DevOps : développement & production

A travers ce dossier, Xebia vous guide afin que « DevOps » ne soit pas un buzzword de plus pour vous.

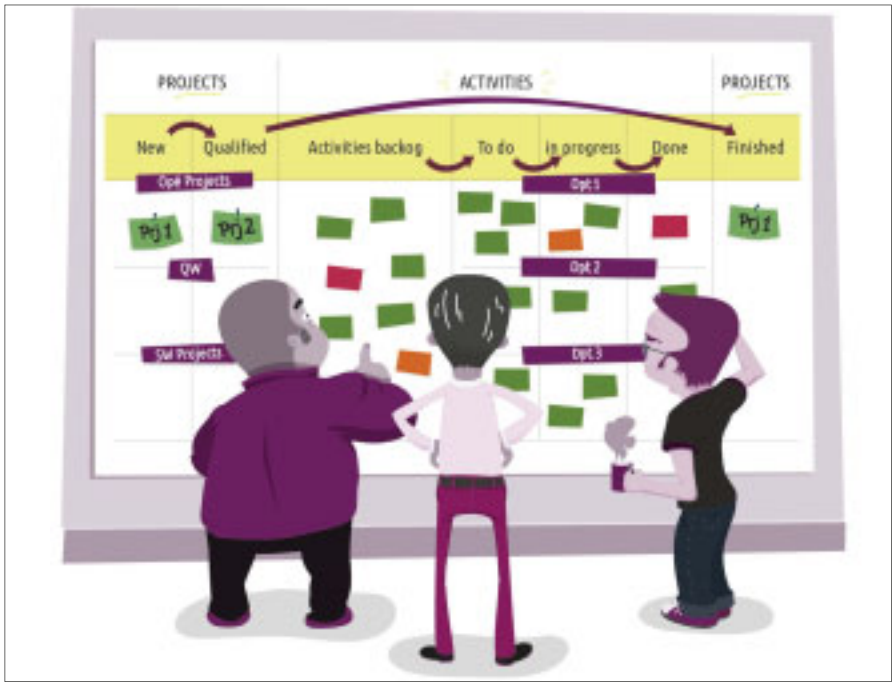


Illustration Xebia

Depuis quelques années, l'agilité connaît un essor grandissant dans les DSI. Aujourd'hui, l'éventail de méthodologies est large et adaptable à tous les contextes. Les valeurs clés de l'agilité sont la collaboration, la transparence, la culture de la qualité, l'adaptation et la simplicité. De l'équipe Scrum à la Feature Team, les transformations agiles ont convaincu de nombreuses directions de SI par leur efficacité au sein des équipes de développement. L'agilité, quand elle n'est appliquée qu'au développement, se trouve néanmoins freinée par les tâches d'exploitation qui surviennent après chaque livraison.
Le but du mouvement DevOps est d'abattre cette frontière en créant une synergie entre les équipes d'exploitation (Ops) et les équipes de développement (Devs). Traditionnellement, Ops et Devs ont des objectifs antagonistes : les uns sont les garants de la stabilité et de la disponibilité des systèmes, là où les autres sont employés à l'évolution de ces derniers. Cela crée un clivage entre ces équipes, appelées à travailler ensemble et à tendre vers un même objectif : délivrer le meilleur logiciel aux clients de l'entreprise. De plus, il est courant que les Ops aient des notions de développement, et les Devs, d'exploitation. Cela entraîne immanquablement des conflits. Qu'elles soient bloquantes ou non, ces frictions dégradent la productivité. Plusieurs symptômes sont révélateurs de ces problèmes :

- En cas de crise, combien de temps faut-il pour lever une alerte, récupérer les logs, les analyser puis identifier la défaillance ? Combien de temps pour livrer un correctif en production ? La rapidité d'exécution de ces actions est fortement liée à la qualité de la coopération entre équipes.
- La fréquence et la simplicité des mises en production sont également des indices révélateurs. Les Ops sont rarement impliqués au démarrage des projets. Il s'ensuit des délais allongés entre la livraison des applications et celle des machines qui serviront de socle. Cela aboutit souvent à une détection de problèmes en pré-production, seul environnement suffisamment proche de la production pour une validation. Les open spaces grouillent d'anecdotes du même acabit. Nous allons vous guider dans notre vision d'une démarche d'amélioration, pour pallier ces problèmes efficacement.

COOPÉRER

La culture et l'organisation de votre entreprise sont un pilier de votre transformation vers DevOps. Les leitmotifs de DevOps peuvent néanmoins paraître galvaudés : qui ne se targue pas de vouloir travailler de manière collaborative et en toute transparence ? De ce fait, quelles sont les implications concrètes sur l'organisation d'un département exploitation derrière le mouvement DevOps ? Est-ce une réelle rupture ou un simple effet de mode ? Les organisations de type « You build it, you run it » sont sans compromis et semblent un objectif utopique à atteindre pour beaucoup. Voyons ici quelques axes de travail et les outils que vous pouvez utiliser pour démarrer rapidement une démarche DevOps sans avoir à révolutionner votre organisation.

Restaurer la confiance

L'activité des Ops est jalonnée de nombreuses crises. Qu'elles soient liées à des pannes matérielles ou à une mise à jour provoquant une instabilité du système. Les Ops sont familiers de la gestion de crise, avec ses horaires à rallonge et ses diagnostics parfois laborieux. Les bureaux d'Ops résonnent d'histoires croustillantes sur ces Devs totalement inconscients qui mettent la production en péril avec des déploiements de binaires hasardeux. L'un des premiers axes de progrès est donc de restaurer la confiance dans un contexte de defiance mutuelle. Pour ce faire, deux outils sont à votre disposition. D'une part, la conclusion d'une mise en production doit être l'objet d'une réunion de retour d'expérience (appelée rétrospective) entre Devs et Ops afin de capitaliser sur les bonnes pratiques et identifier les points d'amélioration. Une pratique régulière des rétrospectives devrait diminuer sensiblement le nombre de crises et restaurer la confiance entre ces deux protagonistes.


Illustration Xebia

Concevoir des produits « Ops-ready »

Les projets agiles manquent souvent d'une vision Ops très tôt dans la conception du produit. Ce manque est induit par le fait que le Product Owner, propriétaire du backlog, a une vision métier centrée sur les fonctionnalités à délivrer à l'utilisateur. La dimension Ops sera, au mieux sous-estimée, au pire passée sous silence. Les Ops sont des parties prenantes dont le PO doit tenir compte. Il est essentiel de les impliquer dès la constitution du Product

Backlog. Le Product Backlog représente tout ce qui apporte de la valeur au produit. Des exigences non fonctionnelles comme certificat apportent de la valeur, elles doivent apparaître dans le backlog. D'autres besoins Ops se traduiront par des critères d'acceptation des user stories. Par exemple, PCA ou sécurité.

Anticiper l'activité ops

Une autre difficulté pour les Ops est d'anticiper les demandes des Devs et les livraisons de nouveaux binaires. Il n'est pas rare de voir des Devs se plaindre du manque de réactivité des Ops, et des Ops se plaindre du manque d'organisation des Devs qui ne savent pas anticiper leurs besoins. Quelles qu'en soient les raisons, les échanges sont souvent tendus et les urgences sont la norme, plus que l'exception. Il suffit parfois de partager un simple outil de management visuel pour créer de la transparence sur les travaux en cours cote Devs et permettre aux Ops d'anticiper les demandes. Il est également possible d'inviter des Ops dans des revues de sprint afin qu'ils s'approprient le produit avant de voir arriver le livrable pour mise en production. Cette dernière pratique est à utiliser avec parcimonie car les Ops sont, en général, peu intéressés par le contenu fonctionnel d'un sprint.

FLUIDIFIER

Maintenant qu'une véritable coopération entre équipes s'est installée, les cérémonies agiles mêlant Devs et Ops sont prolifiques. Chacun écoute les problèmes de l'autre. Cela donne l'occasion d'harmoniser l'ensemble :

- Développer en accord avec la cible qu'est l'environnement de production.
- Adapter, dans la mesure du possible, le SI aux besoins des développeurs.

De plus en plus de structures ont réussi cette transition. Du rapprochement entre les équipes sont nés de nombreux outils d'automatisation. Le but de ces outils, que nous allons voir plus en détail, est de transformer le SI en un self-service pour développeurs, géré par les Ops.

Usine logicielle

Première étape de tout projet : la mise en place des outils de développements. Il faudra à une équipe :

Un dépôt de code versionné

Git ou Mercurial rempliront le job sans difficulté. Ils permettent tous deux de faire des commits locaux aux machines des développeurs avant d'envoyer leur travail par paquet au dépôt central. Cela leur donnera la possibilité de faire des commits plus petits et donc, de réduire grandement les problèmes de

Illustration Xebia

merge en intégrant leur code au tronc commun.

Il doit être possible pour n'importe quel développeur de créer de nouveaux dépôts simplement. Pour cela, des outils comme GitLab, GitBlit ou RhodeCode vous offriront une interface Web de gestion des utilisateurs et des dépôts de code.

Un serveur d'intégration continue

Jenkins, de par sa simplicité d'usage et ses nombreux plugins, est tres populaire. Vous pourriez également regarder TeamCity ou Bamboo pour déterminer quel outil vous convient le mieux. Tous ces outils sont également disponibles en Service Cloud pour vous éviter de les mettre en place en interne et gagner en vitesse de mise en place.

