ÉTUDES, DÉVELOPPEMENT & INTÉGRATION

DÉCOUVRIR DEVOPS

L'essentiel pour tous les métiers

Stéphane Goudeau
Samuel Metias

Préface de Patrick Debois

DUNOD

© Dunod, 2016

ISBN : 9782100748419

Toutes les marques citées dans cet ouvrage
sont des marques déposées
par leurs propriétaires respectifs.

Illustration de couverture :
Verrières du Grand Palais, Paris, 2016
© Christine Goudeau

Visitez notre site Web : www.dunod.com/

All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher, Dunod, 5 rue Laromiguière, 75005 Paris www.dunod.com.

Préface

Le succès d'une fête ne se juge pas au nombre d'interactions entre ses participants. De même une collaboration efficace ne se mesure pas avec de simples métriques, telles que le nombre de réunions, de tickets, etc. Nous savons tous que plus nous sommes ouverts au dialogue et à la compréhension de l'autre, meilleure sera la qualité des interactions. L'empathie devient alors un état d'esprit.

Ce livre présente DevOps sous différents angles, ceux de tous les participants à la chaîne de production d'un logiciel selon leur place dans cette chaîne. Réexaminer les problèmes communs à travers le regard des autres améliorera votre compréhension et élargira votre vision de ces problèmes.

Le long chemin qui va de la décision initiale à la mise en production est ici décrit comme dans un guide de voyage qui expliquerait les étapes d'un développement DevOps dans une entreprise.

Parfaitement équilibré entre les aspects « technos » et les aspects « processus », il explique les interactions entre les deux. Les uns n'existent pas sans les autres et ils s'influencent mutuellement. Faire tomber les silos organisationnels et techniques est essentiel pour changer les mentalités.

Bonne lecture !

Patrick DEBOIS
Pionnier du mouvement DevOps

Introduction

Lorsque Patrick Debois créé le terme DevOps en 2009, il est certainement loin de se douter qu'il est l'un des pionniers d'un mouvement dont l'influence ne cessera de s'accroître dans le monde des technologies de l'information. Comment imaginer qu'aujourd'hui une démarche dont l'un des objectifs est d'établir une collaboration plus efficace entre les équipes de développement et d'infrastructure ait pu susciter un tel intérêt ?

En appliquant une démarche DevOps, les services informatiques sont aujourd'hui à même de poursuivre leurs activités existantes en toute efficience. Ils peuvent notamment réaliser des tâches qui leur étaient jusqu'alors impossibles. Pour ce faire, ils doivent se renouveler, se transformer et s'adapter. Cette évolution est d'autant plus nécessaire que la nature même des métiers de l'IT a changé. Jadis, il s'agissait *juste* d'écrire du code exempt de bug, de livrer une nouvelle version tous les ans, puis de recommencer. Aujourd'hui, les applications doivent être produites et déployées en continu. À l'ère du cloud, les solutions logicielles doivent être évolutives, disponibles, hyperperformantes avec une latence plus faible et, bien entendu, à moindre coût. DevOps permet aux équipes de développement et d'infrastructure d'être plus réactives face à ces nouvelles exigences.

DevOps est par conséquent le moyen de concrétiser cette évolution, avec comme philosophie, l'idée d'un monde dans lequel chaque composante de l'organisation d'une entreprise collaborerait efficacement pour l'atteinte de mêmes objectifs. Il s'agit tout d'abord de pallier les conséquences négatives issues de la séparation des développeurs et des responsables opérationnels d'une organisation : le fameux *wall of confusion*. En effet, le développeur cible avant tout la production de code qui répond aux exigences fonctionnelles. Il est donc fort possible qu'il ne s'intéresse guère à la maintenance de la solution en fonctionnement opérationnel. À l'inverse, les responsables système ne seront guère enclins à favoriser des changements qu'ils considèrent comme autant de risques pour la stabilité de l'application.

DevOps apporte des réponses à cette problématique par la mise en application de différents concepts d'ordre culturel et technologique. De plus, avec l'avènement du cloud computing, DevOps est devenu un passage obligé : le succès de la mise en œuvre d'une démarche DevOps et la réussite d'une évolution vers le cloud sont intimement liés. Toutefois, le champ d'application de DevOps va bien au-delà du périmètre du cloud.

Samuel et moi sommes donc convaincus de l'intérêt de DevOps. Malgré tous les bienfaits que l'on attend de cette philosophie, nombreux sont ceux qui s'interrogent encore sur la nature exacte de cette démarche et qui hésitent encore à l'adopter, et c'est la raison pour laquelle nous avons souhaité rédiger cet ouvrage.

Nous nous proposons de vous faire découvrir notre vision de DevOps et de son rôle dans la transformation digitale des organisations. Nous présenterons les principes de sa mise en application et les illustrerons avec différents exemples de mise en œuvre, au sein de

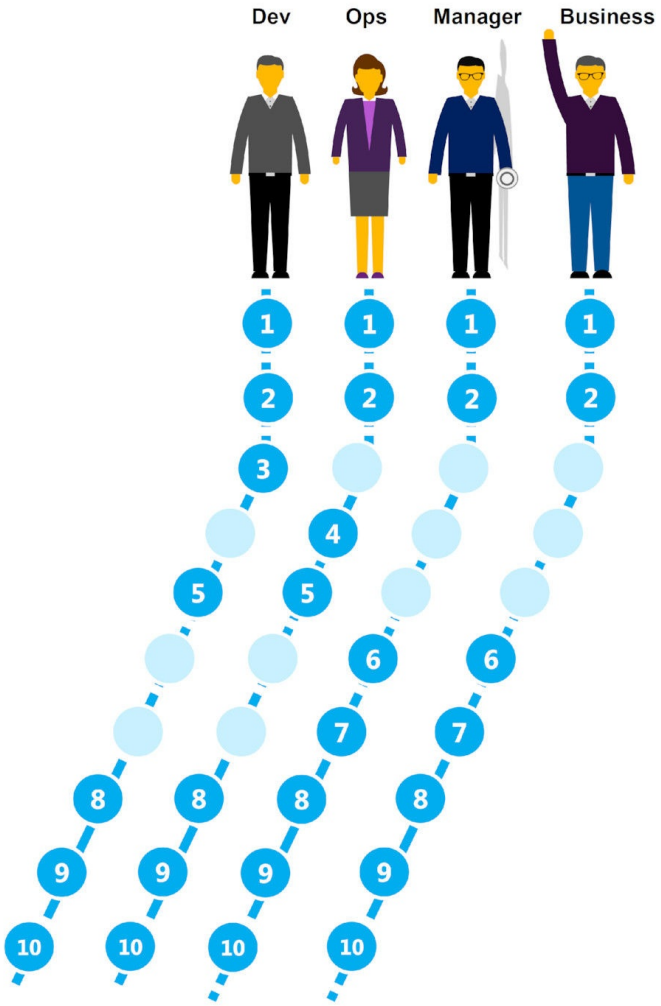
grandes entreprises pour lesquelles cette démarche constitue un élément clé du processus de *continuous delivery*. Enfin, nous étudierons comment les évolutions technologiques les plus avancées peuvent influer les outils et les processus DevOps de demain.

À qui s'adresse ce livre ?

Ce livre s'adresse à toutes les personnes intéressées par les systèmes d'informations modernes et innovants, à tous les passionnés d'informatique qui pensent que l'organisation est aussi importante que la technique pour réussir, ainsi qu'aux familiers de la notion d'agilité dans le monde de l'informatique.

La structure de ce livre se veut facile d'accès. Quel que soit votre rôle au sein de l'entreprise ou votre usage de l'outil informatique, ce livre se veut donc pédagogique avant tout.

Aussi, comme l'illustre la figure suivante, en fonction de votre profil de lecteur, certains chapitres vous seront sans doute plus directement bénéfiques que d'autres bien que tous présentent un intérêt certain.



Nous avons fait le choix d'aborder le sujet sous différents points de vue, notre but étant de répondre au mieux aux interrogations et problématiques pratiques des différents métiers impactés par DevOps. Le fait même d'étudier une démarche qui se veut collaborative du point de vue des différents acteurs nous a parfois amenés à revenir parfois sur le même

sujet dans différents chapitres. C'est un choix volontaire, le regard de différents acteurs posés sur ce même sujet peut parfois amener à une perception différente…

Pédagogique, accessible, pragmatique, voilà les maîtres mots qui ont guidé la rédaction de cet ouvrage…

Qui sommes-nous ?

Stéphane Goudeau est architecte dans la division Développeurs Expérience (DX) de Microsoft France.

Il travaille dans l'industrie des systèmes d'information depuis près de 25 ans. Après un début de carrière chez Bull Intégration Services, il a rejoint Microsoft comme consultant en 1996. Depuis, ses multiples rôles et responsabilités, exercés en tant que consultant principal, architecte, ou directeur technique du Microsoft Technology Center, lui ont permis d'acquérir une expertise de l'offre Microsoft sur les technologies de développement et d'infrastructure, ainsi qu'une une expérience approfondie dans la conception et à la mise en œuvre des systèmes d'information à destination de start-up, sociétés d'édition logicielle ou grands groupes français.

Engagé sur le cloud dès les premières heures de la plateforme Microsoft Azure, il s'est totalement investi dans l'exploration de ses modèles d'architecture, de développement et de gestion opérationnelle. C'est ainsi qu'il est devenu adepte de la philosophie DevOps depuis maintenant plusieurs années.

Passionné par l'agilité, l'innovation et le management, Samuel Metias a une expérience importante en conseil autour des méthodes agiles et de l'innovation. Il a également une expérience en tant qu'adjoint au maire de Colombes avec environ 200 agents sous sa responsabilité.

Concernant l'agilité, il a coaché de nombreux projets, participé à la construction de méthodologies agiles et formé des équipes sur les pratiques.

Plus spécifiquement, il est leader au sein de la division Service de Microsoft France de l'offre Agile & DevOps. Il anime la communauté DevOps de l'ensemble de la filiale française de Microsoft et depuis novembre 2015, il est responsable de l'alignement des offres DevOps de Microsoft Services à travers le monde.

Dans ses postes précédents, il a également une expérience d'architecte d'entreprise. Il a participé à la définition des méta-modèles d'architecture d'entreprise, à la conduite du changement suite aux architectures retenues ainsi qu'au pilotage des projets notamment autour des acteurs métiers.

Chez Microsoft, il s'appuie sur l'expérience importante des groupes produits de Microsoft pour en tirer les meilleures pratiques. Dans la continuité de l'agilité, il aide clients et partenaires à en tirer profit selon leur contexte.

Remerciements

Les auteurs tiennent tout d'abord à remercier leurs familles respectives qui ont patiemment supporté les longues heures de travail passées à la rédaction de ce livre au

détriment du temps passé ensemble. Merci tout d'abord à nos épouses et nos enfants.

Nous tenons à remercier chaleureusement M. Debois qui a accepté de nous relire et de signer cette préface en toute amitié.

Nous tenons également à remercier notre employeur commun qui nous permet de baigner quotidiennement dans l'univers DevOps en interne comme en externe.

Stéphane tient à remercier tout spécialement ses amis visionnaires et ex-collègues Blaise Vignon et Jakob Harttung, qui l'ont invité, il y a quelques années, à s'intéresser à DevOps, ainsi que son manager Guillaume Renaud et son directeur Nicolas Gaume, qui l'ont encouragé dans cette voie : un voyage qui lui aura permis d'apprendre, en continu, sur bien des sujets… Et pour ce voyage, la couverture de notre ouvrage nous offre une nef un peu particulière, celle du Grand Palais : merci à Christine Goudeau pour ses talents de photographe.

Samuel tient à remercier tout particulièrement Monsieur Jean-Patrick Ascenci qui a été son premier mentor et manager dans sa carrière professionnelle et qui lui a donné le gout de son métier ainsi que Messieurs Mario Moreno et Charles Zaoui qui ont été des modèles d'expertise et de bienveillance et l'ont fait plonger dans le monde de l'agilité qu'il n'a plus jamais quitté.

Samuel a également une pensée particulière pour Antoine Durand et Laurent Le Guyader ses accolytes au quotidien autour de l'Agile et de DevOps ainsi que pour Randa Debbi au soutien toujours indéfectible et si précieux.

Enfin, nous remercions chaleureusement nos relecteurs qui nous ont inondés de nombreux et bons conseils. Nous tenions à les citer ici : Jean-Luc Blanc, notre éditeur, Charlotte Dando, Gaelle Cottenceau, Blaise Vignon, Franck Halmaert, Hervé Leclerc, Marc Gardette, Stéphane Vincent, Jean-Marc Prieur, Julien Corioland, Guillaume Davion, Jivane Rajabaly, Laurent Le Guyader, Benjamin Guinebertière…

Chapitre 1

La démarche DevOps enfin expliquée

Patrick Debois raconte qu'il a inventé le terme DevOps, après une série de conférences, peu fréquentées d'ailleurs, tandis qu'il cherchait à qualifier simplement la notion de gestion agile de l'infrastructure.

L'important n'est donc pas tant de savoir comment ni pourquoi Patrick Debois a utilisé pour la première fois ce terme, mais plutôt de comprendre ce qu'il cherchait à expliquer et que l'on pourrait résumer en : les infrastructures informatiques et les *opérations* peuvent être agiles. Plus encore, elles devront, elles auront l'obligation de l'être demain pour réussir.

Il faut remarquer aussi que l'usage du mot DevOps n'est pas anodin. Sa simple concaténation marque l'impérieuse obligation de collaborer étroitement entre les *développements* et les *opérations*. La collaboration est une nécessité, mais collaborer étroitement en partageant largement jusque dans la responsabilité de l'échec ou du succès est la philosophie DevOps. Un véritable choc culturel.

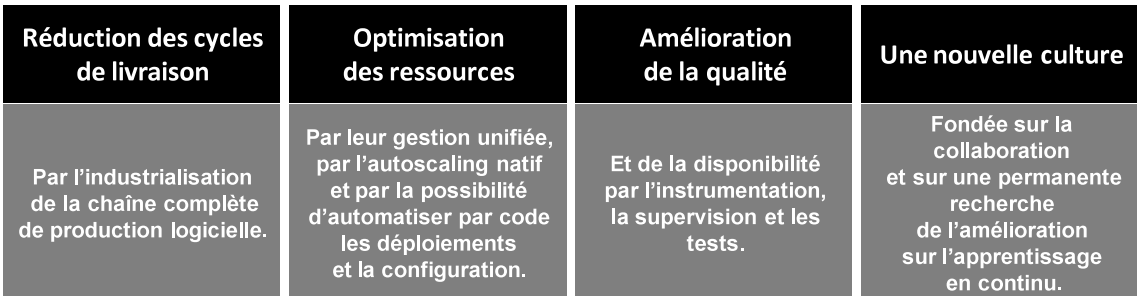


Fig. 1.1 *Les fondamentaux d'une démarche DevOps*

1.1. Culture

Comme nous l'avons dit, même si elle peut être facilitée par l'adoption de nouveaux outils et technologies, la démarche DevOps est avant tout une philosophie. Sa dimension culturelle est donc fondamentale.

1.1.1. Confiance réciproque et compréhension globale du système

Au cœur de cette culture, il y a la volonté de changer le mode d'interaction entre les équipes de développement et les opérations. L'objectif est non seulement de partager l'information, mais aussi les responsabilités, ce qui suppose une évolution des mentalités pour parvenir à établir la relation de confiance requise et l'implication de tous les acteurs.

Au-delà de cette confiance, il faut que chacun puisse acquérir une compréhension globale du système, de sorte que nul ne puisse ignorer les besoins ou les contraintes de l'ensemble des acteurs du système d'information. Cela se traduit par une évolution de l'organisation, de ses processus, du rôle et des périmètres de responsabilités de chacun.

1.1.2. Kaizen : la recherche de l'amélioration continue

Pour parvenir à étendre la portée de l'ensemble des acteurs du système, la culture DevOps favorise le développement des compétences dans une recherche perpétuelle d'amélioration. Cette approche est similaire au processus industriel Kaizen, qui s'est développé au Japon, dans la reconstruction qui a fait suite à la seconde guerre mondiale. Le mot *Kaizen* est issu de la fusion des deux mots japonais *kai* et *zen* qui signifient respectivement *changement* et *bon*.

Fig. 1.2 *Le Kaizen en caractères japonais*

La pratique quotidienne de cette démarche d'amélioration continue est la condition sine qua non de la réussite de sa mise en œuvre. Elle permet une évolution progressive, sans *à-coups*, et incite chaque acteur du système à proposer des optimisations simples et concrètes sur l'ensemble de la chaîne de production. Appliquée au cycle de production logicielle, cette réorientation culturelle impacte l'ensemble de l'organisation, qui devient ainsi plus prompte à s'adapter et à rechercher le changement plutôt que de le fuir.

1.1.3. Lean Startup : construire, mesurer, apprendre

Autre source d'inspiration pour la culture DevOps, la philosophie *Lean Startup* proposée en 2008 par un américain, Eric Ries, initialement à destination des start-up. Depuis, la cible de cette démarche s'est étendue à l'ensemble des entreprises qui souhaitent mettre à disposition un produit sur le marché. Reprenant certains des principes du *Lean Management* (là encore une méthode de rationalisation de la production qui nous vient de l'industrie japonaise, en l'occurrence Toyota), Lean Startup vise à réduire les gaspillages et augmenter la valeur en continue pendant la phase de développement du produit.

Cette approche se traduit par une volonté d'amélioration continue de la performance (en termes de productivité et de qualité), par une réduction des délais, des coûts et la définition d'un *produit minimum viable* que l'on peut soumettre à l'évaluation des consommateurs. Cela suppose la mise en place de systèmes de mesure et des processus de remontée d'information systématique et le développement d'une culture fondée sur l'apprentissage en continu.

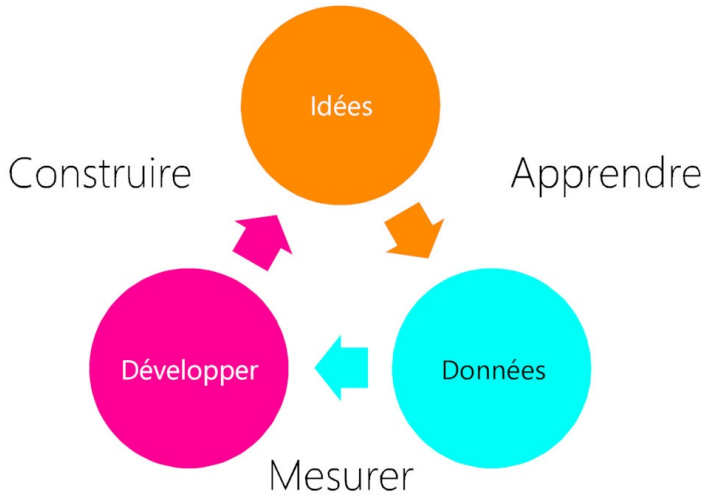


Fig. 1.3 *Le triptyque fondamental du Lean Startup*

Au-delà de *construire*, il y a les hypothèses et la décision de construire. Cette décision est prise en fonction de ce qui est appris du comportement et des attentes des utilisateurs. Cet apprentissage se fait à partir des données mesurées. Ces données sont mesurées par des instrumentations mises en place en fonction des hypothèses à valider. L'objectif est de permettre à l'entreprise de se doter des moyens permettant de contrôler en permanence l'adéquation entre la vision du produit, son implémentation et les attentes du marché.

Ces mesures permettront de valider les hypothèses ayant conduit à la définition du cahier des charges du produit. Mais elles permettront aussi d'optimiser l'intégralité des chaînes de valeur métier dépendant de services informatiques, et de s'assurer de sa fiabilité en résolvant les problématiques au plus tôt afin de limiter leur impact.

Elles permettront de réagir au plus tôt pour procéder aux changements requis et permettront d'étalonner la performance commune, puisque dans la démarche DevOps, les résultats, comme la livraison d'un service en production, sont désormais partagés. Elles nécessiteront la mise en place de processus communs de déploiement, de supervision (détection et prévention d'incidents de performance, de sécurité, de disponibilité), de support et de remédiation.

1.1.4. Une vision positive de l'échec

Enfin, cette culture qui recherche le changement, va encourager l'expérimentation en proposant une vision positive de l'échec. Pour définir de nouveaux besoins, il faut accepter de prendre des risques et cela ne doit pas être un frein. L'échec, au même titre que le succès, devient une source d'apprentissage (*Fail fast*). Anticiper une situation où le système ne répond plus et y remédier dans les plus brefs délais suppose l'implication de l'ensemble des acteurs. La capacité du système à se remettre en service après un dysfonctionnement sera augmentée si les plans d'escalade et processus internes sont partagés entre les équipes.

Une bonne pratique permettant de valider l'efficacité de ces processus collaboratifs consiste à volontairement introduire des erreurs dans le système (*Fault Injection Testing*) et à gérer la situation qui en résulte en maximisant la collaboration entre les équipes. Un

exemple de cette approche est le service *Chaos Monkey* que Netflix a développé sur la plateforme Amazon Web Services. Nous reviendrons d'ailleurs plus en détail sur cette implémentation dans la suite de cet ouvrage.

Autre exemple, la démarche *Resilience Modeling and Analysis* (RMA), issue du standard de l'industrie *Failure Mode and Effects Analysis* (FMEA), qui consiste à identifier en amont les éléments risquant une panne au sein d'une solution et les modes de défaillance qui en résultent. L'objectif est de définir les stratégies de résilience et de disponibilité dès la phase de conception de l'application en bâtissant un modèle qui détaille les possibilités de dysfonctionnement et qui documente l'ensemble des actions destinées à en atténuer l'impact.

Les contre-mesures ainsi définies pour s'assurer que l'incident ne puisse pas se produire ou pour limiter ses effets concernent les équipes de développement ainsi que les opérations. C'est l'un des objectifs de la modélisation des menaces dont nous reparlerons dans le chapitre 5 *DevOps vu par la qualité*.

1.1.5. Une culture aux multiples facettes…

Comme on peut le constater, la culture DevOps s'inspire donc d'une pluralité de disciplines déjà mises à l'épreuve avec succès dans le monde de l'industrie. Elle est fondée sur le respect mutuel des équipes, la confiance réciproque et le partage des responsabilités. Elle s'appuie sur une organisation et un management adapté. Enfin, elle s'inscrit dans une démarche d'amélioration, d'expérimentation et d'apprentissage en continu.

1.2. Collaboration

1.2.1. Une implication du métier

La structure même du mot DevOps nous indique que la démarche s'appuie largement sur la collaboration entre le monde des développements et celui de l'infrastructure. La structure du mot est moins explicite sur le troisième acteur essentiel de cette collaboration : le métier.

En effet, la collaboration qu'insuffle cette démarche n'a qu'un seul objectif : réussir à impliquer les acteurs métiers de manière pertinente et continue. Cet objectif n'est pas nouveau en soi, mais les expériences précédentes n'ont pas toujours été concluantes, même avec des méthodes agiles.

Comment pouvons-nous espérer qu'un acteur métier s'implique durablement dans les projets informatiques si au moindre déploiement, il est le témoin privilégié des mésententes entre ces deux acteurs essentiels de l'IT ?

Réussir à reconstruire la confiance et à la conserver dans le temps est donc le premier objectif de toute démarche DevOps. Pour reconstruire cette confiance, il faut avoir conscience des *clés de confiance* de chacune des parties et travailler prioritairement sur

ces sujets. En général, la confiance n'est jamais très loin. Il manque souvent juste un peu de volonté… et de méthode !

1.2.2. Identifier la clé de confiance des opérations

Pour réussir à construire la confiance entre deux mondes qui cohabitent, il faut identifier les leviers sur lesquels elle peut se construire. Chaque rôle dans cette collaboration n'espère pas bâtir cette confiance pour les mêmes raisons. Leurs objectifs premiers sont différents mais complémentaires.

Les équipes opérations tiennent avant tout à maintenir la production fonctionnelle et disponible. Certes, pour y parvenir sans recourir au paradigme de la stabilité à tout prix et gagner en fiabilité, ils doivent maîtriser les impacts de tous les changements, mais pas seulement. En réalité, l'une de leurs principales attentes vis-à-vis des équipes de développement est la qualité du produit qui leur est livré. De leur point de vue, si le produit est de qualité suffisante, il n'aura pas d'impact néfaste sur la production.

Et un produit de qualité du point de vue de des équipes de production n'est pas un concept particulièrement complexe : c'est un produit sans bug majeur sur un environnement *iso-production*. En théorie, pour y parvenir, il suffit de tester. Mais la théorie n'est pas si simple à appliquer pour de multiples raisons, parmi lesquelles certaines sont très courantes :

- **La stratégie de recette et de tests est rarement partagée**. Puisque ce sont les développements qui assument la responsabilité de la grande majorité des tests, la typologie de tests à réaliser comme les résultats de ces derniers ne sont pas partagés.
- **La stratégie de recette et de tests n'est pas toujours définie et/ou rigoureusement respectée**. Lorsqu'une typologie et un planning de tests existent, ils ne sont pas toujours respectés rigoureusement : cela finit toujours par se savoir et par éroder la crédibilité de tout le plan qualité.
- **La stratégie de recette et de tests n'est pas appliquée par les bonnes personnes**. Il arrive trop souvent que le code soit implémenté, les tests écrits et le *sign-off* réalisé par la même personne. Il est évident que cela nuit à la crédibilité du résultat.

Pour que les équipes en charge des opérations souhaitent rechercher de nouvelles formes de collaboration, tous ces obstacles doivent être levés en définissant ensemble une stratégie de tests et de recette, garantissant un niveau de qualité satisfaisant pour tous et appliquée avec rigueur.

Vous l'avez compris la stratégie de tests et de recette est l'une des clés de confiance des équipes opérations pour réussir une démarche DevOps.

1.2.3. Identifier la clé de confiance des développements

Démontrer la qualité du produit développé n'est pas suffisant en soi. Il ne faut pas croire d'ailleurs que les développeurs rechignent à réaliser des produits sans bugs et sans dysfonctionnements. Au contraire, la véritable difficulté qui constitue leur clé de

confiance est leur capacité à disposer des bons éléments pour réaliser les tests qui les rassurent sur la qualité de leur produit.

Dit autrement, il s'agit pour les équipes de développement de disposer d'environnements *iso-production* à la demande dans des temps raisonnables et d'outils permettant de réaliser un maximum de tests avec le moins d'effort supplémentaire possible.

Il ne faut pas négliger non plus l'importance du cadre défini et généralement accepté qui détermine ce qu'est un produit de qualité du point de vue de l'organisation.

Lorsque se pose la question du niveau de test attendu pour s'assurer d'une qualité produit suffisante, il y a assez rarement une réponse. Et lorsque cette réponse est celle de l'un des développeurs, elle est souvent différente de celle d'un autre membre de la même équipe. Il est pourtant fondamental de connaître les tests minimums requis et les résultats attendus de ces tests.

Il faut néanmoins relativiser : la grande majorité des équipes de développement partagent des stratégies de tests relativement respectées et tous les développeurs ont conscience de la nécessité de faire des tests. La grande souffrance de ces équipes est de ne pas avoir les moyens de réaliser correctement ces tests.

Pour caricaturer, un développeur réalise un code de grande qualité, définit des algorithmes complexes, mais son code, particulièrement optimisé ne fonctionne pas en production parce qu'il n'a jamais pu le tester dans des conditions similaires à cet environnement de production.

En règle générale, la plus grande difficulté des développements est d'obtenir des environnements et des machines de tests dans des délais raisonnables. Au-delà de la difficulté à parfois obtenir un environnement dont les caractéristiques sont le plus proches possible de l'environnement final, ce sont les délais de mise à disposition de ces environnements qui sont problématiques.

Il n'est pas rare de devoir compter les délais en semaines et de devoir faire intervenir sa hiérarchie pour obtenir les outils nécessaires pour faire son travail. Pourtant, la plus grande partie des développeurs se soumettra volontiers à un choix plus restreint d'outils et d'environnements pour peu qu'on leur garantisse l'efficacité des tests et des délais d'approvisionnement raisonnables.

La qualité des environnements provisionnés et les délais de provisioning font donc partie des clés de confiance des développements.

1.2.4. Identifier la clé de confiance du métier

La confiance des métiers est le moteur de leur implication dans le monde de l'IT. A contrario leur défiance tire son origine des décalages permanents de perception entre les deux mondes des équipes informatiques. Il n'est pas rare d'observer des entités métiers préférant travailler avec des sociétés extérieures plutôt qu'avec le service informatique interne.

Pourtant, les métiers sont demandeurs d'un service informatique à la hauteur des meilleurs concurrents du marché. Après tout, qui mieux que les informaticiens de leur société peuvent comprendre les contraintes de leur métier ? Aujourd'hui l'informatique est partout, ils ont donc toute latitude pour voir leur métier dans sa transversalité.

Dès lors que les équipes informatiques font valoir leur expertise, leur réactivité et la qualité de leur collaboration, au moyen d'une démarche DevOps par exemple, les équipes métiers s'impliqueront naturellement. Il est certain que personne ne souhaite s'impliquer dans une coopération avec des services en conflit.

La qualité de la collaboration et l'expertise des équipes informatiques sont donc les clés de confiance des métiers.

1.3. Automatisation

1.3.1. Une automatisation pour pérenniser la confiance

Pour retrouver de la confiance dans une collaboration parfois difficile, il est important de commencer par travailler ensemble sur les clés de confiance identifiées pour chacune des parties prenantes. En trouvant ensemble un mode de fonctionnement qui vous convient collégialement, vous bâtirez les fondations d'une nouvelle culture de collaboration.

Une bonne solution pour démontrer que cette collaboration retrouvée est efficace et pérenne est d'introduire de l'automatisation. Une collaboration qui fonctionne et qui est automatisée (au moins en partie dans un premier temps) démontre une volonté et un investissement qui rassurent les métiers.

En effet, une collaboration qui fonctionne ne doit pas être mise en péril si l'équipe s'agrandit ou les personnes sont remplacées. Pourtant, et c'est normal, toute collaboration repose avant tout sur des femmes et des hommes qui s'entendent sur une manière de fonctionner.

La structuration du processus autour de l'automatisation permet donc de créer un cadre dans lequel de nouveaux collaborateurs entrent naturellement. La pérennité de ce cadre de collaboration n'empêche pas de le faire évoluer.

En effet, le temps aidant et la maturité grandissant, il est plus que probable que le cadre de collaboration soit dans l'obligation de s'adapter au changement de son environnement technique comme humain. Le processus doit intégrer un dispositif d'amélioration continue et la chaîne d'automatisation doit le prendre en compte. Le changement doit se faire de manière pragmatique et structurée avec une rigueur de tous les instants.

Pour que la confiance reste à la fois dans les hommes et dans le cadre procédural automatisé, il est important d'éviter l'instabilité et l'incertitude. Ces notions ne sont pas contradictoires avec la prise en compte de l'amélioration continue, si ces principes sont respectés rigoureusement et évitent ainsi le changement à tout va.

C'est à ce prix que la confiance peut naître, s'accroître et se consolider entre toutes les parties prenantes d'une démarche DevOps.

Néanmoins, la consolidation de la confiance n'est pas le seul avantage à introduire une démarche automatisée, elle permet également de gagner du temps et de la qualité.

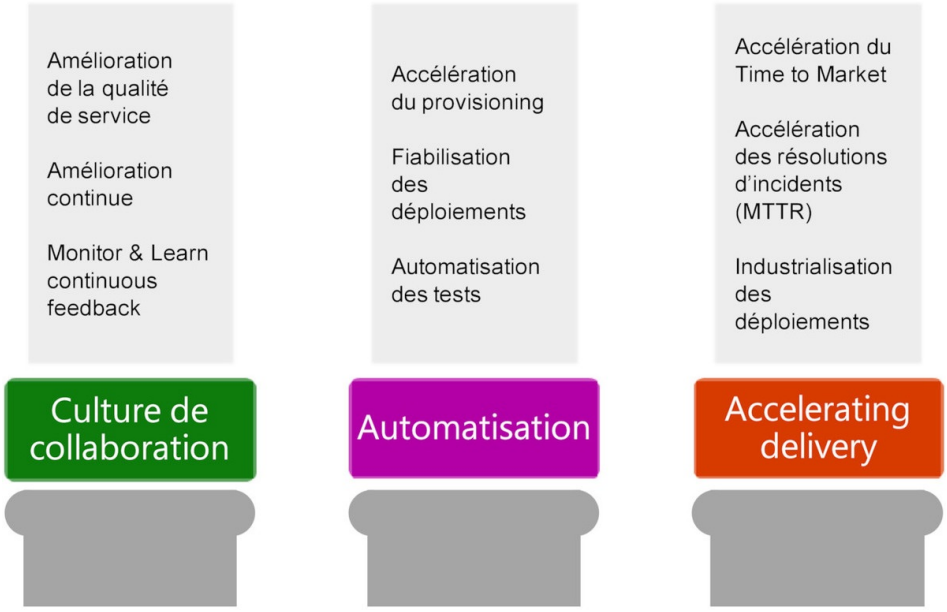


Fig. 1.4 *L'automatisation au centre des piliers d'une démarche DevOps*

1.3.2. Une automatisation pour gagner en qualité

Automatiser, c'est effectivement gagner du temps : d'abord pour les tâches les plus répétitives à faible valeur ajoutée, mais aussi pour les tâches plus complexes techniquement. Ces mêmes tâches sont également celles qui sont les plus susceptibles d'introduire des erreurs d'inattention mettant à mal la collaboration. Automatiser, c'est donc également assurer un niveau de qualité constant et optimal.

Cette notion peut paraître évidente, mais dans les faits, tout n'est pas si simple. Il n'est pas toujours facile d'identifier les tâches à faible valeur ajoutée, sans oublier que personne n'aime voir son travail dévalorisé. La notion même de valeur ajoutée peut faire l'objet d'un débat.

En réalité, les processus qui devraient naturellement porter la collaboration sont rarement définis d'un commun accord, chacun gardant plutôt jalousement son bout de processus dans son coin pour le faire évoluer à sa guise. Rester dans cet état d'esprit avant d'automatiser ne fera pas gagner en qualité. En réalité, il n'y aurait même aucun bénéfice à automatiser.

Pour gagner en qualité il faut donc partager le processus qui porte la collaboration et établir un consensus sur les tâches les plus critiques candidates à l'automatisation. Le consensus ainsi établi évite le débat sur la valeur de ces tâches, on se concentre sur la valeur apportée par l'automatisation.

En général ce sont les tâches les plus pénibles pour les équipes, soit parce qu'elles sont particulièrement chronophages sans intérêt intellectuel particulier, soit parce qu'elles sont particulièrement complexes et éprouvantes. Dans les deux cas, ce sont les tâches les plus

susceptibles de générer des erreurs, et du fait de ces erreurs de générer de la tension entre les partenaires.

Lorsque ces tâches sont identifiées, les automatiser apporte un gain évident et souvent immédiat. L'automatisation peut d'ailleurs se faire à moindre coût si on ne cherche pas à optimiser en même temps que l'on automatise, car, automatiser et optimiser sont deux activités bien distinctes.

1.3.3. Automatiser n'est pas optimiser

Automatiser consiste à rendre systématique à l'aide d'outils technologiques un enchaînement d'activités spécifiques qui n'est pas forcément optimisé ou industrialisé.

Il est pourtant vrai que l'inconscient collectif associe facilement l'automatisation et les automates à l'industrie et donc à une certaine forme d'optimisation. Une démarche DevOps évite ce type de préjugés. Lorsque nous automatisons une collaboration, nous ne nous embarrassons pas de savoir si elle est optimale et industrialisable, nous nous contentons de nous assurer qu'elle fonctionne.

Il est possible et même recommandé pour accélérer et tendre vers du *continuous delivery* de chercher ensuite à supprimer les étapes inutiles et à optimiser la chaîne d'activités.

1.4. Continuous delivery

Lorsqu'une collaboration fonctionne et qu'elle est au moins en partie automatisée et ainsi pérennisée, il faut entrer dans une phase d'amélioration continue avec pour objectif l'accélération continue des cycles de livraison.

Le *continuous delivery* consiste en une automatisation complète de la chaîne de mise en production autant au niveau de la plateforme technologique que du processus de collaboration associé. Sa mise en œuvre implique donc de ne plus avoir aucune interaction humaine entre la réalisation du service ou du produit et sa mise à disposition finale. Il s'agit d'un objectif ambitieux qui peut volontairement n'être que partiellement atteint. On préfère alors parler *d'accélération du delivery* plutôt que de *continuous delivery*.

Pour y parvenir, la recherche de gaspillages et d'une industrialisation des processus est le premier axe de travail important qui est au cœur de la culture DevOps.

L'autre axe de travail fondamental concerne alors l'extension du périmètre d'automatisation. Le degré de confiance dans les processus automatisés augmentant avec le temps et la maturité des équipes sur le sujet, il devient nécessaire d'envisager en permanence d'automatiser un peu plus ce qui fonctionne déjà afin de tendre vers le *continuous delivery*.

Pour y parvenir de manière agile, ou autrement dit de manière itérative et incrémentale, il faut avoir conscience des différents processus et des différentes problématiques que soulève une démarche DevOps.

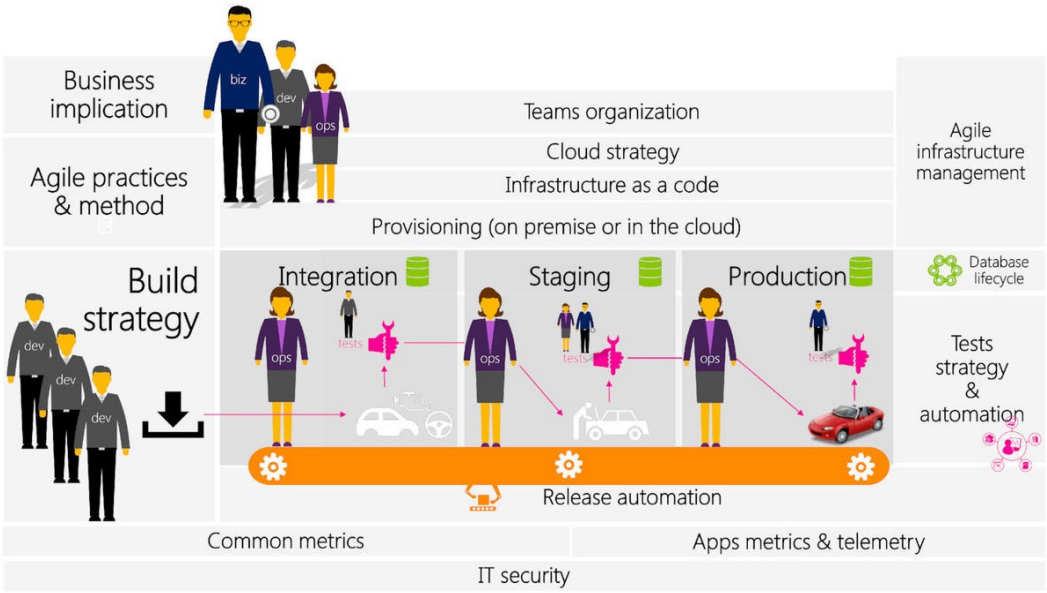


Fig. 1.5 *Les principales pratiques d'une démarche DevOps*

Savoir traiter et mettre en œuvre chacune de ces problématiques séparément pour en démontrer la valeur rapidement est primordial. Il faut avoir constamment conscience que tous ces processus sont liés entre eux. Il s'agit donc de savoir construire sa feuille de route agile autour des sujets DevOps en fonction du contexte pour créer un cercle vertueux d'amélioration continue.

1.5. Perspectives technologiques

Même si DevOps est avant tout une transformation de la culture des acteurs et des processus du système d'information, sa dimension technologique n'est pas neutre.

1.5.1. L'intégration de multiples outils

Le succès de sa mise en application ne repose pas sur un seul outil ou sur une seule technologie : le principe est de s'appuyer sur ce que chaque outil fait de mieux. Mais c'est aussi du niveau d'intégration de ces outils que dépend l'optimisation des processus, la coordination de la gestion des livraisons, l'efficacité de la collaboration, la durée des cycles de déploiement ou la capacité à gérer de multiples environnements. En effet, ces outils se fondent sur la manipulation d'objets communs, idéalement, tous issus de la même source, et partagés entre les équipes de développement et d'infrastructure. Ainsi, il ne peut y avoir de divergence de vue vis-à-vis de ces objets entre les développeurs et les opérations.

1.5.2. Le partage de modèles d'infrastructure

Cette démarche est facilitée par la virtualisation des systèmes et par la prise en charge par de multiples outils d'automatisation de provisioning et de configuration. Aujourd'hui, il est possible de réserver et configurer les ressources processeur, réseau et stockage d'une plate-forme complète à partir d'un environnement *bare-metal* ou d'un hyperviseur disposant d'un référentiel d'images virtuelles.

Ce provisioning est réalisé sur la base de modèles d'infrastructure qui vont pouvoir être partagés et répliqués sur les différentes plates-formes (développement, intégration, recette, pré-production, production) qui pourront varier en termes de dimensionnement mais seront identiques en termes de configuration. Ainsi le comportement observé sur la plate-forme d'intégration sera le même que celui constaté sur la production.

1.5.3. Infrastructure as code

L'environnement de déploiement d'une application est donc devenu un livrable du projet. À ce titre, il doit être archivé et lui aussi géré en versions, en tirant parti des pratiques issues du monde du développement (tels que l'utilisation de référentiels partagés avec contrôle de version, l'automatisation des tests…).

Cette nouvelle approche suppose la capacité à gérer directement par des lignes de code le provisioning et la configuration de l'infrastructure dans un langage permettant de les créer et les faire évoluer. Les services d'infrastructure, qu'ils s'exécutent à demeure ou dans le cloud, doivent donc être dynamiquement modifiables *via* des interfaces de programmation. Cette composante *infrastructure as code* est un élément clé de la dimension technologique de DevOps.

1.5.4. L'instrumentation au cœur du système

Enfin, comme nous l'avons vu précédemment, l'instrumentation d'une solution est essentielle. Il faut pouvoir, en continu, observer le comportement du système sur le plan technique. Ainsi développeurs et équipes de gestion des opérations pourront, au plus tôt, identifier les limites en termes de performance et corriger les dysfonctionnements.

Il faut également, sur le plan fonctionnel, obtenir des informations permettant de répondre aux besoins fluctuants des utilisateurs et valider les hypothèses ayant conduit à définir le cahier des charges du produit.

En résumé

Pour résumer, nous pourrions dire que DevOps va au-delà de ce que laisse supposer son nom, car c'est une démarche de collaboration agile entre le monde des études et du développement (Dev), le monde des opérations, de la production et des infrastructures (Ops) et les représentants des métiers, des utilisateurs et des clients (Business) pour l'ensemble du cycle de vie du service, de sa conception initiale jusqu'à son support en production.

La mise en œuvre s'appuie donc sur trois piliers, à commencer par la collaboration pour construire une relation de confiance, puis sur l'automatisation pour pérenniser la confiance et enfin l'industrialisation pour accélérer et tendre vers le *continuous delivery*.

Enfin, et surtout, DevOps est une transformation culturelle. La dimension technologique n'est là que pour faciliter l'évolution qu'elle sous-tend.

Chapitre 2

DevOps dans la transformation digitale

La transformation digitale est la réponse de chaque entreprise face au défi que représente l'essor des technologies numériques. Concrètement cela suppose la dématérialisation de ses processus, évolution centrée sur une innovation à laquelle participent des concepts et des solutions technologiques tels que l'agile, le cloud, le big data, le *machine learning* et la mobilité.

Cette transformation est efficace si elle s'inscrit dans une démarche globale dont DevOps peut être la clé…

DevOps est agile, c'est un fait, mais il ne suffit pas de l'affirmer, il faut le comprendre.

2.1. DevOps est agile

Pour entendre que DevOps est agile, il faut comprendre ce qui se cache derrière le mot *agile*. Aujourd'hui il a été utilisé dans tellement de contextes différents que sa véritable signification s'est diluée dans la conscience générale.

2.1.1. Une définition de l'agile

La dilution et l'incompréhension de l'agilité ont pour conséquence d'affecter son image au sein de l'entreprise et de complexifier son adoption.

Prenons l'exemple d'une direction générale qui décrète que le groupe doit désormais être agile sans plus de précisions. En réalité, cette direction ne sait pas exactement quelle signification mettre derrière le mot agile. Imaginez maintenant l'écart de compréhension que ce flou va engendrer parmi les directions, les équipes et les salariés.

Finalement, le retour sur investissement semble inexistant pour les principaux intéressés. L'agilité a été perçue comme une notion marketing apparentée à une coquille vide.

Il est vrai que le mot *agile* peut avoir plusieurs sens selon le contexte où il est utilisé :

- Une **organisation** peut être agile quand elle est prête à se remettre en cause en cycles d'amélioration continue et à changer sa hiérarchie et ses règles de management périmètre après périmètre. Le changement n'est plus subi mais perçu comme générateur d'efficacité dans chaque département de l'entreprise. Cette vision de l'agilité est relativement neuve, centrée sur les comportements, les ressources humaines et l'organisation hiérarchique d'une entreprise.
- Le **processus de R&D, l'innovation et le marketing** produit peuvent être agiles. C'est le cas notamment de la conception de nouveaux produits innovants ou des méthodes marketing de recherches de signaux faibles annonciateurs de changement pour l'entreprise. L'iPhone ou l'iPad sont souvent cités en exemple de ces produits

conçus de manière agile. Le Lean Startup est une méthode agile qui formalise un certain nombre de concepts qui se prêtent à cette thématique.

- L'**outil informatique** lui-même peut être agile dans sa conception et son architecture. L'apport des démarches d'architecture orientée services (voire microservices) avec la décorrélation complète du moteur d'exécution applicatif et des interfaces utilisateurs contribuent à l'atteinte de cet objectif. La mobilité, les nouveaux devices que ce soient les smartphones, les tablettes ou les objets connectés, tirent directement parti de cette approche.
- Enfin le **processus de gestion** des projets et des hommes autant que le *run* peuvent être agiles. C'est dans ce cadre que l'on exploite pleinement les méthodes de projet agiles ou la démarche DevOps, en s'appuyant sur un processus agile *de bout en bout*, autrement dit *du besoin client à la livraison client*.

On voit bien que la signification du mot *agile* dépend du contexte dans lequel on l'utilise. Mais on comprend également que l'agilité s'appuie sur des valeurs communes qui s'appliquent totalement à une démarche DevOps. Et ces valeurs ne se résument pas au manifeste agile qui est le socle commun de ces méthodes, ce serait bien trop limité. Pour pouvoir s'appliquer à la fois aux développements, à l'infrastructure, à la conception de produits ou encore au marketing, ces valeurs se doivent d'être plus étendues.

Nous sommes dans une démarche itérative, incrémentale et collaborative, nous sommes dans une démarche agile, quels que soient le contexte ou le domaine d'application.

2.1.2. DevOps dans la continuité des méthodes agiles

DevOps respecte de manière tout à fait évidente les valeurs fondamentales de l'agilité, et s'inscrit dans la filiation des *méthodes agiles* traditionnelles.

Lorsque les méthodes agiles traditionnelles sont correctement mises en œuvre, elles génèrent souvent une forme d'enthousiasme auprès des utilisateurs et des populations métiers.

Malheureusement, l'enthousiasme initial de l'utilisateur se transforme vite en frustration quand la coopération des services de développement et de gestion des opérations n'est pas agile et que les différents services informatiques ne collaborent pas ensemble dans la direction attendue par les métiers.

L'idée de DevOps est de pousser les démarches agiles jusqu'aux équipes de production en passant par une collaboration agile basée sur la confiance. DevOps hérite pleinement de la culture agile des méthodologies projets pratiquées depuis des années.

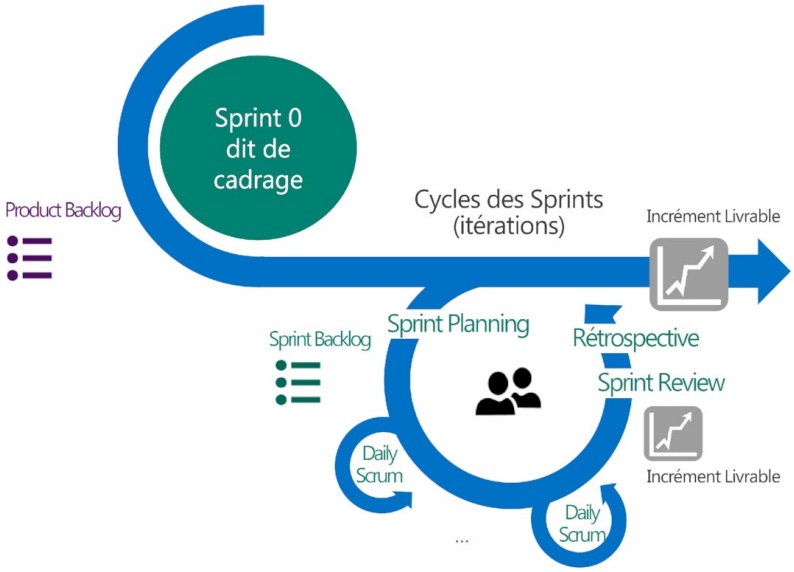


Fig. 2.1 *Le cycle de développement agile*

DevOps n'est pas agile seulement parce qu'elle respecte les valeurs fondamentales de l'agilité, elle est agile parce que, comme les autres méthodes à succès, elle apporte sa pierre dans l'édifice des bonnes pratiques liées à l'agilité :

- **Scrum est la méthode agile la plus populaire aujourd'hui.** C'est effectivement la démarche la plus séduisante car elle se concentre sur l'aspect managérial des hommes. Sa similitude avec le rugby qui en a inspiré le nom est significative : on parle d'équipe et d'esprit d'équipe avant tout. Les hommes, l'équipe et la responsabilisation des membres de l'équipe sont les maîtres mots.
- **XP Programming est la méthode agile la plus en vogue auprès des ingénieurs.** En effet, elle introduit les pratiques d'ingénierie agile les plus abouties et bénéficie d'une réputation sans faille concernant la qualité des produits logiciels développés.
- **Agile UP est la méthode agile partageant le plus de concepts avec les méthodes classiques.** Cette méthode est en effet une évolution du *Rational Unified Process* d'IBM puis du *Unified Process* avec Ivar Jacobson notamment pour y introduire les principes de l'agilité. En conséquence, c'est aussi la méthode agile la plus facile à introduire auprès d'équipes évoluant depuis des années avec des méthodes classiques. En d'autres termes, agile UP est la méthode agile la mieux adaptée à la conduite du changement aujourd'hui.
- **Lean est une méthode issue du monde industriel** dont les principes se sont rapidement imposés au sein du monde agile. La recherche de l'industrialisation et de l'optimisation toujours dans une optique de processus orienté client est fondamentale. Les principes de cette méthode sont, comme nous l'avons vu, au cœur de la culture DevOps.
- **Lean Startup est une méthode dérivée du Lean orientée pour les entrepreneurs** autant internes qu'externes à une entreprise**.** Elle vise à rationaliser l'état d'esprit innovant des start-up tout en supprimant les causes d'échecs et de gâchis. Cette approche scientifique orientée produit se combine également très bien avec une démarche DevOps orientée IT.

- **Kanban est une méthode assimilée aux méthodes agiles, bien que non itérative de prime abord.** En réalité, cette méthode combine plusieurs *itérations de cérémonies* parallèles couplées à un *backlog* à flux tendu. Kanban se concentre sur l'industrialisation du processus agile au moins sur son aspect développement.

Il existe d'autres méthodes agiles, mais nous nous sommes limités ici aux plus populaires. Dans toutes ces approches, de nombreux aspects différents de l'agilité sont couverts : le management des hommes (Scrum), la qualité du logiciel (XP), l'adoption du changement (agile UP), l'industrialisation (Kanban) et l'optimisation (Lean) du processus agile.

Cette énumération ne couvre pas un aspect essentiel : la collaboration. C'est le point fort de DevOps qui se concentre sur les aspects de collaboration du processus agile, du besoin client à la livraison au client. DevOps reprend donc les principes de l'agile et les complète.

2.2. DevOps et le cloud

L'attraction qu'exerce le *cloud computing* sur l'industrie des technologies de l'information rend plus impérieuse encore la nécessité pour les développeurs de logiciels et les responsables des opérations de collaborer, pour la création ainsi que l'exploitation d'applications et de services innovants et massivement *scalables*.

2.2.1. Une évolution globale du processus de développement

Le cloud offre aujourd'hui la possibilité de créer et déployer de nouvelles applications et leur infrastructure sous-jacente en quelques minutes. Toutefois de multiples changements sont requis afin d'en tirer le meilleur parti : définition des modèles de nouveaux services, automatisation de leur déploiement, gestion des configurations et des mises à jour, monitoring des performances, remontée d'alertes, et enfin et surtout intégration des processus de développement.

2.2.2. Une évolution du rôle des équipes de gestion des opérations

Tout cela n'est pas sans incidence sur le métier du responsable opérationnel. À l'ère du cloud, celui-ci devra savoir exploiter au mieux les capacités de virtualisation mises à disposition dans les data centers en déléguant totalement les opérations d'infrastructure qui jadis étaient de sa responsabilité.

Dans un premier temps, l'évolution vers le cloud s'est traduite par une séparation des responsabilités entre la gestion du logiciel et des données, de celle de l'infrastructure physique. Grâce à la virtualisation des serveurs, du stockage et du réseau, les systèmes physiques sont devenus totalement découplés des éléments numériques qu'ils hébergent.

Mais aujourd'hui, la machine virtuelle qui, après le serveur, était devenue peu à peu l'élément de référence pour le déploiement, est en train de céder la place à l'application elle-même.

2.2.3. Une évolution du modèle centrée sur l'application

En effet, avec le cloud, il faut revoir la façon dont les applications sont conçues, produites, déployées et mises à jour. Mais comment gérer le cycle de vie complet d'une application, avec la fourniture de ses ressources, la gestion de la configuration, le déploiement, les mises à jour logicielles, la supervision et le contrôle d'accès ?

Cette évolution vers un modèle centré sur l'application a entraîné une accélération de la mise à disposition de mécanismes d'automatisation de provisioning, de configuration, voire de l'orchestration des services proposés par les différents acteurs du cloud. Les plateformes cloud ont donc elles aussi été adaptées pour faciliter l'application de ce nouveau modèle.

Les services génériques proposés répondent aux multiples scénarios applicatifs cibles. Aussi doivent-ils offrir le niveau de configuration et d'extensibilité requis, mais surtout, ils doivent être exposés par une API. En conséquence, l'infrastructure du cloud, à l'instar de l'application, devient dépendante du code. Les descriptions de configuration peuvent donc être fournies aux services cloud *via* des API bien définies. Elles sont directement corrélées aux profils de services IaaS et PaaS que propose le cloud afin de provisionner et configurer les services cibles et leurs différentes ressources connexes. En conséquence, la composante *infrastructure as code* de DevOps, que nous avons déjà évoquée, devient incontournable lorsqu'il s'agit du cloud.

2.2.4. Une évolution des architectures

Avec le cloud, l'infrastructure est de moins en moins sous le contrôle des directions informatiques et la probabilité d'interdépendances de services liés à des SLA différents devient de plus en plus forte. Toute solution d'échelle significative étant sujette au dysfonctionnement, les architectures doivent évoluer vers un modèle *FailSafe* afin de garantir l'évolutivité, la disponibilité et la fiabilité du système. Cette évolution se traduit par de multiples activités au cours de la phase de conception :

- **Découper l'application** en composants fonctionnels en établissant des niveaux de priorité vis-à-vis de la criticité métier et du coût.
- **Définir un modèle** décrivant le comportement de l'application attendu en production (variations de charge planifiées et facilités par les modèles d'autoscaling proposés par le cloud…).
- **Définir un modèle de disponibilité** pour chaque composante de l'application et hiérarchiser les étapes de remédiation selon une classification de modes de défaillance établie en amont.
- **Identifier les points et les modes de défaillance** en adoptant, comme nous l'avons vu précédemment, une démarche d'analyse et de modélisation de la résilience des services de l'application.

Comme on peut le constater, ces éléments sont susceptibles d'impacter aussi bien les développeurs que les équipes de gestion opérationnelle.

2.3. DevOps, big data et machine learning

Big data et machine learning peuvent également contribuer à la mise en œuvre d'une démarche DevOps. Ces deux notions ne sont pas nécessairement liées mais sont complémentaires.

2.3.1. Les complémentarités entre big data et machine learning

Le big data vise à collecter un grand volume de données (individuellement potentiellement petites) ou des données volumineuses (de l'ordre du péta-octet), ayant un fort potentiel de variabilité ou de vélocité, à les stocker et à les rendre exploitables. Le big data ne cible pas nécessairement l'analyse prédictive des données. Rendre la donnée compréhensible est un but suffisant dans un premier temps.

Le *machine learning* vise à exploiter des données pour les analyser avec une approche scientifique et statistique. Il est important de noter que le *machine learning* est une discipline d'intelligence artificielle et en ce sens le but de l'analyse est clairement de faire des prévisions pour pouvoir prendre des décisions. Pour faire ces prévisions, il n'est pas toujours nécessaire de faire du big data, autrement dit de disposer d'une quantité astronomique de données.

En 2012, un statisticien nommé Nate Silver a pu prédire avec succès les résultats de l'élection américaine dans tous les états en partant de données pesant à peine 200 ko. Autre exemple, en 2014 la société française Data Publica propose un découpage pour la réforme des régions basé sur un algorithme intelligent utilisant uniquement des données fournies par l'INSEE pesant à peine 1,7 Mo. Ces exemples mettent en lumière une notion fondamentale que tout praticien de business intelligence se doit de connaître : la pertinence des données.

2.3.2. L'apport du big data et du machine learning à DevOps

Une démarche DevOps est tributaire de la collecte de données, aussi bien concernant l'usage par l'utilisateur du produit (télémétrie) que dans l'aspect opérationnel de la démarche (monitoring). En ce sens et de par son aspect agile, l'application rigoureuse des principes DevOps permet de limiter la collecte uniquement aux données pertinentes. L'application des notions d'expérimentation et d'apprentissage permet de continuellement remettre en cause ce choix des données collectées, la pertinence pouvant évoluer au cours du temps.

La collecte de données pertinentes ne préjuge en rien de la quantité de données à collecter et en cela les principes du big data s'appliquent pleinement pour réussir à exploiter ces données. Un exemple courant est l'exploitation issue des logs des différents outils de la chaîne de déploiement automatisée DevOps.

Mais l'aspect le plus intéressant reste le lien possible entre les données collectées sur l'usage fait par l'utilisateur du produit (télémétrie) et les modèles de machine learning. En effet, les modèles prédictifs de plus en plus évolués liés à cette discipline pourraient anticiper demain les futurs besoins des utilisateurs en se basant sur leurs usages actuels. Un outil puissant pour inventer les produits de demain…

La mise en œuvre de solutions liées au big data et la définition des modèles de machine learning adaptés au cycle de développement logiciel viennent enrichir DevOps qui bénéficie ainsi de l'expérimentation et du traitement des données les plus pertinentes.

2.4. DevOps et mobilité

2.4.1. Un monde en constante évolution

Si les grandes tendances de la transformation digitale s'apparentaient à une famille, DevOps et la mobilité seraient certainement frère et sœur tant cette démarche est adaptée au monde de la mobilité. En effet, aujourd'hui le monde de la mobilité est constitué d'un modèle applicatif distribué *via* des magasins applicatifs : de l'iPhone à Windows en passant par Android et Mac OS X, tous les systèmes possèdent désormais leurs *stores* qui deviennent de plus en plus le lieu privilégié par les utilisateurs pour s'approvisionner en applications.

Ce modèle a cela de particulier qu'il est réellement hyperconcurrentiel. Les éditeurs rivalisent de créativité pour répondre aux besoins des utilisateurs et il n'est pas rare de trouver une dizaine d'applications différentes pour répondre à une de nos attentes.

Face à cette concurrence exacerbée, on pourrait croire que les applications font la course aux fonctionnalités. C'est peut-être parfois le cas, mais la clé de la réussite est en réalité l'adéquation entre la fonctionnalité offerte et notre besoin réel. Et si une chose est bien constante, c'est le changement : nos besoins évoluent régulièrement et rapidement. L'application qui évoluera avec nous a le plus de chance de rencontrer le succès car elle aura un avantage concurrentiel constant sur son marché.

Il n'existe pas aujourd'hui des centaines de façon de faire pour réussir ce challenge imposé par les utilisateurs : seule l'agilité de bout en bout mise en œuvre avec une démarche DevOps permet de tenir ce rythme effréné.

Mais la mise à jour et l'enrichissement continus des applications ne sont pas le seul atout d'une démarche DevOps dans le monde la mobilité. Il existe un autre paradoxe dont nous nous accommodons sans problème aujourd'hui et qui paraîtra bien incompréhensible demain : les applications mono-plateforme.

2.4.2. Un monde multiplate-forme

Très concrètement aujourd'hui, lorsque vous voulez utiliser votre application préférée, vous devez la télécharger trois fois en moyenne : une fois sur votre ordinateur de bureau, une fois sur votre tablette et une dernière fois sur votre smartphone. Quand cette

application est bien conçue, il est possible de continuer une tâche commencée sur un autre *device* depuis n'importe lequel d'entre eux mais le plus souvent ce n'est pas le cas.

Les *applications universelles* et les *environnements de travail multistations* vont devenir la norme du monde informatique. L'agilité des architectures orientées services, voire microservices, vont s'imposer par nécessité. L'avènement de ce parc applicatif moderne ne fera que renforcer nos attentes en fonctionnalités utiles et évolutives. Nous entrerons alors réellement dans l'ère de la mobilité avec toutes ses caractéristiques espérées depuis longtemps.

Les méthodes traditionnelles de gestion de projet issues de l'analogie avec le monde du BTP seront complètement obsolètes. La démarche DevOps apporte une grande partie de la solution dans ses principes mais il est absolument certain que sa mise en œuvre évoluera grandement en même temps que la maturité du marché.

2.4.3. Un monde sans frontière traditionnelle

À cela on peut objecter que cette analyse s'applique seulement au monde du grand public et que le monde professionnel sera globalement préservé de ces bouleversements à venir. S'il est vrai que le rythme d'adaptation du monde professionnel, notamment en ce qui concerne le parc applicatif interne et les progiciels, est souvent plus lent, ce serait une erreur de penser que les impacts de ces changements majeurs seront minimes pour l'entreprise.

Il y a au moins deux raisons de penser que les professionnels seront en première ligne du changement.

Tout d'abord, il est déraisonnable de croire que les murs de l'entreprise sont imperméables à une culture grand public. De la même manière que nous avons vu le phénomène BYOD (*Bring Your Own Device*) en entreprise, les utilisateurs vont devenir de plus en plus exigeants vis-à-vis des applications professionnelles. Celles-ci, y compris les progiciels des grands éditeurs, vont devoir répondre à des attentes à la fois d'ergonomie et de réactivité qui deviendront la norme dans le grand public.

Ensuite, il ne faut pas oublier que refuser ostensiblement ces avancées laisse une image d'archaïsme au sein de sa force de travail mais également par viralité en dehors. Au-delà de l'impact sur les ressources humaines dont les talents sont de plus en plus disputés aujourd'hui, cela n'est pas de nature à créer un avantage concurrentiel sur son marché. Or, l'ergonomie et le design alliés à la réactivité sont les principaux prérequis à développer pour espérer acquérir un avantage concurrentiel sur son marché.

Pour conclure, il est certain que les exigences d'un monde mobile poussent tous les marchés dans tous les secteurs à adopter un jour une démarche DevOps. Il faut néanmoins être pragmatique et réaliste : toutes les démarches DevOps ne se ressembleront pas, elles dépendront du marché, du contexte, de la taille de l'entreprise, de sa culture et de ses enjeux. Il paraît évident que certains secteurs mettront plus de temps que d'autres à percevoir la nécessité de cette transformation.

Pour certaines entreprises, cette adoption se fera à marche forcée ; pour d'autres ce sera plus en douceur. Mais il paraît évident que pour survivre toutes passeront par là. On peut faire l'analogie avec l'agilité ou d'autres sujets de transformation digitale. On l'a vu, DevOps est complémentaire mais également indispensable au succès de ces démarches.

2.5. DevOps, innovation et design thinking

2.5.1. Les principes du *design thinking* et des méthodes d'innovation

Initialement et principalement destiné aux designers, le *design thinking* est une méthodologie de créativité inventée au milieu du siècle dernier. Cette méthode préconise de situer en amont des projets pour identifier les besoins des utilisateurs, voire les anticiper, en s'appuyant sur trois à sept étapes selon les versions autour desquelles il faut itérer, sans forcément suivre un ordre préétabli.

Les principales étapes de cette méthodologie sont :

- **Empathy.** Au cours de cette étape, l'objectif est de ressentir l'audience cible du service, pressentir ses plaisirs et ses frustrations pour mieux percevoir ces désirs. Dans les faits, une rencontre et des interviews sont très utilisées.
- **Define.** Lors de cette étape, il faut définir et affiner ses objectifs en fonction de la perception empathique que l'on a acquis. Les enjeux sont liés aux objectifs et groupés par audiences pour toujours être au plus proche à la fois du besoin et du désir de son audience cible.
- **Ideate**. C'est la phase de divergence de la méthode. Elle cherche à trouver des idées créatives, simples et intelligentes pour répondre aux besoins et aux désirs de l'audience cible. Cette phase itère de manière très régulière avec la suivante pour également diverger sur les contraintes éventuelles qui peuvent être détectées dans les phases suivantes.
- **Prototype.** C'est la phase de convergence de la méthode. L'objectif est ici de construire des solutions démontrables à partir des idées retenues lors de la phase précédente, de relever les difficultés ou contraintes qu'elles peuvent représenter et de définir les solutions appropriées pour pouvoir ensuite les tester.
- **Test.** Cette étape est celle de l'expérimentation auprès de l'audience cible, de l'apprentissage et de la rétrospective. Elle permet de confronter à la réalité l'ensemble des hypothèses retenues et éventuellement de *pivoter* pour reprendre un terme cher au Lean Startup dont la philosophie est très proche.

Si l'ensemble des méthodes d'innovation ne suivent pas nécessairement ces étapes, elles se retrouvent toutes autour des principes de divergence et de convergence centrées sur l'utilisateur. Nous pouvons facilement transposer les principes du *design thinking* sur la grande majorité des démarches d'innovation connues à ce jour.

2.5.2. L'innovation est agile

Les méthodes d'innovation sont itératives et collaboratives par nature. Il paraîtrait absolument contre-productif, pour ne pas dire impossible, de procéder différemment pour réussir à aboutir à des solutions viables. Or les principes d'itération et de collaboration sont deux des trois principes fondamentaux de l'agilité.

Pour être complètement dans les valeurs de l'agilité, il est important d'avancer par incrément. Les méthodes d'innovation ne sont pas réfractaires à l'incrémentation, d'autant plus que la complexité des produits et services inventés de nos jours rend quasiment indispensable une avancée par petit pas.

En dehors des innovations avec un périmètre relativement modeste, la grande majorité des inventions résultantes des méthodes d'innovation sont itératives, incrémentales et collaboratives, et sont donc des méthodes agiles à part entière.

Ainsi, il n'est pas surprenant de voir par exemple le Lean Startup reprendre une grande partie des principes des méthodes d'innovation étant elle-même à la croisée des chemins avec les méthodes agiles de gestion de projet plus classique.

Et si les méthodes d'innovation sont agiles, elles sont dès lors d'autant plus complémentaires avec DevOps.

2.5.3. Complémentarité avec DevOps

Si les méthodes d'innovation et DevOps sont agiles, elles n'ont pas les mêmes objectifs. Les premières visent à inventer des usages, des services et des produits. DevOps aide à les réaliser et à les mettre en œuvre rapidement et efficacement. Cependant, l'innovation n'étant jamais statique et le *design thinking* préconisant d'itérer en permanence, les solutions mises en œuvre ne peuvent également rester statiques et doivent en permanence s'adapter. Mieux encore, DevOps doit permettre de mieux entendre l'utilisateur pour alimenter de manière efficace le processus de création et d'innovation continue.

La complémentarité devient évidente. Les méthodes d'innovation alimentent les solutions à réaliser selon une démarche agile, et DevOps optimise sa concrétisation en réduisant au maximum les temps de mise en production et en maximisant ainsi les résultats de l'expérimentation.

Le feedback continu, inhérent aux méthodes DevOps et à ses mécanismes de collaboration, permet d'alimenter en permanence le processus de création des méthodes d'innovation afin de l'optimiser.

En résumé

Le digital est aujourd'hui au cœur de la production de valeur de toutes les entreprises. En ce sens, nous pouvons dire que toutes les entreprises aujourd'hui sont des entreprises digitales quel que soit le secteur d'activité ou la chaîne de valeur.

La transformation digitale vise à rendre plus agile et plus collaboratif leur informatique et leur technologie tout en étant en capacité permanente d'innover rapidement. C'est exactement l'objet d'une approche DevOps au travers d'une démarche de collaboration optimisée.

Selon une étude IDC datant de novembre 2015 et portant sur plus de 200 entreprises françaises, dont plus de la moitié a plus de 1 000 salariés, 52 % des entreprises considèrent DevOps comme le principal levier de leur transformation numérique.

Selon la même étude, les entreprises ayant mis en place une démarche DevOps bénéficient d'une nette amélioration de la collaboration entre les équipes, y compris les équipes métiers, d'une amélioration de la satisfaction client, d'un gain en qualité important, d'une vraie accélération des déploiements logiciels et d'une baisse du ratio dépenses/valeur.

DevOps est un élément clé de la transformation numérique des entreprises car complémentaire de concepts et de solutions qui accélèrent leur capacité de réponse à l'essor des technologies numériques.

DevOps respecte et étend les pratiques des méthodes agiles en s'appuyant sur ces trois grands principes : collaboration, itération et incrémentation. DevOps les met en application jusqu'au monde du support et de l'infrastructure et de la production.

Enfin, nous pouvons compléter en nous rappelant que DevOps est un accélérateur d'adoption d'un certain nombre de grands concepts d'aujourd'hui : le cloud, la mobilité, le machine learning ou encore l'innovation.

Chapitre 3

DevOps vu par les équipes développement

La mission des développeurs a bien changé… Aujourd'hui, le développeur doit s'assurer que le code écrit est évolutif, toujours disponible, avec les meilleures performances, une latence plus faible, et produit au meilleur coût.

Ce changement de perspective s'inscrit dans une évolution des architectures applicatives. Oublié le temps des architectures 3-tiers avec la période de maintenance du week-end. La continuité de service doit être assurée, à l'échelle de la planète. Les mécanismes de redondance du cloud apportent un premier niveau de réponse. Mais cela ne suffit pas. Il faut maintenant concevoir les applications comme des suites de services autonomes et faiblement couplés, hautement disponibles, que l'on peut développer, gérer en versions, déployer, et mettre à l'échelle indépendamment, sans oublier le fait que l'on ne peut pas non plus présupposer que les services que l'on va consommer seront toujours sans faille.

Au cœur de cette évolution, le développeur, qui aujourd'hui encore plus qu'hier, joue un rôle clé dans le succès d'une application. Mais pour y parvenir, il doit lui-même se transformer. DevOps devient alors un élément déterminant pour accompagner ce changement plus culturel que technique.

