WAVESTONE

DEVOPS

LE SECRET DE L'AGILITÉ DES START-UPS À LA PORTÉE DES GRANDS COMPTES

WAVESTONE

www.wavestone.com

Wavestone est un cabinet de conseil, issu du rapprochement de Solucom et des activités européennes de Kurt Salmon (hors consulting dans les secteurs retail & consumer goods en dehors de France).

La mission de Wavestone est d'éclairer et guider ses clients dans leurs décisions les plus stratégiques en s'appuyant sur une triple expertise fonctionnelle, sectorielle et technologique.

Fort de 2 500 collaborateurs présents sur 4 continents, le cabinet figure parmi les leaders indépendants du conseil en Europe et constitue le 1er cabinet de conseil indépendant en France.

DEVOPS : VERS L'AGILE DE BOUT-EN-BOUT

À l'heure des disruptions digitales, les entreprises cherchent par tous les moyens à gagner en agilité et à améliorer le *time-to-marke*t de leurs produits.

Avec un *leitmotiv* pour y parvenir : l'agilité !

Mais l'agilité ne se décrète pas. Elle implique une transformation à tous les étages, depuis les méthodes de travail jusqu'à l'architecture même des systèmes d'information, en passant par les organisations et les processus de décision.

Si les méthodes agiles commencent effectivement à se déployer dans la conception et le développement des produits, elles se heurtent encore à un mur infranchissable qui sépare les « dev » et les « ops ».

Ce mur s'est créé progressivement à travers la construction de systèmes d'information de plus en plus lourds, monolithiques et fortement imbriqués entre eux. À cela se sont ajoutés des silos organisationnels éloignant les équipes de développement et de production.

Pour changer, et aller vers le DevOps à grande échelle, il faut nous inspirer des pratiques des GAFA. Ces géants du web ont en effet eu la chance de pouvoir construire leur informatique en mode *green field*, en pensant dès le départ la modularité et l'agilité.

Ce qu'ils nous apprennent, c'est que le Devops est un subtile mélange de modularité et de grande autonomie d'un côté, et de règles et cadre de fonctionnement très stricts de l'autre, pour assurer l'assemblage de l'ensemble.

Cette publication présente les principes de mise en place du Devops et nos convictions pour bien entamer cette transformation en profondeur que doivent mener toutes les DSI.

Excellente lecture à tous !

LAURENT BELLEFIN
Directeur Associé

AUTEURS

Maximilien Moulin

Maximilien est manager en Architecture des Systèmes d'Information, spécialisé sur les sujets Cloud et Agilité. Diplômé de l'INSA de Lyon, il débute sa carrière dans l'architecture du SI interne d'Orange. Après une aventure entrepreneuriale il rejoint Wavestone où il conseille depuis 4 ans les grands comptes sur leur stratégie *Cloud* & Agilité.

maximilien.moulin@wavestone.com

Hasmik Manouchian

Hasmik est manager chez Wavestone dans la practice IT Data Architecture. Elle est spécialisée sur les sujets d'architecture technique, d'agilité et de gouvernance. Diplômée à l'IAE Lyon 3, elle rejoint Wavestone. il y a 6 ans. Elle accompagne ses clients dans leurs projets de transformation.

hasmik.manouchian@wavestone.com

Pascal Bour

Pascal est manager spécialisé en architecture technique et industrialisation de la production. Diplômé de l'ENSEIRB, il accompagne depuis 8 ans les clients de Wavestone lors des phases de conception de leurs projets structurants, ainsi que pour sécuriser et optimiser leurs exploitations.

pascal.bour@wavestone.com

Cette publication a été rédigée en collaboration avec Matthieu Barret, Franck Lenormand et Riyad Yakine.

DEVOPS
ÊTRE AGILE DE BOUT-EN-BOUT

La fourniture rapide, fiable et pertinente de services IT est devenue, quel que soit le secteur d'activité, une composante essentielle de la compétitivité des entreprises.

Pour gagner en réactivité, sans pour autant perdre en fiabilité et qualité, les grandes structures doivent transformer en profondeur leurs pratiques sur tout le cycle de production IT, de la conception au déploiement.

Cette transformation est déjà bien entamée à travers la démocratisation des méthodologies agiles, comme Scrum, qui a permis d'accélérer et d'augmenter le nombre de livraisons applicatives. Pour autant, freiné par des temps de livraison en production trop longs, ces méthodes ne tiennent pas encore toutes leurs promesses. Pour vraiment améliorer le *time-to-value*, c'est-à-dire le temps entre l'expression du besoin du métier et l'ouverture du service idoine, c'est l'intégralité de la chaîne de valeur IT, du métier aux opérations, qui doit être transformée. La démarche DevOps étend les pratiques agiles venues du développement au passage en production pour bénéficier pleinement de l'accélération des livraisons applicatives.

POUR VRAIMENT AMÉLIORER LE *TIME-TO-VALUE*, C'EST-À-DIRE LE TEMPS ENTRE L'EXPRESSION DU BESOIN DU MÉTIER ET L'OUVERTURE DU SERVICE IDOINE, C'EST L'INTÉGRALITÉ DE LA CHAÎNE DE VALEUR IT, DU MÉTIER AUX OPÉRATIONS, QUI DOIT ÊTRE TRANSFORMÉE

AMÉLIORER LE *TIME-TO-VALUE*
EN RAPPROCHANT LES OPS ET LES DEVS

Le DevOps est une démarche d'alignement des parties prenantes de la chaîne de valeur IT (business, développement, opérations mais aussi sécurité, architecture, compliance et autres fonctions transverses) sur un objectif métier commun, dans le but d'accroître la réactivité de l'entreprise sur son marché.

Cette démarche se matérialise par un ensemble de bonnes pratiques, méthodologies et outils qui servent deux objectifs :

- Aligner les objectifs entre Dev et Ops
- Automatiser l'ensemble du cycle de production

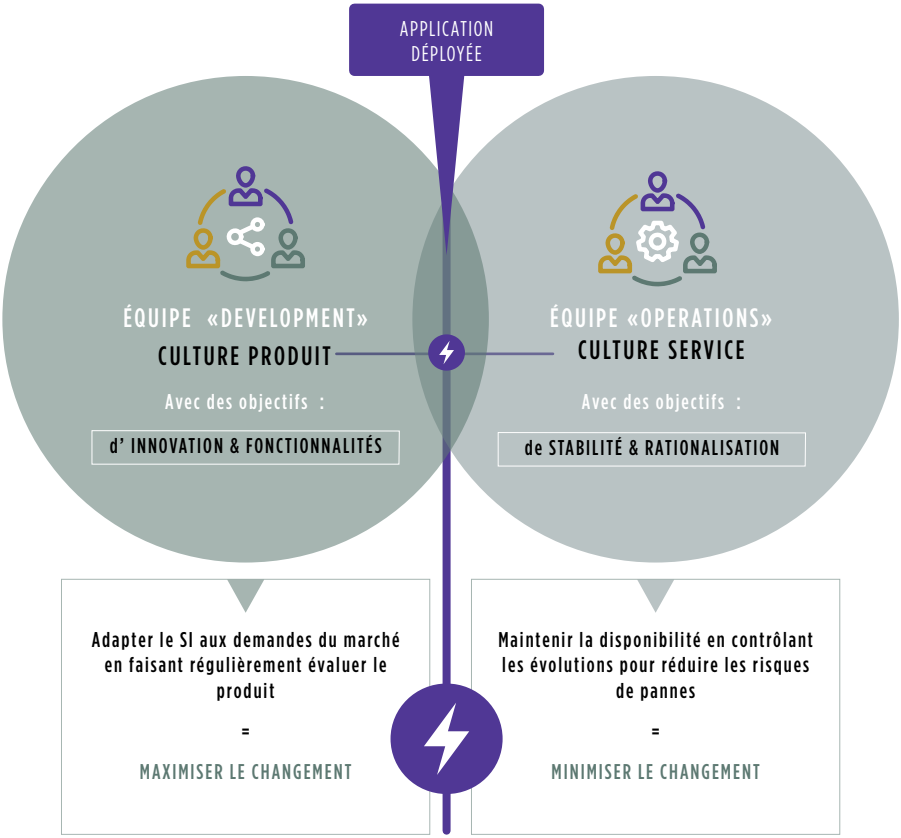
ALIGNER LES OBJECTIFS ENTRE DEV ET OPS

Le terme DevOps est né de la contraction des termes « *Development* » (équipes de développement) et « *Operations* » (équipes d'ingéniérie de l'infrastructure et d'exploitation de la production informatique). Historiquement les équipes assurant la construction et l'exploitation du SI ne faisaient qu'une, mais la complexité des architectures et la croissance importante du SI ont poussé à une séparation nette entre ceux qui font évoluer le SI (les Devs) et ceux qui assurent son exploitation et son bon fonctionnement (les Ops).

DEVOPS WASHING

Il n'y a pas à ce jour de définition unique faisant consensus sur le terme DevOps. Certains, comme Forrester, le définissent comme la création d'un *pipeline* de livraison logicielle automatisé ; d'autres, comme Gartner, insistent sur l'adoption de pratiques *Lean* et agiles pour livrer rapidement un service IT. Cette absence de définition précise et le « *branding* » de nombreux outils sous la terminologie DevOps, donne clairement la sensation de vivre un « *DevOps washing* », succédant les vagues de « *Green washing* » puis « *Cloud washing* » des dernières années. Il convient donc d'être vigilant et de ne pas se laisser prendre au jeu des étiquettes.

Le mur de la confusion



Aujourd'hui, la collaboration entre les équipes de développement et les opérations est souvent douloureuse. Leurs mandats respectifs a priori peu compatibles – innovation, évolution et réactivité pour les Devs, stabilité et fiabilité du SI pour les Ops – ainsi que leurs fortes dépendances mutuelles expliquent la complexité de leur relation. Cet état peut se traduire par un manque de performance mais aussi par un « agacement » mutuel.

Parce qu'elles n'impliquent que trop peu les équipes d'exploitation, les méthodologies agiles, telles qu'appliquées dans la plupart des grandes entreprises, n'ont pas remédié à ce problème. Au contraire, l'accélération du rythme de livraison applicative et la philosophie *test & learn* ont parfois creusé encore plus le fossé. Les équipes d'exploitation doivent déployer en production un nombre croissant d'évolutions, avec souvent une augmentation des taux d'erreurs alors que l'exigence de fiabilité reste constante.

Cela mène non seulement à la création d'un goulot d'étranglement au niveau du passage en production, mais aussi à toujours plus de « frottement » entre les équipes de développement et d'exploitation.

Les pratiques et outils DevOps permettent d'intégrer l'ensemble des acteurs de la chaîne de valeur IT au sein d'un processus unique, intégré et continu reposant sur les principes agiles (itérations, forte collaboration business, *test & learn*…) et sur une véritable culture de la collaboration entre les Devs et les Ops.

AUTOMATISER L'ENSEMBLE DU CYCLE DE PRODUCTION

Le DevOps repose également sur la mise en place d'outils d'automatisation sur tout le cycle de production IT : automatisation de la fourniture d'infrastructure, de la construction applicative, des tests et du déploiement applicatif.

Cette automatisation a pour intérêt premier d'absorber l'augmentation du volume et du rythme des tests découlant du fonctionnement itératif. Elle est donc indispensable pour garantir la qualité du code applicatif à chaque itération sans créer un goulot d'étranglement en aval de la construction applicative.

EXEMPLE DE KPIs

QUELQUES EXEMPLES D'INDICATEURS :	ET LES GAINS ESCOMPTÉS * :
Nombre de livraisons applicatives	▶ 200x plus de livraisons
Temps de mise en production	▶ 2 555x plus rapide
Nombre d'incidents	▶ 3x fois moins d'incidents
Temps de résolution (MTTR)	▶ 24x plus rapides à résoudre
Nombre de rollbacks	▶ 0 rollback
...	