Un dépôt d'artefact

En fin de build, les artefacts produits doivent être stockes pour servir aux processus de déploiement. Là, le choix est extensif en fonction de votre approche :

Si votre intégration continue délivre des fichiers WAR ou un autre artefact type Java, un dépôt Maven tel que Archiva ou Nexus est à conseiller.

Si vous souhaitez pousser l'intégration continue à produire des paquets systèmes Linux (des .deb ou des .rpm par exemple), il faudra alors adopter le mode de distribution ad-hoc.

Dans tous les cas, l'essentiel est d'avoir au moins l'équivalent d'un serveur FTP, ou vous stockerez les artefacts en les rangeant correctement par numéros de version.

Infrastructure

La mise en place de l'infrastructure doit passer par des outils de provisionnement automatique, comme VMWare Cloud Template ou AWS CloudFormation. Cela implique bien sur d'avoir un socle de virtualisation qui permette de scripter le démarrage des ressources.

- La virtualisation des différents environnements, du développement à la production, garantit :
- Une bonne réactivité dans le provisionning,
- La reproductibilité totale de l'installation des plateformes,
- La possibilité d'obtenir à moindre cout de l'élasticité sur la plateforme de production,
- La possibilité de recréer à la demande une plateforme depuis zéro (pour les disaster recovery ou bien les tests de charge par exemple).

En utilisant cette approche depuis la phase de développement, et en considérant que les Ops participent à l'équipe, on obtiendra une infrastructure versionnée, déployable à la demande dans les environnements choisis (développement, recette ou production) en restant isoproduction. La mise en place de répartiteurs de charge, de groupes de scaling

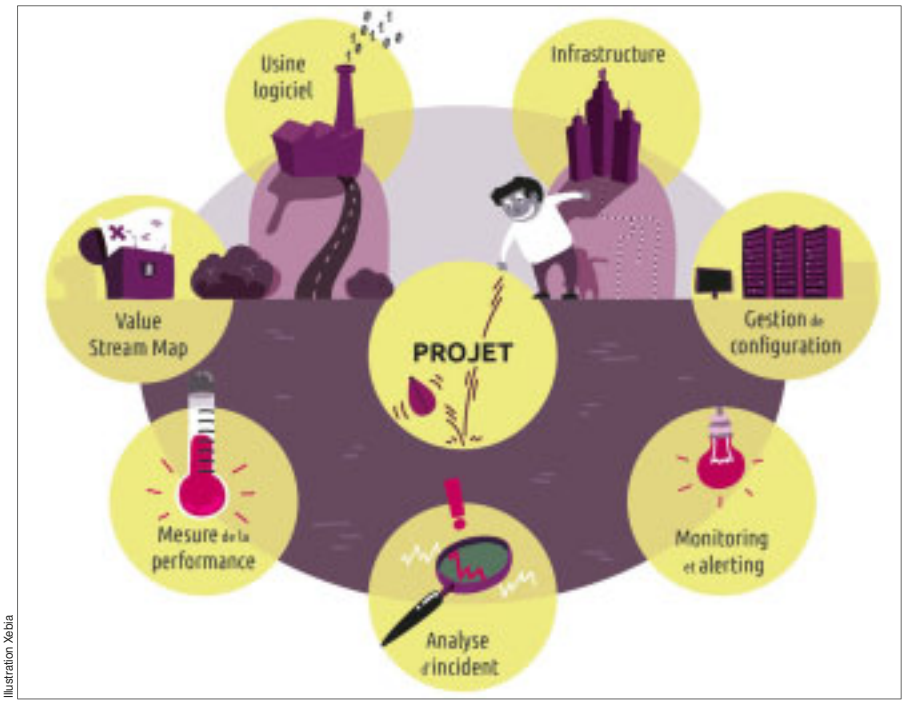


Illustration Xebia

automatiques et de groupes de sécurité doit également faire partie de ce provisionning et être rédigée par toute l'équipe, pour repartir la connaissance.

Gestion de configuration

Avant même de songer à centraliser les fichiers de configuration des serveurs, il faut que l'application développée soit apte à recevoir de la configuration externe. Toutes les valeurs nécessaires doivent pouvoir être surchargées. Une discussion entre Devs et Ops est essentielle pour déterminer si l'application est suffisamment configurable. À la charge des Devs de rédiger la configuration correspondant à leur environnement de validation.

Une fois cette analyse faite, il faudra intégrer un système de centralisation de configuration, tel Ansible, Chef, Puppet ou CFEngine. Ces outils sont bases sur un système de serveur central pilotant la configuration des serveurs clients. Ils peuvent également servir à piloter de façon uniforme un ensemble de machines pour effectuer par exemple, une mise à jour. Tous ces outils sont de tres bonne qualité et utilises par des acteurs majeurs du Web; à vous de consulter la littérature pour voir lequel vous appelle.

Le monitoring et l'alerting

L'expérience de mise en place des méthodes agiles au sein des équipes Devs a montré que l'augmentation de la visibilité sur l'avancement du travail induit des progrès à différents niveaux :

- Afficher les résultats des tests automatises et les indicateurs de qualité de code sur les écrans d'un open space responsabilise les développeurs.
- Certaines équipes opérationnelles barrent, sur un gigantesque calendrier, les jours sans incident notable afin d'apporter un feedback positif à l'équipe sur son travail.
- Certaines startups affichent, sur de grands écrans, des l'entrée de leurs locaux, les chiffres de vente du jour. C'est un moyen de focaliser l'attention de l'ensemble de l'entreprise sur le but final, qui est le fonctionnement du groupe et non d'une équipe isolée.

Pour être efficace, ces outils de supervision doivent pouvoir accueillir un nombre illimité de mesures : Devs et Ops doivent pouvoir exposer autant de métriques qu'ils le souhaitent, sans mettre en péril la production et sans être freinés par des considérations de coût ou de matériel. Il faut donc faciliter la mise en place de nouvelles "sondes" au sein du système. Parmi diverses solutions, nous avons sélectionné un outil open source, baptise Graphite, qui :

- Est facilement clusterisable, donc pourvu d'un stockage virtuellement extensible à l'infini,
- Peut conserver un très grand nombre de points sous forme de séries temporelles, et les agréger lorsqu'une granularité fine n'est plus nécessaire,
- Permet d'appliquer sur ces points des formules mathématiques avancées.

Ce logiciel est un simple collecteur, il est non-intrusif vis à vis des applications de production. De nombreux plugins Graphite sont disponibles : ces plugins vont du traçage des valeurs système à l'interception de trappe Nagios. Couplé avec un autre outil open source, JmxTrans, il devient tres facile de grapher des variables exposées par JMX, le système préférentiel pour exposer des mesures en Java.

On ne peut évidemment pas rester toute la journée les yeux rives sur les compteurs, c'est pourquoi il est bon d'avoir un système d'alerte efficace. Classiquement, on déclenche des alertes sur des passages de seuils, comme "le CPU de telle machine dépasse 90 %". Avec un couple Graphite-JmxTrans, il est maintenant possible de mettre en place des alertes sur des tendances, comme "le CPU de telle machine dépasse en moyenne les 90 % depuis plus de 10 minutes". Comme la collecte des métriques est centralisée, il est même possible de croiser plusieurs sources d'information pour déclencher des alertes tres ciblées, comme « sur les 10 dernières minutes, le CPU dépasse en moyenne les 90%, le pool d'accès aux services externes est sature et le temps de parcours client au moment du paiement est supérieur de 50% au temps moyen constate en fonctionnement normal ». Si cette dernière alerte se déclenche, alors il est temps d'avertir mon partenaire paiement que son service est dégradé. Cet alerting peut se faire via des outils classiques comme Nagios ou, bien sur, un logiciel open source dédié tel Seyren.

Ce monitoring technique et fonctionnel a un autre effet bénéfique : il fait entrer l'entreprise dans la culture de la mesure, qui se résume à cette phrase : « Vous ne pouvez améliorer que ce que vous pouvez mesurer ».

Analyse d'incident

Le monitoring est également utile au-delà de la visualisation de données métier. Les logs générés par les briques de votre SI doivent être traités avec une attention particulière. Ils sont le matériau de base de toute analyse d'incident. Le problème technique qui se pose rapidement (et qui enlise de nombreux SI) est de savoir exploiter les logs générés par un grand nombre de machines, à travers tout le SI. Sans stratégie de collecte et d'archivage centralisée, vos équipes vont devoir commencer par déterminer sur quelle machine un problème a eu lieu, avant même de pouvoir commencer à l'analyser.

Pour parer à cela, il convient de mettre en place sur chaque machine un agent de collecte qui enverra les logs émis par une machine sur un serveur central pour tri, archivage et indexation. De cette façon, n'importe quel membre d'équipe saura immédiatement venir consulter les logs et les métriques de monitoring, collectes depuis toutes les instances de serveur.

Une solution technique puissante, qui évitera d'avoir à faire des recherches exactes dans des fichiers de log de plusieurs Go, est le triptyque LogStash, ElasticSearch, Kibana :

- LogStash vous fournira les agents locaux et la collecte centralisée,
- ElasticSearch est un moteur d'indexation et de recherche full-text qui servira de stockage,
- Kibana est une interface Web pour requêter sur l'ensemble des logs collectes et construire des tableaux de bords clairs.

