

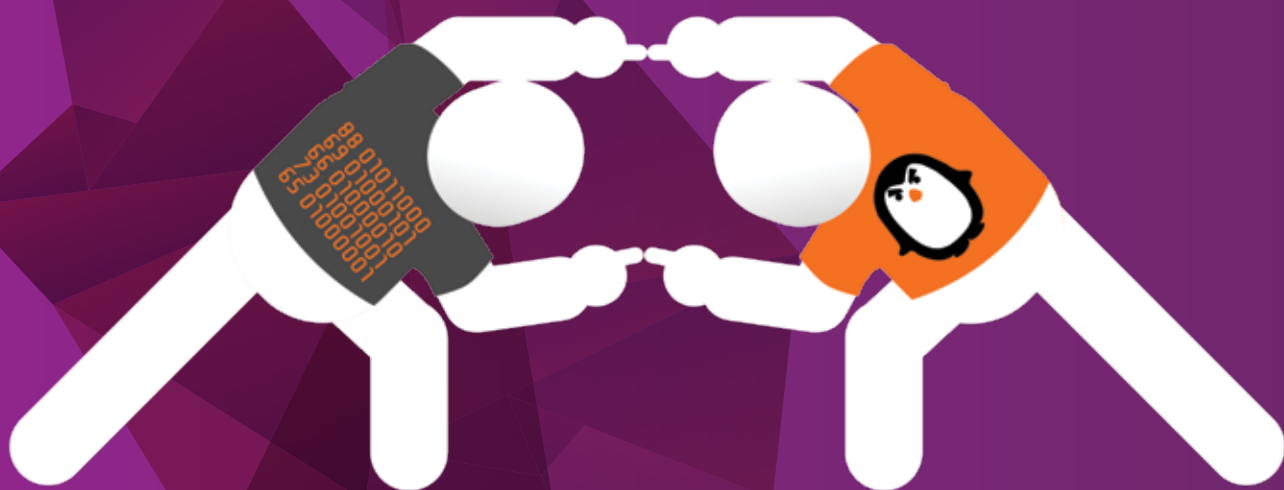
# TECH/TRENDS

#2

NOVEMBRE  
2013

PERCEVOIR LE FUTUR

# DEV'OPS

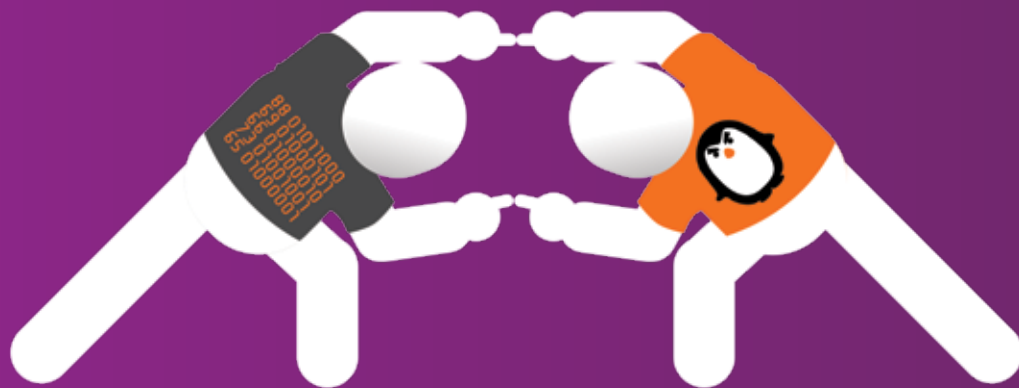


*Coopérer*

*Fluidifier*

*Livrer*

# DEV'OPS



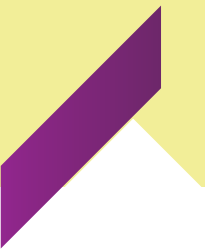
*Coopérer*



*Fluidifier*



*Livrer*



**D**epuis quelques années, l'agilité connaît un essor grandissant dans les DSI. Aujourd'hui, l'éventail de méthodologies est large et adaptable à tous les contextes. Les valeurs clés de l'agilité sont la collaboration, la transparence, la culture de la qualité, l'adaptation et la simplicité. De l'équipe Scrum à la Feature Team, les transformations agiles ont convaincu de nombreuses directions de SI par leur efficacité au sein des équipes de développement. L'agilité, quand elle n'est appliquée qu'au développement, se trouve néanmoins freinée par les tâches d'exploitation qui surviennent après chaque livraison.

Le but du mouvement DevOps est d'abattre cette frontière en créant une synergie entre les équipes d'exploitation (Ops) et les équipes de développement (Devs). Traditionnellement, Ops et Devs ont des objectifs antagonistes : les uns sont les garants de la stabilité et de la disponibilité des systèmes, là où les autres sont employés à l'évolution de ces derniers. Cela crée un clivage entre ces équipes, appelées à travailler ensemble et à tendre vers un même objectif : délivrer le meilleur logiciel aux clients de l'entreprise. De plus, il est courant que les Ops aient des notions de développement, et les Devs, d'exploitation. Cela entraîne inévitablement des conflits. Qu'elles soient bloquantes ou non, ces frictions dégradent la productivité. Plusieurs symptômes sont révélateurs de ces problèmes :

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

Les open spaces grouillent d'anecdotes du même acabit. Nous allons vous guider dans notre vision d'une démarche d'amélioration, pour pallier ces problèmes efficacement, afin que « DevOps » ne soit pas un buzzword supplémentaire pour vous.

# Coopérer



**La culture** et l'organisation de votre entreprise sont un pilier de votre transformation vers DevOps. Les leitmotifs de DevOps peuvent néanmoins paraître galvaudés : qui ne se targue pas de vouloir travailler de manière collaborative et en toute transparence ? De ce fait, quelles sont les implications concrètes sur l'organisation d'un département exploitation derrière le mouvement DevOps ? Est-ce une réelle rupture ou un simple effet de mode ?

**Les organisations** de type « You build it, you run it » sont sans compromis et semblent un objectif utopique à atteindre pour beaucoup. Voyons ici quelques axes de travail et les outils que vous pouvez utiliser pour démarrer rapidement une démarche DevOps sans avoir à révolutionner votre organisation.

## RESTAURER LA CONFIANCE

L'activité des Ops est jalonnée de nombreuses crises. Qu'elles soient liées à des pannes matérielles ou à une mise à jour provoquant une instabilité du système. Les Ops sont familiers de la gestion de crise, avec ses horaires à rallonge et ses diagnostics parfois laborieux. Les bureaux d'Ops résonnent d'histoires croustillantes sur ces Devs totalement inconscients qui mettent la production en péril avec des déploiements de binaires hasardeux. L'un des premiers axes de progrès est donc de restaurer la confiance dans un contexte de défiance mutuel. Pour ce faire, deux outils sont à votre disposition.

D'une part, la conclusion d'une mise en production doit être l'objet d'une réunion de retour d'expérience (appelée rétrospective) entre Devs et Ops afin de capitaliser sur les bonnes pratiques et identifier les points d'amélioration. Une pratique régulière des rétrospectives devrait diminuer sensiblement le nombre de crises et restaurer la confiance entre ces deux protagonistes.

