en 2004 et s'appelait initialement Hudson, mais a été rebaptisé Jenkins en 2011 en raison de différends avec Oracle[16].

Un workflow de base de Jenkins pourrait ressembler à ceci :

1. Un développeur soumet son code à un référentiel de contrôle de version.
2. Jenkins prend les changements et construit et teste la nouvelle version.
3. Si la compilation échoue, Jenkins envoie des notifications aux développeurs.

L'état du build sera disponible sur le tableau de bord Jenkins.

Pour commencer avec Jenkins, un job doit être créé. Un job se compose de trois aspects :

- **Localisation des fichiers à construire**. Il peut s'agir d'un dossier local ou d'un référentiel de contrôle de version.
- **Un ensemble d'instructions à suivre**. Par exemple, l'exécution d'un script ou l'exécution d'une commande batch de Windows.
- **Un déclencheur**. C'est à cette condition que Jenkins commencera à faire le build. Les options principales sont basées sur le temps (ex : toutes les 20 minutes) et sur les événements (ex : chaque fois qu'il y a un commit sur la branche principale). Il peut aussi être plus avancé, comme lier l'exécution d'un build à la réussite d'un autre.

L'interface de Jenkins est accessible par le biais d'un navigateur web. L'interface principale de Jenkins est montrée dans la Figure 2.1.

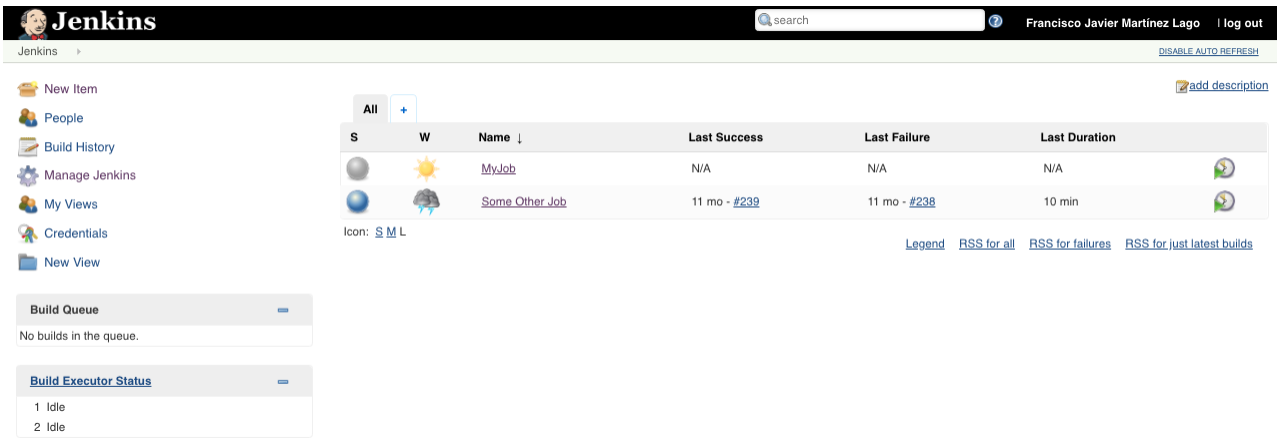

Figure 2.1 – Interface principale de Jenkins

L'une des principales forces de Jenkins est sa polyvalence. Il y a plus d'un millier de plugins disponibles pour étendre ses fonctionnalités. De plus, puisqu'il s'agit d'une application open-source, n'importe qui peut écrire des plugins pour cette application. C'est un avantage énorme pour les entreprises, qui doivent souvent se conformer à des réglementations spécifiques et sont donc intéressées par le développement de solutions personnalisées. En outre, Jenkins est gratuit.

Parmi les aspects négatifs de Jenkins signalés par des utilisateurs [17] , on peut citer les suivants : une courbe d'apprentissage abrupte, une interface utilisateur difficile à utiliser et parfois lente, il dépend trop des plugins et il n'y a pas de moyen facile de chercher dans les logs de toutes les compilations en même temps.

Travis-CI

Travis-CI a été créé en 2011 par la société CB Rank. C'est un service d'IC, distribué et hébergé, utilisé pour construire et tester des projets logiciels hébergés chez GitHub. Travis-CI surveille en permanence les changements dans un référentiel de contrôle de version, construit et teste automatiquement les changements de code et fournit une rétroaction immédiate sur le succès ou l'échec du build [18]. Les builds peuvent être personnalisés en utilisant un fichier spécial appelé « .travis.yml » qui doit être inclus dans le dossier racine du projet.

Un workflow Travis-CI de base peut ressembler à ceci :

1. Un développeur soumet son code
2. Travis-CI recueille les modifications et lit le fichier travis.yml pour prendre en compte les paramètres spécifiques au projet.
3. Travis-CI clone le référentiel de contrôle de version dans un nouvel environnement virtuel et construit et teste le code.

4. Les notifications sur le résultat de la compilation sont envoyées aux développeurs concernés.

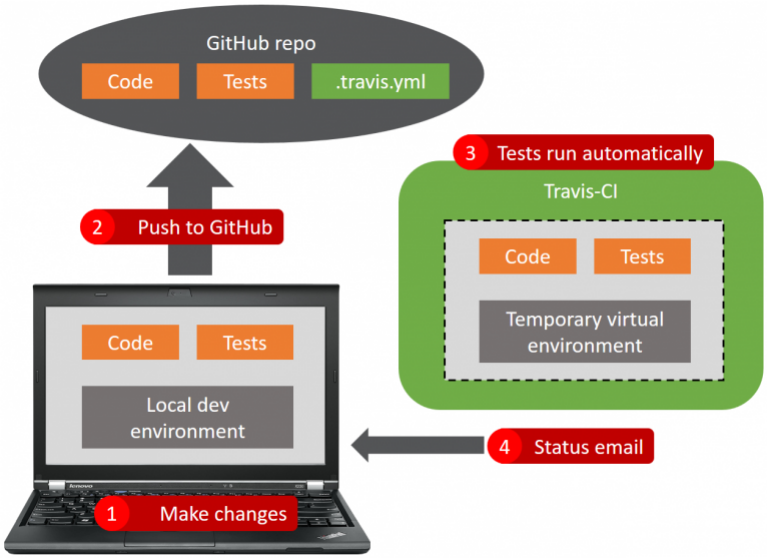

Figure 2.2 – Architecture de Travis [19]

Dans Travis-CI il y a quelques concepts fondamentaux :

- **job** : un processus automatisé qui clone votre référentiel dans un environnement virtuel et exécute une série de phases telles que la compilation de votre code, l'exécution de tests, etc. Un job échoue si le code de retour de la phase de script est différent de zéro.
- **phase** : les étapes séquentielles d'un job. Par exemple, la phase d'installation précède la phase de script, qui précède la phase de déploiement optionnel.
- **build** : un groupe de jobs. Par exemple, une compilation peut avoir deux jobs, dont chacun teste un projet avec une version différente d'un langage de programmation. Un build se termine lorsque tous ses jobs sont terminés.
- **stage** : un groupe de jobs exécutés en parallèle dans le cadre d'un processus de compilation séquentielle composé de plusieurs étapes.

