COMPRENDRE LE DEVOPS POUR LE METTRE EN ŒUVRE

MEHDI EL KOUHEN

LIVRE BLANC - JUILLET 2022

ippon

SOMMAIRE

MEHDI EL KOUHEN

J'occupe depuis trois ans le poste d'architecte DevOps & Cloud chez Ippon Technologies à Nantes.
Mon rôle consiste à intervenir sur la mise en œuvre d'architectures microservices, en m'intéressant tout particulièrement aux aspects DevOps et Cloud.

Je suis également Senseï du programme BlackBelt, qui permet le développement de compétences en interne. Je mentore celles et ceux qui souhaitent améliorer leur expertise Cloud & DevOps.

N'hésitez pas à me contacter sur LinkedIn pour échanger, et à visiter le blog Ippon pour y retrouver quelques-uns de mes articles !

PRÉAMBULE

Le DevOps est un mouvement très en vogue; les attentes des entreprises sont très variées : de l'automatisation des déploiements jusqu'au déploiement continu en s'appuyant sur une collaboration accrue entre les équipes produit et les exploitants.

J'ai souhaité faire un tour d'horizon des impacts de cette démarche sur les processus (déploiement, supervision...) et outillages associés.

Ensuite, j'ai souhaité montrer que le DevOps apporte des changements organisationnels importants, comme la suppression des silos entre équipes pour aligner les équipes sur l'apport de valeur aux clients.

Pour terminer, j'ai souhaité apporter quelques clés sur la mise en place d'une démarche DevOps : je vous partage des bonnes pratiques à suivre ainsi que des points de vigilance.

INTRODUCTION

Le DevOps est une évolution du mouvement agile qui intègre les exploitants dans le cycle de développement des applications afin d'aligner toutes les équipes sur l'apport de valeur aux clients et donc d'améliorer la satisfaction client.

Ce mouvement repose sur les mêmes principes que l'agilité : prioriser la satisfaction du client, collaborer et partager les responsabilités, accepter le changement, favoriser l'excellence technique...

Voici quelques déclinaisons de ces principes d'un point de vue DevOps.

Concernant la priorisation de la satisfaction du client, l'automatisation des tâches permet de livrer fréquemment de la valeur aux clients. La CICD, qui est le point central de l'automatisation, fluidifie toutes les étapes du développement jusqu'au déploiement des applications et de l'infrastructure.

La collaboration entre équipe produit et exploitants permet une compréhension des enjeux de chacun. Elle donne aussi l'opportunité de traiter collectivement des points connexes à ces métiers, comme la mise en place d'une plateforme d'observabilité, le contrôle des coûts…
L'automatisation est un élément fondamental pour accepter le changement. Les différents types de tests automatisés (E2E, performance, chaos…) permettent de s'assurer du bon fonctionnement des applications. La mise en place d'une plateforme d'observabilité simplifie le diagnostic en cas d'anomalie et accélère ainsi leur correction.

Favoriser l'excellence technologique permet de réduire la dette technique, de remettre en question d'anciens choix et de les corriger suite à une meilleure compréhension des besoins et des aspects techniques (services utilisés, modèles d'architecture…).

La mise en œuvre de ces principes crée un cercle vertueux. La collaboration entre les équipes produits et exploitants génère de nouveaux domaines d'expertise (à cheval entre ces deux domaines) qui sont essentiels pour l'amélioration continue.

Dans la partie **Contexte historique**, nous décrirons l'émergence du mouvement DevOps, puis nous en présenterons successivement ses **Valeurs**, ses **Processus** et ses **Outils**.

Dans la partie **Organisation des équipes**, nous présenterons différentes organisations qui encouragent une collaboration entre les équipes de développement et les exploitants.

Dans la partie **Trois voies**, nous aborderons ensuite la métaphore des trois voies qui raconte sous forme d'histoires des stratégies de mise en place du DevOps.

Dans la partie **Mettre en place une démarche DevOps**, nous évoquerons des points importants d'une transformation DevOps.

Puis dans la **conclusion**, nous récapitulerons les bénéfices apportés par la mise en place d'une démarche DevOps.

CONTEXTE HISTORIQUE

01

“ POUR COMPRENDRE L’ÉMERGENCE DU MOUVEMENT DEVOPS, NOUS LE RESTITUONS DANS SON CONTEXTE HISTORIQUE.

La méthodologie projet dite “en cascade” (waterfall) apparaît dans les années 1970 (cf. article Royce, 1970). L’idée est qu’un projet est une succession linéaire d’étapes : de la spécification des besoins à l’exploitation des applications en passant par le développement.

Dans les projets en cascade, les différentes équipes ont une relation contractuelle : chaque équipe livre un contrat à l’équipe en aval qui doit le respecter. En effet, l’équipe métier livre un document de spécifications fonctionnelles à l’équipe de développement qui code l’application puis rédige et fournit un dossier d’exploitation aux exploitants.

Mais, pour de nombreuses raisons, cette méthodologie ne fonctionne pas :

— Les relations contractuelles entre les équipes sont conflictuelles : les exigences définies par une équipe en amont ne sont pas tenables par l’équipe en aval.
— Les applications construites ne répondent pas bien aux besoins des utilisateurs (pas de prise en compte du feedback des utilisateurs).

Les méthodologies projet ont évolué, depuis, pour apporter plus de flexibilité dans la planification et aboutir aux méthodes agiles qui sont orientées satisfaction du client plutôt que suivi d’un plan défini. Cette méthodologie a évolué pour faire naître le mouvement DevOps qui prône une forte collaboration entre les équipes produit et les exploitants. Ceci représente la suppression d’une première frontière entre les développeurs et les exploitants.

En parallèle de ces évolutions sur les méthodologies de projet, de nouveaux fournisseurs de services, de plateformes et d’infrastructures arrivent sur le marché : c’est le début du Cloud.

Le succès du Cloud repose en partie sur la mise à disposition de services d'Infrastructure (VMs, bus de messages…) par l'intermédiaire d'applications et d'APIs Web. Les développeurs utilisent rapidement ces APIs pour automatiser le déploiement des ressources d'Infrastructure.

Avant l'arrivée des services Cloud, la gestion des ressources était statique. En début de projet, les projets devaient évaluer leurs besoins en ressources (VM, disque…), les acheter et les installer. La durée entre l'évaluation des besoins et la mise à disposition des ressources pouvait être de l'ordre de plusieurs mois.

Avec l'arrivée des services Cloud, les projets gèrent les ressources dynamiquement. Ainsi, les utilisateurs des services Cloud peuvent déployer des ressources à la demande (en quelques minutes) et les restituer aussi vite en n'étant facturé que sur le temps d'utilisation. Cette gestion dynamique permet de créer des environnements pour des besoins particuliers (démo, test…) ou encore de redimensionner l'infrastructure (scaling horizontal) en fonction de la charge.

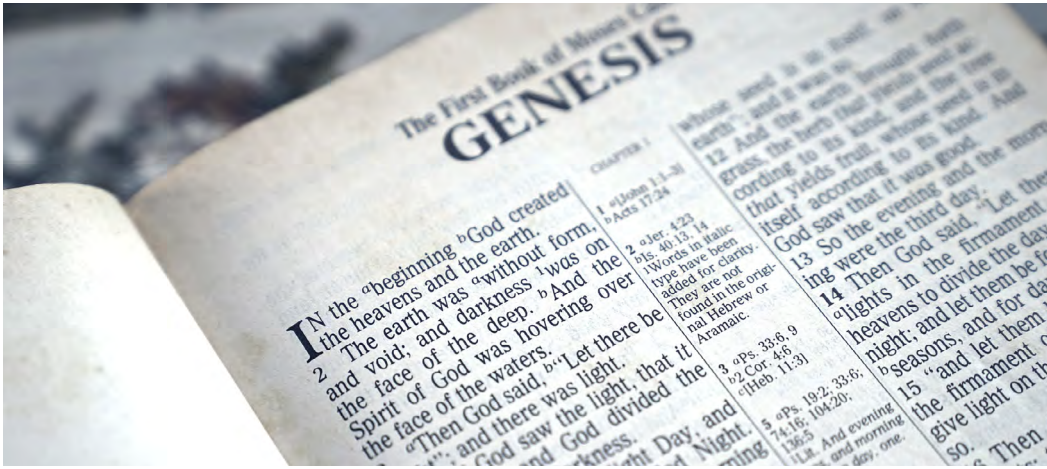
C'est ainsi que l'arrivée du Cloud participe à son tour à la suppression d'une autre frontière entre les développeurs et les exploitants.

01 ORIGINES DU DEVOPS

L'ORIGINE DU MOUVEMENT DEVOPS EST ATTRIBUÉE À PATRICK DEBOIS QUI PARTICIPE EN 2007 À UN PROJET DE MIGRATION D'UN DATA CENTER EN BELGIQUE.

Dans le cadre de ce projet, il collabore avec des développeurs (organisés en agile) et des exploitants. Frustré de la différence d'organisation entre ces deux équipes, il a l'intuition de généraliser les pratiques agiles aux métiers d'exploitation.

Patrick Debois crée, par ailleurs, en 2009, un **groupe de discussion** sur la pratique de l'agilité pour l'administration système. Certaines discussions abordent des sujets très actuels comme les stratégies de test d'infrastructure. En 2009, les solutions de virtualisation de PC x86 existent depuis une dizaine d'années et commencent à permettre ce genre d'usage.



02 ÉVOLUTIONS DU DEVOPS

Depuis son émergence, le mouvement DevOps a inspiré d'autres mouvements similaires :

- Le Site Reliability Engineer (SRE), pour améliorer la fiabilité des sites,
- Le DevSecOps, pour les aspects de sécurité,
- Le FinOps, pour les aspects financiers,
- Le AIops, pour l'Intelligence Artificielle,
- Le DataOps, pour la gestion des données,
- ....

Nous présentons succinctement ci-dessous les mouvements SRE, DevSecOps et FinOps.

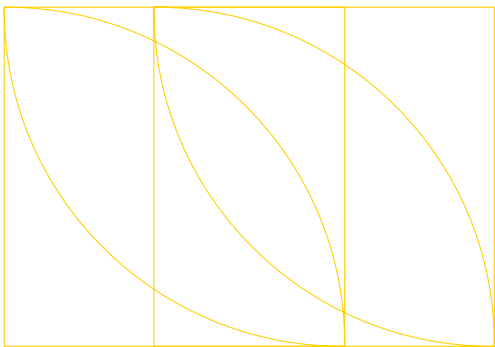
1. SRE

« It's what happens when you ask a software engineer to design an operations function »

Ben Treynor, Google

Comme nous l'avons vu précédemment, le DevOps est un ensemble de principes qui encouragent la collaboration entre les équipes produit et les exploitants. Le SRE (Site Reliability Engineer) est une réponse concrète à la philosophie DevOps : c'est un ensemble de bonnes pratiques et de convictions.

Ce mouvement est d'abord apparu en interne chez Google. Mais comme les équipes Google ont compris que les concepts portés (disponibilité, supervision...) étaient nécessaires pour communiquer avec leurs clients, ils ont rendu leur démarche publique notamment sur le site web **Site Reliability Engineering**.



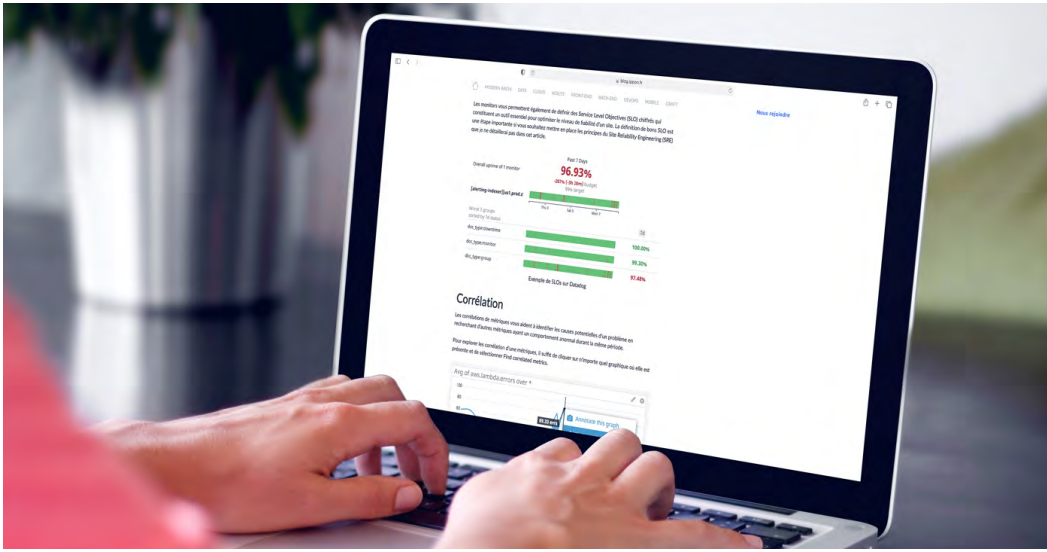
Une des convictions fondamentales du SRE est que les sujets d'exploitation peuvent se résoudre par du logiciel. En utilisant les bons outils, ou en développant de nouveaux outils si nécessaire, il est possible par exemple de détecter des anomalies sur les services et de les corriger automatiquement (détection par exemple d'un disque plein qui déclenche son redimensionnement).

En plus de ces convictions, le SRE apporte des réponses précises aux principes DevOps. Voici deux exemples :

Là où le mouvement DevOps propose de supprimer les silos entre développeurs et exploitants, le mouvement SRE propose d'atteindre cet objectif en co-développant avec les développeurs des outils pour simplifier les tâches quotidiennes (versioning des artefacts, déploiement des applications...).

Tandis que le mouvement DevOps rappelle que pour instaurer un climat de confiance, l'entreprise doit accepter les erreurs, le SRE détermine le SLO (Service Level Objective) des services et automatise le processus de reprise des pannes pour respecter le SLO. Cette garantie de SLO réduit le stress engendré par les pannes ainsi que leurs impacts.