La mesure de la performance

Il est assez courant de rencontrer, dans un SI, des applications éclatées, réparties sur différents serveurs, ou des actions utilisateurs mettant en jeu plusieurs applications. Dans ce genre de cas, le monitoring de chaque machine, même avec une consultation centralisée, ne suffit plus à travailler efficacement. L'analyse du comportement du système complexe que devient alors le SI nécessite un outillage plus puissant.

Deux approches complémentaires permettent de lutter contre l'effet "Works for me" qui peut apparaitre :

- Pratiquer des tests de charge continus,
- Mesurer les performances réelles en production.

Value stream map

Tous les outils et méthodes que nous venons de passer en revue vont donner beaucoup plus de pouvoir à vos équipes sur votre SI. Ils seront en mesure d'observer, de comprendre, de diagnostiquer et de réparer au plus vite. On peut néanmoins aller plus loin encore. Un autre outil permet d'analyser le fonctionnement de vos équipes plutôt que celui de votre SI : la Value Stream Map. Cet outil issu du Lean Management permet de calculer l'efficience de votre processus et d'identifier les points d'amélioration avec une vue d'ensemble. Dans

un contexte DevOps, il s'agira d'identifier précisément toutes les étapes nécessaires au déploiement d'un binaire en production depuis sa livraison par les Devs. Nous associons à chaque étape sa durée de travail effectif et le temps de latence constate avec l'étape précédente. Le ratio entre le cumul des temps de travail effectifs et le cumul des temps d'attente vous donne votre efficience globale. Il s'agira ensuite de cibler votre effort d'amélioration pour augmenter ce ratio.

LIVRER

Maintenant que Devs et Ops sont équipés d'outils efficaces et ont confiance en leur mise en production, ils n'attendent plus qu'une chose : des binaires. Jusqu'ici, nous avons lié Devs et Ops autour de la connaissance du système au sens large.
Si les équipes arrivent à ce stade de maturité, elles vont commencer à réellement partager un processus bout en bout. Une même équipe aura les connaissances et les outils pour développer une fonctionnalité et la mettre en production. C'est ici la terre promise du Déploiement Continu. Voyons ensemble quelques bonnes pratiques et stratégies de déploiement à considérer pour exploiter votre nouveau SI et vos équipes dopées à la DevOps-amine.

Mise en place du déploiement continu

Il est important que vos équipes définissent leur stratégie de gestion de branche dans le code source. Il faut qu'elles soient conscientes qu'il y aura des patchs à faire sur la version de production et qu'elles doivent pouvoir mettre temporairement de cote leur travail en cours pour résoudre un problème. Deux stratégies ont aujourd'hui fait leurs preuves.

Simple Branching

Le Simple Branching est, comme son nom l'indique, la stratégie la plus simple. On garde une branche dont le code est la version déployée en production. On crée également une branche de développement dans laquelle une nouvelle version de l'application est en cours de réalisation.

Feature Branching

Le Feature branching consiste, quant à lui, à avoir une branche principale contenant la version actuellement en production et plusieurs branches de développement. Pour chaque fonctionnalité en cours de développement, une nouvelle branche est créée. Lorsque son implémentation est terminée et validée par le métier, les modifications sont rapatriées sur la branche principale pour livraison en production.

Plasticité applicative

Toggle Feature Pattern

Le Toggle Feature pattern est un outil de conception logicielle. Les Devs peuvent positionner des interrupteurs dans le code de l'application pour encapsuler des fonctionnalités entières. Ces derniers doivent être activables par configuration et manipulables à chaud par les Ops pour modifier le comportement de l'application.
En protégeant les fonctionnalités en cours de rédaction par le biais de ce mécanisme, il est possible de maintenir le rythme et de déployer des fonctionnalités, même si d'autres sont encore en chantier.

Illustration Xebia

Circuit Breaker Pattern

Outre la ségrégation des fonctionnalités non terminées, toute fonctionnalité reposant sur des services externes doit pouvoir être activée ou désactivée à chaud, sans stopper l'ensemble de l'application.
Le monitoring peut avertir les Ops qu'un service est indisponible, mais il est également possible d'intégrer, directement dans les applications, des mécanismes de protection contre les pannes externes.
Le principe du pattern Circuit Breaker est de conditionner les accès vers des composants externes par des interrupteurs logiques qui détectent les appels en erreur. Quand un seuil (configurable) d'erreurs consécutives est atteint, l'application passe en mode dégradé et n'assure plus la fonctionnalité qui dépend du service en panne. Pendant que l'application est en mode dégradé, elle envoie une alerte au monitoring et déclenche une procédure pour vérifier régulièrement si le service est revenu. Dès que cette procédure détecte un retour du service, l'application repasse en mode standard et assure de nouveau la fonctionnalité.

A/B Testing

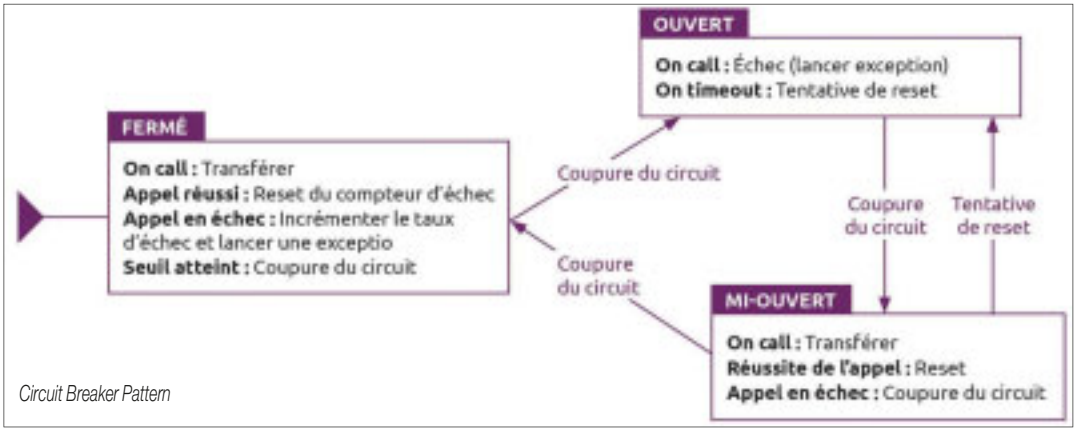
Une fois la technique du Toggle Feature Pattern maîtrisée par vos équipes, il est possible de l'utiliser, non plus seulement pour masquer une fonctionnalité instable, mais aussi pour proposer aux utilisateurs différentes versions d'une même fonctionnalité.
En effet, de plus en plus d'entreprises procèdent à des tests grandeur nature, en production, par échantillonnage. Par exemple, via la mise en place d'un A/B testing : 2 pages d'accueil, au design différent, sont proposées de manière aléatoire à 2 populations d'internautes. Le monitoring permet de dégager des tendances indiquant la page la plus pertinente.
Le principe du A/B Testing n'est pas limité à 2 options et il est toujours possible de jouer sur différentes nuances d'ergonomie, de simplicité de parcours, etc.
La pertinence du A/B Testing doit se mesurer en valeur métier. Typiquement, pour un site marchand, on attribuera le taux de transformation en vente d'une version de l'application. La meilleure version devient la norme et on tente d'autres alternatives sur cette nouvelle base pour améliorer encore l'expérience utilisateur.

Dark Launch

Autre pratique corollaire du Toggle Feature et de l'A/B Testing : le Dark Launch, qui permet de tester des fonctionnalités alternatives considérées comme risquées. On va alors activer une fonctionnalité à une frange des utilisateurs, en se basant sur des certains critères (peu d'utilisateurs, leur type de compte, leur nationalité, leurs préférences, etc.). Une fonctionnalité typiquement considérée comme à risque est la modification du parcours de paiement pour un site marchand. Comme c'est à cette étape que se concentre la majorité des abandons d'achat, il est tres délicat de décider de la modifier. L'utilisation d'un Dark Launch pour confirmer la pertinence d'une modification avant de la généraliser est ici appropriée.

Stratégies de déploiement

Votre projet s'appuie maintenant sur un processus de déploiement bien rodé, au sein duquel l'ensemble des étapes de build, de tests et de provisioning sont automatisées. Vous êtes capable de livrer votre application en continu mais le déploiement en production nécessite toujours un Go/NoGo. L'étape suivante consiste à automatiser le déploiement en production en tant qu'action finale de la chaine de production.
Comme il n'est pas acceptable qu'une



Circuit Breaker Pattern

modification du code aboutisse à des coupures de service à chaque mise en production automatique, il faut mettre en place une stratégie de déploiement sans interruption de service. Différentes techniques peuvent être considérées afin de réussir à diminuer les risques et dépasser les craintes liées à un tel bouleversement dans la manière de livrer une application en production. Le but est de transformer le déploiement en un non-évènement, quelque chose de standard.

Blue/Green Deployment

Cette technique de déploiement sans interruption de service repose sur un système de double run. On maintient 2 environnements de production pleinement fonctionnels. Tous les accès utilisateurs passent par un routeur qui permet la bascule vers l'un ou l'autre de ces environnements. La version vers laquelle pointe le routeur est l'environnement Green, tandis que la partie en stand-by est appelée le Blue. La technique consiste à déployer la nouvelle version sur le Blue, faire toutes les vérifications utiles pour valider le déploiement, puis basculer le trafic du routeur.
Une fois le trafic basculé, l'environnement "nouvelle version" devient le Green, et on peut sereinement mettre à jour le second déploiement pour maintenir l'équilibre entre Blue et Green.