Un avantage de Travis-CI est qu'il n'y a pas de serveur à configurer. Au lieu de cela, il suffit d'inclure un fichier de configuration dans un référentiel de contrôle de version et de lier le compte Travis-CI avec GitHub. De plus, Travis-CI facilite le test du code dans de multiples environnements en utilisant ce qu'ils appellent une matrice de compilation. Cela permet de mettre en place et de construire rapidement différents scénarios de cas pour l'application (ex : tester le logiciel avec différentes versions d'une bibliothèque spécifique).

Travis est gratuit pour les projets open-source et a des plans payants pour d'autres types.

Voici quelques-unes des plaintes les plus fréquentes des utilisateurs de Travis-CI [20] : manque d'intégration avec plus de sites d'hébergement du code, les builds s'exécutent seulement sur Linux ou macOS et lente compilation sur l'offre gratuite.

BuildBot

Buildbot est un système de planification des tâches basé sur Python et publié sous licence GPL. Il met en file d'attente les jobs, exécute les jobs lorsque les ressources nécessaires sont disponibles et rapporte les résultats [21].

Une installation BuildBot se compose de machines agissant en tant que maître et d'esclaves. Le maître vérifie périodiquement les changements dans le code source d'un référentiel de contrôle de version, coordonne les activités des travailleurs et renvoie les résultats des travaux aux développeurs. Les travailleurs peuvent faire son boulot sur une variété de plates-formes.

BuildBot est configuré par un script de configuration Python sur le maître.

Voici à quoi pourrait ressembler un workflow BuildBot typique :

1. Une machine maître interroge le référentiel de contrôle des versions et tire les dernières modifications.
2. Le maître envoie des commandes aux machines esclaves pour construire et tester le code.
3. Le maître reçoit les résultats des travaux et envoie les rapports aux développeurs par e-mail.

Figure 2.3 montre l'architecture de base de BuildBot.

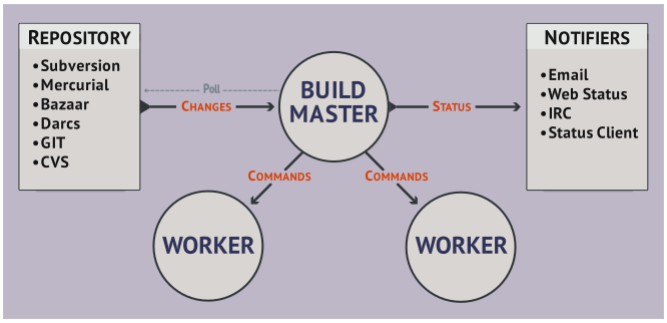

Figure 2.3 – Architecture de BuildBot [22]

Les esclaves s'exécutent généralement sur des machines séparées, qui peuvent correspondre chacune à une plate-forme d'intérêt. Ils envoient une demande au maître d'œuvre pour initier une connexion TCP. Si la connexion réussit, le maître de compilation l'utilise pour envoyer des commandes et les ouvriers l'utilisent pour envoyer les résultats de ces commandes.

Le travailleur doit pouvoir accéder au référentiel de contrôle de version car il n'obtient pas le code source du maître.

Étant écrit en Python, BuildBot peut s'intégrer avec d'autres outils et bibliothèques écrites dans le langage. De plus, il peut être fortement personnalisé et il est gratuit.

Quelques-uns des inconvénients de l'utilisation de BuildBot [23] sont les suivants : interface utilisateur difficile à utiliser, peu de documentation et difficile à comprendre, courbe d'apprentissage abrupte et la configuration initiale prend du temps.

TeamCity

TeamCity est un serveur de gestion de builds et d'IC basé sur Java et crée par JetBrains. Il a été publié pour la première fois le 2 octobre 2006.

Selon la documentation officielle, les fonctionnalités suivantes sont proposées par TeamCity [24] :

- Exécuter des compilations en parallèle sur de plateformes et environnements différentes.
- Optimisez le cycle d'intégration du code et garantir de ne jamais avoir du code cassé dans le référentiel.
- Examiner les résultats des tests à la volée avec des rapports de tests intelligents et le réordonnancement des tests intelligents.
- Personnaliser les statistiques sur la durée de la compilation, le taux de succès, la qualité du code et des mesures personnalisées.

Un simple workflow TeamCity pourrait ressembler à ceci :

1. Le serveur TeamCity détecte un changement dans le référentiel de contrôle de version et enregistre les changements dans une base de données.
2. Le déclencheur des compilations voit le changement dans la base de données et ajoute une compilation à la file d'attente.
3. Le serveur trouve un agent de compilation compatible pas occupé et assigne la compilation en file d'attente à cet agent.
4. L'agent exécute les étapes de compilation. Pendant que les étapes de compilation sont en cours d'exécution, l'agent de compilation rapporte la progression de la compilation au serveur TeamCity en envoyant tous les messages de log, les rapports de test, les résultats de couverture de code, etc. au serveur à la volée.
5. Après avoir terminé la compilation, l'agent de compilation envoie Build Artifacts au serveur.

La Figure 2.4 montre une des interfaces disponibles dans TeamCity. Celui-ci liste toutes les compilations en cours d'exécution avec quelques méta-informations.

Certaines des plaintes courantes des utilisateurs de TeamCity [26] sont les suivantes : configuration initiale difficile, mises à jour manuelles fastidieuses et complexité élevée du système.

Conclusion

Ce ne sont là que quelques-unes des plateformes utilisées pour mettre en œuvre l'intégration continue. Il en existe beaucoup d'autres pour répondre aux différents besoins des équipes. Le choix de la plateforme à choisir doit être fait avec soin, en tenant compte des forces et des faiblesses de chaque plate-forme ainsi que des besoins commerciaux de l'entreprise.

