Plateforme SaaS pour l'intégration continue

Julien Garet, Christophe Demarey

Plateforme SaaS pour l'intégration continue

Julien Garet
Inria – Centre de Recherche Lille – Nord Europe
Direction des systèmes d'information, des infrastructures et des services informatiques
Parc scientifique de la Haute Borne
40, avenue Halley - Bât A - Park Plaza
59650 Villeneuve d'Ascq - France

Christophe Demarey
Inria – Centre de Recherche Lille – Nord Europe
Service d'Expérimentation et de Développement
Parc scientifique de la Haute Borne
40, avenue Halley - Bât A - Park Plaza
59650 Villeneuve d'Ascq - France

Résumé

L'intégration continue est devenue un des piliers des bonnes pratiques du développement logiciel. Nous présenterons au cours de cet article les concepts de l'intégration continue, basés sur les tests et l'automatisation, qui ont amené à son essor dans la distribution de logiciels, le cahier des charges issu des besoins des scientifiques, ainsi que les différentes étapes qui ont conduit à la réalisation de la plateforme actuelle.
Les besoins exprimés par les scientifiques sont divers. Le principal point est de les soulager de toutes les tâches d'administration système et réseau (authentification, sauvegarde, mises à jour de sécurité, etc.) en leur offrant un service clés en main. Ils veulent aussi avoir une grande latitude sur les versions et plugins utilisés.
Après analyse de ces besoins, il est apparu indispensable que la mise à disposition d'un serveur d'intégration continue soit la plus automatisée possible. Nous proposons à chaque projet de pouvoir installer la version d'Hudson qu'ils désirent ainsi qu'un système de mise à jour simplifié, garantissant l'intégrité et la conformité de leur installation ainsi qu'une grande disponibilité du service. La plateforme a donc été pensée en mode SaaS (Software as a Service), basé sur la mise à disposition du service d'intégration continue pour l'utilisateur avec un minimum d'intervention humaine. Le logiciel choisi a d'abord été Hudson, qui malgré une orientation Java peut convenir à tout type de projet de développement. Courant 2011, le projet Jenkins, fork d'Hudson, est né. Nous avons fait le choix de supporter les deux solutions et de garder une architecture ouverte pour accueillir de nouveaux outils.

Mots clefs

Intégration continue, Hudson, Jenkins, Virtualisation

1 Introduction

Le développement logiciel est une part très importante de l'activité scientifique d'Inria. Ces logiciels sont développés par plusieurs personnes, venant de plusieurs établissements, et destinés à des communautés d'utilisateurs parfois nombreuses et d'horizons variés. Il est donc aujourd'hui primordial de s'assurer de la qualité et de la fiabilité de ces logiciels. Le développement de tests permet d'atteindre ces objectifs, mais l'exécution manuelle de ces tests n'est plus suffisante. Les problèmes introduits dans le code sont vus trop tard et sont, par conséquent, plus difficiles à résoudre a posteriori. L'intégration continue a pour objectif de s'assurer de manière automatisée qu'après chaque modification du logiciel, celui-ci ait toujours le comportement attendu (conforme aux spécifications).

C'est dans ce cadre qu'est né le projet de mettre à disposition des chercheurs et ingénieurs d'Inria une plateforme d'intégration continue mutualisée. Le choix initial s'est porté sur le logiciel Hudson [1] de par son adoption par un très grand nombre d'acteurs (open source, recherche, industriels) et de par sa communauté très active.

2 Améliorer la qualité du logiciel

L'industrie du logiciel n'est pas aujourd'hui au même niveau de standard de qualité et de fiabilité que d'autres domaines, par exemple l'industrie chimique, l'électronique. Ce constat n'est pas valable pour tous les logiciels. Certains codes sont très robustes, comme les codes embarqués dans le secteur aéronautique (avions, fusées), le secteur énergétique (centrales), mais globalement le niveau de qualité des logiciels livrés est aujourd'hui insuffisant. Pourquoi? Quelles en sont les causes?

2.1 Détecter les erreurs au plus tôt

Une des difficultés majeures dans la qualité d'un logiciel est le fait de pouvoir détecter que ce logiciel comporte des erreurs ou des dysfonctionnements, plus communément appelés « bogues ». Ces anomalies peuvent être variées : un problème de syntaxe dans le programme, une erreur dans l'algorithme, un non-respect du cahier des charges, etc. Un des challenges pour assurer un certain niveau de qualité / fiabilité est de détecter au plus tôt ces anomalies. Plus une erreur (ou défaut) est identifiée rapidement, plus le temps nécessaire pour la corriger est court.