* Puppet & DORA 2016 State of DevOps Report

Parce qu'elle permet de réduire les erreurs humaines et de faciliter la mesurabilité à chaque étape, l'automatisation dans la démarche DevOps est également un levier pour fiabiliser et améliorer de manière continue l'ensemble du cycle de fabrication et de déploiement.

Une automatisation performante et adaptée de l'ensemble du cycle est le principal levier pour éviter le goulot d'étranglement entre le développement et la mise en production. C'est la clé pour sortir les équipes d'exploitation du rôle de « pompier » (gestion d'incidents en continu) dans lequel elles tendent à être poussées suite à la démocratisation des pratiques agiles.

Le DevOps met en avant l'automatisation de l'ensemble du cycle de fabrication et de déploiement. De ce fait, avec l'outillage approprié tout devient mesurable sur l'ensemble du cycle. **La mesure est la clé qui permet l'adhésion des parties prenantes à la démarche et à son amélioration continue**. C'est par là même que l'on pourra lutter contre le « *DevOps washing* », mais aussi disposer des bons indicateurs pour démontrer les gains sur tout le cycle.

LE DEVOPS, N'EST PAS :

- UN PRODUIT FINI ou une démarche générique. Il n'y a pas une théorie générale applicable à tous, il est nécessaire d'adapter les bonnes pratiques DevOps à un contexte d'entreprise.
- UNE FUSION des missions des Devs et des Ops. En corollaire, DevOps n'induit pas nécessairement un changement de la structure organisationnelle; Devops n'impose pas non plus le regroupement des deux équipes au sein de la même.
- UNE APPROCHE *QUICK & DIRTY* ignorant les processus ou induisant une perte de contrôle. La mise en production reste d'ailleurs généralement à la main des Ops.

4 PILIERS POUR CONSTRUIRE LE DEVOPS

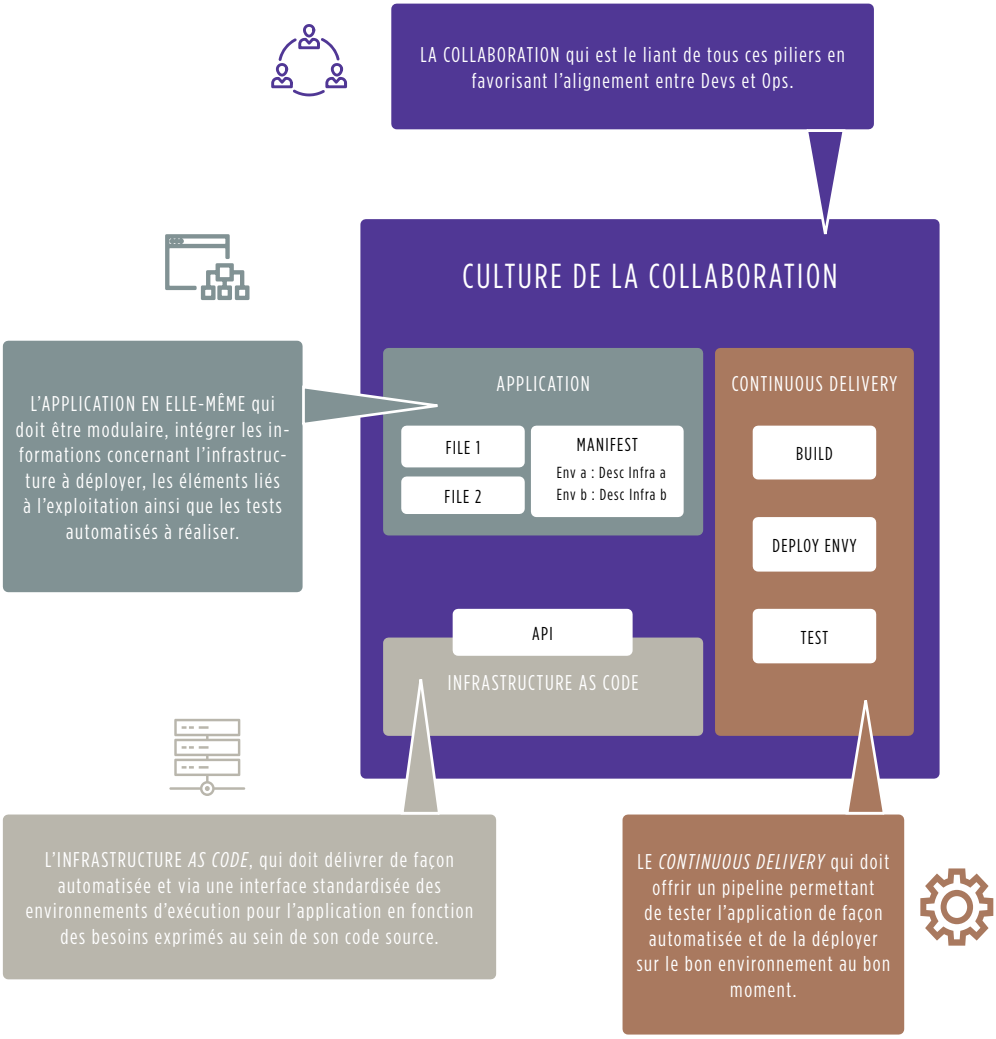
EN RAPPROCHANT LES OPS ET LES DEVS

Pour bien appréhender ce qu'est le DevOps, il faut comprendre ce sur quoi il repose :

/ Un brin d'automatisation

/ Et beaucoup de collaboration !

SI LA COLLABORATION SE DOIT D'ÊTRE OMNIPRÉSENTE, L'AUTOMATISATION PEUT SE DÉCLINER DANS TROIS COMPOSANTES



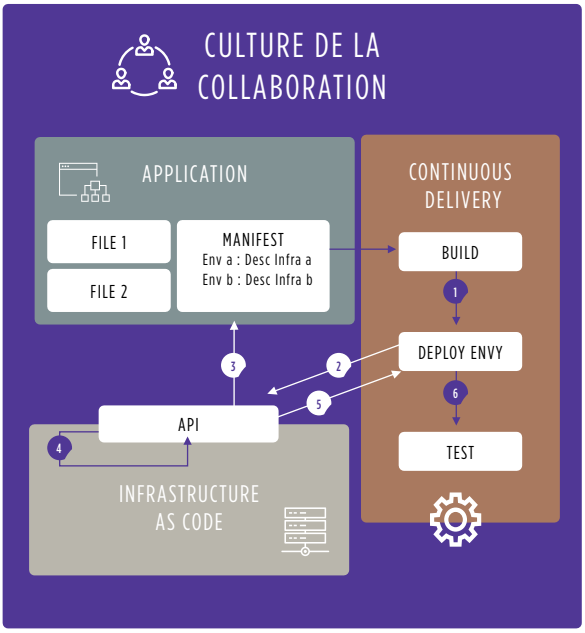
Ainsi, pour réussir sa transformation DevOps, il est nécessaire de travailler sur 4 axes en parallèle, qui constituent les 4 piliers du DevOps :

/ **L'Application** : préparer la transformation des applications en les rendant modulaires et automatisables

/ **Le *Continuous Delivery* :** automatiser la chaîne de livraison de la phase de développement à la mise en production

/ **L'Infrastructure *as Code*** : tendre vers une infrastructure service consommable, puis intégrable à la livraison applicative

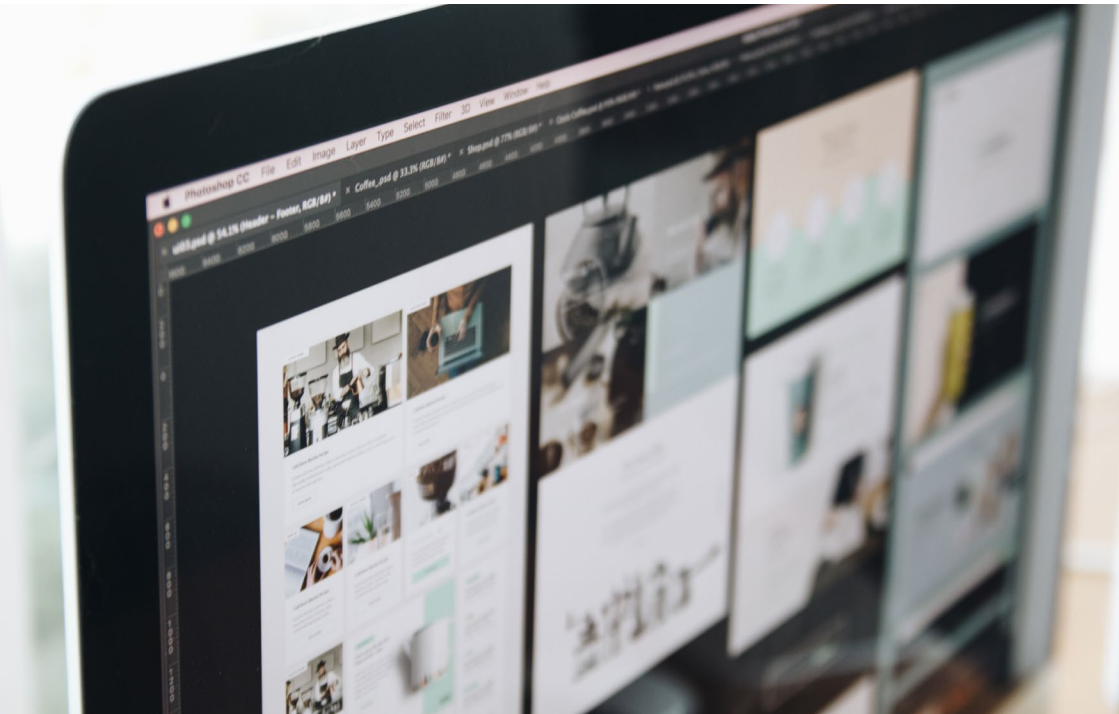
/ **La Collaboration** : changer la culture et les pratiques pour tendre vers une organisation sans silos et tirée par des méthodologies agiles



1. L'application une fois construite (compilée) doit être déployée sur un environnement (de test, d'intégration, de production, ...).
2. Le *pipeline* de *Continuous Delivery* fait appel à l'API de l'infrastructure pour demander la construction de l'environnement.
3. L'infrastructure utilise le fichier de configuration (Manifest) de l'application pour créer l'environnement adapté à l'application.
4. L'infrastructure orchestre dès lors la création effective de l'environnement.
5. Une fois l'environnement en question créée, l'application est déployée dessus.
6. L'application peut alors être testée comme indiquée à partir des tests indiqués dans le code applicatif.

PILIER I

DES APPLICATIONS MODULAIRES ET FAIBLEMENT COUPLÉES ENTRE ELLES



UNE ARCHITECTURE NÉCESSAIREMENT PLUS MODULAIRE

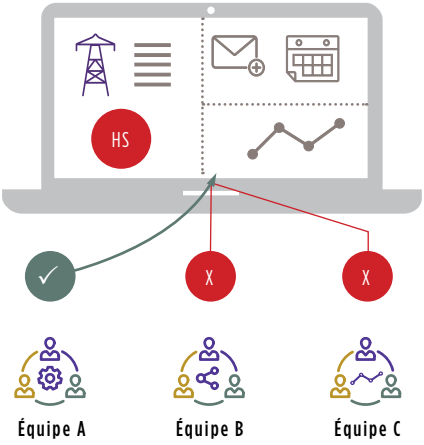
Être plus agile implique de livrer de nouvelles versions applicatives plus souvent en répartissant le travail sur plusieurs équipes (petites et pluridisciplinaires, des « *feature teams* »). **Cela nécessite dès lors de découpler leur travail pour plus d'efficacité et donc d'utiliser des architectures applicatives plus modulaires.**

Si les architectures orientées services ont été étouffées par une gouvernance centralisée et rigide, il s'agira ici de privilégier des approches décentralisées redonnant de l'autonomie aux développeurs dans leur publication et consommation.