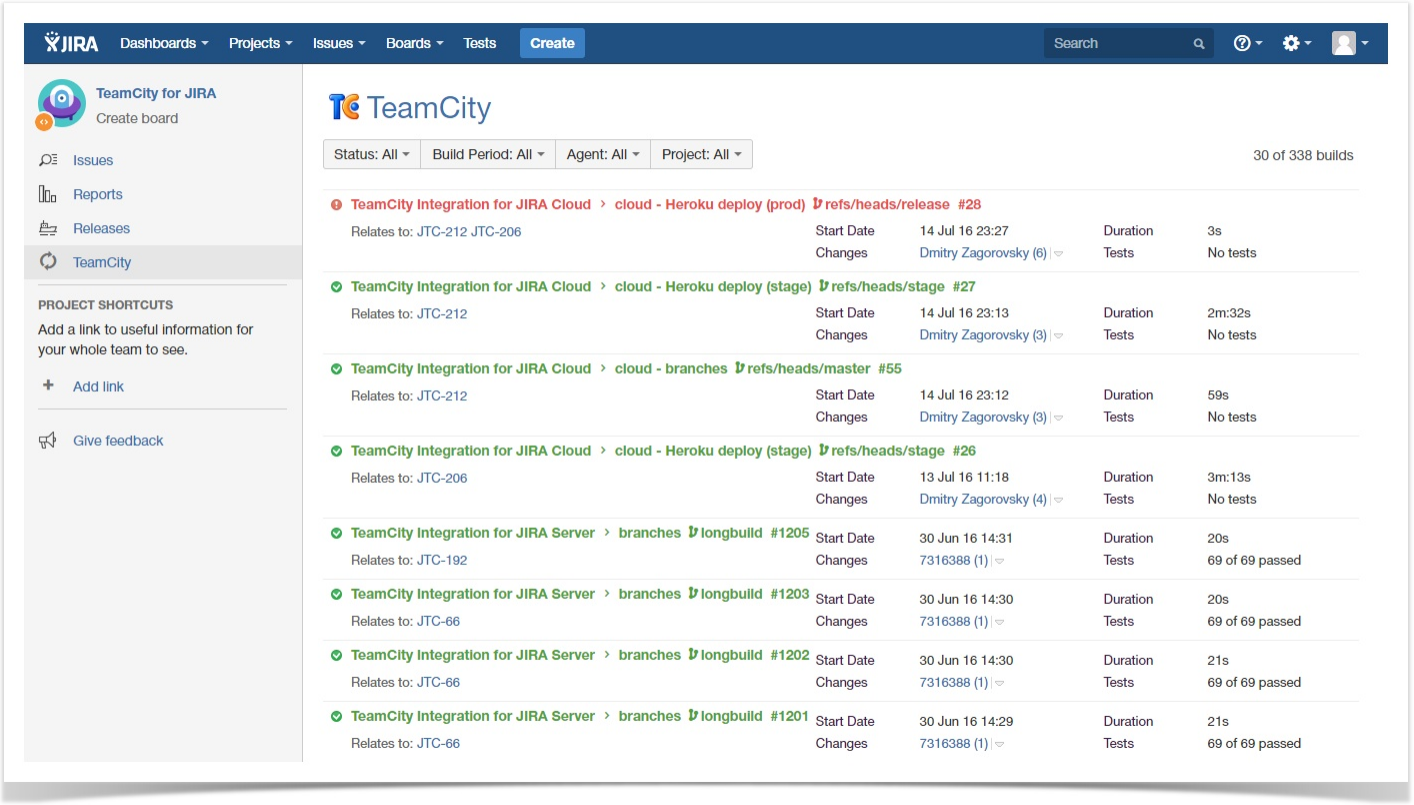

Figure 2.4 – Builds sur TeamCity [25]

CHAPITRE 3

Mise en œuvre de l'IC dans un projet existant

Dans ce chapitre, je parlerai de quelques scénarios possibles qui peuvent se produire lorsque l'on pense à mettre en œuvre l'intégration continue dans un projet existant.

Deux scénarios

Il y a deux scénarios qui peuvent se produire lorsque l'on pense à mettre en œuvre l'IC dans un projet.

Le premier cas est la mise en place de l'IC dans un projet complètement nouveau. C'est l'étape la plus flexible car aucun code n'a été écrit et il y a de la place pour la mise en œuvre de nouvelles idées avant que les objectifs ne soient fixés.

La deuxième option consiste à mettre en œuvre une intégration continue dans un projet existant. Par projet existant, j'entends un projet qui est passé de la phase de conception à la phase de développement. En effet, l'orientation du projet a été fixée et le développement est en cours.

Dans cette section, je vais explorer la deuxième option.

La mise en œuvre de l'IC dans une infrastructure existante pourrait être un effort non négligeable. En effet, il faut tenir compte non seulement du travail de mise en place de la plateforme choisie mais aussi de la culture de codage en place dans l'environnement de développement. Si les codeurs sont habitués à coder de manière isolée, par exemple, il faudra du temps et des efforts pour changer leurs habitudes de codage.

Phases dans un projet de développement

Regardons les phases possibles que nous pouvons rencontrer dans un projet [27] :

1. **Pas de serveur de build** Il n'y a pas de serveur central de compilation. Les compilations sont exécutées sur les machines des développeurs, soit manuellement, soit par le biais d'un script comme Ant. Le code source peut être stocké dans un référentiel de contrôle

de version mais les développeurs n'ont pas l'habitude d'intégrer régulièrement des changements au code. Avant la livraison, un développeur intègre les changements manuellement.

2. **Builds quotidiens** Maintenant il y a un serveur web qui fonctionne régulièrement (généralement la nuit). Le serveur compile le code, mais il n'y a pas de test automatisé. Même s'il y a des tests, ils ne font pas partie du processus de compilation. Les développeurs apportent des changements plus régulièrement, même si ce n'est qu'à la fin de la journée. S'il y a des conflits de code, le serveur de compilation avertira l'équipe le matin.
3. **Builds quotidiens et tests automatisés basiques** L'équipe prend l'IC un peu plus au sérieux maintenant. Le serveur de compilation s'exécute maintenant chaque fois qu'il y a un nouveau code sur le référentiel de contrôle de version. Le script de compilation ne se contente pas de compiler mais exécute également des tests unitaires et/ou d'intégration automatisés. Le serveur de compilation envoie également des notifications instantanées en cas de problème, non seulement par émail mais aussi par messagerie instantanée. Les échecs de build sont traités rapidement par l'équipe.
4. **Arrivée des métriques** Des métriques de qualité automatisées sont maintenant utilisées pour aider à évaluer la qualité du code et la couverture du test. La compilation mesurant la qualité du code a également créé la documentation de l'API de l'application. Un tableau de bord est également à la disposition de tous les membres de l'équipe pour voir l'état d'avancement du projet en un coup d'œil.
5. **Prendre les tests au sérieux** Des tests plus approfondis sont effectués après les tests de base, tels que les tests de bout en bout et les tests de performance.
6. **Tests d'acceptation automatisés et un déploiement plus automatisé** Acceptance Test Driven Development est utilisé tout au long du projet pour faire avancer le développement et générer des rapports de haut niveau sur l'état du projet.
7. **Déploiement Continu** La confiance dans le système d'intégration continue est telle que l'équipe est confiante de laisser les modifications être automatiquement déployées en production.

Cette liste n'est pas exhaustive et la progression entre les phases peut ne pas correspondre exactement aux situations du monde réel mais elle peut quand même nous donner une idée d'où on est dans notre processus d'adoption de l'IC et de ce qu'il reste à faire.

Dans la section suivante, nous verrons les contextes spécifiques qui pourraient survenir lors de la mise en œuvre de l'IC et nous verrons comment ils ont été abordés dans la pratique.

3.1 Adaptation des applications monolithiques

Dans cette section, nous allons examiner les défis uniques de la mise en œuvre de processus d'IC dans des applications monolithiques et quelques solutions possibles.

Qu'est-ce qu'une application monolithique ?

Un monolithe est essentiellement un gros bloc de pierre. De même, les applications monolithiques sont des applications constituées d'un seul bloc dont le but est de traiter toutes les demandes qu'on leurs pose. Ces applications sont écrites dans un seul langage. Quand on veut ajouter une nouvelle fonctionnalité, il peut avoir des problèmes liés aux interdépendances et à la régression.

Les applications monolithiques sont généralement de grande taille et de grande envergure. De plus, elles ne sont pas faciles à déboguer. Comme les différents composants sont étroitement liés, un changement dans une partie du code pourrait facilement avoir des répercussions ailleurs.

Quelques solutions essayées à des problèmes monolithiques

Nous verrons des entreprises qui ont dû faire face à la question de l'intégration continue dans des applications monolithiques. Nous décrirons également les solutions qu'elles ont trouvées.