Prenons un exemple : je dois développer une fonction me permettant d'obtenir la factorielle d'un entier n. Cette fonction sera ensuite utilisée dans des calculs bien plus complexes. Une erreur s'est glissée dans l'implémentation (par exemple 0 != 0). Si cette erreur est détectée dès la fin de l'implémentation de cette fonction, la correction prendra tout au plus quelques minutes. Si vous la découvrez un mois plus tard, vous avez oublié tout ou partie des détails de l'implémentation et devrez passer du temps pour reconstituer l'algorithme correspondant. Mais ce n'est pas le cas le plus défavorable : si vous commencez à utiliser ce code dans d'autres fonctions, elle-mêmes utilisées par d'autres fonctions de plus haut niveau, et que l'erreur se produit dans une de ces fonctions de haut-niveau, alors il faudra remonter toute la chaîne d'exécution afin de déterminer l'élément en cause. C'est cette recherche ou identification du problème précis qui est la plus coûteuse en temps. La résolution du problème en elle-même peut être triviale (comme dans ce cas) mais l'identification de celui-ci complexe. Si on imagine maintenant que notre brique logicielle sera utilisée par une autre brique logicielle, elle-même intégrée dans une plateforme logicielle, l'identification risque de prendre un certain temps ! Cela aura aussi un coût pour l'employeur car une ou plusieurs personnes seront mobilisées pour résoudre le problème.

C'est pourquoi il est indispensable de se donner les moyens de détecter les erreurs au plus tôt.

2.2 Le test indissociable du développement

« Un test est une procédure de vérification partielle d'un système », *Wikipédia*. Il est en effet souvent impossible de certifier qu'un logiciel aura le comportement attendu dans tous les cas. L'objectif des tests est donc de détecter un maximum de comportements problématiques ou pouvant le devenir. Le niveau d'exigence des tests dépendra du niveau de qualité souhaité.

On peut distinguer quatre niveaux de tests :

- tests unitaires : s'assurer du bon fonctionnement d'une partie du logiciel (unité) ;
- tests d'intégration : vérifier que les différentes briques logicielles développées s'intègrent bien les unes avec les autres et ont le comportement attendu ;
- tests fonctionnels : s'assurer que le logiciel offre bien les fonctions attendues dans le recueil des besoins ;
- tests d'acceptation (ou recette).

Les tests de non régression sont un autre type de tests et peuvent faire parti d'un des quatre niveaux cités ci-dessus. Ils ont pour objectif de s'assurer que des défauts n'ont pas été introduits suite à l'évolution du logiciel (ex: il faut s'assurer qu'une fonctionnalité correcte en version 1.0 soit toujours correcte pour la version 2.0 du logiciel).

On trouve également d'autres types de tests comme les tests de performance, de sécurité, etc.

Mais quand écrire ces tests?

Les tests doivent être écrits au fur et à mesure des développements et des intégrations de fonctionnalités. Ils peuvent être écrits avant l'implémentation de la fonctionnalité (développement dirigé par les tests) ou plus classiquement après l'implémentation de

celle-ci. Les tests ne doivent pas être une tâche spécifique prévue après tous les développements. Elle est partie intégrante des développements !

Une fois ces tests écrits, nous avons à notre disposition un moyen de vérifier le comportement de notre logiciel.

2.3 Se donner les moyens d'assurer un niveau de qualité satisfaisant

La qualité d'un logiciel dépend de nombreux facteurs : du respect des besoins exprimés, de patrons de conception, du développement, des algorithmes, du codage, etc. Si l'on souhaite simplifier ce concept de qualité, on pourrait le définir par les questions suivantes :

- Mon logiciel répond-il au besoin ?
- A-t-il le comportement attendu ?

Le seul moyen de répondre objectivement à ces questions est de mettre en place des outils permettant de mesurer le niveau de conformité. Ces outils sont les tests. Il en existe une grande variété, permettant de répondre à de nombreuses questions. Grâce à ces tests, nous pourrons affirmer que, pour un certain nombre de situations, notre logiciel répond au besoin. Pour être tout à fait complet, les tests seuls ne suffisent pas. Il faut également savoir quantifier le nombre de situations testées par rapport au nombre total de situations. Il existe des outils de couverture de code qui permettent de détecter le pourcentage de situations réellement testées. En simplifiant à l'extrême, on pourrait dire qu'un logiciel est de bonne qualité si les résultats des tests sont ceux attendus et que le taux de couverture est d'un niveau satisfaisant.