D'autre part, en cas de crise, l'utilisation d'un format de résolution de problème de type A3 peut s'avérer précieuse. Ce format permet de s'attaquer aux causes racines d'un problème afin qu'il ne se reproduise plus. Constitué de deux parties (droite et gauche), le format A3 se présente comme suit :

*L'un des premiers axes de progrès est donc de restaurer la confiance dans un contexte de défiance mutuel.*



CONTEXTE  
ÉTAT ACTUEL DU PROJET  
OBJECTIF  
ANALYSE DES CAUSES

Partie gauche du A3



# CONTREMESURES CONFIRMATION DE L'EFFET SUIVI DES ACTIONS PRISES

Partie droite du A3



*Le format A3 crée de la transparence car il a vocation à être affiché ostensiblement (porte, couloir, lieu de passage) pour permettre à chacun de suivre le plan d'actions associé. Le A3 permet également d'éviter le syndrome « protéger ses arrières » en étant tourné vers le futur plutôt que sur l'identification des coupables. C'est en effet une dérive assez courante dans les situations de crise de faire des post-mortems consensuels où les protagonistes cherchent à se protéger. Assurez-vous que chaque crise est suivie d'un A3 dont le porteur est un manager.*

RÉTROSPECTIVES

A3

STAND UP  
MEETING

COACHING



MONITORING

HAUTE  
DISPONIBILITÉ

STABILITÉ

FIABILITÉ





## CONCEVOIR DES PRODUITS « OPS-READY »

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

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

## ANTICIPER L'ACTIVITÉ OPS

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



*Il suffit parfois de partager un simple outil de management visuel pour créer de la transparence sur les travaux en cours côté Devs et permettre aux Ops d'anticiper les demandes.*



# Une étude de cas

Voyons ensemble une étude de cas dans laquelle une organisation des opérations initie sa mue en s'inspirant des méthodes agiles de développement logiciel.

Dans cette DSI de 150 personnes qui rassemble la division des développements (les Devs), celle des méthodes et celle des opérations (les Ops), le patron souhaite que tous les projets, développement logiciel comme Ops (modification des infrastructures, installation du Wifi, amélioration du réseau, provisionnement des plateformes, etc), passent à la méthode agile Scrum.

La transition démarre par un programme de formation pour l'ensemble des chefs de projets et des managers de département. La volonté d'adopter Scrum pour les projets de développement fait sens mais paraît plus incongrue pour les projets Ops qui ne gèrent pas des produits mais des flux d'activités. Paradoxalement, certaines pratiques agiles obtiennent un écho favorable de la part des manager Ops. Ils voient là l'opportunité de sortir de certains modes de fonctionnement d'urgence permanente et d'attitudes consuméristes de la part des projets (demande d'environnement du jour pour le lendemain par exemple). Tout ne semble pas applicable mais les principes suivants séduisent fortement : matérialiser la capacité disponible et planifier régulièrement à hauteur de cette capacité, revoir les priorités collectivement, rendre l'avancement transparent.

Nous décidons de mettre en place un affichage visuel qui matérialiserait l'ensemble du travail en cours dans la division et les flux de ces tra-

vaux depuis leur apparition jusqu'à leur clôture. Nous passons alors du temps à identifier le flux de travail et le besoin de visibilité. Les projets sont compliqués à suivre : nombre d'entre eux sollicitent le travail conjoint de plusieurs départements Ops. Par exemple, un projet d'ouverture de flux entre deux machines nécessite des travaux au niveau système mais aussi au niveau réseau. Chaque projet est donc d'abord analysé par un expert, puis découpé en activités gros grains qui sont affectées à un seul département. Chaque département peut, à son tour, redécouper certaines tâches trop importantes et les traiter en plusieurs fois.



*Le suivi des projets nécessite d'avoir une vision détaillée des activités réparties dans les départements et de leur avancement respectif pour en déduire une vision consolidée et publier les indicateurs d'avancement.*



Cela provoque d'ailleurs de nombreux conflits : d'un côté les chefs de projet essaient de favoriser leurs tâches parfois au détriment de l'efficacité de l'ensemble, de l'autre les départements essaient de maximiser le nombre de tâches traitées, tout en travaillant sur des améliorations transverses pour garantir la stabilité des services. Les responsables de département font difficilement face à la pression de tous les projets simultanément, en plus de la maintenance des plateformes et des systèmes. Les priorités ne sont pas partagées entre les projets et tout est urgent. Lorsque certaines priorités ne sont pas satisfaites, les escalades à la direction pour demande d'arbitrage sont courantes.

Après avoir défini les flux, nous cherchons à alimenter le tableau visuel avec le travail en cours. Les managers écrivent sur des post-its le nom des projets et activités en cours. Ils les positionnent ensuite sur le tableau que nous avons préparé au préalable. Nous procédons à quelques ajustements au niveau de la modélisation du flux : ajout d'une colonne, d'une ligne de nage. Nous matérialisons 2 niveaux de flux sur le tableau: l'avancement d'un projet (New, Qualified, Finished) et l'avancement de ses tâches (Backlog, To do, In progress, Done).

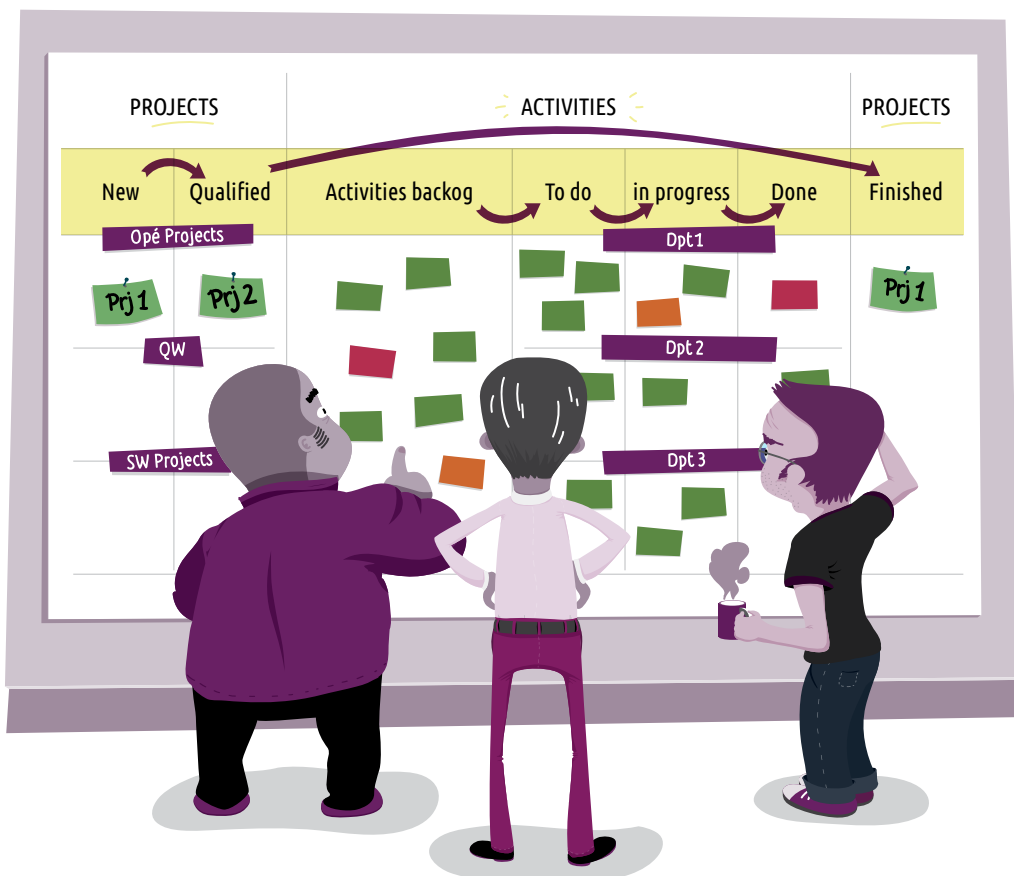


Tableau de modélisation du flux, priorisation du travail des Ops

La dernière étape consiste à mettre en place l'animation du processus autour de ce tableau visuel. Le responsable de division Ops fait déjà un point quasi quotidien avec chacun de ses managers pour revoir les points d'arbitrage et le planning court-terme. Nous proposons alors de systématiser ce point quotidien, non plus avec un seul mais avec les 3 managers et autour du tableau pour revoir le planning de la journée et décider conjointement des arbitrages. La semaine suivante, les points quotidiens sont instaurés et se tiennent systématiquement, le tableau commence à vivre. Les managers s'approprient le format et ajoutent des informations visuelles pour améliorer l'efficacité, par exemple, un code couleur pour matérialiser le caractère urgent ou non de certaines tâches.