Nextdoor Engineering [28]

Nextdoor est une société de services techniques. Ils avaient une équipe de plus de soixante-dix ingénieurs travaillant sur une application monolithique basée sur Django. Ils voulaient simplifier leur flux de travail de déploiement continu, mais ils étaient confrontés au problème que leurs compilations prendraient environ 25 minutes chacune à exécuter. Si chaque compilation ne comportait qu'un seul changement, cela signifie que les compilations feraient la queue et pourraient prendre des heures jusqu'à ce qu'elles arrivent en production.

Pour résoudre ce problème, ils ont introduit le concept d'un train. Un train est simplement un lot de changements. Pour déterminer ce qu'il y a à bord d'un train, ils ont un processus logique simple. Lorsqu'un changement atterrit sur le maître, s'il y a déjà un train existant, il est mis en file d'attente pour le train suivant. Si un tel train n'existe pas, un nouveau train est créé à partir de des changements en attente.

Les trains passent par trois phases : la livraison, la vérification et le déploiement.

1. **Première phase - livraison :** cela inclut la compilation et le déploiement des changements dans l'environnement de staging.
2. **Deuxième phase - la vérification :** cela inclut la vérification automatisée comme les tests et la vérification humaine.
3. **Troisième phase - déploiement automatique en production**

La fonctionnalité des trains a permis à Nextdoor de regrouper plusieurs changements, ce qui résout à son tour la question des compilations à changement unique qui ralentissent l'ensemble du système d'intégration continue.

Conductor [29]

Searchlight est une plateforme permettant d'obtenir des informations sur les habitudes de recherche des clients et de fournir des recommandations personnalisées. Parce que l'application elle-même a été construite comme une grande application monolithique, ils ont trouvé des problèmes. Ils ont constaté que lorsqu'un développeur changeait une partie de l'application, il fallait tester l'ensemble de l'application pour s'assurer que tout fonctionnait bien. De plus, s'ils voulaient améliorer leur performance dans un domaine, ils étaient obligés de considérer l'application dans son ensemble.

Pour résoudre ce problème, ils ont décidé d'utiliser des microservices. Ils l'ont fait en se concentrant sur trois grands principes :

1. **Définir les normes organisationnelles pour la conception, le développement et le déploiement des microservices.** Des normes communes pour la création de microservices simplifient le développement à long terme.
2. **Construisez de nouvelles fonctionnalités en tant que microservices, retirez les fonctionnalités existantes au besoin** Ne construisez que de nouvelles fonctionnalités et des fonctionnalités existantes (là où cela a du sens) en tant que microservices.
3. **Utiliser des solutions prêtes à l'emploi autant que possible**.

Pour construire et tester ces microservices, ils ont décidé d'utiliser l'application d'IC Jenkins. Ils ont créé un flux de travail consistant à vérifier le code pour détecter les erreurs, à effectuer des tests unitaires et contractuel et finalement déployer le code.

Avec cette nouvelle infrastructure de microservices en place, ils ont réussi à simplifier la compilation et les tests, puis ils ont pu déployer quotidiennement du code monolithique et déployer des microservices plusieurs fois par jour.

3.2 Obstacles à l'adoption de l'IC dans l'entreprise

Nous avons brièvement abordé le sujet de l'introduction de l'IC dans l'entreprise. Ici, nous allons le regarder de plus près.

Premièrement, il y a le coût en termes d'argent et d'heures-homme.

L'intégration continue nécessite l'utilisation de logiciels et de services spécialisés. Cela peut avoir un coût monétaire, mais plus important encore, un coût en heures-homme. Les développeurs doivent consacrer du temps à la formation pour apprendre à utiliser, intégrer et mettre en œuvre les nouveaux logiciels et services.

Deuxièmement, il y a le coût psychologique [30].

L'utilisation de pratiques d'intégration continue exige un changement de mentalité. Les développeurs travaillant de manière isolée doivent exposer leur travail dès le début et s'engager dans la collaboration. La perspective d'avoir à le faire peut les amener à montrer une certaine résistance aux changements.

Voici quelques moyens par lesquels les développeurs peuvent faire preuve de résistance à l'adoption de l'IC

- **Le scepticisme.** Pour une équipe qui n'a jamais travaillé avec l'IC, les avantages de la mise en œuvre d'une telle pratique pourraient ne pas être immédiatement évidents. Ils pourraient se demander pourquoi ils doivent consacrer leur temps et leurs efforts à la mise en œuvre d'un nouveau système alors que l'ancien fonctionne très bien. Il est important de pouvoir expliquer clairement les avantages de l'intégration de l'IC et de répondre aux questions des développeurs.
- **Changement de vieilles habitudes.** L'intégration continue exige que les développeurs intègrent leur code fréquemment. Si un développeur est habitué à développer du code de manière isolée pendant longtemps, il peut avoir du mal à changer sa façon de travailler.
- **Exposer le travail.** L'intégration continue exige que les développeurs intègrent leur code rapidement. Les développeurs habitués à intégrer leur code dans un événement big-bang peuvent préférer travailler sur leur branche pendant un certain temps, en corrigeant les bogues qui pourraient survenir. Avec l'IC, ils sont tenus de publier leur travail tôt, que le code soit de haute qualité ou non. Cela peut être décourageant car un développeur peut se sentir anxieux à l'idée que son code soit examiné par ses pairs.
- **Pression accrue.** C'est une augmentation de la pression. La pression de la direction ainsi qu'un sentiment de manque d'expérience pour s'adapter à de nouvelles exigences peuvent démoraliser une équipe. Par conséquent, il serait dans le meilleur intérêt de mettre en œuvre des changements graduels plutôt que d'essayer de tout faire en même temps.

CHAPITRE 4

Intégration continue dans l'entreprise

Dans ce chapitre je vais parler d'un projet dans mon entreprise dans lequel je propose des améliorations, en utilisant l'IC, sur la façon dont un outil est mis à jour.

Introduction

En tant qu'institution financière qui doit suivre des normes strictes de sécurité et d'utilisabilité, la nécessité d'effectuer de multiples contrôles et tests sur une partie de projets est d'une importance capitale. Il serait impensable pour une si grande entreprise d'avoir besoin d'effectuer ces tests manuellement. Il était donc nécessaire de trouver une solution avec un haut degré d'automatisation. L'IC est l'une des solutions qu'ils ont décidé de mettre en œuvre.

Une plateforme d'IC pour tous

Une plateforme basée sur Jenkins a été mise à disposition. Toute équipe qui souhaite démarrer avec l'IC peut faire une demande d'accès. Lorsque leur demande est acceptée, les équipes peuvent bénéficier de Jenkins, la principale plateforme d'IC, ainsi que de nombreux outils et applications qui peuvent être intégrés dans des scripts de compilation (voir Figure 4.1). Il est à noter que les équipes n'ont pas besoin d'utiliser toutes les fonctionnalités disponibles. En fait, chaque équipe peut personnaliser le serveur de compilation Jenkins à ses propres besoins.

Nous allons maintenant examiner les différentes étapes d'une éventuelle compilation d'IC chez Natixis.

Étape 1 : Récupération des dépendances

Cette phase consiste à télécharger les dépendances des logiciels utilisés dans le projet de développement.