3 Intégration continue

Nous connaissons maintenant le moyen permettant de vérifier la qualité d'un logiciel mais nous ne nous sommes pas encore posés la question de quand les utiliser? A la fin du développement? Chaque semaine? A chaque instant? Est-ce une action manuelle ou automatisée? Plusieurs réponses sont possibles à ces questions, selon les méthodes de développement choisies : cycle en v, développement itératif, etc. Une réponse possible à ces questions est l'intégration continue.

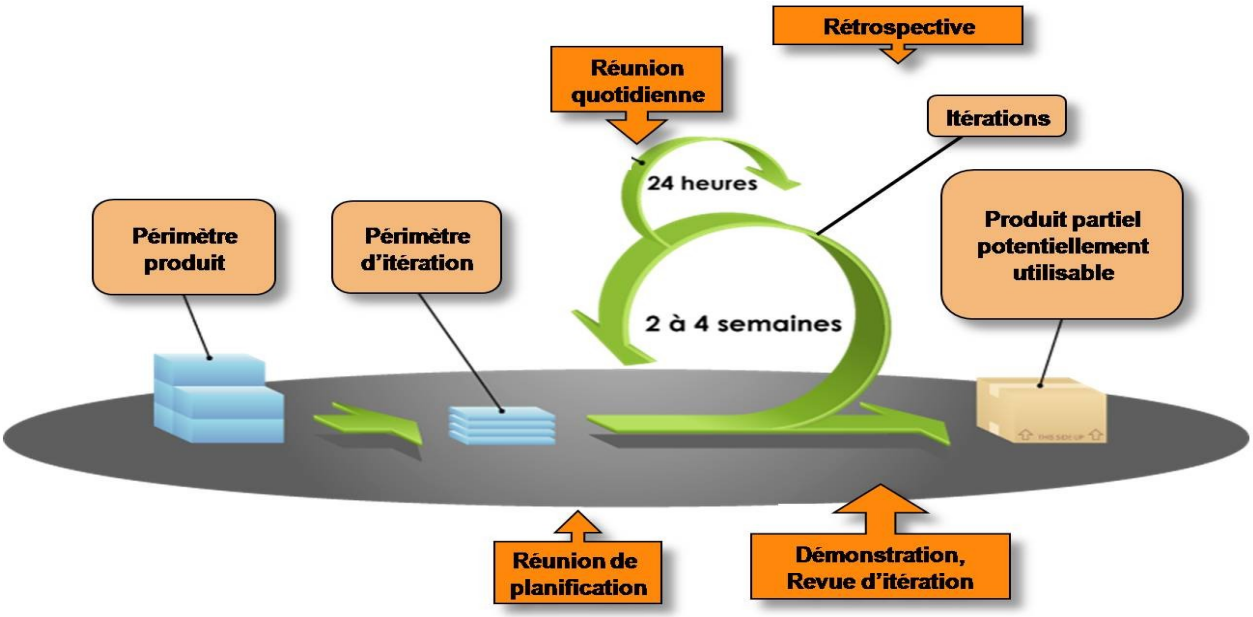

Figure 1: Processus de développement itératif

Le terme d' « intégration continue » vient d'une des douze pratiques prônée par l'eXtreme Programming (XP)[1], une méthode agile de gestion de projet informatique adaptée aux équipes réduites avec des besoins changeants. Elle se définie comme une méthode permettant d'améliorer la qualité des logiciels tout en réduisant les risques. L'idée principale est qu'en intégrant au plus tôt les différentes fonctionnalités développées, on réduit l'effort (et le coût) nécessaire à l'intégration de celles-ci. Bien que mise en avant

[1] http://www.extremeprogramming.org/rules/integrateoften.html

par l'eXtreme Programming, l'intégration continue est utilisable, et même recommandée, dans la plupart des méthodes de développement et de gestion de projet informatique. Le schéma ci-dessus présente une gestion de projet agile avec des cycles courts et des intégrations régulières qui nécessitent l'automatisation des tâches.

Afin de réduire les coûts de corrections d'anomalies, les coûts d'intégration et globalement d'améliorer la qualité du logiciel, nous devons être capables de répondre à certaines questions en permanence : suis-je capable de compiler mon logiciel sur toutes les architectures et matériels ciblés ? La suite de tests s'est-elle déroulée avec succès ? La documentation technique est-elle à jour ? Les performances sont elles toujours au rendez-vous ? Cette liste est loin d'être exhaustive.