Canary releasing

Dès que votre infrastructure de production commence à prendre de l'ampleur, il peut devenir problématique de rester dans une approche Blue/Green Deployment.
Redonder une grappe de serveurs frontaux, un nuage de middleware et plusieurs bases de données peut rapidement couter tres cher.
Vous pourriez alors être tente par une stratégie nommée Canary Releasing.
Cela consiste à déployer la nouvelle version de son application sur un sous-ensemble des noeuds qui composent votre environnement de production. Cela peut se faire soit en augmentant légèrement le nombre de vos instances durant le déploiement, soit en mettant à jour un ensemble restreint d'instances existantes. Dans tous les cas, il faut être capable de maitriser quelle proportion de votre trafic va être dirigée vers ces instances avec la nouvelle version de votre application. Vous devrez porter une attention particulière à ces instances pour valider que la nouvelle version peut être généralisée. En cas de problème, il vous suffira de couper le trafic entrant sur ces instances pour arrêter de perturber vos utilisateurs. Une fois votre stratégie choisie, vous allez avoir besoin d'un outil pour gérer le déploiement à proprement parler de l'applicatif.

CONCLUSION

Grâce à une implication forte des équipes de développements et des équipes d'opérations, le suivi de l'avancement des projets est assuré de manière transverse fournissant ainsi une vision globale et non plus spécifique à chaque équipe. La richesse des échanges vient améliorer, du point de vue de tous, les solutions finales. Les Devs prennent en compte intuitivement les besoins spécifiques des Ops et vice versa. D'autre part, les outils d'automatisation mis en place simplifient les tâches nécessaires au delivery de bout en bout. À terme, le temps perdu entre les développements et la mise en production diminue régulièrement. La value Stream Map permet de constater cette amélioration d'itération en itération.
L'innovation et la veille technologique se développent dans la DSI. D'un point de vue managérial, le planning des releases obtient des niveaux de confiance inégalés jusqu'alors. Un rythme s'est installé, les mises en productions sont régulières et sécurisées, les interventions d'urgence se raréfient.
Attention, il ne s'agit pas d'une recette miracle! Le changement d'organisation s'est accompagné de nombreuses difficultés, la résistance au changement est toujours forte, Nous avons présenté un grand nombre d'outils destinés aux Devs comme aux Ops, qui peuvent aussi trouver leur utilité dans les équipes métiers. Tous ces systèmes ne serviront néanmoins que tres peu sans une coopération forte entre les équipes et une adhésion aux principes du DevOps.
Merci aux auteurs et aux contributeurs du TechTrends #2 DevOps de Xebia ayant permis la rédaction de cette partie.
Retrouvez ce document dans son intégralité sur http://techtrends.xebia.fr/.

Merci à Clément Rochas, Consultant agile chez Xebia, @crochas

Analyse orientée business de vos logs applicatifs

Dans cet article, nous allons voir comment mettre à profit nos logs applicatifs afin de faire de l'analyse orientée "business" grâce à ElasticSearch, Logstash et Kibana.

Vincent Spiewak
Consultant Java et agile chez Xebia
@vspiewak

A la tête d'un site de vente en ligne fictif, nous essayerons notamment de monitorer l'activité de nos clients, et de répondre aux questions suivantes:

- Quel est le taux de conversion (ratio recherches / ventes) ?
- Quels sont les produits les plus consultés ?
- Où nous situons-nous par rapport à nos objectifs de ventes ?

A faire chez vous

L'ensemble de la démonstration a été automatisée sur une VM Vagrant disponible sur GitHub : https://github.com/vspiewak/elk-programmez-2014.git
Vous devez installer uniquement VirtualBox et Vagrant (disponible gratuitement pour Windows, Linux et OS X). Vous trouverez le script d'installation ainsi que l'ensemble des fichiers de configuration.

Format des logs

Ce paragraphe vous présente le format des logs que nous allons exploiter **Fig.1**. Notre application génère deux types de logs, représentant respectivement une recherche ou un achat dans notre magasin de vente en ligne. Un log de vente contient l'adresse IP et le User-Agent de l'acheteur, son sexe, ainsi que les informations sur le produit acheté (marque, modèle, catégorie, options et prix).
Un log de recherche contient l'adresse IP et le User-Agent du visiteur, et quelques informations sur le produit recherché (parfois la catégorie, parfois la catégorie et la marque, etc).

Logstash

Logstash est un ETL léger permettant de collecter, transformer et charger des logs à l'aide de connecteurs d'entrées, de filtres et de connecteurs de sorties. C'est un véritable couteau suisse qui, en version 1.4.2, vous fournit 41 entrées, 50 filtres, et 55 sorties différentes.
Nous allons voir comment transformer nos lignes de logs illisibles en documents JSON structurés, et les enrichir au passage de précieuses informations (géolocalisation, version du navigateur, etc).

Premiers pas

Un agent Logstash a besoin d'un fichier de configuration pour être exécuté. Voici un exemple de configuration minimaliste : **Fig.2.**

Fig.1

```
07-10-2014 09:51:29.994 [pool-1-thread-1] INFO com.github.vspiewak.loggenerator.SearchRequest
 - id=1&ua=Mozilla/5.0 (Windows NT 5.1) AppleWebKit/534.24 (KHTML, like Gecko) Chrome/11.0.69
6.50 Safari/534.24&ip=83.112.78.237&category=Portable

07-10-2014 09:51:30.002 [pool-1-thread-1] INFO com.github.vspiewak.loggenerator.SellRequest -
 id=3&ua=Mozilla/5.0 (X11; U; Linux x86_64; en-US) AppleWebKit/534.16 (KHTML, like Gecko) Chr
ome/10.0.648.205 Safari/534.16&ip=62.23.215.188&email=client4@gmail.com&sex=F&brand=Apple&nam
e=Mac Mini&model=Mac Mini - Core i7&category=Ordinateur&options=Core i7&price=829.0
```

Fig.3

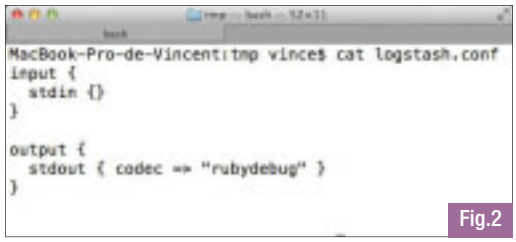


Fig.2

L'entrée standard est utilisée comme "input", la sortie standard est utilisée comme "output" (A noter que nous utilisons le codec "rubydebug" afin que le json soit formaté).
Nous pouvons lancer ensuite l'agent Logstash avec cette configuration, et écrire un message dans l'entrée standard : **Fig.3.**
Logstash a ainsi transformé notre message en un JSON formaté. Logstash ajoute automatiquement les champs @version, @timestamp et host. A noter que le champ @timestamp contient la date de réception du message par Logstash.
Le premier travail va consister à parser nos logs à l'aide du filtre grok...

Filtre Grok

Le filtre Grok nous permet, à l'aide d'expressions régulières, de découper nos logs et leur donner de la sémantique. Logstash vient avec plus d'une centaine d'expressions régulières prédéfinies (disponibles ici : https://github.com/logstash/logstash/tree/master/patterns)
Par exemple, le pattern "COMBINEDAPACHELOG" du fichier grok-patterns permet de parser les lignes de logs d'un serveur Apache.
Vous pouvez utiliser 2 syntaxes :

- %{SYNTAX:SEMANTIC} ou %{SYNTAX:SEMANTIC:TYPE}
- (?<field_name>the pattern here)

La première syntaxe utilise un pattern Logstash prédéfini (ex: INT, NOTSPACE, LOGLEVEL, ...), SEMANTIC étant le nom du champ à mapper. Vous pouvez préciser le type du champ (integer, float, string) via le paramètre TYPE. La deuxième syntaxe vous permet de définir un pattern customisé avec ou sans l'aide de plusieurs patterns Logstash.
Pour nos logs, nous utiliserons le filtre Grok suivant :

```
filter {
  grok {
    match => ["message","(?<log_date>%{MONTHDAY}-%{MONTHNUM}
-%{YEAR} %{HOUR}:%{MINUTE}:%{SECOND}.[0-9]{3}) \[%{NOTSPACE:
thread}\] %{LOGLEVEL:log_level} %{NOTSPACE:classname} - %{GREEDY
DATA:log_msg}"]
    }
}
```

Avec le filtre Grok, logstash découpe nos lignes de log correctement, ajoutant les champs log_date, thread, log_level, classname et log_msg : **Fig.4.**

Filtre date

Le filtre date va permettre de dater notre message correctement. En effet, par défaut logstash valorise le champ @timestamp avec la date courante. Dans le cas de nos logs, nous voulons utiliser la date du log (champ log_date). Nous utiliserons un pattern au format JodaTime (voir :

http://joda-time.sourceforge.net/apidocs/org/joda/time/format/DateTimeFormat.html)

```
date {
    match => ["log_date","dd-MM-YYYY HH:mm:ss.SSS"]
}
```