Pour qu'un SRE puisse mener à bien ses deux missions (exploiter la plateforme et automatiser les tâches), son planning doit être adapté. Les SRE Google consacrent 50 % de leur temps à la gestion des systèmes (analyse - résolution des pannes...) et 50 % à développer la plateforme.



2. DEVSECOPS

Le DevSecOps (Development - Security - Operations) est une évolution du DevOps qui intègre les aspects sécurité dans le cycle de vie du développement logiciel. L'invention du terme DevSecOps est généralement attribuée à **Shannon Lietz** qui a aussi participé à la rédaction du manifeste **DevSecOps**.

L'apparition du mouvement DevSecOps s'explique par différentes raisons.

Pour commencer, le DevSecOps est une suite logique de l'émergence des méthodes agiles et du mouvement DevOps qui encourage la collaboration entre toutes les équipes : les équipes produit, les exploitants, la sécurité…

Ensuite, l'émergence des architectures microservices, pour remplacer les architectures monolithiques, nécessite de déterminer et implémenter les exigences de sécurité service par service.

Pour terminer, le nombre d'attaques (phishing, DDOS, ransomware) n'a jamais été aussi important et est en toujours en augmentation.

Le DevSecOps encourage la mise en place de bonnes pratiques. En voici trois.

Une première bonne pratique est évidemment la formation des équipes aux vulnérabilités des logiciels (cf. site **OWASP** qui liste les vulnérabilités principales ainsi que les techniques de défense) ainsi que la mise en place d'un processus de veille (cf. site de **l'ANSSI** qui permet de suivre les alertes et de déclarer les incidents).

Une deuxième est l'intégration des outils de sécurité dans les processus d'automatisation (analyse statique du code et analyse dynamique des applications) et de supervision des environnements.

Une troisième est la recherche proactive des failles de sécurité. Ceci peut être obtenu par exemple en mettant en place d'un côté des équipes "Rouges" qui cherchent des failles de sécurité et cherchent à les exploiter et de l'autre côté des équipes "Bleues" qui gèrent les incidents de sécurité.

3. FINOPS

Le FinOps (Finance - Operations) est une autre évolution de la démarche DevOps qui incite à maximiser l'apport de valeurs aux clients en guidant les choix techniques par des informations de coûts.

Pour mettre en place une démarche FinOps, il existe aujourd'hui de nombreuses sources d'informations.

Le site **finops.org** énumère notamment les principes du **mouvement FinOps** :

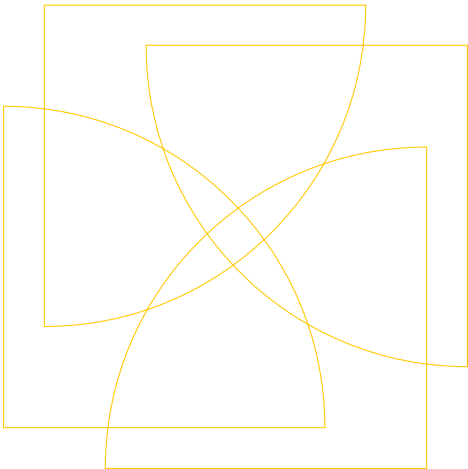
- Les équipes doivent être responsables de leurs utilisations du Cloud et donc avoir accès aux rapports de facturation pour comprendre les impacts de leurs usages et les corriger,
- Les équipes doivent tirer avantage des modèles de coûts variables du Cloud (paiement à l'usage, réservation de ressources).

Ce même site publie annuellement un **rapport sur l'état du FinOps**; ce rapport montre que l'objectif principal aujourd'hui d'une démarche FinOps est d'encourager les équipes techniques à optimiser les coûts en modifiant leurs utilisations des services cloud.

Un autre site intéressant est le **site finops.world**, qui publie des bonnes pratiques FinOps en les organisant par niveau de maturité de l'entreprise sur le sujet (démarrage, pérennité et généralisation) et par processus (stratégie, gouvernance, build...).

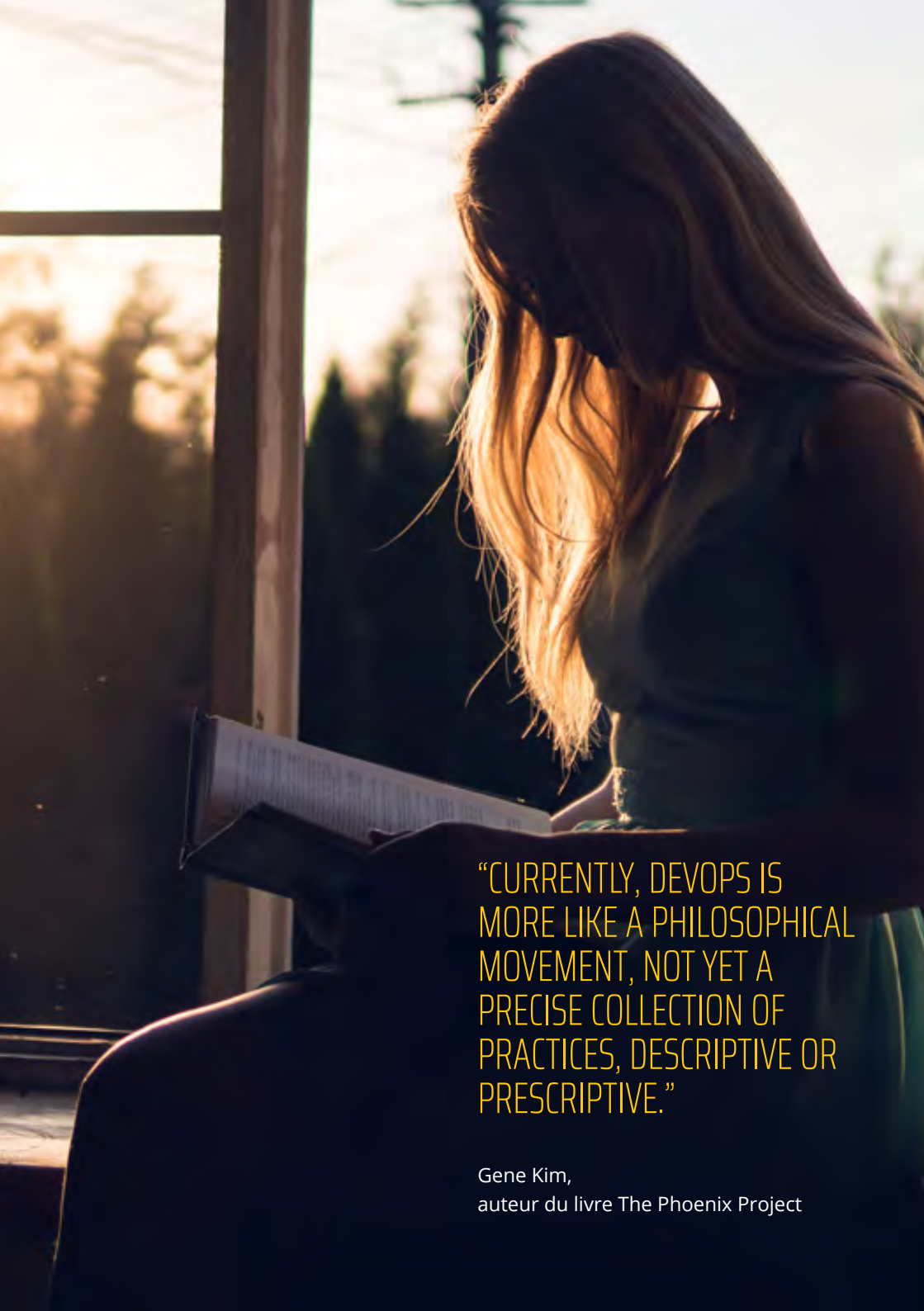
Un premier exemple de mise en place du FinOps est l'analyse régulière par les équipes des rapports de facturation : si un service coûte cher, l'équipe modifie son utilisation pour réduire les coûts ou le remplace par un autre service. Cette analyse nécessite la mise en place d'une politique d'étiquetage (tagging) des ressources.

Une démarche FinOps à l'échelle de l'entreprise peut par exemple aboutir à la réservation de capacités (VMs...). Sur AWS, la **réservation d'instances EC2** permet par exemple de réduire les coûts de 72 %.



VALEURS

02



"CURRENTLY, DEVOPS IS MORE LIKE A PHILOSOPHICAL MOVEMENT, NOT YET A PRECISE COLLECTION OF PRACTICES, DESCRIPTIVE OR PRESCRIPTIVE."

Gene Kim,
auteur du livre The Phoenix Project

Les valeurs DevOps sont une extension des valeurs agiles qui encouragent la collaboration entre équipes produit et exploitants. Ainsi, pour réussir une transformation DevOps, l'entreprise doit avoir mené sa transformation agile (ou être en cours de la réaliser).

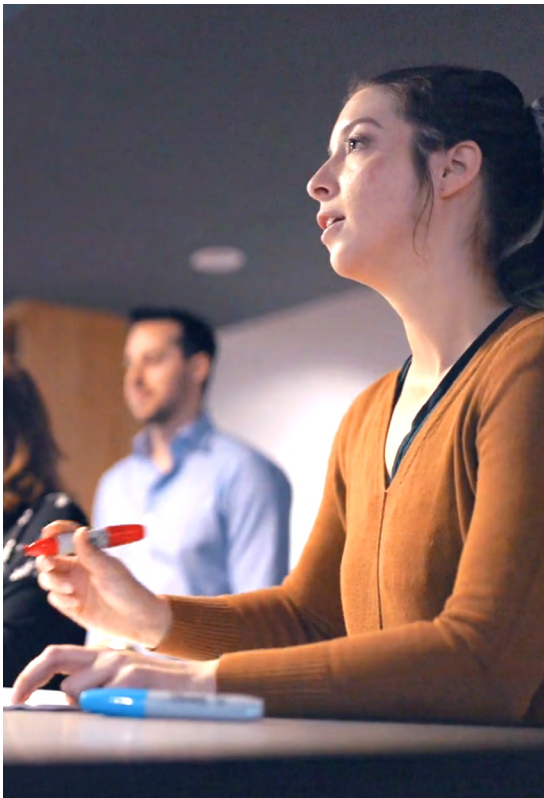
John Willis et Damon Edwards ont résumé (cf. article de blog sur la **Culture DevOps**) les valeurs fondamentales DevOps dans le modèle CAMS défini par :

- **Culture** : créer une responsabilité partagée entre les développeurs et les exploitants,
- **Automation** : automatiser les processus pour les fiabiliser, les accélérer, simplifier leur utilisation et leur évolution,
- **Measurement** : mesurer la satisfaction client ainsi que la performance des équipes et produits,
- **Sharing** : mettre en place un système de partage de connaissance entre les équipes (documentation, travail en binôme, retours d'expérience...).

Depuis, le modèle CAMS a été étendu en CALMS, L comme Lean, pour prendre aussi en compte des pratiques Lean : le Lean management est l'optimisation de l'apport de valeur aux clients en minimisant le gaspillage.

Le modèle CAMS a aussi été étendu en CALMR (R comme Recovery) pour gérer un élément essentiel pour l'exploitation : la reprise en cas de panne.

Dans les chapitres suivants, nous aborderons quelques pratiques DevOps courantes.



01 YOU BUILD IT, YOU RUN IT

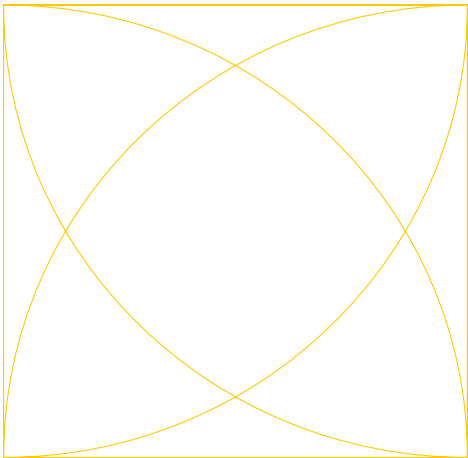
AFIN D'ACQUÉRIR UNE MEILLEURE COMPRÉHENSION DES ENJEUX DE LA PRODUCTION, L'ÉQUIPE DEVOPS ESTIME QU'ELLE DOIT ÊTRE RESPONSABLE DES CHOIX TECHNIQUES ET DE LEURS CONSÉQUENCES.

Ainsi, elle est responsable de la construction du produit (le BUILD) et de son MCO (Maintien en Condition Opérationnelle).

Ceci permet aux développeurs de mieux appréhender les contraintes de production (pannes, performance, volumétrie des données). Et, grâce à une collaboration accrue avec les développeurs, les exploitants acquièrent aussi une meilleure vision du fonctionnement des applications. Ceci les aide bien sûr dans leur mission d'exploitation des services.

Dans les méthodologies agiles, cette philosophie "You build it, You run it" prend tout son sens : elle permet de construire en parallèle le produit ainsi que le cadre nécessaire à son exploitation.

En revanche, dans une méthodologie waterfall, cette manière de faire serait moins pertinente : les équipes développent l'application puis livrent les informations nécessaires (Dossier d'exploitation) aux exploitants qui construisent la plateforme d'exploitation.



02 FAIL FAST. SUCCEED FASTER.

DANS L'ESPRIT DU LEAN START-UP OU DES BOUCLES DE RÉTRO-INFORMATION (FEED-BACK) DE L'EXTREME PROGRAMMING, LES ÉQUIPES DOIVENT TESTER RAPIDEMENT DES IDÉES ET DES TECHNOLOGIES.

Le concept du Fail Fast propose de tester des idées en limitant au maximum les conséquences de l'échec. Et quand une idée fonctionne, elle permet de gagner en productivité, qualité de service... (d'où le Succeed Faster).

Pour permettre l'émergence d'une culture Fail Fast qui est très importante pour l'innovation, l'organisation doit accepter les erreurs tout en essayant de les limiter au maximum.