L'intégration continue est là pour répondre à ces questions. Le concept est simple : à tout instant, nous devons être capable de donner des indicateurs sur la qualité de notre logiciel et donner un retour aux développeurs en cas de défaillance d'un ou plusieurs de ces indicateurs. La mise en place d'un processus d'intégration continue nécessite un minimum d'outillage que tout projet de développement collaboratif se doit de disposer :

- Un **gestionnaire de code source de référence** (également appelé dépôt) où les développeurs viendront apporter leurs contributions et récupérer celles de leurs collègues. Les outils les plus communément utilisés sont Subversion [2] (svn) et Git[3] mais bien d'autres existent.
- Un **processus de build automatisé**. Les outils tels que Maven, Cmake, Qmake, Scons sont les plus répandus.
- Une **batterie de tests**. L'écriture des tests peut s'appuyer sur des outils comme la suite xUnit par exemple.
- Un **outil d'intégration continue** capable de détecter les changements sur le gestionnaire de code source, lancer le processus de build, la suite de tests, divers travaux et de créer un tableau de bord synthétisant les informations. Les plus connus sont Jenkins [2], Hudson [1], Bamboo [3], Cruise Control [4].

L'outil d'intégration continue va alors se charger de détecter les changements opérés sur le dépôt de code source. A chaque changement, il pourra déclencher automatiquement toute une série de travaux (build, suite de tests, génération de la documentation, packaging, etc.). L'équipe de développement pourra visualiser l'évolution des travaux en cours ou passés via une interface web, récupérer les rapports de tests, voir l'évolution de la « santé » du projet. L'outil d'intégration continue est aussi capable de générer des alertes (email, messagerie instantanée) dès qu'il constate une régression ou une anomalie. Il permet d'avertir les développeurs d'une anomalie dans les minutes qui suivent un changement sur le dépôt de code source. Ils peuvent ainsi la corriger rapidement, les changements étant encore frais dans leur tête. Ce mécanisme permet aussi de responsabiliser les développeurs par rapport à l'équipe de développement. Un développeur qui fera lever des alertes se sentira responsable de celles-ci et les corrigera au plus vite pour ne pas impacter le travail de ses collègues. Plus qu'un long discours, il faut essayer l'intégration continue pour s'apercevoir de son intérêt. Une fois adopté, difficile de s'en passer.

4 Mise en place d'une plateforme d'intégration continue à Inria

Le développement logiciel fait partie de l'activité scientifique d'Inria au même titre que la production de connaissances. Cette activité de développement augmente sensiblement d'année en année. Les logiciels sont généralement développés de manière collaborative : plusieurs personnes n'appartenant pas forcément à la même organisation (académiques, industriels) travaillent ensemble à la réalisation d'un même projet. Ces logiciels sont destinés à un public varié : la communauté scientifique, un industriel, le grand public, une communauté open-source, etc. Comme vu précédemment, il est nécessaire de s'assurer de la qualité et de la fiabilité du logiciel que l'on va livrer à nos utilisateurs, même si ceux-ci sont nos collègues. Les scientifiques l'ont compris et une demande croissante d'offre d'intégration continue est apparue. C'est dans ce cadre qu'est né le projet de mettre à disposition des chercheurs et ingénieurs une plateforme d'intégration continue mutualisée à Inria. Cet outil sera un complément idéal de la forge Inria [4] qui héberge déjà 3000 projets et compte plus de 11000 utilisateurs. Le public visé sera un peu plus restreint car la forge Inria héberge également des projets relatifs à la rédaction collaborative d'articles mais montre que les utilisateurs potentiels sont nombreux.

[2] http://subversion.apache.org/

[3] http://git-scm.com/

[4] http://gforge.inria.fr

4.1 Les besoins

Les besoins exprimés par les scientifiques sont variés. L'intérêt principal d'une plateforme commune est de soulager les scientifiques de toutes les tâches d'administration système et réseau (authentification, sauvegarde, mises à jour de sécurité, etc.) en leur offrant un service clés en main. Le coût de mise en place et de maintenance d'un service et du système sur lequel il s'exécute n'est pas négligeable. L'idée d'un service disponible à la demande permet de leur faire gagner du temps et favorise l'adoption de l'outil. Cela permet également de proposer un service de qualité à l'ensemble de l'institut.