Toutes les 2 semaines, les éléments du backlog d'activités sont évalués et priorisés par rapport aux éléments encore en To Do. Les arbitrages sont faits globalement sur le travail des 4 départements en toute transparence. Les projets et leur avancement sont visibles de tous. Les projets logiciel sont notamment matérialisés lorsqu'ils nécessitent une dimension impactant l'infrastructure (provisionnement, étude d'impact, achat de matériel).

Les équipes de chaque département viennent progressivement mettre à jour elles-mêmes le tableau pendant la journée. Les managers n'ont plus à le faire le matin, ils visionnent directement le statut des activités et prennent des décisions pour le planning à la journée. Progressivement, chaque équipe se construit son propre tableau pour le suivi plus détaillé des tâches et des allocations, le tableau commun restant à un niveau de consolidation des informations. Il occupe aujourd'hui un mur de 4m sur 2m, dans une salle de réunion, et vit depuis bientôt 1 an.

*Parallèlement à la structuration du tableau, la méthode Scrum est déployée dans les équipes Devs. Des moments clés des projets sont alors l'occasion de convier les interlocuteurs Ops, afin de les impliquer plus en amont et obtenir un meilleur alignement entre les travaux de développement et les travaux d'exploitation : la préparation du backlog, le sprint 0 ou les démos. Là où certains chefs de projet software n'anticipaient pas bien les travaux d'exploitation, le cadre leur offre désormais une occasion de synchronisation relativement prédictible.*

# Take away DevOps Trends



## COOPÉRER

- *Organiser des rétrospectives régulières entre Devs et Ops plutôt que des post-mortem*
- *Suivre la résolution de problèmes avec un format A3 pour s'assurer de résoudre les causes racines des problèmes*
- *Impliquer les Ops dans la constitution du Product Backlog et les revues de sprint*
- *Utiliser le management visuel pour accroître la transparence et l'anticipation des travaux*

# *Fluidifier*



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

**développer** en accord avec la cible qu'est l'environnement de production,

**adapter,** dans la mesure du possible, le SI aux besoins des développeurs.

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

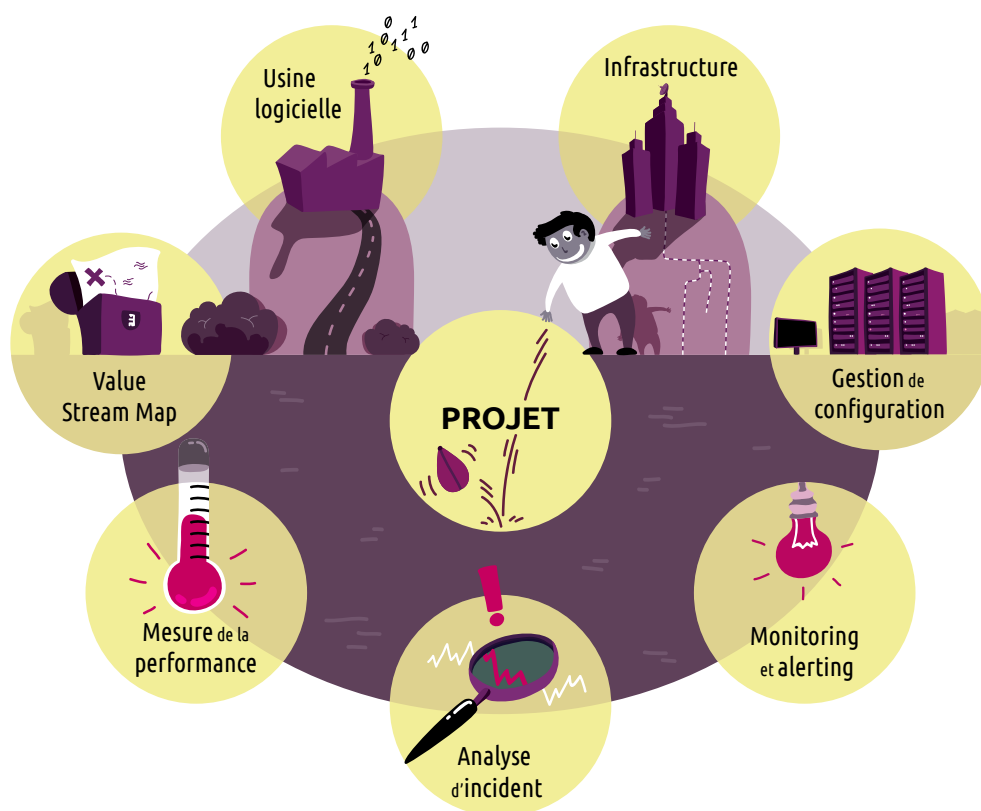
**B**ien sûr, comme tout self-service, on ne peut prendre que ce qui est offert. Les Ops sont les gardiens de la cohérence du SI. Grâce à la connaissance des besoins des équipes de Devs, les Ops vont choisir des solutions techniques orientées Service. Il faut du Infrastructure-as-a-Service, du Log-as-a-Service et du Monitoring-as-a-Service, en interne, pour fluidifier la vie de vos projets.

Une fois que les Ops ont validé et déployé une réponse technique aux besoins des Devs, ceux-ci peuvent s'en saisir pour travailler. Une solution est bonne si les Devs :

- n'ont plus à ralentir les Ops avec des demandes d'actions sans valeur ajoutée,
- ne sont pas ralentis par les services mis en place, et donc, ne sont pas obligés de passer par les Ops pour y accéder.

Par exemple, en cas de bug avéré en production, il y a fort à parier que les Devs seront rapidement impliqués. S'ils doivent courir après les Ops pour accéder aux logs ou aux métriques de monitoring, ou pour déployer un correctif, le temps de résolution des bugs risque d'exploser et la confiance entre les équipes en pâtira.

Concrètement, pour tout nouveau projet qui se lance, il y a de nombreux impératifs techniques à rassembler. Toutes ces facettes doivent correspondre à l'environnement de production et être discutées avec les Ops. Les techniques et outils utilisés en phase de développement doivent être le reflet de ceux utilisés durant l'exploitation, afin de découvrir les blocages au plus tôt.



# USINE LOGICIELLE

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

- **Un dépôt de code versionné**

- Git ou Mercurial rempliront le job sans difficulté. Ils permettent tous deux de faire des commits locaux aux machines des développeurs avant d'envoyer leur travail par paquet au dépôt central. Cela leur donnera la possibilité de faire des commits plus petits et donc, de réduire grandement les problèmes de merge en intégrant leur code au tronc commun.
- Il doit être possible pour n'importe quel développeur de créer de nouveaux dépôts simplement. Pour cela, des outils comme GitLab, GitBlit ou Rhode-Code vous offriront une interface Web de gestion des utilisateurs et des dépôts de code.

- **Un serveur d'intégration continue**

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

- **Un dépôt d'artefact**

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

- Si votre intégration continue délivre des fichiers WAR ou un autre artefact typé Java, un dépôt Maven tel que Archiva ou Nexus est à conseiller.
- Si vous souhaitez pousser l'intégration continue à produire des paquets systèmes Linux (des .deb ou des .rpm par exemple), il faudra alors adopter le mode de distribution ad-hoc.