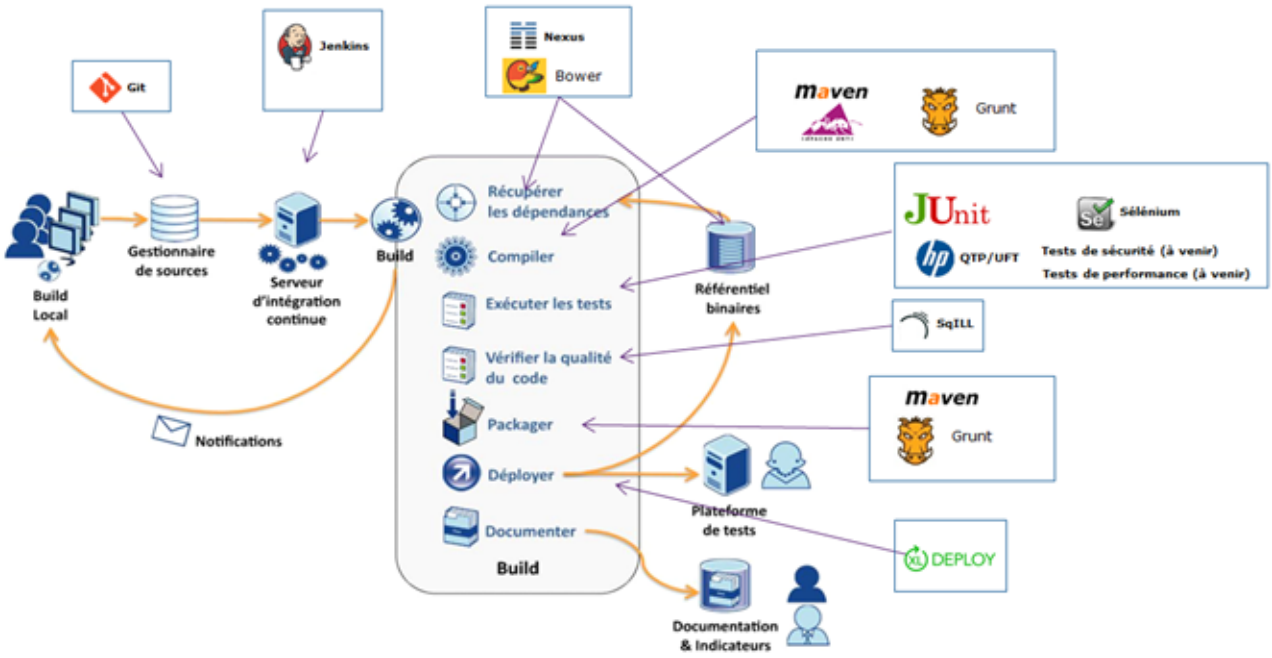

Figure 4.1 – La plateforme d'integration continue chez Natixis

Étape 2 : Compilation

Le code est compilé.

Étape 3 : Exécution des tests

On vérifie le bon fonctionnement du logiciel avec plusieurs types de tests.

Étape 4 : Vérification de la qualité du code

On vérifie la qualité du code (code smells, bugs, vulnérabilités)

Étape 4 : Packaging du code

Le code est mis ensemble dans un format qui peut être distribué.

Étape 5 : déploiement du code

Transfert du package créé à l'étape 4 vers une instance de XLDeploy (hors de sujet).

Étape 6 : documentation

Création de la documentation par les développeurs si besoin (hors sujet).

Logiciels mis à disposition

- **Maven** : résolution des dépendances.
- **Grunt** : automatisation de JavaScript.
- **JUnit** : tests unitaires.
- **QTP/UFT** :test fonctionnels et de régression en simulant des événements souris ou clavier sur des interfaces graphiques.
- **Selenium** : tests en simulant des actions sur un navigateur web.
- **SonarQube (SqILL en interne)** : trouve les bugs, les code smells et les vulnérabilités dans le code.
- **Nexus** : stockage centralisé et organisé de données.
- **Bower** : gestionnaire de paquets.

4.1 Projet Fenergo

Introduction

Fenergo est un logiciel de gestion du cycle de vie du client (CLM), sélectionné par Natixis pour offrir un point d'entrée unique pour le processus KYC (Know Your Client), la Due Diligence réglementaire, les Credit Limit et les Master Agreement Requests. Ces processus étaient auparavant pris en charge par plusieurs applications.

Le projet de mise en œuvre de Fenergo a débuté en août 2017, avec un objectif de mise en service d'un produit minimum viable (MVP) au début du deuxième trimestre 2018. Le projet est toujours en cours de développement en août 2018.

Mon implication dans le projet

L'idée d'étudier le projet de Fenergo est venue lors de conversations avec M.G., qui travaillait, au moment de la rédaction de ce document, dans le département DevOps et infrastructure de Natixis. En parlant avec lui et en regardant la documentation interne, j'ai identifié des inefficacités par rapport au workflow de mise à jour de l'application (voir Figure 4.2). Chaque fois qu'une nouvelle version de Fenergo est disponible, une série d'étapes doit avoir lieu. Ces étapes comprennent le téléchargement du package, sa décompression dans des dossiers spécifiques, la mise en ligne des sources de Fenergo dans un dépôt Artifactory et le déclenchement d'une compilation dans le serveur Jenkins.

Il y a aussi quelques étapes optionnelles. Comme elles ne sont pas obligatoires, j'ai décidé de les laisser en dehors de la solution que je propose. Néanmoins, elles pourraient être également automatisées.

Il peut sembler à première vue que les étapes précédant la compilation elle-même ne font pas partie du processus IC. Cependant, on peut faire valoir que tout ce que la compilation nécessite pour fonctionner devrait y faire partie. Si le processus pouvait être automatisé, non seulement il éliminerait l'inefficacité et l'erreur humaine, mais il augmenterait aussi la productivité en libérant du temps pour d'autres tâches.

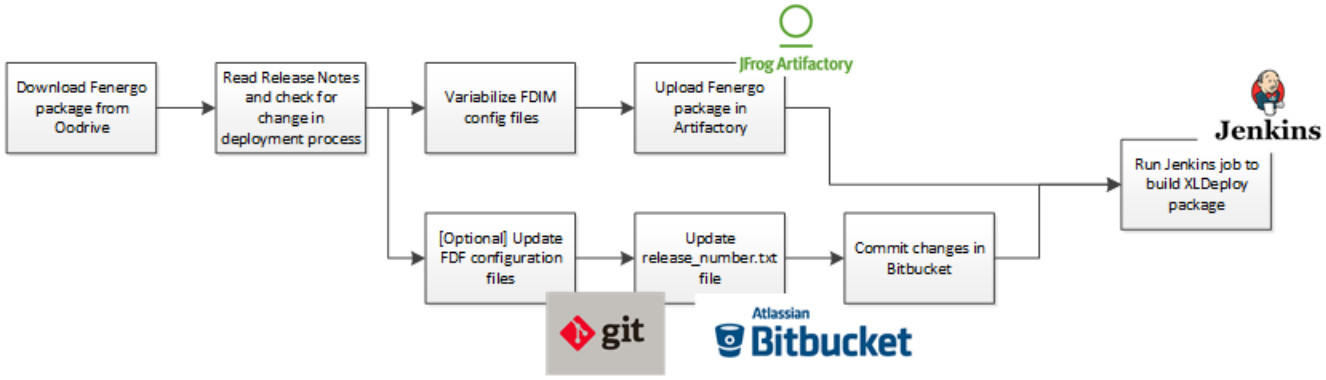

Figure 4.2 – La procédure de déploiement de Fenergo

J'ai contacté les personnes responsables de l'application, mais il ne m'a pas été permis de pouvoir tester toutes mes idées dans la pratique. J'ai fait de mon possible pour vérifier que les commandes fonctionnent sur Jenkins, mais par exemple je n'ai pas pu tester l'accès à une API REST car je n'ai pas les droits d'accès nécessaires. Dans le cas où je n'ai pas pu tester, je parlerai des recherches que j'ai faites sur les outils et les méthodes qui pourraient être mis en œuvre afin d'automatiser le workflow.