Compte tenu de la grande variété des projets de développement, il faut que les utilisateurs puissent utiliser l'outil d'intégration continue de façon publique, i.e. le bilan de santé du projet est visible depuis l'internet, ou alors le conserver en interne pour l'équipe de développement. Les langages de développement (C++, Java, Smalltalk, Ocaml ...), les outils de build (Maven, Cmake, Qmake, scripts …) utilisés sont très différents d'un projet à l'autre. Le choix de l'outil d'intégration continue devait prendre en compte ce critère : est-il facilement utilisable avec d'autres outils, d'autres langages? Mais ce n'est pas le seul critère. Quel est le service rendu par cet outil comparativement aux autres? Quels sont les outils déjà utilisés par les scientifiques? Y a t'il une communauté active autour de cet outil? Pour toutes ces raisons, notre choix s'est porté sur le logiciel Hudson qui a en premier lieu été développé avec une forte orientation vers Java. Malgré tout, l'adoption d'Hudson dépasse aujourd'hui la communauté Java et de nombreux plugins sont disponibles pour d'autres langages et outils.

Une fois les besoins cernés, nous avons mis en place un prototype basé sur l'outil Hudson.

4.2 Première plateforme : Preuve de Concept

Les impératifs qui ont guidé la première plateforme étaient les suivants : fournir rapidement aux utilisateurs des instances de la plateforme d'intégration continue Hudson, utiliser l'authentification centralisée d'Inria, fournir une couche d'isolation entre les différents projets et s'intégrer dans les pratiques d'administration système et réseau existantes. Cette plateforme avait pour but de sensibiliser les personnes qui produisent des logiciels chez Inria à l'intérêt de l'intégration continue, en leur démontrant ses bienfaits par la pratique.

Au final, cette plateforme a été bâtie sur les briques logicielles suivantes : Apache Tomcat pour la gestion des containers Java, Hudson en tant que logiciel d'intégration continue. Côté infrastructure, la virtualisation est réalisée avec Xen, l'administration centralisée sous Cfengine [5], et Apache en frontal web.

Des développements spécifiques ont concerné l'enregistrement de projets, la gestion du workflow de mise à disposition d'un serveur avec des validations manuelles, la gestion des utilisateurs pour chaque instance Hudson en lien avec l'annuaire LDAP (demande d'enregistrement publique avec validation par une interface dédiée aux administrateurs).

Comme indiqué auparavant, notre cahier des charges incluait la nécessité d'utiliser l'annuaire d'authentification central d'Inria pour l'authentification des utilisateurs. Des scripts ont donc été intégrés dans la plateforme pour permettre l'enregistrement des utilisateurs dans l'annuaire à partir de pages web (php, AJAX). Deux cas de figures se présentent pour les utilisateurs. L'utilisateur existe déjà dans l'annuaire, auquel cas il est ajouté aux groupes nécessaires. S'il n'appartient pas à un projet d'Inria, un compte d'invité est créé spécifiquement. Il ne donne accès qu'à la plateforme d'Intégration Continue (branche spécifique de l'annuaire LDAP). Seule l'authentification LDAP est autorisée pour les instances Hudson. Les administrateurs des instances ont un accès complet à la configuration de leur instance. Un script qui manipule le fichier de configuration (XML) de l'instance Hudson vérifie la configuration de l'authentification sans modifier le reste de la configuration.

Pour ce qui est de LDAP, nous séparons les utilisateurs en deux groupes : soit pour un rôle d'utilisateur soit un rôle d'administrateur. Hudson réalise une transformation des noms de groupes de l'annuaire dans sa configuration. Le format est le suivant : ROLE_<NOM DU GROUPE> (majuscules obligatoires). Par exemple, nous définissons pour un projet « projet », nous définissons deux groupes dans LDAP « projet-ic » et « projet-ic-adm » qui sont écrits dans la configuration d'Hudson, respectivement, « ROLE_PROJET-IC » et « ROLE_PROJET-IC-ADM ».