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

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



GIT

DEV@  
CLOUDBEES

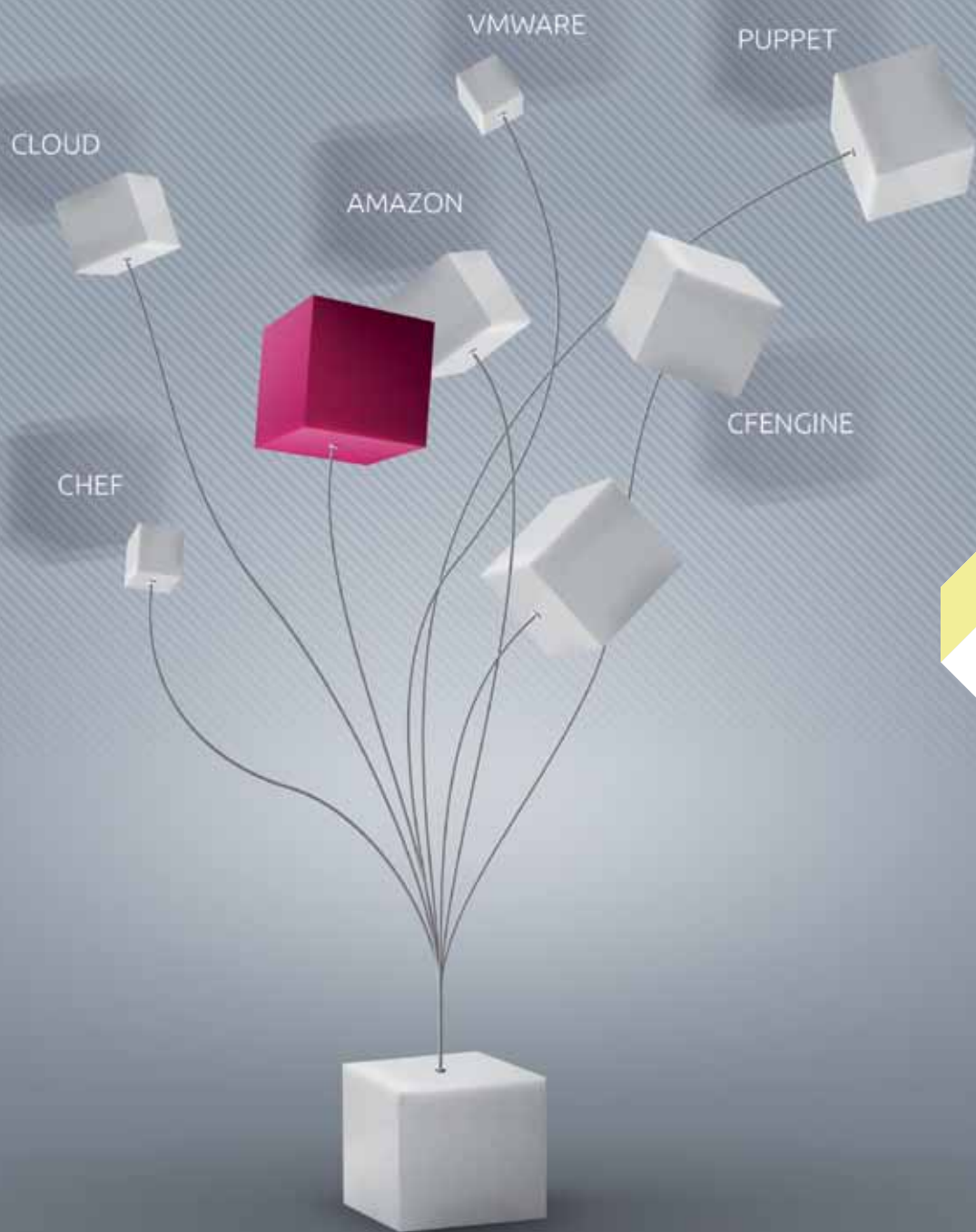
PACKAGE

INTÉGRATION  
CONTINUE

NEXUS



TESTS



## INFRASTRUCTURE

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

La virtualisation des différents environnements, du développement à la production, garantit :

- une bonne réactivité dans le provisionning,
- la reproductibilité totale de l'installation des plateformes,
- la possibilité d'obtenir à moindre coût de l'élasticité sur la plate-forme de production,
- la possibilité de recréer à la demande une plate forme depuis zéro (pour les disaster recovery ou bien les tests de charge par exemple).

En utilisant cette approche depuis la phase de développement, et en considérant que les Ops participent à l'équipe, on obtiendra une infrastructure versionnée, déployable à la demande dans les environnements choisis (développement, recette ou production) en restant iso-production. La mise en place de répartiteurs de charge, de groupes de scaling automatiques et de groupes de sécurité doit également faire partie de ce provisionning et être rédigée par toute l'équipe, pour répartir la connaissance.

## GESTION DE CONFIGURATION

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

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

## LE MONITORING ET L'ALERTING

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

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

Malheureusement, cette culture de l'ouverture reste trop souvent cantonnée à la communication interne à l'équipe : les Devs gèrent leurs indicateurs de qualité de code, tandis que les Ops suivent leurs dashboards d'état de l'infrastructure. L'information circule mal entre les différents services.

Il y a pourtant beaucoup à apprendre en apportant un peu de mixité dans ces tableaux de bords opérationnels. En effet, si en plus de suivre des métriques système comme les consommations CPU ou mémoire, on ajoute :

- quelques mesures techniques liées au middleware, comme la mémoire de la JVM par exemple,
- quelques mesures liées à l'applicatif, comme le taux d'occupation d'un pool de ressources,
- quelques mesures métiers, comme le temps d'attente d'un internaute au moment du paiement,

... alors, l'information affichée s'enrichit et permet une compréhension plus rapide et plus complète des comportements du SI dans son ensemble. Elle devient commune aux équipes et permet donc une collaboration plus efficace.

*Cette tendance à apporter de la visibilité est d'ailleurs contagieuse. On observe, dans les entreprises où le SI utilise ces techniques, un intérêt et une adhésion des équipes métier. Les directions marketing vont demander aux Ops si elles peuvent avoir, dans leur bureau, ce beau dashboard qui expose les chiffres de vente et l'état de santé du business en temps réel.*

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

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

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



*Comparaison J et J-7 des fréquentations et des ventes sur un site marchand, avec Graphite*

On ne peut évidemment pas rester toute la journée les yeux rivés sur les compteurs, c'est pourquoi il est bon d'avoir un système d'alerting efficace. Classiquement, on déclenche des alertes sur des passages de seuils, comme "le CPU de telle machine dépasse 90 %". Avec un couple Graphite-JmxTrans, il est maintenant possible de mettre en place des alertes sur des tendances, comme "le CPU de telle machine dépasse en moyenne les 90 % depuis plus de 10 minutes". Comme la collecte des métriques est centralisée, il est même possible de croi-

ser plusieurs sources d'information pour déclencher des alertes très ciblées, comme « sur les 10 dernières minutes, le CPU dépasse en moyenne les 90%, le pool d'accès aux services externes est saturé et le temps de parcours client au moment du paiement est supérieur de 50% au temps moyen constaté en fonctionnement normal ». Si cette dernière alerte se déclenche, alors il est temps d'avertir mon partenaire paiement que son service est dégradé. Cet alerting peut se faire via des outils classiques comme Nagios ou, bien sûr, un logiciel open source dédié tel Seyren.

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

En conservant des données métiers sur de longues durées, accessibles par tous via des dashboards paramétrables, les décideurs ont à disposition un nouvel outil de pilotage. Ce genre de monitoring temps réel est infiniment plus réactif et plus facile à exploiter qu'un rapport PDF quotidien. Il est, par exemple, très facile, dans le cadre d'un site Web marchand, de superposer sur une même courbe, en temps réel, les chiffres de vente actuels, à J-7, J-30 voire même J-365 si cela a un sens « métier ».

Exposer ces valeurs à la vue de tous, développeurs, opérationnels, décideurs, fonctionnels, permet de donner un sens aux développements réalisés au quotidien et de mobiliser une équipe autour d'une « cause commune ».