Tant qu'une idée est en cours de test sur les environnements de Hors Production, son échec n'a pas réellement d'impact (sauf le temps consacré). À partir du moment où le test est déployé en Production, il peut y avoir de réels impacts sur les services déployés et sur les utilisateurs.

Pour limiter les impacts, il est prudent de procéder par étapes : tester sur une partie des utilisateurs ou sur une partie des services...

L'équipe doit aussi mettre en place des contrôles pour détecter les anomalies et des processus de rollback pour revenir à la version précédente.

03 DÉSACRALISATION DE LA PRODUCTION

LA PRODUCTION NE DOIT PAS ÊTRE UN ESPACE SACRÉ QUI NÉCESSITE UN CÉRÉMONIAL DE PLUSIEURS MOIS POUR FAIRE LA MOINDRE ÉVOLUTION.

La distance crée des appréhensions sur une pratique très rarement mise en œuvre. Nous l'avons tous entendu : "Ça marche, je n'y touche pas ! ".

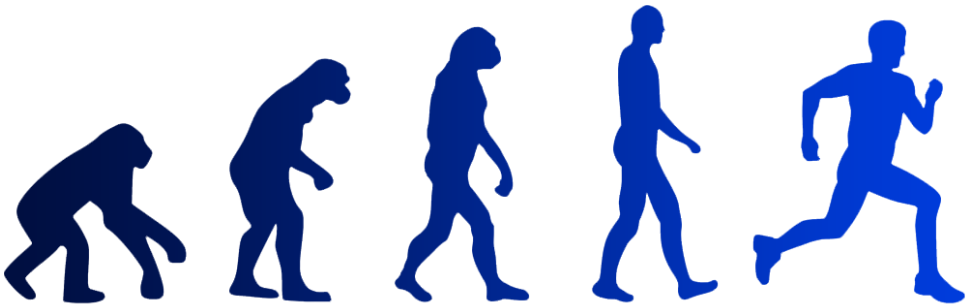
Le meilleur moyen de sortir de ces méfiances naturelles est de se confronter à la production en réalisant des mises en production régulières.

04 DEVOPS EST AGILE

Si l'on considère que le besoin d'une boucle de rétro-information (feedback) fait partie de l'agilité, que SCRUM demande de livrer à l'issue de chaque sprint "un produit susceptible d'être mis en production" et que les pratiques DevOps permettent de mettre automatiquement ce produit en production ; alors, la complémentarité des deux mouvements apparaît facilement.

La mise en production automatisée et régulière est la conséquence directe de la mise en place de l'agilité en entreprise.

La valeur du manifeste qui prône la communication apparaît également en lien direct avec le besoin de briser les barrières entre développeurs et exploitants.



WATERFALL **AGILE** **CONTINUOUS INTEGRATION** **CONTINUOUS DELIVERY** **CONTINUOUS DEPLOYMENT** **CONTINUOUS OPERATIONS**

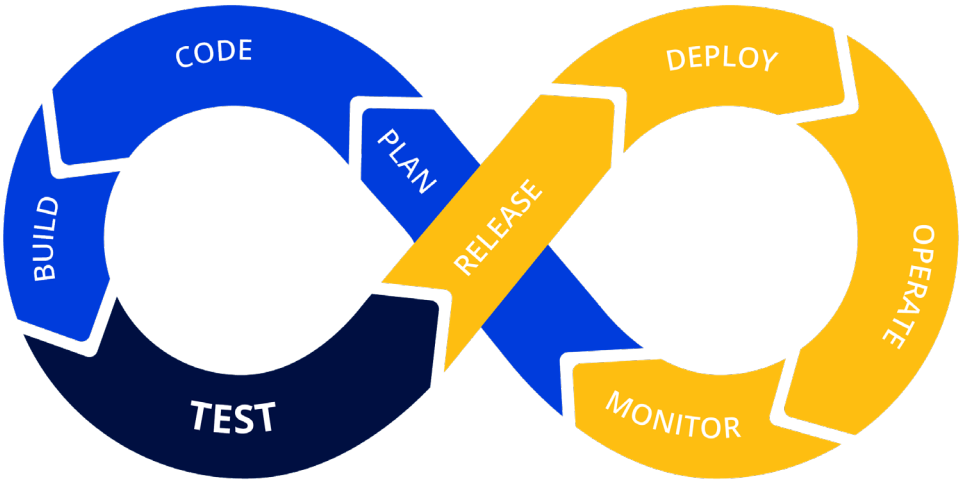
PROCESSUS

03

Un objectif important du DevOps est d'automatiser et d'industrialiser la chaîne de production de manière à :

— Réduire les délais de mise à disposition des fonctionnalités aux utilisateurs (Time-to-Market),
— Améliorer la qualité du produit,
— Améliorer l'efficacité des équipes.

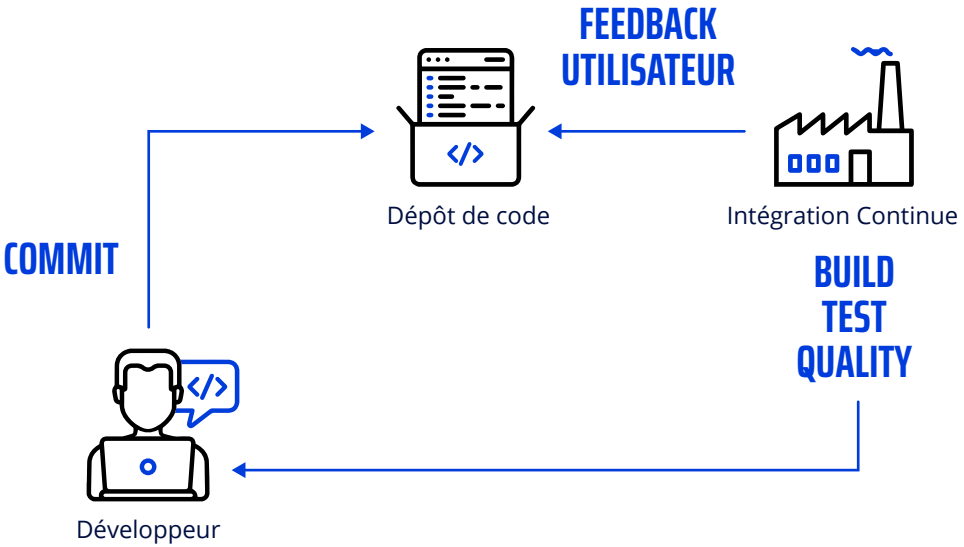
Les processus DevOps sont souvent représentés par le symbole "infini" (cf. figure ci-dessous). Ce symbole illustre le fait que les équipes réalisent régulièrement des cycles complets de développement jusqu'au déploiement tout en exploitant les services déployés.



01 INTÉGRATION CONTINUE

L'INTÉGRATION CONTINUE (CONTINUOUS INTEGRATION) EST UNE MÉTHODOLOGIE OÙ LES DÉVELOPPEURS INTÈGRENT LES MODIFICATIONS DE CODE FRÉQUEMMENT DANS LE RÉFÉRENTIEL DE CODE SOURCE (GIT...).

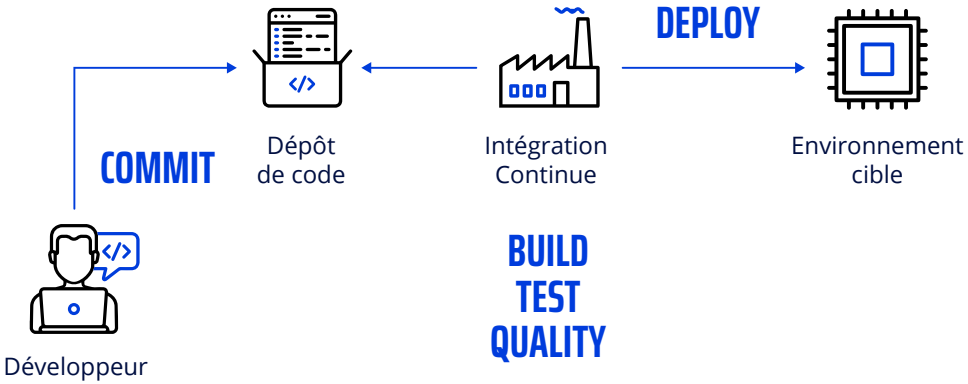
Chaque intégration de code est validée par un processus automatisé qui construit les livrables du produit (notamment les RELEASE), exécute les tests, analyse la qualité de code... Le processus remonte un feedback (rétro-information) aux équipes concernées en cas d'erreur (mail, slack...).



02 DISTRIBUTION ET DÉPLOIEMENT CONTINU

La distribution continue (continuous delivery) est une pratique où le code est prêt à être déployé en production, mais nécessite auparavant une validation (tests manuels d'acceptance, coordination avec d'autres déploiements...).

Le déploiement continu (continuous deployment) consiste à déployer automatiquement (sans approbation humaine) le nouveau code en production.



La mise en place du déploiement continu nécessite un certain nombre de garde-fous. En voici quelques exemples :

- Déployer de petits changements pour réduire les risques (fonctionnalité par fonctionnalité),
- Implémenter le Feature Flipping qui permet de remplacer à chaud une implémentation d'une fonctionnalité par une ancienne implémentation (réputée fiable),
- Implémenter des processus de déploiement progressifs qui déploient l'application à un petit nombre d'utilisateurs et réalisent un retour arrière s'ils détectent des erreurs.

Les exemples les plus connus d'entreprises qui réalisent du déploiement en continu (en production) sont des grandes entreprises (AWS, Netflix…) qui conçoivent des architectures microservices et réalisent quotidiennement des nombres impressionnants de mises en production (plusieurs milliers pour AWS).

Le déploiement continu permet à ces grands acteurs technologiques d'optimiser le Time-to-Market. Pour ces entreprises, ne pas déployer en continu serait synonyme de déploiement par lots de tailles importantes. Et ceci augmente la complexité à déterminer rapidement les origines des anomalies et à les corriger.

Au vu de la maturité nécessaire pour sa réalisation industrielle, le déploiement continu est souvent limité aux environnements de développement. Cette pratique permet aux développeurs de tester en continu leurs développements sur un environnement iso-production.

STRATÉGIES DE DÉPLOIEMENT

Le choix d'une stratégie de déploiement dépend des contraintes des services développés.

Quelle est la qualité de service attendue (mémoire/CPU alloué…) ? Est-il acceptable de réduire la qualité de service durant le déploiement voire d'arrêter le service ?

Quel est le coût acceptable ? Comme nous le verrons dans cette partie, les stratégies de déploiement progressifs nécessitent de l'outillage complexe, ainsi que plus de ressources.

Le choix d'une stratégie de déploiement implique aussi des contraintes.

En effet, pour déployer une nouvelle version d'un service sans rupture, les développeurs doivent gérer la rétrocompatibilité des schémas de données : l'ancienne version du service doit fonctionner avec le nouveau schéma de données.

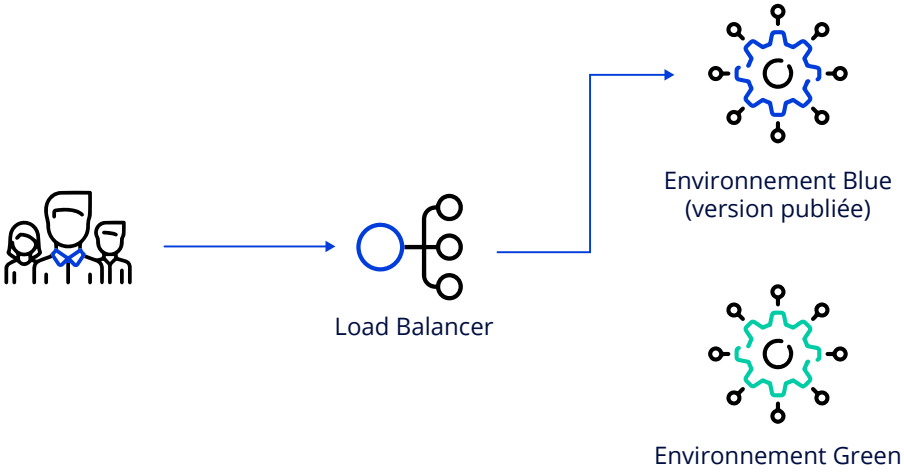
Si cette condition n'est pas respectée, deux versions successives du service ne peuvent pas fonctionner simultanément sur la même base de données. Le processus de déploiement doit donc, dans ce cas, arrêter le service avant le déploiement puis le démarrer après le déploiement.

Sans rétrocompatibilité, le retour arrière sur un déploiement (ou ROLLBACK) impose aussi certaines contraintes comme l'utilisation d'une image de la base de données prise avant le déploiement avec le risque de perdre des données créées ou mises à jour après la création de l'image.

1. BLUE/GREEN DEPLOYMENT

Le Blue/Green deployment permet de déployer une nouvelle version d'un service sans rupture du service et sans impacter ses performances (i.e. conserver la même allocation mémoire/CPU).

Ce modèle de déploiement impacte en revanche les ressources allouées pendant le déploiement. En effet, le déploiement Blue/Green gère deux environnements : un environnement Blue pour l'ancienne version du service et un environnement Green pour la nouvelle version.