La vision de DevOps pour le développeur suppose donc une évolution de son rôle au sein de l'organisation. Il lui faut adopter de multiples typologies d'outils et de technologies. Il doit participer aux processus collaboratifs impliquant les autres acteurs du système d'information, au cours des différentes étapes de la réalisation d'une application.

3.1. L'évolution du rôle du développeur DevOps

La culture DevOps propose un modèle, dans lequel les objectifs assignés au développeur portent sur un périmètre plus étendu. Cette évolution de la mission du développeur se traduit par une transformation de son métier sur le plan technique et sur le plan organisationnel.

DevOps pour le développeur : avant tout, un changement culturel…

Pour le développeur comme pour les autres acteurs du SI, la transformation DevOps dépend avant tout d'un changement de culture. Cette évolution se manifeste par sa capacité à vouloir aller au-delà de son périmètre. Il s'agit pour lui d'étendre la portée de son engagement sur le plan technique en essayant d'acquérir une compréhension globale du système, ce qui suppose une collaboration active avec les équipes opérationnelles et fonctionnelles. Il se doit aussi de comprendre la stratégie et d'avoir conscience des aspects business et financiers liés au produit ou au service dont il a la charge. Il doit être curieux et adepte d'une nouvelle forme d'apprentissage en continu, plus ouverte à la prise de risque et au partage de la connaissance. Dans la suite de ce chapitre, notre objectif est d'illustrer la perspective du développeur dans la mise en place d'un processus de continuous delivery.

Même si notre discours s'oriente souvent sur des points purement techniques, il faudra bien garder en tête le fait, qu'au-delà de l'outillage qui accompagne la démarche DevOps, le principal changement pour le développeur est d'ordre culturel.

3.1.1. Une évolution des objectifs assignés au développeur

À l'origine, le développeur se concentrait d'une part sur la réponse aux exigences fonctionnelles, l'architecture logicielle, le choix des frameworks, le respect des normes de programmation, et d'autre part sur l'implémentation d'un livrable susceptible de passer avec succès les tests de réception de ce livrable. La maintenance de la solution en fonctionnement opérationnel n'était pas de son ressort, aussi n'y accordait-il pas un niveau de priorité très élevé.

Avant DevOps, pour qu'un code soit considéré comme finalisé, il suffisait qu'il passe avec succès les tests associés (et qu'il ait fait l'objet d'une démonstration fonctionnellement concluante devant le responsable produit dans le cas de l'agile).

DevOps demande un niveau d'acceptation plus élevé : pour être considéré comme achevé, ce même code doit être déployé et utilisé en production. La validation de la réception du code est donc directement tributaire de la capacité à le proposer avec un niveau de service satisfaisant sur le plan opérationnel. Le partage de cette responsabilité avec les équipes opérationnelles change la nature même du travail du développeur. Cette évolution se traduit d'abord par une transformation de son métier sur le plan technique.

3.1.2. Une transformation d'un point de vue technique

D'un point de vue technique, la démarche DevOps impacte directement la façon de concevoir et de développer les applications. Le développeur doit acquérir une compréhension globale du système, non seulement du point de vue de son implémentation interne, mais également vis-à-vis des contraintes liées à son contexte d'utilisation. Plus concrètement, cela signifie qu'il ne doit plus se limiter à la connaissance de patterns de code. Il doit également avoir connaissance de patterns liés à la résilience de l'architecture et à son déploiement dans le cloud.

Le développeur DevOps doit comprendre la stratégie produit au-delà du projet et connaître les aspects business et budget lié au produit ou au service. Il en sait également beaucoup plus sur le fonctionnel de l'application qu'il contribue à développer, sur la configuration du système d'exploitation et du réseau des environnements sur lesquels elle va s'exécuter, sur la façon dont elle va être supervisée, ou comment elle va s'adapter à la charge…

Et cette connaissance se traduit par une évolution de l'implémentation de l'application. Comme nous le verrons dans le chapitre 5 sur la qualité, le succès d'une application passe par la mise en place de systèmes de mesure et de processus de remontée d'information L'extension du périmètre de la vision technique de la solution par le développeur s'accompagne donc d'une systématisation de la mise en place de mécanismes d'instrumentation.

Enfin, DevOps encourage le développeur à progresser dans la connaissance par un apprentissage en continu. Cela requiert chez le développeur une certaine curiosité, et la volonté d'explorer les nouveautés de tel ou tel langage, ou tel ou tel framework, et la capacité à procéder à des expérimentations. Il lui faut donc disposer du temps lié à cette veille technologique et des moyens logistiques (environnements d'exécution…) lui permettant d'expérimenter ces nouvelles connaissances avant de les partager avec le reste des participants.

Cette capacité suppose qu'un minimum de latitude lui soit donné d'un point de vue organisationnel et opérationnel. Sur le plan organisationnel, la transformation DevOps du rôle du développeur se traduit par bien d'autres évolutions.

3.1.3. Une transformation d'un point de vue organisationnel

Comme nous l'avons vu, DevOps, au même titre que l'agile invite à plus de collaboration entre les équipes. Le développeur ayant une culture de l'agile devrait donc avoir un temps d'adaptation plus rapide, mais la transformation de son rôle n'en sera pas moins importante, car à la différence de l'agile, DevOps ne se limite pas à fluidifier les échanges entre le métier et les développeurs…

Avec DevOps tous les acteurs du système d'information sont concernés. En effet, DevOps fournit au développeur l'opportunité d'interagir avec de multiples autres intervenants de la chaîne de production logicielle. Le développeur DevOps prend notamment conscience des besoins et des priorités des acteurs de la gestion opérationnelle en étendant son périmètre de responsabilité et sa vision sur l'ensemble des composantes du système. Mais l'évolution de son rôle va au-delà de l'extension de sa capacité à comprendre les implications de ses choix d'implémentation et de son aptitude à prendre en considération des éléments liés à l'opérationnel dans les étapes amont de la mise en production.

En effet, avant DevOps, le manque de confiance entre les parties prenantes se traduisait par la nécessité pour les développeurs de faire appel aux responsables système pour toute action requérant la connaissance du sacro-saint mot de passe administrateur. Il en résultait des délais, des incompréhensions, voire des motifs de tension entre les équipes. Dans un monde DevOps, le développeur est habilité à exercer des responsabilités qui jusqu'alors étaient la prérogative des équipes en charge des opérations, comme par exemple, réinitialiser un environnement de développement ou de tests, sans avoir besoin de passer par une validation par les équipes directement en charge des infrastructures associées.

De même, lorsqu'un problème survient en production, le développeur est lui aussi concerné. De son implication et de sa collaboration avec l'ingénieur système ou réseau dépendra le temps de résolution du dysfonctionnement. Résoudre les problématiques au plus tôt afin de limiter leur impact est l'un des objectifs majeurs de la mise en place d'une transformation DevOps. L'époque du *ça fonctionne sur ma machine* est donc aujourd'hui bien révolue.

Cette évolution du rôle de développeur DevOps sur le plan organisationnel doit s'inscrire dans de nouveaux processus plus collaboratifs mettant en œuvre de nouvelles méthodes de travail, qui requièrent l'acquisition de nouvelles compétences et la participation de l'ensemble des acteurs du système d'information. Ces processus nécessitent l'utilisation de solutions technologiques permettant à chacun d'avoir une vue commune sur l'ensemble des actifs et environnements dont dépend l'implémentation de l'application.

3.1.4. Le continuous delivery

L'un des objectifs de DevOps est d'accélérer l'adoption d'une solution en proposant un flux de nouveaux services pour répondre aux attentes du marché en apportant de la valeur en continu. Dans ce contexte, le processus de *continuous delivery* se caractérise par un objectif ambitieux : celui de faire en sorte que le déclenchement d'une mise à jour sur le code source puisse automatiquement donner lieu à l'intégration des différents éléments constituant la solution et la mise à disposition automatisée d'un nouveau livrable.

Le développeur est un acteur majeur de ce processus. La dimension DevOps d'un projet réalisé dans le cadre d'un processus de continuous delivery se manifeste dans la capacité qui sera offerte aux développeurs et aux équipes de gestion opérationnelle de collaborer plus efficacement pour créer des systèmes complexes avec des niveaux plus élevés de confiance et de contrôle. L'objectif reste toutefois le même : s'assurer que la génération de l'application est effective, en garantissant que le code compile et vérifie avec succès de nombreux types de tests (unitaires, fonctionnels, intégration, performance) qui s'exécutent sur de multiples plateformes automatiquement provisionnées et configurées. Concrètement cela peut se traduire non seulement par une évolution du livrable applicatif et de ses dépendances mais aussi par une évolution de l'infrastructure déployée ou des environnements système.

Le processus de continuous delivery s'organise donc autour de différentes phases qui s'inscrivent dans un cycle itératif : la définition du projet, la conception, l'implémentation et les tests, l'intégration, le déploiement, et le recueil d'informations sur le fonctionnement de l'application pour en tirer un enseignement précieux dans la définition des évolutions à lui apporter. Le schéma de la figure 3.1 offre une vue synthétique de ces différentes étapes.

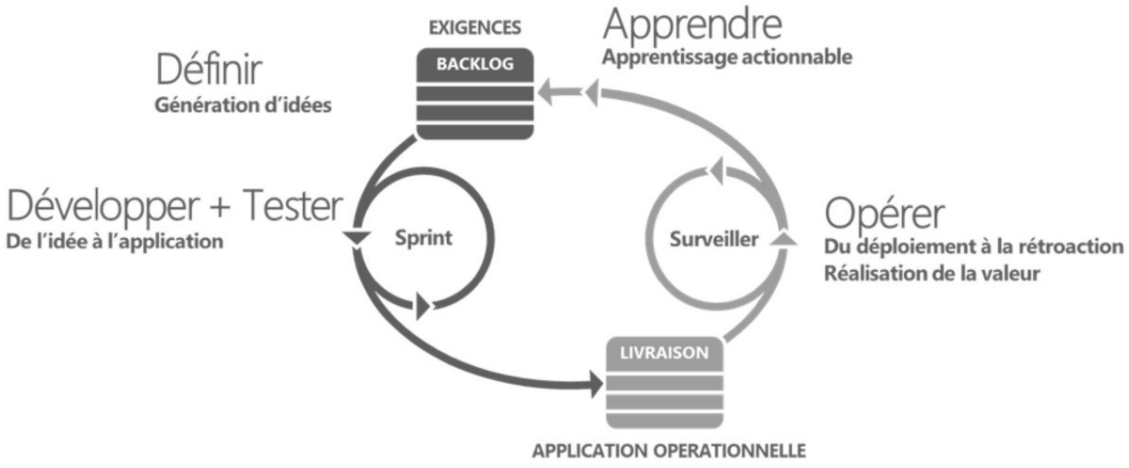


Fig. 3.1 *Le processus de continuous delivery*

3.2. L’environnement de développement

Le développeur doit disposer d’un environnement de développement complet, à jour, lui permettant de mener à bien les différentes tâches qui lui sont assignées. Cet environnement ne se limite pas à la machine sur laquelle sont installés ses outils de développement. En effet, le développeur doit pouvoir tester le code qu’il produit, et cela suppose la mise en œuvre de multiples composants logiciels (serveur web, base de données…). Enfin, il doit être représentatif de l’environnement de production pour anticiper au plus tôt les dysfonctionnements qui pourraient se manifester sur la topologie cible.

3.2.1. Installation en local de l’environnement de test

Une première solution consiste à déployer l’ensemble des composants requis sur la machine du développeur. Cette approche offre au développeur une certaine indépendance, lui donnant même l’opportunité de travailler sans connexion réseau. Mais elle se heurte à de multiples difficultés.

Le système d’exploitation

Un premier obstacle peut être lié au degré d’ouverture de l’application qui peut cibler plusieurs systèmes d’exploitation (Linux, Mac, Windows). De fait, il devient nécessaire pour le développeur de pouvoir tester l’application sur l’ensemble de ces environnements, ou de les émuler sur son poste de travail en s’appuyant sur un hyperviseur qui peut être local.

Le cas épineux du navigateur internet

Considérons maintenant le cas du développeur d’une application web. À différentes étapes de son implémentation, il doit pouvoir tester son fonctionnement avec les différents navigateurs du marché (Chrome, Firefox, Safari, Internet Explorer, Edge…), et parfois même pour de multiples versions du même browser qui n’offriraient pas le même comportement.

Ainsi, il est possible de faire cohabiter plusieurs versions de Firefox et de leur associer un profil créé *via* l’option `-ProfileManager`. Il est ensuite très simple de cibler ce profil avec l’option `-P` :

```
C:\Program Files (x86)\Mozilla Firefox\firefox.exe -P profil-de-la-version-cible
```

Dans le cas de Chrome, considérant le fait que la version se met à jour systématiquement, l’approche retenue dans le développement consiste plutôt à tester l’application systématiquement avec la dernière version et à s’assurer qu’il n’y a pas de régression de l’application due au changement de version.

Enfin, comme il n’est pas possible d’installer de multiples versions d’Internet Explorer sur Windows, Microsoft contourne l’obstacle de multiples façons. Une première approche consiste à utiliser un mode d’émulation directement intégré dans le browser (IE 11 inclut les noyaux des différentes versions de ses prédécesseurs…).

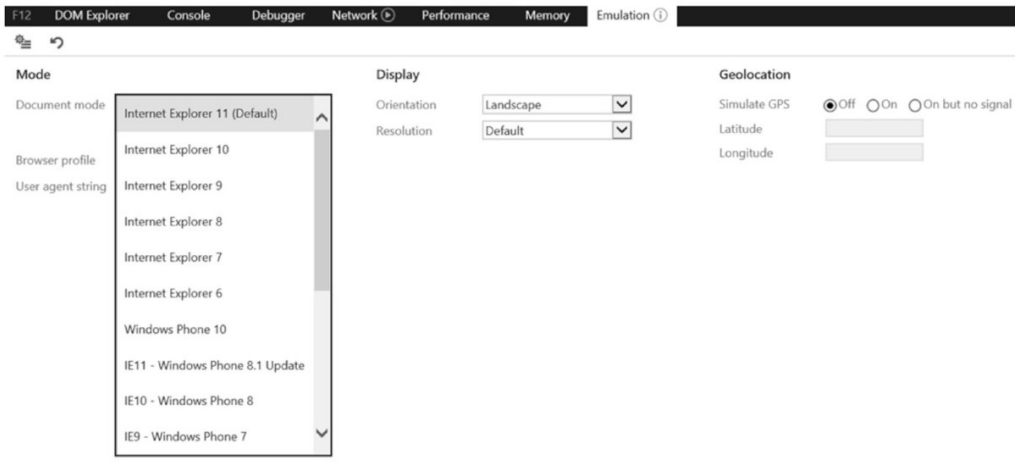


Fig. 3.2 *Émulation du comportement d'une version spécifique d'un browser IE*

En complément, Microsoft propose en téléchargement des images de différentes versions de systèmes Windows, Mac ou Linux incluant chacune une version du navigateur IE 6, IE 10, qui peut être exécutée sur de multiples hyperviseurs comme VirtualBox, Hyper-V, VMWare Player… Pour éviter d'avoir à régulièrement déployer ces images de tests (elles ont une durée de vie de 3 mois), le développeur web pourra utiliser ses propres images ou s'appuyer sur des services en ligne payants, exposant ces multiples machines virtuelles avec leur configuration spécifique de browser, et proposés par des sociétés tierces comme BrowserStack, par exemple.

Le versioning des middlewares applicatifs : la fin du DLL Hell

La machine locale de développement et de test peut nécessiter la présence de plusieurs versions du même environnement d'exécution appelé *runtime*. Ces versions peuvent elles-mêmes être utilisées simultanément par plusieurs versions d'applications et de composants. Suivant les technologies utilisées, ce type de contrainte n'est pas toujours aisé à résoudre…

Qui se souvient aujourd'hui de la période où sur Windows, les applications bâties sur le COM (*Component Object Model*) ne savaient pas faire la distinction entre des versions incompatibles de la même librairie, une situation qui parfois virait au cauchemar pour les développeurs, les opérations et les utilisateurs plongés dans ce qu'il était alors coutume d'appeler le *DLL Hell* ? Fort heureusement, aujourd'hui, cette problématique n'est plus d'actualité.

Toutefois, le développeur qui souhaiterait mutualiser l'utilisation de son poste de travail pour de multiples applications risquerait encore de faire face à des difficultés car les versions d'un middleware ne sont pas nécessairement compatibles. Cette incompatibilité interdirait au développeur de pouvoir tester simultanément différentes versions de l'application (ou différentes applications) lorsqu'elles seraient elles-mêmes associées à différentes versions de middlewares qui ne pourraient pas cohabiter. De plus, elle ne serait pas de nature à faciliter la réutilisation d'une configuration spécifique d'environnement de développement pour un nouvel arrivant sur le projet.

Prenons le cas de Java. Par défaut, les applications Java utilisent la dernière version du JRE (*Java Runtime Engine*) sans tenir compte de la version qui serait effectivement requise. Dans ce contexte, le développeur DevOps devra donc installer de multiples versions du runtime et *forcer* cette affinité par différents moyens techniques. Pour *basculer* d'une machine virtuelle Java à l'autre il suffit d'automatiser avec un script le changement de *classpath* et du répertoire dans lequel sera installée la version de la JVM (*Java Virtual Machine*) cible.

En environnement Linux, cela peut être réalisé par le simple appel de la commande `sudo update-alternatives —config java`. Enfin, il existe la solution open source jEnv dont c'est la fonction. Elle offre un langage de commande permettant d'ajouter une nouvelle version de JDK (*Java Development Kit*) comme par exemple `jenv add /Library/Java/JavaVirtualMachines/jdk17055.jdk/Contents/Home` et de décider de sa visibilité globale ou locale sur la machine.

De même avec le framework événementiel open source Node.js qui est bâti sur le moteur JavaScript de Google complété par un wrapper C++ optimisé pour les entrées-sorties. Il existe de multiples versions de ce framework qui incluent notamment un outil permettant de les gérer sur le poste de travail (NVM, le *Node Version Manager*), ce qui suivant le niveau d'installation, local ou global, permet ou non de cloisonner son usage par applications.

Enfin, il faut parfois gérer des situations plus complexes dans lesquelles de multiples versions de middleware, de composants ou d'applications doivent pouvoir cohabiter. Aujourd'hui, suivant le type de runtime utilisé, la démarche a été considérablement simplifiée par le principe du *side by side* qui permet d'exécuter simultanément ces multiples versions d'une application, d'un composant, ou d'un runtime sur la même machine. Le mode d'exécution proposé offre en général la possibilité de contrôler les associations et les dépendances entre les différentes versions de ces éléments.

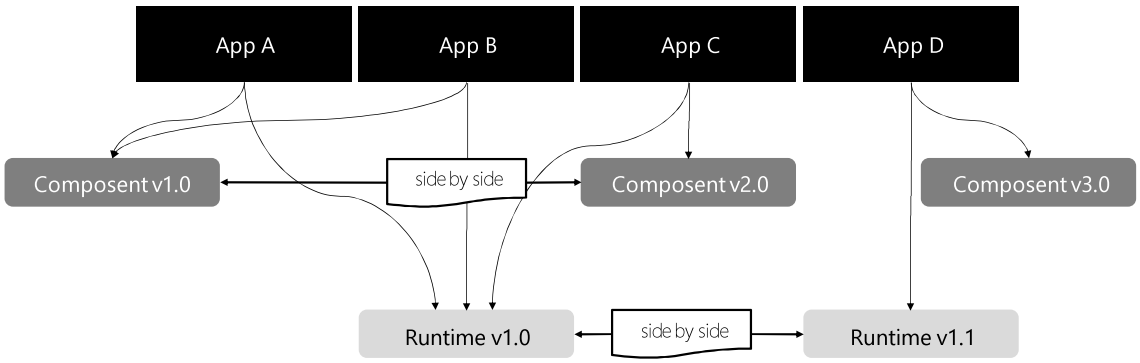


Fig. 3.3 *Exécution side by side de deux versions d'un runtime ou d'un composant*

Dans ce registre on peut citer le cas du .NET Framework. Dans sa version complète, les versions majeures peuvent fonctionner en s*ide by side* de façon globale sur le poste de travail (charge à l'application de préciser quelle version du framework elle souhaite alors cibler). Dans sa version open source .NET Core, chaque application est livrée avec le ou les modules du .NET Core dont elle dépend. Le mode d'exécution proposé est alors encore plus flexible.

3.2.2. Installation mutualisée de l'environnement de développement

Une autre solution consiste à adopter une approche plus centralisée en définissant un espace de travail pour chacun des développeurs sur un ou plusieurs serveurs. Il devient alors plus simple et plus rapide de répliquer un environnement de développement. Par contre, cela ne résout pas les problématiques de cohabitation de versions incompatibles de middleware. De plus, l'approche peut également se traduire par des effets collatéraux liés aux conflits de processus lancés par les développeurs sur le ou les serveurs mutualisés.

3.2.3. Virtualisation de l'environnement de développement

Aucune de ces deux solutions n'est totalement satisfaisante, c'est pourquoi, le développeur DevOps va chercher à disposer d'un environnement de développement qui reflète l'environnement de production sans altérer la configuration de sa machine. Pour ce faire, il va tirer parti de la virtualisation.

Son objectif est alors de disposer d'une ou plusieurs machines virtuelles qu'il pourra provisionner à volonté sur la base de modèles qui pourront être partagés par l'ensemble des développeurs. Ces machines virtuelles peuvent s'exécuter sous des systèmes d'exploitation Linux ou Windows. Elles peuvent être déployées localement (avec un hyperviseur de type 2 comme VirtualBox ou de type 1 comme KVM sur Linux, XenServer de Citrix, ESX Server de VMWare, ou Hyper-V de Microsoft) sur la machine du développeur, sur un serveur local, ou sur un cloud, qu'il soit public ou privé.

À la virtualisation des machines virtuelles s'ajoute le mécanisme de virtualisation par container, notamment avec la solution Docker, dont nous aurons l'occasion de reparler. Pour configurer son environnement de travail sur Mac ou sur Windows, le développeur peut alors utiliser des accélérateurs, comme Docker ToolBox afin d'installer les multiples outils associés (Docker Client, Docker Compose, Docker Machine, Docker Kinematic…).

Qu'il s'agisse de virtualisation machine ou virtualisation par container, il peut, suivant le contexte, s'avérer contraignant d'avoir à gérer les multiples opérations liées aux démarrages des containers ou machines virtuelles associées à l'environnement de développement. D'où l'intérêt de les automatiser…

3.2.4. Automatisation de la virtualisation de l'environnement de développement

Ces opérations de déploiement et de configuration de machines virtuelles peuvent bien entendu être automatisées par différentes techniques de scripting. Toutefois, de nouvelles problématiques viennent contrarier ce type d'initiative.

Une première difficulté est liée aux différences des langages de scripting en fonction du système d'exploitation. Par exemple, même s'il existe des sous-systèmes open source POSIX comme Cygwin ou MinGW (*Minimalist GNU for Windows*) pour Windows, le langage de scripting de référence y est plutôt PowerShell qu'un shell Unix comme Bash.

Nous touchons là du doigt une des caractéristiques essentielles de DevOps. Non seulement, le développeur DevOps doit s'intéresser au langage lié au contrôle de l'infrastructure qu'il sera amené à utiliser dans ses activités quotidiennes, mais cet intérêt doit aller au-delà de ses préférences pour tel ou tel système d'exploitation. En effet, d'un point de vue DevOps, le fait que l'on ait plus d'expérience sur un langage de script comme PowerShell ne doit pas interdire de coder un script Bash et inversement…

Le deuxième écueil est lié aux spécificités des hyperviseurs, que ces derniers soient exposés dans un cloud privé ou dans un cloud public. Nous reviendrons sur ce point dans le chapitre 4 lié aux opérations.

Bref, le développeur (et le responsable des opérations DevOps) devra savoir maîtriser de multiples langages de script et de multiples API liées à l'automatisation des environnements, que ce soit pour le provisioning ou la configuration.

Il faut toutefois nuancer la difficulté que peut représenter cette hétérogénéité. À l'échelle d'un projet, voire d'une entreprise, le choix d'un hyperviseur est souvent globalisé (notamment pour homogénéiser les formats des machines) et les solutions de provisioning des différents acteurs du cloud sont souvent ouvertes à l'usage de langages de script pouvant s'exécuter sur de multiples systèmes d'exploitation. Par exemple, le langage open sourceAzure CLI, fondé sur Node.js s'exécute depuis Windows, Mac ou Linux et permet de provisionner des machines virtuelles sur la plateforme cloud Microsoft Azure. Les autres solutions de cloud comme celles d'Amazon ou Google offrent également cette capacité à utiliser leurs API depuis de multiples environnements.

Enfin, il reste possible d'uniformiser tout ou partie du processus de mise à disposition de l'environnement, en particulier pour les développeurs, en offrant un niveau d'abstraction supplémentaire, permettant de manipuler des machines virtuelles avec un ensemble de commandes indépendantes de la plate-forme de virtualisation. C'est à ce type de problématique que répond l'outil open source Vagrant.

3.2.5. Abstraction de la virtualisation de l'environnement de développement

Vagrant est un logiciel conçu pour faciliter la gestion des environnements de développement ou de test indépendamment des technologies mises en œuvre, avec un double objectif : celui de pouvoir reproduire ces environnements à la demande et celui de donner la possibilité de partager plus facilement la construction de ces environnements.

Le principe de mise en œuvre de Vagrant repose sur la définition d'un fichier, le **Vagrant File** dont la finalité est de permettre de monter très rapidement un environnement de développement entièrement configuré. Ce fichier regroupe de multiples commandes associées à un fichier de configuration afin de définir la topologie et la configuration de l'environnement de développement. Il offre ainsi une isolation des multiples composantes de l'environnement qui peut être partagé au sein d'une équipe de développeurs afin que chacun se crée son propre espace de travail à partir de la même configuration (que ce soit sur Linux, Mac OS X ou Windows). Ainsi le code s'exécute sur le même environnement,

avec les mêmes contraintes et la même configuration quel que soit le poste sur lequel il s'exécute.

Une fois défini ce Vagrant File, les développeurs peuvent lancer la commande vagrant up en spécifiant le provider cible (en l'occurrence l'hyperviseur). Ils disposent ainsi d'un environnement de développement automatiquement provisionné avec la configuration cible. Cet environnement de type SandBox présente de multiples avantages : il peut correspondre plus facilement à l'environnement de production tout en étant directement opérationnel. Cela permet d'éliminer les dysfonctionnements liés à une potentielle divergence des environnements. Les développeurs continuent de travailler, de façon totalement transparente, sur leur propre machine, avec les outils qu'ils ont l'habitude d'utiliser (IDE, éditeurs, compilateur, debuggers, framework de tests unitaires, etc.). Une fois les machines de référence téléchargées sur le poste de travail du développeur, la suppression et la création d'un environnement de développement sont quasi instantanées et extrêmement faciles à partager avec d'autres développeurs.

Ainsi, les développeurs sont rapidement opérationnels, sans avoir à maîtriser les subtilités de tel ou tel langage de script, ou tel ou tel hyperviseur. Et bien entendu, Vagrant fonctionne aussi bien sur Linux, sur Mac, que sur Windows, ce qui garantit la cohérence de la mise à disposition de l'environnement de développement et une uniformisation de la démarche associée entre les systèmes d'exploitation.

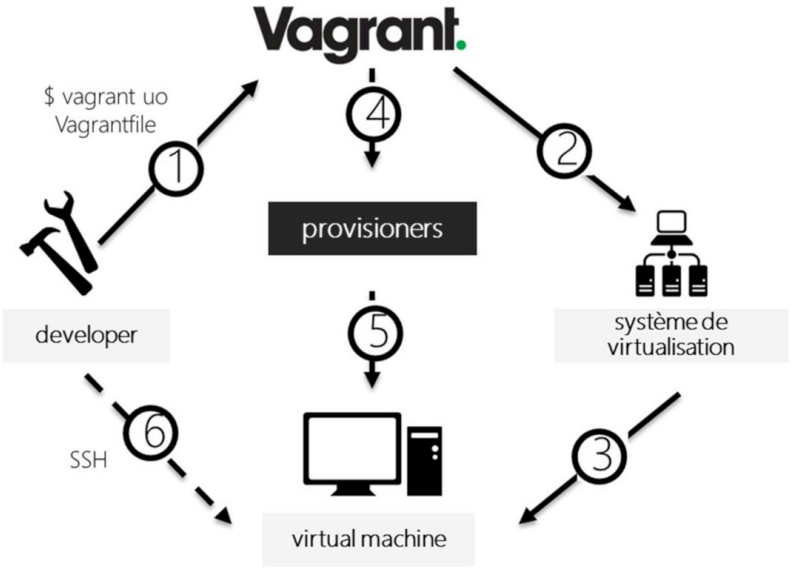


Fig. 3.4 *Principe de fonctionnement de la solution Vagrant*

Par exemple, supposons que le développeur doive déployer dans son environnement de développement une image intégrant les composants suivants : une machine virtuelle Ubuntu 14.04 préconfigurée avec Puppet, Docker, et l'image Docker microsoft/aspnet pour un déploiement sur un provider Hyper-v, soit un mélange de technologies assez variées. Vagrant s'appuie sur un référentiel commun, le site https://atlas.hashicorp.com, qui propose des machines virtuelles préconfigurées, les *boxes* pour les différents types d'hyperviseur supportés. Les commandes Vagrant permettent alors de préciser quelle machine virtuelle obtenir à partir de ce référentiel. Avec une commande vagrant init, il est très facile d'initialiser le **Vagrant File** correspondant à l'environnement souhaité. Il ne

reste plus alors qu'à spécifier, avec la commande vagrant up l'hyperviseur sur lequel la machine virtuelle spécifiée dans le fichier Vagrant File sera déployée.

Une fois cette machine téléchargée dans le répertoire cible (et mise en cache dans un répertoire local .vagrant.d\boxes lié au profil de l'utilisateur), Vagrant lance la configuration, notamment la virtualisation du réseau, la gestion de l'authentification et l'établissement d'un partage de type CIFS, pour permettre à l'ordinateur hôte d'avoir une visibilité sur le système de fichiers de la machine virtuelle déployée. Cette configuration peut être automatisée *via* un outil de configuration permettant de définir son état. Vagrant supporte nativement Chef ou Puppet. Nous reviendrons sur ces solutions dans le chapitre 4 sur les opérations. Par exemple, pour activer Puppet dans Vagrant, il suffit de mettre à jour le Vagrant File avec la ligne config.vm.provision :puppet. et de créer un répertoire manifests dans lequel le développeur ira publier le fichier déclarant la configuration souhaitée.

Précisons que Vagrant n'impose pas l'utilisation de machines virtuelles et qu'il peut s'appuyer nativement sur une virtualisation par les containers. En effet, Vagrant supporte l'utilisation d'un provider Docker et permet ainsi de construire des environnements de développement intégrant des containers construits avec Docker, ce qui offre un niveau d'indépendance supplémentaire par rapport à l'hyperviseur, tout en offrant les avantages que nous avons présentés (en particulier l'automatisation de la configuration et le partage des fichiers avec l'environnement virtualisé). Nous reviendrons sur Docker dans le dernier chapitre de cet ouvrage.

Enfin, n'oublions pas l'importance de pouvoir assurer le partage et la gestion des versions des éléments décrivant l'environnement de développement. Le développeur DevOps a alors tout intérêt à enregistrer le Vagrant File dans le référentiel de code source.

3.2.6. DevOps et environnement de développement

La dimension DevOps liée à l'environnement de développement repose sur la capacité du développeur à tirer parti des possibilités qu'offrent aujourd'hui les mécanismes d'automatisation des environnements. L'utilisation d'une solution comme Vagrant, n'est pas un passage obligé et sera sans doute plus fréquemment rencontrée dans le monde Linux que dans le monde Windows. Par contre, les mécanismes de virtualisation et d'automatisation qui la sous-tendent doivent faire partie des fondamentaux du développeur DevOps.

De multiples outils sont associés à l'activité de production logicielle du développeur comme les outils de gestion de code source, de gestion de tickets, de code review… De leur niveau d'intégration dépend aussi la qualité des processus liés au continuous delivery. En complément de ses outils de production logicielle, le développeur DevOps est amené à utiliser des outils qui permettent de faciliter la communication avec ses pairs, avec les équipes opérationnelles ou avec le système. Certaines solutions sont nativement fournies avec les environnements collaboratifs des différents éditeurs, d'autres comme Slack viennent très simplement s'intégrer dans un processus DevOps en permettant d'établir des canaux de communication entre les différents acteurs afin d'être notifiés en temps réel sur telle ou telle étape du projet.

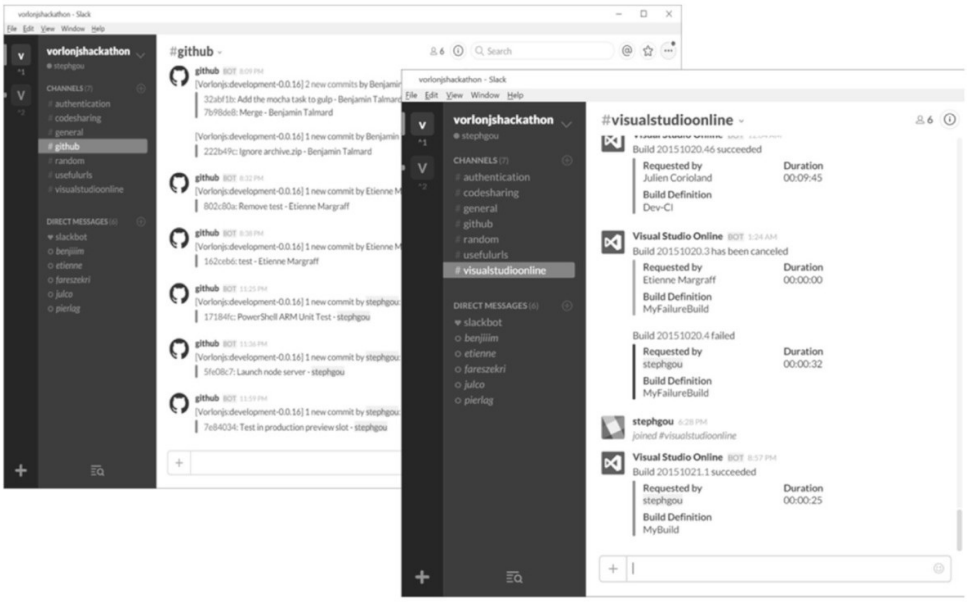


Fig. 3.5 *Suivi d'une intégration continue Git VSTS depuis Slack*

Une fois cet environnement mis à sa disposition (et automatisé par ses soins ou par ceux de l'un de ses collègues), le développeur DevOps peut se focaliser sur l'implémentation de l'application.

3.3. Le développement de l'application

L'influence de l'approche DevOps sur l'implémentation d'une application se manifeste à différents niveaux, dans les phases de conception comme dans celles de mise au point. La phase de conception logicielle est au cœur du dispositif permettant d'assurer la qualité du projet. C'est le respect des règles qui seront établies durant cette étape qui garantira la cohérence du code source et qui permettra ultérieurement de mieux en assurer la maintenance.

3.3.1. La conception

La conception d'un logiciel vu sous l'angle DevOps doit prendre en considération le contexte de mise à disposition de ce logiciel. Comme nous l'avons vu, cela implique la capacité à faire évoluer rapidement ce logiciel en fonction des attentes du marché ou du renouvellement des technologies. Cela se traduit par de multiples conséquences pour la conception. La première d'entre elles est l'absolue nécessité d'une conception dite *évolutive*.

Une conception évolutive

Quel que soit le niveau de complexité d'un système, son développement requiert une phase de conception amont pour en figer les grandes lignes. Ainsi les développeurs savent globalement dans quelle direction avancer, quels composants implémenter. Cela permet d'identifier un certain nombre de tâches dès le début du projet et de procéder à une estimation du temps requis pour leur réalisation. Toutefois, cette phase de conception

suppose chez le développeur un certain niveau d'adaptation au changement. D'un point de vue technique, cela suppose la définition d'une architecture logicielle plus flexible (notamment grâce à des patterns que nous verrons un peu plus loin). Toutefois, cette flexibilité ne doit pas se construire au détriment de la productivité, et les choix d'implémentation qui en découlent doivent être pragmatiques. C'est donc le principe de simplicité qui, souvent, doit prévaloir.

Simplicité

Ainsi, la nécessité de livrer au plus tôt de nouvelles fonctions incite le développeur DevOps à adopter une approche de conception simplifiée que certains appellent YAGNI (*You Aren't Going to Need It*).

Concrètement, cela veut dire qu'il est inutile de se lancer dans une implémentation complexe à plusieurs niveaux d'interface pour un composant qui ne serait utilisé qu'une seule fois. La première logique de ce raisonnement est purement économique. Mais là n'est pas le seul intérêt de la démarche. La complexité de l'implémentation d'un système le rend moins compréhensible et par conséquent, freine son évolutivité. La fameuse maxime d'Antoine de Saint-Exupéry, *« La perfection est atteinte, non pas lorsqu'il n'y a plus rien à ajouter, mais lorsqu'il n'y a plus rien à retirer »* pourrait donc fort bien s'appliquer à la conception logicielle…

Toutefois, il reste important de garder à l'esprit que l'architecture d'un composant sera amenée à changer, et de faire en sorte que ces évolutions soient les plus simples possibles (*design for change*).

Architecture microservices

Au-delà de la fiabilité apportée par les mécanismes de redondance du cloud, l'architecture microservices décrit une manière particulière de concevoir une application comme une suite de services hautement disponibles, autonomes et faiblement couplés, que l'on peut développer, versionner, déployer, scaler indépendamment. L'indépendance du déploiement de microservices pose également la question de la gestion des versions des API et de leur référentiel (question à laquelle les solutions d'API Management peuvent apporter un premier niveau de réponse).

C'est aujourd'hui le type d'architecture vers lequel s'orientent de nombreuses applications, et c'est donc au développeur DevOps qu'incombe la responsabilité de les implémenter. Pour ce faire, il peut utiliser une solution open source comme Docker qui facilite l'implémentation d'architectures microservices par la décomposition en containers.

Il peut également tirer parti de solution PaaS (*Plateform as a Service*) qui au-delà de l'isolation par container, inclut des fonctions liées au state management, ou à la résilience d'application microservices. Par exemple, Service Fabric est une plate-forme de systèmes distribués utilisée depuis plusieurs années par Microsoft, pour créer des applications évolutives et fiables composées de microservices s'exécutant sur un cluster. Cette plateforme héberge les microservices à l'intérieur des conteneurs qui sont déployés et activés sur un cluster en fournissant un runtime pour des services distribués stateless ou

stateful. Ce runtime peut s'exécuter sur tout type de cloud ou à demeure ; il fonctionne sur Windows mais, à terme, il supportera à Linux et les conteneurs (Windows ou Docker). Il permet de déployer et gérer un exécutable développé dans différents langages (Microsoft de .NET Framework, Node.js, Java, C++). Avec Service Fabric, l'application devient une collection de multiples instances de services constitutifs assurant une fonction complète et autonome. Ces services sont composés de code, configuration ou données et peuvent démarrer, fonctionner, être gérés en version et mis à jour indépendamment. Service Fabric permet de gérer des données persistantes au sein d'un microservice dont l'exécution est potentiellement répartie sur plusieurs machines d'un même cluster. Pour construire les services, il propose deux API (*Reliable Actors* et *Reliable Services*) qui diffèrent en terme de gestion de la concurrence, du partitionnement et de la communication.

Le rôle de l'architecte logiciel

L'architecte logiciel est à l'origine des choix de conception. Il doit disposer des compétences qui lui permettent de communiquer sur ses choix vis-à-vis de ses interlocuteurs techniques et fonctionnels, sachant que tôt ou tard, il sera amené à les reconsidérer. Il doit s'assurer que ces choix sont respectés tout en ayant une vision des évolutions à y apporter en identifiant au plus tôt les chantiers de rétro-conception de l'architecture logicielle existante. Il est partie prenante dans le refactoring, et doit s'assurer que ne subsiste pas de code *mort* dans la solution.

La compréhension globale du système qu'acquiert le développeur DevOps en fait un interlocuteur de poids face à l'architecte logiciel, qui de son côté, se doit d'être au fait de telle ou telle contrainte liée à l'implémentation de la solution. Leur interaction suppose donc une égalité dans les périmètres de responsabilité.

3.3.2. L'implémentation

D'un point de vue DevOps, l'implémentation d'une application se caractérise par une approche itérative privilégiant l'expérimentation, le partage de la connaissance, le refactoring et la mise en œuvre de patterns éprouvés. Enfin, elle suppose la mise en œuvre d'outils permettant d'observer les interactions avec le système.

Culture de l'expérimentation

L'implémentation de l'application s'inscrit dans un processus itératif. Il s'agit de pouvoir anticiper au plus tôt les choix de développement qui pourraient se traduire ultérieurement par des contraintes fortes, voire des impossibilités techniques. Dans ce contexte, le développeur DevOps ne doit pas hésiter à initier une démarche de prototypage (*design for change*, un *spike* pour employer la terminologie Scrum).

Refactoring

Pour le développeur, DevOps est synonyme de changement. Le développeur DevOps doit périodiquement revoir son code, échanger avec ses pairs et procéder le cas échéant à un refactoring du code existant. L'objectif este d'offrir une meilleure capacité de réutilisation et une optimisation de la qualité du code, notamment pour réduire la dette technique, un

sujet sur lequel nous reviendrons dans le chapitre 5 lié à la qualité. Bien qu'il n'existe pas de règles absolues, une approche par refactoring a posteriori est sans doute plus efficace et plus réaliste que l'alternative qui consisterait à proposer un modèle d'architecture logicielle susceptible de s'adapter à toutes les conditions et toutes les demandes avant même qu'elles ne se présentent.

Patterns

La réutilisation de patterns de conception logicielle n'est pas spécifique au développeur DevOps. Toutefois, elle s'inscrit totalement dans la culture DevOps de partage de retour d'expérience et d'amélioration continue. Elle permet notamment d'accélérer les développements en capitalisant sur des modèles éprouvés et en évitant les chausse-trappes dans lesquels le développeur n'aurait pas manqué de tomber… De plus, la systématisation de l'usage de ces patterns facilite la relecture de code par les développeurs qui se les approprient et offre également des mots-clés qui permettent aux développeurs d'échanger sur les éléments d'implémentation de la solution.

Le développeur DevOps doit donc avoir une bonne connaissance de modèles de code comme les patterns de création (Singleton, Abstract factory, Object pool…), de structure (Façade, Proxy, Module, Composite…), de comportement (Mediator, Publish/subscribe, Command…) ou de concurrence (Reactor, Event based asynchronous, Scheduler…). Ces patterns ont fait l'objet de multiples publications, dont l'ouvrage produit par le fameux *Gang of Four* sous le titre *Design Patterns : Elements of Reusable Object-Oriented Software*. Dans cet ouvrage de référence, après avoir exposé les subtilités de la programmation orientée objets, Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides nous proposent 23 patterns *classiques* et leur implémentation en Smalltalk ou C++.

Et pour un même pattern, plusieurs implémentations peuvent parfois être proposées. Prenons le cas du pattern Singleton. Ce pattern, sans doute l'un des plus connus, permet de s'assurer qu'une unique instance d'une classe peut être créée et expose une interface d'accès sur cette instance. Ainsi, dans son ouvrage *C# in Depth*, Jon Skeet propose cinq implémentations de ce même pattern, offrant différents niveaux de performance et d'isolation. Voici l'implémentation numéro cinq correspondant au *Fully Lazy Instanciation* du pattern *Singleton*.

```
public sealed class Singleton
{
   Singleton()
   {
   }

   public static Singleton Instance
   {
      get
      {
```

```
            return Nested.instance;
        }
    }

    class Nested
    {
        static Nested()
        {
        }

        internal static readonly Singleton instance = new Singleton();
    }
}
```

La compréhension globale du système qu'acquiert le développeur DevOps le prédispose maintenant à également avoir connaissance de patterns liés à la résilience de l'architecture et à son déploiement dans le cloud (*Command and Query Responsibility Segregation*, *Circuit Breaker*, *Compensating Transaction*, *Cache-aside*…).

Considérons une application interagissant avec des services s'exécutant dans le cloud (services applicatifs, services de base de données, services de recherche, services de cache…). Pour améliorer la stabilité d'une application, le développeur DevOps pourra faire bon usage des patterns *Retry* et *Circuit Breaker*.

Commençons par le *Retry pattern*. Notre application est susceptible d'être confrontée à de multiples erreurs *provisoires*, qui peuvent être dues à l'indisponibilité d'un service trop sollicité ou à une perte de connectivité réseau. Il est donc utile d'anticiper ce risque d'échec et de veiller à inclure une logique applicative permettant l'exécution différée d'une nouvelle tentative. Dans ce contexte, il suffit d'exécuter la même requête après un délai très court pour parvenir, cette fois-ci, à un résultat positif. C'est à ce type de logique que correspond le *Retry pattern*.

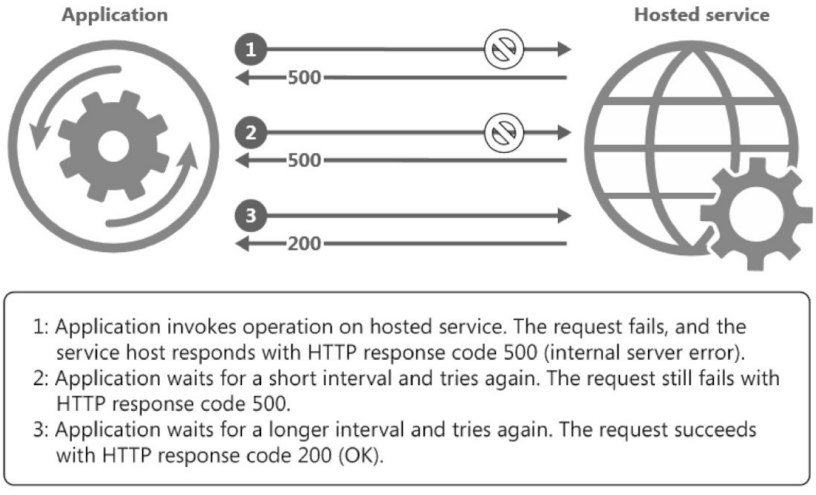


Fig. 3.6 *Retry pattern (Source : livre blanc Cloud Design Patterns: Prescriptive Architecture Guidance for Cloud Applications)*

Le *Retry pattern* peut être combiné avec le pattern *Circuit Breaker* dont l'objectif est de permettre d'éviter qu'une application soit amenée à exécuter de façon répétée une opération qui aboutirait à un échec, en laissant l'exécution se poursuivre sans attendre que l'erreur soit corrigée. Ce pattern permet aussi à l'application de savoir si le problème est réglé et si l'application peut à nouveau essayer d'exécuter l'opération. Le *Circuit Breaker* joue le rôle de proxy pour les opérations qui peuvent échouer en décidant s'il convient ou non d'exécuter l'opération en se fondant sur le nombre récent d'échecs constatés. Ce proxy peut être implémenté comme une machine à état offrant le comportement d'un disjoncteur, d'où le nom de ce pattern.

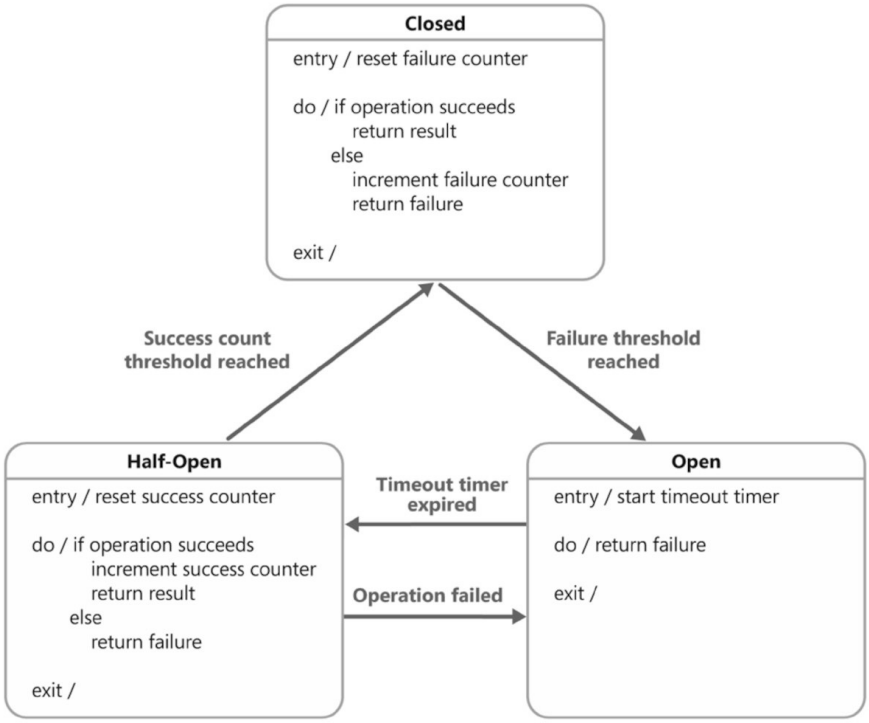


Fig. 3.7 *Pattern Circuit Breaker (Source : livre blanc Cloud Design Patterns: Prescriptive Architecture Guidance for Cloud Applications)*

En utilisant le *Retry pattern* pour faire appel à une opération *via* un *Circuit Breaker*, le code implémentant le Retry pattern abandonnera ses tentatives d'exécution, si le *Circuit Breaker* remonte une erreur qu'il considère comme non provisoire.

Enfin, pour le développeur DevOps, il ne s'agit pas de simplement connaître les patterns de code et de résilience architecturale. Il s'agit de savoir les utiliser à bon escient, et le cas échéant, pouvoir s'en affranchir.

Patterns et frameworks

Dans la plupart des cas, implémenter un pattern signifie reproduire un modèle de code dûment documenté. Mais parfois, il s'agit de s'appuyer sur un framework plus complet.

Considérons par exemple le pattern d'*inversion de contrôle* (IoC) ou plus précisément sa déclinaison dans le monde objet que l'on présente sous le nom d'*injection de dépendances* qui permet de découpler les dépendances entre objets. Il existe de très nombreux frameworks open source IoC dans les différents langages (Spring, Unity,

StructureMap…). Dans la plupart des conteneurs exposant ce type de mécanisme, on fait usage d'une configuration (effectuée par fichier ou par code) permettant d'établir l'association d'une classe avec une interface. Une fois la configuration effectuée et la cible de l'association spécifiée, c'est le rôle du framework d'injection d'instancier la classe cible.

Cela offre au développeur DevOps bien des possibilités comme par exemple celle d'activer un type de journalisation différent sans avoir besoin de modifier le code, en réinitialisant simplement la cible de l'association…

La phase de diagnostic et de mise au point

Produire du code nécessite l'usage d'outils de développement mais également d'outils de diagnostic. Et le développeur DevOps se différentie aussi par sa capacité à vérifier son code, non seulement d'un point de vue fonctionnel, mais également d'un point de vue performances.

Pour ce faire, il fera usage d'outils dont le but est d'isoler et de déterminer la cause première d'un comportement anormal du point de vue des performances, et d'identifier la configuration ou la condition d'erreur qui entrave le fonctionnement normal du système.

Ces outils lui permettent d'accéder à des informations à un instant t pour valider le bon fonctionnement de son application (debugger), mais aussi de disposer d'un historique du comportement de l'application avec une sauvegarde de la *call stack* et des variables locales (comme le propose la technologie Intellitrace par exemple). Enfin, il est en situation de pouvoir effectuer, avec ou sans debugger, une analyse de la performance globale de son code (consommation mémoire et CPU). Les outils de développement les plus avancés permettent aujourd'hui d'associer ces différents types d'information et le cas échéant d'établir une corrélation directe entre les pics de CPU et le code qui s'exécute entre deux points d'arrêt.

Par exemple, dans la copie d'écran suivante, l'utilisation d'un graphique affichant les détails de l'utilisation du processeur permet de connaître très précisément le niveau d'utilisation des ressources processeur utilisées par le code.

Fig. 3.8 *Debugging d'un service et surveillance des compteurs systèmes*

3.3.3. DevOps et implémentation

D’un point de vue DevOps, le développement d’une application se caractérise par une conception évolutive, privilégiant, la simplicité (*la sophistication ultime* selon Léonard de Vinci…). Elle emprunte différents éléments à la culture DevOps, comme l’expérimentation, le partage de la connaissance *via* les patterns et l’amélioration continue par le refactoring. Enfin, elle suppose l’acquisition de nouvelles connaissances, notamment du point de vue du système.

3.4. Le contrôle de code source

Parmi les outils mis en œuvre, dans le cadre d’une démarche DevOps appliquée au développement et à la mise en production d’une application, le contrôle de code source joue un rôle fondamental.

3.4.1. L’évolution des solutions de contrôle de code source

Le processus de développement requiert nécessairement un outil de contrôle de version du code source de l’application, également appelé SCM (*Source Control Management*). Le principe est de proposer la possibilité de conserver un historique des versions, de différencier deux versions du même code, de gérer des branches de code, de les fusionner… L’objectif de ce type de solution est de définir des règles qui régissent les archivages, les fusions et les autres interactions autour du code. Chaque acteur du cycle de production logicielle se doit de les respecter.

Qu’elles soient open source ou propriétaires, les premières solutions proposées reposaient sur un modèle centralisé (CVCS : *Centralized Version Control System*) comme Subversion, ClearCase, Visual Source Safe ou Microsoft Team Foundation Version Control (le système historique de Team Foundation Server). Aujourd’hui il semble que le mode de configuration le plus fréquemment rencontré se fonde sur un modèle distribué (DVCS : *Distributed Version Control System*) comme Mercurial ou Git.

À l’origine, **Git** a été conçu et développé par Linus Torvalds pour le noyau Linux. Aujourd’hui, c’est devenu un standard de fait. Dans le modèle proposé par Git, les développeurs travaillent avec un référentiel local qui est synchronisé avec le référentiel en ligne centralisé. Cette approche leur permet de travailler en mode hors connexion, indépendamment d’un système central et facilite la capacité d’apporter des changements rapidement. Git permet ainsi très facilement de cloner un référentiel dans lequel on peut avoir de multiples sous-référentiels. Git offre les notions de *branches* (que l’on retrouve également dans d’autres systèmes de contrôle de code source) et de *fork*. Le système de branches offre la possibilité de faire évoluer le code dans une direction différente (évolution fonctionnelle, correctif) de l’implémentation principale pour une éventuelle fusion ultérieure. Un fork est une copie personnelle d’un dépôt que l’on peut ensuite faire évoluer sans conserver de lien avec la version d’origine. Nous reviendrons sur ces deux notions un peu plus loin dans ce chapitre.

Git diffère d’autres systèmes de contrôle de version dans la façon dont il traite les données. La plupart des autres solutions traitent les changements comme des différentiels par rapport au fichier d’origine. À l’inverse Git prend un instantané de l’ensemble des

fichiers dans le référentiel. Pour optimiser le stockage, il ne stocke pas les fichiers qui ne sont pas changés entre les versions.

Enfin, Git est disponible sous la forme d'une solution logicielle que l'on peut installer à demeure sous Linux (avec une configuration SSH supplémentaire si on souhaite un serveur Git) ou sous Windows (*Git for Windows*) que l'on peut compléter avec des solutions clientes, open source comme TortoiseGit ou Msysgit, ou serveur propriétaire comme Team Foundation Server). Enfin, Git est également exposé sous forme de services en ligne comme GitHub, Bitbucket ou Visual Studio Team Services et bénéficie ainsi des avantages liés à l'utilisation d'un service managé (haute disponibilité, backup, migration…). Le schéma de la figure 3.9 représente une vue d'ensemble conceptuelle de Git.

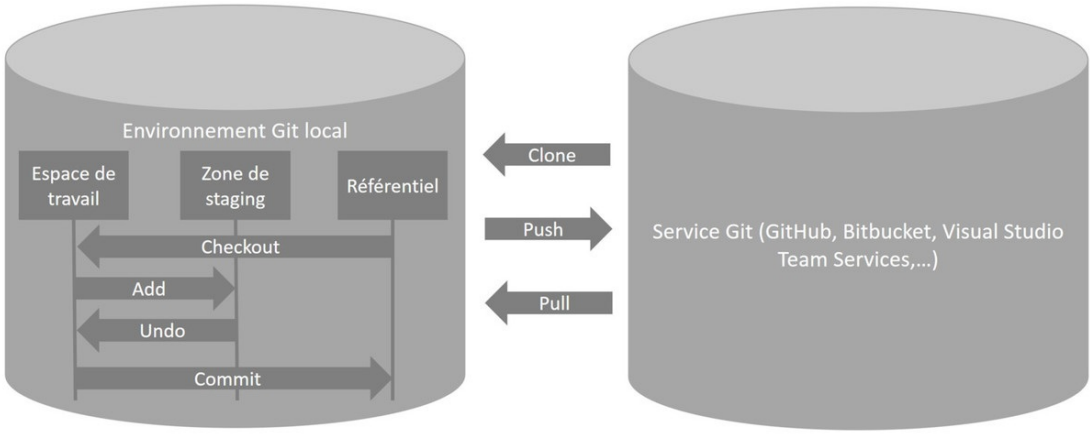


Fig. 3.9 *Interaction avec un serveur (ou un service Git)*

3.4.2. Un interfaçage du contrôle de code source avec d'autres outils

Le contrôleur de code source offre également des interfaces avec les autres outils de la chaîne de production logicielle. Par exemple, lorsqu'un développeur archive des modifications qui peuvent se traduire par une incapacité à générer un build (avec ou sans exécution de tests automatisés sur le code), cela peut entraîner des conséquences négatives pour la productivité globale. Pour éviter ce type de situation, il est fort utile, lorsque le logiciel ou le service de gestion de code source le permet, de lier le contrôle de code avec le système de gestion des builds par une définition de build d'archivage contrôlé (*gated check-in*), qui garantit la qualité du code avant son archivage.

Cette relation avec le système de build est très fréquemment mise en œuvre. Nous en reparlerons un peu plus loin lorsque nous aborderons le sujet de l'intégration continue.

3.4.3. Une extension de l'usage du contrôleur de code source

L'usage du contrôle de code source, va s'étendre dans le contexte d'une démarche DevOps. En effet, il s'agit non seulement de pouvoir assurer la gestion des versions du

code produit par les développeurs, mais aussi de proposer un outil permettant de gérer l'historique des scripts liés à l'automatisation des environnements, à commencer par l'environnement de développement, comme nous l'avons déjà signalé à propos du fichier Vagrant File. Or chaque environnement est différent (système d'exploitation et paramètres de configuration de middleware, emplacement des bases de données et des services externes et toutes autres informations de configuration qui doivent être définies au moment du déploiement).

Il ne suffit donc pas d'archiver les scripts de déploiement. La gestion de configuration logicielle suppose également la conservation (et la gestion en versions) d'un fichier de paramètres distincts pour chaque environnement. Ces fichiers seront ensuite utilisés par le même script de déploiement. Développeurs et responsables opérationnels ont donc tout intérêt à conserver et partager les fichiers de scripts et de paramètres dans le même outil de contrôle de version.

Tous les types d'objets (composants applicatifs, configuration des logiciels d'infrastructure) intervenant dans le cycle de vie l'application sont donc archivés avec leurs versions. Cela permet à chaque instant d'être en mesure de reproduire un environnement, avec sa version d'application, de système d'exploitation, de base de données ou de middleware…

3.4.4. Un outil de collaboration par excellence

Qu'ils soient open source ou propriétaires, qu'ils soient utilisés depuis une interface graphique (intégrée ou non dans l'outil de développement) ou en ligne de commande, le contrôle de code source peut être facilement mutualisé entre les deux types de profil Dev et Ops et permettre une collaboration plus active dans la mise à disposition d'éléments liés à la configuration des environnements.

En plus de permettre de déterminer les composantes logicielles d'une version, le SCM permet également de déterminer quelles sont les modifications réalisées par qui, quand et pour peu que l'information ait été renseignée, pourquoi, ce qui s'inscrit dans une vision DevOps du partage de l'information et de la traçabilité de l'évolution du système dans sa globalité.

3.4.5. Les stratégies de gestion de code source

Dans les systèmes de contrôle de code source, une branche représente une ligne indépendante du développement. En offrant un niveau d'abstraction sur le processus de publication du code, l'utilisation de branches permet aux développeurs de faire évoluer ce code simultanément sur plusieurs lignes de développement tout en préservant les relations entre ces différents chemins.

Feature Branch Workflow

Le modèle de développement Feature Branch consiste à créer une branche chaque fois qu'un développeur commence à travailler sur une nouvelle fonction. Le développement a alors lieu sur une branche dédiée plutôt que sur la branche master, ce qui facilite les

évolutions sans altérer la base principale de code. Il s'agit d'une technique éprouvée et fréquemment mise en œuvre sur les projets de développement de taille significative.

GitFlow

Un workflow plus abouti est celui que propose Vincent Driessen : **GitFlow** définit un modèle de branche conçu pour la gestion des grands projets. Le principe retenu est d'affecter des rôles aux différentes branches et de préciser la nature de leurs interactions. GitFlow met en œuvre deux branches principales pour gérer l'historique du projet, la branche master qui permet de gérer les versions de l'application, et la branche develop qui permet d'intégrer les nouvelles fonctions. Dans ce modèle, le développement de nouvelles fonctions donne lieu à la création de nouvelles branches feature dont le parent est la branche develop (et non la branche master). Lorsqu'une fonction est complète, la branche feature est fusionnée (*merge*) avec la branche develop. Lorsque la branche develop intègre suffisamment de nouvelles fonctions, une branche release est créée par un fork de la branche develop. Lorsque la release est considérée comme prête, elle est fusionnée dans la branche master avec un numéro de version. Cette approche présente également l'avantage de permettre de gérer en parallèle plusieurs branches release (et donc plusieurs versions). À cela s'ajoutent les branches de maintenance, qui sont issues de la branche master. Leur rôle est de permettre le développement de correctifs pour la version en production. Une fois le code correspondant au correctif validé, la branche de maintenance doit être fusionnée avec les branches *master* et *develop*.

GitFlow et processus de continous delivery

Il n'y a pas si longtemps l'ajout d'une branche supplémentaire entraînait nécessairement un surcroît significatif de maintenance sur le code source (efforts de fusion, gestion des conflits…) plutôt incompatible avec une intégration continue. Avec Git, cet impact s'est considérablement réduit.

Toutefois, un processus de *continuous delivery* nécessite une validation des changements sur la branche principale aussi fréquemment que possible, puisque c'est la branche à partir de laquelle on automatise le déploiement de l'application une fois le build effectué.

Pour résoudre cette problématique, certains vont jusqu'à proposer de ne plus faire usage des branches *feature* et de se limiter uniquement à des branches *release*. Plutôt que d'utiliser des branches de code spécifiques pour les nouvelles fonctions, les développeurs DevOps sont alors amenés à apporter leurs modifications sur la branche de code master, à l'aide de logique conditionnelle ou de *features switches*. Les *features switches* permettent en outre de déployer le code en production sans activer la nouvelle fonction ou de l'appliquer pour une population d'utilisateurs contrôlée selon différentes stratégies (*Canary Releases*, *A/B testing*) sur lesquelles nous reviendrons ultérieurement dans le chapitre 5 lié à la qualité.

Nous ne recommanderons pas l'abandon des features branches. Pour tirer le meilleur parti des possibilités offertes par le contrôleur de code source, il convient avant tout d'éviter les branches feature d'une trop longue durée de vie au regard de la fréquence du cycle de livraison. L'approche consiste donc à mettre en œuvre des branches release tout en

utilisant des branches feature par micro-équipes permettant de travailler sur une tâche commune de très courte durée. Comme l'affirme, Jezz Humble : « *Branching is fine, there is no problem with branching it's just this practice of feature branching where developers don't merge into trunk regularly which is problematic and we still see that today frankly, a lot, much more than we should.* »

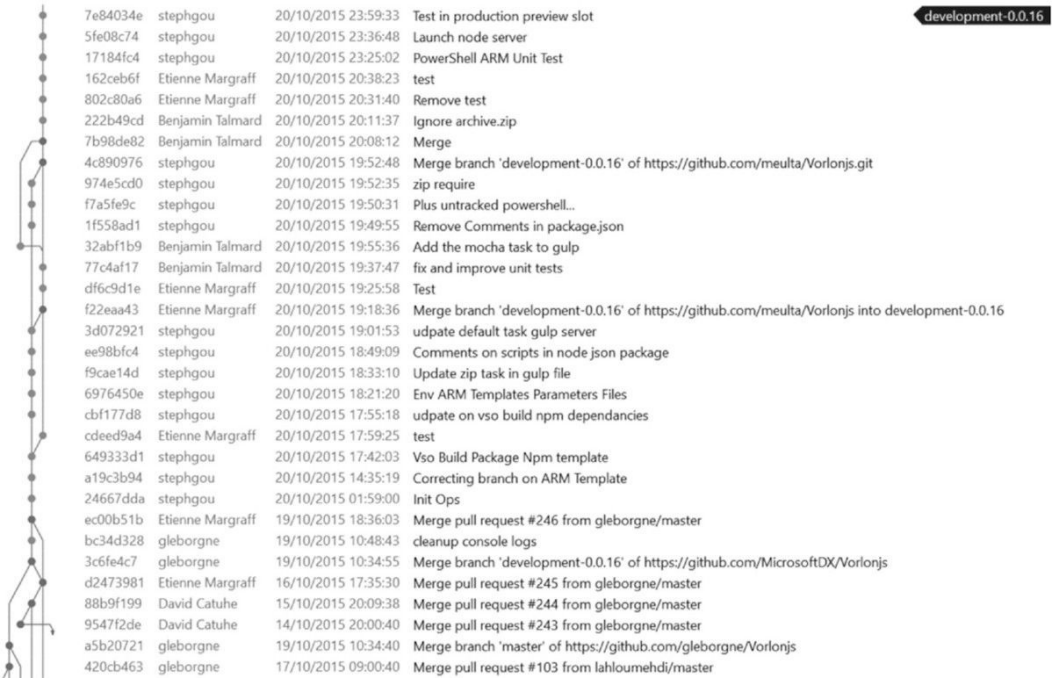


Fig. 3.10 *Contrôle de code source et stratégie de branches*

3.5. Les solutions de gestion de packages

Le développement d'applications invite à la réutilisation de ressources, de composants, de code et autres artefacts pour gagner en efficacité, en fiabilité et en maintenabilité. Cette situation se traduit par la nécessité de disposer de librairies pouvant être partagées au sein de l'équipe de développement ou, à une plus grande échelle, sur des référentiels proposés par des éditeurs ou par une communauté. La gestion de ces librairies, de leurs dépendances et de leurs versions est un processus souvent complexe, qui requiert l'utilisation de solutions de package management.

3.5.1. L'utilisation d'un gestionnaire de package

Les solutions de packaging permettent de rechercher, d'ajouter et de mettre à jour des librairies partagées sur des référentiels publics comme npm.org pour les développeurs JavaScript/Node.js ou nuget.org pour la plateforme de développement .NET. Leur principe est de permettre la construction et la définition des dépendances des librairies qui seront incluses dans l'application au cours du processus de packaging, et le cas échéant publiées par la suite dans le référentiel pour une réutilisation par d'autres développeurs.

Considérons l'exemple de npm. Ce service propose un dépôt public de référence pour tous les modules nodejs construits et maintenus par la communauté : la plupart sont hébergés dans GitHub. En complément, il offre un programme en lignes de commande pour

télécharger, installer et gérer les modules. Le service npm propose deux modes d'installation de ces packages : local, pour une utilisation spécifique au développement d'une application avec un répertoire d'installation différent, et global à l'échelle du poste du développeur. La configuration des modules requis (et de leur version) est réalisée grâce à un fichier manifeste qui décrit les dépendances. La copie d'écran suivante illustre l'association entre ce fichier package.json et les packages référencés grâce à npm.

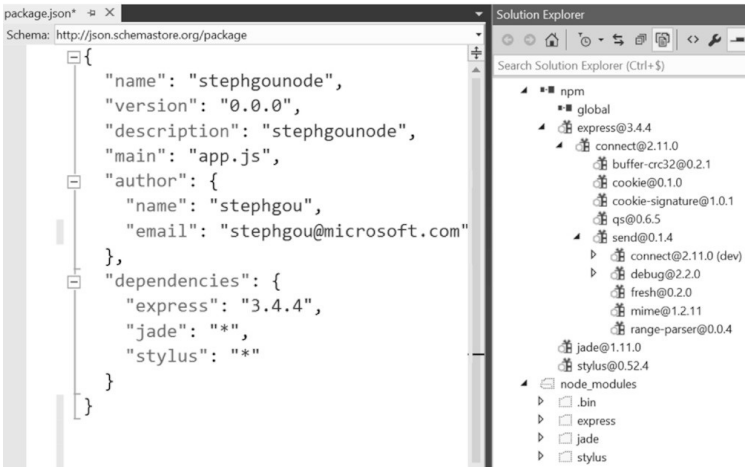


Fig. 3.11 *Configuration des dépendances d'une application Node.js via un fichier package.json*

Le comportement de NuGet est très similaire. La différence réside dans le nom du fichier baptisé package.config et son format XML plutôt que json. De même RubyGems, propose un format standard gem pour la distribution de programmes et librairies Ruby. NuGet et npm offrent, l'un comme l'autre, des utilitaires en ligne de commande pour créer des packages à partir d'un fichier de spécification et le publier si nécessaire. Le développeur DevOps a donc toute latitude pour automatiser les processus de packaging et de publication de ces packages par une extension du processus de build à l'aide de scripts déclenchés en phase de pré- ou de post-génération.

3.5.2. Le package management des frameworks Web Frontend

Considérons le cas des développeurs web. Pendant des années, la plupart ne se sont pas préoccupés du poids des pages de leur site web qui, avec l'imbrication de toujours plus de scripts et de médias, ne cessait de croître. Pourtant, une page trop volumineuse impacte le délai de chargement, a fortiori pour une application mobile qui ne disposerait pas d'un accès à un réseau haut débit. C'est la cause d'une insatisfaction potentielle de l'utilisateur et donc le risque de le perdre. En outre, pour le classement des sites, les moteurs de recherche tiennent comptent du temps de chargement des pages. Autant de raisons de veiller à optimiser le contenu des pages. De multiples techniques ont donc été mises au point pour réduire le volume des pages (minification de CSS/JS, avec des frameworks comme JSHint ou JSLint) et augmenter ainsi les performances du site web (concaténation des fichiers CSS et JavaScript, compression des images, suppression des instructions de debug de scripts).

De plus, avec la généralisation de l'usage de jQuery, le langage JavaScript s'est répandu au point de déclencher des initiatives open source comme l'émergence de compilateurs qui à partir de langages plus évolués tels que TypeScript ou CoffeeScript permettent de générer du JavaScript. Dans un même ordre d'idée sont apparus des pré-processeurs comme SASS (*Syntactically Awesome Style Sheet*s) ou Less (*Style Sheets Language*) qui permettent de générer dynamiquement des feuilles de styles CSS. Ces extensions facilitent la production du code généré. Enfin, pour faciliter le développement de sites web dynamiques, des frameworks ont exploité ces langages de plus haut niveau (comme bootstrap, par exemple, qui est composé d'une série de modèles de feuilles de styles Less, ainsi que d'extensions optionnelles pour le langage JavaScript).