Une application ainsi construite est un assemblage de modules (jusqu'à plusieurs dizaines), qui sont autant d'éléments de code indépendants, chacun sous la responsabilité d'une « *feature team* » : petite équipe autonome qui réalise le *think*, le *build* et le *run* du module.

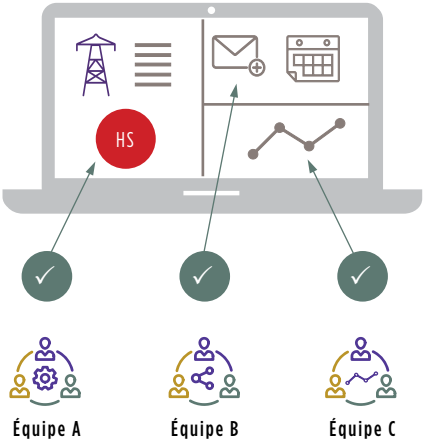
Les échanges entre modules sont standardisés et s'appuient sur des interfaces (API) permettant de les découpler. Ces APIs pourront utiliser des solutions de gestion (API management) afin de rendre les développeurs autonomes sur leur création, publication et consommation. Cela nécessitera toutefois de mettre en place **des solutions de cartographie des services pour éviter toute perte de contrôle du SI**.

ARCHITECTURE MONOLITHIQUE



Sur une architecture monolithique, une seule intervention est possible à un instant t, chaque livraison embarque le monolithe.

ARCHITECTURE MODULAIRE



Sur une architecture modulaire, plusieurs équipes peuvent travailler en même temps sur différents modules.

Au-delà de la tendance, cette approche est aussi un facilitateur important dans une démarche de développement Agile, à la fois pour améliorer la qualité du code, faciliter le travail en équipe, accélérer l'exécution des tests, etc.

LE CAS DES MICRO-SERVICES

Paradigme issu des pratiques des acteurs B2C à très large audience, l'architecture micro-service offre notamment une réponse à un besoin d'extrême scalabilité de fonctions précises d'un système. Dans une logique de décomposition d'un périmètre fonctionnel en services à fine maille autonomes mais collaborant entre eux, les pratiques d'architecture, d'infrastructure et d'exploitation doivent significativement évoluer pour gérer une constellation de micro-services instanciables à la volée et volatiles.

Dans une approche modulaire, les éléments constituants l'application doivent donc être très faiblement couplés, et ceci à tous les niveaux :

- **Vis-à-vis de *framework* ou de bibliothèques**, qui peuvent être utilisés comme des outils mais ne doivent pas définir la structure de la couche applicative.
- **Entre modules de l'application et vis-à-vis de l'extérieur pour simplifier la réalisation de tests**. Les modules de l'application doivent être testables de manière unitaire sans nécessiter d'interface utilisateur, de base de données ou d'autres applications. Ceci impose d'accorder une grande importance aux tests unitaires au sein du code de chaque module de l'application, ainsi qu'à la constitution de bouchons pour permettre de tester un service unitairement.
- **Vis-à-vis de l'Interface utilisateur**. Celle-ci doit pouvoir être remplacée sans aucun impact sur la couche applicative et ses fonctions.
- **Vis-à-vis de la persistance des données**. Pour pouvoir être scalable, ce type d'architecture requiert un fonctionnement *stateless* des services. La persistance des données et des sessions doit être gérée de façon à pouvoir assurer une montée ou une baisse de charge tout en garantissant la cohérence des données.
- **Vis-à-vis de toute autre fonction externe** à l'application issue d'autres applications ou services.

UNE MISE EN PRODUCTION RAPIDE ET MAÎTRISÉE

La conception d'applications en mode DevOps nécessite de renforcer les tests réalisés (notamment par leur automatisation) et de prendre plus de risques dans les mises en production. Cela signifie une évolution des méthodes de développement et l'introduction de nouvelles démarches comme l'approche « *Test Driven Development* », le « *Feature Flipping* » ou le « *Blue-Green Deployment* » pour mettre en production, partiellement ou de façon ciblée, certaines fonctionnalités.

De la même façon, cela induit de concevoir l'application pour qu'elle soit directement exploitable et ne pas attendre le moment de la mise en production pour se préoccuper de sa mise sous supervision, sous sauvegarde, de la conception de nouveaux scripts d'automatisation, des problématiques de performance, de scalabilité et autres problématiques d'exploitabilité.

TESTS UNITAIRES : L'APPROCHE *TEST-DRIVEN DEVELOPMENT*

LORSQUE LE DÉVELOPPEUR CODE UNE FONCTION, IL SUIT GÉNÉRALEMENT 3 ÉTAPES :

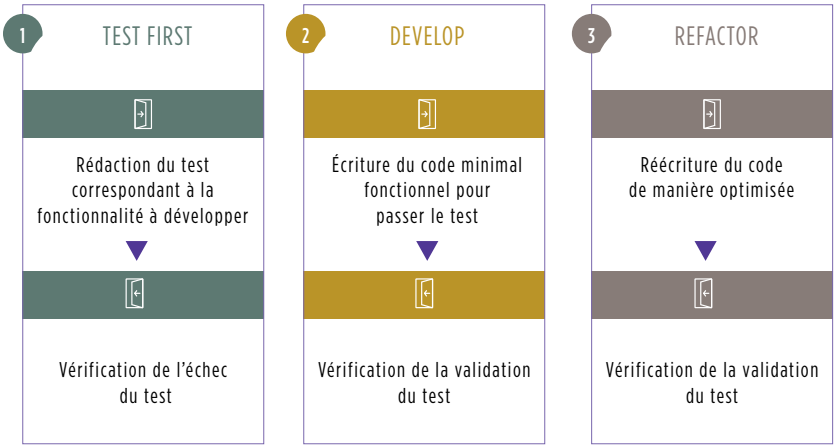
1. Écrire le code source de la fonction ;
2. Écrire le code source du test unitaire relatif à la fonction ;
3. Exécuter le test unitaire relatif à la fonction pour vérifier qu'il passe.

AVEC UNE APPROCHE TEST DRIVEN DEVELOPMENT, LE DÉVELOPPEUR FAIT L'INVERSE :

1. Écrire le code source du test unitaire relatif à la fonction à développer ;
2. Écrire le code source de la fonction permettant de passer le test ;
3. Retravailler le code pour l'améliorer tout en s'assurant que le test continue de passer.

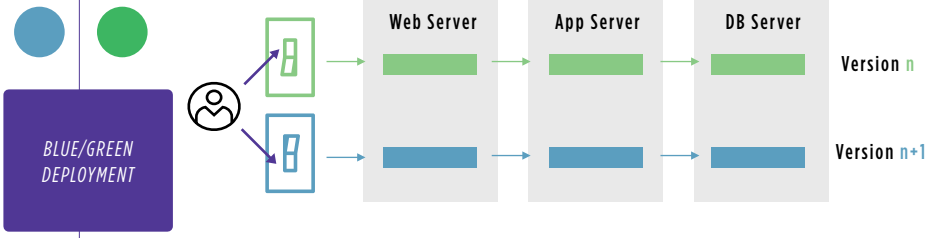
Cette méthode permet de s'assurer que chaque fonction est bien associée à un ou plusieurs test(s) unitaire(s) et facilite ainsi les tests de non-régression tout en précisant les spécifications puisque le test décrit le comportement attendu.

TEST-DRIVEN DEVELOPMENT

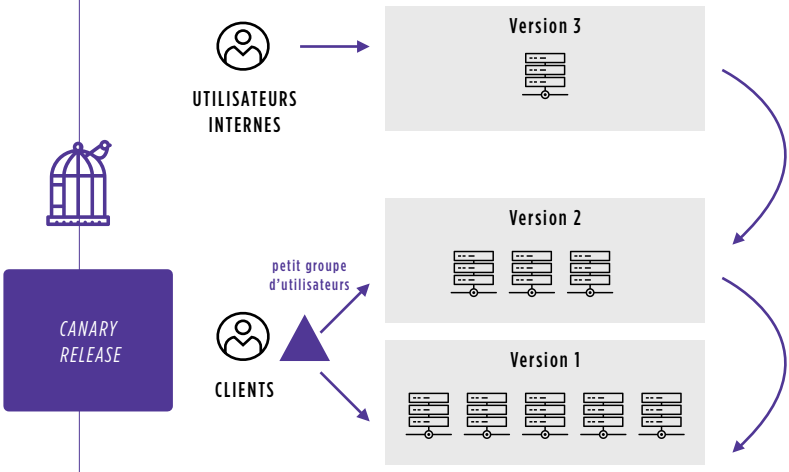


Répéter le cycle pour les nouvelles fonctionnalités

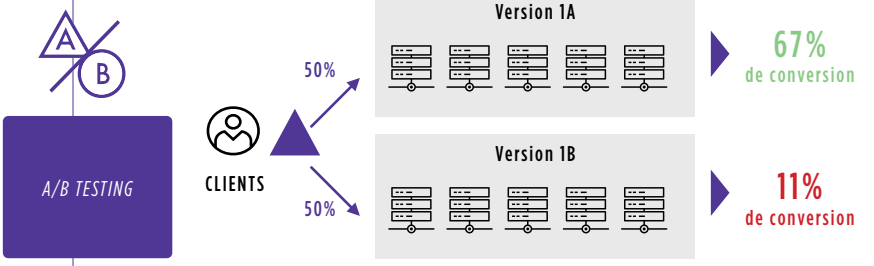
EXEMPLE DES MÉTHODES DE DÉPLOIEMENT



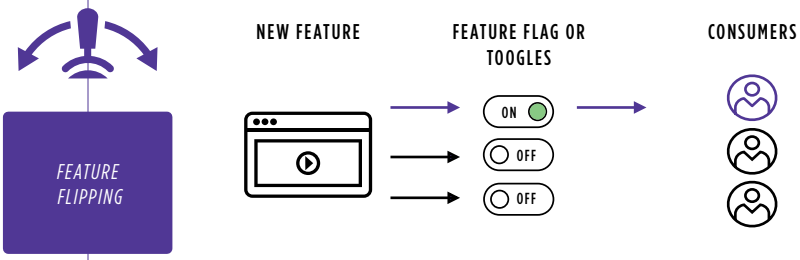
Déploiement d'une version n+1 sur un environnement parallèle à la version de production n et bascule facilitée de n vers n+1 et de n+1 vers n (*rollback*).



Déploiement de plusieurs versions en parallèle, certaines versions (voire fonctionnalités) ne sont ouvertes qu'à certaines populations (alpha puis beta testeurs) avant d'être (ou pas) généralisées.



Déploiement de 2 variantes d'une même version en parallèle pour en comparer les résultats et en déduire laquelle conserver.



Déploiement de fonctionnalités activables via une interface de l'application.
Permet notamment de ne pas activer certaines fonctionnalités si les tests ne passent pas sans pour autant freiner la mise en production. Peut être couplé à *canary release*.

Les tests fonctionnels (ou recettes) doivent eux aussi être largement automatisés pour pouvoir vérifier l'absence de régressions fonctionnelles à chaque itération et valider le bon comportement des fonctions nouvellement créées ou modifiées.