Le workflow de mise à disposition d'un serveur d'intégration continue est le suivant :

- demande de mise à disposition d'un serveur d'intégration continue via un portail web dédié ;
- génération de demandes d'accès à un projet par un utilisateur et de création des ressources nécessaires (par mail) ;

- création d'une machine virtuelle dédiée au projet (allocation d'adresse IP, enregistrement DNS, création de la machine virtuelle, installation du système d'exploitation) ;
- ouvertures pare-feu nécessaires pour les accès aux différentes forges utilisées par le projet ;
- enregistrement de la machine virtuelle dans l'outil d'administration centralisée ;
- configuration automatisée des différents composants (accès SSH, Apache, Tomcat, Hudson) ;
- mise à disposition de l'utilisateur avec accompagnement ;

Dans ce workflow de nombreuses étapes nécessitent une intervention humaine, parfois lourdes, et la mise à disposition d'un serveur d'intégration continue nécessitait beaucoup de disponibilité de la part des intervenants. La réponse à une demande pouvait donc être plus au moins longue suivant la charge du service gestionnaire de l'infrastructure.

4.3 Bilan de la première plateforme

L'usage du serveur d'intégration continue et de ses plugins est varié. Certains projets utilisent peu de plugins et ne changent pas de version d'Hudson. D'autres au contraire utilisent de nombreux plugins et doivent faire évoluer la version du serveur d'intégration continue en fonction de ces ajouts. Les derniers qui veulent toujours la dernière fonctionnalité introduite et mettent à jour très régulièrement Hudson. Après analyse de ces besoins, il est apparu indispensable de proposer à chaque projet de pouvoir installer la version d'Hudson qu'ils désirent (et non pas imposer une version unique à tous) et leur proposer un système de mise à jour simplifié. Enfin, Hudson n'est pas prévu pour être multi projet. Le choix initial pour assurer une isolation entre les différents projets a été de créer une machine virtuelle par projet. Ce choix s'est révélé trop lourd à gérer sur le long terme.

La mise en place de cette plateforme aura requis approximativement 0,2 ETP sur 4 mois. La phase d'expérimentation a duré 1 an avec trois projets pilotes qui ont permis de convaincre les scientifiques du bienfondé de la démarche.

4.4 Seconde plateforme d'intégration continue : Passage à l'échelle

Sur les bases du bilan précédent, une nouvelle plateforme a été mise en place, l'écosystème d'outils dans lesquels intégrer la plateforme ayant également évolué, il a fallu s'y adapter tout en réutilisant le maximum de composants de la première plateforme. Parmi les briques qui ont changées et qui ont eu un impact sur la mise en place de la plateforme, la gestion de configurations est passée de Cfengine à Puppet [6]. De plus, le projet Jenkins [2] est apparu (suite au fork par la communauté de Hudson pour des raisons de propriété intellectuelle). Cette arrivée nous a conduit à faire un choix dans l'implémentation : il faut pouvoir supporter différents types de logiciels d'intégration continue. La seconde plateforme d'intégration continue permettra donc d'instancier au choix Hudson ou Jenkins.

Un des problèmes soulevés par la première plateforme était le nombre d'opérations nécessaires à la mise en place d'un serveur d'intégration continue. Il faut donc pouvoir réduire au maximum l'intervention humaine dans le processus. Ces interventions coûteuses découlaient essentiellement de la création d'un serveur dédié à chaque projet (réservation d'une adresse dans le DNS, demande de certificat pour l'accès sécurisé, ouverture d'ACLs réseau pour les accès HTTP(S) au serveur notamment).

Les automatisations dans le processus de mise à disposition des instances sont essentiellement effectuées grâce à Puppet. Une fois les demandes acceptées par le pilotage du projet, l'instance est mise à disposition de l'utilisateur en une seule ligne de configuration. Le logiciel Puppet permet de décrire les actions à effectuer dans un langage spécifique (DSL). Ces actions sont par exemple la copie d'un fichier, l'installation d'un logiciel, l'exécution d'un script. Les actions sont ensuite adaptées au système cible en fonction de faits (facts dans l'écosystème de Puppet) qui dirigent vers la bonne manière de remplir la tâche (quelle commande permet la création d'un répertoire, quel est l'utilitaire d'installation de logiciel). La version utilisée de Puppet est la 0.25.4, elle ne permet pas l'utilisation de classes paramétrées, donc la déclaration d'une nouvelle instance utilise le concept de définition (qui pourrait être apparentée à une fonction) afin de définir le nom de l'instance. Du fait de la forte proximité entre les projets Hudson et Jenkins, les actions à effectuer pour mettre en place une instance d'un des deux projets sont quasiment identiques, donc une grande partie du code développé dans Puppet pour l'intégration continue est factorisé. L'architecture a été revue afin de réduire le nombre de serveurs impliqués dans la plateforme. Le principe choisi pour la première plateforme de dédier un serveur par instance d'intégration continue était le principal motif qui entraînait les retards de mise en production de l'instance. Dans la solution retenue, un serveur est utilisé comme frontal Apache pour l'ensemble des projets d'intégration continue. Puis un serveur est déployé par logiciel d'intégration continue utilisé, à savoir un serveur qui héberge les instances d'Hudson et un serveur pour celles de Jenkins.