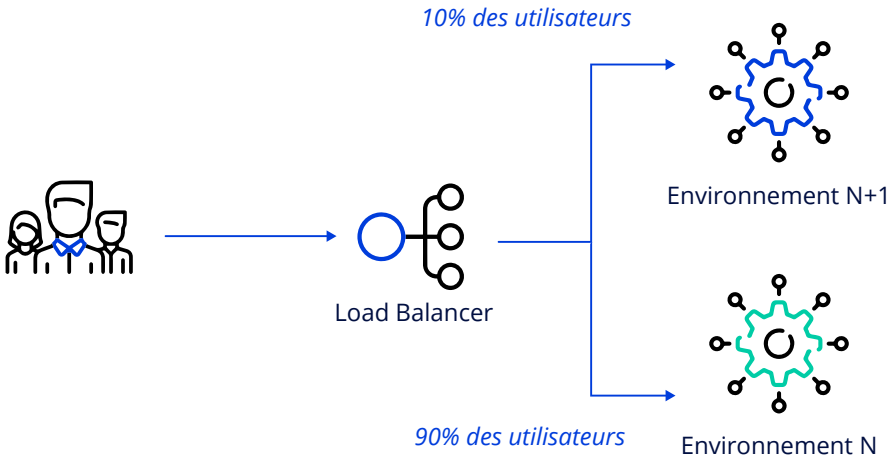


Le "Blue/ Green deployment" est constitué par :

- Le déploiement de la nouvelle version sur l'environnement "Green" sans l'ouvrir aux utilisateurs,
- La validation de l'environnement Green (tests automatiques, tests manuels, supervision de l'application…),
- Si l'environnement se comporte correctement, ouvrir l'environnement Green aux utilisateurs (et fermer l'ancien environnement) et supprimer l'environnement Blue,
- Sinon, annuler le déploiement.

2. CANARY DEPLOYMENT

Le Canary Deployment permet de limiter le risque de déploiement d'une nouvelle version en l'ouvrant progressivement aux utilisateurs.



Le "Canary deployment" est constitué des étapes suivantes :

- Le déploiement de la nouvelle version et l'ouvrir à une partie des utilisateurs,
- La validation de l'environnement Green (tests automatiques, tests manuels, supervision de l'application…),
- Si la version se comporte correctement, ouvrir le service à une plus grande proportion d'utilisateurs,
- Sinon annuler le déploiement.

03 — INFRASTRUCTURE AS CODE

L'INFRASTRUCTURE AS CODE CONSISTE À MODÉLISER LES SERVICES D'INFRASTRUCTURE (VM, BASES DE DONNÉES, DROITS...) PAR DU CODE (FICHIERS DE RESSOURCES YAML, CODE DÉCLARATIF TERRAFORM...).

Cette pratique permet :

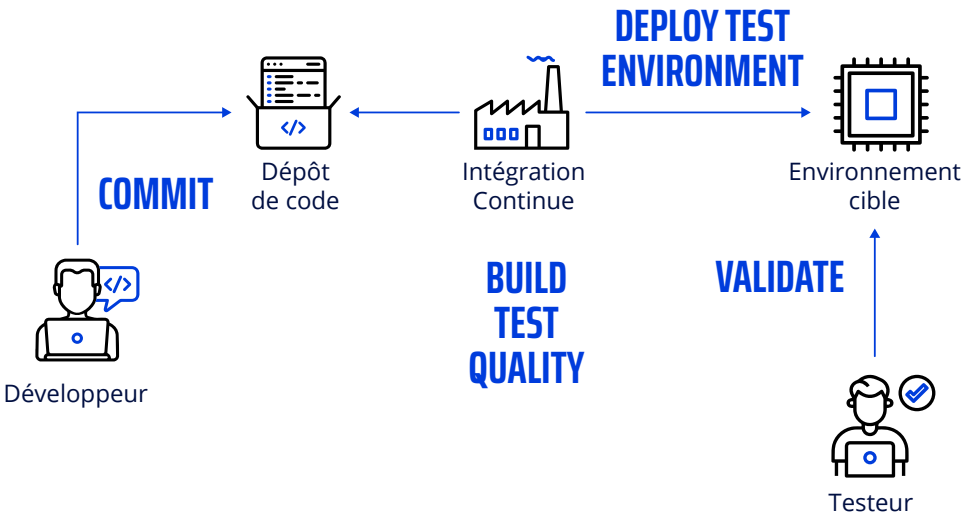
- D'accélérer le déploiement des environnements et de s'assurer que les déploiements sont reproductibles,
- De gérer tous les environnements de la même manière (tous les environnements sont iso-production) et donc d'éviter les différences entre les environnements,
- De simplifier le respect de règles de gouvernance (exemple : tagging des ressources) en développant des modules d'infrastructure réutilisables,
- De simplifier la création de services d'infrastructure (exemple : base de données, vm...) en libre service (catalogue de services).

La capacité de déployer rapidement de nouveaux environnements a engendré de nouvelles pratiques comme la création d'environnements de test éphémères.

Cette gestion d'environnements éphémères s'intègre parfaitement au processus de développement.

Voici un scénario d'utilisation des environnements éphémères :

— Quand un développeur démarre le traitement d'une User Story, il crée une branche Git. Puis le job de CICD réagit à la création de la branche en déployant le code dans un nouvel environnement dédié.
— Quand le développeur valide un changement dans GIT, la CICD le déploie immédiatement sur l'environnement associé; ce qui permet au développeur de tester son développement dans cet environnement (et non pas uniquement sur son poste de développement).
— Quand le développeur finalise son développement, il crée une Merge Request (MR). Et la personne qui valide la MR teste le développement dans l'environnement associé.
— Finalement, quand la MR est validée, l'outil de gestion de code supprime la branche de développement et le job de CICD réagit à la suppression de la branche en détruisant l'environnement.



04 SUPERVISION & OBSERVABILITÉ

Dans ce chapitre, nous aborderons deux notions complémentaires : la supervision et l'observabilité.

La supervision permet aux équipes de détecter les anomalies/pannes du système alors que l'observabilité permet de comprendre l'état du système, comprendre son fonctionnement et bien sûr déterminer les sources des anomalies.

> SI VOUS NE POUVEZ PAS LE MESURER, VOUS NE POUVEZ PAS L'AMÉLIORER
> LORD KELVIN, PHYSICIEN BRITANNIQUE DU 19E SIÈCLE.

1. SUPERVISION

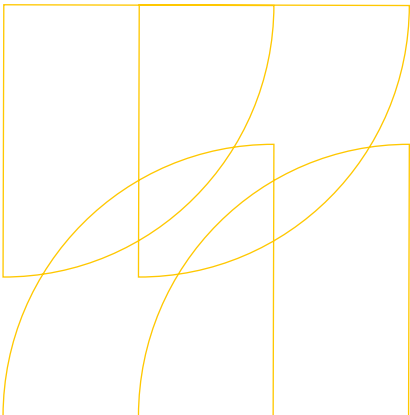
La supervision d'un système est l'activité de collecte et d'affichage, en temps réel, de données relatives au système. Cette activité permet aux équipes de détecter des anomalies du système.

Si par exemple, les temps de réponse d'une API augmentent, les équipes sont notifiées par une alarme et doivent ensuite déterminer l'origine du problème et le corriger.

Les équipes doivent définir leur stratégie de supervision :

- Quels sont les indicateurs à superviser ?
- Quels sont les seuils/valeurs critiques ?
- Quels sont les niveaux de criticité ?

Les SRE google ont construit et publié un ensemble de bonnes pratiques pour la supervision d'application (cf. site web **Monitoring De Systèmes Distribués**).

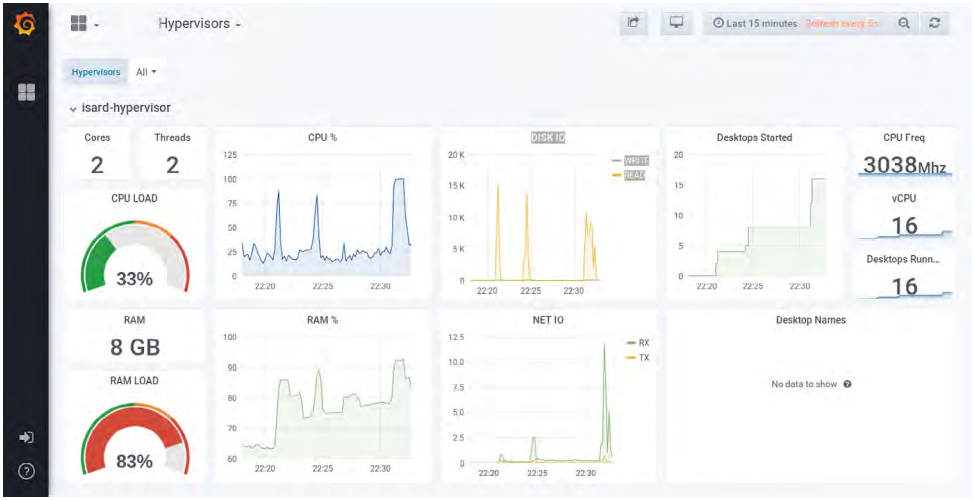


Parmi ces bonnes pratiques, ils préconisent un suivi des 4 Signaux d'Or (4 Golden Signals)

— **Latence** : durée totale de traitement des requêtes,
— **Traffic** : mesure de la montée en charge du système (nombre de requêtes par seconde),
— **Errors** : mesure du nombre de requêtes en erreur,
— **Saturation** : mesure de la saturation des ressources (pourcentage d'utilisation du disque...).

Comme nous le verrons par la suite, il existe différents types de supervision.

La supervision d'infrastructure remonte des indicateurs sur l'état de santé des services d'infrastructure (disponibilité, pourcentage d'utilisation du CPU/Disque/RAM...).



La supervision applicative (Application Performance Monitoring ou APM) remonte des métriques liées à la performance des applications (disponibilité, temps de réponse des API, nombre de threads...).

Le Real User Monitoring (RUM) recueille, au niveau application, les métriques de performance :

— Durée pour que les pages soient prêtes (page ready time),
— Durée de rendu des pages,
— Latence des requêtes.

En réalisant les mesures au niveau de l'application, le RUM est une mesure de l'expérience utilisateur. Le RUM permet aussi d'identifier, du point de vue de l'utilisateur, des anomalies du système qui ne peuvent pas toujours être détectées par les outils de supervision (car les anomalies sont externes).

La supervision fonctionnelle d'une application consiste à collecter des métriques "métier" de l'application.

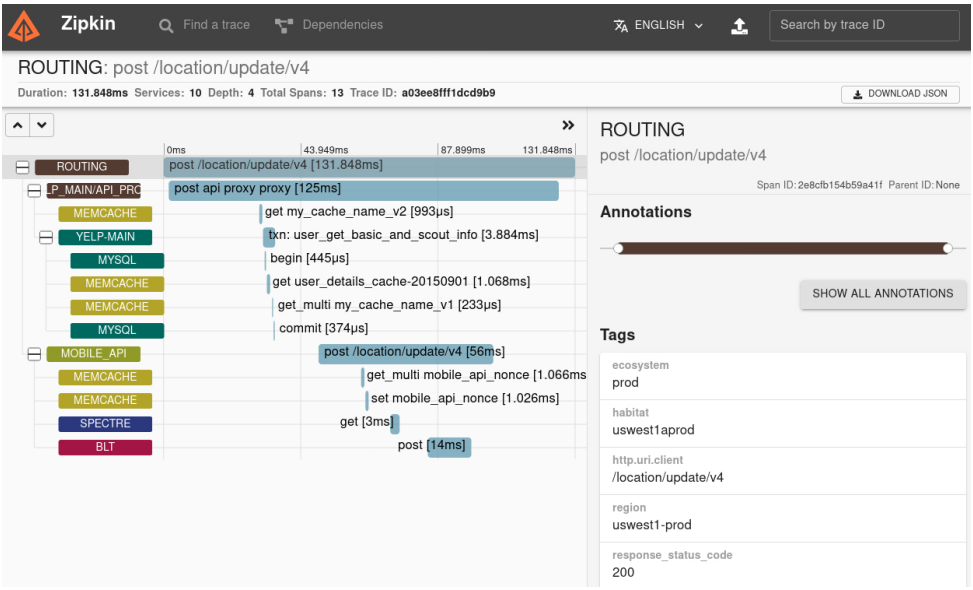
2. OBSERVABILITÉ

L'observabilité est une activité complémentaire à la supervision : c'est la capacité de comprendre l'état interne du système à partir de ce qu'il produit.

L'observabilité est construite sur 3 piliers : les métriques, les logs, et les traces.

Pour un site e-commerce, par exemple, une métrique utile à suivre est le taux de transformation sur un pipeline de vente. Cette métrique doit être suivie de très près par l'équipe : tous les développements réalisés doivent améliorer cette métrique et toute détérioration notable de cette métrique doit être étudiée et corrigée.

Un outil d'observabilité important est l'outil de traçage des requêtes. En effet, il peut construire, à partir d'un appel de service passé, le graphe complet des appels de services engendrés.



Ce graphe d'appel de services permet par exemple de comprendre la latence globale du service et donc envisager des optimisations.

Dans le cas d'une anomalie sur un appel de services, l'outil de traçage permet de visualiser le contexte de l'appel (en effet, l'anomalie arrive sur un appel de service qui fait partie d'un traitement plus global) et de trouver une solution pour la corriger.

OUTILS

04

Bien qu'il soit intéressant de dissocier intellectuellement les pratiques des outils, il faut garder en mémoire qu'il y a un enrichissement permanent entre les outils et les pratiques : les outils poussent de nouvelles pratiques qui donnent naissance à de nouveaux outils.

Le but de ce document n'est pas de faire une étude de ces outils, mais d'en présenter quelques-uns qui répondent aux besoins DevOps.

01 GESTION DU CODE

LES OUTILS DE GESTION DE CODE SONT CENTRAUX DANS LES PROCESSUS DEVOPS.