Filtre KV

Le filtre KV est très utile pour découper un champ au format cle1=valeur2&cle2=valeur2 (paramètres d'une requête http notamment) Nous allons utiliser ce filtre pour découper le champ "log_msg":

```
kv {
    field_split => "&"
    source => "log_msg"
}
```

Filtre mutate

Ajout d'un tag

Le filtre mutate est un filtre "couteau suisse" permettant différentes modifications sur les champs. Nous allons ajouter un tag afin de différencier nos recherches de nos ventes :

```
if [classname] =~ /SellRequest$/ {
    mutate { add_tag => "sell" }
} else if [classname] =~ /SearchRequest$/ {
    mutate { add_tag => "search" }
}
```

Conversion de type

Nous allons convertir les champs id et price respectivement en entier et nombre à virgule flottant :

```
mutate {
    convert => [ "id", "integer" ]
}
mutate {
    convert => [ "price", "float" ]
}
```

Suppression d'un champ

Toujours avec le filtre mutate, nous allons supprimer le champ log_msg que nous avons re-découpé à l'aide du filtre KV :

```
mutate {
     remove_field => [ "log_msg" ]
}
```

Fig.4

Split d'un champ en tableau

Pour finir avec le filtre mutate, nous allons splitter notre champ "options" afin d'avoir un tableau d'options :

```
mutate {
    split => [ "options", "|" ]
}
```

Filtre GeoIP

Le filtre GeoIP permet d'ajouter des informations de géolocalisation via une adresse IP (coordonnées gps, ville et pays notamment). Logstash utilise la base de données GeoLite de Maxmind sous license CCA-ShareAlike 3.0.

```
filter {
    geoip {
        source => "ip"
    }
}
```

Filtre UserAgent

Le filtre UserAgent permet de parser automatiquement un User Agent et d'ajouter des informations comme la version du système d'exploitation ou du navigateur :

```
useragent {
    source => "ua"
    target => "useragent"
    remove_field => [ "ua"]
}
```

Sortie Elasticsearch

Une fois notre log parsé, nous voulons que Logstash l'envoie à Elasticsearch. Rien de plus simple avec la sortie Elasticsearch, puisqu'il suffit simplement de déclarer :

```
output {
    elasticsearch {}
}
```

Note: la sortie est configurée pour se connecter sur l'Elasticsearch locale, avec un mode de transport de type "node" par défaut. Il n'y a donc rien à paramétrer. Logstash nous permet de collecter nos logs, les structurer, les enrichir et enfin de les envoyer à Elasticsearch : Fig.5.

Fig.5

Elasticsearch

Elasticsearch est une base de données full-text orientée document, distribuée et offrant de la haute disponibilité notamment.
Logstash enverra nos logs dans des indices nommés "logstash-YYYY-MM-DD" (équivalent des tables dans le monde SQL traditionnel)
Pour installer Elasticsearch et le plugin “head” (offrant une interface web), rien de plus simple :

```
$ curl -O https://download.elasticsearch.org/elasticsearch/
elasticsearch/elasticsearch-1.3.4.tar.gz
$ tar xvzf elasticsearch-1.3.4.tar.gz
$ cd elasticsearch-1.3.4
$ bin/plugin --install mobz/elasticsearch-head
$ bin/elasticsearch
```

Vous pouvez surveiller l’état de santé d’Elasticsearch à l’adresse : http://localhost:9200/_plugin/head **Fig.6.**

Kibana

Kibana est une application de type “Single Page App” utilisant notamment la stack technique HTML 5, Angular JS et Twitter Bootstrap. Dans notre exemple, nous utiliserons NGiNX, mais tout autre serveur Web capable de servir des fichiers statiques est possible :

```
curl  -O  https://download.elasticsearch.org/kibana/kibana/
kibana-3.1.1.tar.gz
tar xvzf kibana-3.1.1.tar.gz
sudo mv kibana-3.0.0milestone4 /usr/share/nginx/www/kibana
```

Vous pouvez désormais accéder à Kibana à l’url : http://localhost/kibana. **Fig.7.** La page d’accueil de Kibana dispose d’un menu en haut à droite permettant de charger, sauvegarder et partager vos dashboards via :

- Fichier JSON
- ElasticSearch
- Gist
- Permalien

Kibana offre 4 dashboards par défaut :

- Un dashboard générique Logstash (cf. photo),
- Un dashboard générique Elasticsearch,
- Un dashboard non configuré,
- Un dashboard vide.

Un dashboard se "branche" sur tous les index de votre cluster Elasticsearch, un index spécifique, ou les index dont le nom matche un pattern (cas de Logstash notamment).
L’interface reprend le système de grille de Twitter Bootstrap, décomposant les lignes en 12 colonnes, contenant des panels.

Fig.6

Il existe plusieurs types de panels à ajouter et configurer (recherche, période de temps, histogramme, camembert, carte, Markdown, ...)
L’icône “i”, disponible sur chaque panel, permet de visualiser la requête Elasticsearch.

Création d’un dashboard Kibana

Partez d’un Dashboard vierge en cliquant sur le lien "Blank Dashboard". N’oubliez pas de le sauvegarder avant de rafraîchir la page de votre navigateur ! **Fig.8.**
Ouvrez la fenêtre modale "*Dashboard settings*" en cliquant sur la roue crantée dans le coin supérieur droit :

- Onglet "*General*" : choisissez un titre et le thème ("*dark*" ou "*light*"),
- Onglet "*Index*" : sélectionnez "day" pour le champ "*Timestamping*",
- Onglet "*Controls*" : cochez toutes les cases afin de vous donner le maximum d’options,
- Fermez la fenêtre modale en cliquant sur “save”.

Vous pouvez sélectionner une période de temps à afficher dans le menu Time filter (exemple: TimeFilter > Last 15m, TimeFilter > Auto-Refresh > 5s). La barre de recherche utilise le format Lucene Query ou une expression régulière.
En cliquant sur le bouton "+" du champ recherche, créez 3 barres de recherche contenant les requêtes suivantes :

- *
- tags:”search”
- tags:”sell”

Vous pouvez épingler des recherches et leur attribuer un alias.

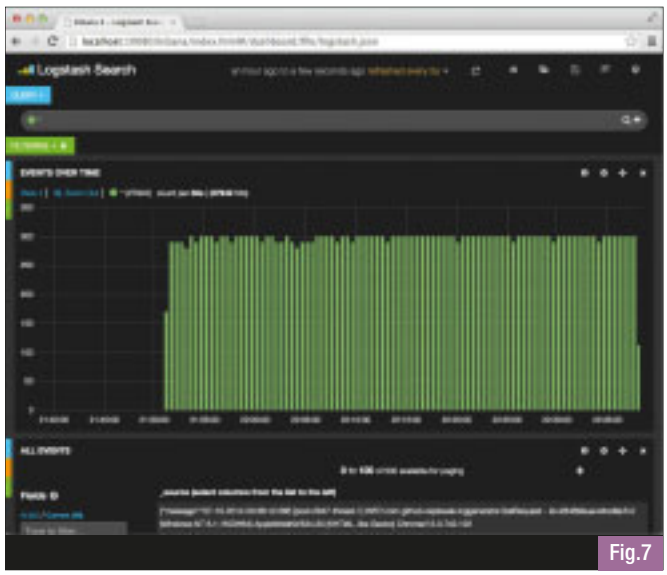


Fig.7

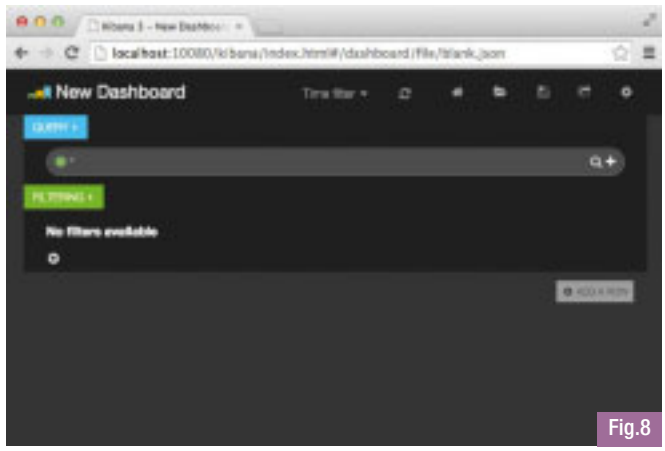


Fig.8

Cliquez sur chaque rond de couleur et sauvegardez les alias suivants :
- All
- Search
- Sell

Ligne 1 - Objectifs, Histogramme et Carte

Ajoutez une ligne en cliquant sur le bouton “Add row”, puis :
- Un panel Histogram de taille 7
- Un panel Bettermap de taille 5

Ligne 2 - Terms

Ajoutez une nouvelle ligne contenant:
- Un panel terms, field “tags”, taille 2
- Un panel terms, field “brand.raw”, length 5, taille 2
- Un panel terms, field “name.raw”, length 5, taille 2
- Un panel terms, field “geoip.city_name.raw”, length 5, taille 2
- Un panel terms, field “useragent.device.raw”, length 5, taille 2
- Un panel terms, field “useragent.name.raw”, length 5, taille 2

Ligne 3 - Données

Ajoutez une dernière ligne contenant uniquement un panel “table” de taille 12.