Automatisation du workflow

Dans cette section, je vais détailler les différentes étapes du workflow d'installation d'une nouvelle version de Fenergo et je vais parler de la solution que je propose.

Les explications des sections suivantes sont valables pour le système d'exploitation Red Hat (utilisé dans une partie des serveurs de Natixis).

Les étapes suivantes contiennent du code qui est à utiliser dans le contexte d'un projet de type pipeline sur Jenkins. Le code de base pourrait être le suivant :

```
pipeline {
    agent any
    stages {
        stage('preparer-build') {
            steps {
                script {
                    // code ici
                }

            }
        }
    }
}
```

On crée un stage qu'on appelle "preparer build" et on dit que n'importe quel agent peut l'exécuter. Ensuite on va mettre nos commandes dans la section "steps". La section avec

le nom "script" sert à pouvoir exécuter un type spécifique de pipeline qui nous permettra d'exécuter notre code.

Vérification de dernière version et téléchargement de fichiers

Dans cette étape, nous téléchargeons la dernière version du logiciel Fenergo auprès d'un fournisseur de services cloud (CSP : Cloud Service Provider).

Cependant, avant de télécharger, nous devons déterminer s'il existe une nouvelle version. La première chose à faire est de vérifier si le CSP fournit une telle fonctionnalité par le biais d'une API. Si c'est le cas, il suffit de créer une solution qui utilise cette API pour récupérer les informations souhaitées.

Même dans le cas où le fournisseur web ne fournit pas une API, c'est toujours possible de concevoir des solutions pour avoir accès aux informations de la dernière version. Cela pourrait se faire en simulant un navigateur web et/ou en utilisant curl ou wget pour envoyer de requêtes HTML par ligne de commande (voir après).

Le problème avec ce genre d'approche, c'est qu'elle est fragile. Si le CSP change la structure de leur site Web, cela pourrait casser le workflow et demander du temps pour faire des modifications.

C'est pour cela qu'une solution basée sur une API est toujours préférable.

API REST

Une API REST permet aux clients d'envoyer des requêtes HTTP à certaines URLs d'un serveur et de recevoir de données en retour, généralement sous la forme d'un fichier JSON ou XML.

Oodrive fournit ce type d'API. Elle accepte les méthodes GET, POST, PUT, DELETE et répond avec du code XML (sauf dans le cas de téléchargement d'un fichier).

La première chose à faire est de nous authentifier en utilisant le service d'authentification de l'API. Pour cela, on envoie une requête POST en utilisant la commande curl. Si la commande n'est pas installé, on peut le faire avec "yum install curl" (valable sur Red Hat Enterprise, autrement il faut aller sur le site officiel de curl et suivre les instructions)

Une fois que curl est téléchargé et installé, on le lance avec quelques paramètres.

```
sh(script: 'curl -X POST -c cookie-jar.txt http://<host>/<context>/
    ↪ authentication -d "login=<user>&passwd=<pass>&workspace=<workspace>
    ↪ "')
```

Lorsque le client se connecte pour la première fois, le serveur écrit dans le flux HTTP de la réponse le cookie avec un identifiant de session. En utilisant l'option -c, on sauvegarde ce cookie dans le fichier "cookie-jar.txt". Chaque fois que nous utiliserons des demandes de curl ultérieures, nous lirons les cookies de ce fichier pour nous authentifier automatiquement.

<host> est l'adresse du serveur. <context> est utilisé pour arriver à la bonne application. Finalement, "authentication" est le mot clé demandé pour l'API pour accéder à la fonctionnalité d'authentification. On a besoin de fournir un utilisateur, un mot de passe et un workspace. On le fait avec l'option -d.

Maintenant, on veut vérifier qu'il existe une nouvelle version de Fenergo. La fonctionnalité de faire ça directement n'existe pas, mais on peut le faire avec la fonctionnalité de l'API qui permet de voir le contenu d'un dossier.

```
def response = sh(script: 'curl -X POST -b cookie-jar.txt http://<host>/<
    ↪ context>/folders/foldercontent -d "folder=<id>"', returnStdout:
    ↪ true)
```

Une requête POST est envoyée pour obtenir le contenu du dossier avec id <id>. Ensuite, nous stockons le résultat de cette requête (le code XML) dans une variable appelée "reponse".

Maintenant il faut parser cette réponse pour obtenir la liste de tous les fichiers. On s'intéresse à savoir s'il y a une nouvelle version par rapport à la dernière qu'on a déjà.

Pour parser, on va utiliser l'outil de ligne de commande xmllint. Cet outil va nous permettre d'extraire le nom du dernier fichier sur Oodrive. Ce nom est composé de la chaîne de caractères "Fenergo" suivie du numéro de version. Comme on veut seulement le numéro de version, on peut l'extraire avec la ligne suivante :

```
String versionNumber = nomFichier.substring(7);
```

Maintenant il faut comparer cette version à celle de la dernière installation. Pour faire cela on a plusieurs options. On pourrait stocker le numéro de version du dernier package Fenergo installé dans un fichier de texte (manuellement la première fois). Si le numéro de version dans le fichier plus récent listé dans le XML est plus élevé, ça veut dire qu'il y a une mise à jour disponible. Il faudrait donc télécharger cette nouvelle version et à la fin du processus du build mettre à jour le fichier avec la dernière version.

Si on stocke le dernier numéro de version dans un fichier de texte, on a besoin de l'extraire. Pour faire cela, on peut utiliser la commande readFile de Jenkins. Cela va nous donner le contenu d'un fichier sous la forme d'un String.

On fait la comparaison entre les deux numéros de version. Si la version lue depuis le fichier xml est plus récente, alors on va la télécharger et l'installer. Ceci peut aussi être fait en utilisant l'API REST.

```
sh(script: 'curl -X POST -b cookie-jar.txt http://<host>/<context>/files/
    ↪ downloadfile -d "id=<id_fichier>"')
```

Où l'id fichier est l'id qui est dans le fichier XML et qu'on peut extraire de la même façon qu'on l'a fait pour le nom du fichier.

Lecture du journal des modifications

Dans cette étape, l'objectif est de lire le journal des modifications afin de savoir s'il faut faire de changements par rapport à la procédure de livraison du package. Ceci demande l'intervention d'un développeur.

Pour tenir compte de devoir attendre une intervention humaine, on peut ajouter à notre pipeline la commande "input". Cela nous permet de demander à l'utilisateur de dire "Oui" ou "Non" à une question du type "Continuer?". S'il y a pas de changements à faire, le développeur peut taper "Oui". Autrement, Jenkins va lui donner le temps de faire le nécessaire.