Le serveur frontal héberge l'ensemble des composants liés au workflow de mise à disposition des instances de serveurs d'intégration continue : la base de données utilisée pour le suivi des projets et les développements web.

Hudson et Jenkins sont des applications Java accessibles par HTTP grâce à un container de Servlets. Par défaut, ils sont packagés avec le container Winstone. Nous avons fait le choix d'utiliser ce container plutôt qu'Apache Tomcat car il fournit un moyen simple de démarrer plusieurs instances en concurrence, utilisant des ports d'accès différents, sans avoir à gérer et modifier des fichiers de configuration externes et il n'y avait pas d'expertise Tomcat disponible dans l'équipe en charge du projet. La suite du projet montrera qu'il n'y a pas eu d'impact en terme de performance sur les instances d'Hudson ou de Jenkins.

Le déploiement d'une instance du logiciel d'intégration continue consiste en la copie en local de l'archive de l'application web (fichier war) dans un répertoire dédié. Un script en charge de la gestion de l'arrêt et démarrage de l'instance ainsi que de scripts permettant de vérifier régulièrement que la configuration nécessaire à l'authentification sur le serveur LDAP est conforme sont également mis en place. Afin de permettre la gestion complète de la version déployée par chaque projet, l'archive de l'application web n'est copiée qu'à l'initialisation. Ensuite, dans l'interface d'administration développée spécifiquement, chaque administrateur a la possibilité de déployer la version du logiciel d'intégration continue qu'il souhaite pour son projet. Pour ce faire, nous déployons 2 instances pour chaque projet, celle de production et une de qualification. Notre processus de mise à jour déploie tout d'abord la version choisie en qualification en recopiant la configuration de l'instance de production. L'administrateur peut alors s'assurer qu'il n'y a pas de régressions ou de problèmes de compatibilité avec des plugins installés. Il pourra ensuite pousser la version en qualification sur l'instance en production. Les versions disponibles pour la mise à jour sont contrôlées afin de n'offrir que des versions qui ne souffrent pas de problèmes avérés de sécurité.

A la différence de la première plateforme, nous avons fait le choix de proposer un point d'entrée unique à tous les projets sur le schéma https://ci.lille.inria.fr/<projet>. L'accès aux instances d'Hudson ou Jenkins, réparties chacune sur leur serveur respectif, est transparent à l'utilisateur final grâce à l'utilisation de règles de réécriture au sein d'Apache. La mise en place de ces règles sur le serveur frontal est permise par l'utilisation de la fonctionnalité de ressources exportées de Puppet. Ainsi, la définition d'une nouvelle instance est faite dans la description du nœud qui l'hébergera. Le serveur frontal, à chaque exécution de Puppet, récupère la liste des nouvelles instances disponibles et met à jour sa configuration Apache en fonction.

Enfin, l'ensemble des applications web (« war » et développements spécifiques) sont partagées en NFS sur l'ensemble des serveurs de l'infrastructure par un serveur de fichiers hautement disponible. Ceci combiné à la gestion de configuration, permet, par exemple, une reprise rapide en cas de panne d'un des serveurs qui héberge les instances de serveurs d'intégration continue.

Le schéma complet de la plateforme actuelle est le suivant :