Cela n'est pas trivial et suppose d'être en capacité de tester les appels aux fonctions ainsi que les interactions homme-machine, de pouvoir récupérer et analyser les éléments (données ou éléments graphiques) retournés et de maintenir et faire évoluer l'ensemble des jeux de données nécessaires à l'exécution des tests.

La mise en place de cette automatisation, encore plus que pour les tests unitaires, peut représenter une charge non négligeable pour une application existante et peut entraîner un allongement des charges et délais à mettre sous contrôle.

LA DETTE TECHNIQUE N'EST PAS GÊNANTE SI ELLE EST SOUS CONTRÔLE

La construction plus rapide d'une application amène forcément à faire des compromis. On « contracte » ainsi rapidement et naturellement une dette technique (code, infrastructure, outils de gestion) que l'on « rembourse » tout au long de la vie du projet.

En revanche il n'est pas utile de chercher à atteindre le zéro dette, au-delà du risque de surqualité, c'est aussi un frein réel à l'agilité.

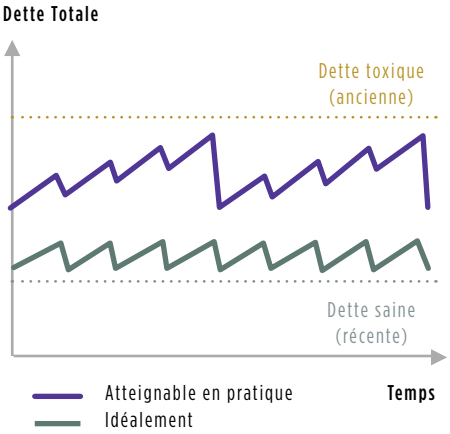
* Puppet & DORA 2016 State of DevOps Report

L'ANONYMISATION DES DONNÉES

Dans certains secteurs (banque et assurance notamment) le besoin d'anonymisation de la donnée devient de plus en plus prononcé. Dans ce contexte, la mise en place du DevOps doit inclure la capacité d'automatisation de l'anonymisation des données de production utilisées pour les tests fonctionnels.

Ceci représente un effort non négligeable à ce jour, le marché n'offrant pas encore d'outils matures.

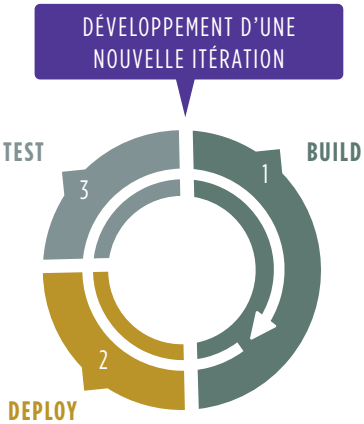
Pour autant, les études montrent qu'une approche DevOps permet de réduire jusqu'à 22%* le temps passé sur du travail non planifié (*bugs*) ou sur la reprise de code.



PILIER II

LE *CONTINUOUS DELIVERY* POUR DES LIVRAISONS AUTO-MAGIQUES

Le *Continuous Delivery* (CD) est une chaîne de construction logicielle automatisée. Il s'agit d'un ensemble de processus, outils et techniques permettant de gérer les livraisons applicatives, depuis la production du code à la livraison de la fonctionnalité en passant par le build, le déploiement, les tests, le packaging… L'objectif ? Augmenter la fréquence et la rapidité des livraisons de manière fiable, rapide et continue.



On considère habituellement que le *Continuous Delivery* s'appuie sur deux chaînes d'automatisation :

- **La chaîne de *Continuous Integration*** (CI), ciblée pour le développement intégrant les processus de *build*, de mesure de la dette technique (qualité du code), des tests unitaires et de *user acceptance* ;
- **La chaîne de *Continuous Deployment*,** étendant la chaîne CI par l'automatisation de la mise à disposition des environnements d'infrastructure et des livraisons applicatives. Cette extension est prise en charge par les exploitants en s'appuyant sur des méthodologies et outils quasi-identiques à ceux des développeurs. L'exploitant gère ainsi la consommation d'éléments d'infrastructure, de configuration management et le déploiement applicatif.

Jusqu'à la mise en production, l'ensemble de la chaîne est automatisée. L'Ops doit dans un premier temps valider un certain nombre de prérequis, comme l'adhérence de l'application avec le reste du SI, la disponibilité suffisante des ressources au sein des équipes pour réagir en cas d'incident, la disponibilité des infrastructures, l'opportunité du moment pour déployer une nouvelle version, etc.

Une fois ceci géré, ne lui reste plus qu'à déclencher le déploiement de l'application (déploiement « *push-button* »).

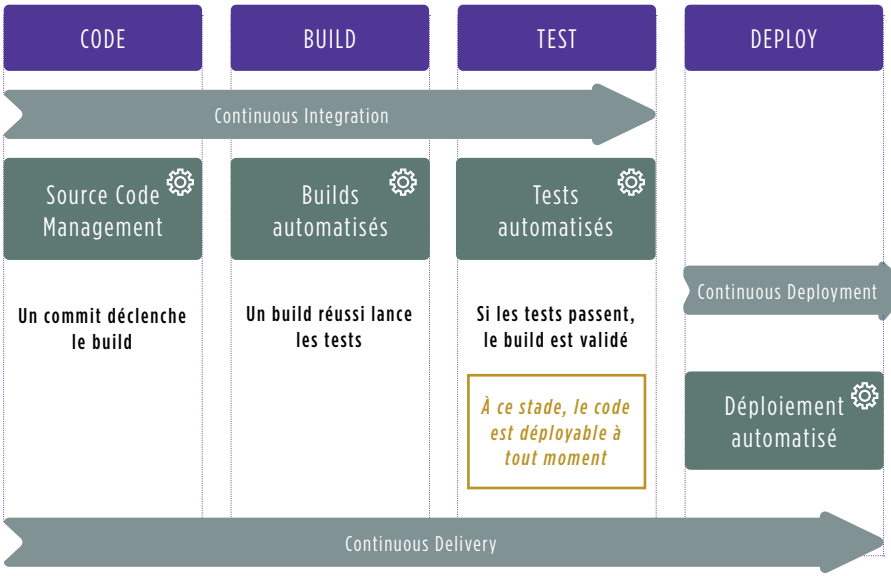
En revanche, le déploiement sur d'autres environnements (hors-production, pré-production) peut être complètement automatisé.

À partir d'un certain niveau de maturité, il est tout à fait envisageable d'imaginer un déploiement automatique de bout-en-bout (cf. schéma ci-contre).

CHOISISSEZ L'OUTILLAGE EN FONCTION DE VOTRE CONTEXTE, PAS DE SON ÉTIQUETTE

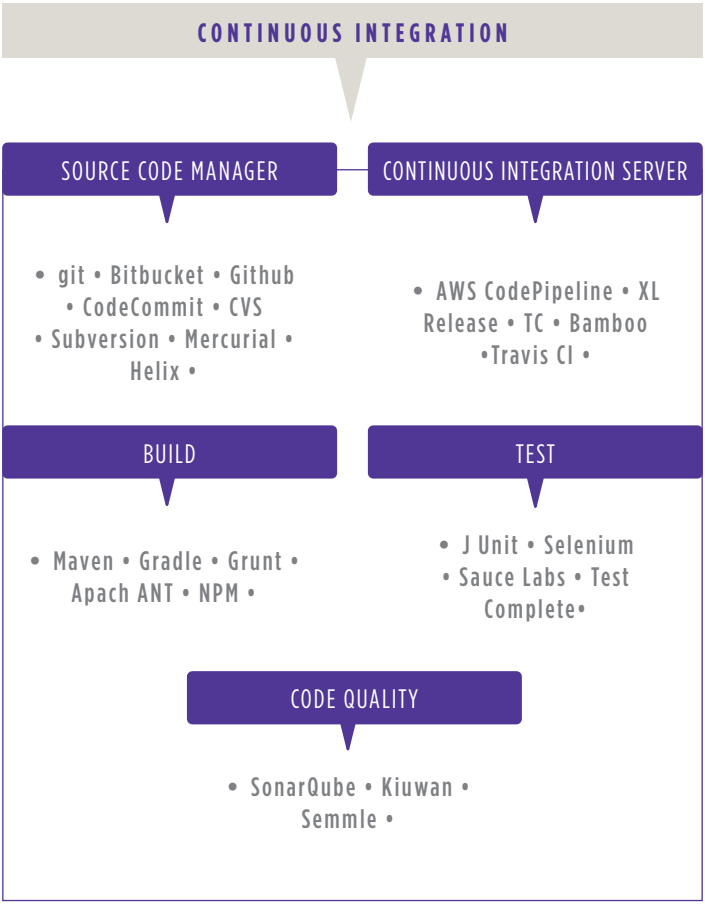
Aujourd'hui, l'outillage de la chaîne de *Continuous Delivery* est assez hétérogène : chaque fonction a son outil à choisir parmi une myriade de solutions, le marché de l'outillage DevOps étant en pleine effervescence.

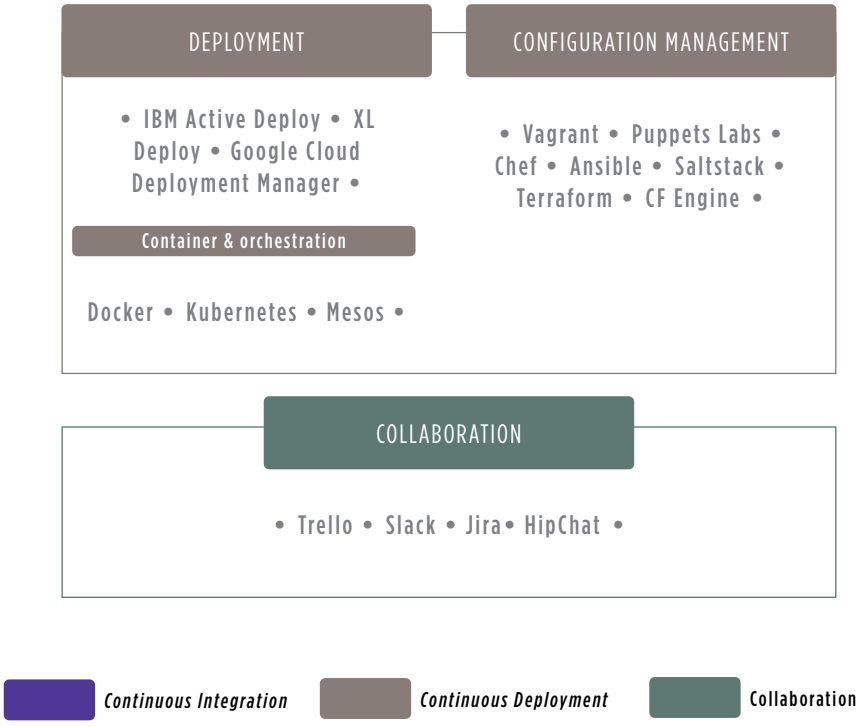
La première raison tient en l'absence de définition claire et partagée de DevOps ce qui permet à de nombreux éditeurs d'appliquer l'étiquette DevOps sur tout outil ayant trait à la conception logicielle, à l'usine logicielle, à la gestion de configuration ou toute autre forme d'orchestration (« *DevOps washing* »).


CODE
BUILD
TEST
DEPLOY
Continuous Integration
Source Code Management
Builds automatisés
Tests automatisés
Continuous Deployment
Un commit déclenche le build
Un build réussi lance les tests
Si les tests passent, le build est validé
À ce stade, le code est déployable à tout moment
Déploiement automatisé
Continuous Delivery

De plus, beaucoup d'outils essaient d'étendre leur scope au-delà de leur champs technico-fonctionnel traditionnel, ce qui contribue à complexifier les réels apports respectifs des différents outils sans pour autant offrir une vraie suite logicielle de bout-en-bout.