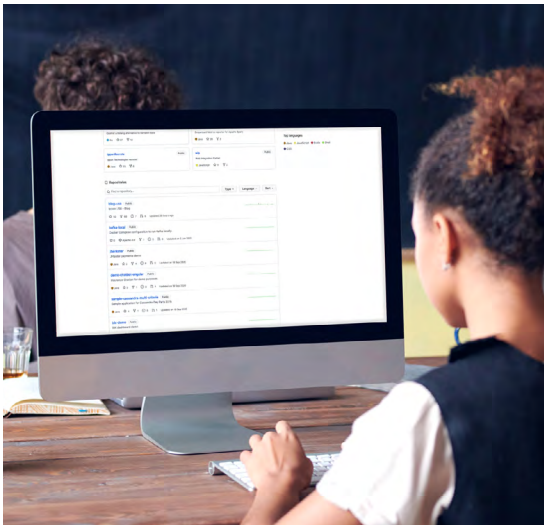
Sur de nombreux sujets, le code est devenu “source de vérité”. Les équipes gèrent, par exemple, les wikis en markdown, les schémas de migration SQL en yaml (**flyway**, **liquibase**), les APIs REST en yaml (**OpenAPI**), l’infrastructure par des fichiers de description de ressources (Cloudformation, **terraform**…).

Certains outils comme les outils d’analyse qualité (SonarQube) s’intègrent aussi avec le gestionnaire de code. Ainsi, le développeur qui réalise une revue de code, trouve les annotations qualité sur le code modifié ; ce qui est une excellente aide pour la revue de code.

Voici quelques exemples d’outils :

Git est l’outil de gestion de code le plus utilisé. Il existe de nombreux sites Web (Gitlab, Github, bitbucket…) qui simplifient son utilisation (fork, pull request…) et fournissent de plus en plus de services annexes (qualité de code, CICD, gestion des tâches…).

Pour illustrer le côté central du code dans les processus DevOps, le service Gitlab s’est énormément transformé et se décrit comme une **plateforme DevOps** (et non plus comme une application de gestion de code).

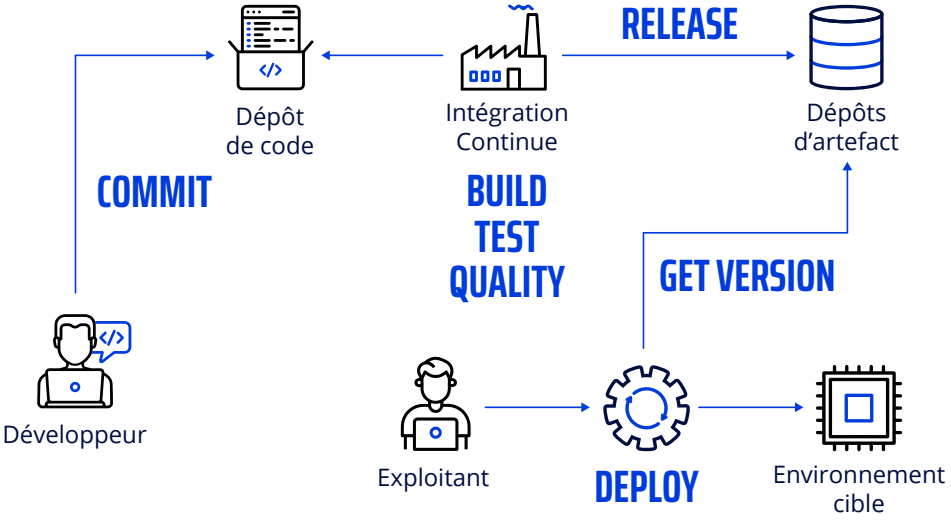


02 INTÉGRATION CONTINUE

LE SERVEUR D'INTÉGRATION CONTINUE (CI) EST LA PIERRE ANGULAIRE DE L'AUTOMATISATION DES PROCESSUS.

Les Jobs d'Intégration Continue sont souvent découpés en étapes :

- **BUILD** :
  - Exécution des tests unitaires
  - Construction des artefacts,
- **TEST** : Exécution des tests d'intégration, des tests E2E,
- **QUALITY** : Analyse de la qualité du code,
- **RELEASE** : Versionning des artefacts,
- **DEPLOY** : Déploiement dans un environnement donné.



Les solutions d'Intégration Continue permettent de définir des processus complexes avec des langages dédiés (exemple : jenkinsfile, Github actions...).

En utilisant ces langages de modélisation de processus CICD, le développeur peut modéliser par exemple un processus qui déclenche la création d'un environnement à la création d'une branche Git. Cette pratique est poussée à l'extrême dans l'approche GitOps où toutes les opérations de déploiement sont déclenchées par des commandes Git.

Voici quelques exemples d'outils :

Jenkins est depuis très longtemps la solution CICD la plus déployée en entreprise. Une des forces de Jenkins est son écosystème extrêmement riche composé de centaines de plugins (intégration avec de nombreux outils DevOps notamment).

Gitlab CI est la solution CICD open source qui concurrence le plus Jenkins (qui est un peu vieillissant). Son langage de description des processus CICD ainsi que certains points clés de son architecture (intégration directe dans un gestionnaire de code, les runners gitlab pour l'exécution des Jobs) le rendent très performant et agréable à utiliser.

Bitrise est une CICD SAAS mobile qui gère les serveurs de Build (MacOS notamment) et l'outillage mobile (SDK Android/iOS/Flutter/...).

AWS CodePipeline est le service CICD d'AWS. CodePipeline encourage les bonnes pratiques AWS (exemple : déploiement Blue/Green avec CodeDeploy sur AWS...) et s'intègre très bien avec l'écosystème AWS (gestion de code codecommit, sécurité IAM, supervision cloudwatch...).

1. ANALYSE STATIQUE

Les outils d'analyse statique permettent de détecter des défauts dans le code : bugs, failles de sécurité (failles OWASP notamment), code trop complexe (exemple : méthode avec trop de paramètres, manque de structure du code...).

Voici quelques exemples d'outils :

SonarQube est aujourd'hui l'outil le plus utilisé pour l'analyse de code. C'est un outil extensible (architecture à base de plugins) qui gère un nombre impressionnant de langages et s'intègre avec de nombreux outils CICD (Jenkins, Gitlab).

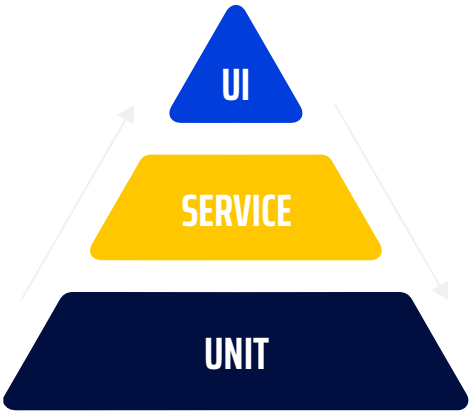
Snyk est un nouveau concurrent de SonarQube capable d'analyser du code, des dépendances applicatives (npm, maven...), des images docker ainsi que du code d'infrastructure (terraform, kubernetes...). En plus de remonter des vulnérabilités, Snyk est capable de fournir un plan de remédiation (proposer par exemple la version à utiliser pour une dépendance détectée vulnérable).

2. TEST

Les outils d'analyse statique permettent de détecter des défauts dans le code : bugs, failles de sécurité (failles OWASP notamment), code trop complexe (exemple : méthode avec trop de paramètres, manque de structure du code...).

La démarche de test d'une application a été formalisée par Mike Cohn dans son livre "Succeeding with Agile" (Cohn, 2010) en utilisant la métaphore de la pyramide :

- Implémenter un nombre suffisant de tests unitaires (de manière à couvrir les fonctionnalités importantes),
- Un peu moins de tests d'intégration (car plus coûteux à développer et à maintenir),
- Et encore moins de tests d'IHM (car plus coûteux et moins fiables).



Bien que cette vision pyramidale soit de plus en plus remise en question (le test d'IHM est par exemple de plus en plus fiable et rapide), il reste important de définir une stratégie de test adaptée à l'application.

Voici quelques exemples d'outils :

Terratest permet de tester des ressources déployées avec un outil IAC. Un scénario de test consiste à déployer des ressources, vérifier certaines assertions sur les ressources déployées (par exemple vérifier le bon fonctionnement d'un conteneur dans kubernetes), puis supprimer les ressources déployées. Initialement basé sur terraform, l'outil permet aujourd'hui de tester d'autres solutions IAC comme Docker ou Kubernetes.

Gatling permet de réaliser des tests de charge sur des applications Web. L'objectif est de vérifier que l'application se comporte bien dans des conditions définies (nombre d'utilisateurs cible avec des parcours réalistes et une volumétrie de données réaliste). Une des forces de Gatling est le choix d'implémenter les scénarios avec un DSL (langage spécifique au domaine) basé sur le langage de programmation **Scala** et le framework asynchrone **Akka**.

Puppeteer est un outil d'automatisation des tests E2E pour des applications Web. Une des forces de Puppeteer est d'exécuter les tests en pilotant un chrome headless (en mode sans IHM) avec le **protocole DevTools de Chrome**.

Chaos Monkey, projet open source initialement développé par Netflix, permet de tester la résilience d'applications aux pannes. Durant un test, les équipes vérifient le bon fonctionnement des applications pendant que l'outil génère des pannes aléatoires sur l'infrastructure.

3. GESTION DES ARTEFACTS

Les processus de CI/CD gèrent le stockage des artefacts dans des dépôts (repositories en anglais).

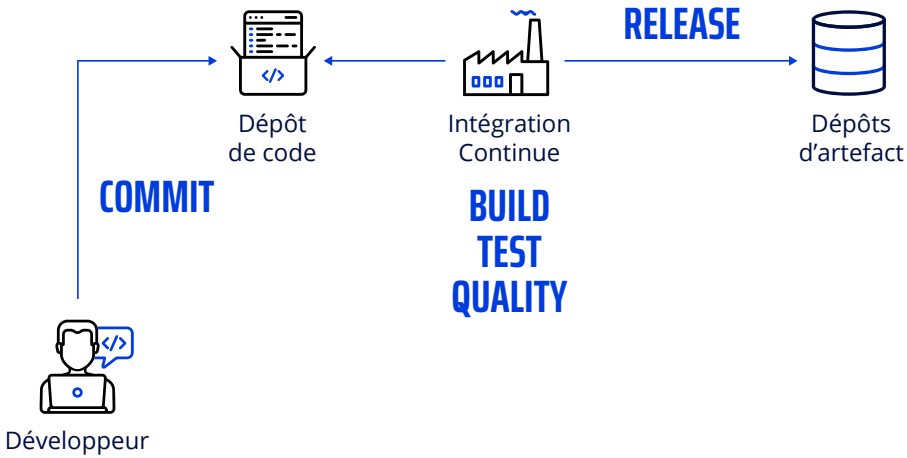
Pour prendre un exemple :

— Le processus BUILD construit et archive l'artefact dans un dépôt,
— Le processus DEPLOY télécharge l'artefact à partir du dépôt et le déploie.

L'utilisation de dépôts d'artefacts permet ainsi de supprimer le couplage entre les processus de construction d'une application et son déploiement.

Les dépôts d'artefacts implémentent, en plus de la fonction de stockage, des fonctions de sécurisation et de cache.

Pour détecter des failles de sécurité, les dépôts réalisent des scans de vulnérabilité (notamment **CVE**) sur les packages stockés.



Pour gérer l'intégrité des artefacts, ceux-ci doivent être signés. La manière de signer un artefact dépend de son type : **Docker Content Trust** permet de signer les images Docker.

Ensuite, l'utilisation des dépôts en tant que cache permet d'éviter l'accès fréquent aux dépôts publics : ces accès peuvent être lents et/ou onéreux. Docker Hub a depuis 2020, pour les comptes anonymes, une limite de 100 téléchargements d'images par jour.

Voici quelques exemples d'outils :

Nexus Repository et **JFrog Artifactory** sont deux dépôts d'artefacts open source qui gèrent les artefacts des langages de programmation les plus courants, des images docker, des packages linux (apt, rpm)...

Harbor est un dépôt d'image Docker "Kubernetes natif" qui implémente des fonctions :

- De stockage d'images Docker et de charts Helm
- De sécurisation des images Docker (scans de vulnérabilité, signature)
- Et de synchronisation avec les principales solutions concurrentes (JFrog, ECR, GCR...); ce qui est très pratique pour les entreprises qui ont une stratégie multi Cloud.

03 — DÉPLOIEMENT

LE DÉPLOIEMENT REGROUPE DIFFÉRENTES ACTIVITÉS COMME LE DÉPLOIEMENT DE L'INFRASTRUCTURE, LA GESTION DE CONFIGURATION (POUR DÉPLOYER DES SERVICES SUR DES VMS ET LES CONFIGURER) ET POUR TERMINER LE DÉPLOIEMENT DES APPLICATIONS.

The **Twelve-Factor App** liste un ensemble de bonnes pratiques à suivre pour le déploiement et l'exploitation des services.

1. DÉPLOIEMENT DE L'INFRASTRUCTURE

Les outils d'Infrastructure As Code permettent de déployer rapidement et de manière fiable des environnements de façon totalement automatisée.

Voici quelques exemples d'outils :

Terraform est aujourd'hui l'outil Infra As Code le plus en vogue. Le langage utilisé HCL (Hashicorp Configuration Language) est agréable à utiliser et bien géré par les IDE. Cet outil prend en charge les providers Clouds principaux (AWS, GCP, Azure...) ainsi que **beaucoup d'autres services** y compris des plateformes de virtualisation comme **VMware**.

Cloudformation est le service Infra As Code managé d'AWS. Étant la solution officielle AWS, ce service est très bien intégré aux services AWS (prise en charge par exemple rapide des nouvelles fonctionnalités AWS) et encourage aussi le respect de bonnes pratiques AWS. **SAM** est une variante de Cloudformation pour les architectures serverless.

2. GESTION DE CONFIGURATION

Les outils de gestion de configuration permettent d'automatiser des tâches de gestion de serveurs : installation et configuration de services (exemple : serveur Web), déploiement de fichiers...

Voici quelques exemples d'outils :

Ansible est un outil de gestion de configuration python très utilisé. Une des raisons de son succès est son architecture qui ne nécessite pas d'installer d'agents sur les machines cibles. Cet outil est complété par l'application Web **AWX** qui fournit une interface Web, une API et une solution d'orchestration des tâches. **Ansible Galaxy** est un référentiel de playbooks ansible open source réutilisables.