Déplacer des fichiers

Quelle que soit la façon dont nous choisissons de télécharger le fichier, nous devons maintenant le déplacer dans un répertoire spécifique. Pour ce faire, on peut utiliser la commande "move".

```
mv chemin/<fichier.source> <nouveau dossier>
```

Si on avait déjà téléchargé le fichier dans le répertoire courant, nous pouvons sauter la partie chemin et simplement mettre le nom du fichier source.

Dézipper des fichiers

Ensuite, on doit traiter le fichier zip. Cette étape consiste à décompresser un fichier en deux fichiers zip.

Jenkins fournit un moyen natif de le faire grâce à la commande de décompression unzip.

```
unzip <zipfile> [dir]
```

Où dir est le chemin du répertoire de base dans lequel extraire le contenu du zip.

S'il faut dézipper un fichier dans un dossier spécifique, on peut créer ce dossier avec la commande mkdir

```
mkdir <chemin du nouveau dossier>
```

Loader de fichiers vers Artifactory

L'un de fichiers dézippés est un package avec les sources de Fenergo. Le but dans cette étape est de loader ce package vers le gestionnaire de dépôt d'objets binaires Artifactory. Il y a au moins deux façons officielles de le faire.

Requête PUT

Artifactory expose une API REST que nous pouvons utiliser pour envoyer une requête PUT (un type de requête HTTP qui place un fichier ou une ressource à une URI spécifique) pour "déployer" ou loader un fichier.

Cette requête pourrait être envoyée par exemple en utilisant curl.

Pour l'authentification, Artifactory offre plusieurs moyens, y compris utilisateur/mot de passe, l'utilisation d'une clé API et un jeton d'accès.

Plugin Jenkins Artifactory

Ce plugin permet aux jobs de compilation de Jenkins de déployer des binaires ainsi que d'autres fonctionnalités.

Pour l'utiliser, il faut d'abord définir le serveur Artifactory qui sera utilisé pour le déploiement des fichiers. Ceci peut être fait dans Jenkins à partir d'un formulaire.

Ensuite, dans notre script pipeline, nous définissons notre serveur avec une ligne de type :

```
def server = Artifactory.server 'my-server-id'
```

Où my-server-id correspond au serveur que nous avons configuré précédemment. Ce champ peut être trouvé dans Jenkins.

Ensuite, pour loader des fichiers, nous devons d'abord créer une spec qui est un fichier JSON qui spécifie quels fichiers doivent être uploadés et le chemin cible [31].

Enfin, nous appelons la fonction de mise en ligne de notre objet serveur et nous lui passons notre spec comme paramètre.

```
server.upload(uploadSpec)
```

Compiler le source

La seule chose qui reste à faire est de lancer le projet déjà défini par l'entreprise qui va chercher les sources dans Artifactory et les compile. Pour faire cela, on peut ajouter une étape à notre pipeline.

```
build job: 'NomDuJob',
```

Gestion d'erreurs

Comment gérer une compilation qui a échouée? On pourrait concevoir d'envoyer des e-mails aux développeurs s'il y a une erreur. Pour cela, on va ajouter la section post à notre pipeline après la section "stages". Cette section contient des étapes qui s'exécuteront à la fin d'une compilation (que ce soit réussie ou pas). Dans la section "post", la sous-section "failure" va s'exécuter si la compilation échoue. La commande "mail" dans cette sous-section va envoyer un e-mail à l'adresse ou les adresses indiqués dans le champ "to". S'il y a plus d'une adresse, il faut les séparer par des virgules [32].

```
pipeline {
    ...
    stages {
        ...
    }
    post {
        failure {
            mail bcc: '', body: "<b>Exemple</b>", cc: '', charset: 'UTF-8',
               ↪  from: '', mimeType: 'text/html', replyTo: '', subject:
               ↪ "Erreur de Compilation - Fenergo", to: "
               ↪ developpeur@natixis.com";

        }
    }
}
```

Diagramme de séquence du workflow proposé

La Figure 4.3 montre le nouveau workflow proposé dans le cas où tout marche bien.

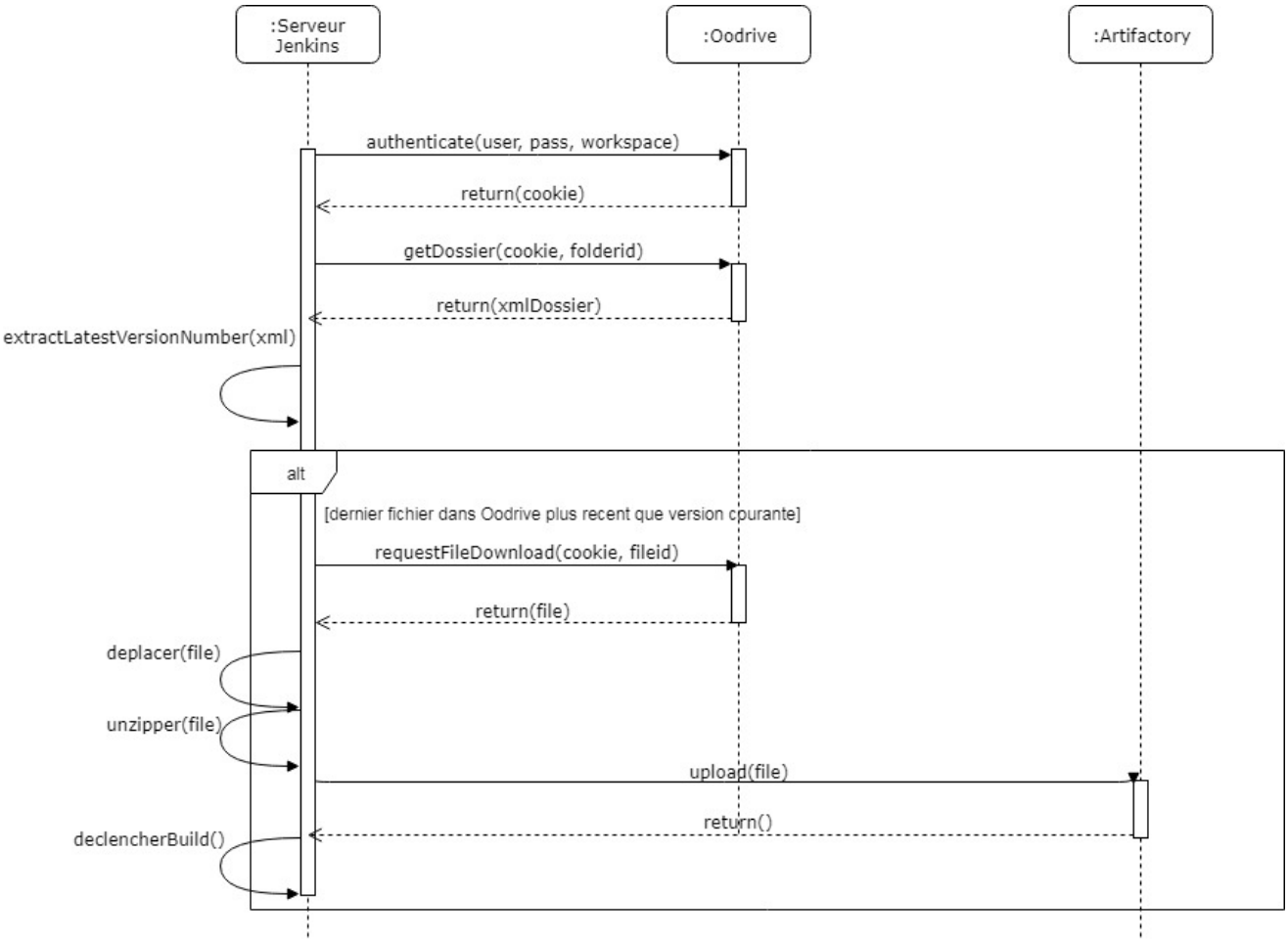

Figure 4.3 – Nouveau workflow automatisé

Conclusion

Malgré les défis associés à sa mise en œuvre, l'intégration continue d'être populaire parmi les entreprises du monde entier.