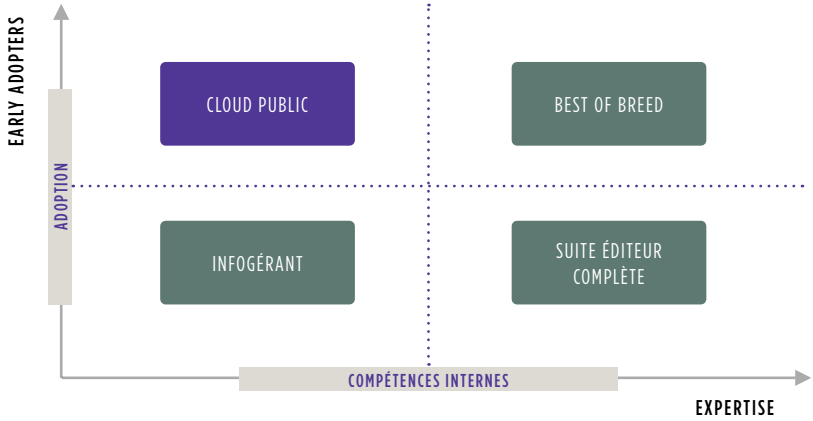
Pour s'y retrouver, nous proposons de classer les outillages selon la matrice suivante :





L'absence de consensus sur la mise en œuvre de DevOps empêche l'industrie logicielle de proposer aujourd'hui des solutions dédiées. Pour autant, ces solutions feront leur apparition dans les prochaines années tout en imposant leur mode de fonctionnement (et donc certaines pratiques).

Le choix du type d'outillage se fera en fonction du contexte de l'entreprise : selon si elle est de type « *early adopter* » elle préfèrera un modèle *Best of Breed* ou une solution se basant sur un *Cloud* public et son outillage associé ; si elle est de type « *follower* » elle pourra choisir la suite intégrée d'un fournisseur du marché ou la solution clé en main d'un infogérant.



Enfin, certains outils doivent naturellement être centralisés quand d'autres peuvent être instanciés par équipe pour que chacune puisse garder la main sur le rythme de mise à jour de l'outil en question ou pour en gérer certaines spécificités de configuration.

Généralement les outils centralisés et partagés par tous sont :

- Le gestionnaire de sources / artefacts (librairies, fichiers de configuration…) ;
- Les outils de gestion de la couverture des tests ;
- Les outils de publication des résultats d'analyse.

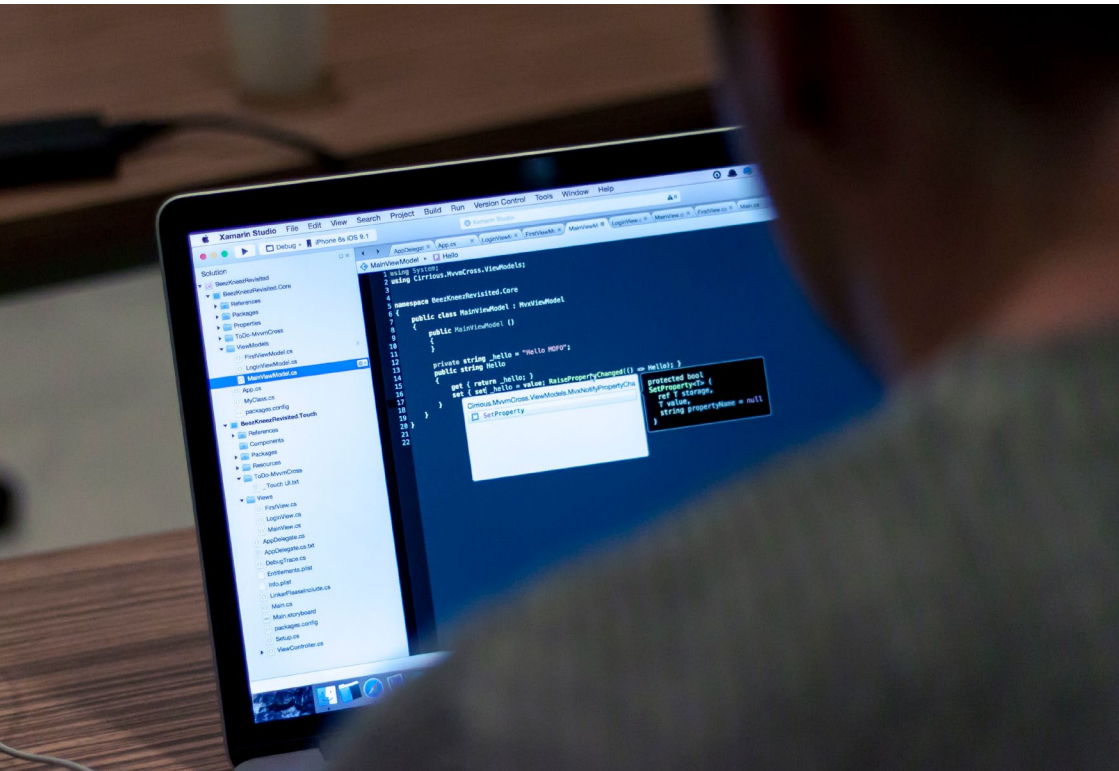
Tandis que les outils que l'on est amené à instancier par équipe peuvent être :

- Le *builder* d'applications ;
- Les outillages de tests unitaires, couverture de code, respect des normes de codage ;
- Les outils de déploiement (configuration management et déploiement applicatif).

PILIER III

L'*INFRASTRUCTURE AS CODE*, UNE INFRASTRUCTURE PILOTÉE PAR LE CODE APPLICATIF

L'*Infrastructure as Code* (IaC) est l'étape ultime d'agilité et de banalisation de l'infrastructure.



Depuis plusieurs années beaucoup de progrès ont été faits sur le marché pour tendre vers l'automatisation et la consommation de l'infrastructure : on parle « *as a service* », « *on demand* », catalogue de services, Cloud, API… L'*Infrastructure as Code* pousse plus loin encore cette idée.

Le but de l'IaC est de permettre la création et la configuration d'un environnement d'exécution complet (éléments d'infrastructure et *middleware*) automatiquement au travers du code applicatif. Les développeurs manipulent ainsi l'infrastructure dans le code même de l'application, au même titre qu'une fonctionnalité.

L'IaC permet donc de définir l'infrastructure d'une manière nouvelle : l'infrastructure n'est plus que simple puissance de calcul, scalable sans difficulté, manipulable par les Devs à travers des langages de codes applicatifs habituels.

COMMENCER À RENDRE L'INFRASTRUCTURE AGILE EN L'AUTOMATISANT

Bien entendu, l'automatisation de l'infrastructure reste un prérequis à l'IaC : pour manipuler de l'infrastructure au travers du code applicatif, il faut que l'infrastructure soit exposée à travers des interfaces programmatiques (APIs).

Cette automatisation peut se faire en 3 étapes, qui correspondent à 3 niveaux de maturité.

La mise en œuvre d'un IaC à l'état de l'art n'est pas un pré-requis au DevOps : vous pouvez démarrer vos premiers projets en ayant automatisé seulement une partie de votre infrastructure et tendre vers une *Infrastructure as Code* de bout-en-bout par itérations successives.

1. **La première étape de fourniture de l'enveloppe de la machine virtuelle est souvent la plus simple.** Dans une optique DevOps, il faudra cependant lever les derniers freins à une automatisation complète (souvent la configuration réseau) et être en mesure de provisionner en un clic cet environnement.

2. **La deuxième étape de déploiement des *middlewares* est déjà plus complexe.** Pour faire simple on pourra par exemple, créer des modèles de machines virtuelles avec des *middlewares* préinstallés mais cette méthode atteint vite ses limites (difficulté à gérer le cycle de vie du modèle, à ajouter un nouveau middleware, etc.). Il est donc préférable de s'outiller pour être en mesure de provisionner directement les machines et *middlewares* requis de façon automatisée éventuellement en ayant même défini des topologies applicatives (déploiement d'un ensemble d'éléments suivant un schéma défini au préalable).

3. Enfin **la troisième étape** est souvent la plus complexe puisqu'elle requiert **d'interagir avec d'autres éléments de l'infrastructure** (un *firewall*, un *load balancer,* une passerelle d'échanges, etc.) pour fournir à l'application **un environnement d'exécution complet**.

Ces trois étapes peuvent être réalisées séquentiellement ou en une fois selon le degré d'automatisation de l'infrastructure.

Il est également nécessaire de tendre vers une automatisation des services d'infrastructure : sauvegarde, supervision technique et applicative, ordonnancement, sécurité. La cible étant que la résilience, la sécurité, les services d'exploitation soient directement gérés dans l'application par les développeurs eux-mêmes, il est important que les développeurs puissent s'abonner à ces services directement au travers d'une API.

LE PAAS, UN ACCÉLÉRATEUR

Les outillages PaaS (*Platform as a Service*) sont de formidables accélérateurs pour peu que l'on relève le défi de leur intégration au SI et notamment à la chaîne d'exploitation.

On se méfiera ainsi des solutions de PaaS Privés (développées en interne ou solutions éditeurs déployées on-premises) souvent peu matures, qui n'intègrent pas simplement les problématiques des services d'infrastructures (sauvegarde, supervision, réseau & sécurité).

BANALISER L'INFRASTRUCTURE POUR RENDRE LES DÉVELOPPEURS AUTONOMES

L'automatisation de ce processus ne suffit pas à elle-seule à constituer une Infrastructure as Code, il est nécessaire de rendre accessible ce déploiement via une API et de pouvoir définir, dans le code source de l'application, l'infrastructure requise pour faire fonctionner l'application :

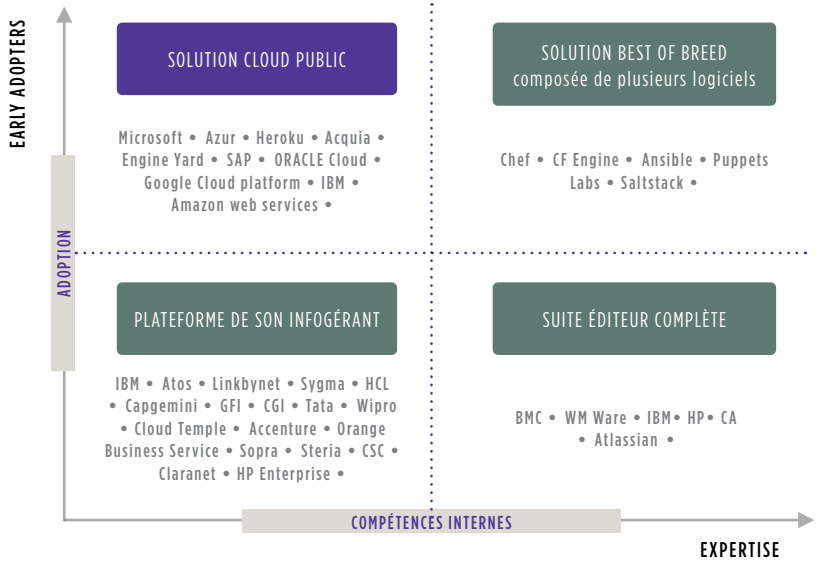
- Nombre de serveurs et *middlewares* associés ;
- Services applicatifs à déployer (cache mémoire, queue de messages, etc.) ;

Cela doit permettre de banaliser l'infrastructure pour ne plus offrir aux développeurs qu'une simple puissance de calcul.

UTILISER LE CLOUD PUBLIC EN CAS DE PROJET PARTANT D'UNE PAGE BLANCHE

L'outillage est avant tout une question de contexte. Les outils sont nombreux : de celui bien limité à une fonction particulière à la suite complète de grands éditeurs en passant par les solutions *pure-player Cloud*, il en existe pour tous les contextes.