Saltstack est aussi un outil de gestion de configuration python. Une de ses forces est sa conception basée sur un bus de messages qui le rend adapté à de très gros systèmes. Salt gère un dépôt GIT de **formules Salt réutilisables**.

3. DÉPLOIEMENT DES APPLICATIONS

Pour schématiser, un processus simple de déploiement d'une application peut être vu comme un processus incrémental qui :

— Sélectionne des VM à mettre à jour,
— Déploie la nouvelle version de l'application sur ces VMs.

Le déploiement se termine quand la nouvelle version est déployée sur toutes les VMs.

Il existe deux grandes stratégies de déploiement des applications.

La première, la plus ancienne, consiste à scripter les déploiements dans une VM. Cette stratégie simple à mettre en place peut poser des problèmes :

— Que faire, par exemple, si le script de déploiement échoue avant d'avoir terminé ? Dans quel état sont les VMs ? Est-ce que le retour arrière est simple ?
— Les scripts de déploiement sont généralement testés dans des conditions adaptées (VM qui contient une version récente de l'application). Comment être sûr qu'un script de déploiement fonctionne sur une ancienne VM ?

La seconde stratégie consiste à remplacer la VM (ou conteneur Docker) par une VM (ou image Docker) qui contient la nouvelle version de l'application.

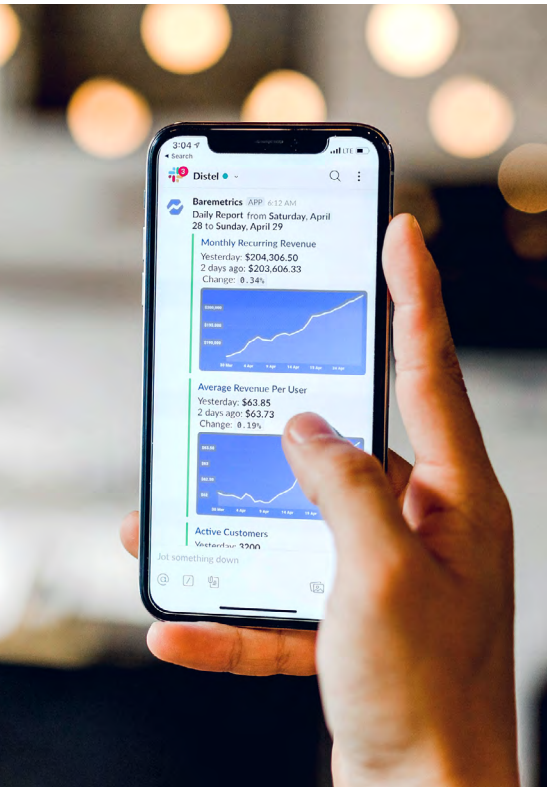
Voici quelques exemples d'outils :

AWS CodeDeploy est un outil AWS de déploiement qui permet de déployer des applications sur des VMs, des applications conteneurisées ainsi que des Lambda en gérant des déploiements progressifs (Blue/Green, Canary).

FluxCD ou **ArgoCD** sont des solutions GitOps de déploiement natives Kubernetes qui implémentent entre autres des fonctions de déploiements progressifs (Blue/Green, Canary). Les déploiements sont réalisés avec des commandes Git. Ainsi pour déployer du code dans un environnement, il suffit de le valider dans une branche du même nom.

04 — SUPERVISION & OBSERVABILITÉ

POUR SUIVRE L'ÉTAT DES SERVICES DÉPLOYÉS (APPLICATIONS, SERVICES MANAGÉS, VMS...), LES ÉQUIPES UTILISENT DES SOLUTIONS DE SUPERVISION ET D'OBSERVABILITÉ.



Voici quelques exemples d'outils :

Grafana est une solution complète de visualisation, d'analyse et de surveillance des métriques.

Prometheus est une solution de collecte des métriques et d'alerting qui s'appuie sur une base de données timeseries. Pour avoir une belle visualisation des données Prometheus, Prometheus s'intègre comme source de données de Grafana.

La Suite Elastic est une solution de centralisation des logs basée sur

- Des agents (beats, logstash) de remontée de logs et métriques
- Un cluster ElasticSearch pour indexer et requêter les données
- Un kibana pour visualiser, interroger et analyser les données

Jaeger est un produit open source de traçage des requêtes développé initialement par **Uber Technologies**. Jaeger permet entre autres de visualiser les transactions distribuées, d'analyser les latences des requêtes.

05 — SÉCURITÉ

LA MISE EN PLACE D'UNE DÉMARCHE DE SÉCURITÉ CONSISTE À DÉTECTER AU PLUS TÔT LES MENACES ET LES TRAITER.

La sécurité doit être gérée à la construction des services (BUILD) et à leur exécution (RUN). Il est utile d'appuyer la démarche sécurité par des règles de gouvernance.

Voici quelques exemples d'outils :

Clair est un outil d'analyse statique de vulnérabilités pour des images Docker. Pour détecter les vulnérabilités, Clair s'appuie entre autres sur la base de données de vulnérabilités **CVE**.

Kube-bench permet de vérifier qu'un cluster Kubernetes respecte les règles de sécurité définies dans le **Benchmark CIS pour Kubernetes**. Le **CIS** est un organisme à but non lucratif qui aide les entreprises et les gouvernements à se protéger des cybermenaces.

Falco est un produit open source initialement développé par **Sysdig**, qui permet de sécuriser et superviser des conteneurs docker. Falco permet de suivre des comportements inhabituels comme l'élévation de privilèges, l'écriture dans des répertoires système en lecture seule (exemple /usr/bin), ou encore l'exécution de commandes ssh…

Cilium est un firewall pour conteneurs. En plus de contrôler les flux entre Pods (ce qui est le comportement par défaut du firewall K8S), Cilium permet aussi d'appliquer des règles de filtrage de niveaux IP, TCP et HTTP. Une règle de niveau HTTP permet par exemple, en fonction de l'origine de l'appel, de filtrer les appels de services autorisés par ressource REST et par verbe HTTP.

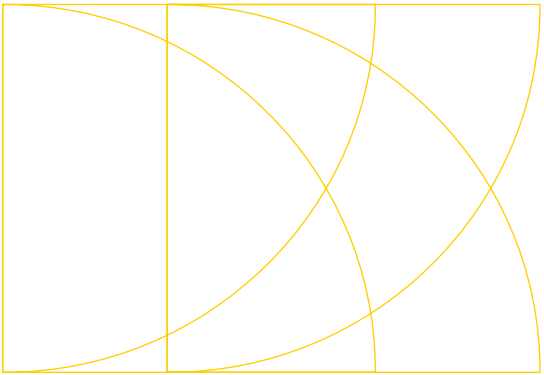
06 GOUVERNANCE

LES OUTILS DE GOUVERNANCE PERMETTENT DE VÉRIFIER LA CONFORMITÉ DES RESSOURCES DÉPLOYÉES AVEC DES POLITIQUES EN VIGUEUR. UN EXEMPLE DE POLITIQUE SOUVENT MISE EN VIGUEUR EST L'ÉTIQUETAGE DES RESSOURCES (TAGGING) UTILISÉ POUR LE SUIVI DE LA FACTURATION.

Voici quelques exemples d'outils :

AWS Config est un service d'inventaire des ressources AWS. AWS Config permet aussi de vérifier la conformité des ressources par rapport à des règles définies et exécuter des actions de remédiation sur les ressources non conformes.

Open Policy Agent (OPA) permet de mettre en vigueur des politiques (policy en anglais). OPA comporte un langage de modélisation des politiques ainsi qu'une API. Ainsi, l'intégration d'OPA dans Kubernetes (**Gatekeeper**) permet de mettre en place des politiques comme "interdire l'exécution de conteneurs dockers issus de dépôts inconnus"



ORGANISATION
DES ÉQUIPES

05

> DANS LA MISE EN PLACE D'UNE DÉMARCHE DEVOPS, L'ORGANISATION DES ÉQUIPES EST FONDAMENTALE.

Ces aspects d'organisation sont très bien décrits pour les équipes agiles dans le livre Team Topologies (Skelton & Pais, 2019) puis déclinés pour les équipes DevOps dans l'article ***DevOps Topologies***.

D'après cet ouvrage, il existe quatre types d'équipes :

- Stream aligned team : équipe alignée sur un flux de valeur,
- Enabling team : équipe qui assiste les Stream aligned teams (équipes supports…),
- Complicated Subsystem : équipe qui implémente un sous-système complexe et qui nécessite un haut niveau d'expertise,
- Platform team : équipe qui fournit un produit aux Stream aligned teams pour améliorer leur productivité.

Et trois types d'interactions entre les équipes :

- La collaboration : les deux équipes collaborent,
- La facilitation : une équipe aide et mentore l'autre équipe,
- Et X-as-a-service : une équipe fournit un service à l'autre équipe.

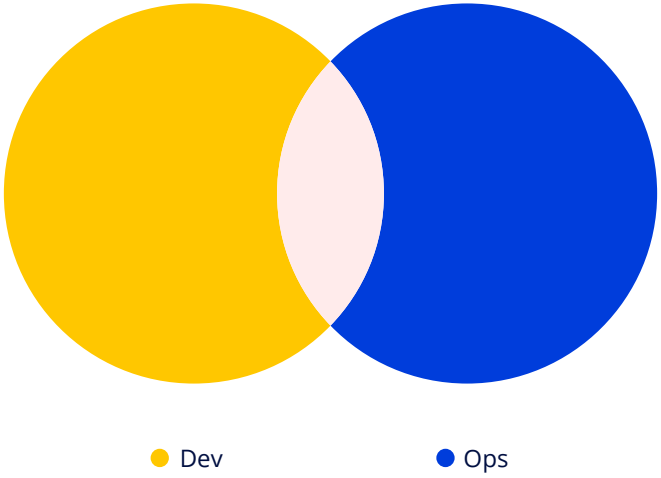
Nous énumérons dans cette partie, des exemples d'organisations DevOps extraits de l'article DevOps Topologies. Pour chaque type d'organisation, nous décrivons son mode de fonctionnement ainsi qu'un contexte où elle est pertinente.

01 COLLABORATION DEV ET OPS

DANS DES ENTREPRISES MATURES TECHNIQUEMENT, LES ÉQUIPES DE DÉVELOPPEMENT ET D'EXPLOITATION COLLABORENT DIRECTEMENT. C'EST LE MODE DE FONCTIONNEMENT "PAR DÉFAUT" DEVOPS.

Voici quelques exemples de collaborations :

- Les exploitants fournissent aux équipes de développement des consignes à respecter pour le déploiement des applications (observabilité, sécurité…),
- Les équipes de développement aident les exploitants à développer des bibliothèques de fonctions Infrastructure As Code réutilisables.

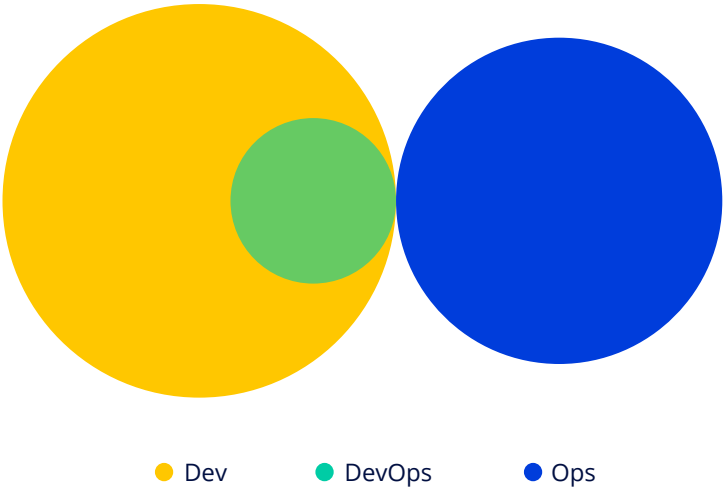


02

OPS AS INFRASTRUCTURE-AS-A-SERVICE

DANS DES ENTREPRISES OÙ L'IT RESTE TRADITIONNELLE, IL PEUT ÊTRE INTÉRESSANT D'AVOIR UNE ÉQUIPE D'EXPLOITANTS QUI AGIT COMME UN FOURNISSEUR DE RESSOURCES D'INFRASTRUCTURE.

Dans ce cas, l'équipe de développement comporte une sous-équipe DevOps qui déploie et exploite les services en utilisant les ressources fournies (par les exploitants). C'est cette équipe qui collabore avec les exploitants.

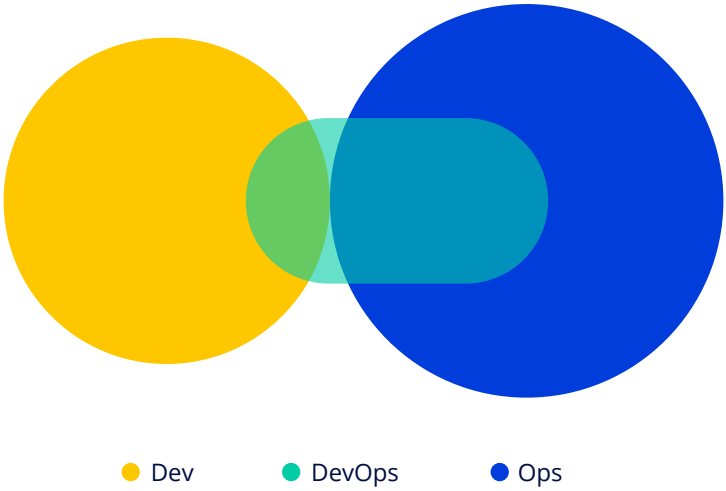


03 DEVOPS-AS-A-SERVICE

DANS LES ENTREPRISES QUI NE PEUVENT PAS, POUR DIFFÉRENTES RAISONS (TAILLE, COMPÉTENCES...), PRENDRE EN CHARGE LES ACTIVITÉS D'EXPLOITATION, IL PEUT ÊTRE INTÉRESSANT DE LES SOUS-TRAITER À DES FOURNISSEURS DE SERVICES.