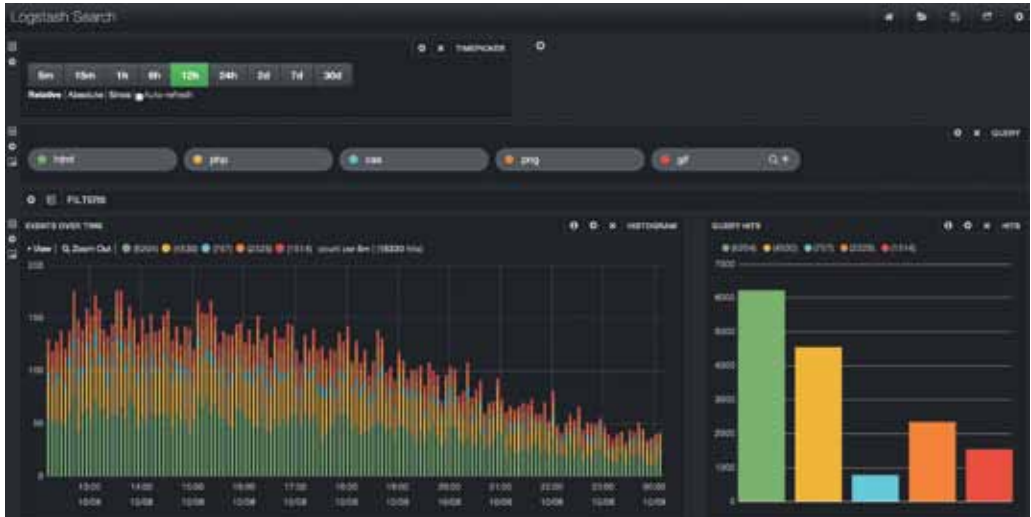
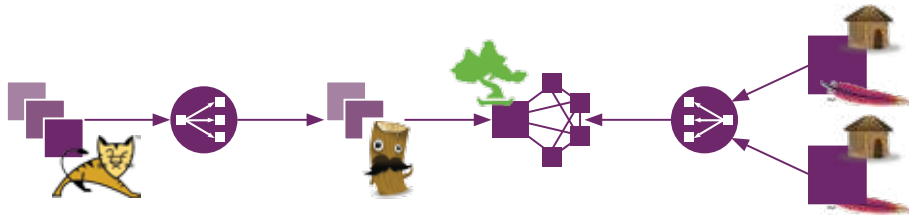
## ANALYSE D'INCIDENT

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

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

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

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



*Le pipeline de collecte de logs, avec LogStash, Elasticsearch, et finalement la visualisation de dash boards Kibana. Ce genre de solution existe aussi dans le Cloud, avec Loggly ou PaperTrail.*

## LA MESURE DE LA PERFORMANCE

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

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

- *pratiquer des tests de charge continus,*
- *mesurer les performances réelles en production.*



## Tests de charge continus

Le principe est simple : en même temps que les tests d'intégration (par exemple lors du "nightly build" quotidien), un environnement multi-instances iso-production est créé, grâce au provisionning automatique déjà mis en place, et un tir de performance est lancé.

Autrefois chasse gardée d'éditeurs spécialisés, le monde des logiciels d'injection s'est ouvert à l'open source, en particulier avec le projet Gatling. Ce dernier permet de scripter des scénarios métiers et de générer un trafic important sur une plateforme cible. Bien que la mise à disposition d'une plateforme iso-production dédiée soit la situation idéale, il n'est pas toujours facile de disposer des supports financiers et de l'infrastructure requise.

Pour ces tests en continu, n'hésitez pas à dimensionner vos ressources d'infrastructure sur des limites basses, mais pas trop éloignées de votre production standard. Dans un environnement virtualisé, vous pouvez également choisir de recycler les ressources des environnements de développement, au profit des tests de performances pendant la nuit ou le weekend.

Grâce à ce procédé, il devient possible pour les développeurs de suivre la performance générale de leur application au jour le jour et de procéder aux ajustements nécessaires au fil de l'eau. On quitte le schéma typique qui consiste à attendre une livraison officielle de l'application avant de déclencher une campagne de tests de charge.

De la même façon, les Ops seront mis au courant plus tôt d'éventuels besoins de nouvelles ressources d'infrastructure. Le temps économisé est précieux, le projet gagne en sérénité.

## Performances réelles

Autre axe d'amélioration possible, la mesure de la performance en live, sur l'environnement de production. Plusieurs cas peuvent vous pousser dans cette direction :

- les développeurs demandent un outil de profiling en production, car ils n'arrivent pas à reproduire les ralentissements constatés sur leurs environnements,
- le parcours d'une requête emprunte tellement de bifurcations dans le SI qu'il devient impossible de la suivre et de diagnostiquer où le temps est consommé,
- la corrélation entre les indicateurs JMX, les indicateurs métiers et les différents tiers n'est pas possible car toutes les couches ne renvoient pas le même niveau d'information.

Il faut alors se tourner vers des logiciels d'APM. Le principe est simple : on ajoute un agent, dont l'overhead est le plus faible possible, à chaque couche logicielle. Ces agents enrichissent les requêtes afin de tracer la vie d'une requête utilisateur au travers de toutes les briques du SI. Cette trace est envoyée à un serveur central qui dispose d'une interface de consultation.

L'avantage corollaire de cet outillage est de permettre de tracer une cartographie complète du SI, non pas à partir des diagrammes d'architecture initiaux, mais à partir du fonctionnement réel.



Grâce à ces solutions, vous découvrirez peut-être que la file d'attente prévue pour lisser certains échanges entre application n'est pas utilisée par tout le monde.

Malheureusement, dans le monde de l'APM, l'open source n'est pas au niveau des éditeurs spécialisés. La création d'un agent ayant une empreinte très faible sur une application requiert de fortes compétences techniques et beaucoup de travail. Il est chimérique de tenter de construire son propre système. L'adéquation entre le besoin et l'investissement parfois élevé est un paramètre déterminant du passage à l'APM.

### Deux éditeurs ont notre préférence :

- AppDynamics, qui propose un logiciel outillant nativement de nombreuses interfaces Java (jms, cache, http, jdbc...), disponible sur site.
- NewRelics, disponible uniquement dans le Cloud, pour un coût plus faible que son concurrent mais des fonctionnalités moins riches.



*Une infrastructure complexe découverte et visualisée via AppDynamics*

## VALUE STREAM MAP

Tous les outils et méthodes que nous venons de passer en revue vont donner beaucoup plus de pouvoir à vos équipes sur votre SI. Ils seront en mesure d'observer, de comprendre, de diagnostiquer et de réparer au plus vite. On peut néanmoins aller plus loin encore. Un autre outil permet d'analyser le fonctionnement de vos équipes plutôt que celui de votre SI : la Value Stream Map.

Cet outil issu du Lean Management permet de calculer l'efficacité de votre processus et d'identifier les points d'amélioration avec une vue d'ensemble. Dans un contexte DevOps, il s'agira d'identifier précisément toutes les étapes nécessaires au déploiement d'un binaire en production depuis sa livraison par les Devs. Nous associons à chaque étape sa durée de travail effectif et le temps de latence constaté avec l'étape précédente. Le ratio entre le cumul des temps de travail effectif et le cumul des temps d'attente vous donne votre efficacité globale. Il s'agira ensuite de cibler votre effort d'amélioration pour augmenter ce ratio.

*L'objectif est de réduire à la portion congrue les temps de latence et les étapes entre le moment où un binaire est prêt pour la production et le moment où il est effectivement actif. Avant d'atteindre le graal que représente la livraison continue, vous pouvez avancer dans la fluidification des échanges à l'aide d'une Value Stream Map.*

Vous pouvez utiliser une Value Stream Map :

- à l'issue d'améliorations techniques du SI (monitoring, alerting, mesures de performances, etc) pour identifier de nouveaux chantiers.
- avant toute amélioration technique, pour mesurer les gains au fil des changements et prioriser les chantiers à mener.