Tout cela a rendu possible la construction d'applications de plus en plus complexes… et a suscité de nouveaux besoins, comme par exemple, la possibilité de mettre en place des liaisons de données (*databinding*) entre les contrôles HTML de présentation et le JavaScript client et l'arrivée de nouveaux frameworks pour y répondre, comme Angular ou Knockout.

Cette évolution du développement web s'est traduite par deux conséquences non neutres dans une perspective DevOps :

- **Avec l'utilisation de langages de plus haut niveau, ce n'est plus le code source qui est directement publié sur le serveur**. Il faut donc maintenant gérer le pré-processing du code client JavaScript ou CSS, qui auparavant était exempt de toute compilation. Ce pré-processing peut être réalisé directement sur le poste du développeur web DevOps ou sur un serveur de compilation. Nous reviendrons sur ce point dans le sous-chapitre consacré au système de build (§ 3.6).
- **Il devient absolument nécessaire de gérer le packaging de ces différents éléments et de leurs dépendances**. De même que dans le cas de solution de gestion de librairies comme Nuget ou NPM, il faut donc proposer un mécanisme pour un packaging de ces librairies côté client (AngularJS, JQuery, BootStrap…). On retrouve d'ailleurs la même logique de fichier manifeste descriptif des dépendances entre les différentes librairies.
  Par exemple, la solution open source Bower apporte une vraie réponse à la deuxième de ces problématiques. Bower offre des extensions pour faciliter l'intégration de ce mécanisme de gestion des packages directement au sein des outils de développement. Cette solution est optimisée pour les serveurs frontaux. Elle utilise une arborescence de dépendances à un seul niveau, ne nécessitant qu'une seule version de chaque package, réduisant le chargement de la page au minimum. Elle permet d'automatiser le téléchargement et l'installation des packages en conservant leur trace dans le fichier manifeste bower.json. La copie d'écran suivante illustre le contenu de ce fichier déclarant les packages de ressources.

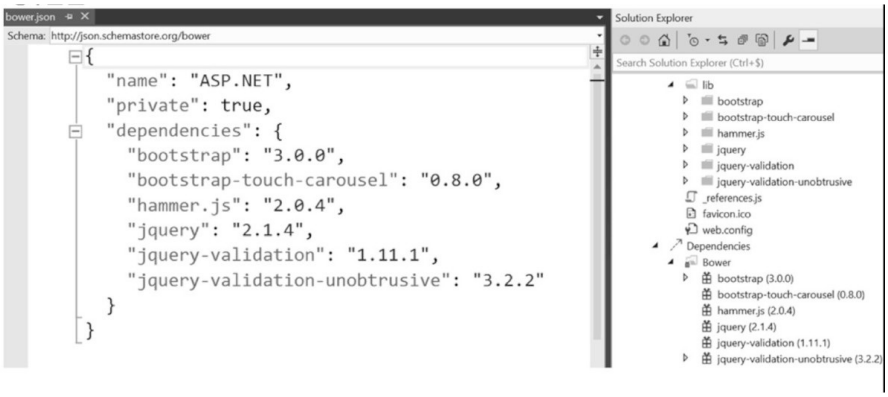


Fig. 3.12 *Configuration des dépendances liées au script client d'une application web via un fichier bower.json*

3.5.3. Les dépôts privés

Le principe proposé pour partager des librairies à l'ensemble des développeurs de la planète est naturellement applicable à une échelle plus réduite. Aussi bien Npm que Nuget peuvent ainsi être utilisés pour packager des librairies dans le cadre d'un projet de développement logiciel pour en restreindre l'accès et pour filtrer les packages que l'on souhaite voir utiliser au sein du développement. Npm.org propose ce type de dépôt en mode service (payant) mais il est également possible de bâtir un référentiel sur sa propre infrastructure. De même, il est possible de définir un serveur Nuget interne, afin qu'il expose la source des données sur le protocole OData (*Open Data Protocol*). Il est naturellement possible de mettre en place des environnements du même type pour le partage de librairies ou de modules sur d'autres types de langages (Java, C++, Python, Fortran…).

3.5.4. Les regroupements de référentiels de librairies

Parfois, le même développeur peut être amené à utiliser des dépôts qui vont varier en fonction de son contexte. Ce type de situation peut représenter une contrainte d'un point de vue DevOps parce qu'un changement de référentiel de librairie peut requérir un changement de configuration sur le poste du développeur.

Pour gérer ce type de problématiques, l'approche DevOps pour le développeur va consister à fusionner plusieurs référentiels différents en un seul et automatiser le processus permettant d'associer le dépôt de packages cible en fonction des circonstances. Cela permet d'offrir une abstraction des détails de configuration des dépôts réels et autorise différents scénarios tels que la mutualisation des référentiels, l'organisation de référentiels distincts pour différentes branches ou projets de développement. L'objectif est d'offrir plus de souplesse dans l'accès à différentes versions de librairies, qui en fonction du contexte, pourront être utilisées avec le minimum d'effort pour le développeur.

3.6. Le système de build

Le système de build cible l'automatisation du processus de création d'une version de logiciel et inclut non seulement la compilation de code source en code binaire, mais aussi

la production des librairies correspondantes. Ce processus peut également donner lieu à l'exécution des tests automatisés, que nous traiterons dans le chapitre 5 présentant l'application de la démarche DevOps à la qualité.

3.6.1. Les définitions de build

La définition d'un build doit être indépendante de l'environnement sur lequel il sera exécuté, d'où l'intérêt des solutions de package management (en dépôts publics et/ou privés) que nous avons déjà évoquées. L'objectif est de faire en sorte que les liens vis-à-vis de modules externes puissent être correctement gérés lors du build. Cette indépendance permet également de s'affranchir de toute spécificité liée à l'environnement dans lequel l'application a été développée. C'est également dans la définition de build qu'est référencé le premier niveau de tests automatisés destiné à accroître la qualité du code produit. Enfin, comme tout artefact lié au cycle de vie de l'application, la définition du build doit être archivée dans le contrôleur de code source ou sur le serveur de build lui-même (comme c'est le cas pour VSTS…).

Il est souvent utile de pouvoir conserver le lien entre la version du code source prise en compte par le build et le numéro du build. Cette fonction peut être nativement assurée par la solution de build. Si l'on souhaite inclure ce numéro de build dans le binaire produit, le développeur DevOps peut alors étendre la définition de build pour y inclure des informations directement issues de la compilation. Prenons l'exemple d'une extension de build pour *Visual Studio Team Services* (on pourrait arriver à un résultat similaire en personnalisant une définition de build Maven géré dans Jenkins pour un code Java). Le principe consiste à développer une librairie exploitant les interfaces d'extension de la solution cible. Là encore, une bonne occasion de faire appel à un package manager comme Nuget.

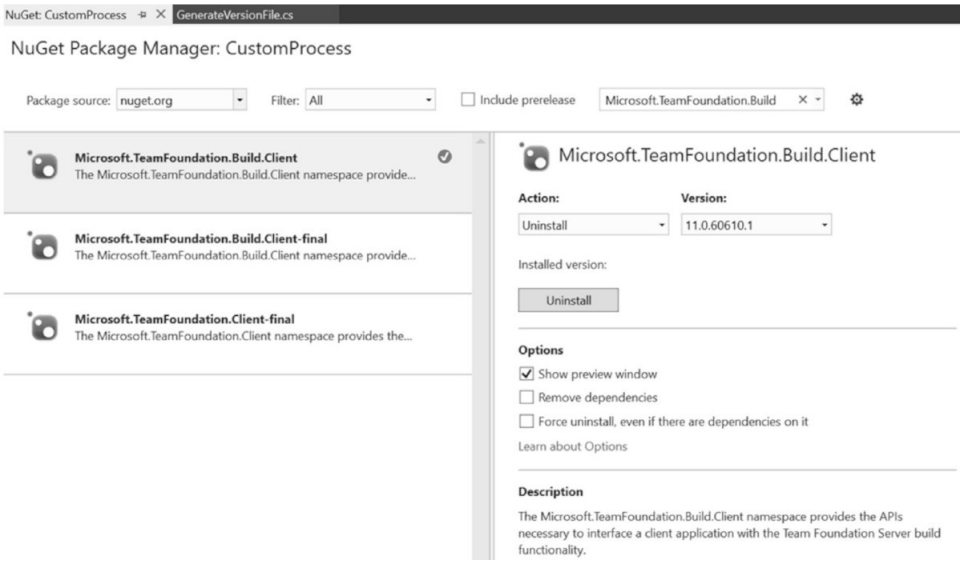


Fig. 3.13 *Personnalisation d'une définition de build : obtention du sdk*

Le développeur DevOps peut alors produire le code correspondant à la fonction souhaitée en référençant la librairie permettant d'étendre la solution de build. Le code suivant

illustre la personnalisation d'une définition de build par l'implémentation d'une activité personnalisée.

```
using Microsoft.TeamFoundation.Build.Client;
using System;
using System.Activities;
using System.IO;

[BuildActivity(HostEnvironmentOption.All)]
public class GenerateVersionFile : CodeActivity
{
    public InArgument<DateTime> InputDate { get; set; }
    public InArgument<string> Configuration { get; set; }
    public InArgument<string> FilePath { get; set; }
    public InArgument<string> TemplatePath { get; set; }

    protected override void Execute(CodeActivityContext context)
    {
        var timestamp = this.InputDate.Get(context);
        var configuration = this.Configuration.Get(context);
        var version = new Version(timestamp.Year, timestamp.Month * 100
                    + timestamp.Day, timestamp.Hour * 100
                    + timestamp.Minute, timestamp.Second);

        var filePath = this.FilePath.Get(context);
        var versionFileTemplate = this.TemplatePath.Get(context);
        File.WriteAllText(filePath, string.Format(File.ReadAllText(
        versionFileTemplate), version.ToString(4),configuration));
        }
    }
}
```

Il suffit alors de compiler la librairie associée à cette activité personnalisée afin de l'inclure dans la définition de build. Dans l'exemple ci-dessous, (qui correspond à l'ancien système de build édité sous Visual Studio), le template de build est modifié pour utiliser la librairie *GenerateVersionFiIe*.

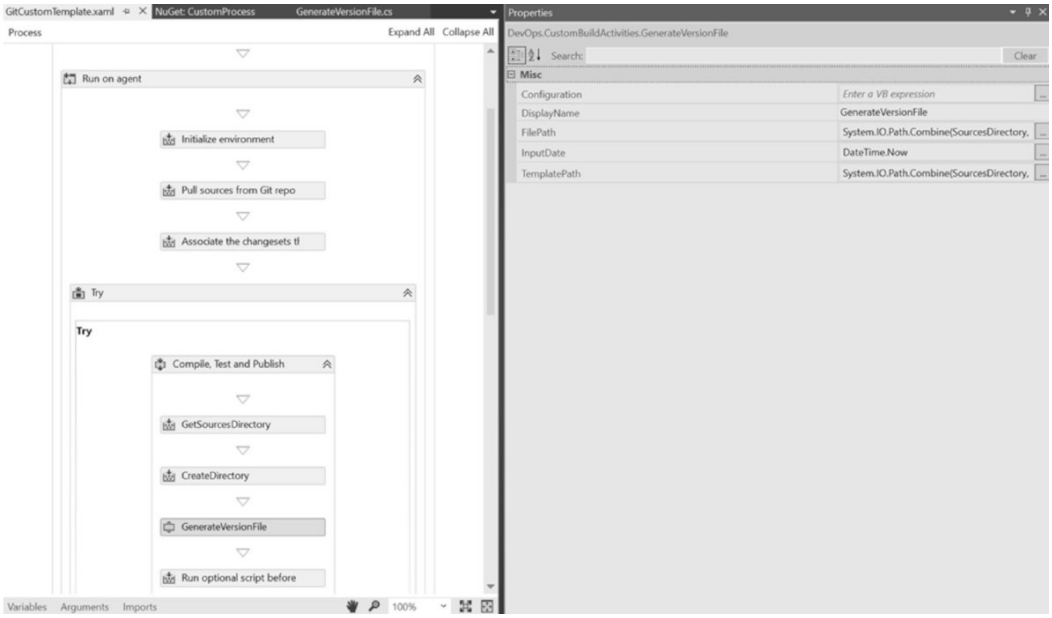


Fig. 3.14 *Personnalisation d'une définition de build : inclusion de l'activité personnalisée*

3.6.2. Les utilitaires de build

Certains langages (comme JavaScript) sont dits *interprétés*, ce qui signifie qu'il n'est pas nécessaire de procéder à une phase de compilation avant leur exécution sur la machine. Pour d'autres langages (Fortran, C, C++, Java, C#…), cette compilation est un prérequis. L'automatisation de la génération de la librairie ou de l'exécutable est réalisée grâce un type de fichier appelé makefile (et ce quel que soit l'environnement Unix, Linux, Windows…) par une compilation et une liaison du code source. La syntaxe de ces fichiers varie en fonction du compilateur du langage et de la plateforme d'exécution.

De multiples utilitaires (Make, Maven, Ant, MSbuild, Gulp, Grunt, Gradl, Cmake…) viennent donc compléter les outils de développement. L'intégration de multiples utilitaires de compilation afin de permettre de générer une application, bâtie sur des services dépendants chacun de *middlewares* différents, est aujourd'hui facilitée par la mise à disposition de solutions de build centralisées.

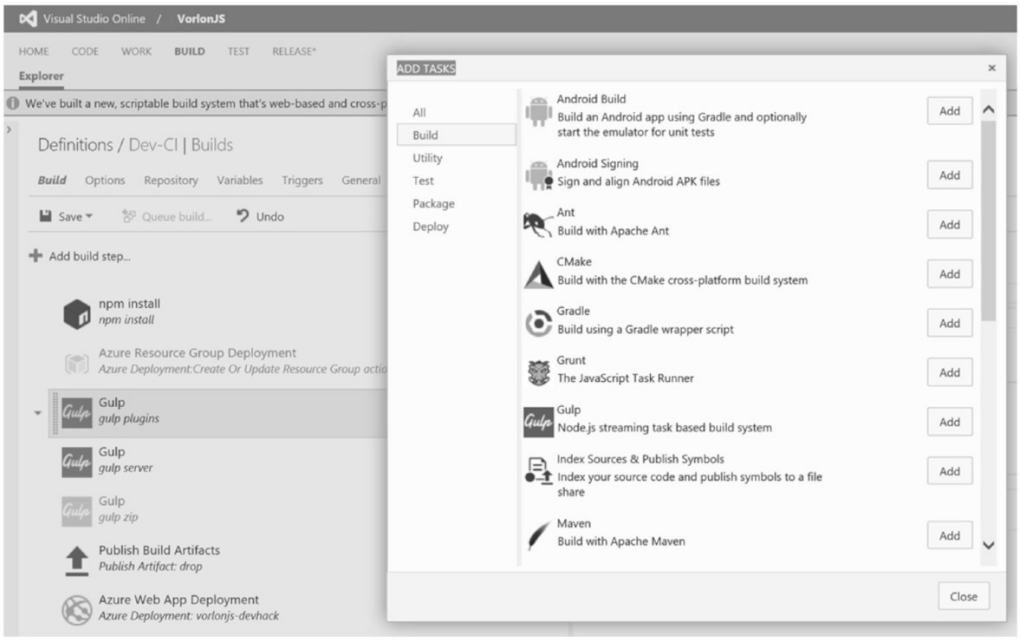


Fig. 3.15 *Grande variété des utilitaires de build*

3.6.3. Les serveurs de build centralisés

Les serveurs de build centralisent l'exécution des processus de build en utilisant les utilitaires précédemment décrits. L'objectif est de s'affranchir de toute dépendance vis-à-vis d'un environnement de compilation. L'utilisation d'un serveur de build centralisé permet également d'organiser les étapes du projet selon un agenda permettant de gérer des contraintes liées au temps de production des builds (avec par exemple, une exécution nocturne des compilations et tests, si elle s'avère particulièrement coûteuse en temps et en ressources).

La solution de build centralisée joue un rôle primordial dans un processus de livraison logiciel. Elle se doit d'être efficace et transparente, afin que les développeurs puissent totalement se concentrer sur la production de leur livrable. Au-delà du temps gagné sur la génération de livrables ayant fait l'objet d'un premier niveau de validation, les implémentations de ce type de solution permettent d'obtenir un premier niveau d'information sur l'avancement du projet et sur la qualité du code produit et de le partager entre les différents acteurs du cycle de vie logiciel de l'application.

Le déclenchement du processus de build est réalisé sur la base d'une planification, ou suite à un évènement comme un archivage de code (dans le cas de l'intégration continue, comme nous le verrons un peu plus loin dans ce chapitre). C'est également à partir de ces serveurs que sont lancés les déploiements automatisés sur les différentes plateformes. À ce titre, ils constituent un élément clé du dispositif DevOps.

Il existe de nombreuses solutions commerciales intégrant des serveurs de build (TeamCity, Team Foundation Server, Visual Studio Team Services…). Dans le monde open source, le serveur de build le plus connu est très certainement Jenkins. Il s'agit d'un outil qui a été développé en Java par Kohsuke Kawaguchi, alors qu'il travaillait chez Sun. La solution

s’appelait initialement Hudson, mais le rachat de Sun par Oracle a entraîné un fork de la solution sous le nom de Jenkins, qui est maintenu depuis par une communauté très active.

Parmi les caractéristiques reconnues de cette solution on peut mettre en avant son ouverture et sa flexibilité. Automatiser l’intégration de multiples projets suppose la capacité à gérer de multiples langages : Jenkins supporte Java, .NET, C++, Ruby, PHP… De plus, un serveur de build doit pouvoir s’intégrer sur de multiples types de contrôleurs de code source, bâtis sur de multiples standards. Jenkins propose donc un système d’extensibilité par plug-in (plus d’un millier…) qui permet, par exemple, de définir un build dans Jenkins sur la base de sources gérés sur un serveur Team Foundation ou sur GitHub et de lancer des commandes Docker au cours de ce build.

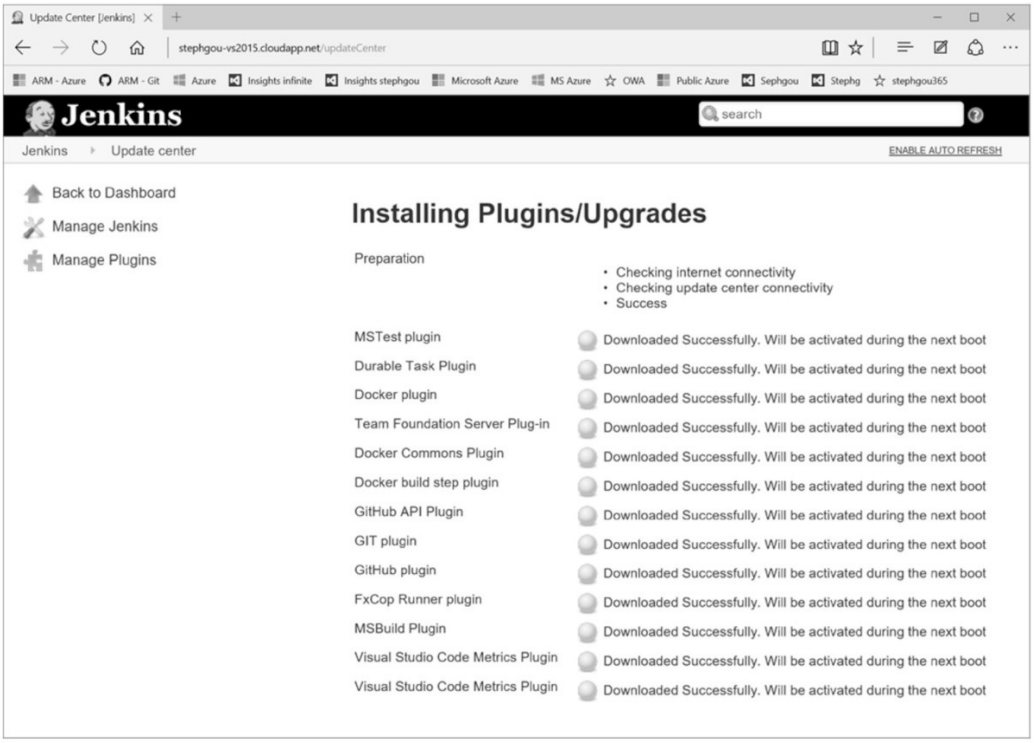


Fig. 3.16 *Extensibilité par plugins de la solution open source Jenkins*

3.6.4. Le processus de build du code client

La centralisation des processus de build n’exempte pas le développeur DevOps de tâches liées à la compilation au niveau de son poste de travail. Comme nous l’avons vu dans le paragraphe sur le package management des frameworks web frontend, l’évolution du développement web impose la gestion d’un pré-processing du code client JavaScript ou CSS, qui auparavant était exempt de toute compilation.

Certes, cette nouvelle contrainte n’est pas fondamentalement révolutionnaire pour un processus de production logicielle classique, mais elle ajoute une dimension nouvelle liée à la nécessité de pouvoir automatiser un certain nombre de tâches côté client afin de compiler, tester, et réduire la taille du code. Ce type de tâches devient rapidement répétitif et sujet à erreur. Le développeur DevOps se doit donc de disposer d’un environnement lui permettant de définir un processus d’automatisation de cette pré-production.

Comme souvent, la définition du processus de génération est plus complexe que l'exécution individuelle de chacune des tâches correspondantes. Mais là encore, cet investissement est rentable, car non seulement il permettra de gagner du temps mais il évitera bien des erreurs. La démarche d'automatisation de ce type de tâche est assez classique. Il s'agit en général d'identifier les tâches répétitives les plus longues comme la concaténation de fichiers, la suppression des instructions de debugging, la réduction de la taille du code…

Gulp et Grunt sont des systèmes de build qui permettent d'automatiser ce type de tâches. Grunt utilise des fichiers de configuration JSON. Gulp est construit sur Node.js, et le fichier Gulp dans lequel sont définies les tâches est écrit en JavaScript (ou en CoffeeScript). Il permet ainsi de compiler et analyser le code permettant de générer JavaScript ou CSS, dès la modification du fichier. Avec ces outils, le build de la partie frontend peut être réalisé aussi bien côté client que côté serveur. Pour le développeur DevOps, cela implique que les outils de build qu'il utilise puissent également être mis à disposition sur un serveur centralisé.

Dans l'exemple suivant (extrait d'une solution open source baptisée *Vorlon.js*), le code est développé en TypeScript. Il faut inclure dans le script Gulp, une tâche permettant d'automatiser la compilation du code TypeScript en JavaScript à l'aide du module gulp-typescript. De plus pour faciliter son déploiement, il peut être utile d'archiver l'ensemble des fichiers à l'aide du module gulp-zip. Enfin, il faut pouvoir automatiser les tests avec Mocha, une infrastructure de test JavaScript bâtie sur Node.js. Le principe est de pouvoir offrir une série de tests asynchrones et de remonter les exceptions non gérées. Pour en faire usage depuis le script Gulp, il suffit de faire appel au module gulp-mocha. Une fois les prérequis en termes de modules définis en en-tête du fichier gulpfile.js, il suffit d'utiliser les alias correspondants en déclarant différentes tâches que l'on peut ensuite séquencer dans une tâche dite default. Le déclenchement de cette tâche default peut alors être commandé manuellement ou associé à l'observation du système de fichier par l'appel à la commande gulp.watch qui dans notre contexte va s'activer lorsqu'un fichier TypeScript sera modifié. Ainsi, à chaque modification, le fichier JavaScript est également mis à jour de façon totalement transparente pour le développeur.

```
var gulp = require('gulp'),
    typescript = require('gulp-typescript'),
    mocha = require('gulp-mocha'),
    zip = require('gulp-zip');


gulp.task('typescript-to-js', function() {
  var tsResult = gulp.src(['./**/*.ts', '!./node_modules', '!./node_modules/**'], { base: './' })
                    .pipe(typescript({noExternalResolve: true,
                                      target: 'ES5', module: 'commonjs'}));


  return tsResult.js
          .pipe(gulp.dest('.'));
});
```

```
gulp.task(‘zip’, [‘tests’], function() {
    gulp.src(‘./**/*.*’)
        .pipe(zip(‘archive.zip’))
        .pipe(gulp.dest(‘dist’));
});

gulp.task(‘tests’, [‘typescript-to-js’], function() {
    gulp.src(‘test/test.js’)
        .pipe(mocha());
});

gulp.task(‘default’, [‘zip’], function() {
});

/**
 * Watch typescript task, will call the default typescript task
   if a typescript file is updated.
 */
gulp.task(‘watch’, function() {
  gulp.watch([
    ‘./**/*.ts’,
  ], [‘default’]);
});
```

3.6.5. Les environnements *cross-platform*

Il reste toutefois une question complexe à laquelle ces solutions ne répondent pas aussi simplement. Il s’agit de la problématique de création d’un environnement de build hybride de cross-compilation permettant de cibler un environnement depuis un poste de développement ne correspondant pas nécessairement à cet environnement cible.

Un premier exemple de ce type de problématique est adressé par la solution Xamarin.

Un code source pour les unifier tous…

Xamarin est une évolution du projet open source Mono qui ciblait le portage de .NET sur plateforme Unix ou Linux. Xamarin cible le développement d’applications natives sur de multiples types de devices (iOS, Android, Windows 10, Windows 10 Mobile…) à partir du même code source.

D’un point de vue développement, cela signifie la possibilité de pouvoir utiliser des librairies communes à l’ensemble des environnements. Par exemple, pour l’accès aux données, le développeur pourra utiliser la librairie open source SQLite qui offre un moteur de base de données permettant de gérer le stockage des données et d’offrir les méthodes pour les lire et les écrire. Pour assurer la compatibilité multi-plateformes, le moteur

SQLite fonctionne sur une grande variété de plates-formes. Il est directement inclus sur iOS, sur Android (API Level 10) et la version C# de l'assembly correspondante peut être déployée avec l'application sur les différents environnements Microsoft. Toutefois, cette disponibilité multi-plateformes peut s'accompagner de subtiles différences sur les signatures des méthodes qu'il conviendra donc d'adresser.

D'un point de vue compilation, l'infrastructure mise en place à destination du développeur DevOps doit permettre de construire l'application sur de multiples environnements. Dans le cas d'une application iOS bâtie sur Windows le développeur doit disposer d'un Mac (ou de l'image virtuelle d'un Mac) connecté en réseau et fournissant le service de génération et de déploiement. De nouveaux services en ligne proposent aujourd'hui de faciliter cette démarche en offrant le processus de compilation iOS sans avoir besoin de monter l'environnement correspondant, qu'il soit virtuel ou physique.

L'adaptation du code source en fonction de la cible

Il est parfois nécessaire d'adapter le code source avant de générer un exécutable pour un système d'exploitation (par exemple Windows) à partir de code source ciblant par défaut un autre système d'exploitation (par exemple Linux). En effet, suivant l'implémentation d'origine, l'existence de compilateurs ciblant les mêmes langages sur ces différentes plates-formes ne suffit pas à elle seule pour garantir la portabilité du code source. Au-delà de la maîtrise conjointe de multiples systèmes d'exploitation, une opération de portage d'un environnement à l'autre suppose la mise en œuvre d'un certain nombre d'aptitudes plutôt orientées DevOps.

Prenons l'exemple de l'application open source Wavewatch III qui simule le mouvement des vagues en fonction du comportement des vents soufflant sur la surface de l'océan. Elle permet d'obtenir des résultats sur une période étendue ou de prévoir la houle en se basant sur les prévisions de vent fournies par le *Weather Forecast System* (WRF) du *National Oceanic and Atmospheric Administration* (NOAA). Elle repose sur un modèle relativement complexe, la croissance des vagues étant gouvernée par de multiples processus. Compte tenu de la puissance de calcul nécessaire pour obtenir des résultats significatifs, le code Fortran proposé par le NOAA est optimisé pour un traitement parallélisé. Il offre ainsi deux modèles de compilation et de déploiement : OpenMP (exécution par de multiples threads en mémoire partagée) et MPI (exécution en mémoire distribuée par de multiples processus).

Cette solution est aujourd'hui proposée par défaut pour Linux. WW3 est codé en Fortran-90 standard, sans dépendance vis-à-vis de l'environnement. Par contre, le code source proposé par la NOAA utilise son propre système de prétraitement pour choisir les modèles de compilation, les options de tests… Ce système de *pre-processing* est associé à des scripts Linux destinés à automatiser et *faciliter* installation et exécution… en environnement Linux. Ces scripts ne sont pas nativement supportés sous Windows, ce qui complexifie fortement le portage de cette application vers Windows. Le développeur DevOps est alors amené à considérer différentes approches pour répondre à cette problématique.

Une première solution consiste à créer un environnement de build hybride de cross-compilation utilisant un sous-système Unix hébergé dans Windows avec une compilation sous Windows. Le principe est alors de construire les exécutables pour Windows en réutilisant directement les scripts en shell Unix permettant de générer les fichiers makefile et en les exécutant dans un sous-système Unix de Windows. SUA (*Subsystem for Unix-based Application*) étant aujourd'hui en voie d'obsolescence, l'approche recommandée consiste à utiliser un autre sous-système, comme les projets open source Mingw64 ou Cygwin.

Une autre solution est de faire l'inverse, c'est-à-dire de créer un environnement hybride de *cross-compilation* s'exécutant sous Linux pour une production d'un exécutable Windows. Ce type de solution propose l'utilisation d'outils tels que Parallel Tools for Windows Binaries on Linux, s'exécutant directement sur un environnement HPC Linux afin de produire un exécutable Windows *via* une cross-compilation fondée également sur MinGW64.

Enfin, et c'est parfois la solution la plus *simple*, il est possible d'aménager la génération du code source et des makefiles sur Linux pour une compilation sous Windows. C'est d'ailleurs l'approche recommandée par Hendrik Tolman, l'auteur du code WW3. Il s'agit de déclarer dans les fichiers de configuration sous Linux les options de compilation, puis d'extraire le code source cible et les makefiles, grâce à un shell script Linux permettant de générer sur la machine Linux les fichiers sources dans un répertoire de travail en vue de leur compilation ultérieure sous un autre environnement (en l'occurrence Windows).

Ce cas un peu extrême met en évidence la multiplicité de compétences dont doit disposer un développeur DevOps. La difficulté consiste alors à pouvoir disposer de l'ensemble des librairies utilisées par le code ainsi généré, comme par exemple la librairie NetCDF (*Network Common Data Form*) très utilisée dans le monde scientifique pour manipuler des tableaux de données dans de multiples langages et sur de nombreuses plates-formes. Il faut alors pouvoir recompiler ces librairies en vue de leur utilisation sur le système cible ce qui, là encore, peut être source de complexité.

Pour faciliter cette démarche, le développeur peut s'appuyer sur une solution de développement cross-platform. Il existe différentes solutions open source qui ciblent ce type d'objectifs comme par exemple Scons, Autotools (GNU), et CMake. Ce dernier se distingue dans la façon dont il adresse la problématique. Son principe n'est pas de générer la librairie ou l'exécutable cible – cela reste le rôle des compilateurs installés sur la machine – mais de permettre de créer les fichiers de configuration en fonction de cette cible. Cela permet au développeur d'utiliser l'outil qu'il maîtrise tout en facilitant grandement sa tâche. La copie d'écran suivante illustre l'utilisation de CMake pour la compilation de la librairie netcdf.

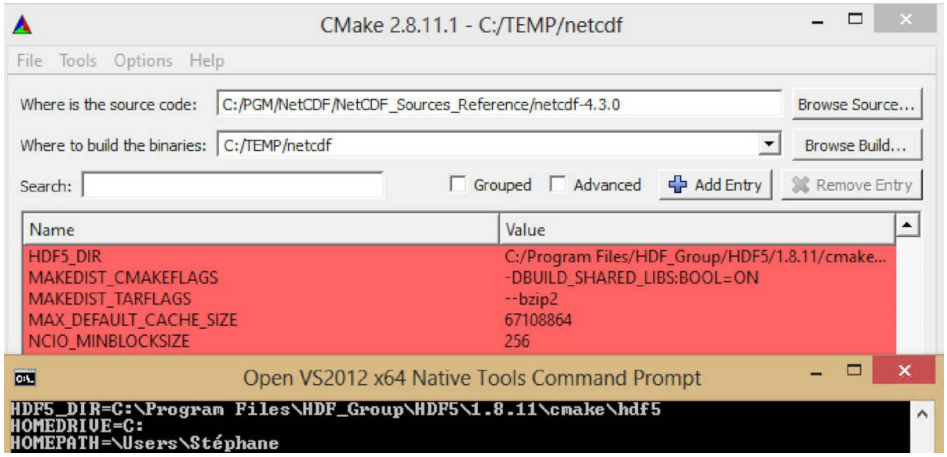


Fig. 3.17 *Création de fichiers de configuration multi-plateformes avec la solution open source CMake*

3.6.6. La nécessité d'intégrer les évolutions de code au plus tôt

Une application est rarement monolithique. Son implémentation suppose donc la mise au point de multiples composants ou modules dont les versions vont évoluer séparément. À l'échelle d'un développeur, cela suppose déjà un minimum d'organisation, mais si l'on considère le cas d'une application plus complexe, sur laquelle de multiples développeurs sont impliqués, la nécessité de valider fréquemment l'intégration du travail de chacun devient impérative.

En effet, au-delà d'un certain seuil de complexité, la validation par le développeur de l'intégration de l'ensemble des composants sur sa machine ne suffit plus. De plus, les vérifications à mettre en place doivent être totalement indépendantes du contexte d'exécution lié au poste du poste du développeur. Il faut pouvoir automatiser l'exécution de tests permettant de vérifier le bon fonctionnement de l'application sur un environnement dont on maîtrise la configuration. L'objectif est de détecter les erreurs d'intégration au plus tôt et de notifier les personnes concernées d'un éventuel souci lié à cette intégration. Le processus peut être effectué régulièrement (par exemple le soir, si le volume de code suppose une compilation de longue durée et l'exécution de séries de tests assez consommatrice en temps). Il peut être également déclenché à chaque mise à jour du référentiel de code source. On parle alors d'intégration continue.

3.6.7. L'intégration continue

L'intégration continue vise à réduire les efforts d'intégration en assurant automatiquement la génération, l'assemblage et les tests des composants de l'application. Voici la définition qu'en donnait Martin Fowler, en mai 2006 :

> « *Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible.* »

La culture avant les outils

Les principes de mise en œuvre d'un processus d'intégration continue sont connus depuis longtemps. Ils étaient mis en œuvre bien avant que l'on parle de démarche DevOps. La dimension DevOps se concrétise dans la capacité qu'elle offre d'inscrire l'intégration continue dans un processus plus étendu, celui du *continuous delivery*. Comme tout processus lié à la démarche DevOps, la réussite de la mise en œuvre d'une chaîne d'intégration continue est d'abord une question de culture. Avant de confier le soin à un outil logiciel, aussi bon soit-il, de déclencher une compilation et des tests, dès lors qu'une modification de code source a été détectée, il faut d'abord avoir validé la définition d'un build, fiable et reproductible. En effet, il ne servirait pas à grand-chose de déclencher en continu des séries ininterrompues de builds qui échoueraient. Intégrer en continu suppose donc d'avoir validé en amont le bon fonctionnement des tests et les différentes étapes de l'automatisation du déploiement, autant d'actions qui nécessitent souvent l'intervention de multiples acteurs. De plus, cela suppose un temps de génération et d'exécution des tests suffisamment réduit pour rendre possible ce déroulement en continu.

La mise en œuvre d'un processus d'intégration continue

Dans sa mise en œuvre, le processus d'intégration continue s'articule autour de plusieurs étapes. On peut considérer qu'il démarre depuis la phase d'implémentation durant laquelle le développeur produit et valide localement le bon fonctionnement de son code avant de l'archiver dans le référentiel de code source. Le contrôleur de code source notifie alors le système de build afin que soit lancée une nouvelle compilation.

Le serveur de build extrait alors la dernière version du code depuis le contrôleur de code source, génère les livrables et exécute les tests permettant d'offrir un premier niveau de validation du code produit. Plus que la mise en place d'une plateforme permettant l'enchaînement de ces opérations, c'est bien l'automatisation des différentes étapes liées à la compilation du code et au déclenchement des tests qui constitue l'une des difficultés majeures de ce type de processus.

Des outils pour faciliter la mise en œuvre du processus

Le choix de l'outil est un élément particulièrement structurant pour le processus d'intégration continue. Compte tenu de l'importance de cette étape du cycle de vie, la confiance dans l'outil doit être absolue.

Il s'agit avant tout d'un orchestrateur. Il peut donc être complètement externe au monde du développement logiciel, et l'on peut rencontrer des mises en œuvre d'outils plutôt orientés infrastructure, comme System Center Orchestration Manager ou Service Management Automation, mais ce n'est pas la norme, car dans la plupart des cas, la solution retenue est une solution dédiée à l'intégration continue.

Il peut s'agir d'une solution open source comme Jenkins. Au-delà de l'extensibilité dont nous avons déjà parlé, l'un des atouts de Jenkins est sa capacité à pouvoir orchestrer de très nombreux processus eux-mêmes liés à de multiples technologies. Pour mettre en œuvre une intégration continue, Jenkins propose un modèle dans lequel un master peut

piloter un cluster de machines esclaves sur lequel des agents virtuels baptisés *executors* peuvent exécuter les tâches qui leur sont assignées. C'est du nombre de ces agents que dépendra le nombre de jobs de compilation qui pourront être lancés simultanément lors de l'intégration. On retrouve des approches relativement similaires sur des solutions commerciales comme TeamCity, Team Foundation Server ou Visual Studio Team Services.

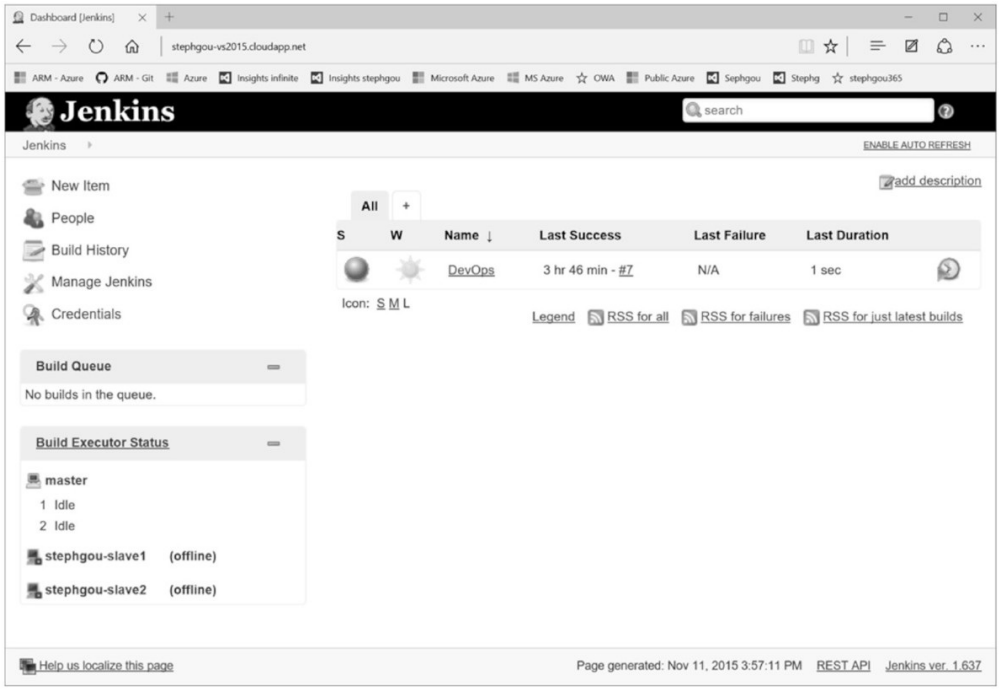


Fig. 3.18 *Cluster de serveurs master et slaves avec la solution open source Jenkins*

Les problématiques soulevées par l'intégration continue

Qu'elles soient commerciales ou open source, ces solutions doivent pouvoir être notifiées par le contrôleur de code source de l'occurrence d'une nouvelle mise à jour. Si le contrôleur de code source n'est pas nativement intégré au service de build, il faut au préalable avoir géré les contraintes liées à l'authentification sur ce système. Dans le cas de GitHub par exemple, on peut définir un jeton d'accès que l'on réutilise ensuite pour configurer la solution d'intégration retenue. Cela permet à cette dernière de se connecter sur le dépôt GitHub. Il faut ensuite demander explicitement l'activation d'un élément déclencheur depuis la solution d'intégration continue sur le système GitHub *via* un *web hook*.

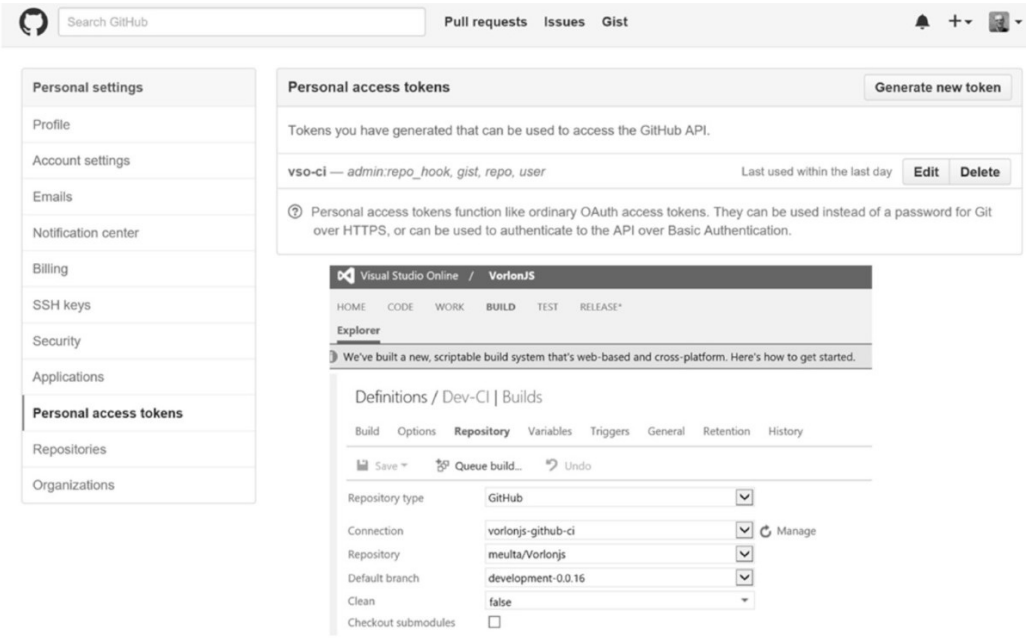


Fig. 3.19 *Définition d'un jeton d'accès à GitHub pour l'intégration continue*

Un problème classique d'un processus de build (auquel l'intégration continue par la fréquence des builds donne encore plus d'impact) est lié au fait que les processus de build partagent les ressources (fichiers, réseaux) qui y sont liées. Ce contexte peut se traduire par des échecs temporaires (le job de build s'exécute normalement dès lors que la ressource n'est plus verrouillée), ou par des échecs plus permanents dans le cas où un précédent job a modifié l'environnement de build le rendant inopérant. D'un point de vue DevOps, ce type de situation doit absolument être évité : la confiance dans la solution de build doit être absolue. Le build ne doit échouer que si le code est incorrect. Ainsi, le développeur saura avec certitude qu'il doit rapidement apporter un correctif à sa livraison et non suspecter une éventuelle défaillance du build.

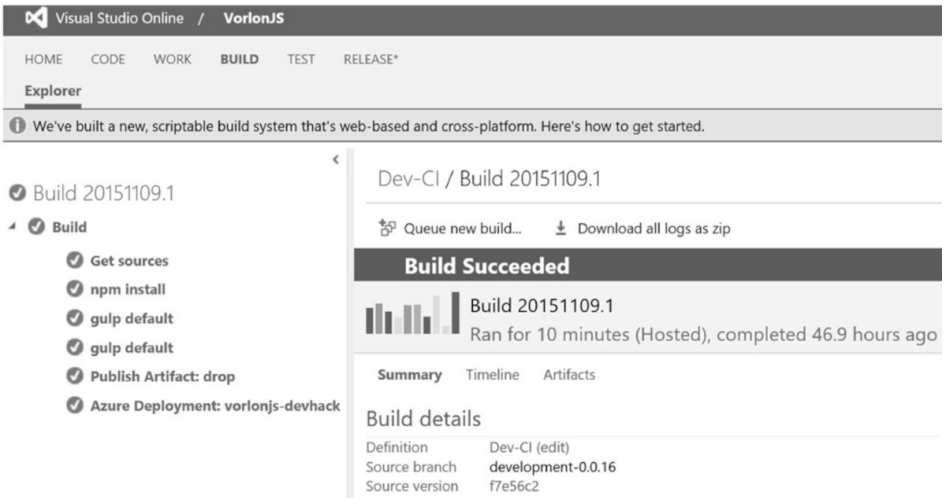


Fig. 3.20 *Un build ne devrait échouer que dans le cas où le code est défectueux…*

Par rapport à ce type d'objectif, le cloud peut apporter un premier niveau de réponse (notamment en proposant un environnement de build réinitialisé pour chaque nouvelle génération). Une autre approche peut également consister à s'appuyer sur des containers,

en faisant le lien depuis la solution de build avec un framework de packaging et de distribution comme Docker. Nous reviendrons sur ce point dans le dernier chapitre.

3.7. Le processus de release management

Une fois que la compilation des éléments constituants le logiciel sur le serveur de build s'est achevée et que les premiers tests de vérification de la qualité du code produit se sont exécutés avec succès, l'application est déployée sur de multiples environnements et validée *via* de nombreux tests (tests d'interfaces graphiques automatisés…) selon un processus baptisé *release management*. Ce processus est souvent complexe et exige la collaboration de plusieurs personnes voire de plusieurs équipes. Le point d'entrée de ce processus est la création de packages de déploiement.

3.7.1. La gestion des packages de déploiement

Il s'agit de pouvoir assembler les données correspondant aux livrables (binaires, fichiers de configuration, fichiers de déploiement, fichiers de test…) dans un format cohérent et exploitable, afin qu'elles puissent être automatiquement acheminées vers l'environnement de déploiement. Les éléments correspondants sont partagés dans un dépôt commun aux équipes de développement et de gestion opérationnelle. En ce qui concerne l'application, la question se pose de savoir quel format retenir pour le transfert entre les différentes cibles de déploiement.

Les risques liés à l'utilisation du code comme format pivot

Certains prônent l'utilisation du code archivé dans le système de contrôle de version, plutôt que le binaire comme l'élément pivot des multiples étapes du processus de déploiement. Dans cette approche, le code est compilé à plusieurs reprises dans différents contextes : pendant le processus d'archivage, lors des tests d'intégration, lors des tests fonctionnels et pour chaque environnement de déploiement cible. Or, chaque fois que le code est compilé, il est possible que le résultat soit sensiblement différent du précédent. La version du compilateur installé sur telle ou telle plate-forme peut varier, au même titre que les librairies dont dépend la solution. En outre, il est préférable d'éviter de rajouter le temps nécessaire à la recompilation, qui, pour certains logiciels, peut être assez significatif.

Le binaire comme format de référence

Il est préférable de déployer le même binaire (quand il s'agit d'un langage compilé) sur l'ensemble des plateformes. Une version des binaires ne devrait donc être produite qu'une seule fois, au début du cycle, lors de la phase d'intégration continue. Cette version pourra ainsi être réutilisée sur chaque environnement cible du processus de déploiement.

Les multiples versions de ces binaires devraient donc être conservées sur un système de fichiers (pas dans le contrôle de configuration source) afin de pouvoir être réutilisées à chaque étape du processus. Le serveur d'intégration continue assurera cette tâche en associant le numéro de build avec la version (comme nous l'avons vu précédemment, il est

possible également d'inclure automatiquement ce numéro dans les propriétés du binaire produit…). Il est par contre inutile d'archiver la totalité des binaires produits. Le système devrait théoriquement permettre si nécessaire, de générer à nouveau ces binaires pour une version de code donnée.

Le cas particulier des configurations

Si le binaire ou le code sont identiques quel que soit l'environnement cible de déploiement, il n'en va pas de même pour la configuration. Cette dernière reste susceptible de différer entre les environnements. La gestion de configuration doit donc pouvoir s'intégrer avec les différents outils jouant un rôle dans la chaîne de release management.

3.7.2. Le pipeline de déploiement

L'objectif d'un processus de release management est d'éliminer les versions inaptes à partir en production et de pouvoir remonter au plus tôt les raisons de cet échec. Une approche DevOps doit permettre à chaque instant d'être en mesure d'évaluer la viabilité en production de la version du livrable en cours. Pour ce faire, il faut être en mesure de définir et maintenir un modèle de gestion des versions d'application offrant la création de chemins d'accès de configuration et la configuration d'orchestrations de déploiement.

L'ensemble de ces étapes constitue le *pipeline de déploiement* (terminologie issue de l'ouvrage *Continuous Delivery* de Jez Humble et David Farley). Il s'agit de l'un des éléments clés du processus de *continuous delivery*. La mise en œuvre de ce pipeline fournit une visibilité sur le statut de l'application à chaque étape de son cheminement vers la mise en production. Chaque étape est associée à une série de tâches qui se dérouleront sur l'environnement cible. Cela suppose l'automatisation des tests et du déploiement sur les multiples environnements de validation. Ainsi les versions déployées en production ont nécessairement fait l'objet de tests permettant de s'assurer de leur bon fonctionnement et de leur adéquation avec l'usage ciblé, ce qui permet d'éviter des régressions, en particulier dans le contexte d'application de correctifs.

Bien que le processus de livraison logicielle puisse être entièrement automatisé, la mise en œuvre d'un pipeline de déploiement n'exclut pas une interaction humaine avec le système. Et bien entendu, elle suppose l'utilisation d'un outil permettant de construire ce pipeline dans lequel développeurs et responsables système vont pouvoir définir les actions et critères de déploiement, de vérification et de validation avec contrôles manuels ou automatiques. Ce pipeline devrait pouvoir utiliser des objets partagés au sein d'un référentiel commun. Ce référentiel doit offrir un contrôle de version, ainsi qu'un tableau de bord permettant de suivre l'état d'avancement du déploiement des différentes versions des applicatifs déployés : une vision historique du pipeline de déploiement.

Il ne suffit pas d'automatiser le workflow lui-même, mais aussi la création des environnements, leur mise en service, et la maintenance de l'infrastructure. L'automatisation du processus de déploiement le rend plus rapide, reproductible et fiable, ce qui permet de l'activer beaucoup plus fréquemment. Cela permet également de faire

des retours sur une version précédente beaucoup plus rapidement. Enfin, ce processus devrait être automatiquement déclenché par l'occurrence d'un nouveau build.

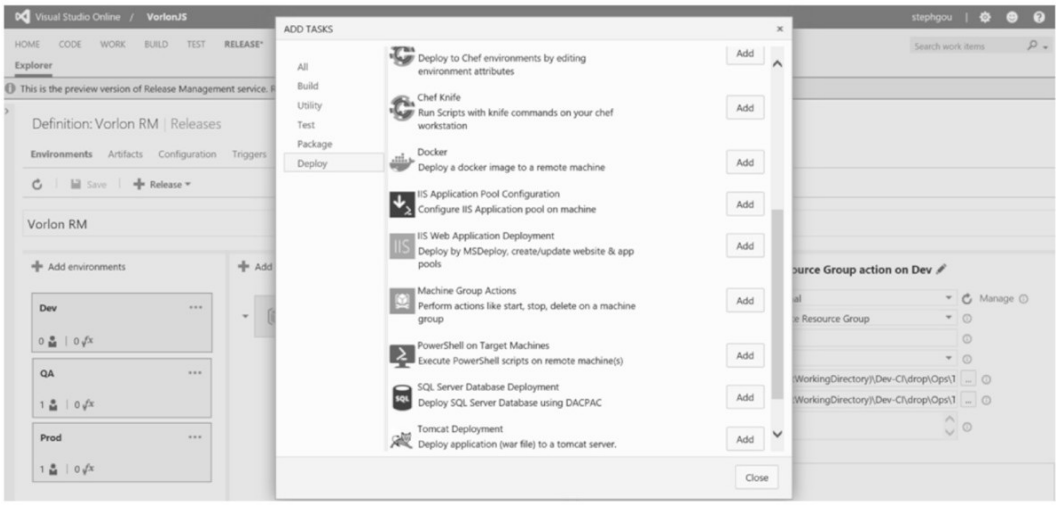


Fig. 3.21 *Le pipeline de déploiement*

3.7.3. Stratégie de branches et release management

Stratégie de *gestion des branches* et *release management* sont intimement liés. Un cas de figure simplifié est représenté sur le schéma de la figure 3.22.

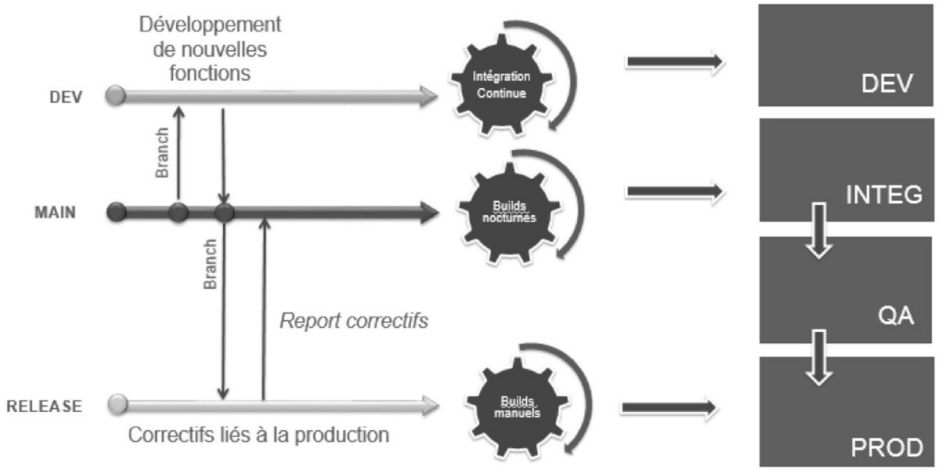


Fig. 3.22 *Stratégie de branches et release management*

Les nouvelles fonctions sont expérimentées sur une branche de développement qui fait l'objet d'une intégration continue. Si le résultat est satisfaisant, une fusion peut avoir lieu sur la branche principale qui dans cet exemple est liée à un build nocturne. Le livrable entre alors en release management et traverse les différentes étapes avec ou sans validation manuelle. En cas de dysfonctionnement constaté sur la production, le bug peut être corrigé sur une branche dédiée, testé directement en production et pouvant faire l'objet d'une fusion sur la branche principale si la correction est effective…

3.7.4. Le processus de continuous deployment

Sur le plan conceptuel, le processus de *continuous deployment* est très proche du processus de continuous delivery. Il se différencie simplement par l'automatisation de la dernière étape, celle de la mise en production.

Dans le cas du processus de *continous deployment*, la chaîne de production logicielle est automatisée de bout en bout. Le déclenchement de l'archivage d'un code source peut donc se traduire par la mise en production d'une nouvelle fonction sans la moindre intervention humaine… Ce qui paraissait totalement illusoire il y a quelques années est donc devenu aujourd'hui une réalité pour bien des fournisseurs d'offres cloud, qu'il s'agisse d'Amazon, de Google ou de Microsoft.

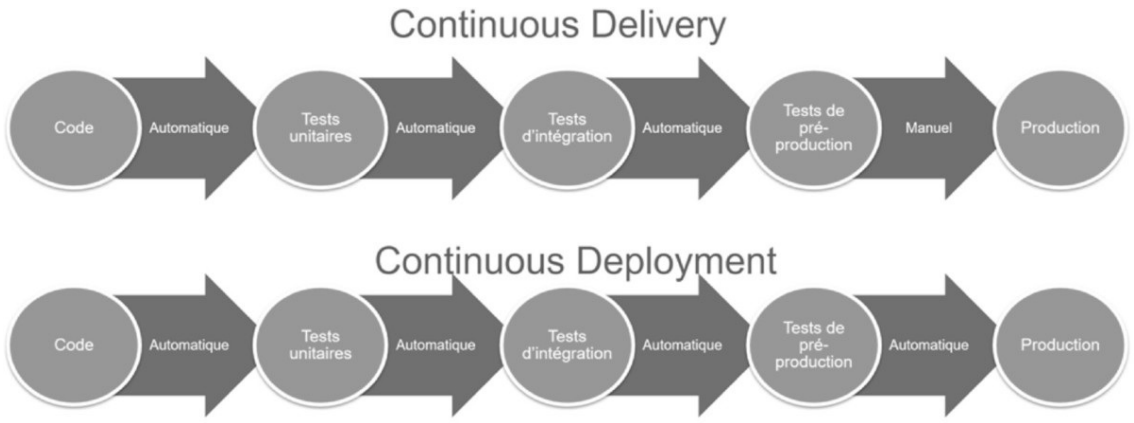


Fig. 3.23 *Continuous delivery et continuous deployement*

3.8. L'instrumentation et la supervision : la vue du développeur

La création de boucles de rétroaction entre les opérations et le développement est un élément essentiel de DevOps. Les développeurs DevOps utilisent les données issues de production pour prendre des décisions sur les améliorations et les modifications à apporter à leur code.

3.8.1. Le diagnostic en production

Tôt ou tard, le code mis en production rencontre un dysfonctionnement. Mais dans ce contexte, il devient beaucoup plus difficile de déterminer pourquoi une application est suspendue, pourquoi une fuite de mémoire s'est produite ou pourquoi l'application a crashé. En effet, lorsque l'application est utilisée en conditions réelles, il faut éviter toute intervention qui soit de nature à altérer la qualité du service. Parmi les multiples tactiques permettant de résoudre au plus tôt la problématique rencontrée, l'une d'elles offre une possibilité de réduire considérablement les coûts de prise en charge et de temps : le *debugging en production*. Le principe est de lancer un debugger sur le processus incriminé s'exécutant sur l'infrastructure de production.

Il diffère du debugging sur un environnement de développement par de multiples points. Ainsi, lorsqu'il cible un environnement de production, déterminer la cause du dysfonctionnement est souvent moins important que remettre le système dans un état

opérationnel. De plus, lorsqu'il est mis en œuvre dans ce contexte, le *debugging* en production doit respecter de strictes contraintes de temps, pour éviter de pénaliser l'utilisateur de l'application.

Enfin, certains dysfonctionnements ne se manifestent que lorsque le système est soumis à une forte sollicitation. Les causes d'une erreur non déterministe sont nécessairement plus difficiles à diagnostiquer. Et les conditions dans lesquelles elle se produit rendent l'analyse plus complexe. En effet, souvent les traitements serveur s'exécutent en parallèle sur de multiples threads et traitent simultanément de multiples requêtes, ce qui peut rendre particulièrement complexe le suivi de l'une d'entre elles. Et si la solution s'exécute sur une ferme de serveurs, la difficulté s'accroît avec le nombre de serveurs impactés. La seule solution réside alors dans l'analyse des journaux, ceux que proposent nativement les systèmes, et ceux qui pourront être renseignés par l'application en cas d'erreur.

3.8.2. La journalisation des évènements

Lorsque l'application rencontre un dysfonctionnement en production, le développeur DevOps fait équipe avec le responsable opérationnel pour parvenir à rapidement corriger ce dysfonctionnement. Il est donc crucial pour le développeur DevOps de comprendre les principes du système d'exploitation sur lequel va s'exécuter le code qu'il a produit, en particulier d'avoir une bonne compréhension des mécanismes de journalisation du système. Nous étudierons plus en détail, les mécanismes natifs de journalisation proposés par les différents systèmes dans le chapitre lié aux opérations.

En général, et quel que soit le système d'exploitation, ces mécanismes de journalisation intègrent un système d'horodatage qui permet d'avoir une idée très précise du délai d'exécution de chacune des opérations tracées et exposent des événements fournis nativement par le système d'exploitation (processus, threads, CPU, changements de contexte, défauts de page, mémoire, I/Os réseau, I/Os disque…). Ils doivent être peu coûteux en termes de performance, ne pas perturber le système observé si on les met en œuvre et doivent pouvoir être facilement désactivés : ils deviennent alors totalement transparents sur le fonctionnement de la solution. Enfin, ils permettent d'ajouter des données applicatives à celles fournies par le système d'exploitation. Et la responsabilité de cette instrumentation incombe alors au développeur.

3.8.3. La centralisation des journaux d'évènements

La centralisation des données d'instrumentation peut être assurée de multiples façons. Une première solution consiste à remonter directement ces données en faisant directement appel à des API qui peuvent être ou non associées à différents protocoles standards tels que SNMP ou WMI. Ces protocoles seront étudiés plus en détail dans le chapitre 4 lié à la gestion des opérations. Mais parfois le besoin va au-delà de la remontée d'un évènement de type exception. Il s'agit par exemple de remonter chaque seconde les caractéristiques (CPU, consommation mémoire, I/Os) sur des centaines de serveurs. La solution peut alors consister à mettre en place un mécanisme permettant de remonter périodiquement les logs pour les centraliser sur le serveur de suivi, soit sous leur forme native, soit en les injectant

dans une base de données. En général, les systèmes de supervision interrogent périodiquement les sources de logs pour consolider les données avec une fréquence de l'ordre de la minute. Certaines solutions de cloud proposent nativement ce type de mécanismes de regroupement périodique des informations de log.

On peut alors s'appuyer alors sur des solutions clés en mains (nous reviendrons sur ce point dans le chapitre 4 *DevOps vu par les équipes opérations*). Mais parfois, par exemple lorsqu'il est nécessaire de disposer de données avec une très faible latence, le développeur DevOps est appelé à participer à une implémentation spécifique d'un système de remontée d'informations de supervision. Ce type d'implémentation se fonde alors sur l'utilisation de technologies CEP (*Complex Event Processing*) ou de *realtime distributed processing*. Nous reviendrons ultérieurement sur ces mécanismes dans le chapitre 5 *DevOps vu par la qualité*.

3.8.4. Les outils de trace système

Au même titre que l'administrateur système, le développeur DevOps doit avoir connaissance des différents mécanismes de trace proposés par les systèmes d'exploitation sur lesquels vont s'exécuter les composantes de l'application qu'il développe. Nous reviendrons sur ce sujet dans le prochain chapitre.

3.8.5. L'instrumentation des applications

Le développeur DevOps doit s'assurer que son application est instrumentée avec les bons paramètres afin qu'elle puisse être supervisée efficacement par les solutions de monitoring utilisées par les équipes en charge de l'exploitation du système. Il peut s'agir de données de diagnostic recueillies lors d'un crash de l'application, qui incluent des informations sur les performances et la disponibilité du système en production. Le développeur peut ainsi analyser les données recueillies pour trouver la cause profonde des problèmes et corriger les bugs ou les comportements non souhaités.

D'un point de vue qualité, la portée de l'instrumentation doit cibler l'intégralité de l'expérience de l'utilisateur, avec une perspective qui soit alignée sur la perception qu'il peut en avoir, et qui ne se limite pas à une combinaison des compteurs de performances relevés sur chacune des composantes de l'infrastructure. Une performance de référence pourra ainsi être établie et toute dérive devra pouvoir être détectée. D'où la nécessité pour les développeurs de communiquer aux responsables des opérations l'ensemble des éléments requis pour une supervision effective…

Pour ce faire, ils peuvent s'appuyer sur les frameworks associés aux mécanismes natifs de journalisation et bénéficier ainsi d'une capacité à instrumenter l'application en étant très proches du système. Toutefois, comme nous le verrons dans le chapitre 4 lié aux opérations, leur utilisation n'est pas exempte de complexité et il n'est donc pas toujours très simple d'intégrer ces mécanismes dans un processus de développement *classique*, d'où l'intérêt de proposer des frameworks proposant un niveau d'abstraction plus élevé.

3.8.6. Les frameworks d'instrumentation

Il existe de multiples frameworks d'instrumentation d'une application, qui la plupart du temps sont liés à l'environnement d'exécution.

Ainsi, dans le cas de Java, le système de logging open source de référence est **Log4J**. Ce produit offre un moyen hiérarchique d'insérer l'enregistrement d'instructions au sein d'un programme Java permettant de faire usage de déclarations de débogage comme les classiques `System.out.println` ou `printf` et de tracer le détail de structures de données sur plusieurs formats de sortie et plusieurs niveaux d'informations de journalisation. L'utilisation de ce framework permet d'éviter d'avoir à gérer un grand nombre d'instructions de débogage en exposant leur contrôle et leur paramétrage dans des scripts de configuration livrés avec l'application. L'implémentation du framework Log4J a également été déclinée sur d'autres environnements d'exécution comme par exemple les versions Log4c pour les applications développées en C, Log4cpp pour celles codées en C++, et Log4Net pour celles qui utilisent le .NET Framework.

L'écosystème Microsoft se distingue en proposant de multiples frameworks permettant d'instrumenter une application en se fondant non seulement sur des frameworks ciblant l'environnement d'exécution, mais aussi sur des frameworks plus proches du système d'exploitation. Ainsi, au niveau applicatif, Microsoft propose, également en open source, des extensions comme le Logging Application Block de l'Enterprise Library permettant de gérer par configuration un usage des multiples API liées à System.Diagnostics dans l'environnement .NET. Tandis qu'au niveau du système d'exploitation, Microsoft propose EWT (*Event Windows Tracing*), un framework que nous reverrons dans le chapitre 4 sur les opérations. Microsoft fournit également la classe `System.Diagnostics.Tracing.EventSource` afin de disposer de l'ensemble des caractéristiques ETW (typage fort, versioning, extensibilité), sans avoir à en maîtriser toutes les subtilités ou en subir toutes les contraintes. Ainsi, le développeur n'a plus besoin de construire explicitement un manifeste EWT, il suffit avec une ligne de code de définir un nouvel événement.

En définitive il ne s'agit pas de débattre sur les vertus supposées de tel ou tel framework de journalisation. Dans ses activités de production logicielle, le développeur DevOps se doit d'utiliser un framework. En général, le choix de ce framework a fait l'objet d'une validation par l'ensemble des acteurs concernés. La responsabilité du développeur est donc de mettre en place un système robuste de gestion des exceptions qui fasse appel à ce framework en respectant les patterns définis en phase de conception.

Enfin, comme nous l'avons vu, la portée de ces frameworks peut aller au-delà de l'environnement d'exécution (machine virtuelle Java, .NET framework…) et des dépendances avec le système d'exploitation, en ciblant une plate-forme au sens le plus large du terme. Cette notion de plate-forme prend toute son importance lorsque la solution s'exécute sur une ferme de serveurs, car il s'agit non seulement d'intercepter et tracer les erreurs, mais aussi d'être en situation de pouvoir consolider le stockage lié à leur journalisation et centraliser leur restitution, indépendamment du nombre de serveurs sur lesquels s'exécute la solution.

En résumé

DevOps pour les développeurs c’est avant tout un changement de culture. Une extension de leur périmètre de connaissance sur tout un ensemble de sujets qu’ils considéraient jusqu’à présent comme connexes à leurs principales activités.

Le développeur DevOps doit avoir soif d’apprendre, envie d’expérimenter de nouveaux langages, de nouveaux frameworks, de partager ses nouveaux savoirs, de participer à l’outillage de la chaîne de production, de prendre position sur les choix d’implémentation. Son rôle s’inscrit dans un processus de continuous delivery dans lequel l’intégration continue est elle-même complétée par d’autres mécanismes issus de l’adoption de pratiques éprouvées.

Parmi celles-ci figurent en bonne place l’automatisation du déploiement du logiciel construit sur la plateforme cible (test, intégration, pré-production voire production dans le cas du continuous deployment) à chaque nouvelle génération de livrable ainsi que l’ensemble des moyens permettant de mettre en place le suivi en continu du comportement de l’application.

Autant de sujets sur lesquels la collaboration entre développeurs et responsables de la gestion opérationnelle est déterminante.

Chapitre 4

DevOps vu par les équipes opérations

La gestion opérationnelle des services d'infrastructure est un élément clé du succès d'une application.

Il intéressant de constater que les pionniers du DevOps avaient un profil plus orienté administration système et réseau que développeur. Les principes de l'agile concernaient tous les acteurs du cycle de développement logiciel (développeurs, analystes métier, testeurs, DBA, responsables qualité…) sauf les équipes opérations. À l'origine, il s'agissait donc d'abord d'appliquer cette vision au domaine d'expertise des responsables opérationnels et de mettre en avant les vertus d'une infrastructure agile.

Mais, notamment, sous l'influence de Patrick Debois, la démarche s'est enrichie d'une dimension nouvelle ciblant à un alignement entre les métiers Dev et les métiers Ops, par l'application des principes Lean. En parallèle, la virtualisation et le cloud ont accéléré le développement de solutions techniques ciblant l'application des bonnes pratiques associées à la démarche DevOps.

La vision du DevOps pour les équipes des opérations s'inscrit donc sur plusieurs axes : une évolution de l'organisation et du métier des opérations, les concepts liés à la mise en œuvre d'une infrastructure agile, le rôle du cloud dans cette évolution, et les multiples technologies et produits qui la rendent possible.

4.1. L'évolution du rôle et de l'organisation des opérations

Comme pour le développeur, la mise en œuvre d'une démarche DevOps par les opérations suppose l'acquisition de nouvelles compétences et s'inscrit dans de nouveaux processus organisationnels plus collaboratifs associés à de nouvelles méthodes de travail et de nouveaux outils.

DevOps pour les opérations : la culture avant la technique…

De même que pour le développeur, la transformation DevOps des équipes de gestion opérationnelle suppose un fort changement sur le plan culturel. Ces équipes se doivent d'accorder un niveau plus élevé de confiance aux autres acteurs, en leur déléguant l'accès aux ressources du système d'information. Elles doivent également s'inscrire dans un cycle d'apprentissage et d'amélioration en continu, pour acquérir de nouvelles compétences jusque-là réservées au monde des développeurs et partager leurs outils et leur processus. Ces équipes auront également à prouver leur capacité à prendre des risques avec une vision positive de l'échec, et à s'impliquer sur la réduction des délais, l'optimisation des coûts par l'automatisation et la mise en place de systèmes permettant de remonter de l'information en continu.

Dans la suite de ce chapitre, notre objectif est de montrer le rôle de ces équipes dans la mise en place du processus de continuous delivery en collaboration avec les développeurs. De même que dans le précédent chapitre, notre

discours aura une forte connotation technique. Mais là encore, il ne faudra pas perdre de vue, que pour ces équipes, DevOps signifie d'abord un changement de culture…

4.1.1. L'évolution de l'organisation des opérations

Une organisation bâtie en silos et qui isole chacun dans un périmètre bien restreint de responsabilité inhibe toute initiative destinée à rendre l'infrastructure plus agile. D'un point de vue organisationnel, une des évolutions majeures de DevOps est la suppression des verrous établis jusqu'alors pour éviter un risque d'erreur lié à un manque de rigueur présumé chez le développeur. Certes, en interdisant au développeur telle ou telle action, on ajoute des niveaux de protection interne au système, mais le résultat est aux antipodes des principes du DevOps : le développeur n'est pas tenté de comprendre ce qui se passe au-delà de son périmètre d'action puisque les opérations lui interdisent de le faire…

Au-delà des contraintes qu'ils occasionnent (délais, frustrations…), ces obstacles sont manifestement le reflet d'un manque de confiance des opérations. Ils doivent donc être progressivement éliminés en accompagnant leur disparition par une sensibilisation sur l'extension des responsabilités de chacun.