Drill down

En cliquant sur les différents éléments des deux dernières lignes, vous créez des filtres dynamiquement (icônes en forme de loupe ou de croix). Vous retrouverez ces filtres en dessous des barres de recherches, vous permettant de les éditer ou les désactiver. Fig.9.

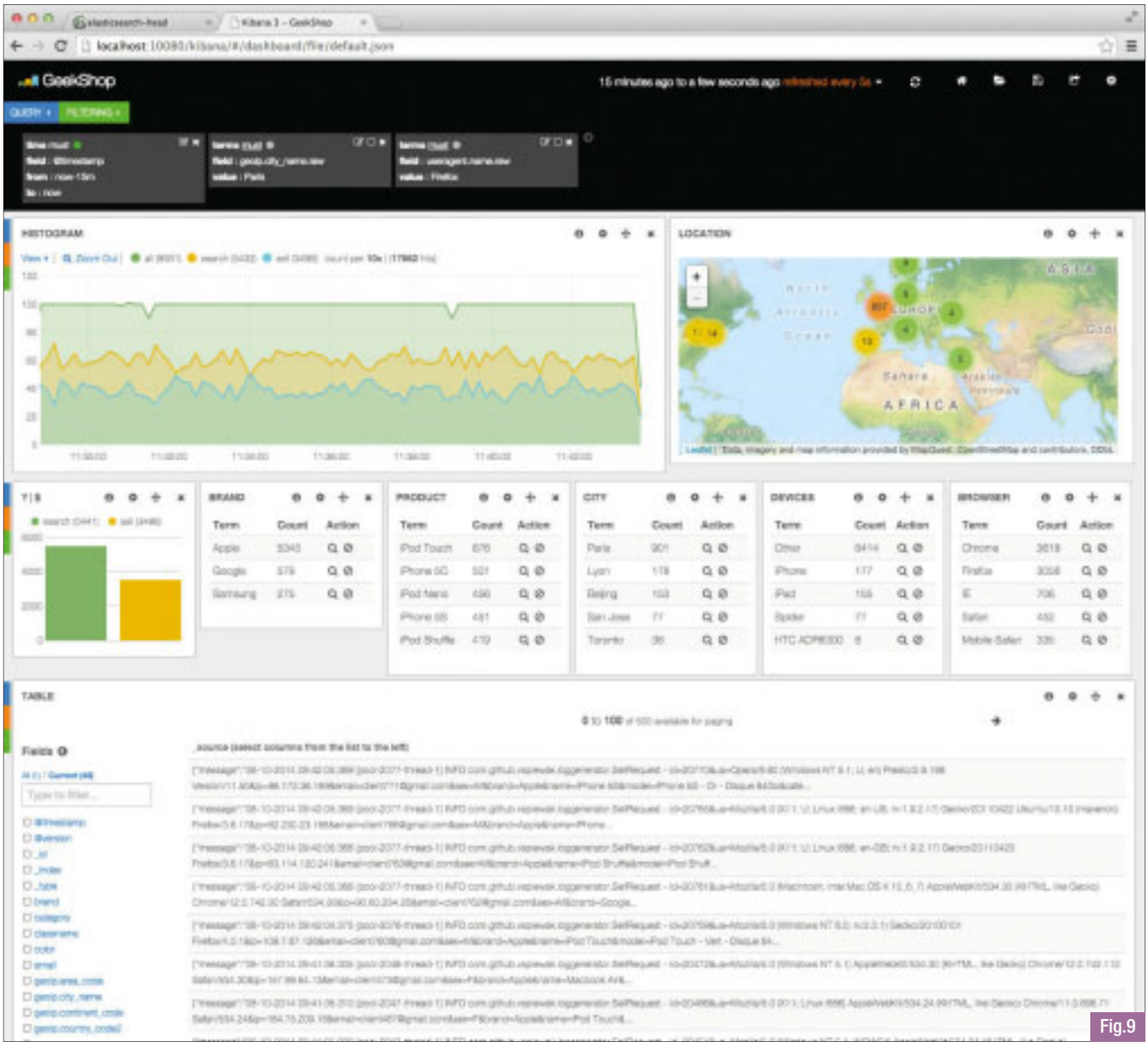


Fig.9

Test Driven Infrastructure avec Chef

Chef est un outil de gestion de configuration permettant de décrire dans un langage clair et compréhensible l'ensemble de son infrastructure ("Infrastructure as code"). Chef permet ainsi d'installer et de maintenir l'ensemble des serveurs et cela de manière automatique.

Emmanel Sciara
Consultant agile chez Xebia
@esciara

Matthieu Nantern
Consultant chez Xebia
@mNantern

Ce tutoriel montre, à travers l'installation de lighttpd qui servira une page comprenant le texte « Wonderstuff Design is a boutique graphics design agency. », un cas d'utilisation simple de Chef. Mais comme tout code il est très important de le tester afin de s'assurer de sa maintenabilité et de son évolutivité.

Cet exemple montre pas à pas comment monter un environnement pour faire du Test Driven Infrastructure avec l'outil de provisioning Chef, en suivant l'exemple du livre Test-Driven Infrastructure with Chef, 2nd Edition by Stephen Nelson-Smith.

Ce cas pratique est destiné aux développeurs débutants ou expérimentés dans la création de cookbooks Chef. Il faut à minima une compréhension des concepts introduits par Chef.

En temps normal, il faut s'assurer que les tests créés ne passent pas avant d'écrire le code qui va les faire passer.

Les outils utilisés dans cet article sont (par ordre d'apparition) :

- Ruby 1.9.x
- Bundler
- VirtualBox
- Vagrant
- Berkshelf
- Test-Kitchen
- ServerSpec
- ChefSpec

PRÉPARER SON ENVIRONNEMENT DE DÉVELOPPEMENT

Installation

L'installation complète de l'environnement de développement se déroule en 7 étapes :

- La première étape pour installer un environnement sain pour le développement de cookbooks est d'installer Ruby. Pour cela suivez l'article très complet : http://misheska.com/blog/2013/12/26/set-up-a-sane-ruby-cookbook-authoring-environment-for-chef/ . Nous allons ici travailler sur OS X.
- Installer ensuite VirtualBox en téléchargant l'application sur https://www.virtualbox.org/wiki/Downloads (version utilisée : 4.3.10)
- Installer Vagrant en téléchargant l'application sur http://www.vagrantup.com/downloads.html (version utilisée : 1.5.3). Vagrant permettra, avec l'aide de Virtualbox, de lancer automatiquement des machines virtuelles afin d'exécuter nos tests d'intégration.
- Les dépendances entre les différents cookbooks seront gérées avec Berkshelf (version 3.1.3). Pour plus d'information voir http://berkshelf.com/. Ajouter Berkshelf au fichier Gemfile que vous devez créer dans le répertoire racine du cookbook. Berkshelf sera alors automatiquement installé par Bundler lors d'un bundle install. Nous allons utiliser le même processus pour tous les autres gems et outils de ce tutoriel.

```
$ vim Gemfile
```

- Ajouter le contenu suivant :

```
source 'https://rubygems.org'

gem 'berkshelf', '3.1.3'
```

- Nous allons ensuite utiliser Test-Kitchen (version 1.2.1, http://kitchen.ci/) qui permet de créer les VM avec Vagrant et lancer les tests automatiquement. Nous utiliserons son plugin kitchen-vagrant (version 0.15.0) permettant à Test-Kitchen de piloter le cycle de vie des VM en utilisant Vagrant. Ajouter Test-Kitchen au Gemfile ainsi que son plugin kitchen-vagrant :

```
gem 'test-kitchen', '1.2.1'
gem 'kitchen-vagrant', '0.15.0'
```

- Il ne reste plus qu'à installer tous les composants avec Bundler :

```
$ bundle install
```

Création du squelette de cookbook

Créer le fichier metadata.rb dans le répertoire racine :

```
$ vim metadata.rb
```

Contenu de metadata.rb :

```
name "wonderstuff"
version "0.1.0"
```

Créer le squelette de cookbook avec berks init (ne pas écraser le Gemfile que nous venons de créer) :

```
$ bundle exec berks init
```

Terminer le squelette :

```
$ mkdir recipes
$ touch recipes/default.rb
```

LANCER DES TESTS À L'INTÉRIEUR D'UN NOEUD AVEC BUSSER

Busser est un framework de setup et d'exécution de tests sur les nœuds créés par Test-Kitchen. Il utilise une architecture de plugin permettant d'ajouter le support de différents outils/stratégies de test tels que MiniTest, Cucumber, Bash, etc.

L'installation de Busser sur le nœud et l'exécution des tests sont

complètement prises en charge par Test-Kitchen. Par défaut, Test-Kitchen va regarder dans le répertoire test/integration pour trouver les fichiers de test à lancer. Ce répertoire est organisé comme suit :

- Directement sous test/intégration se trouveront des répertoires dont les noms définiront les suites de tests. Les suites de tests à lancer sont déclarées dans le .kitchen.yml sous suites: - name: <nom de la suite>. Pour plus d'informations sur .kitchen.yml se référer à http://docs.opscode.com/config_yml_kitchen.html.
- Sous chaque répertoire de suites de tests se trouveront des répertoires dont les noms sont ceux des plugins Busser utilisés.
- Sous chaque répertoire de plugin Busser se trouveront les fichiers de tests à lancer sur les nœuds.