Pour éviter de créer un clivage entre les équipes du client et celles du fournisseur de service, il faut créer une relation de collaboration entre les deux équipes, ainsi que des objectifs communs.

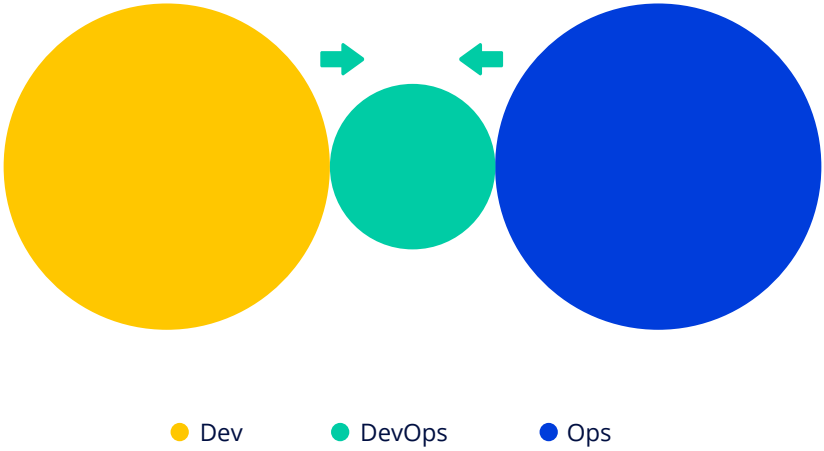
Cette organisation peut aussi être un excellent moyen, pour l'entreprise, de monter en compétences sur les sujets DevOps, en observant une équipe experte travailler.



04 ÉQUIPE DEVOPS TEMPORAIRE

DANS DES ÉQUIPES MATURES MAIS OÙ LA MISE EN PLACE D'UNE COLLABORATION DEV ET OPS SEMBLE RISQUÉE (FRILOSITÉ DES ÉQUIPES), IL EST POSSIBLE DE METTRE EN PLACE UNE ÉQUIPE DEVOPS TEMPORAIRE.

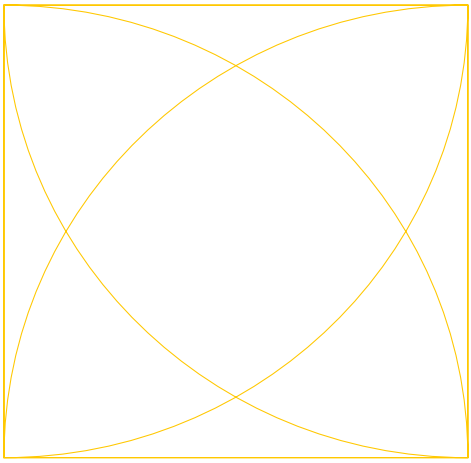
L'objectif de cette équipe est d'apprendre aux deux équipes à collaborer en apportant la culture DevOps.
Quand c'est le cas, l'équipe DevOps Temporaire perd de son utilité : il est temps de la dissoudre pour la remplacer par une collaboration directe entre les équipes produit et les exploitants.



TROIS VOIES

06

> DANS SON ARTICLE DE BLOG THE THREE WAYS: *THE PRINCIPLES UNDERPINNING DEVOPS* PUIS DANS SES LIVRES THE PHOENIX PROJECT (BEHR ET AL., 2018) ET THE DEVOPS HANDBOOK (DEBOIS ET AL., 2016), GENE KIM A SCÉNARISÉ 3 VOIES DE MISE EN PLACE DE PRATIQUES DEVOPS.



La première voie est basée sur une démarche Lean : elle consiste à partir des flux de valeur, les automatiser (CICD, infra as code…) et les optimiser. Dans le Lean, un flux de valeur (ou Value Stream) est l'ensemble des activités réalisées pour délivrer un besoin qui apporte de la valeur au client.

La deuxième voie est une démarche d'amélioration continue basée sur une boucle de rétroaction. Les équipes observent le comportement du système (métriques, log…), détectent des anomalies (bugs, lenteurs…) et les corrigent.

La troisième voie est celle de l'expertise et de l'expérimentation. En se basant sur leur expertise, les équipes consacrent du temps pour améliorer l'existant.

01 — PREMIÈRE VOIE

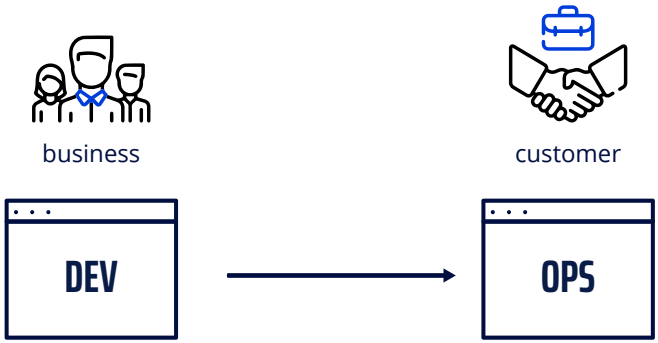
L'OBJECTIF DE LA PREMIÈRE VOIE EST D'OPTIMISER GLOBALEMENT LES VALUE STREAMS DU SYSTÈME :

- Réduire le temps entre la prise en compte d'un besoin d'un client et sa livraison aux utilisateurs,
- Réduire les temps d'indisponibilité du système.

L'optimisation doit être globale : il ne faut donc pas optimiser localement les Value Stream sans vérifier l'impact global.

La mise en place de cette voie repose évidemment sur l'automatisation des processus :

- Construction et validation des livrables (exécution des tests unitaires, analyse qualité du code),
- Déploiement des applications,
- Construction des environnements.



Mais pour dépasser cette première étape nécessaire, l'entreprise doit avoir une bonne vision des flux de valeurs ainsi qu'une démarche pour les optimiser.

Cette vision peut être obtenue en cartographiant les Value Stream par Value Stream Mapping et en utilisant cette cartographie pour identifier des axes d'amélioration comme les goulets d'étranglement (nombre de testeurs insuffisants), ou le gaspillage (temps d'attente entre équipes, travail non fini, travail non nécessaire…).

Les pratiques agiles encouragent aussi des pratiques d'organisation du travail comme :

— Utiliser un outil de management visuel (Kanban) pour rendre visible le travail accompli,
— Limiter le nombre de tâches en cours pour améliorer les délais de livraison.

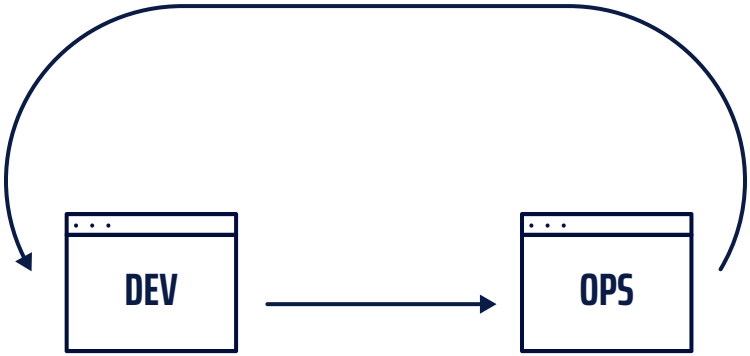
02 — DEUXIÈME VOIE

L'OBJECTIF DE LA DEUXIÈME VOIE EST DE METTRE EN PLACE UNE BOUCLE DE RÉTROACTIONS RAPIDES POUR VALIDER LE TRAVAIL ACCOMPLI ET DÉTECTER LES ERREURS OU ANOMALIES.

Pour illustrer le besoin de feedback, voici un échange classique entre un Développeur et un Exploitant :

— Développeur : Mon déploiement est bon, il est passé sur l'environnement de développement,

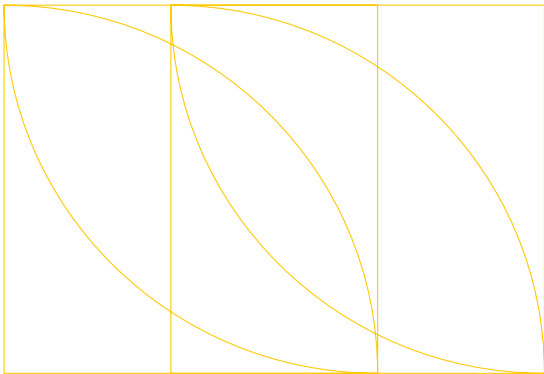
— Exploitant : Certes, mais là tu as déployé en production et ton API utilise un index de base de données en cours de construction : ton API retourne des codes 500. Je dois donc faire un retour arrière sur le déploiement et attendre la fin de la création de l'index pour redéployer.



Après cet échange, la bonne pratique apprise est que le processus de déploiement doit séquencer les deux déploiements : celui de l'index et celui du code métier qui l'utilise. Pour éviter de gérer à nouveau cette situation, le développeur a ensuite modifié le processus de déploiement pour exécuter des tests avant déploiement (test de l'état des index) : si les index sont en cours de construction, le déploiement est arrêté.

Pour réaliser de manière efficace cette boucle de rétroactions, il existe des bonnes pratiques à mettre en place :

— Une base documentaire adaptée,
— Du peer review pour valider la qualité des changements réalisés,
— Des tests d'acceptance du système,
— Des logs, des métriques et des alarmes pour suivre le comportement des services.

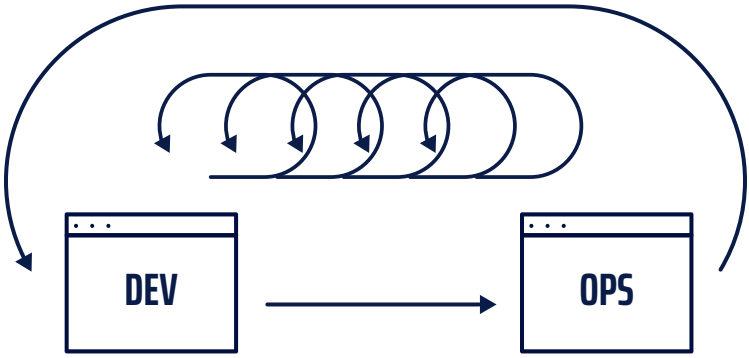


03 TROISIÈME VOIE

L'OBJECTIF DE CETTE TROISIÈME ET DERNIÈRE VOIE EST DE CRÉER UNE CULTURE D'AMÉLIORATION CONTINUE BASÉE SUR L'EXPÉRIMENTATION.

Dans cette voie, les équipes sont encouragées à expérimenter de nouvelles pratiques (code, architecture, services), puis les généraliser à l'existant. C'est un excellent moyen pour réduire la dette technique.

Pour encourager cette pratique, il est important d'allouer du temps (exemple : 20 % du temps) aux équipes.



METTRE EN PLACE UNE DÉMARCHE DEVOPS

07

La démarche DevOps est un excellent levier pour améliorer la qualité des produits développés, la performance des équipes ainsi que l'esprit de collaboration et de responsabilité commune.

Pour initier la démarche, l'équipe responsable de la transformation DevOps définit les objectifs (cf. **Objectifs**, p66) et évalue la maturité DevOps de l'organisation (cf. **Maturité DevOps**, p67).

Ensuite, l'équipe construit une feuille de route qui intègre des éléments structurants comme les processus d'automatisation, le suivi des indicateurs (cf. **Élément clés**, p71) ainsi que des bonnes pratiques reconnues (cf. **Bonnes pratiques**, p72).

> COMME POUR TOUTE TRANSFORMATION, LA MISE EN PLACE DE LA DÉMARCHE DEVOPS PEUT COMPORTER CERTAINS RISQUES. NOUS EN DÉTAILLERONS QUELQUES-UNS DANS LE CHAPITRE **LES RISQUES**, (P 79).

01 OBJECTIFS

LA MISE EN PLACE D'UNE DÉMARCHE DEVOPS PEUT ÊTRE MOTIVÉE PAR DES OBJECTIFS TRÈS DIFFÉRENTS.

Elle peut être guidée par des besoins des clients :

- Accélérer l'apport de valeur aux clients (Time-to-Market),
- Améliorer la qualité et la stabilité des services développés.

Elle peut aussi être déclenchée par les difficultés des équipes à réaliser leur travail quotidien :

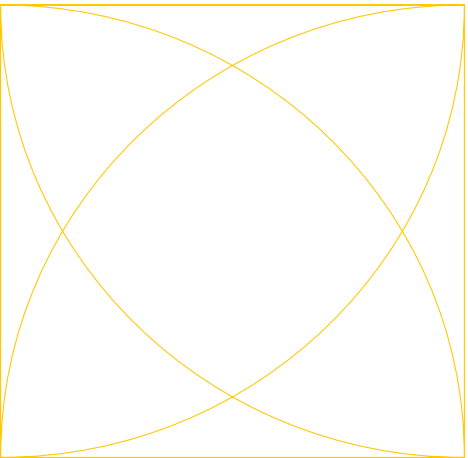
- Des déploiements complexes,
- Une trop grande dépendance entre les équipes (nécessité de se synchroniser avec d'autres déploiements),
- Une difficulté à analyser et corriger les anomalies des services.

Elle peut également être poussée par des objectifs financiers comme le R.O.I (Return On Investment - retour sur investissement) ou la gestion des coûts.

Dans une démarche d'amélioration continue, il faut choisir, pour chaque objectif, un indicateur et suivre son évolution.

02 MATURITÉ DEVOPS

L'ÉVALUATION DU NIVEAU DE MATURITÉ DEVOPS PERMET DE VÉRIFIER QUE LES ÉQUIPES TROUVENT UN ÉQUILIBRE ENTRE LES PRATIQUES AGILES (QUI PERMETTENT AUX ÉQUIPES DE FOURNIR DE LA VALEUR RAPIDEMENT AUX CLIENTS) ET LES BESOINS DE STABILITÉ DES SYSTÈMES (OBJECTIF DES EXPLOITANTS).