4.1.2. La transformation du rôle des opérations

Les multiples phases d'un cycle de production logicielle classique sont fréquemment cloisonnées entre les différents rôles. D'une certaine façon, les équipes de gestion opérationnelles assurent le lien entre les multiples acteurs du système d'information, par la mise à disposition des environnements requis par chacun des acteurs et sont donc au cœur de ce processus de collaboration.

Elles sont donc très logiquement devenues responsables de l'automatisation des processus permettant de déployer l'application sur les différentes plateformes. L'objectif est de gagner en agilité, mais aussi de faire en sorte que les équipes opérations n'aient plus à intervenir sur les processus récurrents, ce qui implique la mise en place d'un système de délégation des autorisations vers une multiplicité de rôles au sein de l'organisation pour déclencher des déploiements à la demande.

L'ensemble de ces éléments (automatisation, définition de niveaux de délégation, voire de quotas de ressources) constitue un modèle des services proposés par les équipes de gestion opérationnelle. L'aboutissement de ce modèle est fonction de son niveau d'industrialisation. Les équipes de gestion des opérations ont pour mission de proposer l'équivalent d'un catalogue de services répertoriant l'ensemble des actions et des possibilités permettant d'homogénéiser les besoins des utilisateurs voire de les limiter. Dans certains cas, ce catalogue pourra être matérialisé par l'ouverture de souscriptions dédiées sur le contrat d'un fournisseur de services cloud.

4.1.3. La transformation du métier des opérations

Le rôle de l'administrateur système DevOps est donc de mettre à disposition de l'ensemble des acteurs du système un certain nombre de capacités avec un niveau

d'automatisation, et un accès en libre-service qui permette à chacun de travailler avec plus d'autonomie sans pour autant remettre en cause les principes de contrôle et de sécurité des services d'infrastructures concernés.

Avec l'avènement des technologies d'automatisation et de virtualisation, l'administrateur système est, dans une certaine mesure, lui-même devenu un développeur capable d'automatiser telle ou telle séquence d'opérations afin qu'elle s'exécute avec le minimum d'interventions manuelles.

Les systèmes d'exploitation proposent des langages de programmation système permettant d'automatiser un certain nombre de tâches. Il s'agit en général de langage de scripting, comme bash pour Linux ou Windows PowerShell. À ces langages nativement proposés sur ces systèmes viennent s'ajouter des solutions de configuration système souvent issues du monde open source, comme nous le verrons par la suite.

L'administrateur système DevOps se doit donc d'acquérir une réelle expertise sur les multiples solutions de scripting qui lui permettent de déployer les infrastructures dont il a la charge. Au même titre que les développeurs, il a donc besoin d'un environnement qui leur permette de développer, gérer en version, tester et exécuter son code d'infrastructure afin de valider la création d'environnements dotés de l'ensemble des prérequis de configuration et d'instrumentation.

4.2. La mise en œuvre d'une infrastructure agile

4.2.1. Un *Manifeste agile* de l'infrastructure ?

Avant DevOps, l'infrastructure d'un système d'information apparaissait alors comme une composante complexe, aux multiples ramifications et dépendances, associées à des processus souvent long et coûteux, bref, un paquebot plutôt difficile à manœuvrer. Les pionniers du mouvement DevOps ont donc souhaité faire évoluer ce modèle en s'inspirant des concepts agiles. Toutefois, plutôt que de se lancer dans une périlleuse transposition à l'identique des principes du *Manifeste*, ils ont préféré en retenir certains et les adapter au domaine de la gestion des opérations.

Plus concrètement, il s'agissait avant tout de repenser le rôle des équipes opérationnelles au sein du système d'information et de mettre en place le socle permettant de faire évoluer une infrastructure, de façon progressive et itérative, avec des capacités plus élevées, tout en gardant le contrôle de l'ensemble des ressources et processus associés.

4.2.2. Complémentarité avec ITIL

ITIL (*Information Technology Infrastructure Library*) est un standard qui a été créé à l'initiative du CCTA (*Central Computer and Telecommunications Agency*), organisme gouvernemental anglais, afin de définir un ensemble de règles pour gérer l'infrastructure et les opérations. ITIL n'est pas le seul framework ITSM (*Information Technology Service Management*) : il y en a d'autres comme le standard ISO/IEC 20000 qui permet de certifier les services informatiques des organisations ou MOF (*Microsoft Operation*

Framework). Mais il s'agit d'une référence pour la communauté de gestion de services de l'IT. Dans sa troisième version, ITIL est d'ailleurs particulièrement adapté au développement de processus conformes à la norme ISO/IEC 20000.

De prime abord, ITIL peut sembler en contradiction avec les principes de l'agilité et par extension avec la démarche DevOps. En réalité, loin d'être antagonistes, ITIL et DevOps se complètent sur bien des points.

ITIL a pour objectif de rationaliser la gestion des services informatiques autour de grandes fonctions et de processus standardisés. Il intègre de multiples « bonnes pratiques » plus ou moins adaptées ou pertinentes, selon le contexte et l'évolution générale des technologies.

La démarche DevOps s'accompagne aussi de « bonnes pratiques » mais comme nous l'avons vu, ses objectifs sont différents. Pour autant, elle ne remet pas en cause la gestion en services préconisée par ITIL, ni même les processus qui peuvent les composer. Au contraire, DevOps s'appuie sur plusieurs de ces services : par exemple, certains processus d'ITIL sont directement au cœur de la boucle de feedback DevOps. Le *Service Desk* est l'oreille du client dans une démarche DevOps et la *gestion des incidents et des problèmes* font partie intégrante de la philosophie de collaboration de la démarche. Enfin le modèle *hub-and-spoke* du cycle de services ITIL n'est pas si éloigné du modèle d'organisation DevOps en *hubs de service* que nous étudierons plus en détail dans le chapitre 6 *DevOps vu par le management*.

Néanmoins, les pratiques DevOps sont susceptibles de faire évoluer ou remplacer les bonnes pratiques incluses dans la boîte à outils ITIL, sans remettre en cause la logique ou l'esprit initial. Ainsi, DevOps peut intégrer de nouveaux rôles qui n'existent pas dans ITIL ou faire faire évoluer les rôles et les responsabilités de chacun, en décloisonnant tel ou tel silo organisationnel. Avec DevOps, l'importance et la visibilité de certains processus, comme la supervision, la gestion d'incidents ou la gestion du changement diffère également. D'autres processus doivent évoluer, comme par exemple, la mise en production avec l'augmentation de la fréquence de déploiement.

Cet impact de DevOps sur l'implémentation d'ITIL ne les rend pas incompatibles, au contraire, l'amélioration continue (*Continual Service Improvement)* est inscrite au cœur des principes ITIL (un autre point commun avec DevOps d'ailleurs). Il suffit pour s'en convaincre de se pencher sur l'évolution d'ITIL entre la version 2 et la version 3. Lors de cette évolution, certains processus ont été complètement refondus et parfois simplifiés. De nouveaux processus que nous qualifierons de plus agiles ont été intégrés, comme le *Service Catalog* par exemple. Des fonctions communes à DevOps, telle que la *gestion des applications*, ont été ajoutées, en sus du *Service Desk*.

Enfin, la logique du *do fast, check fast and fix fast* propre à la philosophie DevOps influe sur la façon de mettre en œuvre ITIL, en focalisant l'attention sur les résultats des processus plutôt que sur leurs pré-requis. Réussir à implémenter une démarche DevOps en complément d'une approche ITIL permet d'atteindre un vrai objectif de fiabilité en disposant d'*opérations* agiles et robustes et garantie la réussite d'une collaboration avec les développeurs comme avec les métiers.

4.2.3. Infrastructure as code

Le poste du développeur ne constitue que la première brique de l'infrastructure hébergeant le cycle de développement d'une application qui suppose la mise en œuvre de multiples plates-formes : plate-forme de test, d'intégration, de pré-production, de production. Chacun de ces environnements doit refléter l'environnement de production. À chaque instant, il doit pouvoir être rapidement cloné (par exemple, pour vérifier le comportement d'un correctif à chaud).

Cloner un environnement implique d'être en mesure de le provisionner (c'est-à-dire de mettre à disposition les ressources matérielles correspondantes), et de le configurer avec le déploiement de l'ensemble des caractéristiques systèmes et applicatives requises. Ces opérations peuvent être automatisées en utilisant différentes techniques de scripting que nous verrons dans la suite de ce chapitre. Il devient donc aujourd'hui possible de développer le code qui construit l'infrastructure et de mettre en application de nombreuses bonnes pratiques et outils issus de l'expérience du développement (notamment le versioning dans un contrôle de code source). Il devient également possible de procéder à un retour arrière à chaque incrément de l'évolution d'une infrastructure.

L'infrastructure *as code* devient alors un élément essentiel pour permettre aux équipes de gestion opérationnelle de proposer des services d'infrastructure adaptés à la vitesse du monde d'aujourd'hui.

4.2.4. Orchestration versus automatisation des tâches

L'automatisation désigne en général l'implémentation d'une tâche spécifique dans un langage (qui le plus souvent est un langage de script). Par exemple, le déploiement d'une machine virtuelle sur un hyperviseur est une tâche de type automatisation.

L'orchestration cible un processus complet. Il s'agit surtout d'une question d'échelle. Reprenons l'exemple de la machine virtuelle. Ce type de tâche est rarement isolé : on peut donc légitimement s'attendre à ce que la demande de création de la machine virtuelle soit associée à d'autres tâches comme la configuration de la ou les interfaces réseau de la machine, l'attribution d'adresses IP, le nom DNS, les disques attachés, la configuration des accès SSH ou RDP pour une connexion distante…

Le schéma de la figure 4.1 représente une orchestration elle-même composée de multiples tâches de provisioning et de configuration.

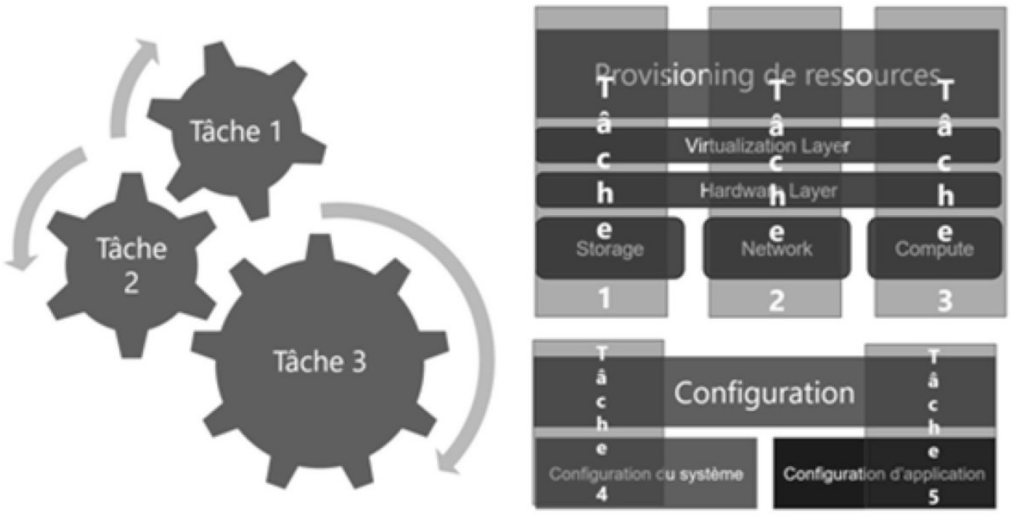


Fig. 4.1 *Orchestrations et tâches*

4.3. Le cloud : un accélérateur pour les opérations DevOps

Le cloud par la mise en œuvre de technologies liées à la virtualisation (machine, réseau, stockage) et à ses promesses en termes de scalabilité représente sans conteste un formidable accélérateur dans l'évolution technologique qui accompagne la transformation DevOps des équipes de gestion opérationnelle.

4.3.1. L'adaptation des caractéristiques du cloud à une infrastructure agile

D'après le NIST (*National Institute of Standards and Technology*), les attributs qui définissent le cloud computing sont la mutualisation des ressources, l'accès aux ressources en mode libre-service, l'accès à un réseau ubiquitaire, l'élasticité des ressources, et la facturation à l'usage… Les quatre premières de ces caractéristiques sont particulièrement adaptées à la mise en œuvre d'une infrastructure plus agile. À ce titre, le cloud qu'il soit public ou privé, peut grandement faciliter la mise en œuvre de DevOps sur le plan technique.

4.3.2. Les modèles de services

Durant la dernière décennie, une grande variété d'environnements pour l'exécution des applications a été mise à disposition des entreprises. Ces environnements peuvent être proposés selon plusieurs modèles de services cloud : IaaS (*Infrastructure-as-a-service*), PaaS (*Plateform-as-a-service*) et SaaS (*Software-as-a-service*). Ces différents types de cloud délivrent de multiples niveaux d'automatisation, et à ce titre ont un impact différent sur le rôle des opérations. Le schéma de la figure 4.2 illustre les différences liées aux options de service en précisant les services automatisés (dits *managés*) et facturés par un fournisseur tiers, et en signalant les services gérés par les équipes de gestion opérationnelle.

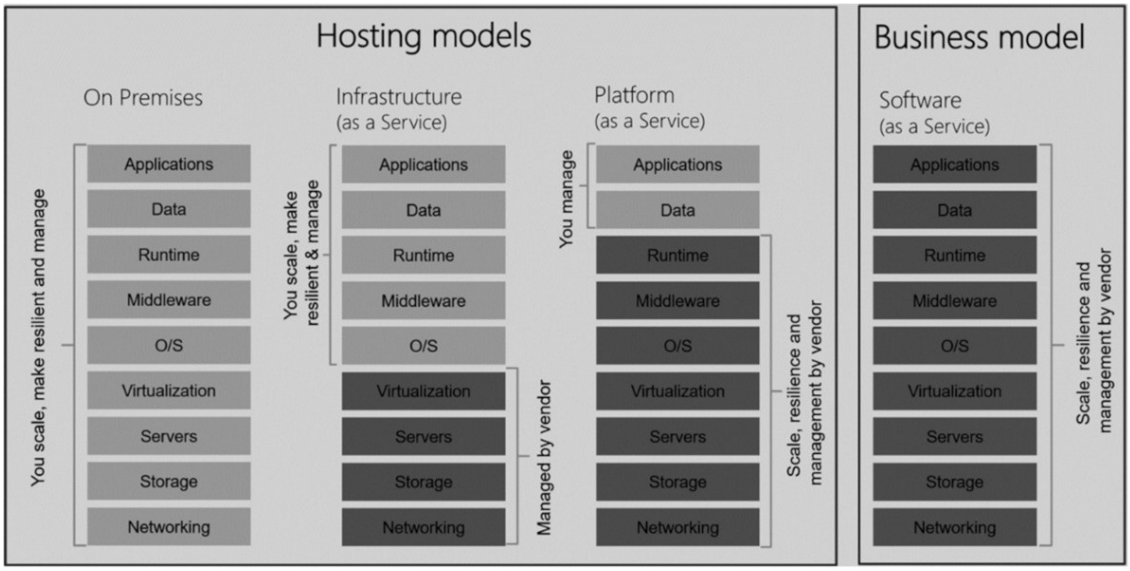


Fig. 4.2 *Les différents modèles de services*

Services IaaS et gestion opérationnelle

Les services IaaS permettent de se libérer de certaines des préoccupations associées à la fourniture physique ou virtuelle en permettant de provisionner à la demande des machines virtuelles dont le format est standardisé. C'est encore le modèle de service le plus fréquemment utilisé de nos jours et celui avec lequel les équipes de gestion opérationnelle ont sans doute le plus d'affinité. La valeur ajoutée de l'administrateur système DevOps est alors de personnaliser les machines virtuelles et de créer des environnements dédiés, ou de construire de nouveaux processus automatisés sur la base de ces services IaaS. En soi, il constitue déjà une rupture par rapport à un modèle de gestion opérationnelle plus ancien (avec des responsabilités cloisonnées entre systèmes, réseaux, stockage…).

Services PaaS et gestion opérationnelle

Le Gartner donne la définition suivante du PaaS : « *A platform as a service (PaaS) offering, usually depicted in all-cloud diagrams between the SaaS layer above it and the IaaS layer below, is a broad collection of application infrastructure (middleware) services (including application platform, integration, business process management and database services* ».

Dans un autre document (*PaaS Road Map: A Continent Emerging*), le Gartner présente la multiplicité des offres PaaS. Leur dénominateur commun est d'offrir un niveau d'abstraction à de multiples concepts d'infrastructure : modèles d'environnement prédéfinis par plate-forme, approvisionnement sur demande, en libre-service, et automatisé de ces modèles, surveillance des ressources utilisées, gestion élastique des ressources en fonction de leur utilisation, garantie de continuité de service pour les applicatifs hébergés, même en cas de panne matérielle, de mise à jour du système d'exploitation ou de mise à jour de l'applicatif lui-même.

L'objectif du PaaS est de faciliter le développement, le déploiement et l'exécution d'applications et services en focalisant l'investissement sur les données et le code plutôt

que la mise en place de l'infrastructure qui les héberge. Pour autant, même si ce modèle décharge l'administrateur système DevOps d'une partie de ses responsabilités, il reste à ce dernier un certain nombre de tâches à accomplir pour en tirer le meilleur parti. Car même si ces services proposent des configurations prédéfinies et des mécanismes intégrés d'autoscaling, il est utile d'automatiser leur création, de tester leur disponibilité, de les intégrer dans un processus les liant à d'autres services pour une gestion commune de leur cycle de vie, de les superviser…

Services SaaS et gestion opérationnelle

Le modèle de cloud le plus visible pour le grand public est le SaaS, modèle dans lequel un vendeur héberge son propre logiciel et le propose à ses clients comme un service accessible *via* Internet. Salesforce.com, à défaut d'en être l'inventeur a été sans doute été le premier à en avoir réellement démontré la viabilité financière. Même si ce modèle permet de s'affranchir de tout un ensemble de tâches liées à la mise à disposition d'une solution logicielle, il implique souvent les équipes de gestion opérationnelle dans sa mise en œuvre. En effet, ces dernières seront nécessairement amenées à construire de nouveaux processus fondés sur ces services SaaS et à les intégrer aux services existants en prenant en compte différentes problématiques et notamment la gestion d'identité. L'influence d'une démarche DevOps sur ce type d'intégration se manifestera par le niveau d'automatisation des processus internes et par l'industrialisation du modèle de service associé.

4.3.3. Les modèles de déploiement

Le cloud propose trois principaux modèles de déploiement : privé, public et hybride. Chaque modèle peut avoir un impact sur la mise à disposition des services avec ou sans intervention des équipes de gestion opérationnelle.

Le cloud privé

L'objectif d'un cloud privé est d'utiliser et de déployer des couches de virtualisation, de provisionnement, d'automatisation et de catalogues de services pour une utilisation interne à l'entreprise. Il peut être déployé et exploité directement par l'entreprise ou par un tiers. Dans les deux cas, il offre un espace très propice à la mise en œuvre d'une infrastructure as code et constitue donc un terrain de jeu idéal pour l'administrateur système DevOps. Au final, il s'agit fréquemment d'en environnement de type IaaS, même s'il existe des exceptions (par exemple, la solution de cloud privé Azure Stack permet d'offrir l'accès à des ressources de type PaaS).

Le cloud public

Un cloud public est un service déployé et géré par des fournisseurs comme Amazon, Google ou Microsoft dans leurs datacenters (ou *via* des partenaires, comme c'est le cas pour Microsoft en Chine avec 21Vianet). L'infrastructure cloud y est donc mutualisée pour être utilisée par les clients. L'exploitation de la composante *non managée*, dont le niveau varie selon le type de service (PaaS ou IaaS), est de la responsabilité des clients du

fournisseur cloud, qui souvent sont les équipes de gestion opérationnelle. Là encore, l'environnement est particulièrement adapté pour les scénarios de type *infrastructure as code*. Les processus organisationnels mis en place pour accéder aux ressources cloud par l'ensemble des acteurs du système d'information sont un indicateur significatif du niveau de maturité DevOps des équipes de gestion opérationnelles et plus généralement de l'entreprise.

Le cloud hybride

Le cloud hybride se caractérise par la mise en œuvre d'infrastructures distinctes cloud et à demeure (ou résultant de la composition de multiples cloud). Celles-ci sont liées par des passerelles réseau permettant le bon fonctionnement des applications et l'échange de données. C'est un cas de figure assez fréquent qui suppose une implication forte des équipes de gestion opérationnelle dans l'établissement des connexions entre les différents environnements. Il offre la possibilité de mettre en place des scénarios d'infrastructure agile comme par exemple celui du débordement de ressources sur un cloud public (ou *bursting*) lorsqu'une entreprise a besoin de ressources supplémentaires (serveurs, stockages, etc.) pour absorber un pic d'utilisation que ne peut supporter son infrastructure à demeure.

4.3.4. Évolution vers le cloud

La bascule de l'existant vers le cloud doit être mûrement réfléchie. Toutes les applications ne se prêtent pas nécessairement à une stratégie d'évolution vers le cloud. Pour diverses raisons, certains types de service ont vocation à rester à demeure (par exemple les contrôles d'authentification d'accès au réseau, ou la conservation de catégories de données non éligibles à la sortie d'un périmètre géographique précis).

Quoi qu'il en soit, de plus en plus de solutions s'exportent aujourd'hui vers les nuages, partiellement ou dans leur intégralité. De multiples raisons expliquent cette évolution : les capacités en termes de scalabilité ou d'automatisation qu'offre le cloud au regard des limitations de systèmes d'information plus *classiques*, la mise à disposition de nouveaux types de service apportant *clé en main* de nouvelles facilités dans de nombreux domaines (y compris ceux liés à la gestion d'identité qui peuvent aujourd'hui tirer parti de l'utilisation de services de IdaaS *Identity as a service*).

Dans une perspective DevOps, cette évolution vers le cloud se traduit par une transformation significative des équipes de gestion opérationnelle qui se voient ainsi dotées d'un périmètre de responsabilité plus étendu. Concrètement, cette mission suppose l'acquisition de nouvelles compétences portant sur un univers extrêmement riche et requiert la définition des nouveaux processus à destination de l'ensemble des acteurs du système d'information. L'équipe de gestion opérationnelle doit alors assurer le provisioning des plates-formes cloud, la configuration des systèmes, le déploiement des composants applicatifs, la supervision et la maintenance de l'ensemble de l'infrastructure.

4.4. Le provisioning d'infrastructure

Que l'infrastructure d'une application soit déployée à demeure ou dans le cloud, la première étape de sa mise à disposition par les opérations est le provisioning des ressources associées (machines, réseau, stockage). L'administrateur système DevOps aura à cœur d'automatiser les actions correspondantes en offrant la possibilité de déléguer cette capacité à d'autres acteurs du système d'information, en fonction du contexte, tout en la contrôlant. Cette démarche requiert l'utilisation de multiples outils et leur intégration en fonction des environnements dans lesquels ils seront employés.

4.4.1. Les environnements

L'infrastructure physique

Dans bien des situations, l'administrateur système DevOps est encore responsable des choix d'infrastructure matérielle liée au provisioning des environnements sur lesquels vont être déployées les applications. Il lui faut alors décider du constructeur et de la solution entièrement packagée (stockage, serveurs, réseaux) ou suffisamment évolutive pour permettre une mise en œuvre rapide et homogène, en prenant en compte différents critères tels que le prix, les performances, la densité électrique, les flux de climatisation, la duplication des éléments réseaux, la mutualisation de la connectique…

Il doit avoir des compétences *étendues* sur la configuration matérielle, maîtriser les multiples technologies liées au stockage (SAS, SATA, ISCSI, Fiber Channel…), avoir connaissance des limitations de compatibilité des composants matériels (firmware, drivers…). Il doit avoir de bonnes compétences en réseaux, être capable de définir le nombre d'interfaces physiques des différents réseaux d'un cluster (teaming, multi-VLANS…).

Virtualisation

Suivant la taille de l'entreprise et son historique en termes d'adoption de système de virtualisation, l'administrateur système DevOps doit maîtriser l'utilisation de multiples hyperviseurs. Les hyperviseurs de type 1 (Citrix XenServer, VMWare ESX, ou Microsoft Hyper-V) ou leur équivalent chez les fournisseurs de service cloud offrent des fonctions relativement comparables. Dans la plupart de ces solutions, l'hyperviseur permet l'exécution de la machine virtuelle et sa reconfiguration dynamique en termes de CPU, RAM, ou capacité de stockage.

Avec la virtualisation, le réseau physique peut être partagé entre plusieurs réseaux virtuels isolés les uns des autres avec l'illusion qu'ils sont chacun seul sur ce réseau physique. Les serveurs et les ressources de ces réseaux virtuels (adresses IP, routes…) fonctionnent comme s'ils étaient directement reliés à un réseau physique. Ce type de solution permet de déplacer des VM dans le réseau physique sans avoir à reconfigurer l'adresse IP pour la virtualisation des réseaux ou des VLAN. La virtualisation des adresses IP est réalisée *via* des mécanismes comme (IP Rewrite) ou des protocoles spécifiques à l'hyperviseur (NV-GRE pour Hyper-V, VXLAN pour VMware).

Virtualisation par container

À la virtualisation des machines, du réseau et du stockage s'ajoute un autre sujet auquel les équipes des opérations doivent aujourd'hui s'intéresser : la virtualisation par container. Cet autre mode de virtualisation, présente des similitudes avec les précédents, notamment pour sa capacité à donner accès à un système d'exploitation à travers le réseau.

La plupart des systèmes d'exploitation offrent des mécanismes de virtualisation : sur UNnix, on parlera des *zones Solaris*, des *jails FreeBSD*, ou plus prosaïquement de *chroot*. Il s'agit d'une opération permettant de modifier le répertoire racine pour un processus système et les sous-processus, ce qui l'empêche d'avoir une visibilité sur d'autres parties du système de fichiers. De même, sur Linux, LXC est un mécanisme qui exploite la fonction *cgroup* de limitation des ressources (CPU, mémoire, réseau…) complétée par une séparation par espace de nommage ce qui, au final, permet l'isolation complète d'une application. Enfin, Windows Server 2016 propose deux types de containers, les containers Windows qui partagent le même système d'exploitation et les containers Hyper-V plus cloisonnés.

Le processus qui s'exécute au sein d'un container est isolé des autres applications dans un espace de noms virtualisé depuis lequel il accède à un certain nombre de ressources (services, fichiers, réseau) avec l'illusion d'être le seul processus à s'exécuter sur la machine. Ce mécanisme permet de réduire la complexité et la variabilité en favorisant le partage et la réutilisation du container ou de son image, par les développeurs ou par les équipes de gestion opérationnelle. Le partage des ressources entre les containers permet également d'optimiser la consommation des ressources et d'accélérer leur déploiement et leur lancement.

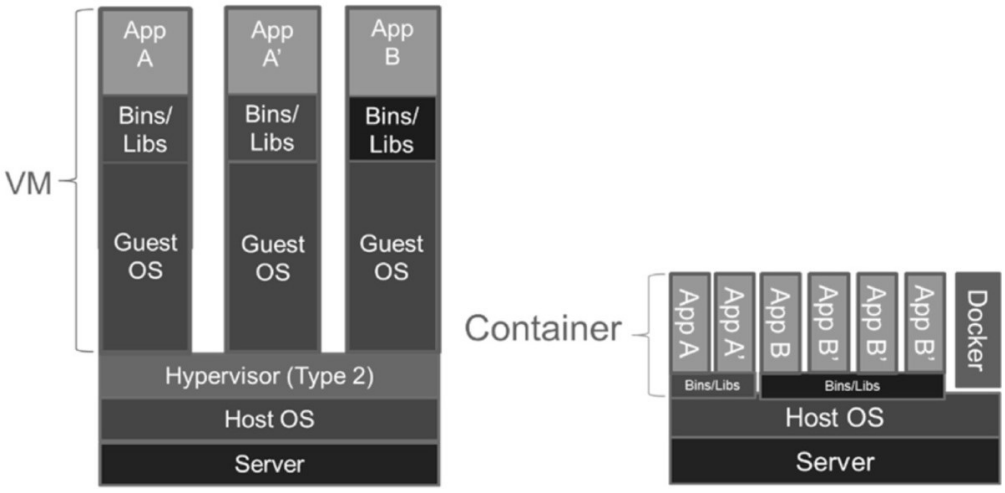


Fig. 4.3 *Machine virtuelle et container*

Aujourd'hui la virtualisation par les containers suscite un intérêt renouvelé grâce à la solution open source Docker qui propose un framework facilitant leur création et leur distribution. Docker permet de déployer des applications de façon très efficace en offrant des caractéristiques d'isolement similaires à celles d'applications s'exécutant dans des machines virtuelles. Nous reviendrons sur Docker à la fin de cet ouvrage.

4.4.2. L'automatisation du provisioning

Dans une société qui a fait le choix du cloud, qu'il s'agisse d'un cloud public ou d'un cloud privé, il s'agit déjà pour l'administrateur système DevOps de définir les règles qui permettront aux utilisateurs de provisionner les environnements dont ils ont besoin, à partir du portail nativement inclus dans la solution cloud. Dans certains cas, la mise à disposition de ce portail (voire d'extensions de ce portail) ne sera pas suffisante. Il faudra alors envisager le développement de capacités spécifiques qui viendront enrichir le système d'information. Automatiser le provisioning d'une machine dépend directement du type d'environnement ciblé.

Provisioning bare metal

Le provisioning *bare metal* consiste dans l'installation d'un OS (ou d'un hyperviseur Type 1 directement sur le matériel cible. L'opération peut être réalisé à distance en communicant le contrôleur BMC (*Baseboard Management Controller*) du serveur cible avec les protocoles IPMI or SMASH. En général ce type de provisioning est proposé à travers les interfaces utilisateurs des solutions de virtualisation.

Provisioning d'environnement virtualisé à demeure

L'automatisation du provisioning d'une machine virtuelle dépend de multiples facteurs. Le développement est fondé sur l'API exposée par l'hyperviseur. Par exemple, supposons qu'il soit nécessaire de provisionner une dizaine de machines virtuelles sur la base d'une image de référence et de les intégrer dans un cluster. Le code suivant présente la syntaxe liée à l'automatisation de ce type de tâche pour un hyperviseur comme hyper-V.

```
$vhdpath = “C:ClusterStorageVolumeVHD”
$vmnetworkName = “ExternalNetwork”
$memorySize = 4GB
$VmtoCreate = 10
$DiskSize = 100GB
$HypervHost = “hyperviseur”
1..$VmtoCreate | % {
New-VHD -Path $vhdpath”VM00$_.VHDX” -ParentPath $vhdpath”BASE2012.VHDX” -Differencing -SizeBytes $DiskSize -ComputerName $HypervHost
New-VM -Name VM00$_ -VHDPath $vhdpath”VM00$_.VHDX” -Memory $memorySize -SwitchName $vmnetworkName -ComputerName $HypervHost
Start-VM VM00$_ -ComputerName $HypervHost
Add-ClusterVirtualMachineRole -VirtualMachine VM00$_ -Name VM00$_ -Cluster $ClusterName
}
```

Sur d'autres hyperviseurs, dans d'autres systèmes d'exploitation, les approches seraient similaires, mais le code serait différent. En effet, quoiqu'elles ciblent le même objectif, les interfaces de programmation exposées par les hyperviseurs KVM (*Kernel-based Virtual Machine*), ESX, XenServer ou Hyper-V ne correspondent pas à une norme. Dans le cloud privé, l'initiative open source OpenStack tente de proposer un modèle d'unification de ces API. Même si l'idée suscite l'intérêt d'importants acteurs du marché, il reste du chemin à parcourir avant d'unifier le modèle de programmation des environnements virtualisés.

Provisioning d'environnement virtualisé dans le cloud

Il n'y a pas plus d'homogénéité dans le cloud public. En effet, il faut s'adapter aux API de provisioning proposées sur les plates-formes AWS, Google ou Microsoft Azure, pour n'en citer que trois… À défaut d'être identiques, les API de *provisioning* sont disponibles sur l'ensemble des systèmes d'exploitation (comme par exemple azure-cli sous Unix ou Mac). De ce point de vue, le cas de PowerShell est intéressant car son extensibilité lui permet d'intégrer des commandes qui permettent de manipuler directement des ressources cloud indépendamment du fournisseur de service cloud. Comme on pouvait s'y attendre, ce type d'usage est possible dans le cloud Microsoft avec Azure PowerShell 1.0 (ainsi que dans Azure Automation, une autre facilité à destination des administrateurs système DevOps tournés vers le cloud). Par exemple, sur Microsoft Azure, une opération similaire à l'exemple de provisioning précédent pourrait être réalisée avec le script suivant.

```
$sqlVms = @()

for($i=1; $i -le 10; $i++)
{
   Write-Host "Creating $sqlVMPrefix${i}"
   $adminPort = $sqlAdminPort + $i

   #create a new VM Config
   $newSqlVM = `
      New-AzureVMConfig -ImageName $WindowsAndSqlServerImageName -InstanceSize $sqlInstanceSize -Name "$sqlVMPrefix$i" `
         -AvailabilitySetName $sqlAvailabilitySet -DiskLabel "$sqlVMPrefix${i}os" `
         -HostCaching $hostCaching -Label "$sqlVMPrefix${i}" |
      Add-AzureProvisioningConfig -WindowsDomain -AdminUsername $adminUsername -Password $adminPassword `
      -Domain $domainName -DomainUserName $adminUsername -DomainPassword $adminPassword -JoinDomain $domainName `
      -NoRDPEndpoint |
      Add-AzureDataDisk -CreateNew -DiskSizeInGB 50 -LUN 0 -DiskLabel "$sqlVMPrefix${i}data1" |
      Add-AzureEndpoint -LocalPort 3389 -Name "RDP" -Protocol tcp -PublicPort $adminPort |
      Set-AzureSubnet $sqlBackendSubnet
   #add the VM config to the collection
   $sqlVms += ,$newSqlVM
}

#show the collection
$sqlVms | format-table

#Create the New cloud Service
New-AzureService -ServiceName $sqlServiceName -Location $location
```

Mais cette ouverture de PowerShell au cloud ne se limite pas à Microsoft Azure : ainsi, les PowerShell AWS Tools permettent également aux développeurs et administrateurs de gérer la plupart des services AWS avec un script PowerShell.

Quoi qu'il en soit, à défaut de disposer d'API identiques, les principes d'invocation de ces multiples API sont relativement similaires. L'essentiel pour l'administrateur système DevOps est de pouvoir être en mesure de scripter et d'automatiser des tâches répétitives de provisioning.

Provisioning de container

Le provisioning de containers Docker requiert la mise à disposition d'un serveur hôte exécutant le moteur Docker sur un server Linux ou Windows Server 2016. Prenons l'exemple d'une machine virtuelle Docker Host Linux hébergée sur Azure. Il existe de multiples façons d'automatiser son provisioning, comme Docker Machine… Cet outil permet de facilement créer des hosts Docker (à demeure ou dans le cloud) et de configurer le client Docker pour qu'il puisse communiquer avec ces serveurs (il génère notamment les certificats et partage les clés assurant la sécurité des échanges).

```
docker-machine -D create -d azure \
—azure-subscription-id="<souscription-id>" \
—azure-docker-port="2376" \
—azure-publish-settings-file="<souscription.publishsettingsfile>" \
—azure-location="West Europe" \
—azure-password="<password>" \
—azure-size="Small" \
—azure-ssh-port="22" \
—azure-username="<username>" \
<nom de la machine docker host>
```

L'administrateur DevOps peut ensuite vérifier si le serveur Docker est opérationnel et si la connectivité est effective. De la même façon, il aurait pu provisionner automatiquement un *cluster swarm* composé de multiples serveurs Docker.

```
C:\DEMOS\21 - CLOUD\AZURE\__CONTAINERS\Docker\Demos\docker-machine>docker info
Containers: 1
Images: 10
Storage Driver: aufs
 Root Dir: /var/lib/docker/aufs
 Backing Filesystem: extfs
 Dirs: 12
 Dirperm1 Supported: true
Execution Driver: native-0.2
Kernel Version: 3.16.0-25-generic
Operating System: Ubuntu 14.10
CPUs: 1
Total Memory: 1.639 GiB
Name: docker-php-stephgou
ID: LPGZ:SRIV:KPHV:XJN3:6UJC:JHPC:PRNS:DGIU:KPRT:DBSC:JY4J:QJ2G
WARNING: No swap limit support
Labels:
 provider=azure
```

Fig. 4.4 *Obtention d'informations sur un Docker Host*

Il est alors possible de rechercher des images dans un dépôt public ou privé, d'en construire de nouvelles et de déployer des containers avec la commande docker run.

4.5. Le déploiement

Une fois l'environnement déployé, que ce soit à demeure ou dans le cloud, il s'agit maintenant pour l'administrateur DevOps d'automatiser la configuration des différentes ressources liées à la machine cible (système, réseau, management…), d'un point de vue système et applicatif, que ce soit à demeure ou dans le cloud et de s'assurer que le résultat obtenu est conforme à ses objectifs. Pour ce faire de multiples outils s'offrent à lui.

4.5.1. De multiples formes d'automatisation du déploiement

Automatiser le déploiement d'une machine consiste à s'assurer qu'elle dispose de l'ensemble des prérequis systèmes et applicatifs pour son intégration dans la solution.

Si la machine cible est une machine virtuelle et si l'on souhaite automatiser un déploiement de machines similaires, l'administrateur peut dans un premier temps configurer chacune des composantes de l'environnement cible et documenter les actions correspondantes, puis archiver la machine virtuelle de référence ainsi construite pour en faire un modèle qu'il pourra cloner autant de fois que nécessaire. Dans ce contexte, l'administrateur DevOps doit avoir établi en amont une configuration exhaustive de l'ensemble des composants pour les environnements qui auraient vocation à être répliqués. Il lui faut ensuite les archiver dans des librairies accessibles par les autres acteurs du système d'information. Cette démarche peut s'inscrire dans un scénario de mise à disposition d'environnements de développement et de tests.

Une autre approche consiste à faire en sorte que chaque nouvelle machine virtuelle démarre en se basant sur une image de base du système d'exploitation, et à la configurer par script en fonction d'un état descriptif correspondant à l'ensemble des prérequis systèmes et applicatifs de l'application. Il devient alors extrêmement simple de répliquer la machine puisque le script de configuration est partagé et géré dans un contrôleur de code source, ce qui de plus a le mérite de documenter l'infrastructure.

Quelle que soit l'approche retenue cette automatisation permet de garantir que les environnements de développement, de test, pré-production ou de production sont identiques et conformes à l'état défini comme étant la référence (qu'il soit scripté ou conservé sous forme d'archivage de modèle de machine virtuelle). Au final, cela se traduit par une diminution du risque d'erreur liée au déploiement sur des plateformes différentes. La responsabilité de l'administrateur DevOps est alors de s'assurer que cette flexibilité se retrouve dans les moyens de mise à disposition de ces environnements que ce soit vis-à-vis de leurs homologues développeurs ou vis-à-vis des services mis en œuvre dans la chaîne de production logicielle.

4.5.2. Les gestionnaires de packages

L'administrateur Système DevOps doit maîtriser les différents gestionnaires de packages correspondant aux systèmes d'exploitation qu'il utilise. En développant des scripts d'interaction avec ces dépôts, il a ainsi la possibilité d'automatiser la configuration des systèmes des environnements dont il a la charge.

Apt-Get

Dans le monde Linux, les administrateurs systèmes DevOps utilisent les gestionnaires de packages tels que YUM, RPM ou APT pour différentes distributions de Linux. L'*Advanced Package Tool* (APT) est une solution open source qui permet de gérer l'installation et la suppression des logiciels sur de multiples distributions Linux en automatisant la récupération, la configuration et l'installation de logiciels.

L'installation d'un package se fait par une simple commande *apt-get* sur le nom du package et permet d'obtenir également tous les packages dont dépend le package cible (avec un système de résolution des conflits). L'administrateur système DevOps peut spécifier le chemin d'accès du dépôt apt-get, dans un fichier de configuration, ce qui lui permet également de définir un dépôt privé. Il peut aussi forcer APT à choisir une version spécifique de package (/etc/apt/preferences). La liste des mises à jour peut être téléchargée à l'aide de la commande apt-get update. L'installation correspondante est déclenchée *via* un apt-get upgrade.

Ces commandes bien connues des administrateurs Linux, car ils les mettent en œuvre régulièrement. Leur usage dans une perspective DevOps peut être étendu par de l'automatisation. Par exemple, un administrateur système DevOps peut décider d'automatiser le téléchargement des mises à jour et faire en sorte d'en être notifié tout en se réservant le droit de décider ou non de leur installation. Pour ce faire, il lui suffit d'installer cron-apt (avec un apt-get install cron-apt), de configurer le fichier /etc/cron-apt/config avec les lignes MAILON="upgrade" et MAILTO= pour y renseigner son adresse, et de copier le script cron dans le répertoire /etc/cron-apt/action.d/ afin qu'il puisse être périodiquement exécuté.

Nuget

Nous avions déjà évoqué l'intérêt d'utiliser Nuget dans une perspective DevOps pour le développeur .NET. Pour l'administrateur DevOps, l'utilisation de ce gestionnaire de package s'inscrit totalement dans un scénario de configuration système (dès lors qu'il s'agit de Windows).

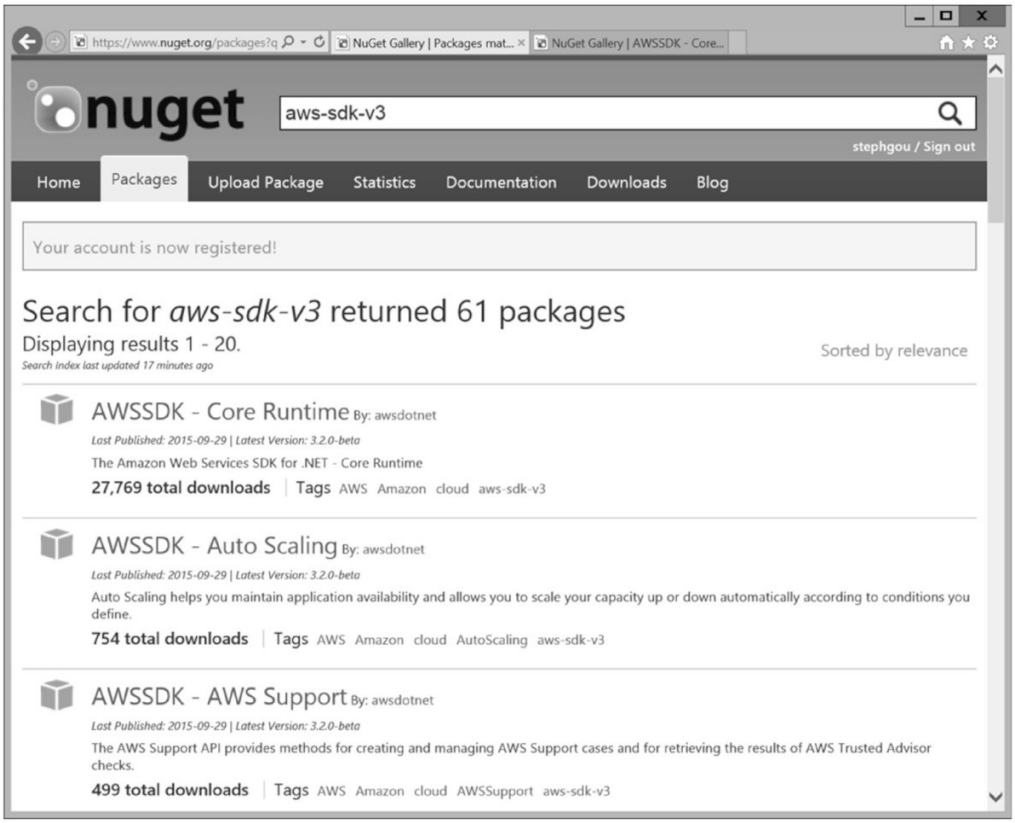


Fig. 4.5 *Rechercher des packages Amazon Web Services avec Nuget*

Chocolatey

Ce type de solution est beaucoup plus récent dans l'histoire de Windows. Le référentiel communautaire Chocolatey est un gestionnaire de packages pour Windows. Il recense aujourd'hui plus de 2930 logiciels uniques. Le fonctionnement de ce dépôt est similaire à ce que propose Apt-Get. Nous l'illustrerons dans son emploi avec PackageManagement, une technologie incluse dans Windows et présentée ci-dessous.

PackageManagement

Dans le monde Windows, il existe de nombreuses technologies d'installations comme par exemple MSI, MSU, APPX… pour ne citer que certaines de celles proposées par Microsoft. L'administrateur DevOps est donc confronté à une difficulté supplémentaire, lorsqu'il souhaite automatiser le déploiement de multiples logiciels. Pour répondre à cette problématique, Windows a récemment intégré un nouveau composant baptisé PackageManagement (initialement appelé OneGet). Il s'agit d'une technologie dont l'objectif est de simplifier la découverte et l'installation de logiciel sur les machines Windows. PackageManagement unifie la gestion de multiples gestionnaires de package *via* une interface extensible. Il permet de gérer une liste de dépôts de logiciels dans lequel les packages logiciels peuvent être recherchés, acquis puis installés (ou désinstallés) silencieusement à partir d'un ou plusieurs référentiels.

PackageManagement inclut nativement de mulitples providers : Bootstrap (celui qui sait où se procurer d'autre providers), Nuget, MSI, MSU (pour gérer les fichiers de mise à jour de Microsoft), ajout/suppression de programmes et PowerShellGet (pour accéder aux

modules PowerShell). Chacun des providers de PackageManagement donne accès à un ou plusieurs référentiels. Ces dépôts peuvent être publics ou privés, accessibles par internet ou uniquement en Intranet. Grâce à l'extensibilité qu'offre le modèle de PackageManagement, il est possible d'ajouter de nouveaux providers, comme Chocolatey.

Le provider vient alors s'ajouter à la liste de l'ensemble des gestionnaires de package (que l'on peut inventorier avec la commande *Get-PackageProvider*). On peut alors rechercher tout applicatif présent sur le dépôt cible dès lors que l'on connaît son nom, avec la commande `Find-Package -Provider chocolatey armclient`. Dans cet exemple, le package recherché est `armclient`, un outil qui permet d'invoquer très simplement l'API Azure Resource Manager, une solution sur laquelle nous reviendrons un peu plus loin dans ce chapitre. Il ne reste plus alors qu'à l'installer, avec la commande `Install-Package armclient`.

Et bien entendu, ces opérations peuvent être assemblées dans un script afin d'automatiser un processus d'installation. Mais pour ce faire, il faut d'abord décider du choix du langage de script.

4.5.3. Les langages de script

Qu'ils ciblent Windows ou Linux, les interpréteurs en ligne de commande font partis des outils incontournables pour l'administrateur système DevOps. Il s'agit pour ce dernier de disposer de commandes qui lui permettent d'interagir avec le système d'exploitation. Il lui faut également pouvoir s'assurer de leur bon fonctionnement par une validation de la syntaxe de l'appel. L'administrateur système a la possibilité de regrouper ces commandes dans un fichier script pour pouvoir les séquencer dans un appel automatisé. Ces scripts peuvent être exécutés manuellement par l'utilisateur ou automatiquement par le système. La plupart des systèmes d'exploitation disposent donc d'un langage de programmation niveau systèmes conçu pour faciliter l'automatisation opérationnelle.

Les langages de script sous Linux

Les systèmes Unix et Linux proposent de multiples interpréteurs en ligne de commande de type script dérivés du Bourne Shell (sh), comme le Korn Shell ou le Bash (*Bourne-Again shell*) qui intègrent plusieurs optimisations issues du C Shell. Bash est le shell par défaut sur de nombreux Unix open source notamment sur les systèmes GNU/Linux (ainsi que sur Mac OS X, et Cygwin l'un des sous-systèmes Unix de Windows). Bash permet d'ordonnancer l'appel de multiples scripts en offrant la possibilité de rediriger la sortie d'un script vers l'entrée d'un autre script (*via* la notion de *pipe*).

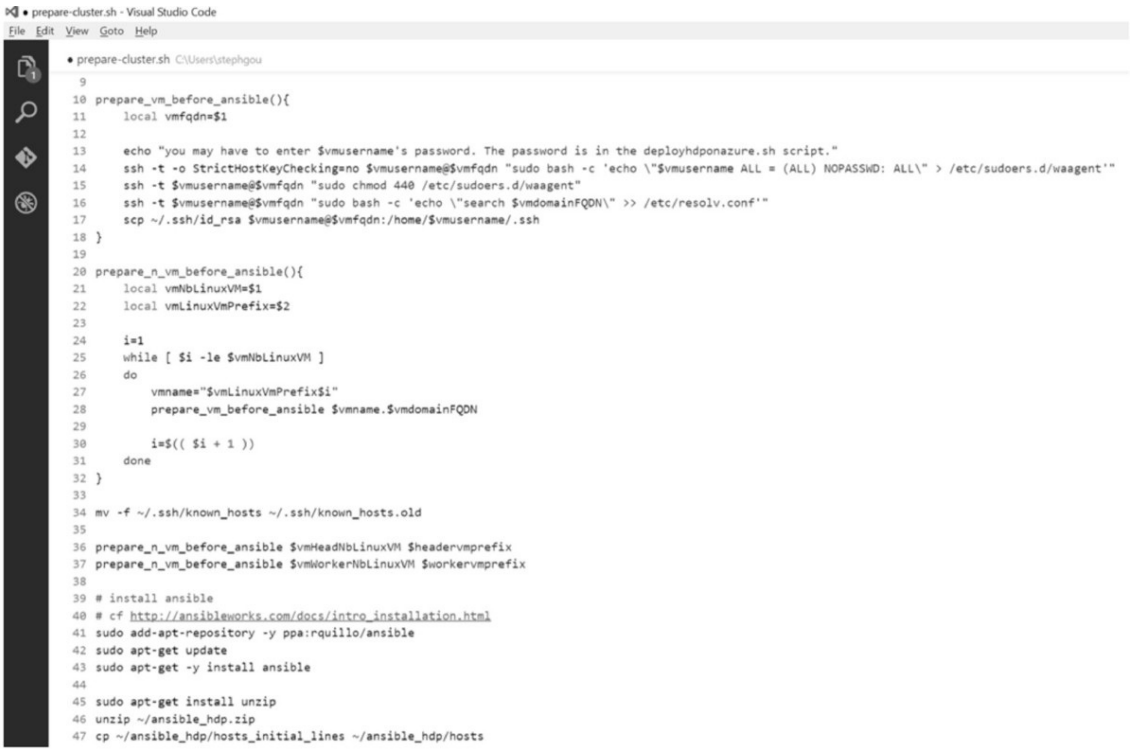


Fig. 4.6 *Shell de configuration d'une machine Linux*

À ces interpréteurs s'ajoutent des langages plus puissants comme Perl ou Python, s'inspirant du C, mais avec un mode d'exécution interprété. Quoiqu'ils puissent être utilisés à d'autres fins que dans le cadre d'une démarche *infrastructure as code*, ils sont très fréquemment utilisés dans ce contexte. Perl (*Practical Extraction and Report Language*) supporte nativement l'utilisation d'expressions régulières ce qui le prédispose aux traitements de séquences de texte dans les environnements Unix, Linux ou Windows.

Les langages de script sous Windows

Pendant de nombreuses années l'administrateur système Windows a dû se contenter de langages batch permettant d'exécuter différents types de commande (système de fichier, réseau…). L'une de ses valeurs ajoutées était alors plus liée à sa capacité à connaître les subtilités techniques de chacune des versions des systèmes d'exploitation qu'il gérait. Par exemple, d'une version à l'autre de Windows, il y a parfois des changements sur la configuration réseau avec l'activation de certains paramètres qui peuvent influencer (dans un sens ou dans l'autre) les performances réseau (*Receive Side Scaling*, TCP *Chimney Offloading*, NetDMA…). Il faut donc connaître la localisation de ces paramètres en base de registre, dans notre exemple (HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters) et les commandes pour modifier ces paramètres, comme netsh int tcp set global chimney=disabled. Fort de ce type expertise, l'administrateur système Windows pouvait jouer un rôle déterminant pour configurer les systèmes au mieux de leur capacité, mais était moins susceptible d'automatiser des tâches complexes sur les environnements dont il avait la charge.

Cette situation a radicalement changé avec l'arrivée de PowerShell, un langage de script développé par Microsoft à partir de la version Windows 7, avec comme objectif de

remplacer les langages batch en s'alignant sur les shells Unix ou Linux. Toutefois, PowerShell se distingue en proposant un développement orienté objet plutôt qu'un modèle d'échange fondé sur un flux d'octets entre l'entrée et la sortie d'un composant. De plus, PowerShell est extensible et permet de manipuler en lignes de commande les classes implémentées dans les bibliothèques du framework Microsoft .NET.

Avec cette évolution, l'administrateur système Windows devient lui aussi éligible à la mise en œuvre de certaines des bonnes pratiques DevOps. Il peut utiliser des outils comparables à ceux du développeur. Par exemple, en plus des lignes de commande, PowerShell propose un environnement graphique d'édition de scripts avec debugger intégré, le Windows PowerShell ISE (*Integrated Scripting Environment*). Il peut archiver ses scripts dans un référentiel de code et les partager avec les autres acteurs du système d'information.

Un script PowerShell (ou un shell Linux) peut être utilisé pour installer un environnement dédié à la configuration (comme Salt, dans cet exemple, un système que nous évoquerons un peu plus loin dans ce chapitre).

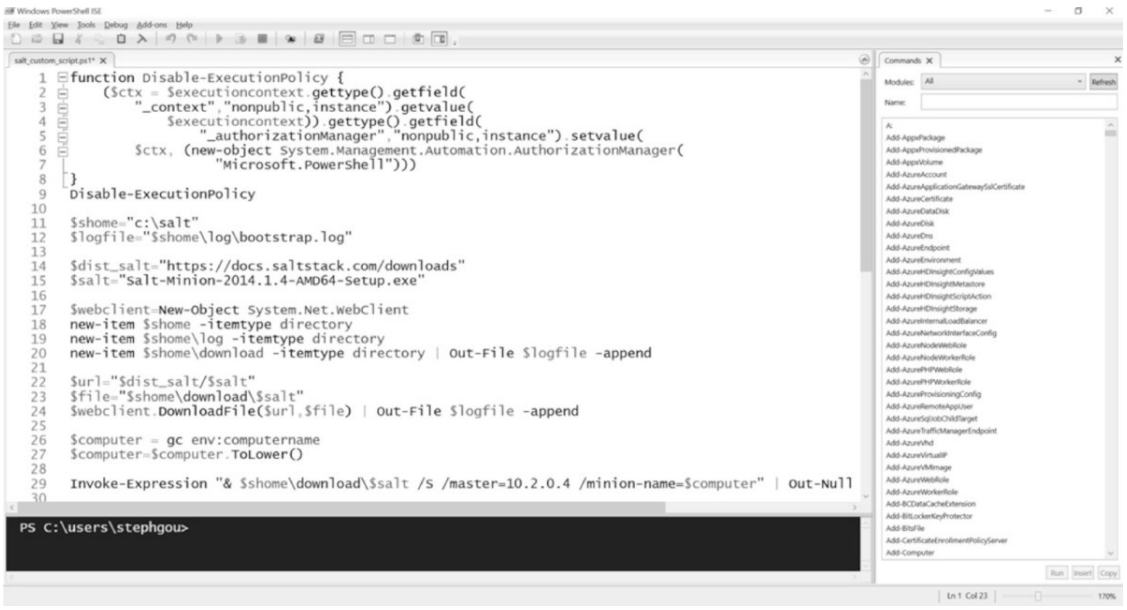


Fig. 4.7 *Installation de Salt depuis PowerShell ISE*

4.5.4. Les solutions déclaratives de gestion de configuration

L'approche programmatique fondée sur l'utilisation de scripts peut être complétée par une approche déclarative pour configurer l'environnement cible en fonction d'une description enregistrée de l'état souhaité d'un ensemble des caractéristiques logicielles. Le langage déclaratif correspondant est implémenté comme un DSL (*Domain Specific Langage*) décrivant des relations de dépendances entre ressources. De multiples solutions open source de gestion de configuration s'appuient sur ce type de modèle.

Des capacités similaires et des différences

Elles permettent toutes de modéliser et de déployer la configuration d'un ensemble de systèmes, physiques ou virtuels, sur des clouds publics ou privés, sur des systèmes

d’exploitation Linux ou Windows. Une de leurs caractéristiques les plus intéressantes est leur *idempotence* : le mécanisme de configuration a le même effet lorsqu’on l’applique une ou plusieurs fois (ce qu’un simple script ne permettait pas). Leurs différences sont principalement liées à la topologie du modèle d’architecture proposé (serveur central comme le proposent Chef ou Puppet ou pilotage en direct par une solution comme Ansible), à la syntaxe du DSL (approche déclarative ou programmatique), à l’affinité de la solution avec tel ou tel langage ou système d’exploitation et au vocabulaire décrivant les éléments constitutifs de la solution de configuration.

L’administrateur DevOps n’aura pas à les connaître toutes, mais très souvent il procédera à une évaluation de la solution la plus appropriée dans son contexte. Parfois, il sera amené à combiner certaines de ces technologies, comme par exemple : Salt avec DSC.

Puppet

Puppet est fondé selon un modèle client/serveur dans lequel l’agent *node* installée sur la machine déployée communique périodiquement avec le Puppet Master afin d’obtenir les configurations associées au *module* correspondant la définition de l’état souhaité. Une fois obtenues ces informations, il suffit alors à l’agent de les appliquer : elles sont fournies sous la forme d’une série d’instructions permettant de configurer le système. Une fois les modifications apportées, l’agent les signale au serveur.

Le code de Puppet est déclaratif. Par exemple si l’on considère le code impératif suivant :

```
if [0 —ne $(getent passwd stephgou /dev/null)$? ]
then
    useradd stephgou —gid sysadmin —n
fi


GID=`getent passwd stephgou | awk —F: {‘print $4]’`
GROUP=`getent group $GID | awk —F: {‘print $1]’`
if [ “$GROUP” != “$GID” ] && [“$GROUP” != “sysadmin” ]
then
usermod —gid $GROUP $USER
fi
```

Le code déclaratif proposé avec Puppet est bien plus concis :

```
user {‘stephgou’:
    ensure =>present,
    gid    =>‘sysadmin,
}
```

Chef

Chef opère également en mode client/serveur. La configuration d’un environnement, baptisée *recipe*, est créée avec le client Chef. Elle résulte de la combinaison d’une série de *policies* implémentées avec le langage Ruby. Les *recipes* et autres ressources sont rassemblées dans un package, le Cookbook. Une fois, construit, ce package est centralisé

sur le serveur Chef afin d'être déployé sur les machines Nodes liées au même environnement (développement, test, pré-production, production). Les Nodes exécutent les instructions du package dans le cadre des policies qui ont été définies et vont régulièrement interroger le serveur afin de savoir si des modifications sont à apporter sur la cible.

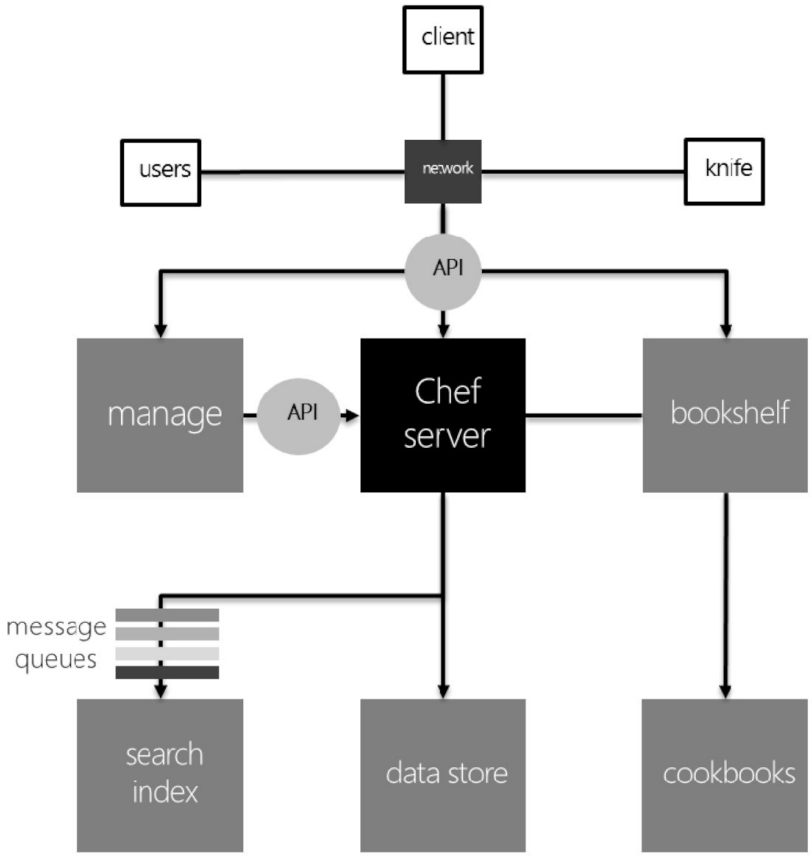


Fig. 4.8 *Composants de la solution de gestion de configuration Chef*

Ansible

Ansible est un système complètement décentralisé : c'est la machine sur laquelle Ansible s'exécute qui pilote directement les systèmes distants *via* le protocole SSH. Il s'appuie sur l'inventaire des machines cibles pour déployer en parallèle ou non les configurations. Les configurations correspondent à des séries d'actions, les *recipes* que l'on peut regrouper dans un playbook en mode impératif, décrit en YAML. Ces actions peuvent être déclenchées en mode push ou en mode pull (avec la commande `ansible-pull`). Ansible est développé en Python mais peut être étendu avec des modules implémentés sur d'autres langages (Ruby…) disponibles sur la poste de contrôle des déploiements… La société qui fournit cette solution vient récemment d'être rachetée par Red Hat.

Salt

Salt est un système distribué qui permet d'exécuter depuis le master des commandes impératives ou déclaratives sur des nœuds distants (les *minions*), individuellement ou selon divers critères de sélection. Les étapes de construction de l'environnement cible sont orchestrées autour d'un bus de communication.

DSC

DSC (*Desired State Configuration*) est une solution déclarative de configuration en mode push/pull permettant de définir l'état d'un environnement. Sur Linux, DSC requiert l'installation d'Open Management Infrastructure (OMI). Sur le système d'exploitation de Microsoft, il est fourni avec *Windows Management Foundation* (WMF qui fait maintenant partie intégrante de Windows 10). DSC est composé de trois éléments : les configurations, les ressources, et le gestionnaire de configuration local (LCM). Les configurations sont des scripts PowerShell déclaratifs qui définissent l'état des instances de ressources. Les ressources sont les blocs de construction impérative permettant de spécifier les composantes à installer et les compléments associés (entrées de clés de Registre dans le cas de Windows, création de fichiers, routines d'installation…). Ils résident dans les modules PowerShell et sont codés pour modéliser les différentes composantes d'un sous-système et mettre en œuvre un flux de contrôle de leurs états. Le LCM est le moteur permettant l'interaction entre les ressources et les configurations. Le LCM interroge régulièrement le système pour assurer le maintien de l'état défini par configuration. Lorsque la configuration est exécutée, DSC (et les ressources appelées par la configuration) fait en sorte que le système s'aligne sur la configuration décrite.

L'utilisation de DSC consiste donc à éditer une configuration :

```
Configuration DevOpsConfiguration{

 param(
  [string[]]$ComputerName="localhost"
 )

  Import-DSCResource -Module nx

  Node $ComputerName
  {
  nxFile DevOpsFile {

     DestinationPath = "/tmp/devops"
     Contents = "Hello DevOps `n"
     Ensure = "Present"
     Type = "File"
  }
  }
}
```

Cette configuration est ensuite compilée pour être transformée en fichier MOF ciblant la machine localhost (on pourrait également appliquer cette configuration à une série de *n* machines et obtenir ce fichier pour l'ensemble des machines concernées).

```
DevOpsConfiguration -OutputPath $DevOpsModulePath
```

Le fichier MOF peut ensuite être déployé en établissant une session distante sur la machine Linux distante avec une CIMSession et en exécutant la configuration qui y est compilée.

```
$Node = “stephgou-dsc-01”
$Credential = Get-Credential -UserName:“root” -Message:“Enter Password:”


#Options for a trusted SSL certificate
$opt = New-CimSessionOption -UseSsl:$true
$Sess=New-CimSession -Credential:$credential -ComputerName:$Node -Port:5986 -Authentication:basic -SessionOption:$opt -OperationTimeoutSec:90


Start-DSCConfiguration -Path: $DevOpsModulePath -cimsession:$sess -wait -verbose
```

Le schéma de la figure 4.9 résume ces trois étapes.

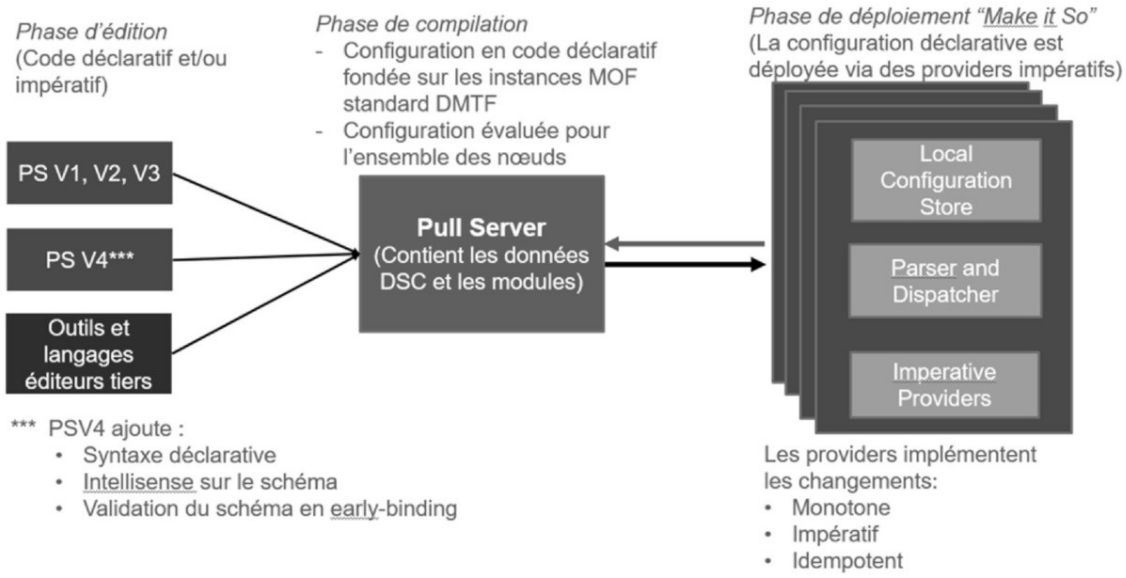


Fig. 4.9 *Édition, compilation et exécution d'une configuration DSC*

4.5.5. Les gestionnaires de modules d'infrastructure

Une des façons de capitaliser sur le code d'infrastructure est de partager le code correspondant. Dans le cas de PowerShell une autre approche permet d'aller encore plus loin dans une optique de réutilisation : les modules PowerShell qui permettent d'empaqueter et de déployer un ensemble de fonctions PowerShell connexes de façon dynamique ou en les persistant sur le disque sous différents formats (script, binaire, manifeste de description…).

PowerShell a suscité dans la communauté un certain nombre d'initiatives pour la réutilisation de ces modules. Plus récemment, Microsoft a créé la Galerie PowerShell, le référentiel central dans lequel sont regroupés les modules PowerShell contenant les commandes PowerShell et les ressources DSC. Certains de ces modules sont créés par Microsoft, et d'autres sont créés par la communauté. Pour interagir avec ce dépôt de modules et faciliter le partage, la découverte, et la consommation des modules

PowerShell, Microsoft propose un dispositif baptisé PowerShellGet (*PowerShell Package Management Module*) que nous avons déjà évoqué. PowerShellGet est un gestionnaire de packages pour PowerShell. Plus précisément, c'est une extension de PackageManagement qui permet de gérer les modules PowerShell sous forme de packages logiciels, en définissant des commandes PowerShell interfaces des commandes PackageManagement associées au fournisseur de la Galerie de ressources PowerShell (baptisé *PSModule*).

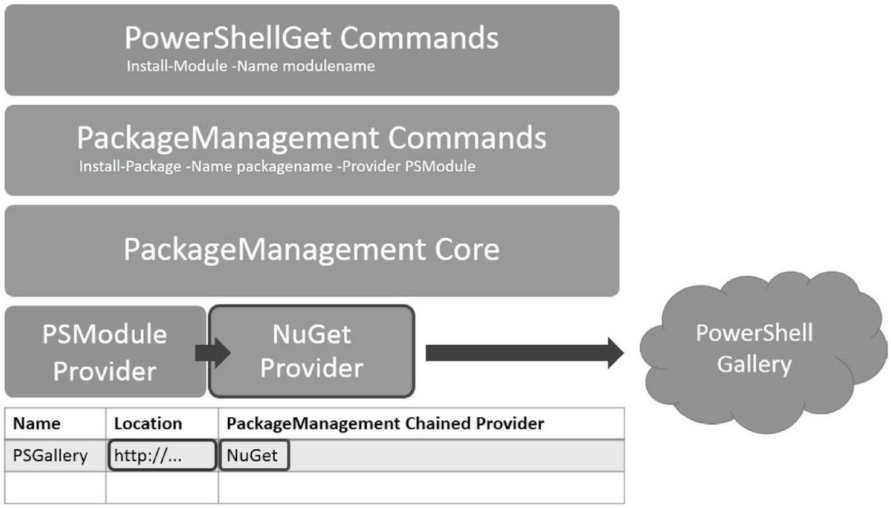


Fig. 4.10 *Gestionnaire de packages pour les modules PowerShell*

Par exemple, supposons que l'administrateur système DevOps souhaite mettre en œuvre le module la ressource xPSDesiredStateConfiguration qui contient les ressources xDscWebService, xWindowsProcess, xService, xPackage, xArchive, xRemoteFile, xPSEndpoint et xWindowsOptionalFeature. Il n'a pas besoin de connaître sa localisation sur la PowerShell Gallery.

Il lui suffit de lancer une recherche et PackageManagement interroge tous les dépôts. Pour le télécharger, il suffit d'utiliser la commande :

```
Install-Module -Name xPSDesiredStateConfiguration
```

Parmi la grande variété de modules proposés, l'administrateur DevOps pourra également s'intéresser au module open source Posh-SSH qui permet une automatisation des tâches sur une machine Linux avec laquelle on communique *via* le protocole SSH. Quoique tous les RFC SSH ne soient pas supportés la solution permet néanmoins d'établir une session SSH ou SFTP avec un couple utilisateur/mot de passe ou des clés OpenSSH, et de lancer des commandes SSH Exec. Pour l'installer, il lui suffira là encore de faire appel à la PowerShell Gallery avec un `Install-Module -Name Posh-SSH`.