Organisation des répertoires de suites de tests à lancer par Busser :

```
test/integratio
  |_ do_something <-- suite name (more on this later)
     |_ cucumber <-- busser (AKA the "tester")
        |_ my_test_files <-- a test
```

Pour plus d'informations sur Busser, se référer à https://github.com/test-kitchen/test-kitchen/wiki/Getting-Started#structure.

ECRITURE D'UN TEST D'INTÉGRATION UTILISANT SPECSERVER

Introduction à ServerSpec

ServerSpec permet d'écrire des tests RSpec permettant de tester que vos serveurs sont correctement configurés.
ServerSpec teste l'état des serveurs par accès SSH. Pas besoin d'y installer un agent. De plus, Busser se charge entièrement de tout le cycle d'exécution des tests sur les noeuds créés par Test-Kitchen.
Plus d'informations disponibles sur http://serverspec.org/.

Configuration de ServerSpec

ServerSpec a besoin de savoir le type d'OS sur lequel il va effectuer les tests avant de lancer ces derniers. Pour cela, créons un fichier de support spec_helper.rb qui fera ce travail. Il servira de référence ensuite dans tous les fichiers de tests effectués :

```
$ mkdir test/integration/default/serverspec
$ vim test/integration/default/serverspec/spec_helper.rb
```

Contenu de spec_helper.rb :

```
# encoding: UTF-8
require 'serverspec'
require 'pathname'

set :backend, :exec
```

Ecriture du test ServerSpec

Créez les répertoires qui vont permettre la prise en charge des tests par Busser et créez le fichier contenant le test.

```
$ mkdir test/integration/default/serverspec/localhost
$ vim test/integration/default/serverspec/localhost/readable
_services_spec.rb
```

Contenu de fichier readable_services_spec.rb :

```
# encoding: UTF-8
require 'spec_helper'

describe 'Wonderstuff Design' do

  it 'should install the lighttpd package' do
    expect(package 'lighttpd').to be_installed
  end

  it 'should enable and start the lighttpd service' do
    expect(service 'lighttpd').to be_enabled
    expect(service 'lighttpd').to be_running
  end

  it 'should render the Wonderstuff Design web page' do
    expect(file('/var/www/index.html')).to be_file
    expect(file('/var/www/index.html')).to contain 'Wonderstuff
Design is a boutique graphics design agency.'
  end

end
```

Vous pouvez vérifier que les tests se lancent et échouent en lançant la commande :

```
$ bundle exec kitchen verify ubuntu
```

ECRITURE D'UN TEST UNITAIRE CHEFSPEC

Introduction à ChefSpec

ChefSpec est un framework de tests unitaires permettant de tester des cookbooks Chef. Il simplifie l'écriture de tests permettant d'obtenir un feedback rapide sur les changements de cookbooks sans avoir besoin de machine virtuelle ni de serveurs dans le Cloud.
ChefSpec est une extension de Rspec. Il lance votre cookbook localement en utilisant Chef Solo sans avoir besoin de faire converger un noeud. Cela a deux avantages principaux :

- C'est très rapide !
- Vos tests peuvent faire varier les attributs des noeuds, OS, les résultats de recherche pour vérifier le comportement des cookbooks dans diverses conditions.

Plus d'information disponible sur https://github.com/sethvargo/chefspec ou encore http://docs.opscode.com/chefspec.html.

Installation de ChefSpec

Ajouter ChefSpec au Gemfile :

```
gem 'chefspec', '4.0.1'
```

Lancer l'installation du Gem :

```
$ bundle update
```

Configuration de ChefSpec

Les tests ChefSpec se trouvent dans le répertoire spec. On peut dire à ChefSpec de faire les tests en simulant un OS spécifique. Nous allons faire cela dans cet exemple particulier en nous cantonnant à Ubuntu 12.04. Pour cela, nous allons spécifier l'OS de manière globale en utilisant le fichier de support spec_helper.rb. Il est également possible de lancer les tests en simulant différents OS grâce à Rake. Pour plus d'information sur la spécification de l'OS cible de ChefSpec, se référer à https://github.com/sethvargo/chefspec#configuration .
Créez le répertoire spec et le fichier spec_helper.rb :

```
$ mkdir spec
$ vim spec/spec_helper.rb
```

Contenu de spec_helper.rb :

```
# encoding: UTF-8
require 'chefspec'
require 'chefspec/berkshelf'

RSpec.configure do |config|
  config.platform = 'ubuntu'
  config.version = '12.04'
end
```

Ecriture du test ChefSpec

Créez les répertoires et le fichier contenant le test.

```
$ mkdir -p spec/unit/recipes/
$ vim spec/unit/recipes/default_spec.rb
```

Contenu de fichier default_spec.rb :

```
# encoding: UTF-8
require 'spec_helper'

describe 'wonderstuff::default' do
  let(:chef_run) do
    runner = ChefSpec::Runner.new(
      log_level: :error
    )
    Chef::Config.force_logger true
    runner.converge('recipe[wonderstuff::default]')
  end

  it 'installs the lighttpd package' do
    expect(chef_run).to install_package('lighttpd')
  end

  it 'creates a webpage to be served' do
    expect(chef_run).to render_file('/var/www/index.html').
with_content('Wonderstuff Design is a boutique graphics
design agency.')
  end

  it 'starts the lighttpd service' do
    expect(chef_run).to start_service('lighttpd')
  end

  it 'enables the lighttpd service' do
    expect(chef_run).to enable_service('lighttpd')
  end
end
```

Vérifiez que les tests se lancent et échouent en lançant la commande :

```
$ bundle exec rspec -fd
```

ECRITURE DU COOKBOOK

Maintenant que tous les tests sont en place, il ne reste plus qu'à écrire le cookbook.

```
$ vim recipes/default.rb
```

Contenu de fichier default.rb :

```
# encoding: UTF-8
#
# Cookbook Name:: wonderstuff
# Recipe:: default
#
# Copyright (C) 2014 Emmanuel Sciara
#
# All rights reserved - Do Not Redistribute
#

package 'lighttpd'

service 'lighttpd' do
   action [:enable, :start]
end

cookbook_file '/var/www/index.html' do
   source 'wonderstuff.html'
end
```

Cette recette utilise un fichier html comme ressource qu'il copie sur le serveur.

```
$ mkdir -p files/default/
$ vim files/default/wonderstuff.html
```

Contenu de fichier wonderstuff.html :

```
<html>
  <body>
    <p>Wonderstuff Design is a boutique graphics design agency.</p>
  </body>
</html>
```

Donnez une adresse IP privée à votre VM vagrant afin de pouvoir accéder à la page html du serveur. Pour cela modifier le fichier .kitchen.yml :

```
$ vim .kitchen.yml
```

Remplacez la portion du fichier .kitchen.yml comme ci-dessous :

```
suites:
  - name: default
    driver:
      network:
        - ["private_network", {ip: "192.168.33.40"}]
    run_list:
      - recipe[wonderstuff::default]
    attributes:
```

Détruisez la VM existante pour que le changement d'IP soit pris en compte :

```
$ bundle exec kitchen destroy ubuntu
```

Et voilà ! Voyez les tests serverspec passer en lançant la commande suivante :

```
$ bundle exec kitchen verify ubuntu
```

et les tests chefspec avec la commande :

```
$ bundle exec rspec -fd
```

Vous pouvez voir la page sur http://192.168.33.40/

Premiers pas avec XL Deploy

XL Deploy est une solution éditée par XebiaLabs destinée aux Devs et aux Ops. Elle permet de prendre en charge la phase délicate du déploiement de l'application et de sa configuration dans les différents environnements.

Benoit Moussaud
Technical Director Southern Europe - XebiaLabs
@bmoussaud

La première spécificité de la solution XL Deploy est qu'elle est basée sur le modèle UDM (Unified Deployment Model). Ce modèle stipule que pour avoir une application déployée (DeployedApplication), il faut créer un déploiement qui prend en entrée :

- Un **package** sous la responsabilité des Devs. Il regroupe l'ensemble des éléments qui constituent une version d'une application. Ces éléments sont :
  - Des *artefacts*, les fichiers ou les répertoires de fichiers : .exe, .war, fichiers SQL, fichier de configuration .properties etc.
  - Des *ressources*, les éléments de configuration tels que les datasources, virtualhost etc.
  - Des *méta-données* telles le nom de l'application et sa version.

Note : Le package doit toujours être complet, avec l'ensemble des éléments.

- Un **environnement** sous la responsabilité des Ops. Il est caractérisé par :
  - Un ensemble de *containers,* des éléments d'infrastructure ou de middleware décrits du point de vue du déploiement (machine, serveur d'application, serveur web, base de données...) sur lesquels on va déployer l'application,
  - Un ensemble de *dictionnaires* qui décrivent la configuration à appliquer. Exemple : username, password, répertoire, ports, placeholders à remplacer dans les fichiers de configuration.

La seconde spécificité de XL Deploy est la génération automatique des plans de déploiement. Basé sur la spécification de ce déploiement (un package, un environnement, des dictionnaires), sur l'état initial (comme une précédente version déjà déployée), et l'état cible (je veux déployer cette version sur cet environnement avec cette configuration), le moteur « Autoflow » de XL Deploy calcule et applique les différentes étapes à effectuer sur les différents middlewares afin de refléter le nouvel état. La connaissance des actions à effectuer est définie dans le serveur XL Deploy sous forme de règles de déploiement qui seront appliquées quel que soit l'environnement (l'ensemble des équipes partage donc les mêmes règles de déploiement). Une règle s'exprime de la façon suivante : si un élément du package est créé (modifié ou supprimé) alors il faut effectuer les actions suivantes.