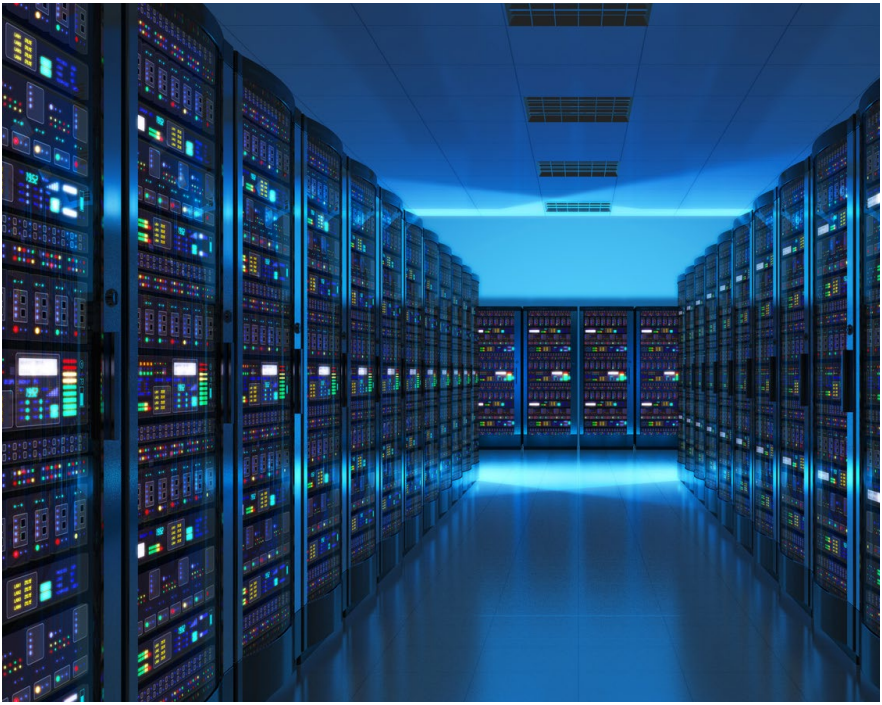
Ainsi une organisation avec de fortes compétences techniques en interne n'aura pas la même stratégie qu'une entreprise ayant l'habitude de s'appuyer sur des ressources externes. De la même façon une entreprise prête à prendre des risques pour se différencier (« e*arly adopter* ») ne réagira pas de la même façon qu'une entreprise qui cherche de la stabilité et des solutions matures pour assurer son fonctionnement.



L'approche *Cloud* public est quoiqu'il arrive une approche à considérer, tant les acteurs de ce type de *Cloud* sont en avance sur le marché. Amazon, Microsoft, Google atteignent un niveau de maturité qui ne saurait être rattrapé par des équipes internes d'entreprises dont ce n'est pas le cœur de métier. Il convient donc dans une stratégie *Cloud* privé de savoir pourquoi et pour quels *workloads* l'on adopte cette stratégie.

Enfin lorsque l'on conçoit une IaC, il est indispensable de standardiser l'infrastructure et donc faire des choix :

- Choisir où investir l'effort pour automatiser et industrialiser des composants qui font sens.
- Accepter de ne pas intégrer les *workloads* qui n'entreraient pas dans ce standard. Ce qui implique également que toute nouvelle application doit être conçue pour suivre ces standards.



PILIER IV

CASSER LES SILOS, DÉVELOPPER L'AGILE ET LE TRAVAIL COLLABORATIF

La culture DevOps puise ses sources dans les méthodes Agiles et le *Lean* (responsabiliser, faire confiance, respecter, communiquer avec transparence) ainsi que sur une approche « *Business-centric* », catalyseur d'une plus grande collaboration entre les Métiers, le Développement et la Production.

Ce changement culturel ne peut pas se faire du jour au lendemain, et il convient de l'échelonner sur 2 grands paliers :

Pour ce faire, il faut adopter un langage, des indicateurs et des outils communs et partagés :

/ Avoir une vision « produit » et non plus « projet » : les équipes réalisent un produit qui sera utilisé par un client final et ne contribuent plus seulement à un projet lambda.

/ Faire évoluer les indicateurs pour ne plus objectiver les développeurs sur la seule qualité de leur code ou fréquence de leur *release* mais aussi sur leur bonne compréhension de la production (et vice-versa pour les exploitants).

/ Tirer parti des outils collaboratifs (type Trello ou Jira agile) pour faciliter et renforcer le lien entre les Dev et les Ops

« IT IS ALL ABOUT PEOPLE » #HUMANFIRST

Une transformation DevOps est avant tout une question de personnes, d'humains, de collaboration.

Les premiers freins à lever seront donc des questions de collaboration, des personnes à convaincre que l'on peut faire mieux différemment, des process à casser, des objectifs (y compris ceux de fournisseurs) à revoir, des outils à changer, etc. Tout cela ne peut se faire qu'en s'appuyant sur des personnes fiables, motivées, très compétentes (Google parle de « *Highly Skilled Engineers* » dans sa méthodologie « *Site Reliability Engineering* » - SRE), avec de fortes capacités d'adaptation.

De plus comme dans tout projet de conduite du changement, il sera nécessaire de beaucoup communiquer, sur les réussites et les échecs, sur les objectifs, sur les impacts, l'organisation et les métiers.

LES ORGANISATIONS S'AGILISENT EN S'INSPIRANT DES PRATIQUES DES GÉANTS DU WEB

Les pratiques popularisées par Google (SRE), Spotify, Amazon, Facebook et consorts font aujourd'hui des émules. Ces pratiques sont le prolongement du chemin vers le « tout agile ». Elles ont fait émerger des principes et des méthodes plus ou moins complexes pour développer l'agilité à l'échelle de l'entreprise.

SPOTIFY

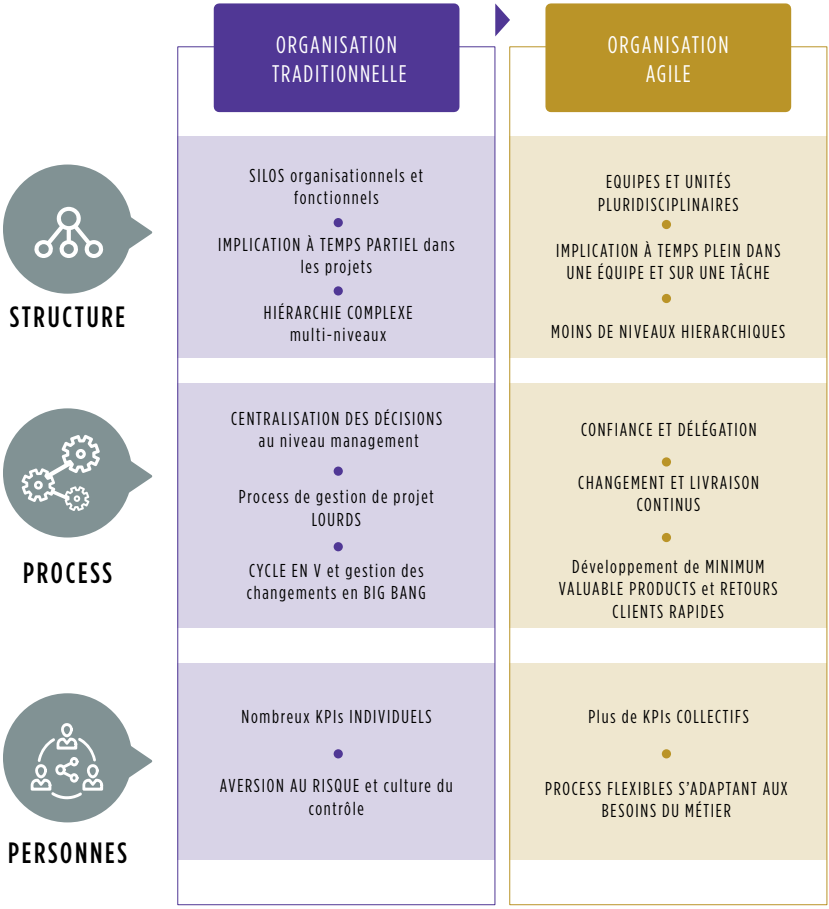
- Spotify est à l'initiative d'une méthodologie dont le pilier principal est l'autonomie donnée aux équipes (*feature teams*). Afin de maintenir la cohérence de son SI, Spotify a mis en place des communautés regroupant les sachants d'une thématique (*chapters, tribes, guilds*) qui forment une sorte de hiérarchie matricielle.
- Reconnue comme l'état de l'art des organisations Agiles, c'est aussi une méthodologie assez peu documentée et demandant un haut niveau de maturité tant en termes d'agilité qu'en termes de management des équipes.

SAFe Scale Agile Framework

- SAFe est un cadre de travail développé en 2011 par une équipe d'experts Agile pluridisciplinaires. Il a été conçu pour les organisations traditionnelles (par opposition aux *pure-players* web) pour adresser des programmes complexes impliquant de nombreuses équipes.
- Il s'adresse à des entreprises maîtrisant déjà la méthode *Agile Scrum*.

SCRUM OF SCRUMS

- Le *Scrum of Scrums* est une technique permettant de synchroniser très simplement plusieurs équipes *Scrums* entre elles sans mettre en place un *framework* très complexe.
- Si cette technique ne permet pas réellement de développer l'agilité à l'échelle de l'entreprise, elle peut être une étape pour la développer à l'échelle d'un projet d'envergure.


ORGANISATION TRADITIONNELLE
ORGANISATION AGILE
STRUCTURE
SILOS organisationnels et fonctionnels
IMPLICATION À TEMPS PARTIEL dans les projets
HIÉRARCHIE COMPLEXE multi-niveaux
EQUIPES ET UNITÉS PLURIDISCIPLINAIRES
IMPLICATION À TEMPS PLEIN DANS UNE ÉQUIPE ET SUR UNE TÂCHE
MOINS DE NIVEAUX HIERARCHIQUES
PROCESS
CENTRALISATION DES DÉCISIONS au niveau management
Process de gestion de projet LOURDS
CYCLE EN V et gestion des changements en BIG BANG
CONFIANCE ET DÉLÉGATION
CHANGEMENT ET LIVRAISON CONTINUS
Développement de MINIMUM VALUABLE PRODUCTS et RETOURS CLIENTS RAPIDES
PERSONNES
Nombreux KPIs INDIVIDUELS
AVERSION AU RISQUE et culture du contrôle
Plus de KPIs COLLECTIFS
PROCESS FLEXIBLES S'ADAPTANT AUX BESOINS DU MÉTIER

MOBILISER QUELQUES CHAMPIONS

POUR AMORCER LA TRANSITION VERS LE DEVOPS

IaC, CD, Culture… Par où commencer ? Comment initier une démarche DevOps ?

Le passage des méthodes classiques au DevOps représente une véritable rupture dans l'organisation du travail. Son déploiement est une entreprise progressive et délicate qui exige des investissements humains et techniques à ne pas sous-estimer.

Les premiers temps de cette transformation sont clés pour embarquer les acteurs, justifier des investissements futurs et créer une dynamique de transformation durable. La trajectoire adoptée peut et doit permettre de réaliser des retours sur investissement dès les premiers temps de la transformation.

Une trajectoire projet par projet permet ainsi d'échelonner l'effort sur le long terme et dégager rapidement des gains visibles et mesurables.