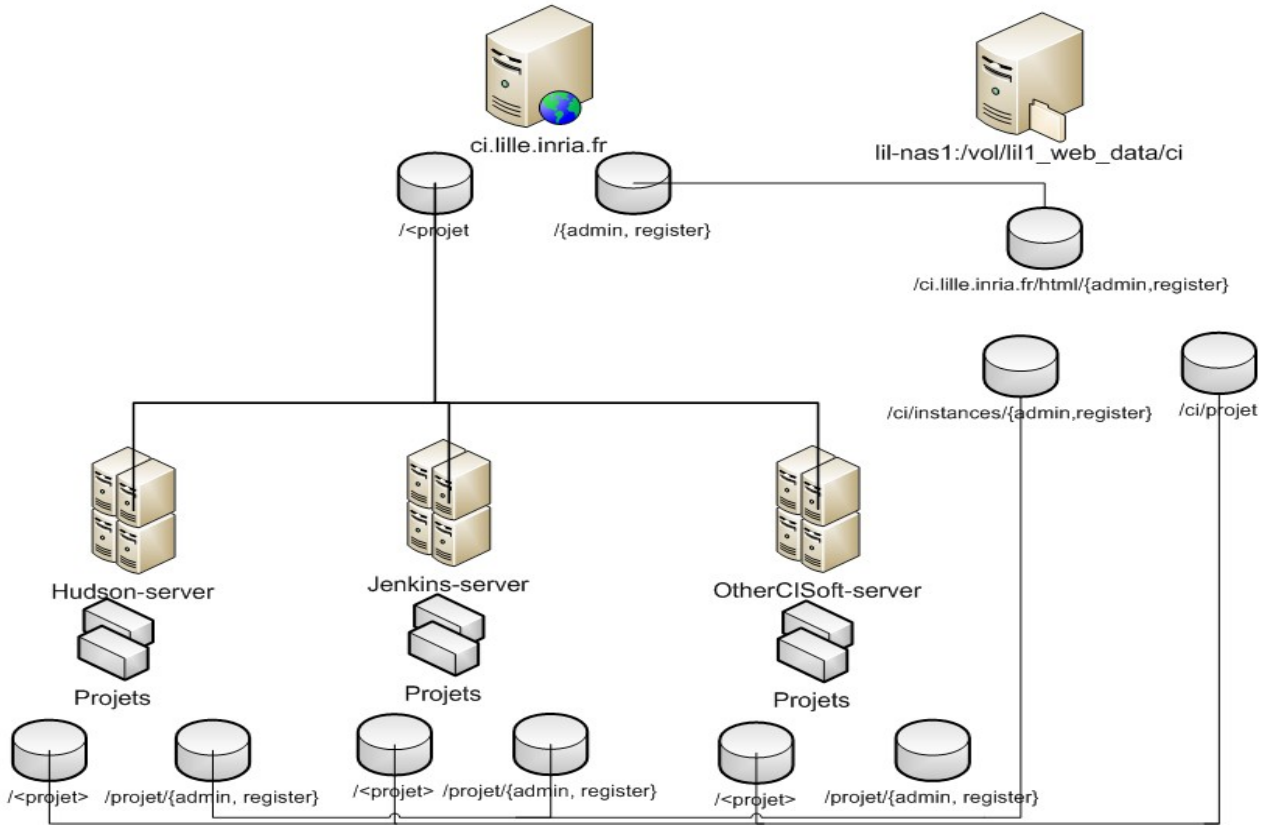

Figure 2 : Eléments intervenant dans la plateforme d'Intégration Continue.

5 Bilan et perspectives

Cette seconde plateforme apporte un réel progrès, aussi bien du point de vue des administrateurs (moins d'opérations manuelles) que pour les utilisateurs (délai de mise à disposition raccourcis, plus grande flexibilité offerte). Au total, sa mise en place a requis, approximativement, l'équivalent de 1,2 ETP sur 1 mois, en bénéficiant des développements déjà effectués. Entrée en production en septembre 2011, la montée en charge progressive permettra de voir si elle répond à toutes les attentes. Dans un premier lieu, les projets existants sur la première plateforme sont portés sur la seconde, des discussions sont en cours pour ajouter d'autres projets. Fort de ce succès, Inria a pour objectif d'en faire un service opérationnel pour tous les scientifiques d'Inria en 2013. .

Une composante importante d'un service d'intégration continue est la mise à disposition de machines de compilation de manière statique ou dynamique. Ce type de service entraîne d'autres enjeux, notamment en terme de rationalisation des équipements. Un projet de ce type va être lancé dans les prochains mois pour accompagner la plateforme Inria.

Une perspective à moyen terme serait de mieux intégrer ce service à la forge Inria. Enfin, étendre nos outils pour offrir des métriques sur le code (quantitatives, qualitatives) comme le permet le logiciel Sonar serait un plus.

6 Bibliographie

[1] Hudson Extensible Continuous Integration Server, http://hudson-ci.org/

[2] Jenkins, an extendable open source continuous integration server, http://jenkins-ci.org/

[3] Bamboo, Continuous Integration and Release Management, http://www.atlassian.com/software/bamboo/overview

[4] CruiseControl, http://cruisecontrol.sourceforge.net/

[5] CfEngine, http://cfengine.com/

[6] Puppet, http://puppetlabs.com/