Exemple : si un fichier WAR ciblé sur un serveur Tomcat est modifié, alors il faut

- Arrêter le serveur Tomcat,
- Supprimer l'ancienne version du fichier WAR,
- Déposer le nouveau fichier WAR,
- Démarrer le serveur Tomcat.

XebiaLabs propose des ensembles des règles prêtes à l'emploi pour les principaux middleware Java (IBM WebSphere, Oracle Weblogic, Apache Tomcat,RedHat JBoss), le monde Microsoft (IIS, BizTalk, Services Windows), les serveurs Web, les bases de données, des ESB (WebSphere MQ, Oracle Service Bus, Tibco) et les load-balancers (F5, NetScaler).

La dernière spécificité d'XL Deploy est sa capacité à se connecter aux machines distantes en utilisant les protocoles standard (SSH dans le monde Unix et WinRM dans le monde Windows). Il n'est donc pas nécessaire d'installer et de configurer un agent propriétaire sur l'ensemble du parc de serveurs. Nous allons maintenant décrire les différentes actions à réaliser pour déployer une application Java (une application Microsoft, par exemple .NET, suivra les mêmes principes).

Création d'un package

Tout d'abord il faut créer le package de notre application. Nous allons partir d'une application Java classique. Elle est composée de 2 fichiers War (PetClinic.war et PetClinic-Backend.war) – type jee.War, une définition d'une DataSource - type tomcat.DataSourceSpec et d'un répertoire qui contient des fichiers de configuration – type tomcat.ConfigurationFolder. Le fichier *Manifest* correspondant à cette description est le suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<udm.DeploymentPackage version="1.0-20140829-182124" application
="xldeploy-petclinic-tomcat">
  <deployables>
    <jee.War name="petclinic" file="PetClinic.war" />
    <jee.War name="petclinic-backend" file="PetClinic-Backend.war" />
    <tomcat.DataSourceSpec name="petclinicDS">
<jndiName>jndi/pet</jndiName>
      <username>{{DATABASE_USERNAME}}</username>
      <driverClassName>com.mysql.jdbc.Driver</driverClassName>
      <password>{{DATABASE_PASSWORD}}</password>
      <url>{{DATABASE_URL}}</url>
    </tomcat.DataSourceSpec>
    <tomcat.ConfigurationFolder name="configuration" file="config" />
  </deployables>
</udm.DeploymentPackage>
```

Le package doit être indépendant de l'environnement cible. Les éléments paramétrables sont déclarés en utilisant un *placeholder* avec le format *mustache* {{..}}. Dans l'exemple ci-dessus le username, le password et l'url de la datasource sont respectivement gérés par les placeholder DATABASE_USERNAME, DATABASE_PASSWORD et DATABASE_URL. En revanche le nom JNDI est le même quel que soit l'environnement; il peut donc y être déclaré dans le *Manifest* du package.

La structure du package est donc la suivante :

```
├── PetClinic-Backend.war
├── PetClinic.war
├── configuration
│   └── config
│       ├── log4j.properties
│       ├── petclinic-backend.properties
│       └── petclinic.properties
└── deployit-manifest.xml
```

Ce répertoire est ensuite transformé en une archive (.dar) et importé dans XL Deploy pour y être stocké dans son référentiel. **Fig.1.**

L'application xldeploy-petclinic-tomcat version 1.0-20140829-182124 est importée. Afin de faciliter la création des packages et leur import dans XL Deploy, XebiaLabs propose des intégrations avec les outils de *build* tels que Maven ou MS-Build et les serveurs d'intégration continue comme Jenkins ou TFS.

Définition d'un environnement

Avant de définir l'environnement cible, il faut d'abord décrire l'infrastructure (du point de vue du déploiement). Elle sera composée d'un host 'test-machine' – type overthere.SshHost sur lequel est installé un serveur Tomcat 'tomcat.test' – type tomcat.Server et son host virtuel par défaut tomcat.test.vh – type tomcat.VirtualHost Fig.2.
Ensuite il faut définir un dictionnaire qui portera les valeurs de configuration pour cet environnement.
Pour les données sensibles, XL Deploy propose un autre type de dictionnaire de type 'udm.EncryptedDictionary' dans lequel le mot de passe de la datasource sera défini.
L'environnement 'Test' sera donc composé des 3 *containers* et des 2 dictionnaires qui viennent d'être créés.

Déploiement d'une application

Le déploiement est la mise en relation entre une version d'une application (xldeploy-petclinic-tomcat/1.0-20140829-182124) et un environnement (Test). La phase de mapping va associer chaque élément du package sur un ou plusieurs containers de l'environnement Fig.4.

> **Remarque :** cette modélisation montre que le déploiement concerne un artefact de type 'jee.War 'sur un container de type 'tomcat.VirtualHost' bien loin d'une vision infrastructure où le fichier 'petclinic.war' serait associé à une machine 'test-machine'.

Les propriétés Url, Username et Password de la datasource 'petclinicDS' sont entièrement configurées avec les différentes valeurs proposées par les 2 dictionnaires. Basé sur cette spécification, XL Deploy va générer la tâche de déploiement initiale suivante : Fig.5. Lors d'une mise à jour avec une nouvelle version, XL Deploy va analyser les éléments modifiés (par exemple un seul des 2 wars) et générer le plan suivant.
Si en même temps, le username de la datasource a été modifié dans le dictionnaire, XL Deploy va le détecter et inclure la re-création de la datasource dans le plan de déploiement Fig.6.
L'exécution de ces tâches par le moteur XL Deploy consiste pour chacune des étapes (Step) à se connecter sur la machine distante en utilisant Ssh ou WinRM et à exécuter la commande comme le ferait un opérateur.

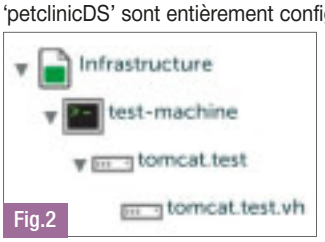


Fig.2

Définition de l'infrastructure

Et mon intégration continue ?

Quel que soit le moteur d'intégration continue choisi (Jenkins, Bamboo, TFS), les équipes ont la plupart du temps intégré une partie déploiement - appelée à la suite de la tâche de Build- via l'utilisation de script custom ou de plugins. Si effectivement cela remplit la fonction de déploiement, celle-ci n'est pas utilisée par l'ensemble des acteurs de la chaîne (de Dev à Ops) et elle n'intègre pas les spécificités des différents environnements (changement de topologies) ou les besoins propres aux plateformes de production (exemple intégration de fonction de sauvegarde ou de gestion des load-balancers). XL Deploy propose une intégration avec les différents serveurs d'intégration continue en proposant les fonctions suivantes : constitution du package de déploiement, import du package, et déclenchement du déploiement sur l'environnement idoine. Rapidement les équipes mettent en place du déploiement continu : le développeur « committe » son code, l'intégration continue se déclenche et construit l'application, lance les tests unitaires et déploie l'application. De cette manière, la procédure de déploiement est testée plusieurs fois par jour coté Devs et elle est rendue plus robuste pour les équipes Ops qui déclencheront leurs déploiements avec l'interface graphique ou l'interface ligne de commande (CLI) via le même outil.
XL Deploy est une solution transverse qui prend non seulement en charge le délicat sujet du déploiement des applications mais permet également aux équipes Devs & Ops de partager un vocabulaire commun (Environnement, Application, Version et Configuration avec les dictionnaires) et ainsi de créer entre les deux équipes une coopération efficace, qui est souvent une des clés de l'approche DevOps !


Fig.5

Tache de déploiement initial

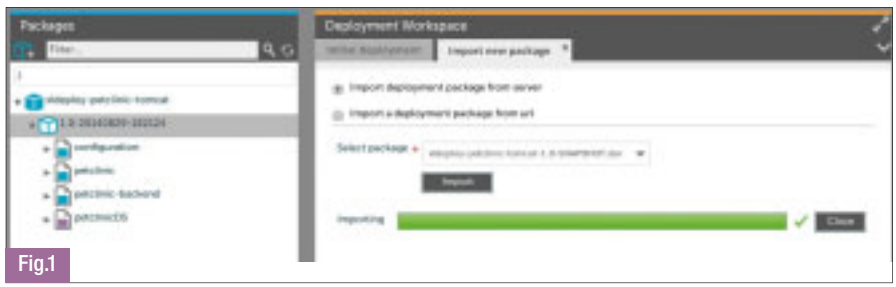


Fig.1

Import d'un package dans XL Deploy


Fig.6

Tache de déploiement (War et Datasource modifiés)


Fig.3

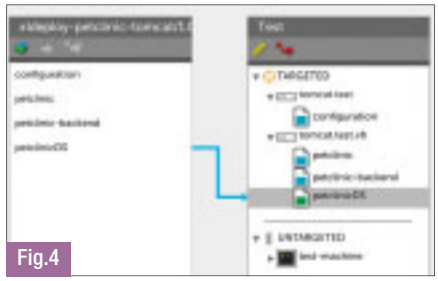


Fig.4

Figure 3 Définition des dictionnaires