# Take away DevOps Trends



## FLUIDIFIER

- *Si c'est difficile, faites-le souvent. Par la répétition vient la fluidité et la fiabilité.*
- *Penser "as a service", l'infrastructure, la plateforme, les middleware, la configuration sont des services automatisés.*
- *Augmenter la visibilité, le monitoring doit être le plus riche possible et vu par tous. Toutes les métriques sont bonnes et l'alerting doit inclure les Devs.*

# Livrer



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

# MISE EN PLACE DU DÉPLOIEMENT CONTINU

Voici les bonnes pratiques à suivre lorsque vous souhaitez mettre en place du Déploiement Continu, afin que vous puissiez en tirer tous les bénéfices :

- Construisez votre application une seule fois et faites de la promotion de build. Cela implique que votre application doit pouvoir être configurée par son environnement, et que vous livrez en production le même binaire qui a été testé en recette, en validation métier, etc.
- Déployez de la même manière sur chacun de vos environnements, que ce soit sur le poste de travail du développeur, sur un environnement de test ou en production. Votre processus de livraison est ainsi partagé par tous et donc plus sûr. Pour cela, la configuration propre à chacun de vos environnements (serveur d'application, base de données et autres middlewares), ne doit pas être contenue dans votre package applicatif. Nous vous conseillons de mettre en place un serveur de configuration qui aura la charge de configurer votre application lors de son déploiement.
- Exécutez systématiquement un Smoke test lors de chaque déploiement de votre application afin de vérifier que celle-ci est bien disponible. Ce smoke test doit rester simple (par exemple, vérifier la disponibilité de votre page d'accueil) afin de pouvoir être rapide, systématique et maintenable dans le temps.
- Déployer au sein d'environnements similaires à votre production. Pour éviter toute surprise lors des mises en production, veillez à utiliser, pour votre intégration continue ou vos tests, des environnements les plus proches possibles de celui de la production. Pour ce faire, nous vous conseillons de maintenir vos environnements simples et d'automatiser leur provisionning.
- Chaque changement doit être propagé immédiatement au travers de l'ensemble des étapes de votre deployment pipeline (ensemble des validations permettant la mise en production), afin que les erreurs soient détectées au plus tôt. Nous vous recommandons d'utiliser un système visuel de type "Build Wall" (Visuwall par exemple), partagé par l'ensemble des équipes, permettant de visualiser toutes les étapes de votre deployment pipeline.
- En cas d'échec sur l'une des étapes de votre deployment pipeline, la chaîne doit être interrompue et la priorité de tous devient la correction du problème. Cela est indispensable pour maintenir un processus de déploiement rapide, répétable et sûr.

## GESTION DES BRANCHES

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

### Simple Branching

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

C'est le fonctionnement le plus répandu à ce jour. La version de production est maintenue sur une branche permettant de livrer des corrections, tandis qu'une ou plusieurs équipes travaillent sur des versions futures. Lorsqu'une version de la branche de développement est jugée satisfaisante, on déverse ses modifications sur la branche de production, qui devient la nouvelle version standard. Puis, les Devs continuent sur la branche de développement.

### Feature Branching

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

C'est une approche très puissante car elle permet de se concentrer sur la livraison unitaire de valeur, et donc, de faire beaucoup de mises en production rapides et à faible risque. Cependant, elle implique un effort supplémentaire en intégration continue, puisqu'il faut pouvoir construire et tester chaque branche de développement.

## PLASTICITÉ APPLICATIVE

*Dans une optique de livraison continue, il peut arriver qu'une fonctionnalité doive être livrée, alors qu'une autre est encore instable. Si vous avez choisi le Feature Branching, pas de problème. Mais que faire dans le cas contraire ?*

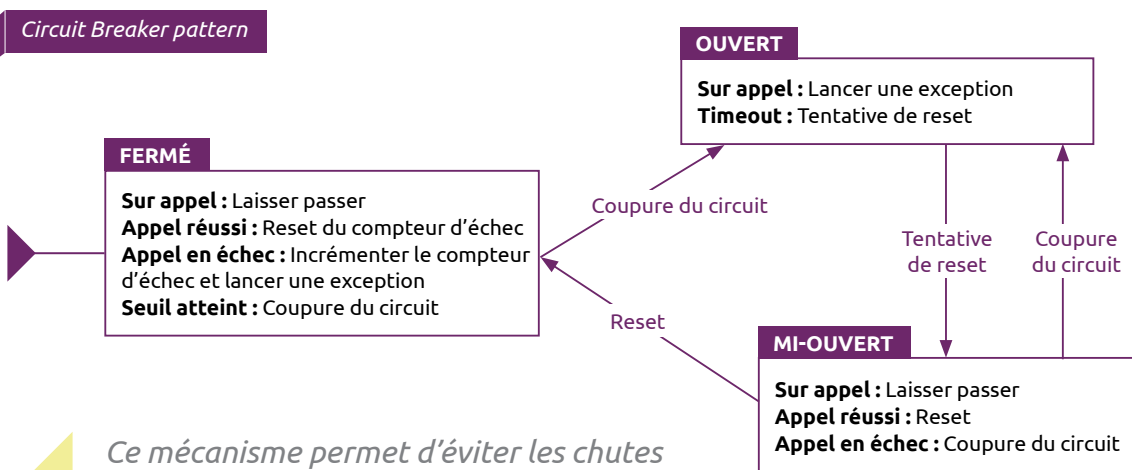
## Toggle Feature Pattern

Le Toggle Feature pattern est un outil de conception logicielle. Les Devs peuvent positionner *des interrupteurs* dans le code de l'application pour encapsuler des fonctionnalités entières. Ces derniers doivent être activables par configuration et manipulables à chaud par les Ops pour modifier le comportement de l'application.

En protégeant les fonctionnalités en cours de rédaction par le biais de ce mécanisme, il est possible maintenir le rythme et déployer des fonctionnalités, même si d'autres sont encore en chantier.

## Circuit Breaker Pattern

Outre la ségrégation des fonctionnalités non terminées, toute fonctionnalité reposant sur des services externes doit pouvoir être activée ou désactivée à chaud, sans stopper l'ensemble de l'application. Le monitoring peut avertir les Ops qu'un service est indisponible, mais il est également possible d'intégrer, directement dans les applications, des mécanismes de protection contre les pannes externes.



*Ce mécanisme permet d'éviter les chutes en cascade et minimise l'impact d'une panne dans le SI. De plus, si le problème provient d'une charge d'appels trop importante et que les applications consommatrices se mettent en pause, cela permet d'analyser sereinement la panne mère, sans être attaqués par toutes les briques dépendantes.*

Le principe du pattern Circuit Breaker est de conditionner les accès vers des composants externes par des interrupteurs logiques qui détectent les appels en erreur. Quand un seuil (configurable) d'erreurs consécutives est atteint, l'application passe en mode dégradé et n'assure plus la fonctionnalité qui dépend du service en panne. Pendant que l'application est en mode dégradé, elle envoie une alerte au monitoring et déclenche une procédure pour vérifier régulièrement si le service est revenu. Dès que cette procédure détecte un retour du service, l'application repasse en mode standard et assure de nouveau la fonctionnalité.

## **A/B Testing**

Une fois la technique du Toggle Feature Pattern maîtrisée par vos équipes, il est possible de l'utiliser, non plus seulement pour masquer une fonctionnalité instable, mais aussi pour proposer aux utilisateurs différentes versions d'une même fonctionnalité.

En effet, de plus en plus d'entreprises procèdent à des tests grandeur nature, en production, par échantillonnage. Par exemple, via la mise en place d'un A/B testing : 2 pages d'accueil, au design différent, sont proposées de manière aléatoire à 2 populations d'internautes. Le monitoring permet de dégager des tendances indiquant la page la plus pertinente.

Le principe du A/B Testing n'est pas limité à 2 options et il est toujours possible de jouer sur différentes nuances d'ergonomie, de simplicité de parcours, etc. La pertinence du A/B Testing doit se mesurer en valeur métier. Typiquement, pour un site marchand, on attribuera le taux de transformation en vente d'une version de l'application. La meilleure version devient la norme et on tente d'autres alternatives sur cette nouvelle base pour améliorer encore l'expérience utilisateur.