4.5.6. Configuration applicative

Une fois le système configuré, reste à installer l'application. Là encore l'administrateur DevOps a tout intérêt à s'appuyer sur des frameworks existants, à qualifier la ou les solutions retenues et à industrialiser leur usage dans la chaîne de production logicielle. Cette étape étant très liée à l'application et à ses dépendances, la collaboration avec les

équipes de développement et l'intégration dans la chaîne de production logicielle est impérative.

Capistrano

Capistrano est un framework open source fondé sur les mêmes principes qu'Ansible (pas de centralisation) et extensible avec le langage Ruby. Il permet de déployer des applications en parallèle de serveurs distants grâce à un fichier de *recipe*, le Capfile qu'il communique *via* le protocole SSH. Capistrano définit un lien symbolique entre le site courant défini et une version du code source (avec un support de gestionnaires de code source tels que CVS, SVN, Mercurial ou Git…). Il gère les retours sur les versions précédentes en modifiant le lien symbolique. Il est possible d'indiquer à Capistrano le nombre de versions à garder en ligne. La purge des anciennes versions est gérée de manière automatique.

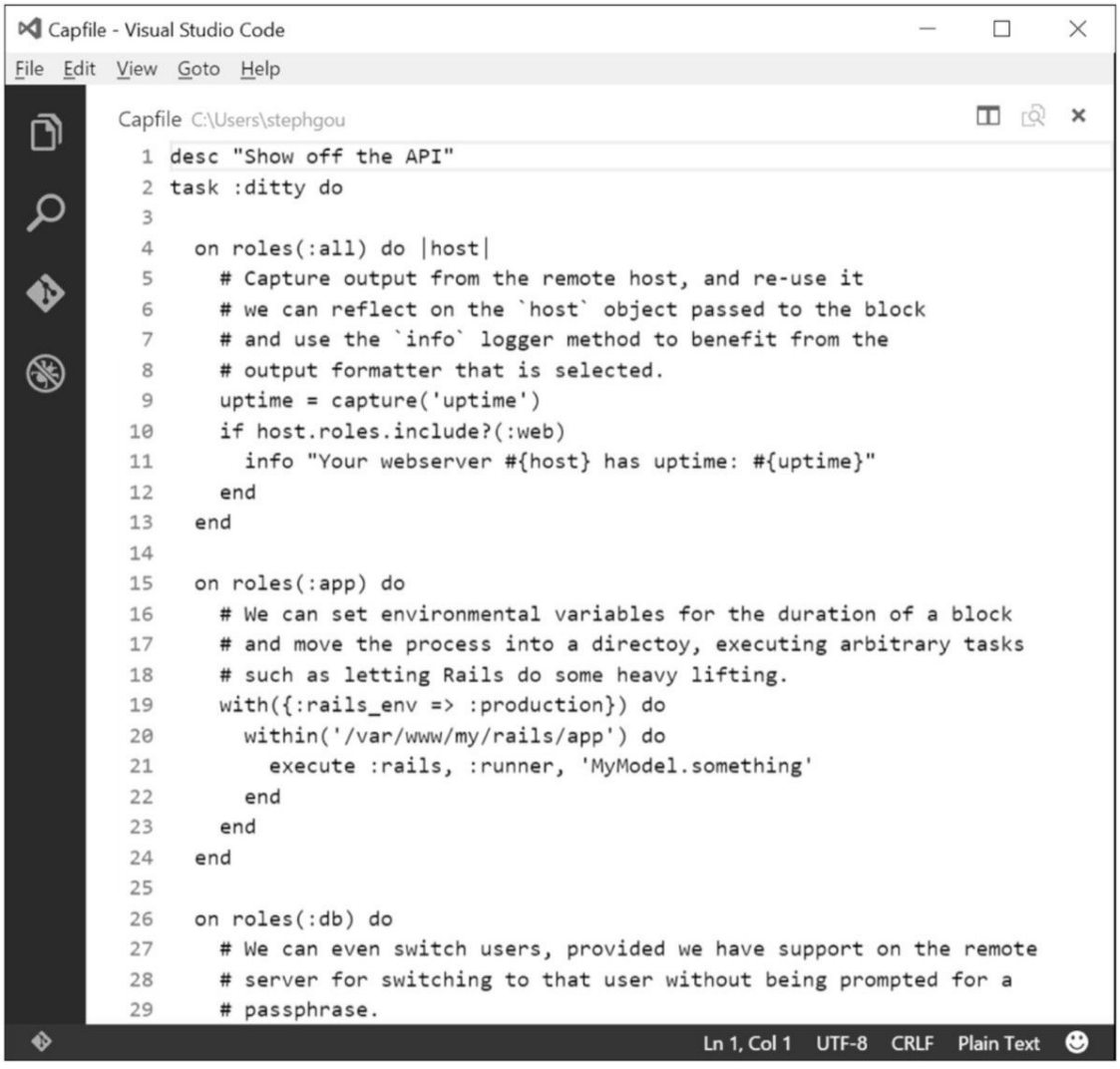


Fig. 4.11 *Fichier Capfile extrait de la documentation Capistrano*

Configuration applicative en environnement Microsoft

De même que sur Linux, la configuration d'une application en environnement se limite rarement à une simple copie de fichiers (mise à jour de la base de registres, droits spécifiques liés à la création de service…). Différentes technologies sont proposées pour

encadrer le processus d'installation. Windows Installer (MSI) est le moteur de référence pour l'installation, la mise à jour et la désinstallation de logiciel. Il est complété par des solutions propriétaires comme InstallShield qui permet de créer des packages logiciels (notamment au format MSI) et WIX (*Windows Installer XML Toolset*), une solution open source permettant de créer des packages MSI à partir de documents XML et de les exécuter en ligne de commande. L'usage de ces différentes technologies peut tout à fait s'adapter à une automatisation par script.

4.5.7. Mise au point et validation de l'automatisation des déploiements

Dans un contexte *infrastructure as code*, l'administrateur DevOps est confronté aux mêmes problématiques que le développeur. Il doit par exemple pouvoir debugger ses scripts d'automatisation, en les instrumentant, ou lorsque la technologie le permet, en les debuggant. Ainsi, PowerShell offre nativement la possibilité d'être exécuté, en pas à pas, avec une vue sur la pile des appels, les points d'arrêt, les sorties d'erreur. De plus, il peut être étendu avec des modules, comme par exemple DebugPx qui fournit un ensemble de commandes s'appliquant aux scripts et aux modules PowerShell en exploitant et en étendant ces fonctions natives de debugging.

La solution open source Pester, permet d'aller un cran plus loin dans l'application de pratiques issues du monde du développement à l'infrastructure, en offrant à l'administrateur DevOps la possibilité de pratiquer une approche TDD (*Test Driven Development* : un sujet sur lequel nous reviendrons dans le chapitre 5 lié à la qualité). Pour ce faire, Pester fournit un framework d'exécution de tests unitaires pour exécuter et valider le résultat de l'exécution de commandes ou de modules PowerShell. Il associe de multiples fonctions à une convention de nommage qui lui permet d'avoir connaissance des éléments à tester.

```
PS C:\Users\stephgou> Get-Module -Name Pester | Select -ExpandProperty ExportedCommands

Key                             Value
---                             -----
AfterAll                        AfterAll
AfterEach                       AfterEach
Assert-MockCalled               Assert-MockCalled
Assert-VerifiableMocks          Assert-VerifiableMocks
BeforeAll                       BeforeAll
BeforeEach                      BeforeEach
Context                         Context
Describe                        Describe
Get-MockDynamicParameters       Get-MockDynamicParameters
Get-TestDriveItem               Get-TestDriveItem
In                              In
InModuleScope                   InModuleScope
Invoke-Mock                     Invoke-Mock
Invoke-Pester                   Invoke-Pester
It                              It
Mock                            Mock
New-Fixture                     New-Fixture
Set-DynamicParameterVariables   Set-DynamicParameterVariables
Setup                           Setup
Should                          Should
```

Fig. 4.12 *Les commandes du framework open source Pester*

Une fois installé Pester (par exemple *via* la solution de PackageManagement déjà évoquée avec un Install-Package depuis un provider nuget.org ou Chocolatey), il suffit de lancer la

commande New-Fixture pour créer un répertoire incluant deux fichiers, un pour le code qui sera la cible des tests et un fichier pour exécuter ces tests. Ce deuxième fichier, judicieusement suffixé par .Tests, intègre en en-tête trois lignes lui permettant de très simplement établir la liaison avec le premier fichier et destiné à valider la ou les fonctions qui y sont définies. Il implémente ensuite dans un bloc *Describe* différentes assertions (un terme bien connu des testeurs et des développeurs) qui comparent les valeurs en entrée avec celles qui sont attendues. Il ne reste plus qu'à lancer la commande Invoke-Pester pour exécuter l'ensemble des tests contenus dans le répertoire courant et ses sous-répertoires, et vérifier, en fonction du résultat de l'évaluation de chaque assertion, si le test s'est achevé avec succès ou pas.

Une des difficultés du test d'infrastructure est liée au fait que l'administrateur DevOps interagit avec des ressources dont l'état varie. Par exemple, un fichier peut être déplacé, verrouillé… Pester offre un niveau d'abstraction à partir de fonctions comme Mock donnant notamment la possibilité de les remplacer avec une autre implémentation ou TestDrive, un répertoire qui n'a d'existence que dans le contexte du Describe. Naturellement, l'administrateur DevOps aura tout intérêt à combiner l'usage de cette approche avec d'autres mécanismes déjà évoqués. Le script suivant illustre cette approche pour valider la compilation des ressources DSC associées à la configuration d'un environnement.

```
$here = Split-Path -Parent $MyInvocation.MyCommand.Path
$sut = (Split-Path -Leaf $MyInvocation.MyCommand.Path).Replace(“.Tests.”, “.”)
. “$here\$sut”

Describe “DevOpsWebSiteDSC” {
   It “DSC MOF file”{

      Set-StrictMode -Off

      $ComputerName = “localhost”

      configuration DevOpsWebSiteConfig{

         param
         (
            # Target nodes to apply the configuration
            [string[]]$NodeName = ‘localhost’
         )
         Import-DscResource –Module PSDesiredStateConfiguration
         Import-DscResource -Module xWebAdministration



         Node $NodeName
```

```
{
    # Install the IIS role
    WindowsFeature IIS
    {
        Ensure          = “Present”
        Name            = “Web-Server”
    }

    # Install the ASP .NET 4.5 role
    WindowsFeature AspNet45
    {
        Ensure          = “Present”
        Name            = “Web-Asp-Net45”
    }

    # Stop the default website
    xWebsite DefaultSite
    {
        Ensure          = “Present”
        Name            = “Default Web Site”
        State           = “Stopped”
        PhysicalPath    = “C:\inetpub\wwwroot”
        DependsOn       = “[WindowsFeature]IIS”
    }

    # Copy the website content
    File WebContent
    {
        Ensure          = “Present”
        SourcePath      = “C:\users\stephgou\DevOpsWebsite”
        DestinationPath = “C:\inetpub\DevOps”
        Recurse         = $true
        Type            = “Directory”
        DependsOn       = “[WindowsFeature]AspNet45”
    }

    # Create a new website
    xWebsite DevOpsWebSite
    {
        Ensure          = “Present”
        Name            = “DevOps”
        State           = “Started”
```

```
                PhysicalPath    = “C:\inetpub\DevOps”
                DependsOn       = “[File]WebContent”
            }
        }
    }
    DevOpsWebSiteConfig $ComputerName -OutputPath Testdrive:\dsc

    “TestDrive:\dsc\$ComputerName.mof” | Should Exist
  }
}
```

Enfin, Pester peut directement être intégré dans les scripts de compilation d’un système d’intégration continue et de release management.

4.5.8. Configuration et container

Tous les mécanismes que nous venons de voir s’appliquent également dans le contexte d’une virtualisation par container. En associant ce type d’approche avec l’utilisation d’une solution comme Docker, les différentes étapes de provisioning et configuration deviennent alors encore plus simples à automatiser ce qui facilite les mises à jour et la maintenance. Les équipes de gestion opérationnelle peuvent alors standardiser les environnements pour les équipes de développement, de qualité et de production indépendamment de l’infrastructure sous-jacente.

4.5.9. Configuration et cloud

Les offres de services d’infrastructure cloud permettent de créer à la demande des machines virtuelles Windows ou Linux, à partir de modèles de machines proposés par le fournisseur ou son écosystème, depuis le portail ou un langage de script. Une fois ces machines virtuelles déployées dans le DataCenter cible, il est fréquent de souhaiter compléter leur installation par une configuration complémentaire.

Extension de machines virtuelles

Le fournisseur d’offre cloud propose souvent des mécanismes d’extensibilité permettant de simplifier les diverses opérations de gestion de VM en fonction du système d’exploitation et pouvant être ajoutées, mises à jour ou retirées tout au long de son cycle de vie. En général, ces mécanismes sont également ouverts aux technologies open source de gestion de configuration reconnues par le marché, comme celles que nous avons précédemment évoquées. Ainsi, les machines virtuelles provisionnées dans le cloud peuvent disposer d’un agent de type Chef, Puppet, DSC… dès la phase de provisioning.

Dans le cas d’Azure, un VM agent propose ce type de mécanisme *via* la notion de VM Extension que l’on peut ajouter manuellement ou par un script. L’administrateur DevOps peut alors associer l’utilisation d’un script de création de machine virtuelle, avec une configuration déclarative, implémentée par exemple avec DSC comme l’illustre l’exemple suivant :

```
#
# Publish the DSC configuration to Azure
#
Publish-AzureVMDSCConfiguration -ConfigurationPath $resolvePath -StorageContext $storageContext -
ContainerName $dscContainerName -Force


#
# Create the VM configuration to use
#
$vmConfig = New-AzureVMConfig -Name $vmName -InstanceSize Small -ImageName $vmImage
$vmConfig = Add-AzureProvisioningConfig -VM $vmConfig -Windows `
                -Password $password `
                -AdminUsername $userName
#
# Setup the DSC extension
#
$vmConfig = Set-AzureVMDSCExtension -VM $vmConfig -ConfigurationArchive "$configScript.zip" `
          -ConfigurationName $dscConfigurationName -ContainerName $dscContainerName

# Add 80 endpoint
$vmConfig = Add-AzureEndpoint -VM $vmConfig -Name "HTTP" -Protocol tcp -LocalPort 80 -PublicPort 80

# Finally create a new VM with the above configuration
New-AzureVM -Location "West Europe" -ServiceName $serviceName -VM $vmConfig -WaitForBoot

# You can use the Azure management portal to examine the VM creation process
# or retrieve the VM object as follows:
$myVM = Get-AzureVM -ServiceName $serviceName -Name $vmName

$status = $myVM | Get-AzureVMDscExtensionStatus

$status

$status.DscConfigurationLog
```

Ces mécanismes d'extensibilité ne sont pas propres à Microsoft Azure. On les retrouve sur d'autres solutions cloud.

Configuration d'applications composée

Les solutions techniques que nous avons vues jusqu'à présent offrent une forte capacité d'automatisation, mais la gestion des ressources associées (base de données, machine virtuelle, compte de stockage, site web…) reste *unitaire*. C'est donc à l'administrateur DevOps d'implémenter les scripts permettant d'agréger l'ensemble de ces opérations.

Toutefois, la gestion d'une application composée de plusieurs ressources reste complexe lorsqu'il s'agit de gérer les permissions ou la supervision technique à l'échelle d'un groupe de ressources. C'est pourquoi les fournisseurs de service cloud proposent des mécanismes permettant de créer et gérer des groupes dans lesquels sont intégrés des regroupements de multiples ressources de même type ou non. Dans ce contexte, un groupe de ressources est un conteneur logique destiné à faciliter la gestion du cycle de vie d'un regroupement de multiples ressources, comme dans le cas d'une application construite autour de sites web, de multiples serveurs applicatifs et bases de données.

La solution proposée pour le déploiement et la configuration d'un groupe de ressources est déclarative afin de faciliter la configuration des ressources, de leurs dépendances, de leurs interconnexions, de la gestion d'identité entre ces ressources, et, le cas échéant, de leur facturation. Elle se fonde sur l'utilisation de modèles.

Par exemple, l'offre cloud AWS, propose tout une série de modèles AWS Cloud Formation que l'administrateur DevOps peut bien entendu étendre, avec pour objectif de garantir l'idempotence, simplifier l'orchestration, la gestion du cycle déploiement, ou le retour sur une version antérieure.

Fig. 4.13 *Extrait du template AutoScaling AWS*

Avec Azure Resource Manager, Microsoft propose une démarche très similaire. Comme dans le cas d'AWS, ces templates sont implémentés en JSON et peuvent donc être gérés dans un contrôleur de code source. Voici par exemple l'extrait d'un template de déploiement.

Fig. 4.14 *Extrait d'un template de déploiement Azure Resource Manager*

4.6. L'optimisation des ressources

L'administrateur système DevOps a également pour objectif d'optimiser les ressources dont il dispose, qu'elles soient à demeure ou dans le cloud. C'est la dimension Lean de DevOps.

4.6.1. L'unification de la gestion des ressources

Avant de pouvoir optimiser les ressources, il faut déjà disposer des moyens pour les gérer efficacement et de préférence de façon unifiée, en termes d'outils et d'API. Du côté des solutions commerciales, Microsoft propose une offre baptisée AzureStack qui unifie les services et les API entre cloud privé et cloud public, mais cette unification ne s'étend pas aux offres concurrentes. Côté open source, à l'initiative OpenStack vient s'ajouter la solution Apache Cloud Stack, sans toutefois offrir de compatibilité avec la précédente.

4.6.2. L'automatisation de la planification de la consommation

Un premier niveau d'optimisation des ressources peut être assuré par l'automatisation des environnements. Ainsi, dans la plupart des cas, une plate-forme de développement, d'intégration, ou de QA n'a pas vocation à fonctionner 24 heures sur 24. L'administrateur DevOps a donc tout intérêt à définir des plannings de libération de ressources, en particulier si ces ressources sont facturées à la consommation par un fournisseur de service cloud. Et bien entendu, le provisioning/deprovisioning de ces ressources doit être automatisé, ce qui requiert l'utilisation de l'ensemble des technologies que nous avons précédemment évoquées.

Les infrastructures sont soumises à des sollicitations variables, ce qui au-delà des plages de fonctionnement/arrêt, suppose la planification de périodes durant lesquelles davantage d'instances de machines virtuelles devront être démarrées et inversement. Parfois, cette

approche par anticipation de la consommation ne suffit pas et il faut mettre en place des moyens techniques permettant d'adapter la puissance de traitement au volume et au nombre des requêtes qui sont soumises sur la plate-forme cible.

4.6.3. La gestion de l'évolutivité des infrastructures

Il faut alors disposer de moyens permettant de calibrer l'infrastructure dynamiquement en fonction de la charge à laquelle elle est soumise. Les offres des grands acteurs du cloud public (Google App Engine, AWS, Microsoft Azure) proposent des mécanismes d'autoscaling, a minima sur la partie compute (machine virtuelle, site web…). Le principe consiste généralement à augmenter le nombre d'instances, en fonction de la charge CPU moyenne sur une période considérée et à diminuer ces instances en prenant en compte une période d'observation plus longue. Ainsi, seule la durée effective de fonctionnement des instances est facturée tout en offrant les meilleurs temps de réponse possibles aux utilisateurs de l'application. Ce type de mécanismes d'autoscaling s'étend maintenant à d'autres types de services comme par exemple les services de base de données avec Azure Database Pools.

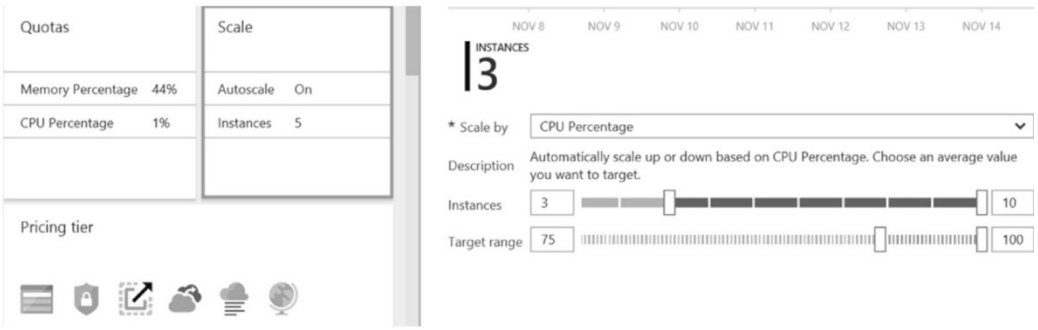


Fig. 4.15 *Configuration manuelle de l'autoscaling d'une Web App*

Ces mécanismes d'autoscaling sont également programmables *via* API comme nous avons pu le voir précédemment avec le template AWS. L'approche déclarative proposée avec le service NetScaler de Google App Engine serait similaire.

Fig. 4.16 *Configuration déclarative de l'AutoScaler de Google App Engine*

L'outil Azure Resource Explorer nous donne un aperçu de la configuration déclarative des métriques permettant de déclencher l'autoscaling sur une Web App Azure.

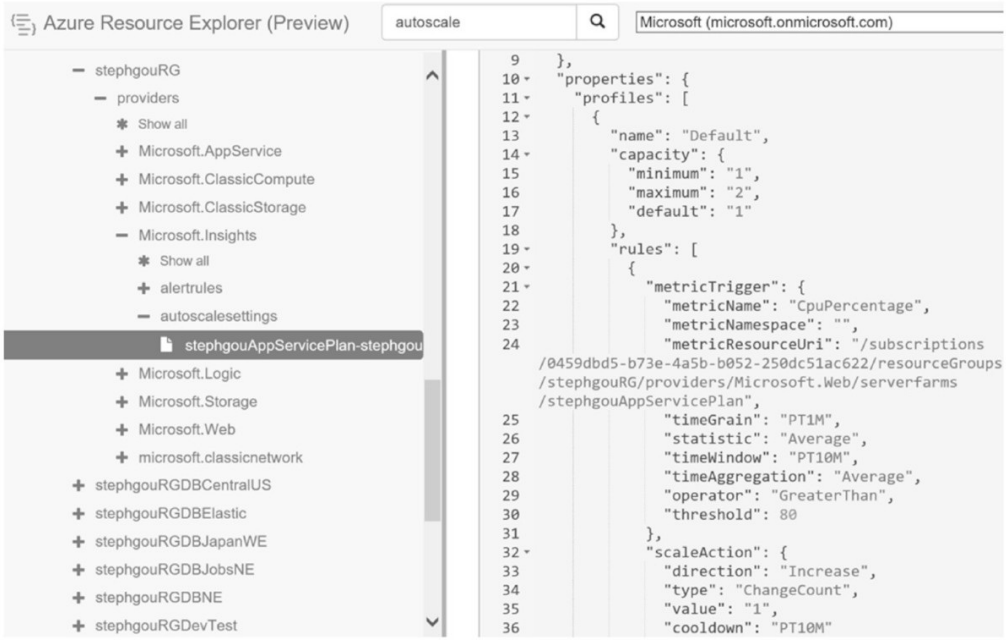


Fig. 4.17 *Configuration déclarative de Azure Resource Manager*

4.6.4. La gestion de la haute disponibilité des infrastructures

Comme nous l'avons vu dans le précédent chapitre, une partie de la responsabilité liée à la résilience du système incombe maintenant au développeur DevOps qui a ajouté à ses compétences la connaissance de patterns de gestion de la haute disponibilité de l'application (comme les patterns Retry et Circuit Breaker par exemple).

Bien entendu, l'administrateur DevOps sera lui aussi fortement impliqué dans la démarche accompagnant la conception d'une architecture dite FailSafe que nous avons décrite dans le chapitre 2. Assurer la continuité des services d'une application nécessite une analyse de ses dépendances, de ses composantes critiques et la définition d'objectifs tels que le RPO (*Recovery Point Objective* : perte de données maximum acceptable suite à un sinistre ou un problème majeur) ou le RTO (*Recovery Time Objective* : délai maximum acceptable de redémarrage de l'activité suite à un sinistre ou un problème majeur). L'objectif pour l'administrateur DevOps est donc d'intervenir dans la définition du cycle de vie de l'application en production (plan de charge), dans la définition du plan de disponibilité, et dans l'identification des différentes étapes de remédiation. L'ensemble de ces éléments s'inscrivent dans le processus qui permet de déterminer un modèle de résilience pour l'application, qui, dans un monde DevOps a vocation à être partagé entre tous les acteurs du système d'information.

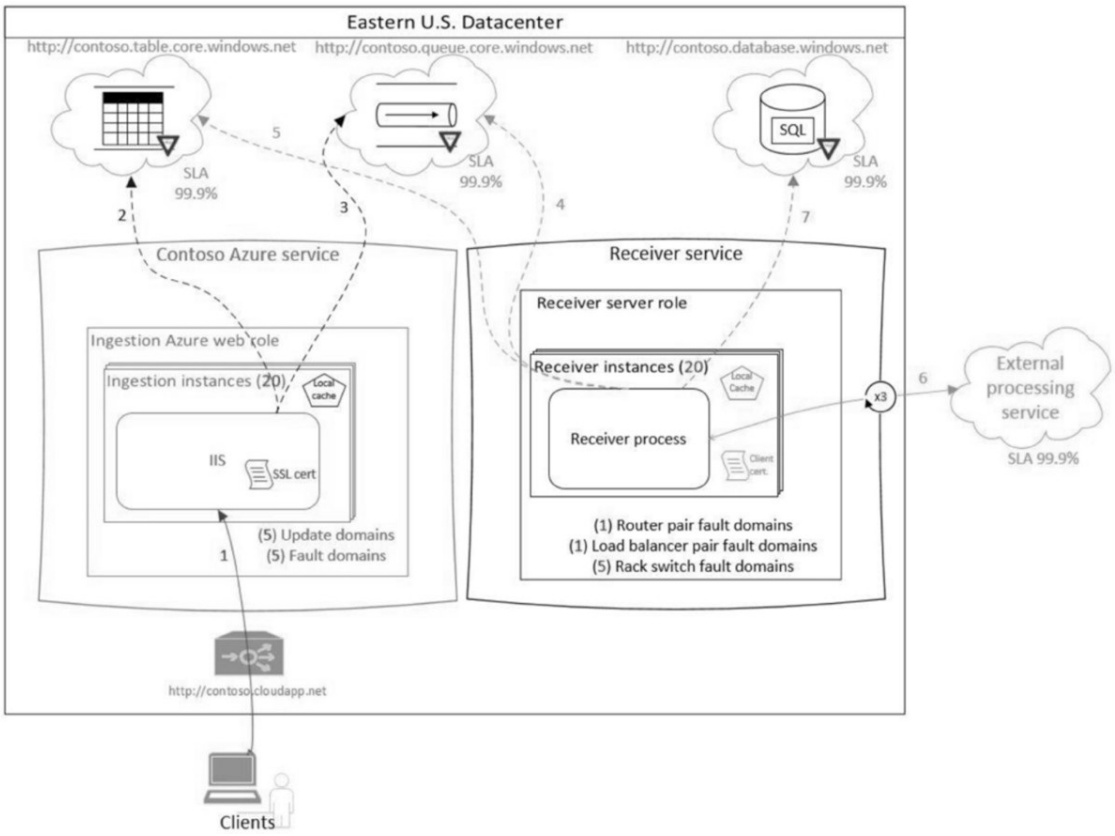


Fig. 4.18 *Modèle de résilience : diagramme d'interaction entre composants*

À cette démarche, s'ajoutent des pratiques expérimentales qui visent à valider la capacité du système à résister à la défaillance de l'un ou l'autre de ses composants, avec l'exemple de l'introduction volontaire de défauts afin de constater la capacité du système à se remettre en service dans une perspective de garantie de la résilience du système après un dysfonctionnement. Nous reverrons plus en détail cette approche (*Fault Injection Testing*) dans l'étude du cas Netflix.

Enfin le clustering de services, de bases de données, la géoréplication des données, le load balancing de datacenter (comme par exemple avec Azure Traffic Manager), la migration *live* d'environnements virtualisés à demeure vers le cloud, sont autant de mécanismes qui participent à la mise en œuvre de solutions permettant d'assurer dans le meilleur des cas la haute disponibilité, et dans le pire, un socle suffisant pour un Plan de continuité d'activité (PCA) acceptable.

4.7. La supervision des infrastructures

DevOps part du principe que la solution finira par rencontrer un dysfonctionnement. Il ne s'agit donc pas de s'affranchir de tout risque d'erreur, car cela reste impossible. Il s'agit de mettre en place des mécanismes permettant d'assurer la remise en fonction du système dans les plus brefs délais. Pour l'administrateur système DevOps, la première étape consiste donc à mettre en place une solution de monitoring, permettant non seulement de déclencher les alertes en cas d'indisponibilité des services, mais également d'anticiper les dysfonctionnements en détectant des franchissements de seuil…

4.7.1. Les mécanismes système de journalisation des évènements

L'ingénieur système DevOps se doit de totalement maîtriser les subtilités des systèmes d'exploitation sur lesquels s'exécuteront les applications, et en particulier d'avoir une connaissance très pointue des mécanismes de journalisation du système. Ces mécanismes peuvent être de très bas niveau. Il faut être en mesure de monitorer le maximum de métriques afin de pouvoir réagir au plus vite en cas de problème. Les métriques d'utilisation de services et de mesure de la performance des services doivent être partagées.

Linux propose nativement un mécanisme de journalisation. Ainsi, le daemon klogd lit les messages du kernel (/proc/kmsg) et les redirige vers le répertoire approprié. Cette approche peut être complétée par l'emploi de syslogd (daemon configurable au niveau utilisateur) pour rediriger ces mêmes messages dans le répertoire /var/log et définir la façon dont ils seront analysés. L'équivalent côté Windows est le mécanisme de journalisation NT Kernel Logger que l'on peut gérer avec des outils comme TraceLog ou TraceView pour activer le suivi d'évènements sur les processus, threads, réseau, I/Os disque… Toutefois, ces services de niveau système doivent être manipulés en ayant connaissance de leurs effets. Par exemple, l'utilisation de syslogd peut conduire à la dégradation des performances du système, si la configuration associée impose de très fréquentes opérations d'écriture sur de nombreux fichiers journaux.

À ces systèmes de journalisation d'évènements système s'ajoutent de nombreux mécanismes de trace de bas niveau (que ce soit pour Linux, Unix ou Windows).

4.7.2. Les outils de trace système

Sous Unix, DTrace est l'un des frameworks d'instrumentation système de référence. Développé à l'origine pour Solaris, il est publié en open source et a été porté sur plusieurs autres systèmes de type Unix. Il s'agit d'un framework global très évolué permettant l'analyse et le suivi système ou applicatif sur les solutions temps réel en production. Il permet ainsi d'obtenir une vue globale des performances du système en cours d'exécution (processeur, mémoires, réseau, I/O disque) et peut naturellement être étendu pour tracer des informations applicatives. Le principe de DTrace consiste à définir des sondes (points d'instrumentation), et à les associer à des actions. Lorsque la condition de déclenchement (démarrage d'un processus, ouverture d'un fichier, exécution d'une ligne de code…) se manifeste, l'action associée est exécutée et les variables liées au contexte de l'appel ou allouées dans la call stack sont capturées pour faciliter une analyse a posteriori par le développeur DevOps.

Linux propose de très nombreux outils de trace intégrés au Kernel, comme ptrace, pour tracer les entrées et sorties de syscall, ftrace capable d'intercepter des évènements en utilisant un système de sonde (tracepoint, kprobes, uprobes), et qui fournit des traces d'évènements avec filtre et timing, ou perf_events pour le profiling CPU.

```
stephgou@stephgou-chef:~$ sudo apt-get install linux-tools-common
Reading package lists... Done
Building dependency tree
Reading state information... Done
linux-tools-common is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
stephgou@stephgou-chef:~$ sudo -s
root@stephgou-chef:~# perf record -F 99 -p 733 -g -- sleep 30
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.015 MB perf.data (~637 samples) ]
root@stephgou-chef:~# perf report -n --stdio --header
# ========
# captured on: Thu Nov 26 22:41:31 2015
# hostname : stephgou-chef
# os release : 3.19.0-28-generic
# perf version : 3.19.8-ckt5
# arch : x86_64
# nrcpus online : 2
# nrcpus avail : 2
# cpudesc : AMD Opteron(tm) Processor 4171 HE
# cpuid : AuthenticAMD,16,8,1
# total memory : 3523168 kB
# cmdline : /usr/lib/linux-lts-vivid-tools-3.19.0-28/perf record -F 99 -p 733 -g -- sleep 30
# event : name = cpu-clock, type = 1, config = 0x0, config1 = 0x0, config2 = 0x0, excl_usr = 0, excl_kern = 0,
# HEADER_CPU_TOPOLOGY info available, use -I to display
# HEADER_NUMA_TOPOLOGY info available, use -I to display
# pmu mappings: software = 1, tracepoint = 2, breakpoint = 5
# ========
#
# Samples: 1  of event 'cpu-clock'
# Event count (approx.): 10101010
#
# Children      Self       Samples  Command  Shared Object       Symbol
# ........  ........  ............  .......  ..................  ......................
#
    100.00%   100.00%             1  waagent  libcrypto.so.1.0.0  [.] 0x00000000000a12ca
              |
              ---0xa12ca

    100.00%     0.00%             0  waagent  libcrypto.so.1.0.0  [.] 0xffff80f9858ac2ca
              |
              ---0xa12ca
```

Fig. 4.19 *Profiling CPU sous Linux : utilisation de perf_events*

L'homologue de DTrace sur Linux est SystemTap. À l'origine implémenté sur Red Hat, on le retrouve depuis sur l'ensemble des distributions Linux. Bien qu'il n'offre pas nécessairement autant de possibilités que DTrace, c'est sans doute l'un des outils de trace les plus puissants sur Linux. Il supporte les sondes USDT (*User-level Statically Defined Tracing*) au même titre que LTTng (*Linux Trace Tool Next Generation*) qui lui se distingue par ses performances dans la collecte d'évènements.

LTTng est une solution de trace très efficace, surtout lorsqu'on l'associe avec des bases de données de type *time series* (influxdb, prometheus…) et des *dashboards* (comme promdash, grafana, kibana…). L'administrateur DevOps a tout intérêt à centraliser les logs des *middlewares* (apache, mysql, nginx, postgrrsql, php-fpm…), les logs applicatifs et les métriques système (CPU, ram, I/O, hits caches…) dans ces mêmes bases et à les superviser avec ces mêmes outils. L'ensemble de ces logs pourront ainsi être mis à disposition des différents acteurs DevOps (développeurs, opérations, business, management, clients) *via* des tableaux de bord dédiés.

Sous Windows, le framework *Event Tracing for Windows* (ETW) offre au développeur DevOps la possibilité de démarrer, instrumenter et arrêter des sessions de suivi d'événements. Ces événements de trace peuvent associer des informations propres à l'application enrichies par des données permettant d'effectuer une analyse de capacité et de performances. Sa mise en œuvre ne suppose pas l'application de verrous sur le dossier de journalisation. L'écriture dans un fichier est gérée de façon asynchrone par le système d'exploitation, l'impact sur les performances se limite donc juste à la consommation du processeur pour exécuter les opérations liées à la journalisation. Enfin, EWT est un mécanisme fortement typé, extensible et versionnable. Pour l'utiliser on ne peut pas se

contenter d'appeler quelques API de journalisation. Il faut créer un fichier manifeste incluant la description des événements que l'on souhaite intercepter et la publier afin qu'elle soit prise en compte. Chaque source dans le modèle de trace System.Diagnostics peut être associée à de multiples *trace listeners* qui vont déterminer la façon dont les données sont tracées et leur format. L'exemple illustre l'ajout d'un listener EWT à la configuration.

```
<system.diagnostics>
  <sources>
    <source name="System.ServiceModel"
        switchValue="Verbose,ActivityTracing"
        propagateActivity="true">
      <listeners>
        <add type=
              "System.Diagnostics.DefaultTraceListener"
          name="Default">
          <filter type="" />
        </add>
        <add name="ETW">
          <filter type="" />
        </add>
      </listeners>
    </source>
  </sources>
  <sharedListeners>
    <add type=
          "Microsoft.ServiceModel.Samples.EtwTraceListener,
           ETWTraceListener"
        name="ETW" traceOutputOptions="Timestamp">
      <filter type="" />
    </add>
  </sharedListeners>
</system.diagnostics>
```

Que ce soit sous Unix, Linux, ou Windows, la connaissance précise de ces mécanismes et leurs effets permet à l'administrateur DevOps d'échanger avec son homologue développeur pour l'inviter à instrumenter l'application en étant très proche du système. Cette collaboration est d'autant plus nécessaire qu'il n'est pas toujours très simple d'intégrer ces mécanismes dans un processus de développement traditionnel, d'où la nécessité pour l'administrateur système DevOps d'expérimenter pour parvenir à un réel de degré de maîtrise de ces solutions en fonction des systèmes d'exploitation utilisés, sur les infrastructures dont il est responsable.

4.7.3. Le monitoring des infrastructures

Dans le cadre d’un processus de continuous delivery, la fréquence de livraison logicielle est fortement réduite, il devient encore plus important de recueillir des informations techniques en phase de tests, mais aussi en production afin de résoudre plus rapidement les problématiques opérationnelles. Il faut pouvoir capturer des données de réseau ou de performance, et établir des corrélations entre les données historiques ou en temps réel, afin d’en tirer le maximum d’informations utiles. Les données qui intéressent les équipes de gestion opérationnelle au premier chef sont celles qui sont liées à la disponibilité du service, à son dimensionnement, à la remontée d’erreurs applicatives, ainsi qu’aux performances des serveurs de l’infrastructure en termes de CPU, mémoire, réseau, I/O disque. Les outils de supervision offrent le suivi de métriques de façon chronologique (*time series*), et des fonctions d’alerte.

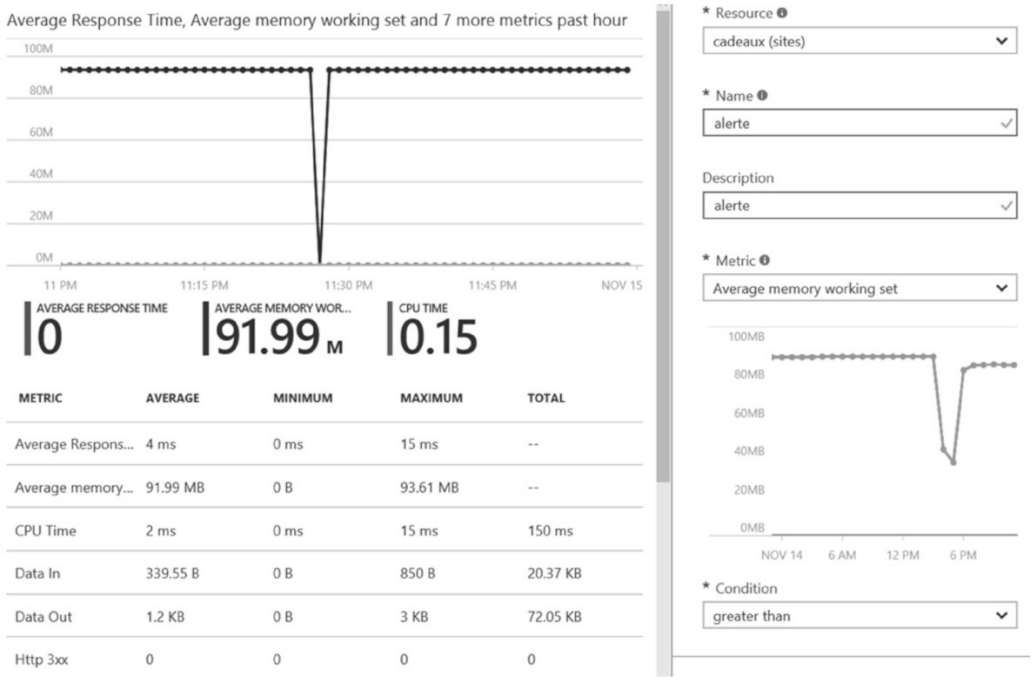


Fig. 4.20 *Supervision d’informations techniques sur une application web*

Le niveau de pertinence de ces données est tributaire de celui de l’instrumentation mise en place par le développeur et couplée avec les données nativement capturées par les systèmes. Dans ses activités liées à la supervision, l’administrateur DevOps est amené à utiliser des tableaux de bord pour la visualisation, avec des règles d’acquisition préconfigurées et des algorithmes pour faciliter le *capacity planning*, le suivi de l’application des mises à jour, la détection de changement ou l’audit de sécurité.

Il existe un grand nombre de logiciels qui offrent ce type de fonctions : des solutions propriétaires comme *System Center Operations Manager* (SCOM) et *Operations Management Suite* (OMS) ou open source comme Zabbix, Nagios, Prometheus ainsi que le triptyque issu de la combinaison de Logstash, ElasticSearch et Kibana pour assurer une lecture et une recherche efficace sur les centaines de milliers de lignes de traces log que peuvent générer les services d’une application.

Fig. 4.21 *Tableaux de bord de supervision*

À cela s'ajoute des solutions offrant des fonctions plus spécifiques comme, par exemple, les outils IPAM (*IP Address Management Software*) qui permettent, au sein d'une organisation, de gérer, superviser et auditer les adresses IP allouées, et d'anticiper un éventuel dépassement de capacité. Parmi elles on peut citer des solutions open source comme GestioIP, IPplan, OpenIPAM, ou des mécanismes nativement intégrés dans le système comme dans le cas de Windows Server.

4.7.4. Monitoring et API

De nombreuses possibilités s'offrent aujourd'hui à l'administrateur DevOps pour requêter ou étendre les solutions de monitoring avec du code.

Par exemple, l'outil open source Consul permet (par le DNS ou *via* HTTP) de découvrir et de configurer les services d'une infrastructure. Il offre également un mécanisme de vérification de disponibilité (utilisé pour interrompre le trafic en direction des serveurs considérés comme indisponibles).

Autre exemple, le framework open source de monitoring Shinken.io, extension Python qui permet de bâtir des plugs in pour Nagios afin de pouvoir mieux superviser les systèmes cibles. La démarche est complétée par un partage de ces plugs in en open source sur GitHub. De même, Microsoft propose la possibilité de bâtir des Management Packs .mpx de monitoring ou d'alerte pour venir enrichir la solution *System Center Operation Manager*.

Les fournisseurs de services de cloud public, qu'il s'agisse d'Amazon, de Google ou de Microsoft proposent des API REST permettant d'interroger les bases de monitoring intégrées. Le code suivant présente le corps JSON de la requête timeseries que l'on peut adresser sur l'API googleapis.com.

```
{
  “commonLabels”: {
    (key): string
```

```
  },
  “timeseries”: [
   {
    “timeseriesDesc”: timeseriesDescriptors Resource,
    “point”: {
     “start”: datetime,
     “end”: datetime,
     “boolValue”: boolean,
     “int64Value”: long,
     “doubleValue”: double,
     “stringValue”: string,
     “distributionValue”: {
      “underflowBucket”: {
       “upperBound”: double,
       “count”: long
      },
      “buckets”: [
       {
        “lowerBound”: double,
        “upperBound”: double,
        “count”: long
       }
      ],
      “overflowBucket”: {
       “lowerBound”: double,
       “count”: long
      }
     }
    }
   }
  ]
}
```

À cela s’ajoutent des API de recherche sur les différents types d’informations souhaités et la capacité de configurer le déclenchement d’alertes. Et toutes ces configurations doivent pouvoir être automatisées. Par exemple, l’API REST Log Analytics Search peut être accédé par l’API Azure Resource Manager, ce qui permet à l’administrateur DevOps de communiquer le service OMS et d’exécuter des requêtes JSON. En fonction des résultats à ces requêtes, il est alors possible de prendre des actions correctives ou remonter des alertes. Le script suivant illustre l’utilisation d’une orchestration Azure Automation exploitant l’application ARMClient pour lancer une requête afin de vérifier si des tentatives de connexion avec un compte particulier se sont produites.

```
workflow Invoke-OMSsearchQuery
```

```
{
param
 (
 [Parameter(Mandatory=$false)]
 [string]$StartDate = “2015-07-10”,
 [Parameter(Mandatory=$false)]
 [string]$StartTime = “09:00”,
 [Parameter(Mandatory=$false)]
 [string]$EndDate = “2015-07-15”,
 [Parameter(Mandatory=$false)]
 [string]$EndTime = “09:00”,
 [Parameter(Mandatory=$false)]
 [string]$query = “Type=SecurityEvent EventID=4625 | Measure Count()
                by TargetAccount”
 )
#Retrieve stored variables
$AzureSubscription = Get-AutomationVariable -Name ‘AzureSubscription’
$OMSWorkspaceName = Get-AutomationVariable -Name ‘OMSWorkspaceName’
$OMSResourceGroupId = Get-AutomationVariable -Name ‘OMSResourceGroupId’

Write-Output “Executing runbook on hybrid runbook worker: $env:ComputerName”

#login with a Service Principal, syntax armclient, spn(=TenantID), Username,
 Password
$SPNforOMS = Get-AutomationVariable -Name ‘SPNforOMS’
$cred = Get-AutomationPSCredential -Name ‘AzureUserNameForOMS’
$null = ARMClient.exe spn $SPNforOMS $cred
#Use a dynamic OMS search
$QueryRange = $StartDate + “T” + $StartTime + “:00.231Z’,‘end’:’” + $EndDate
            + “T” + $EndTime + “:00.231Z’”
$query = “{‘query’:’$query’,‘start’:’$QueryRange}”
$json = armclient post /subscriptions/$AzureSubscription/resourceGroups/$OMSResourceGroupId/
providers/Microsoft.OperationalInsights/workspaces/$OMSWorkspaceName/
search?api-version=2015-03-20 $query | ConvertFrom-Json
$queryResults = $json.value
Write-Output “**** Executing query*** ” $query
Write-Output “Query results: ” $queryResults
}
```

En résumé

DevOps représente avant tout une transformation culturelle pour les équipes opérationnelles. DevOps suppose un changement majeur de leur rôle au sein d’une organisation dans laquelle chacun a désormais la possibilité de travailler avec plus d’autonomie, tout en garantissant la sécurité des services d’infrastructures concernés.

L’administrateur système DevOps a pour mission de rendre l’infrastructure plus agile en la dotant de capacités de provisioning d’environnements virtualisés et de leur configuration automatisée. Pour y parvenir, il doit lui aussi devenir un développeur afin de maîtriser les multiples langages lui permettant scripter les différentes opérations liées à l’automatisation, d’utiliser les solutions de packaging et les outils de contrôle du versioning de code.

À cela s’ajoute la nécessité d’optimiser la consommation des ressources, notamment en s’appuyant sur les services d’autoscaling offerts par le cloud. Enfin, l’administrateur système DevOps participe à la mise en place du suivi en continu du comportement de l’application.

L’ensemble de ces éléments remettent en perspective le rôle clé de l’administrateur système DevOps dans un processus de continuous delivery et la nécessité pour lui de collaborer encore plus activement avec l’ensemble des acteurs du système d’information, en particulier les développeurs.

Chapitre 5

DevOps vu par la qualité

S'il est simple de reconnaître un produit ou un service de bonne qualité, il est plus difficile d'en donner une définition absolue. C'est d'ailleurs ce que l'architecte et anthropologue Christopher Alexander avait baptisé *Quality Without a Name*, dans son ouvrage *The Timeless Way of Building* sur une nouvelle théorie architecturale fondée sur les modèles des langues (et qui par la suite est devenu une source d'inspiration pour le monde logiciel…). En premier lieu, le concept de qualité sous-tend un certain nombre d'idées liées à l'optimisation d'un produit ou d'un service et les moyens que l'on se donne pour faire en sorte qu'il apporte le maximum de valeur à nos clients.

Et en définitive, c'est un point de vue qui reste subjectif, celui du consommateur, qui va permettre de juger ou non de cette qualité. Naturellement, il s'agit de s'assurer que le produit ou le service est fiable et que son comportement est bien celui souhaité par l'utilisateur. D'un point de vue organisationnel, la qualité c'est surtout la capacité à respecter les délais et à s'aligner sur un *time to market* conforme à la stratégie business de l'entreprise. Enfin, la qualité est également liée à la façon dont le produit ou le service sera produit. La qualité sera alors considérée à l'aune de critères qui permettent de mesurer l'efficacité de la chaîne de production logicielle. Enfin, ne perdons pas de vue la sécurité, une composante majeure de la qualité, qui elle aussi est à reconsidérer avec DevOps.

Optimisation et fiabilité du service, alignement sur le *time to market*, efficacité de la chaîne de production logicielle, et sécurité autant d'éléments qui font de la qualité un impératif de la transformation DevOps.

5.1. Évolution des métiers de la qualité

La qualité concerne tous les acteurs de la chaîne de production logicielle. Aux développeurs et équipes de gestion opérationnelle viennent s'ajouter les ingénieurs qualité et les testeurs. Leurs compétences restent d'actualité mais l'évolution vers un monde de services a quelque peu modifié la nature de leurs activités, leur périmètre de responsabilité et l'organisation du système mis en place pour satisfaire aux exigences de qualité.

DevOps : la culture de la qualité…

Adopter une démarche DevOps, c'est souhaiter mettre en place une culture centrée sur l'amélioration continue de la qualité, de la performance, et de la productivité. Cela suppose l'adhésion et l'engagement de l'ensemble des acteurs avec la nécessité, là encore, d'un apprentissage en continu, non seulement pour faire monter les équipes en compétence sur les nouveautés technologiques liées à une évolution de leur métier (comme le *Data Driven Engineering*), mais aussi pour interpréter et comprendre le flux ininterrompu de données remontées par les utilisateurs du système. Avec DevOps, l'utilisateur est plus que jamais au centre des systèmes et processus liés à la qualité…

5.1.1. Évolution du marché

Comme nous l'avons vu dans les précédents chapitres, le monde du logiciel s'est progressivement transformé en un monde de services, dans lequel chacun aspire à bénéficier régulièrement de mises à jour, qu'il s'agisse d'évolutions fonctionnelles ou de corrections de bugs. Et ce nouveau modèle s'est également imposé sur les produits logiciels et sur de nombreux types de périphériques qui tirent parti des possibilités qu'offre la large diffusion des protocoles de communication pour pouvoir, eux aussi, être régulièrement mis à jour, jusqu'à la prochaine rupture technologique. L'explosion du marché de la technologie grand public révèle l'appétit des consommateurs pour des solutions plus innovantes mises à disposition avec un rythme toujours plus soutenu. En contrepartie, ils sont sans doute plus prêts à accepter un dysfonctionnement provisoire, s'il est corrigé rapidement et s'il offre l'accès à une nouvelle fonction.

L'évolution du marché du logiciel se traduit donc par l'émergence de nouveaux modes de production logicielle avec une fréquence accrue de livraison ce qui n'est pas sans impact sur la qualité. Il faut aligner les objectifs et les métiers liés à la qualité avec ces nouveaux impératifs du marché. Cela n'est envisageable qu'à condition de maintenir un niveau de qualité acceptable. Cela suppose une évolution significative de ces métiers, évolution dans laquelle une démarche DevOps peut très clairement jouer un rôle de facilitation et d'accélération.

5.1.2. Impact organisationnel

D'un point de vue organisationnel, en simplifiant et fluidifiant leur communication, DevOps rapproche le monde des testeurs fonctionnels de celui des développeurs responsables de la correction de bugs, ou de celui des équipes de gestion opérationnelle en charge de la validation de la disponibilité des infrastructures et du bon fonctionnement des systèmes. Par exemple, les campagnes de tests ont tout intérêt à être définies dans un référentiel commun accessible à l'ensemble des acteurs du cycle de vie logiciel : développeurs, testeurs et responsables de la gestion opérationnelle auront accès aux données fonctionnelles, aux éléments permettant de lancer les tests de performance, ainsi qu'aux résultats de chacun de ces types de test.

DevOps impacte également la définition des modalités d'application de ces tests que ce soit dans leur planification dans les différentes phases du cycle de vie logiciel (fréquence, conditions de déclenchement de ces tests), dans la mise en place de tests en production ou dans l'implication des différents acteurs du projet. Quel que soit le système mis en œuvre pour la gestion du plan de tests, il devra s'intégrer avec les mécanismes permettant d'automatiser l'exécution de ces tests sur l'infrastructure cible.

5.1.3. Évolution du rôle de l'ingénieur qualité

Selon l'ISO la qualité est *l'aptitude d'un ensemble de caractéristiques intrinsèques à satisfaire des exigences*, une définition qui ne permet pas d'établir une vision très précise du périmètre des activités d'un ingénieur qualité. De nombreuses difficultés peuvent se

présenter au cours du développement d'une solution et le risque ne peut jamais être éliminé. Le rôle de l'ingénieur qualité est d'auditer les actifs du projet (documents de design, code, architecture, sécurité), d'évaluer les risques, et de définir les stratégies de pilotage du projet et de tests de la solution. Il doit gérer les risques considérés comme acceptables et faire en sorte d'éviter tout autre type de menace.

Tout cela suppose une gamme de compétences assez étendue. L'ingénieur qualité a pour principale mission d'aider les développeurs et les équipes de gestion opérationnelle à atteindre leurs objectifs en prévenant tout risque qui puissent les empêcher de les atteindre. Dans une chaîne de production DevOps, il doit également s'assurer que chaque outil est utilisé à bon escient et identifier les tâches d'automatisation complémentaires pour exécuter des vérifications de différents ordres (configuration système, applicative…). Son rôle est aussi de se focaliser sur l'ajout de valeur pour le client et sur ses attentes. À ce titre, il doit décider des données d'utilisation qu'il convient de capturer et de la façon de les collecter et de les exploiter. Il devient donc l'un des acteurs principaux de la qualité pilotée par les données (*Data Driven Quality*), un sujet que nous explorerons un peu plus loin dans ce même chapitre. L'évolution de son rôle implique également l'acquisition de connaissances sur les sciences et les outils liés aux données.

5.1.4. Évolution du rôle du testeur logiciel

Comme leur nom l'indique, les testeurs ont pour mission d'assurer la qualité du livrable par la mise en œuvre de tests logiciels. Le test logiciel s'appuie sur des cas de test et leurs exécutions. Les résultats des tests fournissent les données permettant de faire une évaluation de la qualité. À l'origine, un testeur logiciel doit être en mesure de définir, documenter, puis automatiser les tests permettant de valider le bon fonctionnement de l'application cible. Mais aujourd'hui, pour se différentier sur le marché, il convient de pouvoir identifier les pistes d'optimisation de la qualité du service ou du produit, et agir plus tôt dans le cycle de vie afin d'augmenter la probabilité de succès et de réduire les risques. Pour ce faire, il faut réduire les incertitudes en capturant différents types de mesures.

Dans le cas du service, il est maintenant possible de contrôler le déploiement et de superviser la disponibilité, les performances, les dysfonctionnements et l'usage. Si le système est bien conçu, un flux constant de données de télémétrie est émis par le produit ou le service. Il devient alors possible d'exploiter ce flux de données pour améliorer la qualité des produits.

Moins de tests à implémenter et une partie de ces tests directement assurée par les développeurs ou les utilisateurs (dans le cas des tests en production) : tout cela n'est pas sans incidence sur le métier des testeurs logiciels, qui ont donc tout intérêt à développer de nouvelles capacités. Le testeur logiciel est maintenant en situation d'obtenir des données sur un problème quasiment au moment où il se produit et le rôle du testeur est de faire en sorte que les développeurs et les équipes de gestion opérationnelle soient plus efficaces pour y remédier alors que la solution est en production. Non, le test n'est pas mort, au contraire de ce qu'affirmait Alberto Savoya pour lancer sa keynote et obtenir un maximum

d'attention, lors de l'Annual Google Test Automation Conference 2011. Certes, le rôle du testeur évolue, mais il conserve sa mission de pourfendeur de bugs avec toutefois une réorientation de ses priorités.

Il s'agit de veiller à ce qu'aucun bug *critique* ne vienne interrompre durablement le bon fonctionnement du système et de s'assurer que l'infrastructure et les processus mis en place permettent de réagir très rapidement pour y répondre, ce qui au final représente sans doute une diminution du nombre de tests à mettre en place. Il faut adapter le code de test pour qu'il alimente les données d'instrumentation du système et qu'il apprenne à analyser les journaux d'évènements qui vont fusionner le résultat des tests et les données capturées.

Le testeur DevOps doit appliquer la même démarche quel que soit l'environnement sur lequel s'exécute l'application. Il doit connaître et comprendre le fonctionnement du système en charge de l'utilisation réelle. Il doit également se concentrer sur les données associées aux comportements des utilisateurs et apprendre à rechercher des modèles dans les données. Il doit être en situation de pouvoir distinguer les zones dans lesquelles l'expérience utilisateur est potentiellement insatisfaisante (dysfonctionnement, latence, navigation complexe…) de celles dans lesquelles le service est rendu dans d'excellentes conditions (sans doute plus difficiles à identifier). Il passe donc d'un monde où la qualité était validée de façon binaire (le test a réussi ou échoué) à un monde probabiliste dans lequel il doit faire usage de calculs statistiques pour comprendre ce que font les utilisateurs et pourquoi ils le font, en établissant des correspondances entre les données capturées et la qualité du produit.

5.1.5. Évolution du rôle du développeur

Les tests sont inhérents au développement logiciel. Durant les différentes étapes du cycle de production logicielle, de nombreux types de tests doivent s'exécuter. Tester un logiciel va d'abord consister à lancer les composants logiciels ciblés dans un contexte d'exécution permettant de mesurer les résultats obtenus afin de déterminer si les objectifs de ces composants sont atteints. Avant DevOps, le développeur était principalement concerné par les tests unitaires et les tests d'intégration. Avec DevOps, le développeur est directement responsabilisé sur la qualité du code produit. Il ne se limite plus à de simples tests unitaires. DevOps diversifie les types de tests, en impliquant le développeur sur des tests ayant une portée plus globale pour l'acceptation du code. Dans ce deuxième cas, il peut s'agir non seulement de valider le comportement fonctionnel de l'application, mais aussi ses temps de réponse. Le développeur DevOps doit donc implémenter et instrumenter ces tests en prenant en compte ces contraintes. Comme nous l'avons vu, il est déjà en situation de pouvoir vérifier le bon fonctionnement de son code sur des environnements semblables à la plate-forme de production. Il dispose en outre de beaucoup plus d'informations sur la façon dont le code a fonctionné (ou non) en production.

Compte tenu de son niveau d'interaction plus fort avec le métier, le développeur peut d'ailleurs être amené à intégrer les exigences fonctionnelles dans le développement même des composants logiciels concernés (*Test Driven Developement*). Le développeur doit rester responsable de sa méthode de conception et s'assurer du fait que cette démarche se

traduit globalement par un réel gain de productivité. Le développeur DevOps qui livre un composant logiciel, doit prendre en considération toutes les contraintes techniques auxquelles ce composant est susceptible d'être soumis : consommation de mémoire, de bande passante, performance, sécurité, instanciations multiples sur la même machine, sur de multiples machines… Il doit donc être capable d'implémenter les tests permettant de valider le respect de ces contraintes. Il participe notamment au développement de tests de charge permettant de simuler de très nombreux utilisateurs afin de vérifier les performances de l'application sous différents niveaux de stress. Dans certains contextes, la validation effective de la charge ne peut se faire qu'en production. C'est d'ailleurs au cours des périodes de pic de charge qu'il convient de mettre à disposition (en mode contrôlé et non généralisé) de nouvelles fonctions, afin d'obtenir le maximum d'information sur leur usage.

5.1.6. Évolution du rôle des équipes de gestion opérationnelle

Des environnements de déploiement dédiés sont souvent nécessaires pour l'intégration fonctionnelle et la mise en œuvre de tests de performances pertinents. Le cloud offre une réponse particulièrement adaptée à ce type d'usage, avec sa capacité à cloner et déployer des configurations fournies à la demande et facturées à l'usage. Et là encore la composante *Infrastructure as Code* de DevOps joue un rôle déterminant par sa capacité à automatiser la mise à disposition de ces environnements de tests. Les équipes de gestion opérationnelle sont alors en mesure de développer et faire évoluer ces scripts d'automatisation, qu'ils ciblent des environnements Linux ou Windows. De plus, ces équipes sont directement impliquées dans les tests et concernées par le suivi de l'application en production. À ce titre elles sont les garantes de la capacité à remédier à un dysfonctionnement dans les plus brefs délais.

5.1.7. Évolution du rôle des experts en sécurité

Les experts en sécurité ont de solides connaissances en réseau, en système d'exploitation, et dans différents langages en fonction des types d'environnement dont ils doivent assurer la protection. Avec DevOps, ils sont maintenant directement intégrés dans les équipes de développement et de gestion opérationnelle. Leur rôle est de transmettre le savoir et déléguer certaines de leurs responsabilités à l'ensemble des acteurs de la chaîne de production logicielle, afin de faire en sorte que chacun puisse avoir une connaissance des menaces auxquelles sont exposées les ressources et des moyens à mettre en œuvre pour les préserver.

Pour ce faire, les experts en sécurité doivent eux aussi comprendre les processus de développement et de gestion opérationnelle. Leur mission est de s'assurer que ces derniers puissent être automatisés en incluant un paramètre supplémentaire : la sécurité. Le système devient ainsi plus réactif à d'éventuelles attaques, car le déploiement de plateformes intègre des mécanismes de protection sur des cycles itératifs, plutôt qu'au terme d'une longue implémentation.

D’un point de vue culturel, cela suppose un engagement de la part de chacun, une active communication entre les différents acteurs, une capacité à s’adapter au changement, et la mise en avant de préceptes déjà évoqués tel que l’expérimentation, les architectures *failsafe* ou le principe de simplicité…

5.2. Le sens de la mesure…

La confiance que suppose la démarche DevOps n’exclut pas le contrôle. Pour garantir un certain niveau de qualité, il faut mettre en place outils et mesures permettant d’assurer un suivi constant de la solution sur de multiples critères. Il s’agit donc de définir les métriques qui vont permettre d’identifier les pistes d’amélioration dans l’évolution du produit, dans sa fiabilité, dans la disponibilité du service qui l’expose et dans la chaîne de production logicielle. C’est également sur la base de ces métriques que pourront se décider les grandes orientations à donner d’un point de vue fonctionnel et organisationnel.

5.2.1. DevOps et mesures de la qualité

La production d’une solution logicielle impose parfois des choix qui ne plaisent pas à tous les acteurs du système. Disposer de paramètres quantifiables permet de réaliser ces choix de façon plus objective et de les faire plus facilement accepter. En outre, quantifier la qualité et l’avancement de la réalisation d’une application permet de faciliter l’établissement d’une relation de confiance entre les acteurs de la chaîne de production, prérequis indispensable à la mise en œuvre d’une démarche DevOps.

Comment choisir ses métriques

Il n’est pas toujours aisé de choisir les bonnes métriques. Une approche efficace peut consister à appliquer une démarche top-down de type GQM (*Goal*/*Question*/*Metric*) en six étapes. En premier lieu, il s’agit de se fixer un certain nombre d’objectifs prioritaires (du point de vue business, du point de vue du client), puis d’énoncer les questions qui permettront de savoir si cet objectif est atteint en termes quantifiables, enfin de définir un certain nombre de métriques qui apporteront une réponse à ces questions.

Par exemple, si l’un des objectifs est d’améliorer la disponibilité d’un service web, l’une des questions sera : « *quels sont les indicateurs qui permettent de mesurer la disponibilité ?* » Les métriques à considérer pourraient être le Average Response Time, le Http Server Errors, le Http Success, le CPU Time, le Memory Working Set. À cela s’ajoute la définition des mécanismes pour la collecte de données, leur capture et leur analyse. Ce type d’approches présente en outre l’avantage d’être facile à partager avec le management et d’éviter de s’intéresser à des métriques qui au final s’avéreraient inutiles.

Typologies de mesures

Suivant l’objectif ou la perspective considérée les mesures sont de différents types. D’un point de vue organisationnel, c’est le *time to market* qui prévaut. Il correspond à différentes métriques telles que le *lead time* ou le *cycle time*. Les métriques liées à la chaîne de production logicielle permettent d’évaluer sa capacité à s’adapter : taux

d’innovation, bug, coût de production. Les métriques liées à la qualité de la solution proprement dite permettent de mesurer la disponibilité du service (temps de fonctionnement…), sa performance (temps de chargement page, latence…) et son utilisation (comportement et fidélité de l’utilisateur…). Ces informations sont obtenues par l’analyse des données issues de la télémétrie de l’application complétée par celle des logs du système.

Des métriques sur les métriques…

Les métriques doivent être partagées et régulièrement revues. Il est pertinent de superviser leur évolution et leur adéquation par rapport à des objectifs qui eux aussi peuvent varier dans le temps. Leur utilisation en tant qu’outil d’évaluation de la performance des acteurs de la chaîne de production logicielle n’est pas à encourager, surtout dans une optique DevOps. Il est préférable d’éviter toute tentation d’altération de ces métriques. Dès lors qu’un individu sait qu’il est *mesuré* il modifie son comportement pour optimiser les critères d’évaluation plutôt que la solution elle-même (*Hawthorne Effect*). Enfin, la collection de ces métriques doit être totalement automatisée, transparente et partagée entre les différents acteurs du système d’information, afin que chacun puisse prendre connaissance de ces éléments et agir en conséquence.

5.2.2. Mesurer la qualité pour l’organisation

L’organisation doit disposer de différentes métriques pour définir sa stratégie.

Lead time

Le *lead time* est le délai de la déclaration d’une tâche d’implémentation dans le backlog jusqu’à sa mise en production. Il permet de savoir combien de temps est nécessaire avant de pouvoir offrir un nouveau service. Un délai de livraison plus court permet d’obtenir des résultats plus rapidement et augmente la valeur du service ou du produit. Ces mesures peuvent également exposer des inefficacités (temps d’inactivité) dans le process, ce qui d’un point de vue organisationnel est à proscrire.

Work-in-progress (WIP)

Work-In-Progress (WIP) définit le nombre de travaux démarrés mais non achevés. Idéalement, cet indicateur devrait être relativement constant, avec un équilibre entre les nouveaux traitements de demandes et ceux qui s’achèvent. Cela permet d’offrir plus de prédictibilité sur la durée. À l’inverse, si cet indicateur augmente l’organisation est en droit de s’inquiéter, car cela signifie plus de changements de contexte pour les différents acteurs du système d’information et potentiellement plus de dépendances entre les éléments constitutifs de la solution.

Autres indicateurs

À cela s’ajoute la fréquence de livraison, le délai de stabilisation d’une release, le *cycle time* : la durée nécessaire pour implémenter une fonction ou compléter une tâche (le temps est mesuré entre le début de la réalisation de la tâche et son achèvement).

5.2.3. Mesures de la qualité sur la chaîne de production logicielle

Il existe de multiples métriques liées à l'évaluation d'une chaîne de production logicielle.

Des métriques fonctions de la méthode

Suivant la démarche ou la méthode appliquée, le nombre de métriques peut être très significatif. Par exemple, CMMI appliqué au développement est très consommateur de métriques. L'approche DevOps sera plutôt alignée sur celle de l'agile. L'objectif n'est pas de disposer des preuves de la mise en application de l'ensemble des processus requis par l'ingénierie, mais de se doter des moyens permettant d'évaluer le niveau de qualité d'un sprint et des axes d'optimisation pour les suivants.

Mesure de la vélocité de développement

Il s'agit de pouvoir mesurer l'évolution de la vélocité d'un sprint à l'autre sur les phases de développement liées à la production du service. Les métriques associées au design et au développement correspondent à ce qui peut être mesuré entre la phase de définition du backlog et le déclenchement du build suite à l'archivage du code dans le référentiel de source. Elles correspondent à différents indicateurs comme l'évolution du backlog, le nombre de bugs actifs, le *code churn* (le nombre de lignes modifiées depuis le dernier build), le *code coverage* (le pourcentage du code couvert par des tests) ou le statut des tests.

L'un des intérêts de collecter ces métriques est de pouvoir les analyser conjointement. Supposons par exemple que le nombre de bugs augmente en proportion de l'évolution du code churn. Cette information est déjà intéressante en soi, mais peut-être faut-il la corréler avec le code coverage. Peut-être est-ce aussi l'occasion d'instrumenter les éléments de code concernés pour essayer de déterminer les conditions qui les prédisposent à une défaillance…

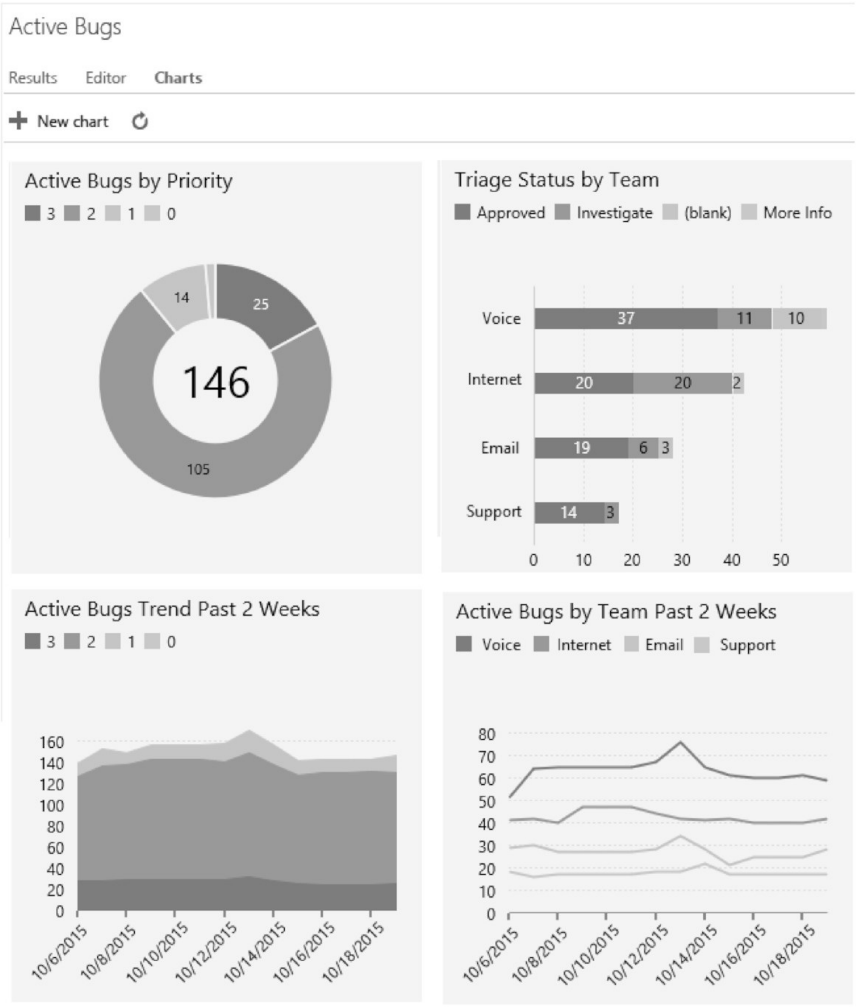


Fig. 5.1 *Suivi de l'évolution du nombre de bugs actifs*

Mesure de la vélocité sur les phases opérationnelles

Les métriques liées à la gestion opérationnelle sont concentrées sur le déploiement vers les différents environnements. Un indicateur très intéressant est le *Mean Time To Deployment* (MTTD) qui correspond au temps requis pour déployer une nouvelle version en production du service. Dans un monde où l'on est prêt à plus facilement pardonner une défaillance en production, ce qu'il convient de privilégier, c'est la réduction du temps de détection (*Mean Time To Detection)* et de remise en service *(*MTTR : *Mean Time To Recovery*). Ceci étant dit, ne pas avoir un *Mean Time To Failure* peut rester un objectif en soi…

Mesure de la vélocité globale du projet

Par essence, le développement est moins prédictible que la gestion opérationnelle qui, par l'automatisation, peut réduire la variabilité de leur processus. De même, les processus de conception et de développement sont souvent plus longs que ceux de mise en production. Au final, dans un monde de services, la qualité première d'un logiciel reste sa capacité d'adaptation aux demandes du marché : *taux d'innovation, bug, coût de production* sont donc les mesures à considérer en priorité. Elles sont obtenues directement sur la chaîne de production, par analyse des données issues de l'intégration du système de build avec le contrôle de code source et le système de gestion des releases.