Pour les propriétaires d'entreprises, c'est un moyen d'être plus compétitif grâce aux gains de rapidité et d'efficacité. Pour les développeurs eux-mêmes, cela signifie moins de temps à faire des tâches répétitives et plus de temps pour travailler sur leurs objectifs grâce aux vastes possibilités d'automatisation. Les clients en profitent également. Quand ils verront le produit qu'ils ont demandé, il y aura moins d'erreurs et il aura été plus développé qu'un projet où l'automatisation n'existe pas.

Cependant, il est possible que dans certains cas, l'introduction de l'intégration continue dans un projet de développement logiciel en cours ne soit pas justifiée..

S'il s'agit d'un projet ponctuel et qu'il se terminera bientôt, alors peut-être que l'effort de mise en œuvre de l'intégration continue n'en vaut pas la peine. Un deuxième cas est également possible. Si nous voulons implémenter l'IC dans un projet monolithique, nous devons déterminer si c'est efficace ou si, par exemple, les compilations vont prendre trop de temps à s'exécuter.

Les défis psychologiques doivent également être pris en compte et abordés. Le succès de l'introduction de l'intégration continue dans la culture d'entreprise peut dépendre autant des défis techniques que de la réticence des gens au changement.

En fin de compte, l'intégration continue représente une amélioration marquée par rapport à la manière précédente de faire les choses. Si une entreprise peut surmonter les barrières psychologiques de développeurs et les considérations de coût à court terme, elle peut très bien acquérir un avantage considérable en termes d'efficacité par rapport à ses concurrents.

Bibliographie

[1] Wikipédia. Natixis — wikipédia, l'encyclopédie libre. http://fr.wikipedia.org/w/index.php?title=Natixis&oldid=150673472, 2018. Consulté le 27-juillet-2018.

[2] Natixis. https://www.natixis.com/natixis/jcms/ala_5361/fr/profil. Consulté le 27-juillet-2018.

[3] Alek Sharma. A brief history of devops. https://circleci.com/blog/a-brief-history-of-devops-part-i-waterfall/, 2018. Consulté le 27-juillet-2018.

[4] Grady Booch. *Object Oriented Design with Applications*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1991.

[5] Kent Beck and Cynthia Andres. *Extreme Programming Explained : Embrace Change (2nd Edition)*. Addison-Wesley Professional, 2004.

[6] Martin Fowler. Continuous integration. https://www.martinfowler.com/articles/continuousIntegration.html, 2006. Consulté le 27-juillet-2018.

[7] Sam Guckenheimer. What is continuous integration? https://www.visualstudio.com/learn/what-is-continuous-integration/, 2017. Consulté le 27-juillet-2018.

[8] H. H. Olsson, H. Alahyari, and J. Bosch. Climbing the "stairway to heaven" – a mulitiple-case study exploring barriers in the transition from agile development towards continuous deployment of software. In *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*, pages 392–399, 2012.

[9] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. Continuous integration, delivery and deployment : A systematic review on approaches, tools, challenges and practices. *CoRR*, abs/1703.07019, 2017.

[10] Ambysoft. Agile practices and principles survey. http://www.ambysoft.com/downloads/surveys/PracticesPrinciples2008.pdf, 2008. Consulté le 27-juillet-2018.

[11] Perforce. Continuous delivery : The new normal for software development. https://www.perforce.com/sites/default/files/files/2017-07/continuous-delivery-migration.pdf, 2017. Consulté le 27-juillet-2018.

[12] Brett Slatkin. Continuous delivery at google. https://air.mozilla.org/continuous-delivery-at-google/, 2013. Consulté le 27-juillet-2018.

[13] Ben Schmaus. Deploying the netflix api. https://medium.com/netflix-techblog/deploying-the-netflix-api-79b6176cc3f0, 2013. Consulté le 27-juillet-2018.

[14] Meet jenkins. https://wiki.jenkins.io/display/JENKINS/Meet+Jenkins. Consulté le 27-juillet-2018.

[15] Best continuous integration software. https://www.g2crowd.com/categories/continuous-integration. Consulté le 27-juillet-2018.

[16] Hudson's future. https://jenkins.io/blog/2011/01/11/hudsons-future/. Consulté le 27-juillet-2018.

[17] https://www.trustradius.com/products/jenkins/reviews/pros-and-cons. Consulté le 28-août-2018.

[18] plaindocs et al. Core concepts for beginners. https://docs.travis-ci.com/user/for-beginners/, 2015. Consulté le 27-juillet-2018.

[19] DMAHUGH. http://mahugh.com/2016/09/02/travis-ci-for-test-automation/, 2016.

[20] https://www.trustradius.com/products/travis-ci/reviews. Consulté le 28-août-2018.

[21] Buildbot : the continuous integration framework. https://buildbot.net/. Consulté le 27-juillet-2018.

[22] http://docs.buildbot.net/0.8.3/System-Architecture.html.

[23] https://www.g2crowd.com/products/buildbot/reviews. Consulté le 28-août-2018.

[24] Pavel Sher et Julia Alexandrova. Continuous integration with teamcity. https://confluence.jetbrains.com/display/TCD10/Continuous+Integration+with+TeamCity, 2015. Consulté le 27-juillet-2018.

[25] https://docs.stiltsoft.com/display/public/JTC/Viewing+TeamCity+Builds.

[26] https://www.g2crowd.com/products/teamcity/reviews. Consulté le 28-août-2018.

[27] John Ferguson Smart. *Jenkins : The Definitive Guide*. O'Reilly Media, Inc., 2011.

[28] Niall O'Higgins. 3 hard lessons from scaling continuous deployment to a monolith with 70+ engineers. https://bit.ly/2mLJ7L9, 2017. Consulté le 27-juillet-2018.

[29] Niall O'Higgins. Revamping continuous integration and delivery at conductor. https://www.conductor.com/nightlight/revamping-continuous-integration-delivery-conductor/. Consulté le 27-juillet-2018.

[30] Adam Debbiche, Mikael Dienér, and Richard Berntsson Svensson. Challenges when adopting continuous integration : A case study. In Andreas Jedlitschka, Pasi Kuvaja, Marco Kuhrmann, Tomi Männistö, Jürgen Münch, and Mikko Raatikainen, editors, *Product-Focused Software Process Improvement*, pages 17–32, Cham, 2014. Springer International Publishing.

[31] Rami Honig. https://www.jfrog.com/confluence/display/RTF/Working+With+Pipeline+Jobs+in+Jenkins, 2018. Consulté le 28-août-2018.

[32] Jenkins. https://jenkins.io/doc/pipeline/steps/workflow-basic-steps/#mail-mail. Consulté le 28-août-2018.