## **Dark Launch**

Autre pratique corollaire du Toggle Feature et de l'A/B Testing : le Dark Launch, qui permet de tester des fonctionnalités alternatives considérées comme risquées. On va alors activer une fonctionnalité à une frange des utilisateurs, en se basant sur des critères certains (peu d'utilisateurs, leur type de compte, leur nationalité, leurs préférences, etc.).

Une fonctionnalité typiquement considérée comme à risque est la modification du parcours de paiement pour un site marchand. Comme c'est à cette étape que se concentre la majorité des abandons d'achat, il est très délicat de décider de la modifier. L'utilisation d'un Dark Launch pour confirmer la pertinence d'une modification avant de la généraliser est ici appropriée.



# STRATÉGIES DE DÉPLOIEMENT

*Votre projet s'appuie maintenant sur un processus de déploiement bien rodé, au sein duquel l'ensemble des étapes de build, de tests et de provisioning sont automatisées. Vous êtes capable de livrer votre application en continu mais le déploiement en production nécessite toujours un Go/NoGo. L'étape suivante consiste à automatiser le déploiement en production en tant qu'action finale de la chaîne de production. Comme il n'est pas acceptable qu'une modification du code aboutisse à des coupures de service à chaque mise en production automatique, il faut mettre en place une stratégie de déploiement sans interruption de service.*

Différentes techniques peuvent être considérées afin de réussir à diminuer les risques et dépasser les craintes liées à un tel bouleversement dans la manière de livrer une application en production. Le but est de transformer le déploiement en un non-événement, quelque chose de standard.

## **Blue/Green Deployment**

Cette technique de déploiement sans interruption de service repose sur un système de double run. On maintient 2 environnements de production pleinement fonctionnels. Tous les accès utilisateurs passent par un routeur qui permet la bascule vers l'un ou l'autre de ces environnements. La version vers laquelle pointe le routeur est l'environnement Green, tandis que la partie en stand-by est appelée le Blue.

La technique consiste à déployer la nouvelle version sur le Blue, faire toutes les vérifications utiles pour valider le déploiement, puis basculer le trafic du routeur.

De cette façon, il est possible de basculer instantanément les utilisateurs sur une nouvelle version, de façon transparente. De plus, en cas de problème, il est tout aussi rapide et indolore de faire la bascule inverse pour rester sur la version stable.

Une fois le trafic basculé, l'environnement "nouvelle version" devient le Green, et on peut sereinement mettre à jour le second déploiement pour maintenir l'équilibre entre Blue et Green.

Le point délicat de cette approche est la prise en compte des éventuelles migrations de bases de données. Comme tout doit être dupliqué dans une stratégie Blue/Green, la base de données doit l'être également. Pour procéder à la bascule de base, on met la base Green en lecture seule, ce qui constitue tout de même une dégradation de service, mais pas une coupure. On recopie l'intégralité de la base Green vers la Blue, on y applique les migrations nécessaires, puis on peut faire la bascule de routeur.

Il va de soi que l'utilisation de mécanismes de propagation de modifications master-slave est à conseiller pour rendre la dégradation de service la plus courte possible.

## Canary releasing

Dès que votre infrastructure de production commence à prendre de l'ampleur, il peut devenir problématique de rester dans une approche Blue/Green Deployment. Redonder une grappe de serveurs frontaux, un nuage de middleware et plusieurs bases de données peut rapidement coûter très cher. Vous pourriez alors être tenté par une stratégie nommée Canary Releasing. Cela consiste à déployer la nouvelle version de son application sur un sous-ensemble des noeuds qui composent votre environnement de production.

Cela peut se faire soit en augmentant légèrement le nombre de vos instances durant le déploiement, soit en mettant à jour un ensemble restreint d'instances existantes. Dans tous les cas, il faut être capable de maîtriser quelle proportion de votre trafic va être dirigée vers ces instances avec la nouvelle version de votre application.

Vous devrez porter une attention particulière à ces instances pour valider que la nouvelle version peut être généralisée. En cas de problème, il vous suffira de couper le trafic entrant sur ces instances pour arrêter de perturber vos utilisateurs.

Une fois votre stratégie choisie, vous allez avoir besoin d'un outil pour gérer le déploiement à proprement parler de l'applicatif.

## Rundeck

Dans une grande majorité des cas, les Ops ont déjà réalisé une base solide de scripts chargés de déployer les applications. Ils se chargent la plupart du temps de jouer les scripts SQL sur la base de données, de distribuer le ou les nouveaux binaires sur les serveurs, pour finir par générer ou rafraîchir la configuration des instances applicatives avant de recharger l'application.

Rundeck propose de tirer le meilleur des scripts existants en les centralisant dans une application capable de les exécuter à distance sur un parc de serveurs. Vous définissez votre Workflow de commandes (les scripts à exécuter dans un certain ordre) ainsi que les machines sur lesquelles le Job sera exécuté. Les Ops et les Devs pourront facilement enrichir l'outil avec des scripts maison. Un même Job peut-être utilisé pour déployer la production, la pré-production et la plateforme de validation.

Devs et Ops auront donc un outil commun pour réaliser les déploiements et d'autre tâches administratives. Cette solution open source a le mérite d'être simple et facilement utilisable. Ce projet ayant bonne presse dans la communauté Java, il bénéficie de plugins permettant de l'intégrer avec beaucoup d'outils comme Chef et Jenkins par exemple.

Comme souvent dans le monde de l'open source, Rundeck fournit une interface simple pour ne pas dire épurée ou simpliste. Enfin, n'oubliez pas que Rundeck ne fournit aucune intelligence particulière pour la problématique des rollbacks par exemple. Pour gagner en confort d'utilisation et en richesse d'utilisation, vous devrez vous tourner vers une solution éditeur.

## Deployit

Les équipes IT se retrouvent néanmoins confrontées aux difficultés suivantes :

- Écriture et maintien des scripts de déploiement.
- Gestion des centaines de paramètres de configuration propres à chaque application et environnement (de nombreuses équipes gèrent l'ensemble de ces paramètres - clés et valeurs - avec difficulté dans des fichiers Excel ou des applications maison).
- Utilisation de packages et de procédures différents en fonction des environnements.
- Retour à une application dans un état antérieur (rollback) manuelle et fastidieuse.
- Manque d'accès à des dashboards permettant de voir les tendances de déploiement sur les derniers mois pour les managers afin d'identifier les applications les plus longues à déployer ou les plus sujettes à rollbacks.

La solution Deployit a notre préférence. Elle est basée sur la même approche architecturale que Rundeck : l'exécution de commandes sur des serveurs distants sans agents propriétaires. Elle propose :

- Une approche où il n'est plus nécessaire de scripter. C'est Deployit qui génère à la volée, à chaque changement effectué, les séquences de déploiement nécessaires pour mettre à jour l'application dans son environnement. Pour cela, Deployit s'appuie sur un ensemble de plugins permettant de déployer nativement sur la majorité des plateformes Java et .Net du marché, ainsi que sur un grand nombre de bases de données et load balancers.
- De jouer le rôle de serveur de configuration à travers une approche unifiée, où les paramètres de configuration sont stockés au sein de la solution dans des dictionnaires centralisés, versionnés et partageables entre applications et environnements. Ces dictionnaires permettent très simplement de configurer correctement les applications au moment du déploiement.
- En s'appuyant sur un référentiel centralisé, de livrer des packages de déploiement identiques quelque soit l'environnement sur lequel ils sont installés, configurés via les dictionnaires.
- Différentes interfaces permettant de manipuler le serveur en fonction de son rôle dans l'organisation : plugins pour les usines logicielles (Maven, Jenkins, Bamboo, TeamCity, TFS), interface Web ergonomique et une Interface en Ligne de Commande (CLI) pour l'intégration à des outils tiers ou automatiser au maximum certaines tâches. Une intégration avec des outils de provisioning comme Puppet permet également de déclarer automatiquement l'infrastructure créée par Puppet dans le référentiel de Deployit.
- Une fonctionnalité de rollback en cas de problème sur une tâche de déploiement.
- Des tableaux de bord complets et un module d'audit trail permettant de tracer toutes les actions réalisées sur le référentiel.