Dette technique

La dette technique est due à un non-respect des règles établies en phase de conception (patterns, framework, règles de codage…) ou à des mauvais choix de conception à l'origine. Elle peut être involontaire (manque de maîtrise, absence de communication et d'outils permettant de garantir le respect des règles) ou volontaire (choix pragmatique pour répondre à des soucis de délais de livraison). Dans un cas comme dans l'autre, il faudra, tôt ou tard, s'acquitter de cette dette et la rembourser avec des temps de développement de plus en plus longs et de bugs de plus en plus fréquents. Elle est mesurée en fin de sprint, en comptabilisant les problèmes rencontrés, les bugs, mais aussi les dérives du code par rapport aux règles définies au lancement. D'où l'intérêt de se familiariser avec des méthodes génériques d'évaluation du code comme SQALE (*Software Quality Assessment based on Lifecycle Expectations*) et avec des outils d'analyse de la qualité du code comme la solution open source SonarQube.

SQALE permet aussi d'aider à prioriser les efforts de remédiation de la dette technique. Toutefois, dans une approche plus agile, il est aussi possible de considérer l'alternative suivante dite de la *fuite d'eau* : lorsque vous avez une fuite d'eau, vous commencez par la stopper, avant d'essuyer le sol. Appliquée à la dette technique, cela devient :

- Prendre acte que vous avez de la dette technique et prendre une *baseline* pour ne plus la considérer. À partir de maintenant ne plus laisser de dette technique s'accumuler.
- Au fur et à mesure que vous fixez des bugs ou codez de nouvelles fonctionnalités, vous pouvez aussi corriger la dette technique existante dans les méthodes que vous modifiez (vous êtes couverts par les tests unitaires).

5.2.4. Mesures de la qualité de la solution

La **qualité du service** est mesurée par la collecte de données sur les composantes de ce service et permet d'évaluer sa disponibilité, ses performances, son évolutivité et sa fiabilité.

La **disponibilité** du service est mesurée en pourcentage de temps d'activité et peut faire l'objet d'un SLA. Elle peut être impactée par des soucis liés à l'infrastructure, à des bugs applicatifs, à un dimensionnement insuffisant.

La **performance** est la capacité du service à exécuter une action dans un laps de temps acceptable. Cela inclut la mesure et le suivi de l'évolution temporelle de la consommation de ressources (CPU, mémoire, réseau, I/O disque…) sur chacun des serveurs de l'infrastructure cible, ainsi que sur les middlewares (nombre de messages déposés dans une file d'attente, nombre de requêtes par seconde, temps de réponse des serveurs web, latence sur les appels d'API…) ou sur les bases de données (nombre de transactions par seconde…).

L'**évolutivité** est la capacité du service à gérer une charge plus conséquente sans interruption de service ni impact sur les performances.

La **fiabilité** d'un service est sa capacité à ne pas rencontrer de dysfonctionnements. La difficulté est de faire le lien entre les données liées à un plantage de l'application en production (erreurs dans le journal des évènements, état de la pile d'appels) et les données d'utilisation (l'état dans lequel était l'application lorsque le problème s'est produit). D'où l'intérêt pour le testeur DevOps de mettre en place le niveau d'instrumentation permettant d'établir cette corrélation ou de s'appuyer sur des technologies comme IntelliTrace sur des environnements .NET.

5.2.5. Mesures de l'usage d'une application

En définitive, la qualité est avant tout le résultat d'un ensemble de caractéristiques laissées à la libre appréciation de l'utilisateur. Certes, offrir un excellent niveau de fiabilité du logiciel avec une très bonne disponibilité du service est fondamental, mais proposer une solution exempte de tout défaut qui ne serait utilisée par personne ne présenterait aucun intérêt. Parmi les critères à retenir, l'un des plus importants reste donc doute celui de l'adéquation entre les fonctions proposées par ce produit ou ce service et les attentes de l'utilisateur. Sont-elles effectivement appelées ? Quelle valeur ajoutée apportent-elles ? La solution offre-t-elle globalement un niveau de finition (performance, ergonomie et design de l'interface) aligné sur le niveau d'exigence de l'utilisateur ? Autant de questions auxquelles la capture de **données d'utilisation** peut apporter un premier niveau de réponse.

En effet, les données d'utilisation d'une application sont aujourd'hui un élément clé dans la détermination de la qualité du service. Afin de savoir comment un service est réellement utilisé, il est nécessaire d'obtenir de grandes quantités de données et de pouvoir en analyser certaines en temps réel. La télémétrie permet de savoir comment les clients interagissent avec l'application. Mais pour garantir la bonne utilisation d'un service, il faut étendre la capture de données utilisateurs à d'autres sources comme par exemple les logs du support en charge du service, ou les feedbacks remontés par les utilisateurs sur des canaux internes ou externes (réseaux sociaux…). L'analyse de ce type de données avec un fort potentiel de volume et de vélocité suppose l'utilisation de technologies liées au monde du big data.

La collecte des données sur l'utilisation que l'utilisateur fait du service doit respecter sa vie privée, en particulier s'il s'agit de données sensibles comme le numéro de carte bleue ; le développeur ou le testeur DevOps devront donc veiller à ne pas archiver ce type de données dans les logs et à rendre anonymes celles qui concernent directement l'utilisateur.

5.3. La qualité au service du cycle de développement logiciel

Du fait de l'évolution du marché et de la mise à disposition de nouvelles versions d'un service ou d'un produit dans des temps plus réduits, il devient nécessaire d'adopter une approche plus pragmatique pour les tests. Il faut éviter de gaspiller des ressources, en réalisant, documentant et en exécutant des tests inutiles ou qui ne tiennent pas compte des nouvelles possibilités qu'offrent la capture et l'analyse de flux d'information.

5.3.1. Lean Testing

Le concept du *Minimum Viable Product* issu de la méthode Lean Startup s'applique aux équipes qualité DevOps. Selon ce principe, il s'agit de livrer un logiciel avec un niveau de qualité suffisant et de nouvelles fonctions qui permettent de livrer au plus tôt pour fidéliser les utilisateurs existants et en attirer de nouveaux… Ce niveau de qualité *suffisant* associé à une prise de risque contrôlée correspond à une évolution de la démarche de garantie de la qualité par les tests. Il s'agit donc d'éviter de trop tester, car non seulement cela a un coût mais cela traduit aussi par un accroissement des délais de mise à disposition des services ou des produits. On parle alors de *Lean Testing* : tester moins, mais de livrer plus souvent des évolutions sur des périmètres plus réduits et de garder le contrôle grâce à la capture en continu de données issues de la production. Au final, la qualité reste plus maîtrisée.

5.3.2. Gestion des risques par la catégorisation des utilisateurs

Comme nous l'avons vu, garantir la qualité consiste à gérer des risques. Une approche équilibrée consiste à définir des catégories d'utilisateurs avec des populations susceptibles d'avoir un niveau variable d'acceptation d'une défaillance provisoire et de capitaliser sur les possibilités qu'offre l'automatisation de l'infrastructure pour procéder à d'éventuels retours arrière.

Ainsi, chez Microsoft le premier cercle des testeurs est l'équipe en charge du développement du produit ou du service. Ils acceptent plus facilement la présence de bugs, et sont aux premières loges pour les corriger. De même, on y pratique depuis de nombreuses années le *dogfooding*, approche qui consiste à valider le bon fonctionnement des dernières versions des logicielles en les mettant en production sur l'informatique interne… Les bugs sont acceptés, tant qu'ils ne sont pas préjudiciables à l'activité de l'entreprise. Enfin, le programme Windows Insider, qui a accompagné (et continue d'accompagner) le développement de la dernière version du système d'exploitation de Microsoft, a impliqué 4,1 millions d'utilisateurs dans plus d'une centaine de pays. Une population qui est prête à souffrir de défaillances prévisibles d'une pré-version, mais qui les accepte afin de pouvoir faire partie des *early adopters* et de pouvoir contribuer à l'évolution fonctionnelle du produit. On retrouve la même approche sur les choix d'interfaces proposées aux utilisateurs de Facebook ou sur les expériences que Google mène sur son portail de recherche.

Lorsqu'elle est correctement implémentée, cette approche permet d'organiser plusieurs typologies d'utilisateur avec différents niveaux d'acceptation du risque. En général, plus le nombre d'utilisateurs augmente, plus le niveau de risque diminue. Et en définitive, pour le consommateur, la tolérance à l'échec est très faible : il faut offrir aux clients un niveau de qualité en adéquation avec leurs attentes, sous réserve de les perdre à jamais.

5.3.3. Data Driven Engineering

Avec l’évolution du métier du testeur logiciel, le *Data Driven Engineering* a remplacé le *Software Testing*.

Prévenir plutôt que guérir

Plutôt que détecter les bugs *a posteriori*, l’approche DevOps consiste à essayer de les anticiper.

Ainsi Alan Page, architecte au sein du groupe Microsoft Engineering Excellence et responsable de la mise en œuvre des tests logiciels, raconte comment, en constatant la fréquente occurrence de bugs liés à une mauvaise utilisation d’une librairie, il avait écrit un code permettant de détecter cette erreur. Il l’avait ensuite lancé sur le référentiel de code source pour permettre la correction de l’existant et l’avait ajouté à la liste des vérifications à effectuer par le développeur avant chaque archivage de code, prévenant ainsi de multiples bugs. Cette démarche est applicable dès lors qu’ont été établis un modèle d’erreur reproductible et le correctif de code associé. En général, ces risques d’erreurs récurrentes sont gérés lors des revues de code.

Une démarche outillée permet d’aller plus loin en pratiquant une analyse prédictive de la qualité. L’objectif est d’utiliser l’information extraite de l’analyse des données pour anticiper les modèles de comportement des utilisateurs et de fonctionnement du système en utilisant les techniques du machine learning.

Une nouvelle stratégie d’ingénierie

Cette stratégie correspond à un type d’expérimentation en ligne permettant de piloter un projet par les données. Elle consiste à utiliser de multiples sources de données, y compris de production, pour permettre une action rapide destinée à améliorer la qualité des produits. Les données de production permettent de vérifier la qualité réelle de l’application. Elle suppose la mise en œuvre d’un cycle itératif organisé sur différentes étapes de définition des KPI, de conception pour la collecte des données de qualité de production, de sélection des sources de données, d’analyse des résultats, et d’apprentissage. Les données de production sont issues de télémétrie sur la navigation des utilisateurs mais aussi d’informations directement générées par le système.

Le challenge, dans une perspective DevOps, sera de s’intégrer dans la globalité d’un système qui manipule également des données issues de la production et capturées en temps réel. Une première approche peut consister à injecter ses propres tests dans le système, de sorte qu’ils puissent aider à affiner l’exploration de pistes de remédiation ou d’optimisation. Il s’agit donc d’étendre les données d’instrumentation avec des données purement liées aux tests. L’approche DevOps consiste alors à émettre des hypothèses et à les instrumenter de façon à les confirmer ou les infirmer. Une autre démarche consiste à se plonger dans les données capturées et à utiliser des connaissances statistiques complétées par des techniques et des outils liés au big data pour rechercher des corrélations et identifier des modèles.

Le big data au service du développement logiciel

Même si un service ne génère pas de gros volume de données, son utilisation à l'échelle d'internet le rend potentiellement éligible à l'emploi de procédés liés au monde du big data afin de pouvoir mesurer sa qualité.

Les données brutes ne présentent pas nécessairement d'intérêt. Il faut les transformer et les analyser pour y découvrir des modèles de corrélation, y valider des hypothèses sur le fonctionnement du système et de nouvelles questions à se poser sur l'itération suivante. C'est le volume de ces données, leur variété et leur vélocité qui vont permettre ou non d'obtenir des réponses à ces questions. Et ce sont les outils et les solutions retenues qui permettront de les traiter et d'acquérir ainsi une meilleure compréhension des comportements des utilisateurs.

Il existe de multiples technologies ciblant le *realtime distributed processing* (comme la solution open source Apache Spark qui offre un traitement parallèle de données pour les applications analytiques du big data) ou le *complex event processing* (comme la solution open source Apache Storm qui est un système de calcul distribué et résilient pour le traitement de données en temps réel, inclus dans Hadoop). Ces solutions peuvent s'exécuter dans le contexte d'un service managé comme HDInsight Service.

Dans le même ordre d'idée existent également des solutions propriétaires comme le couple Event Hub et Azure Stream Analytics, qui ciblent la comparaison de plusieurs flux en temps réel. Ces solutions que l'on rencontre souvent dans le monde de l'IoT (*Internet of Things*) sont totalement indépendantes du contexte de la supervision. Toutefois, elles sont naturellement conçues pour remonter et analyser un grand nombre d'évènements et peuvent fort bien être adaptées pour mettre en place une remontée d'informations liées à la plateforme sur laquelle s'exécute une application et à leur analyse. Par exemple, elles peuvent être utilisées pour la détection d'anomalies et offrir ainsi la possibilité de déclencher une alerte lorsqu'une erreur ou une condition spécifique apparaissent.

En complément de ces solutions *génériques*, il existe des frameworks capables d'analyser un flux de données en temps réel, qui ont pour objectif de répondre spécifiquement à une problématique de type supervision. Par exemple, la solution open source Riemann, implémentée sur Riak (un système distribué NoSQL orienté documents, évolutif, hautes performances et sans schéma) permet de disposer de retours d'information avec une très faible latence permettant la notification d'un dysfonctionnement dans des délais de l'ordre de la milliseconde (et être notifié ultérieurement du résultat effectif de la correction). Elle repose sur l'utilisation d'agents capables de relayer les évènements à très haute fréquence. Le débit est ensuite impacté par la durée de traitement du flux, mais la solution est parallélisée pour traiter un grand volume de messages à très faible latence. Riemann agrège dans un flux de données les événements qui se produisent sur les serveurs et dans les applications grâce à un langage de transformation. Par exemple, le code suivant issu du site http://riemann.io nous montre comment combiner des flux de données en définissant un flux pour le taux total, baptisé ici aggregate :

```
(let [aggregate (rate 5 (with :service "req rate" index))]
 ; Return a stream which splits up events based on their service
 (splitp = service
```

```
; HTTP requests pass straight to the aggregate stream
“http req rate” aggregate
; But well double the metrics for 0mq requests
“0mq req rate”  (scale 2 aggregate)))
```

Ce langage permet de gérer des évènements qui se matérialisent comme des structures de données véhiculées *via* des *Protocol Buffers* (déclenchement d’alerte sur une exception, suivi de la latence de l’application web, caractéristiques sur chacun des serveurs…).

Une transformation DevOps

Cette évolution nécessite une profonde transformation qui s’inscrit totalement dans une démarche DevOps. Les possibilités qu’offrent DevOps en termes de supervision et de déploiement favorisent la mise en place d’une stratégie d’ingénierie pilotée par les données. Une fois détectée et analysée la cause du dysfonctionnement, les évolutions ou les correctifs (ou les mesures d’atténuation) peuvent être déployés sur le service, l’ensemble de ces opérations s’effectuant dans un cycle de détection continu. De plus, la transformation DevOps pour la qualité nécessite un accompagnement (formation, outils, processus…) au niveau des équipes, que ce soient les testeurs, les développeurs ou les opérations. Il s’agit en effet de définir les métriques et indicateurs clés, le processus de traitement de ces données, les référentiels du stockage qui leur est alloué, les outils d’analyse, historiques ou prédictifs pour les exploiter.

Les acteurs du système d’information doivent donc progresser en connaissance sur les domaines du big data, voire du machine learning et étendre leur périmètre de vision sur la globalité du système. Au-delà de l’évolution des compétences des équipes, c’est l’occasion de voir émerger de nouveaux rôles au sein des équipes intervenant sur le cycle de production logicielle : les *data scientists* dont le rôle est aussi d’accélérer la progression de leurs collègues sur l’utilisation de techniques issues du big data, afin que chacun soit en mesure de s’inscrire dans un cycle d’ingénierie pilotée par les données. Cela suppose également un changement de culture au sein de l’organisation pour qu’elle accepte l’expérimentation en ligne et le pilotage de la qualité par les données. Enfin, et surtout, pour que cette évolution soit possible, il faut assurer une intense collaboration entre les différentes équipes, avec un partage des connaissances, des outils, des processus et des environnements. DevOps est donc très clairement la démarche appropriée pour évoluer vers cette Data Driven Quality.

5.4. Le cycle de développement logiciel au service de la qualité

L’une des façons d’offrir plus de qualité consiste à décomposer le projet en plusieurs phases plus courtes afin de gérer les priorités et d’anticiper les tests et les correctifs… La modularité favorise les tests, mais il est important de tester chacun des services sans présupposer du bon fonctionnement des autres. Un des moyens qu’offre DevOps pour aller vers plus de qualité est de réaliser les tests dans le prolongement du développement, avec un déclenchement dans le cadre d’un processus d’intégration continue qui permet

aux développeurs, qui archivent fréquemment leur code d'obtenir très rapidement un premier niveau d'information sur la qualité de leur code.

5.4.1. Continuous delivery

Livrer plus fréquemment réduit les risques car la quantité et le niveau de complexité des changements sont plus réduits et donc plus faciles à contrôler. De plus, l'itération régulière d'un processus facilite sa maîtrise par l'ensemble des participants. En outre, l'automatisation de l'ensemble de la chaîne autorise les retours arrière. Il en résulte une augmentation du niveau de qualité.

Le *continuous delivery* permet également d'obtenir plus rapidement des données sur le code de fonctionnement en production avec des utilisateurs réels. Cependant le niveau de qualité doit être suffisant pour garantir une utilisation de l'application, prérequis indispensable à la capture des données permettant de valider le bon fonctionnement du code…

D'un point de vue qualité, la gestion des releases peut être mise en œuvre selon de multiples stratégies. Plutôt que de générer à chaque nouvelle version, un package complet, une première démarche dite de *release partielle* peut consister à ne livrer que les composants modifiés, ce qui suppose une architecture fortement découplée. L'empreinte de la mise à jour étant nativement plus réduite, le risque de défaillance est plus réduit.

Une autre approche baptisée *Toggle Features* consiste à déployer les nouvelles versions des services en y intégrant les évolutions mais en s'assurant que les conditions de vérification sont validées avant de les activer par configuration. En cas d'erreur, il devient alors très simple de faire machine arrière.

Une dernière alternative consiste à exploiter la catégorisation des utilisateurs proposée par le modèle de Lean Testing en déployant progressivement vers des groupes qui acceptent un niveau de risque de défaillance du service plus ou moins élevé. Chez Netflix, cette stratégie a fait l'objet de l'appellation ***Canary Releases***, par analogie avec le système jadis mis en place par les mineurs pour se protéger de coups de grisou, gaz inodore dont la présence ne pouvait être détectée que par l'arrêt du chant (voire la mort prématurée) du canari.

5.4.2. Continuous learning

Le *continuous learning* est l'une des valeurs fondamentales de DevOps. Il s'agit pour chacun d'étendre ses connaissances et ses compétences dans une démarche d'apprentissage permanent pour répondre à un environnement en constante évolution. Cela passe par l'observation, l'expérimentation, le partage d'expérience, la prise de risque, l'application de modèles éprouvés et la formation. Cette démarche doit être personnelle et organisationnelle afin de permettre aux équipes d'être à même de s'adapter au changement du marché et des technologies.

D'un point de vue qualité, l'apprentissage en continu va concerner les différents moyens qui permettront d'orienter la stratégie produit et d'améliorer l'application.

La connaissance passe par la donnée

Comme nous l'avons vu dans la définition des métriques, pour prendre de meilleures décisions sur les évolutions et correctifs à apporter à une solution, un ensemble de questions clés doit être déterminé en amont du processus de mise à disposition du produit ou du service. Il s'agit ensuite d'analyser les données capturées en production pour extraire des informations, puis d'utiliser les résultats obtenus pour améliorer la qualité du service et l'expérience utilisateur. Ces données véhiculées par télémétrie sont issues du besoin de mieux comprendre comment les clients utilisent l'application. Cette démarche se déroule dans le cadre d'un cycle alimenté en continu par de nouvelles questions faisant suite aux connaissances acquises dans les précédentes itérations.

Tests en production

Malgré la mise en œuvre de tests en charge, de laboratoires de tests, il est souvent impossible de recréer les mêmes conditions qu'en production sur un environnement de pré-production. Les tests en production (TiP) proposent une façon d'apprendre sur la qualité du système en l'exposant directement aux utilisateurs réels sur les environnements de production. L'expérimentation en ligne, en conditions réelles, permet d'obtenir une meilleure connaissance de ce qui correspond ou non aux attentes des utilisateurs, d'avoir une meilleure compréhension de leurs modèles d'utilisation et d'identifier des cas de dysfonctionnement non anticipés. Cela s'inscrit totalement dans la démarche d'ingénierie pilotée par les données que nous avons précédemment décrite.

Les tests en production ne sont pas exempts de risque, mais ce risque peut être atténué en gardant le contrôle sur le niveau de diffusion du service segmenté par audience. Pour qu'une stratégie de tests en production soit couronnée de succès elle doit être associée à la mise en place d'un système de supervision et de gestion des opérations très réactif. Ainsi si la nouvelle version rencontre des dysfonctionnements, un retour arrière peut immédiatement être déclenché.

Les tests en production sont très souvent utilisés dans le monde des services, mais peuvent également être mis en œuvre sur des produits, à condition qu'ils soient connectés. Toutefois, cela suppose des contraintes supplémentaires comme la limitation des ressources de l'environnement sur lequel s'exécute l'application (mémoire, stockage, processeur…). Souvent, cela justifie l'envoi d'un ensemble d'informations issues de l'agrégation de multiples évènements, plutôt que d'une communication de ces données en continu. D'autres critères peuvent également être considérés comme la gestion des données privées ou la sécurité des réseaux.

Un exemple de processus de tests en production est l'A/B Testing. Il s'agit d'un procédé qui permet de savoir quelle est la solution qui répond le mieux aux attentes des utilisateurs en combinant l'analyse d'informations issues de la télémétrie et le déploiement par catégorie d'utilisateur. Chaque déploiement est l'occasion de déployer en production de multiples versions de la même application offrant chacune une interaction et une expérience différente pour l'utilisateur. Le système est couplé avec un mécanisme permettant de router dynamiquement les différentes typologies d'utilisateurs sur le service

ciblé. L'objectif est ensuite de mesurer par analyse statistique quelle version offre le meilleur niveau de satisfaction aux objectifs de la fonction et d'engagement de l'utilisateur (achat, recherche…). Une variante peut consister à déployer une nouvelle version, proposant par exemple une nouvelle interface utilisateur en parallèle de l'ancienne, pour les comparer deux à deux, grâce à la capture de données qualitatives en production. C'est une pratique des plus courantes chez Amazon, Google ou Microsoft.

L'échec comme source d'apprentissage

Enfin, une autre façon d'apprendre est d'échouer. On a beau essayer de prévenir l'échec, tôt ou tard il finit par se produire. L'approche DevOps consiste alors à apprendre et à progresser à partir de ces défaillances. Comme nous l'avons déjà dit, une façon d'accompagner cette démarche va consister à injecter volontairement des défauts dans le système afin de s'assurer que le service est tolérant aux pannes et qu'il sera capable de se reconfigurer (ou d'être reconfiguré) en cas d'interruption de service. De bons exemples de cette démarche sont les initiatives de Netflix (la fameuse Simian Army avec ses Chaos Monkey et Chaos Gorilla) ou d'Amazon (Amazon Game Day).

5.5. DevOps et sécurité

Déjà considérée comme un frein pour les processus IT existants, la sécurité peut devenir un véritable obstacle pour DevOps : avec le *continuous delivery*, les mécanismes traditionnels liés à la mise en production sécurisée d'une nouvelle application ne sont plus applicables. Un changement culturel est donc requis pour intégrer la sécurité au cœur des processus DevOps. Les outils de sécurité existants doivent également être revus ou complétés pour éviter les délais induits par une utilisation trop statique ou ponctuelle.

Depuis quelques années, différentes initiatives participent à cette évolution.

5.5.1. Les initiatives de sécurisation du cycle de vie logiciel

De multiples initiatives portées par des communautés ou des éditeurs recommandent l'inclusion des contrôles de sécurité tout au long des différentes phases du cycle de vie de développement de logiciel.

Open Web Application Security Project (OWASP)

OWASP est une communauté dont l'objectif est d'aider les organisations à devenir plus efficaces dans la protection de leurs systèmes sur Internet en proposant des recommandations (*OWASP Testing Framework, OWASP Developer Guide*) et en mettant régulièrement à jour un inventaire des dix types de menaces liées à la sécurité les plus critiques pour les applications web : injection, violation de gestion d'authentification et de session, *cross-site scripting* (XSS), références directes non sécurisées à un objet, mauvaise configuration de sécurité, exposition de données sensibles, manque de contrôle d'accès au niveau fonctionnel, falsification de requête intersites (CSRF, *Cross-Site Request Forgery*), utilisation de composants avec des vulnérabilités connues, redirection et renvois non validés.

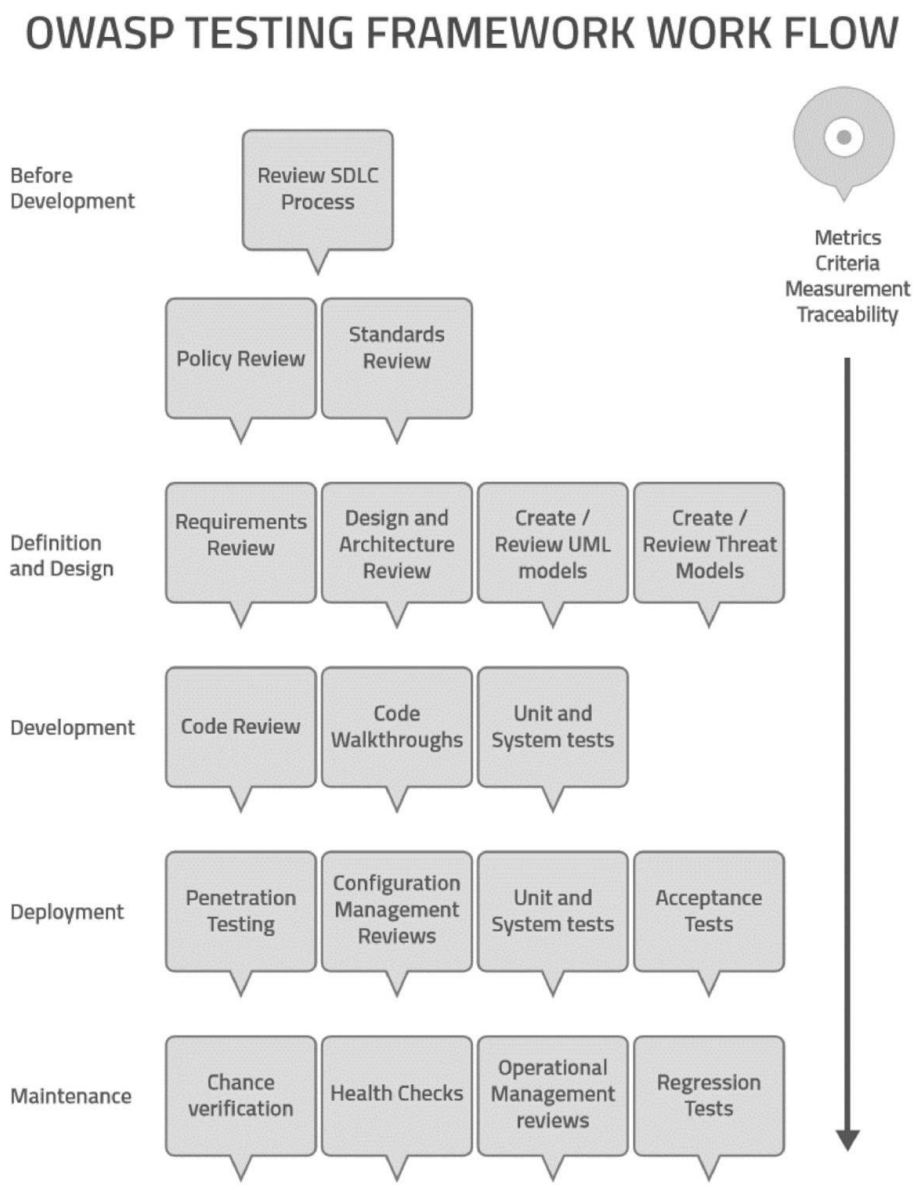


Fig. 5.2 *Extrait du document OWASP Testing Guide (version 4/0)*

Microsoft SDL

Microsoft SDL (*Security Development Lifecycle*) est un processus d'assurance qualité centré sur le développement de logiciel. Depuis 2004, SDL est appliqué sur chacun des produits, technologies et services Microsoft. La formation et la responsabilisation des acteurs du cycle de vie logiciel sont au cœur de cette démarche : chacun doit avoir une compréhension des causes et des effets des vulnérabilités en sécurité, dans un environnement dans lequel les typologies de menaces sont en constante évolution. Pour permettre aux organisations de réagir le plus efficacement face à ces changements, SDL recommande la mise en œuvre de revues régulières du processus en vue d'une amélioration en continu, notamment par la collecte de métriques pendant son déroulement. Enfin, SDL inclut un plan de réponse aux incidents et impose l'archivage de l'ensemble des données liées à leur résolution.

Rugged Software Development

L'objectif de cette initiative, lancée en 2009 est de créer des applications hautement

disponibles, tolérantes aux pannes, sécurisées et résilientes, en anticipant les risques de menaces et en créant les défenses appropriées. Cette démarche consiste à instrumenter le code, expérimenter en continu pour vérifier s'il existe des failles et aboutir à un code vraiment robuste.

Rugged DevOps

Rugged DevOps lancée lors de conférence RSA en 2012, par Gene Kim, CTO et créateur de Tripwire, avec Joshua Corman, directeur de la sécurité pour Akamai Technologies, adapte les principes du *Rugged Software Development* à une gestion sécurisée de l'infrastructure, du développement et des opérations. La démarche consiste à intégrer la sécurité à chaque étape du cycle de vie logiciel dans l'architecture et le design du projet, dans les pratiques et outils opérationnels associés au pipeline de déploiement et de réserver aux tâches liées à la sécurité une priorité plus élevée dans le backlog.

5.5.2. Sécurisation des développements

La sécurisation du cycle de vie logiciel suppose la prise en compte de principes de sécurité, dès la phase de conception.

Conception

La sécurité est intégrée dans la définition des besoins et de l'architecture, avec un focus particulier sur le modèle de gestion d'identité appliqué aux utilisateurs ou aux composantes des applications, et plus généralement sur le chiffrement des données et des échanges.

Le modèle de menaces est défini pendant la phase d'expression des exigences. Il s'agit d'un document qui inventorie les attaques possibles et leur traitement dans le backlog. Les développeurs s'attachent également à réduire la surface d'attaque des ressources les plus exposées vis-à-vis de l'extérieur, ce qui suppose l'identification (et la limitation) des services requis par l'application. Souvent, l'application évolue par rapport aux choix fonctionnels et techniques initiaux. Il convient donc de revoir régulièrement la modélisation des menaces et la définition de la surface d'attaque, afin de pouvoir prendre en compte tout nouveau type de risque de sécurité.

Implémentation

Le code fait régulièrement l'objet de revue pour s'assurer du respect des règles définies en phase de conception. Cette démarche est complétée par l'utilisation d'outils d'analyse de la conformité du code.

Le code est archivé dans un contrôle de code source et peut faire l'objet d'audits (horodatage de la modification, identité de l'utilisateur). Il faut éviter d'y persister tout élément lié à l'authentification (mot de passe, clés d'API, clé privée SSH). Dès lors qu'une donnée sensible a été archivée dans un référentiel de code source, elle doit être considérée comme compromise et doit être renouvelée, même lorsqu'il est possible d'éliminer le fichier de l'historique des versions (par exemple dans le cas de Git, il existe des commandes comme *git filter-branch* ou des outils comme *BFG Repo-Cleaner*).

Déploiement

L'automatisation du déploiement de l'infrastructure (et la validation de sa mise en œuvre sur les différentes plateformes complémentaires à la production) permet de fiabiliser l'infrastructure en supprimant le risque d'erreur humaine. Cependant les scripts de configuration doivent faire l'objet de revue de sécurité et, comme le code de l'application, ne pas contenir d'information sensible. Lorsqu'ils requièrent des informations d'identification, celles-ci doivent être chiffrées. Par exemple, dans Azure Automation, elles sont chiffrées l'aide d'une clé unique, générée pour chaque compte.

Contrôle de la sécurité

L'instrumentation du code permet de suivre et mesurer le détail de chaque action ayant entraîné tel ou tel comportement défectueux de l'application. Elle garantit la mise à disposition des éléments liés au respect de la conformité et susceptibles d'être soumis à un audit.

Le fonctionnement sécurisé de l'application est contrôlé dès la phase de build qui échoue, si des vulnérabilités ont été détectées.

5.5.3. Détection et identification de vulnérabilité

Assurer la sécurité d'une application et éviter les régressions suppose la capacité à détecter ses vulnérabilités en continu, ce qui suppose le lancement automatisé de tests dont il convient de définir les spécifications.

Behavior Driven Development (BDD)

La démarche BDD consiste à faire usage d'un langage formel (comme Gherkin), compréhensible par chacun et à le rendre exécutable. Description des tests et actions sont alors couplées et directement intégrés dans des processus automatisés. Au langage Gherkin peuvent être associés des solutions open source comme JBehave ou Cucumber pour les compléter par une implémentation des fonctions correspondantes (features) réalisée en association avec des frameworks de tests unitaires.

Ainsi, dans le monde .NET, l'extension Cucumber s'appelle *SpecFlow* et elle peut être directement couplée à NUnit. Autre exemple, dans les environnements Ruby, la solution open source *Gauntlt* (fournie sous forme de gem) offre une infrastructure de tests de sécurité automatisée. Elle complète la formalisation de tests Gherkin avec une combinaison d'outils de sécurité (dont on peut automatiser l'installation avec le script vagrant *Gauntlt Starter Kit*). Gauntlt est livré avec un ensemble de définition d'attaques prédéfinies associées aux outils de sécurité permettant de vérifier le niveau de protection vis-à-vis de ces attaques : comme NMap pour tester quels sont les ports ouverts, SQLmap pour détecter les vulnérabilités aux attaques de type SQL injection, Arachni pour se prémunir contre le *Cross-Script Scripting* (XSS)…

```
@slow @announce
Feature: nmap attacks for example.com
  Background:
```

```
  Given “nmap” is installed
  And the following profile:
   | name          | value          |
   | hostname      | scanme.nmap.org |
   | tcp_ping_ports | 22,25,80,443    |

Scenario: Using tcp syn ping scan and the nmap fast flag
  When I launch an “nmap” attack with:
   ”””
   nmap -F -PS<tcp_ping_ports> <hostname> -oX foo.xml
   ”””
  Then the file “foo.xml” should contain XML:
   | css                                              |
   | ports port[protocol=“tcp”][portid=“80”] state[state=“open”] |
   | ports port[protocol=“tcp”][portid=“22”] state[state=“open”] |
```

Cette approche présente en outre l’intérêt de partager la connaissance sur les exigences de sécurité et de les inclure dans les critères d’acceptation de la solution. Elle suppose la mise à disposition de différents types de tests automatisés.

Tests automatisés

L’inclusion de mécanismes permettant de sécuriser l’application directement dans le pipeline d’intégration et de déploiement continu repose sur la mise en œuvre d’outils et de tests de sécurité automatisés sur l’application en exécution. Le résultat de ces tests est ensuite conservé pour en assurer la traçabilité.

Ils existent de nombreux outils génériques de test permettant de déceler les vulnérabilités (XSS, CSRF…) sur les serveurs cibles, de les classer par criticité, de proposer une assistance à la remédiation et de faciliter le contrôle de conformité, avec des fonctions de scan réseau, d’interception des échanges http, de découverte des pages (spider), d’analyse, modification et jeu des requêtes web… Parmi les solutions les plus connues, on peut citer Nessus (ou son f*ork OpenVas*), w3af (*web application attack and audit framework*), OWASP *Zed Attack Proxy* (ZAP), Burp Intruder…

Mais, comme l’a rappelé Michael Howard lors de la Conférence AppSec OWASP de 2006 : *Tools do not make software secure! They help scale the process and help enforce policy.* Les outils génériques ne suffisent pas à identifier l’ensemble des failles que peut présenter une application. Pour aller plus loin dans la détection des vulnérabilités d’une application, des tests dynamiques spécifiques à l’application peuvent être réalisés. Par exemple, les tests fuzz par injection de données mal formées ou aléatoires sur les interfaces de l’application sont directement liés à ses caractéristiques fonctionnelles et techniques. Dans le cas des applications web, l’implémentation de ce type de tests peut être facilitée par l’utilisation d’outils open source comme Sélénium, WatiR –(Ruby), WatiN (.Net) ou propriétaires (Visual Studio…).

Le principe consiste alors à automatiser ces tests, qu'ils soient génériques ou spécifiques avec des frameworks comme Cucumber, Gauntlt, Behave et à les intégrer dans le processus de build, afin de pouvoir les appliquer le plus fréquemment possible. Enfin, plus généralement, l'ensemble des composants de l'application doivent également faire l'objet de tests permettant de valider leur robustesse avant intégration.

Tests d'intrusion

Une fois l'application déployée, la démarche de détection des vulnérabilités se poursuit par la mise en œuvre de tests d'intrusion (*pentest*). Il s'agit d'une simulation d'attaque ciblant un défaut de code ou de configuration, en vue de proposer un plan d'actions permettant d'évaluer les risques et améliorer la sécurité d'un système. Ces tests sont de différents types.

Test de type blackbox

Le testeur ne dispose d'aucune information. Les tests blackbox débutent donc par une recherche d'informations sur l'entreprise que l'on tente d'infiltrer (ou ses employés) par des techniques non intrusives : réseaux sociaux, moteurs de recherche, outils DNS (nslookup, whois), outils de géolocalisation d'adresse IP (traceroute). À partir de ces premières informations, l'approche consiste à établir une cartographie du système en inventoriant les ressources (*via* des protocoles tels que SNMP, RPC et SMB), en analysant les flux de communication avec un sniffer, en balayant les ports (avec des outils comme Nmap), et en identifiant les services. L'objectif est de tenter de compromettre le système en détectant (avec des outils comme Nexus), voire en exploitant ses vulnérabilités.

Test de type greybox

Le testeur dispose d'un nombre limité d'informations qui lui permet notamment de passer l'étape d'authentification (par exemple couple identifiant - mot de passe). L'objectif d'un test greybox est d'évaluer le niveau de sécurité vis-à-vis d'un simple utilisateur.

Test de type whitebox

Le testeur possède de toutes les informations requises pour lui permettre de rechercher les failles (architecture, code source, identifiants…) directement depuis l'intérieur du système.

Stratégie Assume Breach

Quel que soit le niveau de sécurisation de la chaîne logicielle, des failles peuvent subsister dans le système. D'où l'intérêt de mettre en place une stratégie fondée sur l'existence présumée de ces vulnérabilités. Ainsi, chez Microsoft une équipe (*Red Team*) d'experts en sécurité simule en permanence des attaques sur le réseau, la plateforme et les services pour valider la capacité du cloud Microsoft Azure à résister à ces attaques ou la capacité des équipes à y remédier en suivant un processus de gestion des incidents en cinq étapes (identification, isolation, éradication, restauration, et analyse a posteriori). De même, chez Netflix, l'outil *Security Monkey* est lancé périodiquement pour détecter les vulnérabilités. Nous en reparlerons au chapitre 8.

Réduire le MTTRQ

Il est impossible de pouvoir prévenir toute attaque. Là encore, c’est la réduction du MTTR qui permet d’avoir une infrastructure plus sécurisée en offrant un niveau de réponse plus rapide et des contre-mesures qui n’ajoutent pas de nouvelles vulnérabilités.

En résumé

Les nouveaux impératifs de fréquence accrue de livraison qu’impose le marché, dans un monde de services en perpétuel changement ont un impact très significatif sur les métiers liés à la qualité. Que ce soit l’ingénieur qualité, le testeur logiciel, le développeur ou l’administrateur système, chacun à un rôle à jouer dans cette évolution. Pour l’accompagner, il devient plus que jamais nécessaire de disposer d’outils permettant de mesurer en continu la qualité sur la base de critères organisationnels, techniques et fonctionnels. Le fondement de la qualité devient donc la donnée.

Avec le **Data Driven Engineering**, la qualité joue aujourd’hui un rôle majeur dans la stratégie d’ingénierie, en permettant de directement alimenter et orienter la chaîne de production logicielle avec des données issues de la production. La qualité vient donc enrichir la production logicielle qui dans le même temps va vers toujours plus de qualité, dans un mode de *continuous learning*.

Cela suppose pour l’ensemble des métiers l’acquisition de nouvelles compétences sur les domaines du big data, voire du machine learning et l’émergence d’un nouveau rôle au sein des équipes intervenant sur le cycle de production logicielle : celui du *data scientist*. Cette transformation est favorisée par les possibilités offertes par la mise en œuvre d’une démarche DevOps, tant sur le plan culturel, avec l’expérimentation en ligne et le pilotage de la qualité par les données, que sur le plan de la collaboration entre les différentes équipes.

Enfin, intégrer la sécurité au cœur des processus DevOps requiert la responsabilisation de chacun des acteurs du cycle de production logicielle et une évolution des outils et des processus de l’automatisation de la détection et de l’identification des vulnérabilités.

Chapitre 6

DevOps vu par le management

Pour les managers des équipes, une démarche DevOps peut être vue avec enthousiasme et défiance à la fois. En effet, promouvoir, améliorer et optimiser la collaboration est généralement accueilli favorablement par les responsables qui ont pour mission de rendre leur équipe la plus efficiente possible. Dans le même temps, ces démarches de collaboration imposent des évolutions, voire des transformations pour les organisations comme pour les personnes et comme souvent le changement amène une certaine forme de méfiance.

L'être humain refuse et craint naturellement ce qu'il ne connaît pas. Levons donc le voile sur les impacts des démarches des DevOps pour les managers.

6.1. Adopter le modele Devops

Si nous voulons viser la mise en œuvre d'une démarche de collaboration agile comme le préconise DevOps, il faut comprendre que cela implique des changements et nécessite d'adopter préalablement un certain nombre de pratiques.

Aussi, avant même de parler de collaboration, il est nécessaire que chaque partie prenante marque sa volonté de collaborer et adopte les prérequis nécessaires à la réussite de cette collaboration.

Dans l'objectif ultime et vital de s'aligner toujours mieux et toujours plus vite sur la stratégie globale de l'entreprise, il existe donc des préalables indispensables à mettre en œuvre par tous : être industrialisé, agile et fiable dans sa production, être agile dans ses développements, être prêt à travailler ensemble pour trouver un mode de collaboration consensuel et enfin être en capacité de communiquer avec les métiers et les décideurs business.

DevOps : la culture de la collaboration

Choisir de mettre en œuvre des pratiques DevOps signifie clairement souhaiter améliorer la collaboration et la confiance entre les équipes internes mais aussi avec l'ensemble des parties prenantes de la DSI.

Il n'est pas faux de dire que toute l'entreprise ou presque fait partie de ces parties prenantes. Dès lors, la mise en œuvre de DevOps a un impact sur la qualité de la collaboration de l'ensemble de l'entreprise.

Réussir à mettre en œuvre une collaboration agile au travers de pratiques et d'une organisation adaptée semble donc le modèle d'avenir dans un monde toujours plus collaboratif et réactif. Il est évident que ce modèle est un changement de culture pour l'entreprise mais sa nature agile permet de le mettre en œuvre avec douceur.

Enfin, il faut s'employer à diffuser cette culture à ses partenaires, fournisseurs et de manière générale l'ensemble des parties prenantes externes à votre organisation. Sans qu'il soit nécessaire qu'ils soient aussi matures que vous,

vous collaborez de toute façon avec eux : un alignement et une culture partiellement commune ne pourront que vous être bénéfiques.

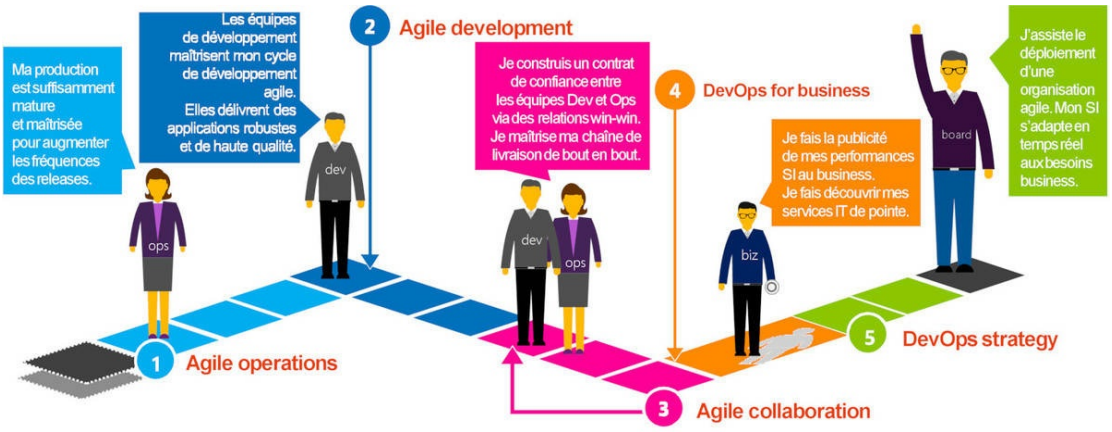


Fig. 6.1 *Le modèle d'adoption de DevOps est une succession de petits pas*

6.1.1. Un modèle de production et d'opérations agile

Concrètement, cela signifie que le monde de la production et des infrastructures doit être en mesure d'absorber une accélération des mises en production logicielles et une amélioration significative de sa réactivité face aux demandes comme face aux incidents.

Pour y parvenir, il n'existe ni recette miracle, ni incantation mystique, ni potion magique. Les *opérations* doivent être suffisamment structurées et industrialisées pour être en capacité d'absorber ce changement d'environnement et dans le même temps éviter toute rigidité qui mettrait à mal cette réactivité. On peut parler ici de la recherche du « *just enough process* ».

Pour y parvenir, il ne semble pas inutile de s'appuyer sur des frameworks de bonnes pratiques tel qu'ITIL ou sur des normes ISO, quand la norme n'est pas directement imposée par les régulateurs du secteur d'activité. Rien de cela n'est incompatible avec l'agilité tant que l'on ne tombe dans l'excès de la rigidité. De même, DevOps étant par nature agnostique aux technologies, un framework orienté technologiquement, y compris pour le cloud ou la mobilité, ne sera pas adapté à cette démarche.

La rigidité non, mais la fiabilité oui, afin de maîtriser suffisamment son patrimoine et les impacts qu'il peut subir tout en cherchant à les anticiper en permanence. Être industrialisé et structuré semble donc être un indispensable. De plus, tout ce qui est figé est voué à mourir et on ne peut absorber le changement et être suffisamment réactif si l'on recherche systématiquement la stabilité.

Il est également possible de s'appuyer sur des technologies de dématérialisation et aller jusqu'à l'adoption du cloud, qu'il soit privé ou public. Ce n'est pas un prérequis, simplement un formidable accélérateur potentiel pour adopter ce changement.

Ce n'est qu'en réussissant à être à la fois proactif et réactif face aux demandes du monde des études et développements, mais surtout face aux demandes des métiers qui leur sont

liés que les *opérations* seront en mesure de réussir à collaborer efficacement au service de toute l'entreprise.

6.1.2. Un modèle d'études et de développements agiles

Si le monde de l'infrastructure et de la production doit être un minimum agile, c'est tout autant le cas du monde des études et du développement. Pourquoi en effet, demander d'accélérer les déploiements pour passer d'une fois par an à plusieurs fois par semaine, si la moindre fonctionnalité nécessite dix-huit mois pour être développée ?

Les démarches DevOps visent à optimiser la collaboration et donc à aligner les modes de fonctionnement sur une temporalité et un rythme de travail commun. Les études et le développement comme les infrastructures et la production doivent donc être suffisamment agiles pour y parvenir.

Aussi, pour les équipes études et développement, il s'agit d'adopter un certain nombre des pratiques préconisées dans les méthodes agiles sans chercher toutefois à coller nécessairement à une méthode particulière et l'appliquer à la lettre.

Il s'agit avant tout d'adopter les valeurs et de faire le tri entre les pratiques proposées dans les principales méthodes agiles pour adopter ce qui semble pertinent dans votre contexte. Traditionnellement, on s'appuie beaucoup sur Scrum pour les pratiques de collaborations, sur XP pour les pratiques d'ingénierie et sur Agile-UP pour les principes de cadrage et de gestion de projet. Je n'hésite pas à compléter mes investigations avec Lean Startup ou Kanban pour la gestion d'anomalies.

Il n'y a pas une seule bonne façon d'être agile mais des centaines de bons choix en fonction de chaque contexte. Il est indispensable d'adopter les pratiques compatibles avec votre contexte pour réussir une démarche DevOps, mais il est impossible de spécifier dans un livre celles qui vous correspondent vraiment.

C'est en vous appuyant sur ces pratiques que vous allez réussir à créer une collaboration efficace tant avec les opérations qu'avec le métier.

6.1.3. Un modèle de collaboration agile pour s'aligner sur le métier

Dès lors que les différentes parties prenantes du monde de l'informatique deviennent mutuellement suffisamment agiles, elles peuvent initier une démarche de collaboration DevOps. Il s'agit alors de chercher à construire une confiance mutuelle et d'introduire les principes d'automatisation pour pérenniser cette confiance.

Une bonne collaboration n'est pas uniquement nécessaire pour assurer un meilleur fonctionnement de la direction des systèmes d'informations globalement, elle est surtout indispensable pour pouvoir s'aligner sur les exigences du métier et des utilisateurs.

Par ailleurs, il faut comprendre qu'il est impossible de s'aligner sur les besoins des utilisateurs et du métier sans chercher à impliquer ces représentants autant et aussi souvent que possible.

Et pourtant, il parait peu probable que les représentants du métier et des utilisateurs s'investissent dans cette démarche, si une tension constante plus ou moins visible existe entre les différentes composantes des équipes informatiques. Et quand bien même, ils accepteraient de collaborer dans un premier temps, malheureusement, cela ne saurait se confirmer dans le temps.

C'est pourquoi il est si important d'initier les mécanismes d'organisation et de collaboration DevOps au sein des équipes informatiques en travaillant les clés de confiance de chacun avant d'impliquer pleinement les équipes métiers pour réussir cet alignement nécessaire sur leurs besoins.

Il est d'ailleurs probable qu'une relation existe déjà avec les équipes métiers. L'implication est alors plus ou moins importante en fonction de la culture de l'organisation ou de la mise en œuvre ou non de méthodes agiles.

La mise en œuvre de DevOps va aider à renforcer et à étendre les relations avec les équipes métiers en s'appuyant sur les échanges existants tout en les faisant évoluer pour intégrer les équipes de production. C'est particulièrement le cas lorsque des méthodes agiles sont pratiquées préalablement à DevOps. Par exemple, le formalisme des scénarios de tests pourra être partagé par tous tout en ne sollicitant les interlocuteurs métiers qu'une seule fois.

6.1.4. Un modèle d'intégration stratégique

Les démarches de collaboration DevOps ne visent pas seulement à améliorer la collaboration de manière agile et efficace, elles s'alignent complètement dans la stratégie future des entreprises et elles deviendront certainement indispensables à toutes les entreprises de demain avec plus ou moins d'urgence selon leurs marchés et leur secteur d'activité.

Une étude du Forrester d'octobre 2015 montre, par exemple et sans surprise, une appétence plus grande du secteur financier ou des services que de l'industrie traditionnelle.

En effet, les bénéfices délivrés par ce type de démarche permettent de gagner en flexibilité, en réactivité et en qualité pour l'ensemble du système d'information. Or l'informatique est aujourd'hui au cœur de toutes les entreprises, au centre de toutes les activités et en support de tous les produits et services. Ne pas réussir à adopter ce type de démarche revient à rater sa transformation digitale et à se mettre en retrait par rapport aux capacités des concurrents.

L'ensemble des modèles métiers de demain s'appuieront sur des innovations technologiques existantes et inexploitées ou qui seront inventées. Leur adoption facilitée par une démarche agile et DevOps et la capacité à les exploiter est sans doute une des clés de ces futures *sucess story*.

Adopter ces démarches, c'est s'y préparer ; les refuser ou les négliger risque fort de vous pénaliser pour longtemps.

6.1.5. Illustration de la valeur de ce modèle

Il ne s'agit pas de vous faire croire que sans DevOps, le *Uber* de votre activité va surgir pour vous faire disparaître, mais simplement de prendre conscience du niveau d'ancrage de ce modèle dans nos usages quotidiens même à notre insu.

Ce sont ces usages que les entreprises devront supporter, adopter et reproduire car ils vont se généraliser et ils sont entièrement bâtis inconsciemment sur des modèles DevOps.

Pour illustrer la valeur de ce modèle, choisissons un exemple simple et parlant. Ainsi imaginez que vous êtes éditeur d'un service destiné au grand public qui s'appuie sur l'outil informatique. Pour faire simple, prenons l'exemple d'une application disponible sur smartphone, peu importe que vous soyez une grande entreprise ou un développeur indépendant, vous proposez une expérience disruptive valorisée par les utilisateurs.

On peut d'ailleurs supposer que même si vous avez été le premier à proposer cette expérience, une concurrence féroce est vite apparue et qu'il est fort probable que votre application n'est pas restée longtemps la seule de ce type dans les stores d'applications mobiles.

Grâce aux mécanismes de collaboration DevOps et notamment à ses cycles de feedbacks continus, vous avez détecté un nouveau besoin auquel vous pouvez répondre par une fonctionnalité logicielle. Toujours dans notre logique, le métier a pu rapidement qualifier le besoin, le définir et l'expliciter avant de demander sa réalisation technique et sa mise en production. On peut introduire ici toutes les notions d'itération agile, de *minimum valuable product* et de *rings* de déploiement par exemple.

Le fait est que grâce à cette mécanique vous avez pu être réactif et avez proposé une expérience valorisante avant vos concurrents. Vous avez pu créer un avantage concurrentiel plus ou moins durable et allez sans doute gagner des parts de marchés.

Il faut également noter dans cet exemple, la mécanique de mise à disposition *via* des magasins d'applications qui se généralisent sur tous nos appareils y compris sur Windows et sur Mac OS.

Fig. 6.2 *Les six axes de valeur des démarches DevOps pour le management*

Imaginez maintenant que ce n'est pas vous qui avez mis en œuvre ces démarches DevOps mais vos concurrents. Ils sortent avant vous cette nouvelle expérience dans leur application. Que se passe-t-il ? Dans un premier temps, l'utilisateur peut être fidèle et espérer que vous serez réactif et proposerez rapidement quelque chose de similaire puis se

lassant, il risque fort de supprimer votre application pour installer l'application concurrente.

On peut d'ailleurs noter que les *périodes de fidélité* se réduisent de plus en plus dans un monde d'hyperconsommation. Il est ensuite d'ailleurs peu probable que l'utilisateur ainsi perdu revienne à votre application après avoir été déçu de la confiance placée en vous.

Autrement dit adopter une démarche et une organisation DevOps dans un monde où la consommation des services informatiques tend de plus en plus vers le modèle de la mobilité va devenir réellement indispensable pour tous, à plus ou moins long terme.

6.2. L'organisation d'une équipe DevOps

Si DevOps est une approche de collaboration agile, il n'en demeure pas moins qu'une collaboration efficace nécessite une organisation adéquate. Nous verrons que de notre point de vue, il n'existe pas une organisation type mais des organisations différentes selon les contextes, les contraintes et les objectifs.

Aussi variées soient ces organisations, nous pouvons les classer en trois grandes familles et en extraire les grands principes communs.

6.2.1. Le modèle d'organisation Amazon

Le modèle prôné par Amazon est sans doute le plus connu et il peut facilement passer pour le modèle ultime dans l'esprit de certains. Il présente sans aucun doute un certain nombre d'avantages mais il nous paraît difficilement applicable dans de nombreux contextes.

Il faut commencer par rappeler qu'Amazon a très tôt appliqué les principes du DevOps dans sa jeune histoire sans avoir eu besoin de se transformer depuis un modèle plus traditionnel et il est difficile de transposer son modèle principalement orienté web à l'ensemble des cas de figures de l'industrie.

Néanmoins, cette entreprise a poussé très loin les principes de collaboration et d'automatisation du DevOps et son modèle d'organisation est un cas d'école. Il faut tout de même se rappeler que d'autres entreprises sont allées aussi loin dans ce modèle, mais de manière différente avec des modèles d'organisation différents. Nous pouvons par exemple citer Facebook, Google ou encore Microsoft.

Le principe de l'organisation d'Amazon a été dicté par le désormais emblématique paradigme *You build it, you run it* de Werner Voegels CTO du groupe.

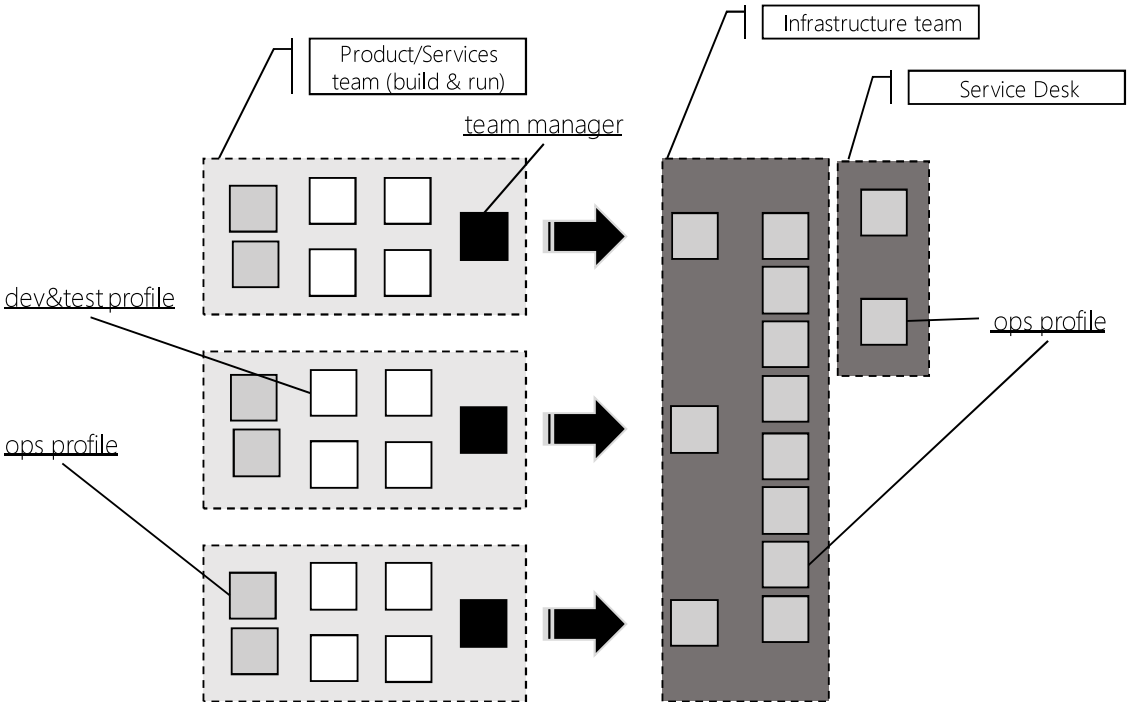


Fig. 6.3 *Le modèle d'organisation intégré selon les principes d'Amazon*

Exprimé autrement, cela signifie que les équipes sont organisées autour d'un périmètre fonctionnel le plus indépendant possible et sont responsables de la qualité de ce service vis-à-vis du client final de l'entreprise. Ce ne sont pas de grosses équipes, de 4 à 7 personnes chez Amazon, moins de 10 personnes le plus souvent. On parle parfois de *pizza team* dont le principe veut que la bonne taille d'équipe corresponde au nombre de personnes que peut théoriquement nourrir une pizza. Pour répondre à une question typique d'un manager français, le concept étant américain, on pense ici à une pizza américaine. Il est vrai que nous pouvons avoir une pizza par personne dans certaines familles françaises.

Ces équipes intègrent l'ensemble des parties prenantes nécessaires à son fonctionnement, du développement au support en production en passant par les tests et la qualité. Ce modèle limite complètement le nombre d'équipes en support avec respectivement un hub gérant les infrastructures et un autre pour le service desk et c'est à peu près tout.

Comme ces équipes s'occupent autant du développement que de la production et de tous les environnements associés en partenariat étroit avec les hubs, Amazon a pu généraliser des pratiques comme, par exemple, le test quasi exclusif en production bien qu'en passant toujours par quelques environnements préalables où sont exécutés des tests exclusivement automatiques.

Ce modèle semble très séduisant de prime abord notamment sur l'aspect de la collaboration entre les équipes, tout le monde est dans la même barque et vogue dans la même direction.

Mais comme tout modèle, il a également ses limites à commencer par le périmètre fonctionnel qui bien que le plus indépendant possible, il doive tout de même s'intégrer harmonieusement dans un ensemble plus important. Ensuite, la responsabilisation des équipes est évidemment une bonne chose mais il ne faut pas tomber dans l'excès

relativement facile de finir par mettre une pression disproportionnée face à des responsabilités excessives.

On peut par exemple mettre en exergue le principe qui est en vigueur chez Amazon de régler l'ensemble des tickets ouvert dans la journée suite à un déploiement.

Amazon a su mettre en œuvre ce modèle brillamment sur bien des aspects mais il se dit aussi qu'ils n'ont pas forcément réussi à en éviter les travers. Aussi, faut-il bien peser le pour et le contre avant de faire le choix de ce type d'organisation qui n'a rien d'exclusif.

6.2.2. Le modèle d'organisation traditionnel

Les modèles d'organisation traditionnels ne sont pas incompatibles avec les démarches DevOps qui visent avant tout à optimiser la collaboration. Il faut néanmoins comprendre que si une optimisation est possible elle implique des changements et ces changements peuvent bien entendu avoir un impact organisationnel.

Dans un modèle d'organisation traditionnel, les équipes de développement et les équipes de production sont clairement séparées, tant dans leurs activités que dans leur hiérarchie respective qui n'ont souvent de commun que le DSI.

Ce modèle ne semble pas, de prime abord, favoriser la collaboration même si elle existe. En conséquence, ce modèle peut sembler disqualifiant pour démarrer une démarche DevOps.

Pourtant, aucune société ne peut fonctionner si les développements et la production ne collaborent pas un minimum et c'est précisément parce que cette collaboration existe déjà même si elle n'est pas optimale.

Mais il faut également distinguer trois cas relativement courants :

- le cas où il n'existe pas d'externalisation des équipes de la DSI ;
- le cas où les développements sont externalisés ;
- le cas où la production est externalisée.

Compatibilité de DevOps avec une organisation traditionnelle non externalisée

Lorsque ni les développements, ni la production ne sont externalisés, la majorité des pratiques DevOps peuvent s'appliquer. Mais elles sont souvent plus complexes à mettre en œuvre.

La recherche des clés de confiance, la construction d'un suivi commun des activités ou encore la capacité à automatiser ce qui fonctionne, ne nécessite en rien de modifier son organisation dans un premier temps. En réalité, le facteur déterminant est bien plus la proximité physique que le lien hiérarchique.

En effet dans ce type d'organisation il est souvent impossible de faire changer les lignes concernant le rapport hiérarchique, et même l'évolution des fiches de postes peut être particulièrement complexe à mettre en œuvre.

Nous partons également du postulat, qu'il est impossible de regrouper dans une même équipe des personnes venant d'équipes hiérarchiquement distinctes car si tel était le cas, nous serions alors dans un modèle d'organisation que nous qualifierons d'intermédiaire.

Dans de tels cas, à défaut de pouvoir s'organiser en équipe, il faut réussir à se donner des occasions de travailler de manière transverse sur des sujets bien spécifiques. Il existe plusieurs façons d'y arriver sans que cette liste ne soit exhaustive :

- **Le hub service**. Il s'agit d'un comité d'expertise transverse limité au périmètre d'un projet, d'un produit ou d'un service. Il s'agit le plus souvent de trouver des moyens de sortir d'une crise ou d'éviter qu'elle ne se répète ou encore parfois de répondre à une commande qui paraît impossible à honorer de prime abord. Le hub réunit le plus souvent des personnes de différentes hiérarchies d'origine avec pour objectif de trouver des solutions sur un objectif bien défini.
- **Le hub d'expertise**. Le hub d'expertise est le pendant du hub service sans que l'objectif ne soit réellement défini. Il existe une sorte de clause générale de compétence. Il peut souvent être sollicité par plusieurs équipes projets et doit donc assurer une sorte de cohérence ou de cadre commun à tous.
- **Les comités thématiques et groupes de réflexions.** Les comités thématiques sont des groupes d'experts issus de différentes hiérarchies ayant pour objectif de réfléchir à un cadre commun à tous les produits, services ou projets. Ils sont donc les cabinets de réflexion de la collaboration et peuvent autant soumettre des propositions d'évolution de l'organisation que des processus. Ils ont en général l'écoute des directeurs des services informatiques et l'influence nécessaire pour faire évoluer les choses.
- **Les task forces**. Ces groupes de travail particuliers sont les pendants des comités thématiques mais avec en général une commande précise de la part de la direction. Ils ont donc non seulement pour but de réfléchir à des propositions mais souvent également de préparer et parfois de piloter la mise en œuvre de l'évolution ou la transformation préconisée. Il peut par exemple exister une task force spécifique sur DevOps dans l'organisation.

La mise en œuvre d'un ou de plusieurs de ces moyens de collaboration peut vous permettre de démarrer une démarche DevOps sans heurter l'organisation. Disposer d'un sponsor de votre démarche avec une position hiérarchique suffisamment importante et transverse est également un accélérateur de succès.

Néanmoins, une fois démontrée la valeur de cette démarche, vous tendrez assez naturellement vers la mise en œuvre d'une organisation intermédiaire prenant en compte les particularismes de votre contexte.

Compatibilité de DevOps avec une organisation externalisant les développements ou la production.

Lorsqu'une partie de l'organisation est externalisée et en particulier les développements, les objectifs d'une démarche DevOps diffèrent légèrement et avec eux les pratiques à

mettre en œuvre. L'organisation traditionnelle n'est donc pas incompatible mais les objectifs de ces équipes vont s'aligner sur les pratiques choisies dans la mise en œuvre.

L'intérêt d'une démarche DevOps existe à la fois pour le client qui externalise que pour le prestataire qui fournit le service, mais nous nous mettrons principalement ici dans la peau du client. Toutefois, avant d'aller plus loin, il est certain que si vous adoptez une démarche DevOps en tant que client, votre prestataire devra évoluer avec vous pour continuer à vous donner satisfaction.

L'objectif lorsque les développements sont externalisés est de garantir de manière agile et automatisée la qualité de ce que le prestataire va livrer. Autrement dit, les pratiques DevOps vont se concentrer sur l'agilité de la production au sens large, c'est-à-dire en comprenant la gestion des livrables du prestataire.

La démarche va alors se concentrer sur deux pratiques essentielles : l'automatisation des mises en production et le monitoring des activités. La pratique de gestion des environnements est également essentielle mais passe au second rang.

Dans une démarche DevOps de ce type, le succès va dépendre de l'interface prévue avec le prestataire : trop rigide et vous serez dans l'incapacité d'aller vers une agilité et une réactivité nécessaire, trop souple et vous ne pourrez garantir la qualité des livraisons de manière factuelle et systématique.

Pour être clair, il n'est pas nécessaire d'avoir des équipes mutualisées avec leur prestataire mais il ne faut pas non plus avoir le sentiment d'être pris en otage dans une relation contractuelle, dont on ne voit pas les bénéfices. Si tel était le cas, le monitoring avec des métriques partagées deviendrait alors une priorité absolue, car c'est à travers ces indicateurs communs que la confiance va pouvoir de nouveau se construire.

Pour faire simple, une démarche DevOps est possible dès lors que la relation est suivie de manière factuelle par des indicateurs pertinents et partagés. Elle fonctionne sur la base d'un processus de mise en production pensée en collaboration et automatisant autant que possible les validations nécessaires du produit livré par le prestataire. Enfin, les environnements sont fournis selon un catalogue de service et des caractéristiques maîtrisées en transparence avec le prestataire.

C'est sur cette base que vous pourrez réussir une démarche DevOps avec un prestataire externe. Et cette base est identique dans le cas d'une externalisation des opérations.

Pour être compatible avec une démarche DevOps, une production externalisée doit pouvoir à la fois garantir un très haut niveau de disponibilité et de réactivité, tout en étant capable de mettre en production des produits ou services jusqu'à plusieurs fois par minute. Le prestataire devra certainement mettre en œuvre les pratiques DevOps que nous avons vues pour le client, mais le client devra également gagner en agilité dans ses développements pour pouvoir proposer des mises en production plusieurs fois par jour, par semaine ou par mois.

Dans ce dernier cas, il est indispensable de mettre en œuvre les pratiques agiles utiles dans son contexte mais également de peaufiner sa stratégie de build, sa gestion des branches de

développement ainsi que sa gestion des sources. En sus du monitoring, la télémétrie va devenir centrale et la qualité ne pourra pas être négligée.

La réussite d'une démarche DevOps avec une organisation en partie externalisée n'a rien de facile ou de simpliste sans pour autant être compliquée, il faut simplement redoubler de rigueur dans sa mise en œuvre.

6.2.3. Le modèle d'organisation intermédiaire

Un modèle d'organisation intermédiaire consiste à garder une organisation hiérarchique traditionnelle et à transformer l'organisation fonctionnelle en équipe transverse aux développements et aux opérations ou aux différents hubs fonctionnels.

Autrement dit, vous conservez une organisation verticale au sens RH du mot, avec une hiérarchie en fonction des métiers.

Business Analyst incluant le rôle de Product Owner – Les *Business Analysts* sont les représentants des utilisateurs et des métiers, ils sont en capacité de retranscrire les besoins pour les rendre compatibles avec les mécanismes de collaboration DevOps, ils peuvent participer aux prises de décisions concernant les priorités et les validations des services et des produits. On y inclut généralement les Product Owners quand ceux-ci ne sont pas directement rattachés à un rôle métier différent au sein de l'entreprise.