> « La trajectoire adoptée peut et doit permettre de réaliser des retours sur investissement dès les premiers temps de la transformation »

La stratégie de déploiement du DevOps peut se décliner selon les 3 axes suivants :

- Le choix du premier périmètre
- Le déploiement du socle[1]
- Le développement des compétences

CHOISIR UN PROJET VITRINE DE LA TRANSFORMATION, AMBITIEUX ET « SAFE TO FAIL »

La priorisation du portefeuille projet doit permettre d'embarquer les Métiers dès le démarrage et de pérenniser la dynamique projet après projet.

Pour cela, il est clé de choisir en priorité un périmètre qui concentre de fortes attentes des Métiers et pour lesquels les bénéfices du DevOps seront réellement significatifs et simples à mettre en œuvre.

Dans la démarche DevOps, l'approche itérative associée à une forte proximité des équipes, du Métier jusqu'à la production, permet de s'adapter rapidement aux évolutions en réduisant les freins liés aux opérations. Le DevOps garantit donc pleinement une réponse adaptée au besoin même si celui-ci se précise – voire change – au fur et à mesure du projet. Ce sont par ailleurs les projets ou produits dont le besoin évolue souvent qui

1- Est entendu par déploiement du socle : le déploiement d'outils collaboratifs, le déploiement de plateformes de développement/déploiement (Continuous Delivery), et l'automatisation (Infrastructure as Code, automatisation des tests...).

permettront de démontrer que le DevOps fait mieux, et plus vite, que les méthodes classiques, y compris sur des périmètres embarquant fortement les opérations.

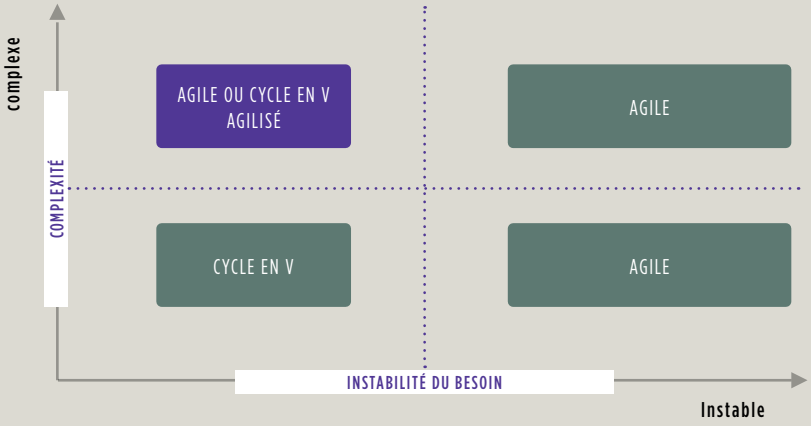
Pour autant le périmètre choisi doit être sécurisé (« *Safe to Fail* »). Le risque d'échec n'étant pas négligeable il est important d'avoir un plan B, ce qui facilitera d'autant plus l'acceptation du risque.

TOUS LES PROJETS PEUVENT TROUVER DES BÉNÉFICES AUX MÉTHODES AGILES, LE GRAAL ÉTANT POUR LES PROJETS DONT LE BESOIN ÉVOLUE RAPIDEMENT

Les projets dont le besoin est instable (le résultat final n'est pas prédictible), sont très naturellement éligibles aux méthodologies Agiles et DevOps puisque bénéficiant directement du caractère itératif de ces démarches.

Les projets simples et stables peuvent sans grand risque être traités par les méthodes classiques. À terme, ils seraient aussi bien couverts par des méthodologies Agiles.

Enfin les projets complexes mais dont le besoin est stable sont éligibles à l'un ou l'autre si l'on est sûr de la compréhension du besoin. Dans le cas contraire, mieux vaut privilégier des méthodologies Agiles.



COMMENCER PAR LES APPLICATIONS FACILEMENT AUTOMATISABLES

Le choix des premiers projets doit également prendre en compte les applications impliquées.

L'effort d'intégration d'une application à la chaîne du *continuous delivery* est rarement négligeable. Du côté des Devs, il faut opérer la transformation des processus d'intégration habituels et parfois le changement d'une partie de l'outillage. Du côté des Ops, tous les scripts d'automatisation des processus de livraison doivent être revus pour qu'ils soient gérés par les même outils que ceux utilisés par les Devs *(versionning, build, testing…)*. **Eviter les ruptures dans la chaîne de delivery est au cœur des principes du DevOps.**

Pour minimiser les investissements au démarrage, il est donc important de sélectionner les applications pour lesquelles l'effort d'intégration à la chaîne de *continuous delivery* est minimal.

Typiquement, des progiciels qui imposent une intervention quasi manuelle pour l'acquisition d'une clé de licence ou requièrent l'usage d'une interface graphique pour leur installation sont à mettre de côté dans les premiers temps de la transformation.

De même, certaines applications se prêtent par nature très mal à l'automatisation des tests.

Parmi les applications éligibles identifiées, il est intéressant de cibler en priorité celles dont le processus de livraison est déjà un frein identifié au regard des enjeux de *time-to-market* et de qualité de service, ainsi que celles qui ont des cycles d'évolution relativement courts. Les bénéfices de la refonte de telles applications seront d'autant plus évidents que le frein est important. Ils serviront à justifier les investissements engagés pour l'automatisation de la livraison applicative.

PRIORISER LA CONSTRUCTION DU SOCLE IAC & CD SELON VOS POINTS DE DOULEUR

Le déploiement du socle va demander des investissements importants sur le long terme, alors que paradoxalement il est peu visible du Métier. Pour autant, c'est sur lui que repose la pérennisation du succès de la démarche.

Pire encore, s'il est mal ou trop timidement réalisé, on peut à la fois accentuer les points de douleur des Ops et mettre en danger la réussite des projets, jusqu'à entraîner un rejet du DevOps et un retour aux méthodes classiques.

Les premiers projets seront ceux qui permettront de **construire un premier socle pour l'*Infrastructure as Code* et le *Continuous Delivery***. Le périmètre de ce socle sera élargi dans la durée, projet après projet.

Chaque projet DevOps doit être vu comme une occasion de dégager du budget pour faire évoluer les briques du socle – et chaque transformation du socle comme un moyen de réduire l'effort pour les projets suivants. Pour rester dans ce cercle vertueux, la priorisation est clé.

Une bonne pratique est de **se fixer, pour chaque projet, un objectif d'automatisation** qui sera traité au sein d'un sprint au même titre qu'une fonctionnalité applicative. Dans certains cas, si le projet en question est suffisamment long et complexe, il est même possible d'observer un retour sur investissement avant même qu'il se termine.

Comme pour l'automatisation des livraisons applicatives, il est particulièrement efficace de profiter des projets pour transformer en priorité les briques déjà identifiées comme des points de douleur par les équipes.

Le déploiement d'outils collaboratifs, que ce soit pour la gestion de projet ou du code applicatif, doit être réalisé en priorité. Ces outils sont clés pour améliorer les interactions entre les équipes et les aligner sur des objectifs communs.

Enfin, **certaines organisations**, face à la complexité de gestion d'un SI très ouvert et hétérogène (multipliant par exemple les socles IaC) **auront besoin de pousser l'automatisation jusqu'à la mise en place d'une couche d'orchestration de leurs différents IaC** (internes et externes s'ils s'appuient sur des socles infrastructures externes) et d'une plateforme d'orchestration des chaînes de *delivery*.

Un *Meta-Orchestrator* (ou BPM) peut également être nécessaire pour gérer les processus de déploiement sur un ensemble d'applications en garantissant la cohérence Métier. Le déploiement d'un *Meta-Orchestrator* est le plus haut niveau d'automatisation des livraisons - il permet d'automatiser la prise en compte des dépendances de livraisons entre les différentes applications d'un même périmètre Métier.

LE CLOUD PUBLIC
UN ACCÉLÉRATEUR DE LA TRANSFORMATION DEVOPS

La création d'un socle à partir d'un existant peut être longue et coûteuse.

Le Cloud Public peut dés lors être un accélérateur fort à moindre coût pour démarrer une expérimentation DevOps en utilisant une suite déjà intégrée.

Tous les grands Cloud Publics (Microsoft Azure, Amazon Web Services, Google App Engine) proposent leur propre suite composée d'IaC et d'outils permettant de constituer une base de *Continuous Delivery* en la complétant avec quelques outils libres (pour le testing et la qualité du code notamment).

Nous recommandons fortement de démarrer ainsi pour bénéficier plus rapidement de résultats concrets et d'une transformation plus ambitieuse qui ne préjuge pas de l'existant et des processus installés.

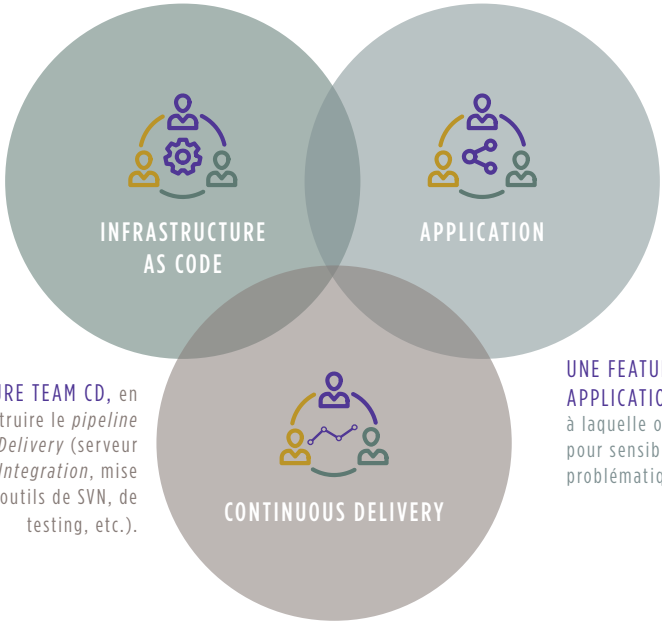
DÉMARRER AVEC 3 « *FEATURE TEAMS* » DE CHAMPIONS POUR CONSTRUIRE LE SOCLE ET LA PREMIÈRE APPLICATION EN MODE DEVOPS

Afin d'assurer le succès du premier projet et la bonne diffusion des pratiques, trois équipes doivent être réunies – et leurs membres choisis parmi les champions de leurs domaines. (cf schéma ci-dessous)

La *feature team* IaC et la *feature team* CD vont, en parallèle du premier projet, mettre en place les premiers éléments de socle nécessaires au lancement de ces projets en mode DevOps. Elles seront dimensionnées en fonction de la vitesse à laquelle on souhaite faire aboutir ces premiers éléments, et bien sûr, des moyens engagés.

Au démarrage, il est conseillé de regrouper ces équipes dans un même lieu, ou a minima de les lier par des outils collaboratifs efficaces. Dans la même logique, il est conseillé de leur dispenser des formations communes afin de créer des liens dès le démarrage.

UNE FEATURE TEAM IAC, en charge de construire le socle de l'*Infrastructure as Code* (orchestration, API, templates de déploiement).



UNE FEATURE TEAM CD, en charge de construire le *pipeline* de *Continuous Delivery* (serveur de *Continuous Integration*, mise en place des outils de SVN, de testing, etc.).

UNE FEATURE TEAM APPLICATION (ou équipe Produit) à laquelle on adjoindra un Ops pour sensibiliser les Devs aux problématiques des Ops.

L'ORGANISATION SERA CONTRE VOUS : PROUVEZ-LEUR QU'ILS ONT TORT !

La transformation vers un modèle agile va nécessairement se confronter à une résistance au changement. **Le premier projet doit absolument communiquer largement sur ses réussites.**

Il est aussi indispensable de mettre en place, dès le premier projet, les indicateurs chiffrés permettant de factualiser la réussite du DevOps.