Deployit apporte une philosophie et un fonctionnement très DevOps :

- les 'Dev' construisent des packages applicatifs, les importent dans Deployit (via leurs usines logicielles - Jenkins par exemple) et déploient sur leurs environnements de développement / test,
- les 'Ops' construisent et maintiennent à jour l'infrastructure et les environnements dans Deployit (via leurs outils de Provisioning - Puppet par exemple) et déploient sur les environnements qui sont de leur responsabilité (en s'appuyant sur des procédures de déploiement déjà rodées en amont du cycle de vie),
- les équipes de QA peuvent accéder au référentiel de Deployit pour installer des packages applicatifs sur leurs environnements et réaliser des tests à la demande et sans avoir à se soucier des problématiques d'infrastructure.

*La solution masque donc la complexité des déploiements. Elle permet la collaboration entre Dev, QA et Ops et favorise le "self service deployment". Elle fournit également une grande traçabilité, tout en accélérant la livraison, en sécurisant les MEP et en réduisant les actions manuelles fastidieuses. Deployit est éditée par XebiaLabs, marque de l'alliance Xebia.*

# Take away DevOps Trends



## LIVRER

- *Mettre en place un pipeline de déploiement en automatisant votre processus de livraison et le transformer en Pull System.*
- *Supprimer les barrières du déploiement continu en s'appuyant sur les techniques du pattern Circuit Breaker et du Toggle Feature.*



# Conclusion

Dans cette démarche DevOps que nous avons initiée, nous nous sommes attachés à rapprocher les équipes d'opération des équipes de développement. Par sa sociologie, Xebia est une organisation de développeurs, les méthodes et outils sont donc proposés sur la base de notre expérience de Devs et de coachs agile. Les Ops ne manqueront pas d'apporter leur pierre à l'édifice de cette nouvelle organisation, cet échange entre les équipes créera naturellement une synergie propre au contexte de chaque DSI.

Grâce à une implication forte des équipes de développements et des équipes d'opérations, le suivi de l'avancement des projets est assuré de manière transverse fournissant ainsi une vision globale et non plus spécifique à chaque équipe. La richesse des échanges vient améliorer, du point de vue de tous, les solutions finales. Les Devs prennent en compte intuitivement les besoins spécifiques des Ops et vice versa. D'autre part, les outils d'automatisation mis en place simplifient les tâches nécessaires au delivery de bout en bout. À terme, le temps perdu entre les développements et la mise en production diminue régulièrement. La value Stream Map permet de constater cette amélioration d'itération en itération.

Nous assistons à une diminution nette du time-to-market. Les développeurs passent moins de temps en début de projet à monter les environnements de développement et de test. La possibilité de bénéficier d'environnement iso-prod dès le début des projets permet de réaliser des tests efficaces qui lèvent les anomalies plus tôt. L'émergence de bugs est lissée sur chaque sprint, contrairement à la situation initiale qui faisait apparaître la plupart des bugs, parfois structurants, dans les dernières phases des projets. Le capacity planning est désormais plus fiable car il est mis à jour régulièrement en prenant en compte les entrants en avance. Cela évite les effets palier liés à la nécessité d'investir dans de nouveaux serveurs dont la livraison est chronophage. Lorsque l'application est fonctionnellement prête, l'exploitation en a déjà fait l'intégration, elle est donc déjà disposée pour la production.

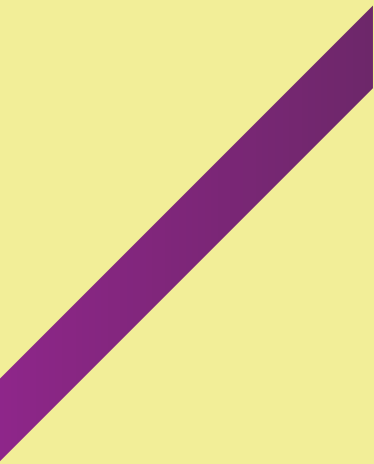
Au delà du time-to-market, les dates de livraison sont stabilisées, le délai entre le lancement des développements et la mise en production est mieux maîtrisé. La découverte des bugs lissée permet d'éviter l'effet

d'urgence et la multiplication du "vite fait, mal fait". Les équipes sortent du fonctionnement en mode pompier et peuvent se concentrer sur la qualité des applications et de l'infrastructure. L'innovation et la veille technologique se développent dans la DSI. D'un point de vue managerial, le planning des releases obtient des niveaux de confiance inégalés jusqu'alors. Un rythme s'est installé, les mises en productions sont régulières et sécurisées, les interventions d'urgence se raréfient.

Attention, il ne s'agit pas de recette miracle ! Le changement d'organisation s'est accompagné de nombreuses difficultés, la résistance au changement est toujours forte, mais petit à petit le dialogue s'est installé et le fonctionnement s'est équilibré. N'oublions pas que nous parlons de développement logiciel et d'informatique, la vision idyllique consistant à dire qu'aucun bug n'apparaît en production n'est pas de mise ici. Toutefois, le monitoring avancé, accompagné du système d'alarme mis en place sur les applications, permet de d'identifier rapidement les erreurs. La visibilité étant maximale, chaque acteur du projet peut identifier et remonter l'anomalie.

Dans ces cas de crises, les équipes concernées disposent de moyens conséquents pour analyser et identifier la source du problème. Qu'ils soient Devs ou Ops, ils peuvent utiliser le système de monitoring pour cibler l'application voire même le serveur responsable de la panne. Une fois cette première analyse réalisée, plusieurs options sont disponibles, les uns préféreront utiliser le moniteur de performance, les autres identifieront un cas concret et analyseront les logs pour déterminer le code source incriminé. Les rétrospectives permettent d'améliorer la démarche en cas de crise. Nous notons également une diminution du temps de résolution des incidents grâce à l'aide des outils mis en place, de la supervision à la centralisation des logs en passant par les analyseurs de performance et le déploiement continu.

Nous avons présenté un grand nombre d'outils destinés aux Devs comme aux Ops, qui peuvent aussi trouver leur utilité dans les équipes métier. Tous ces systèmes ne serviront néanmoins que très peu sans une coopération forte entre les équipes et une adhésion aux principes du DevOps.







# Take away DevOps Trends



## COOPÉRER

- *Organiser des rétrospectives régulières entre Devs et Ops plutôt que des post-mortem*
- *Suivre la résolution de problèmes avec un format A3 pour s'assurer de résoudre les causes racines des problèmes*
- *Impliquer les Ops dans la constitution du Product Backlog et les revues de sprint*
- *Utiliser le management visuel pour accroître la transparence et l'anticipation des travaux*



## FLUIDIFIER

- *Si c'est difficile, faites-le souvent. Par la répétition vient la fluidité et la fiabilité.*
- *Penser "as a service", l'infrastructure, la plateforme, les middleware, la configuration sont des services automatisés.*
- *Augmenter la visibilité, le monitoring doit être le plus riche possible et vu par tous. Toutes les métriques sont bonnes et l'alerting doit inclure les Devs.*



## LIVRER

- *Mettre en place un pipeline de déploiement en automatisant votre processus de livraison et le transformer en Pull System.*
- *Supprimer les barrières du déploiement continu en s'appuyant sur les techniques du pattern Circuit Breaker et du Toggle Feature.*





## *Les auteurs*



Aurélien  
Maury



Pablo  
Lopez



Séven  
Le Mesle



Gilles  
Mantel



Audrey  
Pedro



Benoit  
Moussaud



Richard  
Mathis



Jean-Louis  
Rigau

**Xebia**



SOFTWARE DEVELOPMENT **DONE RIGHT**

*est une entreprise agile qui délivre des logiciels*  
***sur mesure***  
*à ses clients*

**nos valeurs fondatrices,  
clé de notre succès**