Les Program Managers incluant le rôle de chef de projet ou de ScrumMaster – Le Program Manager est le responsable du périmètre fonctionnel qui lui est confié. Dans un modèle d'organisation DevOps intermédiaire, il n'a aucun lien hiérarchique avec son équipe ce qui le rapproche du rôle de ScrumMaster. Dans le cas où un lien hiérarchique pourrait exister, il serait alors un véritable chef de projet mais il ne pourrait alors cumuler avec un rôle de ScrumMaster au sens agile d'un tel poste.

Les développeurs – Les développeurs sont les ingénieurs logiciels en charge de la conception et de la réalisation technique des fonctionnalités du service ou du produit. Ils peuvent également être le dernier niveau de support du produit ou encore avoir sous leur responsabilité la maintenance du service.

Les testeurs – Ce rôle est différent de celui des qualiticiens bien qu'il soit question de qualité du produit ou du service. En effet, il s'agit pour eux de garantir la qualité du produit développé au sens de sa robustesse et de sa maintenance. Les testeurs doivent également souvent vérifier la conformité du produit ou du service avec les besoins spécifiés par les utilisateurs.

Les ingénieurs qui regroupent les deux précédents rôles – Dans les organisations DevOps, nous rencontrons de plus en plus de situations ou les développeurs et les testeurs sont organisés dans une unique entité hiérarchique et leurs fiches de poste sont fusionnées. Dans ce cas, il est important d'introduire un certain nombre de bonnes pratiques telles que la fonction est développée et testée par deux personnes distinctes et la mise en production est validée par un tiers par exemple.

Les qualiticiens incluant les data scientists – Ce rôle est relativement nouveau, mais il est fondamental dans une démarche DevOps. En effet, le qualiticien moderne est en charge de la collecte, de l'analyse et de l'exploitation des données disponibles sur l'utilisateur final et ses usages. Il est capable de comprendre ces données et au travers de ces analyses, d'orienter la conception ou les priorités du produit ou du service.

Les opérations – Ce rôle est relativement générique, car il conserve quasiment sans changement les rôles de la production : gestion des livraisons, garant de la sécurité, maintenance des infrastructures, suivi et *monitoring* des technologies… Cette dernière activité devient centrale dans une démarche DevOps. Les activités sont donc amenées à évoluer dans une démarche DevOps mais l'organisation n'est pas pour autant nécessairement impactée de manière importante.

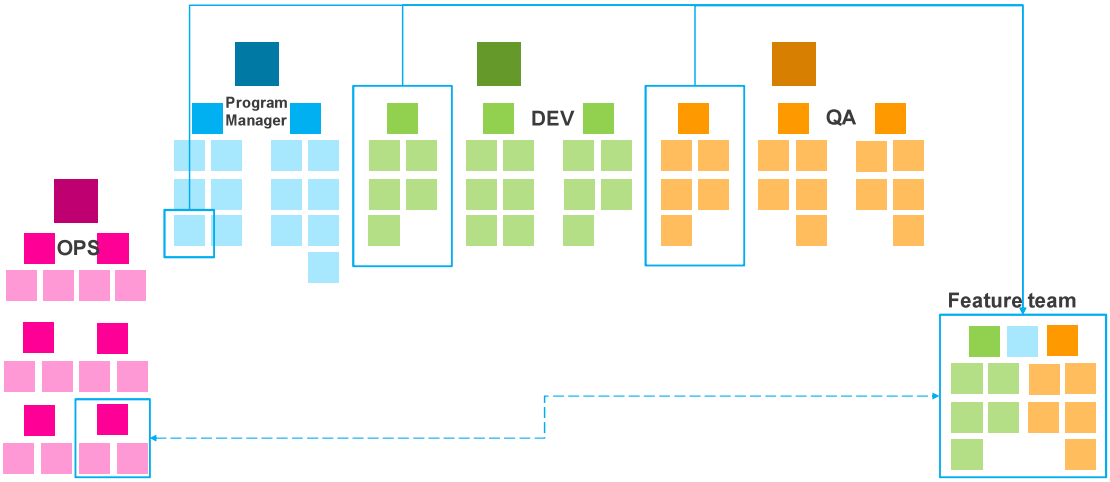


Fig. 6.4 *Un modèle d'organisation DevOps intermédiaire*

Dans le même temps vous avez une organisation horizontale avec des équipes qui n'excèdent pas une dizaine de personnes et qui ont une responsabilité commune sur un périmètre fonctionnel donné. Cette équipe est composée de ressources dédiées issues des différents corps métiers hiérarchiquement distincts.

Le plus souvent, l'équipe est animée fonctionnellement par un Product Manager qui n'a donc pas de lien hiérarchique direct avec les autres membres de l'équipe. Cependant, c'est bien l'équipe qui est responsable collectivement de l'expérience de l'utilisateur sur son périmètre fonctionnel. Le plus souvent ces équipes sont appelées des *feature teams*.

Dans le même temps, certaines fonctions transverses sont organisées en équipes distinctes dont les compétences sont utilisées par l'ensemble des feature teams. C'est la notion organisationnelle de hub. Dans la réalité, il peut exister plusieurs hubs dédiés à un certain nombre de feature teams pour une même expertise afin d'éviter que ces équipes ne deviennent trop grosses.

Parmi les fonctions classiques des hubs nous pouvons retrouver :

- **La gestion de l'infrastructure**. Il s'agit de dédier la gestion de l'infrastructure et le contrôle des mises en production à une équipe chargée de suivre les indicateurs adéquats, de réagir aux problèmes et d'assurer un support 24h/24h si nécessaire

- **La gestion du service de support**. Il s'agit de permettre aux utilisateurs d'avoir un point de contact unique et une aide immédiate en cas de besoin. Ce mode de fonctionnement assure donc une plus grande satisfaction client et une meilleure prise en compte du feedback
- **La gestion du monitoring et des données**. Il s'agit de monter un cadre de contrôle, de suivi et d'analyse communs aux différentes équipes pour pouvoir assurer une gestion efficace et cohérente du périmètre global du produit ou du service.
- **La gestion des tests et des normes**. Il n'est pas rare d'intégrer directement ces ressources dans les features team plutôt que de les organiser sous forme de hubs mais les deux approches sont possibles. Le hub sera particulièrement pertinent quand il s'agira de contrôler rigoureusement l'aspect réglementaire ou normatif du périmètre.
- **La gestion de la sécurité**. Il s'agit d'assurer la sécurité transverse du service ou du produit tant sur l'aspect de son architecture que sur celui des données.

Il est important de noter que rien n'interdit à un hub de cumuler plusieurs fonctions.

Le hub est une notion fondamentale de l'organisation DevOps que l'on retrouve dans nos trois grandes catégories d'organisation DevOps. Il s'agit donc d'un principe commun au même titre que la proximité et le périmètre transverse d'une équipe.

6.2.4. Les principes communs à toutes les organisations DevOps

Nous pouvons donc constater qu'il existe différentes organisations compatibles avec les démarches DevOps. On peut même voir dans les différentes organisations évoquées une évolution dont l'objectif serait de toujours être en capacité de mieux collaborer.

Néanmoins, nous pouvons extraire de ces modèles des principes communs qui forment le socle de compatibilité entre la démarche et l'organisation : ce sont la **proximité**, la **co-responsabilité** du périmètre fonctionnel, le sponsorhip et la **notion de hub** de service organisationnel.

Ces éléments sont absolument fondateurs. Il paraît évident que pour mieux collaborer, il faut créer une forme de proximité qui n'est pas nécessairement physique. Même si une proximité physique entre les équipes facilite objectivement les relations, la proximité recherchée est davantage sociologique : les personnes qui ne sont pas rattachées aux mêmes départements se fréquentent, se connaissent et échangent volontiers sur des sujets qui ne sont pas liés à leur environnement de travail quotidien.

Pour que cette proximité devienne possible, il est nécessaire de partager un intérêt commun. Le partage des objectifs et des responsabilités d'un même périmètre fonctionnel au travers d'organisations distinctes permet de créer et de maintenir cet intérêt commun au sein duquel peut être cultivée la proximité.

Les réunions régulières autour de ce périmètre partagé permettront aux personnes d'échanger fréquemment, de mieux se connaître et peut-être de s'apprécier et ainsi de construire la **proximité** nécessaire.

Enfin, parce que tous les métiers ne se prêtent pas à un partage des ressources au sein d'une équipe unique mais aussi parce que la responsabilité du périmètre fonctionnel doit tout de même être partagée pour créer de la proximité, nous les organisons sous forme de *centre d'expertise et de service* pour les autres équipes. C'est la notion de hub que nous avons pu retrouver dans chacun des modèles organisationnels que nous vous avons présentés. Elle permet à une équipe de se voir attribuer la responsabilité d'une fonction tout en étant au service de toutes les autres. Nous retrouvons régulièrement ce concept dans la gestion d'éléments d'infrastructure par exemple.

Cependant, une bonne organisation ne fait pas toute la gestion d'une équipe et c'est encore plus vrai dans une démarche DevOps. Elle doit être combinée à un certain nombre de concepts de management indispensables.

6.3. Le pilotage par l'expérimentation

6.3.1. Émettre des hypothèses

Le principe du pilotage par l'expérimentation est largement inspiré de l'expérience des start-up et des principes du Lean Startup. Il part objectivement d'une constatation de bon sens : pourquoi perdre du temps à développer un produit ou un service alors que l'accueil des utilisateurs est par nature incertain. Il est également bon de noter que cette incertitude augmente exponentiellement avec l'innovation, or les start-up se veulent par nature innovante.

L'exemple particulièrement connu mais extrêmement parlant est celui de Dropbox qui a publié en premier lieu une vidéo de présentation de son service avant même de le construire pour recueillir un maximum de retours sur ces hypothèses.

Les démarches DevOps qui sont agiles par nature correspondent parfaitement à cette philosophie en promouvant l'expérimentation des hypothèses sans pour autant les transformer en certitudes. Mais pour expérimenter, il faut savoir émettre des hypothèses.

Une hypothèse est un paradigme que l'on considère comme vrai car justifié par des données. Les data sont fondamentales dans une démarche DevOps et toute hypothèse doit être fondée sur des données factuelles à partir desquelles on tente une interprétation.

Ces données ne sont pas nécessairement directement collectées sur le produit, surtout quand celui-ci n'est pas encore conçu. Les données doivent autant prendre en compte le produit ou le service que son environnement direct et indirect. Ainsi, les données de marchés ou les chiffres de la concurrence sont des données fondamentales pour émettre des hypothèses.

Mais les données ne suffisent pas toujours, et, en particulier dans les premiers temps, elles peuvent et parfois doivent être complétées par l'intuition. Il est donc nécessaire d'associer aux données très factuelles une dose de créativité avec du *design thinking* par exemple.

En effet, les démarches DevOps promeuvent une forme de créativité continue centrée sur l'utilisateur ou le bénéficiaire du service et nécessitent donc d'émettre en permanence des hypothèses à tester.

6.3.2. Conception simple et produit minimum viable, expérimentation et feedback

Les hypothèses émises sont par nature incertaines même si elles sont émises sur la base de données factuelles. Une hypothèse doit donc être testée et aucune ne peut être considérée comme une certitude avant d'avoir été validée.

Or il se trouve que l'agilité n'est pas une science et certainement pas une science exacte faite de certitudes. L'idée est donc d'itérer autour d'une hypothèse en ne la considérant jamais comme une certitude, mais en tentant d'évaluer *l'adhésion* des utilisateurs ou des bénéficiaires tout en restant conscients que nous vivons dans un monde extrêmement mouvant. Dans ce monde, les certitudes d'un jour s'effondrent sans peine le lendemain.

Aussi, il est nécessaire d'investir le moins d'effort possible dans chaque hypothèse testée pour pouvoir justement la tester le plus rapidement possible et en tirer des enseignements applicables avant que la tendance n'ait changé.

C'est ce qu'on appelle le **produit minimum viable**. Autrement dit il s'agit du produit le plus simple, mais suffisamment abouti pour donner la valeur attendue au bénéficiaire et pourvoir mesurer sa réaction.

Chaque hypothèse est donc modifiée en fonction des réactions enregistrées et devient alors une nouvelle hypothèse qu'il faudra tester. À la suite de l'évolution des hypothèses, soit il est nécessaire de repartir d'un produit minimum viable complètement différent, soit il est nécessaire de faire évoluer le précédent produit minimum viable pour tester les hypothèses ainsi modifiées.

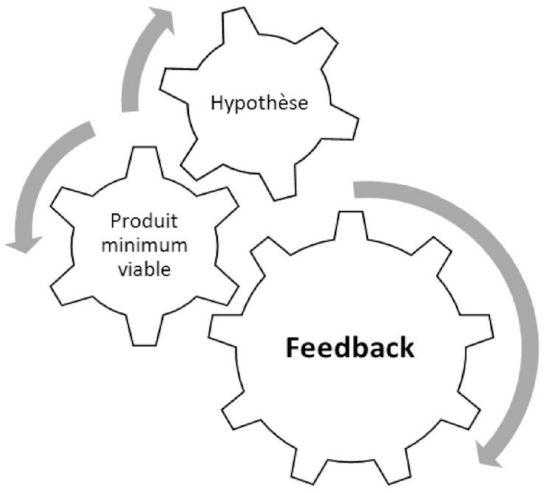


Fig. 6.5 *Hypothèse, produit minimum viable et feedback sont des notions liées comme dans un engrenage : ils bougent ensemble en permanence.*

Cette notion de conception minimale a des impacts jusque dans la manière de développer le produit ou le service. D'une part, parce que désormais, son développement est piloté par des hypothèses et d'autre part parce que sa conception ne peut plus être réellement anticipée. Parce que la conception ne peut être anticipée, il est impossible de spécifier dans un long cahier des charges le besoin pour le faire ensuite produire par des équipes

séparées. Il est absolument nécessaire de revenir en permanence sur les hypothèses en intégrant le feedback, il est donc indispensable de mettre en œuvre des méthodes agiles pour piloter le développement de ces produits et services dans une démarche DevOps.

De même, parce que le produit évolue en permanence sans que l'on soit capable de prédire les composants qui seront nécessaires et qui vont arriver, il n'est pas possible de concevoir par anticipation des *composants optimisés* qui devraient faciliter l'arrivée d'une nouvelle fonctionnalité future en partageant un périmètre fonctionnel ou technique plus ou moins large.

Cette recherche de l'optimisation par l'anticipation à outrance est souvent la source d'une complexification injustifiée des architectures techniques ou logiques. La conception simple se refuse d'entrer dans ce type de paradigme. La logique de conception veut que l'on implémente directement la solution la plus simple pour délivrer la valeur attendue, qu'elle ne semble ou non optimisée pour les évolutions futures. Cette logique de conception devient indispensable dans le cadre d'hypothèses et d'expérimentation continue.

Mais tout cela ne peut fonctionner sans un feedback de qualité pour réellement pouvoir s'améliorer. Il est donc tout autant indispensable de ne pas négliger les moyens de l'obtenir. Il est fondamental de sélectionner et mettre en œuvre les canaux de feedback qui seront les plus efficaces, et, dans le digne esprit de notre démarche, de les tester pour les choisir !

6.3.3. Les différents canaux de feedback

Les canaux de feedbacks sont les voies, usages et moyens utilisés pour permettre de mieux comprendre les comportements et les attentes des bénéficiaires du service ou du produit. Un canal de feedback permet systématiquement et nécessairement de recueillir des données factuelles et quantifiables.

Il existe deux moyens d'obtenir ces précieuses données : soit en les captant directement depuis les bénéficiaires eux-mêmes, soit en les déduisant de leur environnement ou de facteurs extérieurs. Explicitons maintenant les moyens mis en œuvre dans une démarche DevOps pour les obtenir.

Les canaux directs

Sans chercher à être exhaustif, parmi les moyens de recueillir de l'information directement de l'utilisateur, nous pouvons citer :

- **Les démonstrations**. C'est la méthode traditionnelle issue des méthodes agiles. Il s'agit de montrer un prototype suffisamment abouti pour être manipulé par un utilisateur afin de pouvoir ouvrir une discussion et recueillir du feedback.
- **La télémétrie**. Notamment lorsque les démonstrations sont impossibles, la télémétrie permet de recueillir des données factuelles d'un utilisateur qui sont ensuite analysées de manière pertinente pour en tirer des enseignements.

- **Les forums et sites dédiés**. On peut notamment citer les sites de votes permettant d'écouter les utilisateurs sur leurs suggestions d'évolutions et d'améliorations. Si ce type d'interface permet une interaction directe avec les utilisateurs, elle rend aussi beaucoup plus complexe l'analyse du feedback et ouvre la porte à tous les types de tricherie possibles comme les comptes frauduleux ou les votes fantômes.
- **Les fonctions dédiées du produit**. Les produits ou services peuvent également directement inclure des moyens d'interaction avec l'utilisateur qui permettent de réagir dans un contexte précis et facilement identifiable. On peut par exemple citer les *smileys* que l'on trouve parfois dans certains services numériques pour noter et commenter son expérience client.

Les canaux indirects

Parmi les moyens de déduire de l'information sur l'utilisateur, et toujours sans chercher à être exhaustif, nous pouvons citer :

- **Les analyses de marché**. Ce sont de puissants produits d'anticipation des comportements des utilisateurs. Les analystes sérieux font en général un travail particulièrement sérieux pour découvrir les tendances en cours et à venir. Ils cherchent à anticiper les besoins des utilisateurs à court et moyen terme.
- **L'analogie avec des comportements similaires**. Les analyses de comportement sont des outils puissants mais il arrive parfois sur un produit innovant que les comportements n'existent pas encore. Dans tous les cas, une analyse de comportement sur des usages proches et ciblés peut permettre par analogie d'obtenir un feedback de manière détournée mais particulièrement précieux.
- **Les analyses de la culture**. L'analyse des comportements est certes particulièrement intéressante mais il est un paramètre essentiel qu'il ne faut pas oublier : la culture du lieu géographique où le produit ou service est proposé. L'analyse de la culture peut apporter des éléments intéressants notamment sur les choses à ne pas faire. Cette analyse peut également être associée à une analogie des comportements pour fournir un feedback supplémentaire de grande valeur.
- **Les panels expérimentaux**. À l'image des sondages et des expérimentations marketing, il peut parfois être utile de prendre le feedback de panels représentatifs de la population cible. La limite est comme souvent dans la méthode de constitution du panel afin d'assurer une représentativité suffisante pour éviter le risque d'un feedback contre-productif.

Mais le feedback est inutile, si nous ne savons pas l'utiliser et tout en apprendre.

6.4. Continuous feedback and learning

Créer de la proximité avec les bénéficiaires ou leur représentant, générer des retours et du feedback de leur part de manière régulière et continue n'est pas suffisant. Il faut également savoir exploiter ces formidables mines de données pour apprendre continuellement. Le lien entre feedback et apprentissage semble-t-il évident ? Ce pourtant pas toujours le cas.

Il existe une différence majeure entre **écouter** l'utilisateur et l'**entendre.** Et cette différence se ressent jusque dans les outils en général utilisés pour traiter les feedbacks.

Écouter l'utilisateur consiste à mettre en œuvre des mécanismes qui permettent aux utilisateurs de s'exprimer et de prendre connaissance de leurs doléances. Il ne s'agit pas nécessairement de prendre en compte leurs demandes mais simplement d'en prendre connaissance.

Entendre l'utilisateur consiste non seulement à l'écouter mais également à analyser ses dires et à se forger des convictions qui incluent une part d'hypothèse. Il s'agit de prendre en compte ces convictions directement dans les backlogs, à suivre toute la boucle d'expérimentation des hypothèses et à mesurer toutes les métriques nécessaires pour s'assurer de l'atteinte des objectifs.

Pour apprendre de l'utilisateur, il faut donc commencer par l'entendre et non par simplement l'écouter.

Lorsque l'on se forge une conviction, il est nécessaire de la challenger le plus rapidement possible. Les démarches DevOps sont donc indispensables ne serait-ce que par leur capacité à accélérer la transformation.

Au sein de ces pratiques, il existe plusieurs techniques d'apprentissage basées sur le feedback et l'écoute des utilisateurs :

- **Le machine learning**. Nous avons déjà évoqué à plusieurs reprises le machine learning et son apport dans les démarches DevOps. Il est évident que c'est une source d'apprentissage et d'analyse de premier plan.
- **Le test en production**. Il existe un malentendu courant quand on évoque le test en production. Il ne s'agit pas d'éliminer tous les tests de recette et les environnements *anté-prod*, loin de là. L'objectif est plutôt de tester une hypothèse directement en production pour mesurer les réactions et l'adoption des utilisateurs. Il s'agit bien d'un test mais pas dans le but de détecter les anomalies, il s'agit plutôt de mesurer le comportement, le produit ou service doit donc être idéalement sans anomalies comme pour toute mise en production.
- **Les scenarii A/B testing**. Les scenarii A/B testing sont une extension du principe des tests en production. En effet, il s'agit non plus de tester un mais deux scénarios distincts directement en production afin de pouvoir comparer les comportements associés et donc l'adoption potentielle. Au cours de cette étape, l'objectif est de ressentir l'audience cible du service.
- **Les Canary Releases**. Il existe plusieurs conceptions du principe des *Canary Releases*, celle plutôt utilisée par Facebook, par exemple qui consiste à déployer une nouvelle fonctionnalité sur un groupe restreint d'utilisateur avant d'immédiatement l'étendre à tous les autres utilisateurs, si tout fonctionne comme prévu, et celle de Microsoft, *via* son principe de *rings* et de multiples groupes utilisateurs croissants. Dans les deux cas, il s'agit toujours de tester les nouvelles fonctionnalités sur un groupe restreint d'utilisateurs avant de le diffuser plus largement.

Nous avons déjà détaillé ces mécanismes dans les précédents chapitres mais ils sont au cœur de la boucle d'apprentissage des démarches DevOps.

C'est par l'apprentissage que l'on garantit la satisfaction des bénéficiaires du service et de leur représentant. Mais au-delà de la satisfaction, l'apprentissage vous permet de connaître mieux que quiconque vos clients. Ce précieux savoir fait de vous un partenaire naturel pour les directions métiers en apportant de la valeur directement dans leurs affaires et ils vous en seront désormais reconnaissants.

6.5. La reconnaissance par l'implication du métier

Pour réellement réussir à impliquer les métiers il faut qu'ils y trouvent un intérêt. Et pour intéresser les métiers, il vous faut démontrer que vous leur apportez de la valeur sur du long terme et de manière régulière. Autrement dit, vous devez instaurer une relation gagnant-gagnant.

6.5.1. Intéresser par la valeur pour impliquer

Il n'est pas forcément aisé de prime abord de trouver les arguments qui éveilleront l'intérêt des interlocuteurs métiers au sein des organisations. Après tout, les services informatiques ont toujours été vus comme des fonctions de support avec peu d'emprise sur la chaîne de valeur et les clients.

Il est évident que de nos jours, cette vision correspond de moins en moins à la réalité. L'intégration du numérique à tous les niveaux et l'énorme quantité de données que nous pouvons fournir changent complètement la dimension des directions des services informatiques au sein de l'entreprise. Désormais, grâce à DevOps, elles peuvent faire valoir leur rôle sur de multiples plans, dont nous donnons ici une liste non exhaustive :

- **Une connaissance du client inégalée**. Les mécanismes de télémétrie, d'expérimentation et d'apprentissage que nous avons évoqués vont vous permettre de connaître les comportements des utilisateurs de vos produits et services comme jamais auparavant cela n'avait été possible. Vous être désormais le lien direct entre votre organisation et ses clients, cela ne pourra pas laisser vos interlocuteurs insensibles.
- **Des mécanismes d'intimité inespérés**. L'outil informatique et le numérique sont déjà partout. Avec la mobilité tant sur les devices que dans les *tâches continues* qui leur sont associés, vous êtes désormais en lien avec vos bénéficiaires quasiment en permanence. La frontière entre vie personnelle et professionnelle étant de plus en plus fine, vous êtes en capacité d'inventer de nouveaux modes de relations avec une intimité que l'utilisateur pourra lui-même décider de régler du plus confidentiel au plus partagé.
- **Une maximisation de la réactivité aux besoins**. La réactivité est au cœur des démarches DevOps, désormais grâce à elle vous pourrez répondre plus vite et mieux aux besoins de vos bénéficiaires. Cette promesse, longtemps espérée, rarement concrétisée est un prérequis à l'ensemble des bénéfices du DevOps et fonde l'ensemble des pratiques DevOps. Elle peut donc enfin devenir réalité.

Il est certain que si vous mettez tout en œuvre pour faire valoir ces arguments auprès des populations métiers, vous aurez toutes les clés pour les impliquer durablement. À vous maintenant de savoir tirer de la valeur de cette collaboration.

6.5.2. Tirer de la valeur de l'implication du métier

Pour réussir à tirer de la valeur de vos relations avec vos interlocuteurs métiers, il faut créer des relations de confiance qui ne seront pas mises en danger par diverses tensions internes. C'est le sens du modèle DevOps dont nous avons parlé. Mais ce n'est pas suffisant.

Les interlocuteurs doivent également se rendre compte que leur précieux temps est utile et que l'ensemble de leur feedback est optimisé. Il est donc important de ne pas demander aux métiers de répéter deux fois un même besoin parce que dans un cas il sera utile pour les développeurs et dans l'autre cas il répondra à une question des équipes de production.

La capture des besoins et le feedback des métiers doivent impérativement être unifiés pour réussir à en tirer de la valeur. Il s'agit tout autant de réussir à entendre le métier, que de disposer d'un cadre de description des besoins uniques pour l'ensemble des équipes informatiques.

Il ne s'agit pas de laisser croire que les besoins métiers peuvent être décrits d'une unique façon pour l'ensemble des équipes que ce soit pour un développement agile ou pour un test de mise en production mais plutôt de disposer d'un recueil d'information unique qui sera ensuite décliné autant de fois que nécessaire pour les différentes équipes.

Du point de vue du métier, celui-ci décrit son besoin une seule fois. Du point de vue des équipes informatiques, ils ont un même message en amont qui sert de base au travail de spécification en fonction de leurs besoins spécifiques.

Les démarches DevOps étant fondées sur les valeurs de l'agilité, il est aisé de retrouver des *use cases* ou *cas d'utilisation* pour décrire les besoins du métier qui sont ensuite déclinés en un ou plusieurs scénarios de type *user story* pour les développements ou *test case* pour les équipes de production.

C'est en effet ainsi que vous pourrez à la fois aligner la description des besoins dans une démarche de collaboration DevOps et en tirer un maximum de valeur pour vos produits et services.

6.5.3. Communiquer pour être reconnu

Il faut parfois reconnaître que les équipes informatiques n'ont pas une image de marque suffisamment valorisée dans la majorité des organisations. Il peut même exister une forme de défiance vis-à-vis de ce service. Il serait irréaliste de penser que l'on peut mettre en œuvre une relation gagnant-gagnant avec l'ensemble de l'entreprise du jour au lendemain.

Pour y arriver, il va falloir démontrer la valeur de cette relation sur un périmètre réduit, avec des équipes métiers ciblées par exemple et ensuite communiquer largement sur ces succès communs.

Négliger l’aspect communication compliquera énormément vos efforts de diffusion de la culture DevOps et avec elle votre capacité d’adoption de la démarche. La communication est également un facteur essentiel pour faire naître la reconnaissance.

Établir un plan de communication préalable qui vivra tout au long de votre mise en œuvre de cette relation privilégiée avec les équipes métiers est absolument indispensable. Il n’est pas rare de voir des organisations le négliger totalement, c’est une erreur fondamentale.

En résumé

Le modèle des démarches DevOps repose sur des fondations agiles, autant pour les développeurs que pour les responsables de production. Dans le premier cas, un minimum de pratiques agiles est requis. Dans le second, une industrialisation et une fiabilisation des infrastructures sont attendues pour gagner en agilité et donc en réactivité face au changement.

C’est uniquement sur ces bases qu’une relation de confiance peut être établie, d’abord au sein des équipes informatiques et ensuite avec les équipes métiers. Enfin, il faut prendre conscience du fait que ce modèle est lié à l’ADN même des stratégies des entreprises de demain.

Pour évoluer vers ce modèle, nous avons constaté que de nombreuses organisations différentes peuvent être adoptées en fonction du contexte, dès lors qu’elles répondent à des principes fondateurs : la proximité, la co-responsabilité du périmètre fonctionnel et la notion de hub de service organisationnel.

Nous avons compris que ce modèle repose sur un principe fondateur d’expression d’hypothèses, d’expérimentation et de recherche continue de feedback. C’est bien à partir des retours des utilisateurs que les démarches DevOps mettent en œuvre des pratiques d’apprentissage en continu et c’est grâce à l’ensemble de ces éléments que nous pouvons créer une relation mutuellement valorisante avec les équipes métiers.

Chapitre 7

DevOps pour la stratégie business

Le *business* est un terme générique pour parler aussi bien des directions métiers au sein des entreprises que des *décideurs* stratégiques. Des tentatives de définition de ce terme très général cherchent à distinguer les personnes visées en fonction de la typologie ou de la taille de l'entreprise. Par exemple régulièrement on voit la distinction entre *start-up/petites entreprises*, *entreprises du web* et *entreprises multifonctionnelles*. Ce type de classement est particulièrement réducteur et surtout très peu représentatif de la réalité.

En effet et même si ce n'est pas particulièrement intuitif, ce type de division logique utilise comme clé de répartition la taille des équipes *opération*s pour évaluer leur collaboration au sein d'une démarche DevOps. Elle présuppose que les *start-up et petites entreprises* sont *No-Ops*, une autre façon de dire que les opérations sont gérées sans distinction de rôle. Dans la suite logique de ce point de vue, les *entreprises du web* sont naturellement organisées en petites équipes orientées fonctionnalités : on fait ici du modèle Amazon une généralité, alors que c'est plutôt une exception sur le marché. Tous ceux qui ne sont pas visés par cette clé de répartition entrent dans une troisième catégorie très générique.

Le pragmatisme doit dominer, il est inutile de complexifier la réalité qui est déjà suffisamment complexe. Les principes d'une démarche DevOps sont les mêmes pour tous, et, une démarche DevOps ne peut fonctionner que si ses pratiques sont adaptées au contexte.

Il existe quelques grands objectifs métiers pour lesquelles mettre en œuvre une démarche DevOps est particulièrement pertinent.

Découvrons-les ensemble.

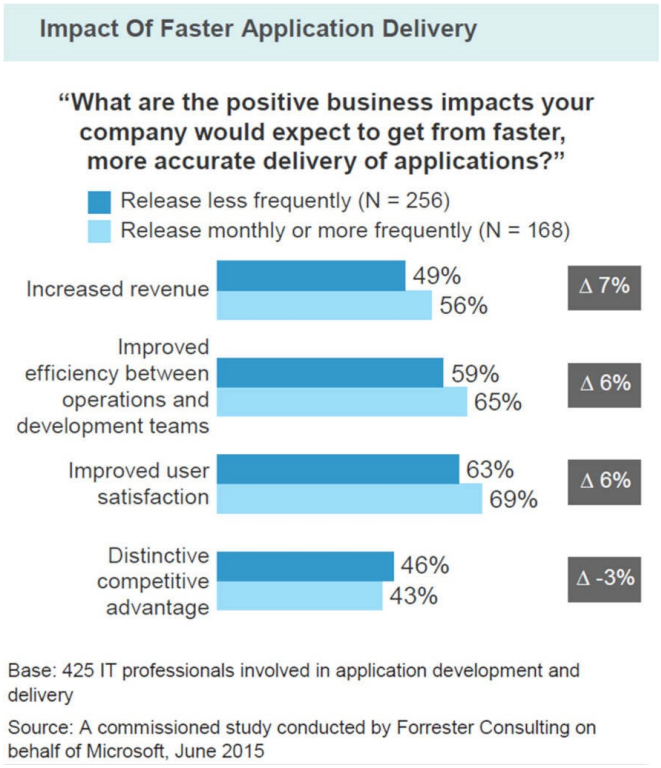


Fig. 7.1 *Impact des résultats des démarches DevOps*

DevOps : une culture au cœur du business

« *Every business is a software company* ». Tel est le message que Satya Nadella, l'actuel CEO de Microsoft, ne cesse de marteler. Il aime souligner à travers cette affirmation à quel point l'outil informatique et la production logicielle sont aujourd'hui au cœur de la création de valeur des entreprises et ce, quel que soit leur secteur d'activité. Dès lors, on comprend vite pourquoi une démarche DevOps s'inscrit dans le *business model* de demain. En effet, c'est cette culture pleine de bon sens, qui permet d'être en permanence centré sur le client, tout en accélérant la production de valeur et la réactivité, qui va permettre de construire les prochaines *success stories* du marché.

7.1. L'adoption facilitée de l'innovation

7.1.1. Une innovation nécessaire

Pourquoi l'adoption des innovations technologiques de plus en plus nombreuses et diversifiées est-elle devenue une priorité stratégique pour les organisations ? Répondre à cette question est une priorité si l'on souhaite percevoir les bénéfices d'une démarche DevOps dans ce contexte.

L'intérêt de se trouver en capacité d'adopter rapidement les innovations technologiques est tellement évident que la question pourrait paraître superflue. Mais ce serait négliger un élément essentiel : comprendre les enjeux fondamentaux permet d'anticiper les bonnes stratégies.

La technologie au sens large et l'informatique en particulier sont aujourd'hui devenues indispensables à la quasi-totalité des tâches quotidiennes dans l'ensemble des secteurs

d’activités. Les bénéfices obtenus ou espérés par ces innovations technologiques sont directement quantifiables, que ce soit en économie réalisée ou en avantage concurrentiel généré.

Autrement dit, adopter rapidement une innovation technologique pertinente pour son activité génère rapidement des bénéfices mesurables financièrement de manière importante. Pourtant, tout le monde ne perçoit pas nécessairement ces bénéfices, car les investissements ne sont pas toujours judicieux, et les solutions retenues sont parfois fort coûteuses à mettre en œuvre.

La solution tient en deux mots : pertinence et adoption.

7.1.2. Une adoption pertinente

Nous ne discuterons pas ici des critères de pertinence liés à l’adoption d’une innovation technologique particulière. Ce n’est pas que le sujet soit totalement étranger à une démarche DevOps car les principes de pilotage par le coût et la valeur sont au cœur de cette discussion, mais le sujet peut faire l’objet d’un livre à lui seul.

Nous nous concentrerons donc sur un message : une démarche DevOps facilite l’adoption de l’innovation en limitant les coûts.

Attention, adopter une démarche DevOps n’est pas forcément simple ni facile, elle nécessite un investissement qui selon votre contexte peut être important. La grande force de cet investissement est bien que, s’il est réalisé correctement, il permettra de limiter l’effort et le coût de l’adoption de l’ensemble de vos choix technologiques innovants dans le futur.

En effet, une démarche DevOps vous permet de construire une collaboration de confiance, pérenne et agile entre les représentants du métier et les acteurs du système d’information. Pour y parvenir, elle impose un minimum d’agilité et de fiabilité à votre informatique afin d’évoluer naturellement vers une architecture plus flexible.

7.1.3. Une démarche pertinente et innovante

Avec le temps, on se rend rapidement compte que l’analogie traditionnelle inspirée du BTP entre maîtrise d’ouvrage et maîtrise d’œuvre disparaît pour laisser la place à des experts dont les compétences sont plus transverses. Avec eux et la maturité, les réflexes de l’industrie sont transposés au monde du logiciel et du matériel.

Au final, de manière plus ou moins importante selon les secteurs, nous nous retrouverons avec le *moteur* des logiciels informatiques complètement découplés des *interfaces* et des *ressources*. Le couplage le plus faible possible sera recherché et les services deviendront la clé de voûte des architectures.

Le grand avantage de cette recherche du couplage faible entre les composants du système d’information inspirée par DevOps, c’est que l’adoption d’une innovation technologique ne va faire qu’enrichir le patrimoine informatique sans en remettre en cause des pans entiers.

Il faut néanmoins être conscient du patrimoine de certaines grandes entreprises qui rendra ce type de démarche plus coûteux pour elles. L'investissement sera néanmoins nécessaire, plus tôt il sera fait, plus il sera lissé dans le temps.

Enfin, cette adoption suppose une évolution de la culture même de l'entreprise, seule façon de pouvoir réellement atteindre un objectif de réduction du time to market.

7.2. Enfin réussir à réduire le time to market

7.2.1. Une promesse souvent entendue, rarement réellement tenue

Réduire le temps nécessaire pour mettre à disposition des utilisateurs finaux une fonctionnalité logicielle est une promesse qui a été survendue ces dernières années en particulier par les adeptes de l'agilité.

Malheureusement, la promesse n'a été que trop peu souvent tenue et ce n'est pas faute de mauvaise volonté. Pour réussir le pari du time to market il est nécessaire d'agir sur toutes les activités du processus, tandis que les méthodes agiles ne se sont concentrées que sur la moitié du problème : les développements.

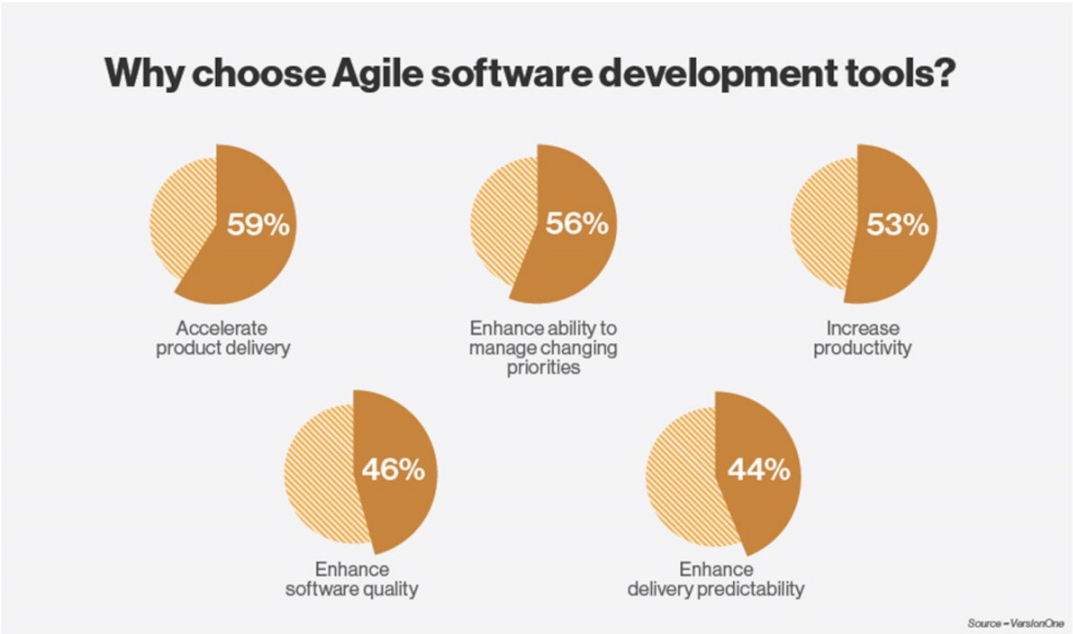


Fig. 7.2 *Le time to market, l'atout charme de l'agilité*

La démarche DevOps semble donc être la plus à même de réellement réussir ce challenge, mais pour cela encore faut-il comprendre les fondamentaux de la démarche afin de ne pas répéter la même erreur.

- DevOps ne renie pas les méthodes agiles, au contraire DevOps en tire parti ;
- DevOps vise à accélérer le processus de mise en production en améliorant la collaboration de manière agile et automatisée, mais ce n'est pas suffisant ;
- DevOps implique aussi de faire évoluer les processus de production et la gestion des infrastructures pour prendre en compte les impacts de mises en production plus

nombreuses et plus rapides.

Si vous oubliez l'un de ces trois aspects, les mêmes maux auront les mêmes effets : ce que vous gagnerez dans une partie du processus, vous le perdrez dans une autre. Le plus triste n'est pas dans la promesse non tenue, mais dans la frustration que cela génère ensuite avec les partenaires métiers. Et ce type de frustration n'est pas de nature à améliorer l'image des directions informatiques auprès d'eux.

7.2.2. Une démarche construite pour tenir cette promesse

A contrario, réussir une démarche DevOps permet de tenir réellement cette promesse. En effet, réussir à aligner la fréquence des releases sur celui de l'intégration des développements logiciels est un pré-requis fondamental mais également réducteur. Il ne suffit pas d'accélérer la fréquence des mises en production pour réussir à réduire significativement les délais. Il faut également être en capacité de gérer toute son infrastructure qui évoluera forcément de manière importante avec des releases qui passent de quelques-unes par an à plusieurs fois par minute : négliger ce changement et vous ne serez pas capables de tenir cette promesse.

De la même façon, il vous faut gagner en agilité face à l'obsolescence pour pouvoir prendre en compte de manière continue dans le temps ces changements que vous opérez, faute de quoi, votre accélération ne sera qu'éphémère. Enfin, il ne faut pas oublier de mieux tendre l'oreille aux feedbacks et d'industrialiser leur traitement car l'accélération du time to market n'a de sens que si les services proposés sont alignés sur les besoins, car autrement, vous accéléreriez à contre temps…

Pour résumer, une démarche DevOps permet de réellement réduire le *time to market* en travaillant sur trois axes majeurs : la collaboration et l'écoute entre les individus et le marché, la capacité à augmenter la fréquence des mises en production et la lutte continue contre l'obsolescence des technologies.

C'est autour de cette promesse et de leur capacité à l'honorer que les démarches DevOps vont bâtir leur réputation, positivement ou négativement.

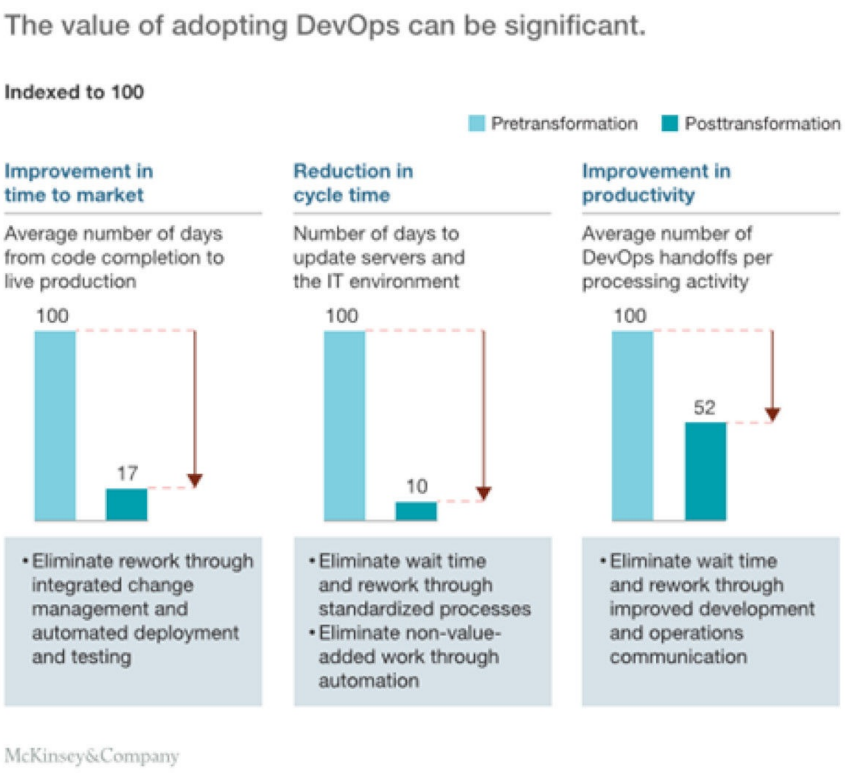


Fig. 7.3 *DevOps une démarche construite pour tenir la promesse de l'accélération du time to market*

7.3. Optimiser et rationaliser les coûts de l'IT

7.3.1. Un accélérateur plus qu'une solution

La recherche de l'optimisation et de la rationalisation des coûts est une démarche permanente et continue qui n'a pas attendu DevOps et dont DevOps n'est d'ailleurs qu'un accélérateur et non une solution en soi.

En réalité, DevOps vise d'abord à améliorer et pérenniser une collaboration qui fonctionne. De prime abord, cela peut paraître ne pas être en rapport avec une recherche d'optimisation des coûts telle que nous pouvons la concevoir de manière classique. Néanmoins, on comprend facilement qu'une collaboration qui fonctionne est préférable en termes de rationalisation des coûts qu'une collaboration bancale et cela quelle que soit l'ampleur des moyens mis à disposition.

En partant de ce constat simple, une collaboration optimisée dans une démarche DevOps est déjà un facteur de rationalisation important des coûts de l'IT.

Le coût ou le retour sur investissement d'une démarche DevOps est régulièrement perçu comme un frein alors que c'est en réalité un accélérateur à la mise en œuvre de cette démarche.

Cela est d'ailleurs confirmé par une étude IDC de novembre 2015 où le coût est perçu comme la première difficulté pour se lancer dans une démarche DevOps par 48 % des entreprises étudiées alors que ce n'est confirmé que par 36 % des entreprises qui se sont véritablement engagées dans cette démarche.

7.3.2. L’automatisation ou la réduction mécanique du coût

Il est un autre paramètre que l’on appréhende facilement dans la capacité d’une démarche DevOps à optimiser les coûts : c’est sa capacité à automatiser. En effet, une collaboration qui fonctionne optimise en soi la productivité mais automatiser cette collaboration permet d’actionner le levier direct de la réduction du coût.

Une meilleure productivité pour moins cher ? Cela semble relever du miracle mais il n’en est rien. La mise en œuvre de DevOps se fait sur le long terme en se forgeant des convictions fortes et une capacité d’évolution généreuse. Il ne s’agit pas d’aborder la question sous l’angle de considérations purement financières. Il est cependant certain que cet investissement n’est pas vain. L’automatisation est la pierre angulaire de la démarche, il faut néanmoins savoir où et comment automatiser pour atteindre les objectifs attendus.

Nous l’avons déjà précisé, l’automatisation dans DevOps ne signifie pas immédiatement une recherche d’optimisation et d’industrialisation. Il faut savoir s’appuyer sur ce qui fonctionne, automatiser ce qui apporte de la valeur rapidement et itérer ensuite pour optimiser.

C’est à ce prix qu’une démarche DevOps est réussie et c’est de cette manière que DevOps contribue à l’optimisation et la rationalisation des coûts sur le long terme. Mais ne nous y trompons pas, il s’agit d’une transformation majeure, nécessaire pour l’avenir, mais qui nécessite la conduite du changement adaptée à toute transformation majeure.

Enfin, il faut conclure sur un conseil majeur : ne pas initier une démarche DevOps dans le seul but de faire des économies. Les économies viendront à n’en pas douter, mais il faudra tout d’abord investir et convaincre pour ne pas se laisser distancer par les concurrents.

Avec DevOps, il faudra d’abord investir et construite sur le long terme : la valeur de DevOps est bien plus importante dans tant d’autres domaines qu’il serait réducteur et dangereux de se limiter à un débat purement budgétaire.

7.4. Réduire le MTTR et augmenter le MTBF

7.4.1. La qualité dans les gènes de DevOps

La principale raison avancée par ceux qui souhaitent mettre en œuvre une démarche DevOps est d’accélérer les rythmes de livraison. Pourtant, on oublie souvent qu’au début de l’application des principes de l’agilité au monde de l’informatique, il ne s’agissait pas uniquement de gains de vitesse mais aussi de gains de qualité. L’agilité est née autant pour gagner en qualité que pour gagner en vitesse d’exécution, et les démarches DevOps héritent pleinement de ce paradigme.

Comme nous l’avons vu, la mise en place de rythmes de livraison rapides participe à une amélioration de la qualité puisque dès lors qu’un bug est découvert, il peut être rapidement corrigé. Pour autant, il reste impératif de livrer un produit logiciel comportant le moins possible de bugs, si l’on souhaite pouvoir conserver ses clients. Différents mécanismes permettent de le garantir.

7.4.2. Les mécanismes à mettre en œuvre

Dire que la qualité est un enjeu fondamental de la démarche est une chose, comprendre en quoi cette démarche permet de réduire le temps de résolution des incidents tout en réduisant la fréquence de ces incidents dans le temps en est une autre.

Plusieurs mécanismes concourent à y parvenir :

- **La fréquence des mises en production** est un premier axe, car il permet effectivement plus de réactivité face à une anomalie, mais seul, il est impuissant à garantir le succès.
- **L'automatisation des tests et la clarification de la stratégie de tests** sont les fondements de la qualité dans la démarche, car c'est à travers ces mécanismes automatisés que l'on va matérialiser l'ambition de qualité que nous souhaitons réellement atteindre.
- **La capacité à capturer les besoins des utilisateurs** et à les retranscrire dans des cas d'usages pertinents, car c'est le cœur de l'implication métier et donc de votre capacité à faire ressentir cette qualité dans le système de valeur des utilisateurs.
- **La gestion de l'obsolescence des infrastructures** car c'est uniquement à travers une gestion irréprochable dans le temps que la qualité obtenue se maintiendra.

À cela s'ajoute des accélérateurs comme l'automatisation des releases ou de la gestion automatisée du provisioning des environnements. On peut considérer à juste titre que ces quatre mécanismes jouent un rôle fondamental dans la réduction des indicateurs MTTR et l'augmentation du MTBR.

7.4.3. Un challenge collaboratif autour de ces indicateurs

On remarquera que l'évolution positive de ces indicateurs vient de la collaboration entre Dev, Ops et Business. Comment augmenter les mises en productions ou penser à les automatiser sans celle-ci ? Cette collaboration est une absolue nécessité et son impact sur les indicateurs est indéniable.

Fig. 7.4 *Les gains significatifs d'une démarche DevOps sur le MTTR (source : Puppet Labs 2015 State of DevOps Report)*

L'automatisation des tests en collaboration Dev et Ops est un facteur d'augmentation du MTBF moins évident mais non moins important. La capacité à fournir le niveau de tests suffisants pour rassurer les Ops est fondamentale, il est donc tout aussi important d'agir en transparence et de construire cette stratégie de tests ensemble. Une stratégie de tests bien construite et automatisée au moins partiellement a un impact direct sur le MTBF.

L'implication du Business et la capture de ses besoins fondamentaux vont également permettre de focaliser les efforts de tests et qualité sur les scenarii réellement utiles au métier et aux utilisateurs. Leur perception sera meilleure même avec une moyenne de bugs identique par release. Les anomalies seront moins critiques et l'impact sur le MTBF et le nombre d'incidents sera significatif.

Tous ces efforts ne peuvent être maintenus dans le temps sans une gestion rigoureuse de l'obsolescence et les mécanismes innovants de mise à jour continue des infrastructures, qu'une démarche DevOps outillée peut aider à mettre en place.

7.5. Réussir l'amélioration continue des applications

Avant de proposer des solutions qui permettent de réussir l'amélioration continue des applications, il faut comprendre ce qui est concrètement derrière cette expression. On peut en effet envisager que les applications, s'améliorent continuellement de version en version. Que ce soit les systèmes d'exploitation ou les applications bureautiques, les logiciels au sens général n'ont jamais cessé d'évoluer en s'améliorant, du moins en théorie, d'une version à la suivante.

L'amélioration continue ne vise donc pas forcément à casser cette mécanique mais comme souvent avec une démarche DevOps, l'objectif sera plutôt de l'accélérer et de l'optimiser. En réalité, DevOps va plus loin qu'une simple accélération, elle vise à rester en temps réel en phase avec les besoins des utilisateurs voire à les anticiper.

Une démarche DevOps axe donc sa stratégie d'amélioration continue autour de trois principes :

- la capacité à absorber le changement en continu pour rester au plus proche des utilisateurs ;
- les moyens donnés aux utilisateurs pour s'exprimer directement, forme de démocratie directe du monde informatique ;
- et enfin la mesure implicite des comportements pour pouvoir les anticiper.

7.5.1. L'amélioration continue par la capacité à prendre en compte le feedback

Cette notion unique de *prise en compte des retours utilisateurs* et de *gestion du changement dans les spécifications d'un produit ou d'un service* est issue du monde des méthodes agiles. Elle est fondamentale. Il est bien peu utile de prévoir une accélération de tout le cycle de livraison et de toute l'infrastructure, du poste de travail au serveur, si tout est prévisible à long terme. Si tel était le cas, il serait opportun d'anticiper cette évolution.

Mais le monde s'est accéléré, et aujourd'hui bien malin est celui qui saura prédire les usages des produits et services informatiques de demain. Toutefois, s'il est impossible de prédire avec certitude l'avenir, il reste envisageable de percevoir les tendances et de s'adapter au changement.

Une démarche DevOps vise donc à la fois l'adaptabilité des développements logiciels, mais également celle des infrastructures liées et celles du poste de travail. Lorsque toutes ces composantes seront suffisamment agiles et alignées pour fonctionner de concert, alors il existera une vraie capacité à aborder efficacement le changement et à le transposer dans les produits et services quasiment en temps réel.

Pour y parvenir, il faut naturellement que grâce à des méthodes agiles, il soit possible d'introduire des changements dans les plans de développement des produits. Une fois en production, ces services doivent être capables de gérer une infrastructure incertaine avec différentes configurations de serveur. Ils doivent être capables de faire seuls les meilleurs choix pour l'utilisateur tant en termes de capacité d'exécution que de versions du logiciel. Autrement dit, le service sera capable de choisir le serveur le plus à jour et celui dont la configuration est la plus adaptée à son exécution tout en étant capable de choisir la version à exécuter. En cas d'anomalie, le service sera donc capable de s'adapter en toute transparence pour l'utilisateur.

Ce type de mécanisme ne peut voir le jour qu'avec une démarche de collaboration poussée entre les utilisateurs et l'informatique des développeurs et de l'infrastructure. Cela comprend également la gestion d'un poste de travail intelligent capable de charger les bonnes versions de ces services en fonction des profils.

C'est seulement avec la mise en place de cette démarche que les changements pourront faire l'objet d'une gestion continue. Encore faut-il savoir ce qu'il est pertinent de changer en sachant écouter les utilisateurs.

7.5.2. L'amélioration continue par la prise en compte du feedback explicite

Si nous souhaitons écouter les utilisateurs, il est nécessaire de mettre en place des mécanismes pour les écouter. Ces mécanismes sont des fondamentaux d'une démarche DevOps encore plus que dans les méthodes agiles.

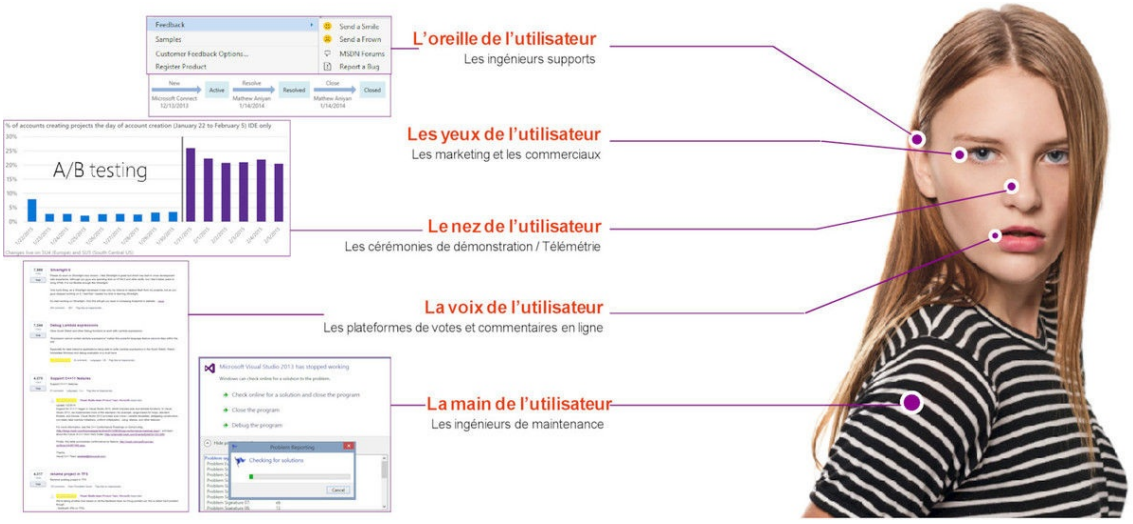


Fig. 7.5 *Les canaux de feedback du DevOps orienté utilisateur*

En effet, les méthodes agiles se focalisent sur une cérémonie dédiée à la prise en compte du feedback : les démonstrations. Une démarche DevOps va plus loin en combinant plusieurs canaux de prise en compte de la voix des utilisateurs. Nous pouvons citer, sans que cette liste soit exhaustive, les exemples suivants :

- **La cérémonie de démonstration** : issue des méthodes agiles, elle est évidemment conservée et encouragée. Le plus souvent, il s'agit d'un mécanisme de feedback indirect, les utilisateurs présents lors de cette démonstration étant souvent délégués pour représenter les utilisateurs finaux.
- **L'intégration du support** : les équipes opérationnelles en charge du support d'un logiciel sont des experts des besoins des utilisateurs sur ce périmètre fonctionnel. Ne pas les prendre en compte c'est se priver d'une richesse importante. Il s'agit là encore d'un mécanisme de feedback indirect, les ingénieurs support étant des relais des utilisateurs.
- **La maintenance** : les équipes en charge de la maintenance et de la gestion des problèmes en support ont une expérience et un historique important sur le périmètre fonctionnel du produit ou du service qui est particulièrement utile. Mais il s'agit encore une fois d'un mécanisme de feedback indirect.
- **Le marketing et les commerciaux** : intégrer les forces commerciales peut sembler évident. Ils sont face aux clients qui pointent les faiblesses de leurs argumentaires. C'est d'ailleurs sans doute le mécanisme de feedback le moins indirect de notre liste.
- **Les plates-formes de votes et commentaires en ligne** : réussir à redonner la parole aux utilisateurs dans leur diversité est un vrai défi. Des dispositifs visent à créer un lien le plus direct possible entre les équipes en charges du service et les utilisateurs. Pour parvenir à créer une dynamique constructive et exploitable de part et d'autre, il est nécessaire de structurer cette collaboration. Les plates-formes qui permettent aux utilisateurs de proposer des améliorations, d'en débattre et de voter selon des règles simples mais rigoureuses sont aujourd'hui une des meilleures réponses à cette problématique. Rien n'empêche de trouver quelque chose de plus innovant à l'avenir et qui réponde au même objectif. Il s'agit d'un mécanisme de feedback direct.
- **Les partenariats privilégiés** : les partenaires sont des utilisateurs privilégiés qui ont de fait un accès direct aux décideurs qui président aux destinées du produit ou du service. Le choix de ces partenaires est fondamental pour obtenir un échantillon représentatif de l'ensemble des utilisateurs et des retours pertinents, au nom du plus grand nombre. Il s'agit d'un mécanisme de feedback direct par représentation, un peu à l'image d'un sondage.

L'ensemble de ces mécanismes vise clairement à rapprocher l'utilisateur des concepteurs du produit ou du service. Les mécanismes de feedback direct sont les plus compliqués à mettre en œuvre car il faut faire adhérer les utilisateurs et donc animer les communautés qui peuvent se créer. Elles nécessitent un investissement plus important, mais lorsqu'elles fonctionnent ce sont également les plus riches d'enseignements.

Cette prise en compte du feedback de manière explicite et la qualité des données ainsi recueillies est absolument indispensable, si nous souhaitons ensuite imaginer des modèles

de prévision permettant d'anticiper des usages que les utilisateurs n'imaginent même pas eux-mêmes. Mais pour y arriver, il faut également intégrer la notion de feedback implicite et en tirer des tendances.

7.5.3. L'amélioration continue par la prise en compte du feedback implicite : la mesure et la télémétrie

La prise en compte des retours des utilisateurs de manière directe ou indirecte est fondamentale mais ce n'est pas suffisant. Il faut être en capacité de garantir la qualité de ces retours tout en s'assurant de la fraîcheur de ces informations. L'ensemble de cette démarche peut donc se résumer en deux mots *contrôler* et *anticiper*.

Lorsque nous parlons de contrôler le processus de feedback, il faut l'entendre sans l'aspect répressif qui est associé en général à ce mot. Il s'agit seulement de s'assurer que le processus fonctionne comme prévu. Dans une démarche DevOps, l'accent est mis sur la responsabilisation des individus, il est donc inutile de sanctionner, mais encore faut-il être alerté lors d'un dysfonctionnement. Ce mécanisme d'alerte est d'autant plus important que la part d'automatisation peut être prédominante dans ce processus.

Tous les indicateurs de contrôles comme des objectifs sur lesquels ils sont bâtis sont partagés par l'ensemble des acteurs, de la direction métier à l'ingénieur informatique, quel que soit leur domaine d'expertise. La plus grande transparence est prônée dans la mesure du possible et des contraintes de confidentialité.

Mais même si tout fonctionne parfaitement, ce n'est pas suffisant. Il faut savoir exploiter cette information de façon rapide et intelligente afin d'anticiper au plus tôt les comportements des utilisateurs.

Pour y parvenir, il faut commencer par mesurer en temps réel les comportements actuels pour en avoir une parfaite connaissance. C'est donc principalement pour prévoir et non pour contrôler, que des mécanismes poussés de télémétrie sont mis en œuvre dans les démarches DevOps.

Dans la continuité des principes des méthodes agiles, il s'agit ici de comprendre ce dont l'utilisateur a réellement besoin et non nécessairement ce qu'il peut désirer parfois de manière éphémère. Mettre en œuvre de la télémétrie en temps réel sur les services ou les produits génère une quantité énorme de données à exploiter. Les mécanismes de big data et de machine learning sont d'une grande aide comme nous l'avons déjà expliqué.

Néanmoins, il faut garder à l'esprit que ce type de démarche est ambitieux et comme souvent dans ce genre de cas, c'est une bonne idée de revenir aux fondamentaux : mettre en œuvre sur un périmètre réduit dans un premier temps (principe incrémental), tâtonner pour trouver les bons modèles de données (principe itératif) et impliquer toutes les parties prenantes de manière plus ou moins importante (principe de collaboration). Avec ces principes, vous devriez être en mesure de démontrer rapidement la valeur de ce type d'approche à l'ensemble de l'entreprise.

7.6. Assurer la continuité de l'espace de travail

La perspective business d'une démarche DevOps s'étend à la prise en compte de l'environnement global de fonctionnement de l'entreprise. Nous verrons plus loin l'exemple de Netflix, qui fait le choix de regrouper au même endroit l'ensemble de ses collaborateurs. Cet environnement doit offrir une certaine continuité.

Pour bien aborder le sujet de la continuité de l'espace de travail, il faut distinguer la différence entre *l'espace de travail* en lui-même et le *poste de travail* qui en est le support. L'un est attaché directement à l'utilisateur et a vocation à être unique tandis que l'autre peut lui être propre, mais ne sera pas unique, indépendamment de toute temporalité. Il peut même être partagé.

Seulement, l'un ne fonctionnant pas sans l'autre, il faut pouvoir assurer la disponibilité de l'un et de l'autre en un même temps. Les démarches DevOps visent à faciliter principalement la continuité du poste de travail.

Il est important de commencer par définir l'objectif qui est sous-entendu par l'expression *continuité du poste de travail*. Nous considérerons que cette expression vise à disposer d'un poste de travail fonctionnel en tant que support physique. Plusieurs machines sont donc considérées pour cette section comme des postes de travail distincts.

La continuité du poste de travail vise à pouvoir disposer en continu d'un poste de travail fonctionnel et à jour.

Le principe même de continuité suppose que l'harmonie entre toutes les parties prenantes numériques du poste de travail est préservée en permanence. Cet objectif idéaliste est quasiment impossible à mettre en œuvre dans le monde réel mais il existe des mécanismes basés sur la philosophie DevOps qui permettent de tendre vers cet idéal.

Le premier d'entre eux est la collaboration, indispensable pour préserver au maximum une certaine harmonie. Ces mécanismes peuvent être (et seront très certainement) automatisés en tout ou partie.

Mais le plus important est avant de changer d'état d'esprit pour adopter celui des démarches DevOps : l'harmonie ne pourra être préservée qu'à condition de se donner les moyens de la rétablir au plus vite lorsqu'elle sera perturbée.

Pour y parvenir, il faut mettre en œuvre plusieurs éléments, dont la plupart, pris séparément, sont familiers à tous les gestionnaires des postes de travail :

- **Des lots et itérations de déploiement**. Au-delà de la phase de recette classique, le parc en production sera divisé en lots dont la taille dépendra du groupe utilisateur qui lui est lié. Chaque lot sera mis à jour séparément dans un ordre prédéfini. Chaque mise à jour est indépendante quel que soit son périmètre (système d'exploitation, composant technique, applications…) et suit son propre cycle.
- **Des groupes utilisateurs**. Tous les lots du parc en production doivent être rattachés à des utilisateurs et suivre un ordre précis selon deux critères.
    - Le premier est la criticité du poste des utilisateurs : plus les dysfonctionnements potentiellement introduits sont critiques pour l'activité de l'entreprise, plus le

poste de travail est inclus dans un lot mis à jour tard. Il est donc indispensable d'introduire une échelle de risques pour évaluer les machines.
    - Le second critère est quantitatif, il s'agit de la taille des groupes utilisateurs. Plus le groupe utilisateur est important, plus le lot qui lui ait rattaché est mis à jour tard. Autrement dit, les premiers lots doivent être rattachés à des groupes utilisateurs dont le nombre est limité et dont les postes ne sont pas considérés comme critiques.

- **Un store d'entreprise**. Le magasin d'applications n'est pas une nouveauté en soi mais son usage devient central dans une démarche DevOps appliquée à la gestion du poste de travail. En effet, le *store* partagé de l'organisation est le lieu de mise à disposition privilégiée des applications mais également de l'ensemble des mises à jour liées au système d'exploitation ou ces composants. Rien n'empêche d'avoir plusieurs stores ou de centraliser différemment les différentes updates. Toutefois, leurs stratégies devront être alignées. En effet, le store d'entreprise doit pouvoir gérer plusieurs versions d'une même application, en même temps que les droits liés aux groupes utilisateurs et aux utilisateurs eux-mêmes. De cette manière, il saura distribuer la bonne update de manière adéquate pour le bon lot et le bon utilisateur.
- **Un mécanisme de *rollback*.** Les mécanismes induits par les démarches DevOps prennent en compte la possibilité qu'une mise à jour du poste de travail rende un lot donné inopérable. L'objectif est évidemment que ce lot soit détecté le plus tôt possible et donc impacte un minimum d'utilisateurs. Il est également évident que si cela se produit, généralement parce que la phase de recette a été déficiente. Néanmoins, dans ces rares cas, un mécanisme de *rollback* peut être le bienvenu, si possible de la façon la plus transparente possible pour l'utilisateur.
- **Un mécanisme de collaboration**. Qui dit DevOps dit forcément collaboration : le cœur de la démarche de continuité du poste de travail est le travail de collaboration pour remonter les anomalies détectées sur un lot avant son déploiement sur le lot suivant. Dans le même temps, ces anomalies doivent avoir été corrigées pour le lot où elles ont été découvertes, avant le délai imparti pour le déploiement sur le lot. La collaboration, tant pour la remontée d'information que pour la descente des corrections est fondamentale.

Ainsi, plaçons-nous du point de vue de l'utilisateur pour mieux comprendre ces nouveaux mécanismes de continuité du poste de travail. Si je suis un utilisateur identifié comme *précurseur*, je serai éligible très tôt aux différentes mises à jour. Prenons, pour l'exemple, une mise à jour fonctionnelle du système d'exploitation, fréquemment redoutée par les professionnels de l'informatique, car avec un fort impact potentiel sur les utilisateurs et les applications. Décrivons alors le scénario type lorsque les mécanismes DevOps sont mis en œuvre.

Je suis averti d'une mise à jour disponible pour le lot de machines dont mon poste fait partie. En passant par le store d'entreprise, je décide de faire la mise à jour proposée. En général, la phase de recette a permis de lever la plupart des difficultés pouvant se présenter durant l'installation et mon poste est mis à jour avec succès.

Malheureusement, le plus souvent après ce type de mise à jour et malgré une phase de recette réalisée le plus sérieusement possible, un certain nombre d'applications ne fonctionnent plus correctement, voire ne fonctionnent plus du tout.

En fonction du niveau d'impact des différentes anomalies détectées, un mécanisme de *rollback* sur l'élément mis à jour peut être activé, automatiquement ou non. Dans notre cas, il s'agit du système d'exploitation, mais la logique est la même pour une application, un service ou un composant.

Les processus DevOps appliqués à la gestion du poste de travail tirent parti des mécanismes de reporting autour des incompatibilités applicatives pour alerter directement les équipes en charge de la remédiation des anomalies rencontrées. Les informations nécessaires à la correction de l'anomalie sont donc réputées connues e, temps réel des équipes en charge de leur maintenance.

Le contrat de service autour de la collaboration doit engager l'équipe en charge de la maintenance à fournir le correctif dans le temps prévu pour le test du lot en cours et donc avant le déploiement sur le lot suivant. À moins d'un *rollback* fonctionnel, nous pouvons ainsi considérer que le poste de travail de l'utilisateur restera inopérant le temps nécessaire pour développer le correctif. L'engagement de livraison du correctif doit par conséquent être le plus court possible dans la limite des ressources disponibles. Un axe d'optimisation important peut parfois être découvert sur ce point au sein de l'entreprise.

Une fois le correctif développé, il est nécessaire de le déployer rapidement. Il doit donc suivre un processus de mise en production accéléré afin, ici encore, de limiter le temps d'indisponibilité du poste de travail. Il n'est pas impossible de disposer d'un processus de mise en production spécifique aux corrections liées à la mise à jour du poste de travail.

Une fois le correctif arrivé avec succès en production, il est disponible *via* le store d'entreprise qui centralise la distribution des services logiciels pour le poste de travail. C'est donc à travers le store que l'utilisateur va mettre à jour ses services désormais pleinement fonctionnels.

Le store d'entreprise doit donc être capable de proposer une mise à jour pour un lot donné de machines et non pour tous les utilisateurs. En effet, ce correctif pourrait introduire des anomalies non désirées pour tous les utilisateurs qui n'ont pas mis à jour le système d'exploitation.

Pour un utilisateur *précurseur*, il est particulièrement important que le temps nécessaire à la correction des services qu'il utilise soit le plus court possible.

Mais pour les utilisateurs du lot suivant, l'opération doit logiquement être beaucoup plus transparente. En effet, dès la mise à jour du système d'exploitation, le store de l'entreprise va proposer les correctifs identifiés sur le lot précédent et l'ensemble des services devraient fonctionner directement.

Autrement dit, en répétant ce mécanisme de collaboration DevOps sur plusieurs lots, la probabilité que des anomalies soient détectées sur le dernier lot qui est lié au plus grand nombre d'utilisateurs est particulièrement faible.

Mais assurer la continuité du poste de travail n’est plus suffisant aujourd’hui. Le cœur de la problématique est d’assurer la continuité du travail entre différents postes de travail, autrement dit, assurer la continuité des activités et des tâches sur plusieurs postes de travail ou devices.

En résumé

DevOps s’inscrit totalement dans la stratégie métier des entreprises et des dirigeants en tant qu’accélérateur indispensable à la réactivité de l’entreprise. DevOps est un facilitateur d’adoption des innovations technologiques qui permet de rationaliser les coûts et d’optimiser la valeur fournie par le système d’information à l’ensemble des forces vives de l’entreprise.