Les outils mis en œuvre dans la démarche permettent d'analyser finement la qualité du code, les temps de déploiement, le nombre de livraisons applicatives, le nombre de bugs, etc. Ces éléments sont autant de points de mesure qui permettent de prouver l'intérêt de la démarche et son retour sur investissement.

Il s'agira de bien mettre en valeur ces gains, en faire la promotion pour justifier les investissements réalisés et futurs.

LES ERREURS À ÉVITER

Les premières transformations conséquentes permettent de ressortir quelques écueils à éviter :

- LE SYSTÈME SERA CONTRE VOUS : il est donc nécessaire d'en faire ressortir le ROI.
- PROUVEZ QUE VOUS AVEZ RAISON : dans cette optique il faut donc mesurer les bénéfices de la transformation.
- VOUS ALLEZ ÉCHOUER : il est indispensable d'accepter l'échec et l'erreur dans une démarche itérative « *test & learn* ».
- « *ONE SIZE DOES NOT FIT ALL* » : chaque démarche DevOps doit être adaptée au contexte et aux enjeux de l'entreprise.
- RESPECTEZ LE MONOLITHE : il est indispensable de s'intégrer avec les systèmes historiques / legacy et de respecter un certains nombre de processus existants. Ne pas chercher à tout casser.

FACTEURS CLÉS DE SUCCÈS

De la même façon l'on peut ressortir quelques clés du succès :

- SIMPLIFIEZ : les processus, les infrastructures, etc. Tout doit être standard.
- AUTOMATISEZ : vos infrastructures, vos processus et toutes les actions répétitives où à faible valeur ajoutée.
- RECRUTEZ DES INGÉNIEURS ET DES DÉVELOPPEURS DE HAUT NIVEAU : pour en faire des Ops qualifiés en charge de la conception des nouveaux services et briques du SI.

VERS UN MODÈLE OPÉRATIONNEL POUR LA DSI

Une fois que les premiers projets auront démontré l'intérêt de la démarche, il s'agira alors d'étendre ces bonnes pratiques à l'échelle de l'entreprise.

Les membres des premières équipes doivent servir de coachs, d'ambassadeurs, pour diffuser les pratiques au sein des équipes produits qui se formeront au fur et à mesure au sein des DSI.

Les équipes ayant contribué à la construction du socle IaC et CD doivent constituer les premières briques des centres de compétences qui feront évoluer ce socle en fonction des besoins des équipes produits et qui appuieront les Ops.

Afin d'assurer une certaine « pollinisation » (diffusion du savoir entre les équipes), il est important d'aider à la construction de communautés pour partager les bonnes pratiques entre Ops, Dev, mais aussi Architectes, etc.

Ces communautés doivent également permettre d'enrichir un cadre de référence commun sur les méthodologies agiles (projet, management, animations de communautés, etc.).

Enfin cette transformation devra respecter le legacy et les personnes qui continueront de le maintenir. Il ne faut donc pas seulement prévoir de gérer les aspects techniques (interopérabilité, évolutivité, maintenance, etc.) mais également les aspects humains (attractivité des postes, gestion des compétences, etc.) pour éviter de mettre en risque

LES ÉQUIPES PRODUITS VONT SE DÉVELOPPER POUR CRÉER DE MULTIPLES ÉQUIPES PAR FONCTIONNALITÉ AU SEIN DES DSI :

Au plus proche des Métiers pour s'adapter à leurs besoins

Intégrant des Devs et des Ops et utilisant l'IaC et le CD pour livrer plus rapidement et plus fréquemment de nouvelles fonctionnalités

LA CRÉATION D'UNE COUCHE D'INTÉGRATION ET DE GESTION DES SERVICES

S'appuyant sur des centres de compétences IaC et CD constitués des équipes ayant contribué sur les premiers projets DevOps

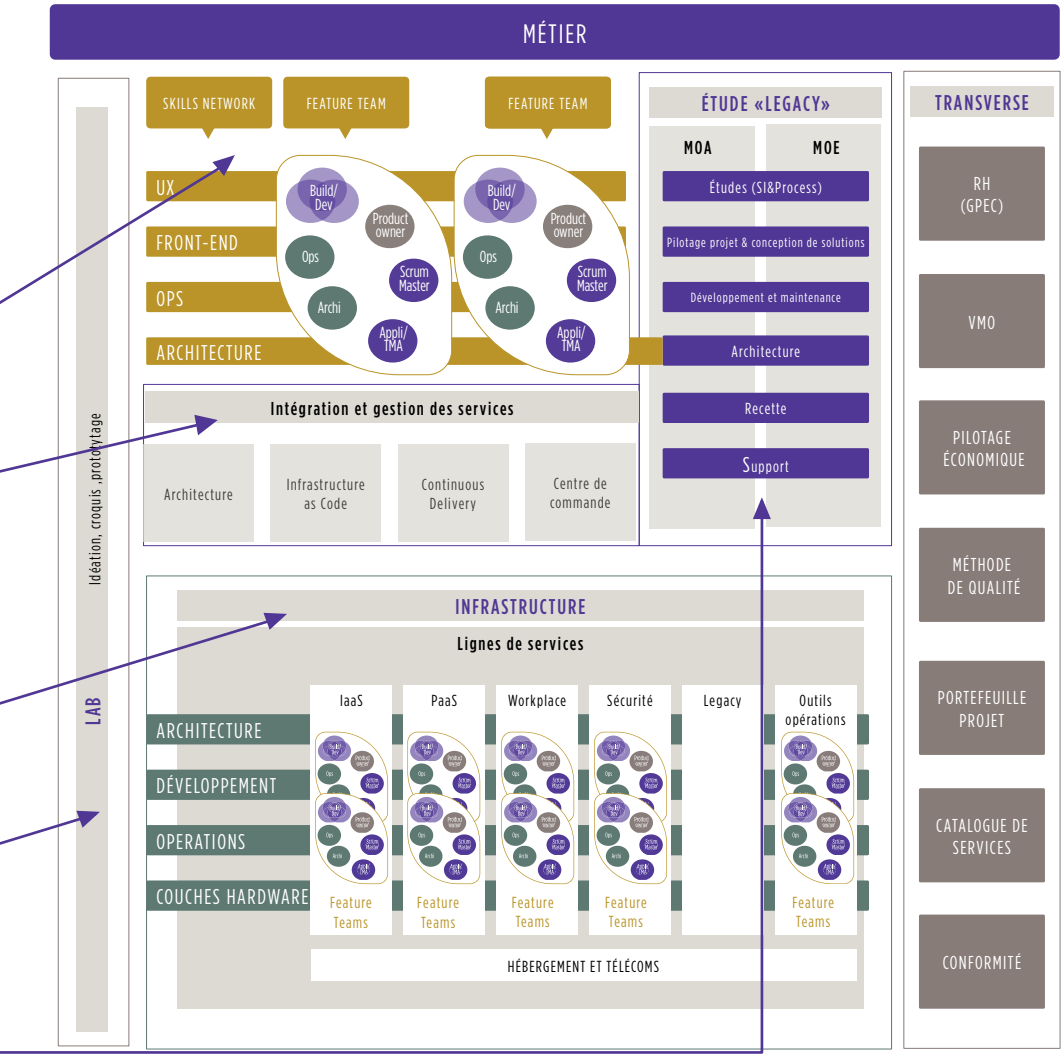
En charge également de la cohérence d'ensemble (architecture, sécurité, outillage…)

DES LIGNES DE SERVICES D'INFRASTRUCTURES SIMPLES, modulaires, « *secured by design* », et des opérations automatisées

Le développement de CAPACITÉS D'INNOVATION ET DE R&D pour garder une longueur d'avance et un lien étroit avec les Métiers

Mais aussi une COHABITATION DURABLE avec les modèles opérationnels traditionnels

cette partie. Dans un second temps, il sera nécessaire de le faire évoluer également, soit en le « rafraichîssant » (avec de nouvelles technologies), soit en le transformant complètement (réécriture morceau par morceau dans une philosophie agile).



ACCOMPAGNER VOS DEVS & OPS DANS LEUR CHANGEMENT DE MÉTIER

La nouvelle organisation de travail a nécessairement un impact sur les métiers des exploitants et des développeurs. Il ne s'agit pas de transformer les Ops en Devs, ou l'inverse, mais bien de faire évoluer les deux métiers vers de nouvelles pratiques.

Ainsi l'Ops va évoluer vers **la préparation d'éléments d'infrastructure réutilisables et automatisés.** Tout en continuant, le temps de la montée en compétence des Devs d'**assurer le développement des scripts de déploiement de chaque application et ce en utilisant les mêmes outils que les Devs**.

Les compétences de développement et la connaissance des outils de configuration management deviennent ainsi particulièrement importantes. Les Ops ont déjà une culture de scripting. L'enjeu pour eux sera de passer des pratiques « à l'ancienne » - scripts peu réutilisables, peu commentés et difficilement maintenables - à des pratiques et outils uniformes sur toute la chaîne de valeur IT.

En contrepartie, le renforcement de l'automatisation et de la qualité des déploiements va entraîner une diminution du nombre d'incidents et de demandes de changement gérés par les Ops jusqu'ici.

De leur côté, les Devs ne peuvent plus se contenter de produire le code de l'application. Ils vont intégrer les contraintes de déploiement des environnements, concevoir les modèles de déploiement applicatif et permettre la variabilisation de la matière applicative pour la rendre déployable quel que soit l'environnement visé, sans surcoût de gestion. Ils vont désormais s'intégrer dans une chaîne complète de *continuous delivery*, intégrer (à la différence des habitudes prises jusqu'ici) les processus de *build,* la mesure de la dette technique (qualité du code), ainsi que les tests unitaires et de *user acceptance* automatisés.

Cela induit pour eux de :

- **mieux comprendre la production et les modèles de déploiement applicatifs afin de « variabiliser l'application » pour permettre son déploiement différencié en fonction de l'environnement visé ;**
- **s'inscrire dans une démarche dite de « *test driven development* » pour assurer un haut niveau de qualité et de contrôle de non-régression.**

CONCLUSION

S'il est aujourd'hui impératif pour les entreprises d'être plus Agile, de déployer plus vite, les pratiques DevOps à mettre en œuvre n'en ont pas moins des impacts profonds sur les DSI :

- **Transformation de la façon de concevoir une application** en la rendant plus modulaire et en intégrant les infrastructures et les opérations dans son code ;
- **Automatisation et « softwarisation » des infrastructures**, ainsi que la mise à disposition des services d'infrastructures via des interfaces programmables directement dans le code ;
- **Évolution des métiers d'Ops vers le développement** et vice-versa (et donc nécessité de recruter différemment, de former, d'accompagner au changement).
- **Modification des façons de travailler avec les métiers**, via des « *feature teams* » intégrant toutes les parties prenantes et **modifiant également l'organisation et le modèle opérationnel de la DSI.**

Il faut donc démarrer dès maintenant, adopter ces pratiques progressivement, en *test & learn*, en s'inspirant des pratiques des *pure-players* du web. Cela requiert un changement d'état d'esprit : il faut être orienté résultat, penser produit et non plus projet, faire des pas plus petits mais en plus grand nombre. C'est un changement profond qui ne se fera pas en quelques mois, mais qui donnera des résultats très rapidement.

Et il faut le faire d'autant plus vite que la cible est loin : le NoOps. Ce concept selon lequel ce sont les développeurs eux-mêmes qui assurent l'exploitation de l'application quand les opérations se concentrent sur l'automatisation (et la supervision de bout-en-bout). Cela peut paraître une utopie mais c'est déjà une réalité pour des géants comme Amazon qui en tire de vrais bénéfices en recentrant les personnes sur la valeur ajoutée qu'elles peuvent apporter.

WAVESTONE

www.wavestone.com