Adopter DevOps représente un vrai changement de culture en acceptant le risque d’erreur, a fortiori dans un contexte dans lequel les environnements sont en perpétuelle évolution. Pour pallier ces anomalies impossibles à anticiper, DevOps permet d’optimiser les indicateurs rois de la production : le MTTR et le MTBF.

Enfin, DevOps est un prérequis pour les scénarios particulièrement innovants et modernes de gestion du système d’information, pour les applications et pour l’espace de travail.

Chapitre 8

DevOps dans la vraie vie

DevOps est une approche, une culture, une philosophie. Mais DevOps est également un condensé de pratiques qui sont mises en œuvre concrètement dans le quotidien de certaines entreprises. Certaines sont emblématiques et leurs retours d'expérience font école dans la mise en œuvre des approches DevOps.

8.1. Facebook

Facebook est un service en perpétuelle croissance tant sur le plan fonctionnel, que sur la volumétrie. Les stratégies de développement et de gestion des opérations doivent s'aligner sur ce modèle et comme nous allons le voir, l'une des clés du succès de Facebook est liée à la mise en œuvre d'une démarche DevOps, dans laquelle, plus que la technique, la culture joue un rôle prépondérant.

8.1.1. Culture

Pour garantir la qualité à l'échelle d'un système aussi vaste que Facebook, la méthodologie et les outils ne sont pas suffisants, et Facebook se distingue aussi par une très forte culture d'entreprise.

Mais où sont passés les Dev et les Ops ?

Facebook Research publie de nombreux articles relatifs aux processus et outils mis en œuvre dans la chaîne de production du service Facebook. En complément d'une discussion avec un *Engineering Manager* de Facebook, le document *Development and Deployment at Facebook* constitue d'ailleurs notre principale source d'informations pour la rédaction de cette étude de cas. Dans les 14 pages de cette publication, il est intéressant de constater que le mot *developer* n'est employé que trois fois, tandis que le mot *engineer* est utilisé soixante-deux fois… Au final, même si ce document passe sous silence le rôle des équipes de gestion opérationnelle dans l'automatisation du provisioning et de la configuration, la gestion de la haute disponibilité, de l'allocation des ressources, il est évident que parmi ces engineers certains ont un profil plus *ops* que *dev* (et d'autres le profil inverse), en particulier dans l'équipe des *release engineers*.

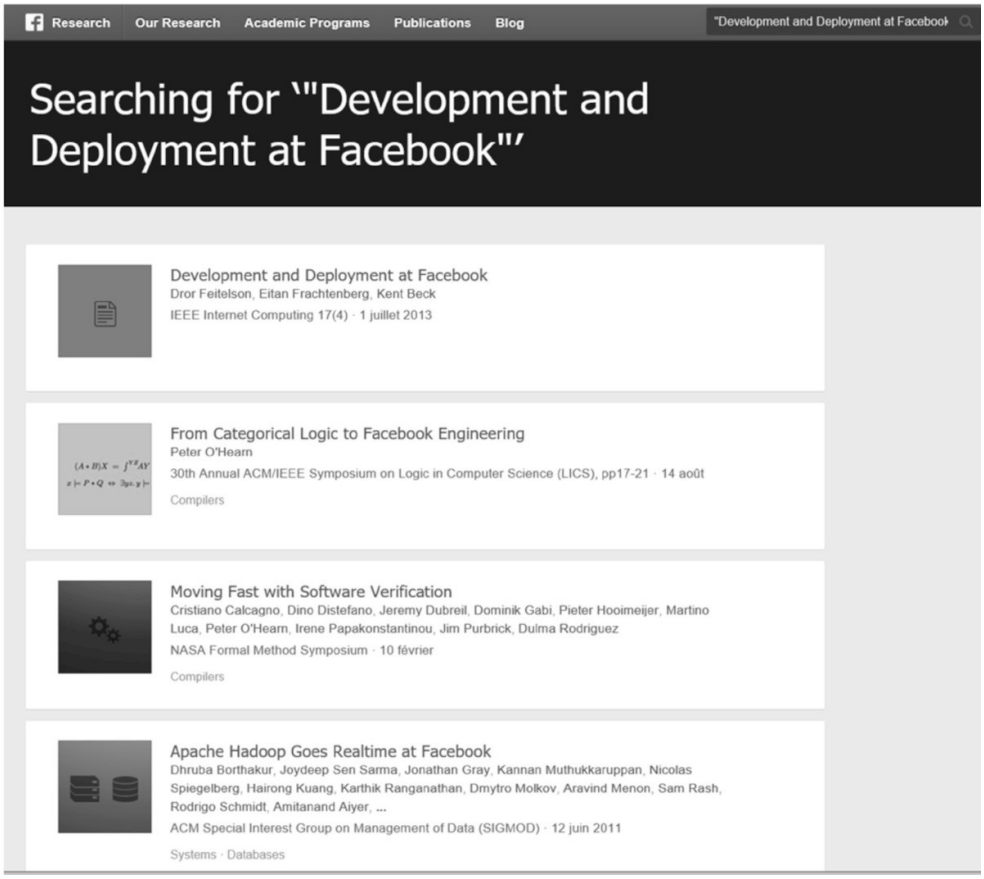


Fig. 8.1 *Facebook Research documente publiquement ses pratiques DevOps*

Bootcamp

Tous les nouveaux employés de Facebook participent à un bootcamp de six semaines avec pour objectif de se familiariser avec le code de Facebook, la culture et les processus. Dès cette première immersion, ils sont encouragés à publier leur code dès que possible, afin de les débarrasser au plus tôt de toute angoisse liée à la livraison d'un nouveau code.

Responsabilisation

La culture de la responsabilité personnelle est essentielle. Chaque ingénieur est responsable du code qu'il produit et, si nécessaire, du code d'un tiers qui serait à l'origine d'un dysfonctionnement. Les échecs sont traités comme une opportunité d'amélioration plutôt que comme un motif de sanction.

Ils sont eux-mêmes utilisateurs de Facebook, avec un usage plus intensif que le grand public et sont ainsi en situation de voir comment se comporte le système mis à jour. Cela leur donne également l'occasion de proposer de nouvelles idées pour faire évoluer le produit.

Expérimentation

À cela s'ajoute une culture de l'expérimentation y compris lorsqu'il s'agit de répondre à des défis majeurs. L'exemple le plus connu est certainement celui de l'exploration des trois alternatives permettant de répondre aux limites de performances rencontrées par PHP et son impact sur les coûts d'infrastructure. Au final, c'est la solution la plus audacieuse

(l'écriture du compilateur open source *just in time* HHVM (*HipHop Virtual Machine*) qui a permis de réduire la consommation mémoire et d'augmenter le nombre de requêtes traitées par seconde sans remettre en cause l'existant PHP.

Mobilité au sein des équipes

Ce sont les ingénieurs qui décident de l'équipe qu'ils souhaitent rejoindre, en fonction de leurs affinités et de leur valeur ajoutée au regard des priorités de l'entreprise. Un mécanisme baptisé *hackamonth* permet également de rejoindre une autre équipe pendant plusieurs semaines pour y travailler sur de nouvelles idées, ce qui favorise une mobilité ultérieure.

8.1.2. Développement

À la différence de produits logiciels encadrés par une date de livraison fixe et un périmètre fonctionnel limité, le service Facebook est en perpétuel développement.

Développement continu

En juillet 2013, le code de la partie frontale comptait déjà plus de 10,5 millions de lignes et son taux de croissance était linéaire en fonction du temps. Des centaines de développeurs publient leurs mises à jour sur le contrôleur de code source jusqu'à 500 fois par jour, en enregistrant les modifications sur plus de 3 000 fichiers. La plupart des fichiers source sont modifiés par seulement quelques ingénieurs.

Stratégie de gestion de code source

Les développeurs de Facebook publient leur code directement sur la branche principale ce qui évite les contraintes liées la fusion des branches. Néanmoins, il y subsiste une distinction entre le code en cours de développement et le code prêt à être déployé. Les développeurs utilisent le système de contrôle de version Git local pour leur production quotidienne. Lorsque leur code est prêt à être publié, il est d'abord fusionné dans la branche principale du référentiel centralisé, puis les modifications sont publiées pour le déploiement.

Si une fonction à implémenter requiert d'importantes modifications, elle fait l'objet d'une décomposition en une séquence des changements plus limités, dans un modèle baptisé *branche par abstraction*. C'est d'ailleurs cette approche qui a permis aux équipes de Facebook de migrer des tables de base de données contenant des centaines de milliards de lignes sur de nouveaux supports de stockage, sans que les utilisateurs ne s'en aperçoivent…

8.1.3. Déploiement

Le perpétuel développement du service Facebook est couplé à un processus de déploiement quotidien et hebdomadaire.

Déploiement continu

Par défaut, le code est publié chaque semaine et intègre des milliers de changements. Cela se traduit par une mise à jour rapide couplée à un contrôle précis sur les versions et configurations. Le dimanche, le *latest build* est déployé sur un environnement supervisé par les ingénieurs responsables de la release et soumis à des tests automatisés d'interface et à des tests de performance.

Ce code devient alors la version par défaut utilisée par les employés de Facebook : la nouvelle version du service peut s'exécuter dans des conditions proches du réel, et il est ainsi possible de détecter d'éventuels dysfonctionnements au plus tôt.

La publication à l'échelle de la planète a lieu le mardi après-midi. Les ingénieurs ayant contribué au code doivent être joignables au cours du déploiement. Le système le vérifie en communiquant avec eux automatiquement à l'aide d'un système de bots IRC ; en cas d'absence, la mise à jour de leur code ne sera pas publiée.

Revue de code

Les ingénieurs responsables de la release assignent un niveau (interne) de *push karma* aux développeurs en fonction des précédents déploiements de leur code, afin de mieux gérer le risque et d'anticiper les efforts à prévoir pour la revue du code, qui chez Facebook est fondamentale.

En effet, toute ligne de code est examinée par un développeur différent de celui qui l'a produite, ce qui permet ainsi d'en garantir doublement la qualité. Pour ce faire, ils utilisent une suite d'outils open source qui facilitent le contrôle des évolutions du code et le suivi des bugs. Comme il est impossible de se prémunir contre tout dysfonctionnement, il faut optimiser l'usage des ressources liées au contrôle, en gérant les priorités. Par exemple, le code ayant un impact sur les garanties de respect de la vie privée est considéré avec beaucoup plus d'attention que le code qui traite de questions moins sensibles.

8.1.4. Qualité

La capacité de déployer du code fréquemment et avec de petits incréments facilite l'innovation et la validation de la qualité du service. À cela s'ajoute la mise en œuvre de différents outils associés à des pratiques telles que le test en production, le contrôle de la qualité par groupe d'utilisateurs ou l'A/B testing.

Supervision

Après le déploiement, les ingénieurs surveillent attentivement le comportement du site afin d'identifier tout signe de détresse. Dès lors qu'un problème survient, il est plus facile de l'identifier, car il est probable qu'il soit lié à une récente évolution du code.

Le système est supervisé avec des outils internes tels que Claspin, un outil de monitoring qui permet d'afficher sur une *heatmap* l'état des systèmes observés dans un format qui facilite la détection des problèmes ou de tendances. Sur un seul écran, avec un système de codes de couleur, il permet ainsi de suivre, en temps réel, l'évolution d'une trentaine de paramètres, sur plus de 10 000 serveurs. Techniquement cela suppose la collecte d'une

grande quantité de données, et leur mise en forme dans une combinaison de JavaScript et de SVG. À cela s'ajoutent des sources externes, telles que l'analyse de tweets.

Le service Gatekeeper

Déployer le nouveau code n'implique pas nécessairement qu'il soit immédiatement disponible pour les utilisateurs. Facebook utilise un Gatekeeper qui contrôle le degré de visibilité des évolutions du code en activant ou non les nouvelles fonctions (*Toggle Features*) selon l'appartenance des utilisateurs à différents regroupements basés sur des critères tels que le groupe d'âge ou de pays.

Il permet également de désactiver un nouveau code qui pose des problèmes, ce qui réduit la nécessité de déployer immédiatement une correction. Enfin, il permet de déployer l'implémentation d'un nouveau service sans l'exposer aux utilisateurs finaux (*dark launch*) afin de tester son impact sur l'évolutivité et les performances.

A/B Testing

Le déploiement continu favorise les tests en production mettant en œuvre de l'A/B testing. Les ingénieurs peuvent ainsi étudier, en comparant avec le scénario de référence, comment les nouvelles fonctions affectent le comportement de l'utilisateur. Même avec un sous-ensemble d'utilisateurs, les volumétries liées à l'usage de Facebook font qu'ils disposent rapidement de gros volumes de données pour procéder à cette étude d'impact. Cela permet également de mieux connaître la diversité des utilisateurs et de pouvoir faire évoluer le service en fonction de l'usage qui en est fait.

8.2. Netflix

Netflix est aujourd'hui l'un des plus grands sites fonctionnant presque exclusivement sur une infrastructure de cloud public. Le succès de cette entreprise dépend très largement des capacités de son service à évoluer et à s'adapter à la demande, ce qui suppose la mise en place de processus et d'outils s'inspirant de principes DevOps associés, là encore, à une forte culture d'entreprise.

8.2.1. Culture

La culture de Netflix est une culture fortement orientée développeur, sans doute parce que leur CEO, Reed Hastings, a été le fondateur et le développeur de Purify, un outil de debugging bien connus des développeurs C/Unix. Le principe est de privilégier la performance en éliminant tout obstacle.

Collaboration des équipes

Netflix concentre toutes ses ressources humaines au même endroit, quitte à investir dans des nouveaux locaux plus grands, si cela s'avère nécessaire. L'objectif est de favoriser la communication entre les personnes et d'éviter de perdre du temps dans des synchronisations inutiles.

Les équipes sont généralement constituées de 3 à 7 personnes. Elles se réunissent une fois par semaine. La priorité est donnée à l'embauche d'ingénieurs ayant une forte expérience. Les coûts salariaux sont plus que compensés par leur productivité supérieure et leur autonomie qui permet de réduire les frais de gestion.

Les responsables d'un service ont le contrôle sur les ressources et définissent le contexte pour leurs équipes, avec une réunion hebdomadaire en 1/1 avec chaque ingénieur. Les responsables d'équipes sont les garants de la qualité des embauches. Ils doivent être suffisamment techniques pour avoir une vue claire de l'architecture et de la gestion du projet.

Responsabilisation

Il n'y a pas de comité d'architecture, ni de centralisation des normes de codage. En dehors de la recommandation de certains patterns, Netlix part du principe que les développeurs savent déjà comment être productifs. Les ingénieurs sont responsabilisés sur leur code, dont ils maximisent la qualité pour de leurs pairs.

Chaque équipe gère son propre calendrier de mise à jour. Les équipes sont directement responsables de la gestion de l'évolution de l'interface et de ses dépendances. Les développeurs publient directement sur la production.

8.2.2. Continous Delivery dans le cloud

Netflix a exploité au mieux les possibilités qu'offre le cloud, en responsabilisant les développeurs sur le déploiement et en automatisant tout le cycle de continuous delivery.

Déploiement et provisioning

Comme nous l'avons vu, les développeurs apportent des changements en production eux-mêmes, et doivent juste s'assurer de la traçabilité de leurs actions en production, en enregistrant un billet de gestion du changement. Ils peuvent créer et supprimer jusqu'à 1 000 serveurs par jour, juste en déployant un nouveau service.

Avec l'évolution de la plateforme Netflix vers Amazon Web Services, les développeurs et les responsables des opérations ont collaboré pour automatiser la construction d'une image AMI (Amazon Machine Image) basée sur Linux. En utilisant de multiples outils (Perforce pour le contrôle de version, Ivy pour la gestion des dépendances, Jenkins pour l'automatisation du processus de build et Artifactory comme référentiel binaire), ils ont ainsi pu s'assurer que chaque instance d'un service serait totalement identique pour une version donnée.

Outillage du processus de continuous delivery

L'équipe Cloud Operations Reliability Engineering (CORE) a pour mission de développer les outils pour automatiser les processus. Un de ces outils est la solution de continuous delivery Spinnaker, publiée en open source sur GitHub pour optimiser les modèles de déploiement vers le cloud. Il s'agit d'une évolution de la solution open source Asgard, une interface web pour le déploiement des applications et de gestion de cloud. Comme la

console de management AWS n'était pas adaptée aux opérations à grande échelle menées quotidiennement pas Netflix, l'équipe CORE avait construit un portail intégrant les concepts qu'AWS ne prenait pas directement en charge. Parmi eux, l'objet *Cluster* contenait un ou plusieurs groupes d'autoscaling Amazon, qu'il associait avec une convention de nommage. Lorsqu'un nouveau ASG était créé au sein d'un cluster Netflix, un numéro de version incrémenté était ajouté au préfixe du cluster pour former le nom du nouvel ASG. Cette approche permettait à Asgard d'effectuer un déploiement qui pouvait ainsi être rapidement annulé en cas d'incident : une solution plus coûteuse en nombre d'instances mais qui permettait de réduire la durée des interruptions de service de façon significative.

Spinaker reprend les fonctions Asgard de gestion de clusters et propose aujourd'hui un modèle d'extensibilité permettant de provisionner des environnements sur de multiples plates-formes (Amazon, Google, cloud Foundry et Microsoft Azure). Il facilite la création de pipelines représentant un processus de livraison qui peut commencer par la création de certains éléments à déployer (par exemple, une image de machine, un fichier Jar ou une image Docker) et se termine par un déploiement. Le processus peut être déclenché par l'exécution d'un job Jenkins, par un autre pipeline ou être planifié.

8.2.3. Qualité

La garantie de la qualité est assurée par la mise en œuvre d'outils développés en prenant en compte des principes DevOps comme la prise de risque et l'expérimentation comme l'illustre l'armée simienne de Netflix.

L'armée des huit singes

Puisqu'aucune composante d'une solution n'est capable de garantir 100 % de disponibilité, Netflix a conçu une architecture dans laquelle chaque composant individuel peut échouer sans affecter la disponibilité de l'ensemble du système. En complément de cette démarche Failsafe, Netflix a mis au point un outil permettant de régulièrement tester la capacité de la solution à survivre en cas d'échec : le *Chaos Monkey*, un service qui désactive aléatoirement les instances de machines virtuelles en production pour s'assurer que cela ne se traduit pas par une interruption de service pour les clients. En exécutant le Chaos Monkey pendant un délai limité au milieu d'un jour ouvré, dans un environnement attentivement surveillé par les ingénieurs capables de résoudre les problèmes, il devient plus facile d'apprendre sur les faiblesses du système et de mettre au point les mécanismes de récupération automatique.

À ce premier singe périodiquement lâché dans les datacenters sont venus s'ajouter d'autres primates, eux aussi investis de la mission de provoquer d'autres types de défaillances, ou de détecter des conditions anormales : *Latency Monkey* pour induire des délais artificiels dans les communications, *Conformity Monkey* pour identifier les instances qui ne respectent pas les bonnes pratiques, *Doctor Monkey* pour vérifier les données de santé sur chaque instance, *Janitor Monkey* pour faire le ménage, *Security Monkey* pour détecter les vulnérabilités, *10-18 Monkey* pour gérer les problématiques liées

à la localisation, *Chaos Gorilla* capable de provoquer une interruption de service étendue à l'ensemble d'une zone de disponibilité AWS.

Supervision

Cette périodique injection volontaire de défauts dans le système est complétée par une passerelle de supervision qui collecte les alertes provenant de plusieurs systèmes différents, qui les filtre et les envoie par email aux développeurs ou qui les route vers l'application PagerDuty afin d'assurer la gestion des escalades et la centralisation des appels.

Pendant une panne de production, l'équipe CORE fait toujours appel au développeur pour appliquer la modification. Les alertes concernent en général le flux de transactions commerciales (plutôt que des opérations de bas niveau sur Linux) et contiennent des liens vers des outils orientés Application Performance Management afin de permettre aux développeurs de rapidement voir où le problème se situe.

8.3. Microsoft

8.3.1. La culture agile et DevOps

Il est de notoriété publique que les groupes produits chez Microsoft ont une certaine indépendance dans leur organisation et leurs méthodes de travail. Certains en ont donc profité pour avancer plus rapidement que d'autres sur les méthodes agiles et les approches DevOps.

Aujourd'hui, il semble que la firme de Redmond cherche à harmoniser ces pratiques au sein des différents groupes produits et l'un d'eux semble inspirer l'ensemble de l'entreprise par son expérience acquise sur ces approches.

Nous nous concentrerons donc ici sur la présentation de quelques-unes des pratiques emblématiques du groupe produit Visual Studio.

8.3.2. La gestion des spécifications du produit

Les spécifications sont décrites selon les préconisations de l'agilité, c'est-à-dire sous forme de scénarios orientés utilisateur. Elles sont décomposées en plusieurs niveaux de détails allant des macro-fonctionnalités du produit jusqu'aux *user stories* et *test cases* :

- **Les fonctionnalités du produit ou *epics*.** Il s'agit des fonctionnalités principales du produit tel que nous pourrions les voir résumées sur l'emballage du produit. Ce sont les engagements commerciaux de l'éditeur et l'objet des annonces faites au grand public. Cela correspond aux objectifs du projet de développement par exemple.
- **Les scénarios du produit ou *use cases* de haut niveau.** Définis et arbitrés par les managers produits, ces scénarios déclinent les usages attendus autour des fonctionnalités du produit pour les rôles utilisateurs ciblés. L'implémentation de ces scénarios est alors confiée aux *feature teams* menées par un *program manager*.

- **Les Minimal Marketable Feature.** Il s'agit de la description des fonctionnalités minimales utilisées dans les scénarios qui ont été assignés à l'équipe. Elles constituent le backlog de l'équipe et l'élément visible qui sera livré dans le produit. Il fait l'objet d'un suivi quotidien par le program manager et suit en général les états suivants : On Deck, Ready, In Progress, Completed, Shipped.
- **Les user stories.** Entre les MMF et les tâches, il existe les User Stories. La différence est que les MMF apportent un incrément de valeur ajoutée à l'utilisateur end-to-end. Les MMF sont décomposées en Stories qui sont des incréments, mais ne constituent pas un *end-to-end*. En d'autres termes, si toutes les stories d'un MMF ne sont pas implémentées, il y a des trous dans l'expérience, des choses à faire à la main, etc. Notons que les stories peuvent être décrites de la forme *en tant que X je peux faire Y afin de Z*. Elles sont testables en soit.
- **Les tâches.** Il s'agit de la décomposition des étapes concrètes de réalisation des *user stories* concernées. La décomposition en stories est réalisée collégialement par l'équipe mais les tâches sont l'affaire de chacun. La complexité des tâches est également estimée.

8.3.3. La mise en œuvre de l'agilité

Avant d'adopter une approche DevOps, le groupe produit a d'abord cherché à mettre en œuvre les méthodologies agiles mais en souhaitant dès le départ en adapter les concepts à son contexte particulier.

Il en résulte une mise en pratique originale qu'il serait impossible de détailler à moins d'un livre dédié. Nous pouvons néanmoins tenter d'en synthétiser la substance en nous focalisant sur l'organisation du sprint et les cérémonies associées.

Il faut insister à nouveau sur le choix qui a été fait de laisser une grande liberté dans la mise en œuvre des pratiques retenues par le groupe produit. Chaque équipe choisit d'adapter les principes globaux retenus par la direction produit en fonction de ses propres besoins.

Il n'en reste pas moins un cadre, certes léger, qui est homogène pour tous. Ainsi, la durée des sprints est de quatre semaines pour tout le monde. Ces sprints sont décomposés en trois semaines à proprement parlé et une semaine dédiée à la *release*.

Les contenus des sprints sont également homogènes mais ils n'en restent pas moins originaux par rapport à la théorie classique des méthodes agiles. En effet, si les trois premières semaines sont dédiées à l'implémentation des *MMF*, la dernière semaine est dédiée à la qualité et au test. De plus, la première semaine d'implémentation est réservée aux activités de conception et d'architecture.

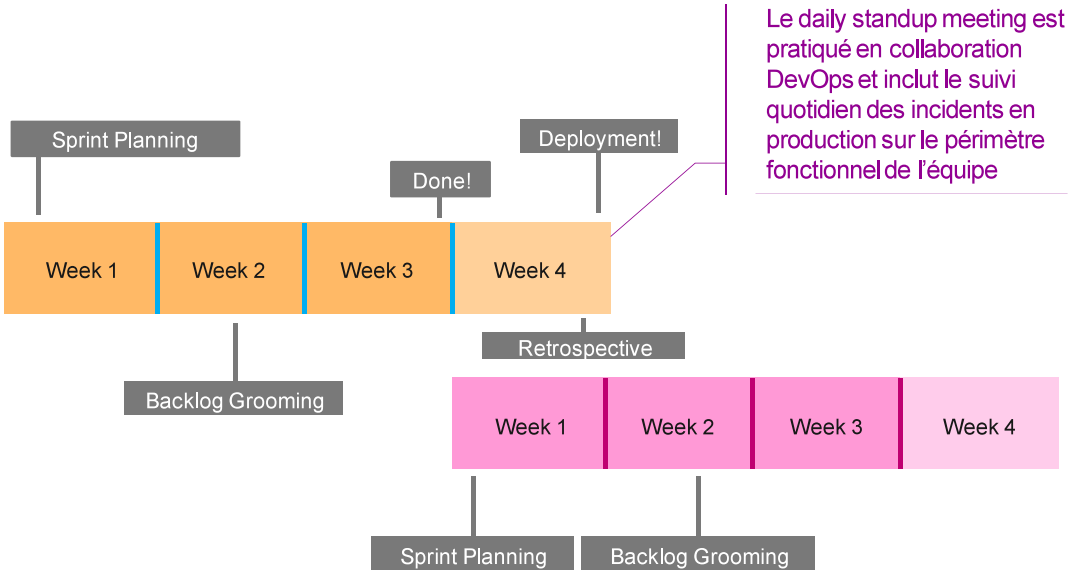


Fig. 8.2 *Les sprints et cérémonies du groupe produit*

Ces particularités permettent aux sprints de se superposer. En effet, pendant la semaine dédiée aux tests et à la qualité, le sprint suivant peut démarrer en parallèle sur les activités de conception et d'architecture.

Par ailleurs, l'ensemble des cérémonies majeures sont respectées : *daily stand-up meeting*, *sprint planning*, *backlog grooming*, rétrospective. Les équipes ont également dès le départ cherché à disposer d'une intégration continue efficace avec tous ce que cela comporte de nécessaire.

Pour conclure, il est important de noter que ces méthodes étant aujourd'hui appliquées dans le cadre d'une démarche DevOps, des adaptations ont été nécessaires, comme par exemple, les *daily meetings* communs avec les équipes de production.

8.3.4. La mesure de la performance

La particularité du groupe produit Visual Studio est de gérer à la fois un logiciel *en boite* et un logiciel *en SaaS* qui partagent en permanence leurs fondations.

La mesure de la performance qui est fondamentale dans une architecture logicielle *SaaS* a donc été portée naturellement vers l'ensemble des produits à l'architecture plus traditionnelle.

Il faut également prendre en compte que le tableau de bord et les KPI sont des éléments ancrés fortement dans la culture d'une entreprise comme Microsoft. De fait, de très nombreux indicateurs sont dans l'ADN du produit.

Pour faire simple, nous savons que les logs d'activités, les traces techniques, l'historique des activités, les compteurs de performances divers et variés, la disponibilité du réseau, les indicateurs standards d'infrastructure et les indicateurs d'usage utilisateur sont suivis quotidiennement par les équipes.

8.3.5. Les mises en production automatisées

Puisque le produit concerné bénéficie d'une double mise à disposition *online* et *offline*, le produit est mis en production de manière complètement automatisée *online* et c'est sa version distribuée la plus stable qui est ensuite distribuée en téléchargement.

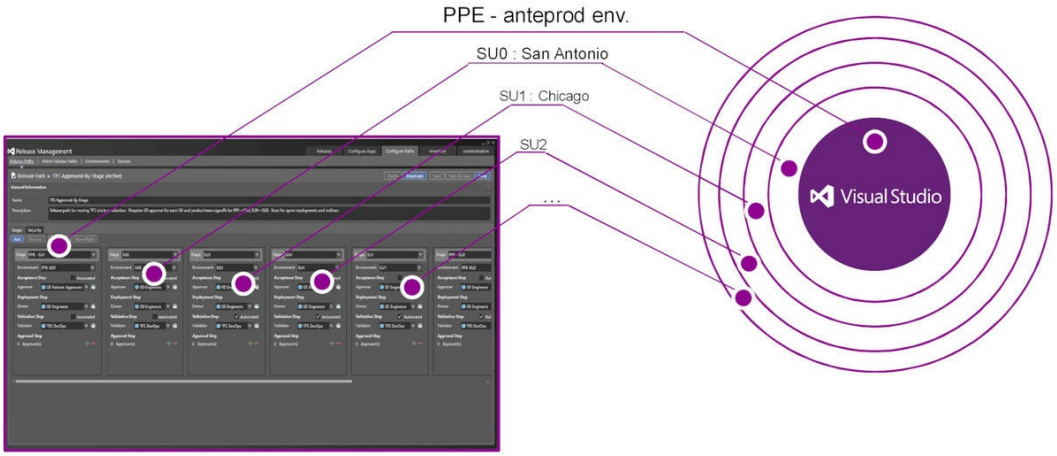


Fig. 8.3 *Les rings de déploiement automatisé du produit SaaS*

Le processus de mise en production est complètement orchestré par un des produits développés par les équipes *Release Management* de la suite Visual Studio. À travers cet outil, l'ensemble des activités techniques, du provisioning des machines dans le cloud et la répartition dans les différents datacenters est planifié et paramétré.

Désormais chez Microsoft, on retrouve une notion de *ring*. Ici, ces anneaux de déploiements correspondent à une quinzaine de datacenters du groupe répartis à travers le monde. La mise en production se fait successivement sur chacun des datacenters, les uns après les autres et avec quelques heures d'intervalle à chaque fois.

Si une anomalie importante est détectée sur l'un de ces datacenters, le déploiement est immédiatement arrêté et un mécanisme de correction est enclenché pour relancer ensuite le cycle de déploiement dès le départ.

Si l'anomalie est particulièrement impactante, un mécanisme de *rollback* existe mais présente l'énorme désavantage de faire perdre une partie des données non sauvegardées au moment où il se produit.

C'est d'ailleurs la raison pour laquelle le premier datacenter est réservé à l'hébergement des données du groupe produit lui-même et des *early adopters* interne à Microsoft. Au jour où ces lignes sont écrites, la procédure de *rollback* n'a été activée qu'une seule fois et sur cet unique datacenter.

8.3.6. La stratégie de tests et son automatisation

Pour réussir à déployer en continu, plusieurs fois par jour, des mises à jour vers les utilisateurs le groupe produit a automatisé la très grande majorité de ces procédures qualité.

Les tests sont donc répartis en quatre niveaux allant du *level 0* au *level 3* :

- Le *level 0* correspond principalement aux tests techniques de premier niveau, incluant les tests unitaires. Ceux-ci sont réalisés à chaque itération.
- Le *level 1* correspond aux *test cases* liées aux MMF concernés. Ceux-ci incluent les tests avec les dépendances techniques du produit ainsi que les tests liés aux données. Ils sont joués pour que le MMF passe vers un statut *Completed*.
- Le *level 2* est prévu pour réaliser les tests fonctionnels. Ceux-ci permettent de tester le service et ses dépendances. Ils permettent de valider une livraison des fonctionnalités produites.
- Enfin, le *level 3* correspond à l'ensemble des tests réalisés en production. La majorité des méthodes évoquées dans ce livre sont d'ailleurs mises en pratique.

8.3.7. La télémétrie et le pilotage par la donnée

La particularité d'un produit destiné au grand public, même s'il s'agit ici d'une tranche ciblée mais particulièrement importante, est de ne pas pouvoir avoir des *représentants des utilisateurs* suffisamment représentatifs pour pouvoir faire des démonstrations pertinentes.

Le groupe produit contourne cette problématique en s'appuyant énormément sur la télémétrie et sur les statistiques pour piloter *via* les données récoltées les évolutions pertinentes pour les utilisateurs.

Pour y parvenir, plusieurs mécanismes sont mis en œuvre, mais le premier d'entre eux est évidemment la remontée de données directement depuis le produit qui sont analysés entre *data scientists* et *program manager* quand ces deux rôles ne sont pas assignés à une unique personne.

Mais les données proviennent également de plusieurs autres sources d'informations : les réseaux sociaux qui sont régulièrement scannés, les forums de type *user-voice* du produit mais également les mécanismes d'expression intégrés directement dans le produit tel que *donner un feedback* ou *envoyer les informations du crash*.

8.3.8. L'organisation des équipes

L'organisation retenue par l'équipe produit est plutôt de nature hybride. Elle est composée de quatre entités hiérarchiques distinctes : les *program managers*, les *engineers*, les *quality managers* et les *operations*.

Chacune de ces entités est organisée avec plusieurs niveaux de middle management et un manager global pour l'ensemble du groupe produit.

Pourtant, une équipe dédiée à une fonctionnalité est consitutée avec un unique program manager et est constituée d'engineers et d'au moins un quality manager.

Les engineers regroupent indistinctement les développeurs et les personnes chargées de la qualité et des tests. Autrement dit, les développeurs testent et vice-versa. Il existe bien entendu des règles d'or tel qu'un *développeur ne valide pas les tests de ce qu'il a lui-même développé*.

Les quality managers sont des spécialistes de la donnée, des data scientists et statisticiens de très haut niveau.

Enfin, les opérations ne sont pas directement intégrées dans une équipe dédiée à une fonctionnalité mais fonctionne en hub de service avec des personnes identifiées comme les interfaces privilégiées des différentes équipes.

Chapitre 9

Quelques idées reçues sur DevOps

Il existe aujourd'hui autant de définitions de DevOps que d'acteurs sur le marché, chacun cherchant à l'orienter en fonction de son activité et ses intérêts. Cette richesse a fait émerger un certain nombre de contre-vérités qui ont fait naître des idées reçues pas toujours très heureuses.

Nous vous proposons de revenir sur les plus répandues et de partager notre point de vue que nous espérons cohérent avec l'ensemble des approches, démarches et pratiques DevOps que nous avons pu exposer.

9.1. DevOps remplace / est incompatible avec ITIL

De notre point de vue, DevOps est compatible avec ITIL et les pratiques ITSM (*IT Service Management*). Nous pensons même que DevOps est complémentaire d'ITIL.

En effet, ITIL apporte un cadre ou *framework* permettant d'industrialiser ses processus de production qui paraissent absolument nécessaires pour conserver une rigueur suffisante, surtout, lorsque le rythme de mise en production s'accélère considérablement dans le cadre d'une mise en œuvre des démarches DevOps.

ITIL en tant que receuil de bonnes pratiques n'est en rien opposé aux possibilités d'automatisation qu'apportent les démarches DevOps.

Il ne faut simplement pas tomber dans l'excès de formalisation en voulant mettre en œuvre tout le référentiel ITIL sans réfléchir à la valeur apportée par ces concepts. Le *just enough process* s'applique tout particulièrement dans notre cas.

9.2. DevOps remplace agile

Non seulement DevOps ne remplace pas les principes de l'agilité mais DevOps est totalement agile et en applique les principes, les utilise et les intègre complètement dans son approche.

Dans le même temps, les méthodes agiles ne sont pas un prérequis à une démarche DevOps, mais un minimum de pratiques agiles doit néanmoins être implémenté. Il ne s'agit pas nécessairement d'adopter une méthode agile en tant que telle mais de mettre en œuvre les pratiques qu'elle inspire dans votre contexte.

9.3. DevOps = automatisation

Si l'automatisation est au cœur du support technologique des approches DevOps et s'il s'agit selon nous de l'un des piliers de l'outillage de la démarche, ce n'est absolument pas l'élément fondamental de DevOps.

L’automatisation vise à supporter une collaboration pour la pérenniser et la diffuser. Si votre collaboration est déficiente, elle ne sera pas meilleure une fois automatisée. Le cœur de DevOps vise donc à rétablir des mécanismes de collaboration basés sur une confiance réciproque retrouvée.

La confiance est bien plus difficile à construire que l’automatisation et elle est bien plus importante.

Il est vrai que l’aspect automatisation intéresse beaucoup les éditeurs et les experts en différentes technologies, y compris open source, qui peuvent le supporter et qui visent de s’imposer sur ce marché. Mais là n’est pas le cœur de DevOps qui se veut avant tout une transformation culturelle.

Résumer DevOps à l’automatisation est donc une erreur même s’il ne faut pas négliger son importance dans la démarche.

9.4. DevOps = « infrastructure as code »

La notion d’*infrastructure as code* est le bras armé de l’automatisation dans l’esprit de ceux qui cherchent à résumer DevOps aux outils et technologies. Parce que DevOps est avant tout une approche et une culture et non un assemblage d’outils, nous ne pouvons que contredire cette affirmation.

La totalité des arguments différenciant DevOps d’une simple démarche d’automatisation peut de nouveau être appliquée.

Si *infrastructure as code* résume pour certains l’idée que l’on peut se faire de DevOps, elle ne reste qu’une pratique parmi tant d’autres, qui contribuent à cette démarche.

9.5. DevOps = open source

DevOps a connu un essor particulier dans la communauté open source qui l’a vite adopté, et a rapidement proposé des solutions technologiques permettant d’outiller les pratiques associées à la démarche.

Néanmoins, il existe de nombreuses implémentations réussies de pratiques et approches DevOps s’appuyant sur des technologies ou des produits commerciaux. DevOps ne peut donc se résumer au monde de l’open source.

Si les pratiques DevOps cherchent à automatiser des tests, des mises en production ou du monitoring par exemple, de nombreuses solutions technologiques peuvent être utilisées, voire combinées, inclure des solutions open source et des solutions d’éditeurs, voire des solutions issues de vos propres équipes.

DevOps est une culture et une démarche mais certainement pas un cadre technologique prédéfini.

9.6. DevOps = No-Ops

DevOps est parfois interprété comme la fin des équipes de production et d'infrastructure (No-Ops) qui deviendraient alors totalement inutiles. On prête même ce nom à des organisations qui auraient complètement supprimé ce pan des équipes informatiques, souvent en se proclamant full cloud ou full web.

Cette affirmation est fausse. D'abord, parce que DevOps ne se limite pas à ce type d'organisation mais surtout parce que même si ce poste n'existe plus en tant que tel, les activités propres aux équipes de production continuent d'exister.

En effet, si DevOps est une approche de collaboration, il faut être plusieurs pour pouvoir collaborer. DevOps ne vise donc pas à encourager l'extinction naturelle des équipes opérations mais tout au plus à transformer partiellement leur rôle pour tirer le maximum de valeur d'une collaboration réussie.

Donc non, DevOps ne signifie la fin des Ops, bien au contraire, DevOps prépare l'avenir des Ops.

9.7. DevOps fusionne les équipes Dev et Ops

C'est parfois le désir profond de certains managers qui veulent implémenter une approche DevOps mais, comme nous l'avons vu, il existe de nombreuses organisations possibles pour mettre en œuvre DevOps.

Il n'est donc en rien obligatoire de fusionner les *Dev* et les *Ops* que ce soit dans la hiérarchie ou au sein d'une même équipe dédiée à l'implémentation d'un domaine fonctionnel.

Il est important de distinguer la fusion des équipes de la fusion des rôles. La fusion des équipes est possible avec des rôles distincts mais sans que cela ne soit un prérequis ni même une recommandation formelle pour mettre en œuvre DevOps.

Par contre, la fusion des rôles n'est clairement pas recommandée.

9.8. DevOps est une intervention des Dev contre les Ops

Parfois, dans le cadre d'une transformation DevOps, les développeurs ont pu être tentés d'adopter en priorité les pratiques qui leur apportaient de la valeur directement avant de penser à une forme de collaboration plus globale.

Fort heureusement, cela ne s'inscrit pas dans une volonté d'opposer les parties prenantes, bien au contraire, et autant les développeurs que les équipes de production ont intérêt à adopter ces pratiques.

9.9. DevOps ne fonctionne que dans les start-up et les petites entreprises

Cette affirmation a été souvent utilisée pour bon nombre d'innovations. Quel serait l'intérêt de DevOps, si les apports de cette démarche se limitaient uniquement dans les

petites entreprises où les personnes ont un profil plus polyvalent et où les risques sont relativement limités ?

Pourquoi chercher à améliorer la collaboration lorsque les équipes informatiques partagent deux bureaux et lorsque le métier est dans le bureau d'en face ?

Vous l'avez compris, les démarches DevOps apportent le cadre et la rigueur nécessaires pour faire collaborer de nombreuses personnes de manière efficace. DevOps n'est donc ni lié aux start-up qui l'adoptent naturellement, ni à la taille de l'entreprise.

9.10. DevOps ne fonctionne que pour le web

Par ailleurs, le support technologique important qui peut être apporté à ce type de démarche amène à penser qu'elles ne peuvent être implémentées que dans des architectures web.

Il est peut-être parfois plus complexe de mettre en œuvre ces principes sur d'autres architectures plus traditionnelles mais ils ne sont en rien spécifiques aux architectures web, petites ou grandes.

DevOps n'est lié ni au développement web, ni à internet.

En résumé

DevOps est une approche de collaboration qui est agile. Les méthodes agiles, sans être un préalable nécessaire, sont parties intégrantes de la démarche.

De la même manière, DevOps n'est en rien incompatible avec ITIL et les deux approches sont complémentaires. L'implémentation d'ITIL doit néanmoins chercher à respecter le principe du *just enough process*.

DevOps n'est pas un cadre technologique prédéfini et n'est pas limité à la communauté open source. L'automatisation et les pratiques telles que l'infrastructure as code ne sont que des supports à cette approche et sa culture.

DevOps n'impose aucune organisation particulière, et ne préconise pas un modèle *No-Ops*. La fusion des équipes *dev* et *ops* est possible sans être ni nécessaire, ni particulièrement préconisée. La fusion des rôles n'est pas recommandée car elle n'offre pas de certitude sur la valeur apportée.

Enfin, DevOps ne se limite pas aux petites équipes. DevOps est totalement adapté aux grandes équipes, aux grandes entreprises et aux grands défis.

Chapitre 10

DevOps demain ?

Aujourd'hui déjà, toutes les entreprises ont dû intégrer l'outil informatique au cœur de leurs modèles métiers. En ce sens, toutes les entreprises sont déjà des entreprises informatiques.

Nous sommes convaincus que cette tendance, loin de s'atténuer, va au contraire s'accélérer avec l'essor de la réalité augmentée et des objets connectés.

Nous pensons que l'esprit, les objectifs des démarches DevOps vont continuer à s'étendre et à se généraliser, tant les bénéfices qu'ils concourent à atteindre sont vitaux pour les stratégies à long terme des entreprises.

DevOps demain, c'est donc une culture du *fail fast, check fast and correct fast*, du découplage des services, des applications et des interfaces tout autant que des infrastructures. Et c'est la capacité à faire évoluer les infrastructures sans limitation liée à une adhérence applicative.

En y parvenant, nous pourrons lutter contre l'obsolescence logicielle. Un moyen efficace de réussir est de s'appuyer sur la containérisation des infrastructures virtualisées et hébergées dans des datacenters gérés à grande échelle avec la généralisation du cloud.

10.1. La fin de l'obsolescence applicative pour les Ops

Les démarches DevOps impliquent un changement de culture mais surtout d'état d'esprit. Demain, nous admettrons beaucoup plus facilement qu'une anomalie puisse être détectée en production en particulier parce que l'environnement aura évolué rapidement. Demain, nous serons également beaucoup plus à l'aise avec ces environnements qui seront de moins en moins contrôlés par les entreprises et beaucoup plus alignés avec ce que nous connaissons dans le monde de la consommation grand public.

Mais nous ne serons réellement apaisés face à cette liberté retrouvée que parce que nous disposerons de réponses adéquates et réactives aux problèmes que ces situations peuvent faire naître. Autrement dit, les démarches DevOps seront partout puisque c'est à travers elles que nous serons suffisamment réactifs et agiles.

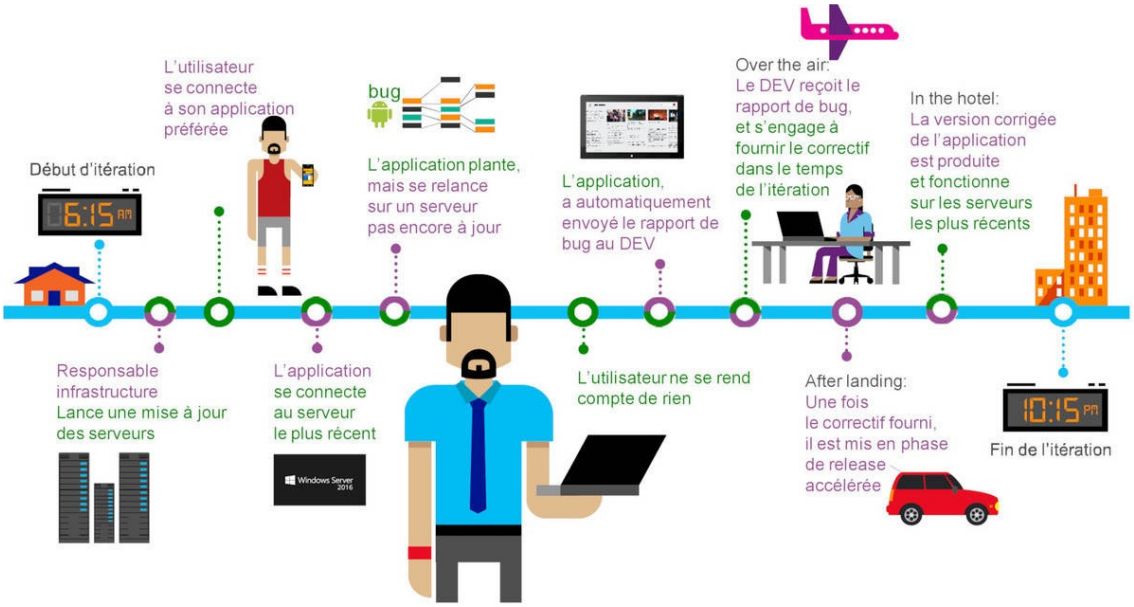


Fig. 10.1 *Les mécanismes de gestion de l'obsolescence applicative stay current*

Néanmoins, afin que les pratiques apportées par les démarches DevOps soient pleinement fonctionnelles pour enterrer définitivement l'obsolescence applicative, les applications et les infrastructures doivent évoluer.

Parmi ces prérequis à mettre en œuvre en sus des démarches DevOps, nous pouvons citer :

- La notion de **lots d'infrastructures complémentaires.** La division en lots est un principe connu des équipes en charge de l'infrastructure depuis longtemps. La démarche DevOps étend ce principe au périmètre applicatif et à la notion d'itération. L'idée est donc d'itérer pour mettre à jour son parc de serveurs sur un périmètre applicatif donné, avec potentiellement, plusieurs itérations sur un même périmètre. L'objectif est ici de permettre à une application d'utiliser l'environnement le plus à jour et en cas de problèmes de revenir sur le dernier environnement fonctionnel connu. Il est donc nécessaire de disposer de ces deux environnements à chaque fois qu'une évolution est envisagée : c'est la notion de lot d'infrastructures complémentaires.
- La notion d'**applications balancer.** Si nous disposons d'environnements fonctionnels complémentaires afin de proposer à une application de les utiliser en cas de problème, encore faut-il que ce mécanisme de continuité et *rollback* applicatif soit prévu. Ce mécanisme doit permettre simplement à l'application de chercher et utiliser en permanence l'environnement compatible le plus récent peu importe où il se trouve, et seulement en cas de problème revenir de manière transparente sur un environnement différent. Ce mécanisme d'*applications balancer* peut-être intégrée directement à vos applications ou mis à disposition de l'ensemble des applications sous forme de service.
- La notion de **store applicatif.** Nous ne reviendrons pas en détail sur cette notion déjà évoquée dans un précédent chapitre mais nous en rappellerons simplement le principe : celui de disposer d'un entrepôt applicatif intelligent capable de distribuer la bonne application au bon groupe d'utilisateurs en fonction de leur environnement de

travail. C’est un outil indispensable à une gestion DevOps du poste de travail contre l’obsolescence applicative.

Tout cela étant bien entendu complémentaire de la démarche de collaboration DevOps pour pouvoir alimenter les circuits de collaboration de manière hyper-réactive.

Enfin, il faut comprendre que ces mécanismes sont très similaires que l’on parle d’obsolescence applicative suite à une évolution de l’infrastructure ou du poste de travail tel que nous l’avons détaillé dans les chapitres précédents.

10.2. L’avènement du SaaS comme standard dans les entreprises

10.2.1. Le cloud un accélérateur formidable qui deviendra indispensable

Le monde du cloud n’est pas directement lié aux démarches DevOps, il n’est en rien indispensable à l’amélioration de la collaboration ou dans la mise en œuvre des principes de l’agilité.

Néanmoins, les infrastructures *cloud*, qu’elles soient privées, publiques ou hybrides, apportent une flexibilité et une offre de service que les infrastructures traditionnelles ont du mal à égaler. L’ensemble des principes et pratiques que nous avons évoqués se trouvent grandement facilités dans leur mise en œuvre.

Enfin, ces infrastructures généralisent une approche services à des coûts mutualisés à grande échelle. Les freins qui peuvent exister aujourd’hui autour de la sécurité, la localisation des données ou la compatibilité avec des infrastructures existantes vont être levés dans les années, pour ne pas dire les mois qui arrivent.

Le rapport qualité, prix, service que le cloud va bientôt proposer le rendra absolument indispensable et permettra une accélération réelle de l’adoption des pratiques DevOps. Les objectifs stratégiques des entreprises vont également accentuer ce phénomène.

En tout état de cause, cette transformation de l’infrastructure informatique vers une architecture complètement orientée services permettra de proposer des applications complètement et uniquement hébergées dans *le nuage*. Nous aurons alors tous les ingrédients pour entrer dans l’âge d’or des applications SaaS.

10.2.2. Le SaaS et ses bénéfices

Le principe du SaaS est de déporter la totalité de la couche logicielle dans les infrastructures gérées par l’éditeur plutôt que sur le poste de travail de l’utilisateur. De cette manière, l’application est accessible depuis n’importe quel poste de travail, personnel ou professionnel et offre la même expérience en permanence.

La nature de ces applications déportées est de fonctionner totalement en services, du frontal généralement accessible *via* des navigateurs web jusqu’à la conception et jusqu’au

fonctionnement de l'ensemble de leur architecture.

Il est quelque peu abusif de parler de solutions SaaS par exemple lorsqu'une application traditionnelle développe une simple interface web sur un moteur logiciel traditionnel.

Du point de vue de l'utilisateur, la simplicité d'accès sans s'encombrer de problèmes d'installation, de paramétrage ou de maintenance associé à l'assurance de bénéficier en permanence des dernières fonctionnalités est particulièrement séduisante.

Pour l'éditeur de son côté, pouvoir bénéficier d'un parc harmonisé de services, homogène et sous contrôle simplifie considérablement sa gestion et ses coûts de distribution.

Le logiciel devient un simple service consommable à la demande. Pour comprendre le potentiel de ce type d'approche, il faut se remémorer le parc applicatif installé dans les grandes entreprises, que ce soit du développement spécifique ou du progiciel et imaginer que tout cela bascule sur du SaaS.

Au-delà de l'ensemble des difficultés techniques pour migrer vers ce modèle, il est évident que ce scénario est une transformation majeure qui change durablement le modèle de gestion du système d'information.

Et cette transformation est complètement dans la logique portée par les démarches DevOps.

10.2.3. Parenté directe du SaaS et de DevOps

L'un des principaux avantages du SaaS est la rapidité de déploiement. Les démarches DevOps visent quant à elles à accélérer le déploiement tout en se focalisant de manière continue sur les utilisateurs.

Le lien de parenté est donc évident, des applications SaaS exploitent complètement leur potentiel lorsqu'elles sont associées à une démarche DevOps avec notamment les pratiques de monitoring et de télémétrie par exemple.

Il est évident que lorsque les mécanismes de feedback existent et sont exploités et qu'à travers eux un besoin utilisateur est détecté, quoi de plus efficace qu'une application SaaS pour déployer rapidement la fonctionnalité qui répond à ces attentes ?

Les démarches DevOps promeuvent les usages autour de la mobilité et de la continuité des activités sur plusieurs devices. Quelle application est plus adaptée à ce type d'usage que les applications SaaS ? Même les applications mobiles des smartphones s'appuient généralement sur l'ensemble de l'architecture des applications SaaS alors même que leur interface est native.

Les démarches DevOps ne compensent pas les inconvénients du SaaS qui existent encore mais accélèrent et amplifient les bénéfices de ce type de services.

10.2.4. Passer une application traditionnelle en application SaaS

Il n'est pas question ici de détailler les scénarios possibles de migration d'une application traditionnelle vers le SaaS, le rapport avec DevOps n'est pas forcément évident et il existe d'éminents spécialistes du sujet.

Il est plutôt question de comprendre pourquoi DevOps facilite ce passage.

La démarche DevOps s'appuie sur les principes de l'agilité et encore une fois une approche itérative et incrémentale est sans doute la manière la plus efficace de réussir cette transition. Néanmoins, l'expérience a montré que déployer complètement les pratiques de mise en production et de tests automatisés facilite énormément les tests de non-régression sur un périmètre fonctionnel constant. Lorsque l'on applique ce même principe au développement d'une application SaaS avec un même périmètre fonctionnel, les bénéfices des pratiques DevOps s'en retrouvent décuplés.

Mais passer à un modèle SaaS n'empêche pas de faire cohabiter des versions *full SaaS* hébergées dans le cloud et des versions plus traditionnelles en *boîte*. Pour tirer tous les bénéfices d'une transition vers ce modèle hybride, il est cependant indispensable de partager l'ensemble de l'architecture et du code entre ces deux formes de service ou de produit.

Il existe quelques exemples qui montrent la voie. On peut citer les versions web et apps des applications de productivité d'Apple, les ambitieuses applications universelles d'Office, la plupart des nouveaux services des réseaux sociaux tel que Facebook qui partage son code entre son application mobile et son site ou encore Team Foundation Server et sa version SaaS Visual Studio Team Services qui partagent la quasi-totalité de leur code source.

C'est d'ailleurs ce compromis entre un modèle SaaS et des applications plus traditionnelles qui a le plus d'avenir et qui saura le mieux tirer parti des pratiques issues d'une démarche DevOps.

En effet, c'est le modèle qui peut toucher le plus grand nombre d'utilisateurs et les démarches agiles et DevOps agiront sans doute comme des accélérateurs de cette transition inévitable pour répondre aux enjeux de l'économie de demain.

10.3. Automatisation des environnements avec Docker

Dans un contexte dans lequel les applications sont développées en continuous delivery, selon les principes DevOps, avec une implémentation fondée sur une architecture de type microservices, Docker est susceptible d'avoir un impact très significatif sur l'évolution des processus de développement et de livraison de logiciels. La révolution des containers est en marche, et tout porte à croire que demain, l'usage de la solution Docker devrait se généraliser.

10.3.1. La fin de la virtualisation classique ?

Le modèle de virtualisation par les containers est fondé sur un partage du système d'exploitation et le cas échéant de certaines librairies. Il en résulte un déploiement et un redémarrage beaucoup plus rapide, ainsi qu'une consommation de ressources beaucoup

plus faible que dans le cas des machines virtuelles. L'utilisation de containers permet donc d'envisager des scénarios d'évolutivité en *scale-up* et en *scale-down* plus avancés que ceux dont nous disposons aujourd'hui avec les machines virtuelles.

Pourtant, les machines virtuelles ne sont pas prêtes de disparaître. Elles présentent toujours l'avantage de permettre l'exécution de systèmes d'exploitation différents sur la machine hôte et la migration d'environnements virtualisés d'un serveur à un autre tout en conservant leur état. De plus, elles offrent un plus haut niveau d'isolation en termes de ressources et de sécurité. C'est d'ailleurs pour cette dernière raison que Microsoft propose le concept du Hyper-V container en complément des containers Windows. Enfin, il est également possible de faire s'exécuter et de combiner l'usage des containers et des machines virtuelles afin de tirer avantage des deux modèles en offrant plus de flexibilité (comme la *live migration*, une fonction qui permet de migrer des machines virtuelles d'un serveur physique à l'autre sans interruption de service) tout en augmentant la densité d'applications afin d'optimiser les ressources.

10.3.2. Principe de fonctionnement de Docker

Docker est une solution open source de manipulation des containers exposée *via* une API REST dans un modèle client-serveur et complétée par un ensemble de commandes. L'objectif est d'offrir une gestion des containers fondée sur un mécanisme de déploiement vers un Docker Host, Linux ou Windows. Docker propose un modèle de packaging (l'image qui inclut l'ensemble des dépendances logicielles de l'application construites par couches successives, les layers) et de distribution (on peut rechercher ou publier ces images sur un référentiel privé, le Docker Trusted Registry) ou public, leDocker Hub).

L'image décrit le contenu du container : système d'exploitation, configuration et inventaire des processus qui s'exécuteront lorsque le container est lancé… et peut être déclarée dans un DockerFile.

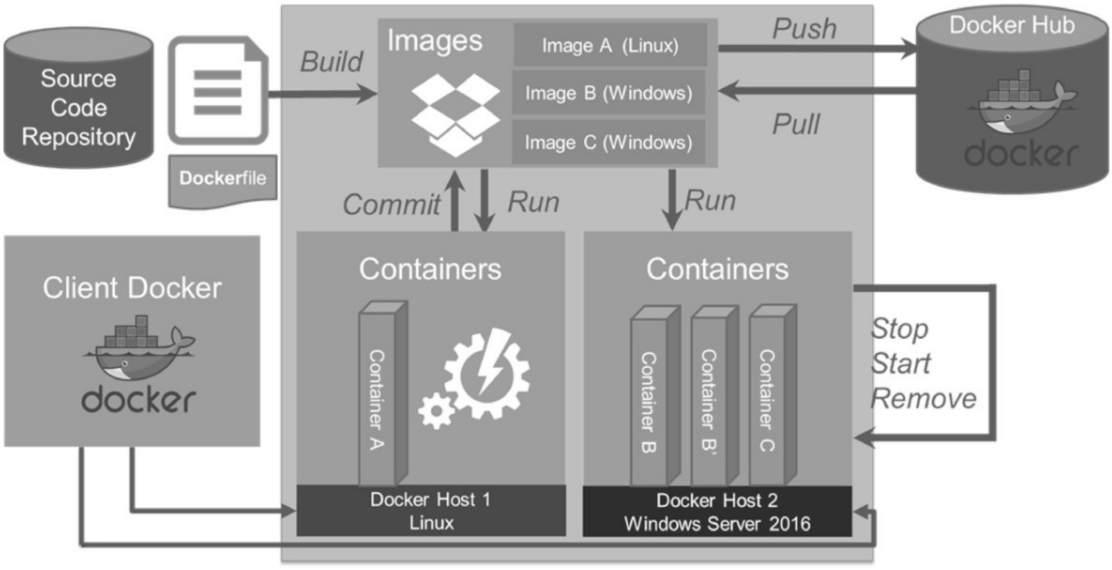


Fig. 10.2 *Principe de fonctionnement de Docker*

Docker complète ces mécanismes fondamentaux par des fonctions de composition pour la gestion d'applications multicontainers (Docker Compose) et d'orchestration pour le clustering de Docker Hosts (Docker Swarm).

10.3.3. Docker et le cloud

L'optimisation de la consommation des ressources et les temps de démarrage accélérés font de Docker un bon candidat pour les scénarios cloud en permettant aux applications d'évoluer encore plus efficacement. La plupart des solutions de cloud public supportent donc Docker et ses composantes de composition ou d'orchestration, comme Docker Swarm qui présente l'intérêt d'être conçu avec un point d'extensibilité permettant de le remplacer par des solutions pour les déploiements de production à grande échelle, comme Apache Mesos. Cette solution open source de gestion d'un cluster de ressources est utilisée par des services comme Twitter, eBay ou NetFlix on la retrouve dans l'implémentation de la solution DCOS (*DataCenter Operating System*) de la solution Mesosphere disponible chez Amazon, Google ou Azure (c'est d'ailleurs le socle de la solution Azure Container Service).

10.3.4. Docker & DevOps

Docker facilite la collaboration entre développeurs et équipes de gestion des opérations et permet ainsi de faciliter la mise en place d'une démarche DevOps pour apporter plus de valeur dans des délais plus rapides.

Le mythe du write once, run everywhere

En offrant un niveau d'abstraction entre l'image et le système d'exploitation et de l'infrastructure, Docker permet de déployer et faire s'exécuter le container *n'importe où*, avec une limite : une image définie pour Windows n'est pas directement utilisable sur Linux et inversement. Par contre, il est possible de déployer un container d'une distribution Linux à une autre, à condition que le Docker Host respecte certains pré-requis (version du noyau Linux et support d'AUFS). De même, en environnement Windows, une image prévue pour s'exécuter sur la version Nano Server pourra également être utilisée pour instancier un container sur une version Server Core.

Des avantages pour les développeurs

Avec Docker, les développeurs ont aujourd'hui la possibilité de construire une application issue de la composition de multiples images et de la déployer comme un ensemble de containers, pour qu'elle s'exécute sur un Docker Host (voire un cluster de Docker Hosts). Ils bénéficient ainsi de gains en termes de productivité (par le partage des images), d'efficacité (par la rapidité de temps de démarrage et de déploiement), et d'agilité (par la liberté de contrôle). L'exécution fiable et reproductible fait donc des containers un environnement idéal pour le développement et le test qui facilite les itérations. De plus, Docker répond à un grand nombre de problématiques liées à la résolution des dépendances pour les architectures fondées sur des microservices et facilite la réutilisation d'images existantes.

Des avantages pour la gestion opérationnelle

Pour les opérations, les avantages de Docker sont multiples : gouvernance et optimisation des ressources, augmentation de la densité d'applications proposée par la virtualisation des containers, rapidité du déploiement et adaptation du dimensionnement, contrôle uniforme des opérations de démarrage en environnement Linux et Windows… De plus le modèle de packaging, très efficace, permet une distribution par *layer* au lieu d'un remplacement intégral de l'image comme le nécessiterait une machine virtuelle.

Des capacités uniques qui favorisent l'outillage d'une démarche DevOps

Docker offre des ajouts majeurs pour la mise en œuvre de l'outillage d'une démarche DevOps. Cette solution ne se limite pas à un mécanisme de packaging, mais permet également de tester, maintenir, publier et déployer les applications plus efficacement. Dans ce nouveau modèle d'ingénierie logicielle, les images Docker constituent le fondement de l'application. Il devient alors impératif de proposer un environnement fiable et sécurisé pour les générer, les composer et les maintenir, en intégrant directement ces nouvelles capacités dans la chaîne de production logicielle.

Nous avons évoqué précédemment les difficultés liées au partage de ressources pour les tests effectués au cours d'un processus d'intégration continue (et les erreurs qui pouvaient en découler). Docker apporte une réponse immédiate à ce type de problématique. Il suffit de déployer le code, pour chaque build, dans un nouvel environnement construit sur des containers et de le supprimer une fois qu'il n'a plus d'utilité. Ce qui suppose l'intégration de Docker dans des solutions d'intégration continue comme Jenkins ou Visual Studio Team Services ou la simple mise en œuvre des automated builds du Docker hub, qui permettent de lier un dépôt de code GitHub ou Bitbucket, et de créer automatiquement des images Docker à chaque commit, afin que ces images soient toujours à jour par rapport au code.

Docker apporte un niveau d'abstraction supplémentaire dans la mise en œuvre du processus de release management. L'utilisation d'un référentiel central de distribution en continu permet de s'assurer que les mêmes fichiers binaires sont utilisés à chacune des étapes du pipeline de release management. Docker élimine tout risque d'altération de l'application et de son environnement d'exécution et garantit la mise à disposition d'une infrastructure de référence identique à chaque étape du cycle de production logicielle. Les containers peuvent ainsi être ajoutés ou supprimés sans la contrainte d'avoir à mettre à jour un environnement existant. De plus, Docker permet de faire évoluer indépendamment les applications et son environnement d'exécution.

Développeurs et équipes de gestion opérationnelle manipulent les mêmes outils dans des processus partagés. L'association entre l'application et son environnement d'exécution est matérialisée dès le début, ce qui suppose une réflexion conjointe de la part des développeurs et des équipes de gestion opérationnelle sur la mise en place de référentiels liés au code source, au package management et aux images Docker des différents environnements correspondants aux multiples briques de la solution (système, serveur d'application, base de données…). À l'issue du processus de build, l'application sera la

résultante de l'assemblage de multiples images dont les instances pourront être déployées afin de générer les différents environnements requis pour valider la solution.

L'image Docker devient alors le format pivot qui va transiter entre les différentes étapes ce qui garantit non seulement que l'on teste bien la même chose, mais aussi qu'on teste l'application dans le même environnement. L'application de changement sur l'environnement n'est plus un processus qu'il faut appliquer en parallèle du processus de mise à jour du code (ce qui représentait un risque potentiel de désynchronisation). La mise à jour d'une image déclenche le processus de build au même titre que la modification du code qui s'y exécute.

Développeur et responsable de l'IT partagent ainsi le même pipeline de release management. Les processus de développement et de gestion opérationnelle sont réellement unifiés…

10.3.5. Docker n'est pas le futur de DevOps

Toutefois, même si ce chapitre s'inscrit dans le chapitre « DevOps demain ? », Docker n'est pas le futur de DevOps. C'est une solution technique dont l'adoption devrait aller croissant, et dont la mise en œuvre s'aligne sur un certain nombre de principes DevOps. Mais le paysage informatique de demain ne se le limitera pas aux seuls containers, quant à la démarche DevOps, comme nous l'avons dit et répété dans cet ouvrage, c'est avant tout un changement de culture qu'elle suppose.

En résumé

La lutte contre l'obsolescence applicative passe par le combat contre l'adhérence des applications et des infrastructures. La mise en place d'interfaces automatisées rend ces interactions plus agiles et permet ainsi de rendre générique l'ensemble des échanges.

Cette pratique est identique que l'on parle du poste de travail ou d'infrastructure serveur, mais sa déclinaison concrète peut être différente tout comme les choix d'implémentation technique et technologique.

On retrouve une similitude dans les perspectives qu'offre aujourd'hui l'avènement de la containérisation. Docker révolutionne le processus de *continous delivery* en repensant de manière totalement innovante les modèles traditionnels. En appliquant les principes issus de la culture DevOps, les développeurs et les responsables opérationnels sont aujourd'hui en situation de pouvoir unifier l'application et l'environnement dans un container lui-même générique.

De multiples bénéfices en résultent : la configuration est gérée dans des bases dédiées et appliquée dynamiquement en cas de changement. La gestion de l'élasticité des middlewares est simplifiée et peut être appréhendée dès la phase de développement.

Mais Docker n'est que l'une des nombreuses composantes du système d'information et ne saurait constituer à elle seule le futur de Docker.

Conclusion

Le numérique, le digital et plus généralement l'informatique et le logiciel sont aujourd'hui les principaux vecteurs de création de valeur pour les entreprises. DevOps accélère cette évolution, en offrant aux organisations une capacité à mieux collaborer et à mieux s'adapter au changement, dans un cycle d'innovation en continu.

DevOps permet d'accélérer le *Lead Time*, d'augmenter la fréquence des déploiements, de réduire le *Mean Time To Recovery*, autant d'indicateurs de la capacité de DevOps à représenter un axe stratégique de différenciation sur le marché. Les principaux analystes confirment cette tendance en identifiant DevOps comme le principal moteur de la transformation digitale des entreprises.

DevOps est une démarche de collaboration agile entre les études et développements (les Dev), la production et les infrastructures (les Ops) et les métiers (le Business), du recueil de la conception client, jusqu'à son suivi en production. L'adoption de DevOps devrait donc susciter un intérêt croissant avec un impact sur chacun de ces rôles au sein de l'entreprise.

En gagnant en réactivité et optimisant la collaboration entre toutes les parties prenantes, l'entreprise devient alors capable de tirer profit rapidement des innovations indispensables pour survivre, vivre et grandir dans un contexte concurrentiel. Mais comment aborder une telle transformation ?

DevOps requiert d'abord un changement de culture. Avant de lancer une initiative en vue de l'évolution de tel ou tel processus, ou d'évaluer l'opportunité de l'acquisition de telle ou telle solution logicielle, il convient de bien intégrer des principes comme la prise de risque, l'acceptation de l'échec et son analyse décorrélée de toute notion de reproche, le *fail fast*, le *continuous learning* et le *continuous improvement*… Dans l'entreprise DevOps, chacun dispose d'un niveau d'information lui permettant d'avoir une vue complète du système et se sent responsable de la globalité de la solution. Chacun contribue à son évolution et bénéficie du niveau de confiance requis. Les objectifs y sont partagés et la structuration des équipes n'est plus nécessairement liée à leur expertise.

Pour les développeurs, adopter une démarche DevOps est une formidable opportunité d'amplifier les bénéfices apportés par les pratiques agiles contextualisées dans l'organisation. Les pratiques agiles existantes sont alors prolongées de manière mutuellement efficace avec les équipes de production qui gagnent en réactivité. Cela suppose une évolution des process et outils, en veillant à accompagner ce changement et à ne pas vouloir aller trop vite. L'objectif ultime est de déployer, le plus fréquemment possible, un code qui fonctionne. Des erreurs peuvent résulter de ces mutations, il faut alors les corriger et recommencer, jusqu'à ce que la chaîne de production logicielle fonctionne dans ce nouveau mode.

Pour les équipes opérationnelles, DevOps apporte un gain indéniable en qualité et en responsabilisation des développeurs concernant la production. Les pratiques mises en

œuvre permettent d'optimiser la collaboration et l'automatisation des processus de déploiement d'infrastructure. Pour les opérations, cela signifie également l'adoption progressive de nouveaux process et l'adaptation des outils existants, ce qui ne se fera pas sans échec. Et ces échecs devront être analysés pour en tirer des enseignements et améliorer le système, pas pour juger des responsables.

Pour les métiers ou le *Business*, DevOps est avant tout le moteur de leur digitalisation en vue de répondre toujours mieux à leur marché et à leurs clients. L'adoption de DevOps par les métiers se fera très naturellement si développeurs et opérations sont déjà sur le chemin de la transformation. Ils bénéficieront ainsi de nouveaux moyens pour prendre en compte des *feedbacks* utilisateurs et pourront agir positivement sur leur satisfaction et sur les principaux indicateurs de performances traditionnels.

DevOps doit être voulu par le management qui doit inscrire ses principes dans la culture de l'entreprise et faire évoluer l'organisation pour qu'elle puisse les intégrer dans son fonctionnement. C'est une étape prioritaire de la transformation DevOps.

Enfin et surtout, il faut susciter l'adhésion de l'ensemble des parties prenantes. Ainsi que l'écrit Saint-Exupéry : « *Quand tu veux construire un bateau, ne commence pas par rassembler du bois, couper des planches et distribuer du travail, mais réveille au sein des hommes le désir de la mer grande et belle.* »

Index

Table des matières