L'évaluation permet aussi de définir des axes d'amélioration et peut être utilisée comme base d'une démarche d'amélioration continue. En effet, en évaluant régulièrement son niveau, l'équipe détermine si elle s'améliore.

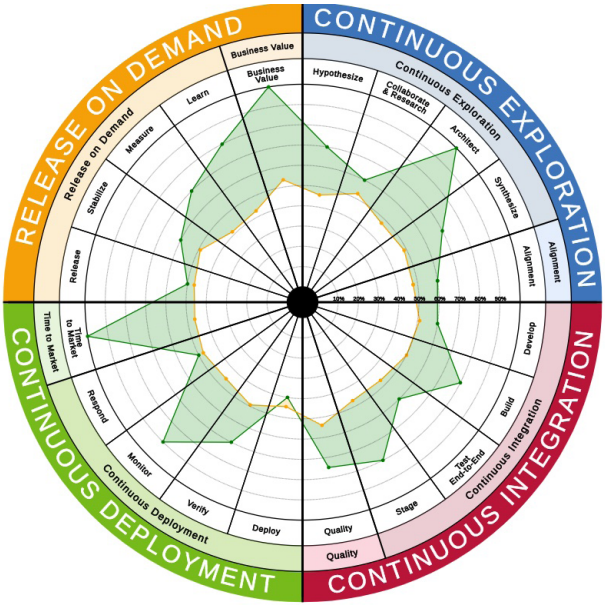
Nous présenterons, dans ce chapitre, deux outils d'évaluation de maturité.

Le premier est développé par le framework Agile SAFe (cf. site web **SAFe**). En tant que framework agile à l'échelle, SAFe gère la mise en place de l'agilité au niveau de l'entreprise en abordant par exemple des aspects comme la collaboration entre équipes. SAFe est aujourd'hui le framework agile à l'échelle le plus utilisé.

Le second est développé par le groupe de recherche DORA DevOps Research and Assessment (cf. site web **DORA**) qui aide les entreprises à améliorer leurs pratiques DevOps.

1. SAFE

Pour évaluer son niveau de maturité, SAFe propose un questionnaire d'auto-évaluation (cf. Outil d'évaluation SAFe). Le résultat est un radar similaire à la copie d'écran ci-dessous.



Les niveaux de maturité sont définis métaphoriquement par 5 types de mouvement du plus lent au plus véloce : SIT (assis, niveau 0), CRAWL (rampe, niveau 1), WALK (marche, niveau 2), RUN (cours, niveau 3) et FLY (vole, niveau 4).

Comme SAFe ne définit pas clairement les niveaux, il est intéressant de définir sa propre interprétation. Voici un exemple :

— SIT : Build et Déploiement manuel,
— CRAWL : Build et Déploiement automatisé,
— WALK : Respect de règles "d'entreprise" (qualité, gouvernance, sécurité),
— RUN : Démarche d'amélioration continue,
— FLY : Déploiement en continu.

2. DORA

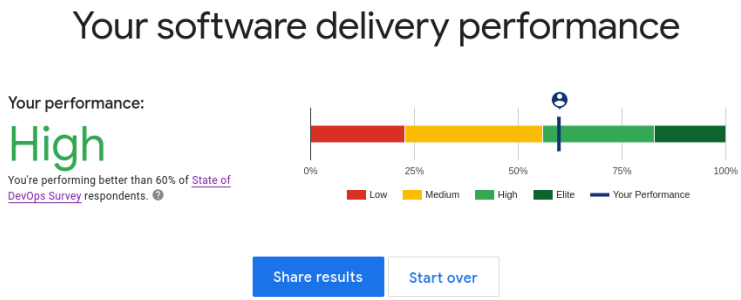
DORA propose aussi un questionnaire (cf. site **Outil d'évaluation DORA**) qui permet de visualiser graphiquement sa maturité DevOps (cf. copie d'écran ci-dessous).

Le questionnaire définit 4 niveaux de maturité : Low (faible), Medium (moyen), High (élevé), Elite (élite).

Ces 4 niveaux de maturité sont basés sur les métriques DORA (cf. **DORA**, p75) qui mesurent le niveau d'agilité des équipes et le niveau de stabilité des systèmes construits :

— Une équipe peu mature (Low) est peu agile (fréquence faible de livraison) et le système construit est peu fiable (temps de reprise de panne important),
— Une équipe très mature (Elite) est très agile (fréquence élevée de livraison) et le système construit est très fiable (temps de reprise de panne faible).

L'outil permet aussi d'afficher son niveau par rapport aux autres entreprises du même domaine.



L'outil propose aussi en fin de questionnaire des axes d'amélioration adaptés et très bien documentés.

Key DevOps capabilities recommended for you

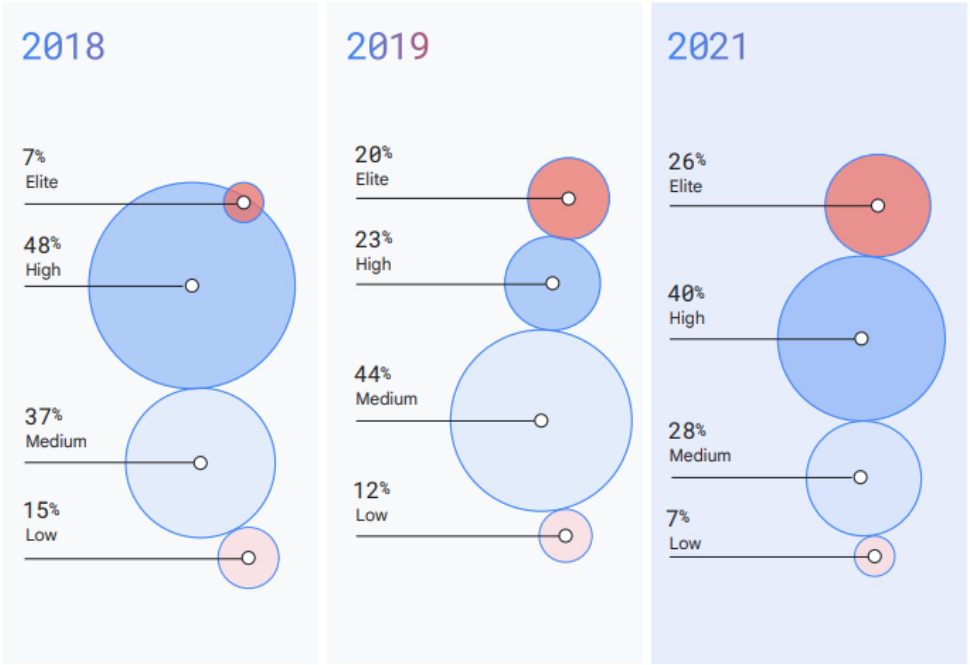
Based on your performance profile, we think you should work on the three capabilities listed below. Based on your responses to the capabilities assessment, **we recommend you focus on working in small batches first.**

Working in small batches KEY FOCUS AREA

Teams should slice work into small pieces that can be completed in a week or less. The key is to have work decomposed into small features that allow for rapid development, instead of developing complex features on branches and releasing them infrequently. This idea can be applied at the feature and the product level. (An MVP is a prototype of a product with just enough features to enable validated learning about the product and its business model.) Working in small batches enables short lead times and faster feedback loops.

View solution

Le groupe de recherche DORA publie aussi annuellement un **rapport** qui fait un statut sur le DevOps. Ce rapport montre une progression rapide du niveau de maturité des entreprises (7% d'entreprises Elite en 2018, 26 % en 2021).



03 — ÉLÉMENTS CLÉS

NOUS AVONS DÉJÀ ABORDÉ DANS CE DOCUMENT LES ÉLÉMENTS CLÉS D'UNE DÉMARCHE DEVOPS.

Comme le DevOps est une extension de l'agilité, l'entreprise doit déjà avoir mis en place une démarche agile ou être en train de le faire.

Ensuite, comme expliqué par les trois voies, l'organisation doit mettre en place des processus :

D'automatisation :

- Intégration continue,
- Déploiement (des applications et de l'infrastructure),
- Test des applications,

D'amélioration continue :

- Mise en place de boucles de rétroactions le long des flux de valeur,
- Mise en place de référentiels de bonnes pratiques,
- Supervision de la performance des équipes et de la qualité des services (exemple : métriques DORA),

Et d'exploration :

- Test de nouveaux patterns d'architecture et services,
- Impact de l'existant à partir des nouveaux apprentissages.

04 BONNES PRATIQUES

NOUS ABORDERONS DANS CE CHAPITRE, QUELQUES BONNES PRATIQUES À SUIVRE POUR AUGMENTER LES CHANCES DE SUCCÈS DE LA MISE EN PLACE DU DEVOPS.

Pour se familiariser avec les enjeux et apports de la démarche, il est important de s'appuyer sur une documentation de référence (cf. **Documentation**, p72).

Pour aligner les équipes sur les flux de valeur, il faut commencer par les modéliser (cf. **Value stream mapping**, p76).

Il est intéressant d'améliorer sa démarche DevOps en travaillant aussi sur des aspects techniques comme la conception basée sur le modèle d'architecture microservices (cf. **Architectures microservices**, p77) ou la migration des applications vers le Cloud (cf. **Move to cloud,** p78).

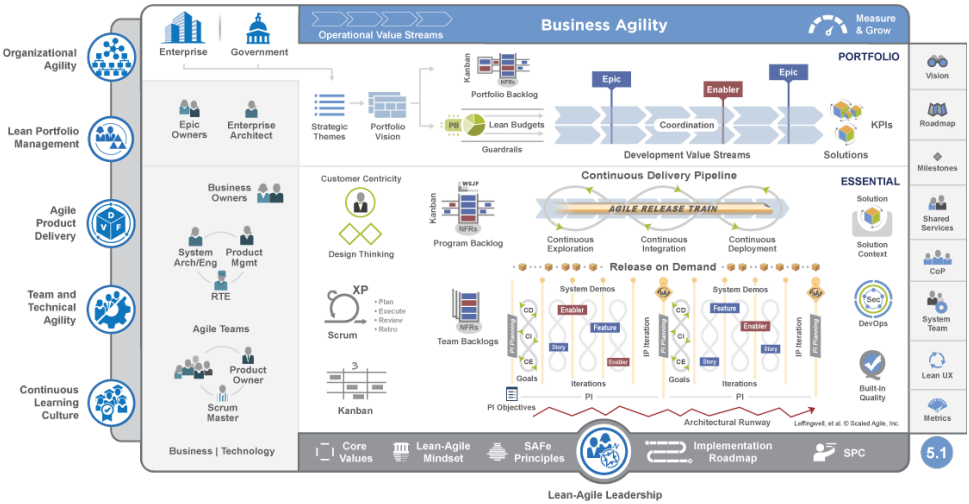
1. DOCUMENTATION

Pour mettre en place une démarche DevOps, les équipes doivent s'appuyer sur une documentation de référence. Dans cette section, nous aborderons deux sources d'informations importantes : SAFe et DORA.

Au-delà de ces deux sources, un ouvrage de référence à lire absolument est The DevOps Handbook (Debois et al. 2016).

1.1 SAFE

Une des spécificités du framework SAFe est qu'il est très prescriptif et cette caractéristique en fait une source d'information intéressante. En effet, comme illustré par la copie d'écran ci-dessous, SAFe décrit le mode de fonctionnement des équipes (compétences, processus…).



Sur les aspects DevOps, SAFe encourage par exemple la mise en place de processus d'Intégration Continue, de Déploiement Continue, de Release On Demand (Versionning des artefacts) et d'Exploration Continue (pour l'amélioration continue).

Comme illustré par le radar ci-dessous, SAFe découpe ces 4 processus en activités. L'Intégration Continue comporte par exemple des activités de Développement, de Build, de Test…

SAFe aborde aussi des aspects organisationnels comme la coordination des activités des équipes (livraisons, test...) qui participent à un même flux de valeur, par un train de livraison (Agile Release Train).

1.2 DORA

DORA propose des axes d'amélioration dans 4 thématiques (technique, processus, métrologie et culture) et propose pour chaque thématique des bonnes pratiques.

Au sujet de la gestion du code, DORA recommande par exemple de suivre une démarche trunk based : les développeurs fusionnent quotidiennement leur code dans une branche commune à tous les développements (la branche trunk). Cette recommandation s'adresse notamment à des équipes expérimentées qui souhaitent mettre en place un déploiement continu.

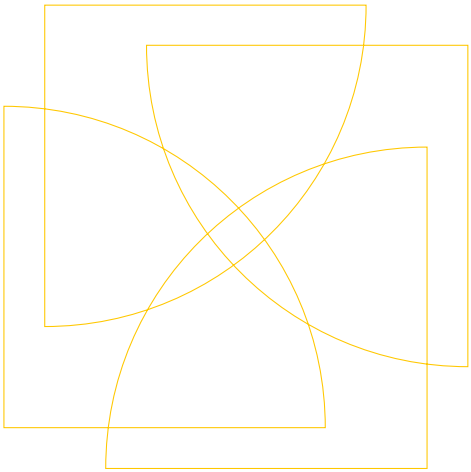
Concernant l'amélioration continue des processus, le groupe de recherche DORA recommande de contrôler les changements avec les 4 métriques suivantes :

- **Deployment Frequency** : Fréquence de déploiement réussi en production,
- **Lead Time** : Temps nécessaire entre la validation du code jusqu'à sa mise en production,
- **Mean Time to Recovery** : Temps moyen d'une panne jusqu'à la restauration complète du service,
- **Change Failure Rate** : Taux des modifications du code qui ont dégradé le système.

Les deux premières métriques mesurent l'agilité des équipes, et les deux dernières mesurent la stabilité des services développés.

La recommandation DORA est cohérente avec la philosophie DevOps qui cherche un équilibre entre l'agilité des équipes et la stabilité des services.

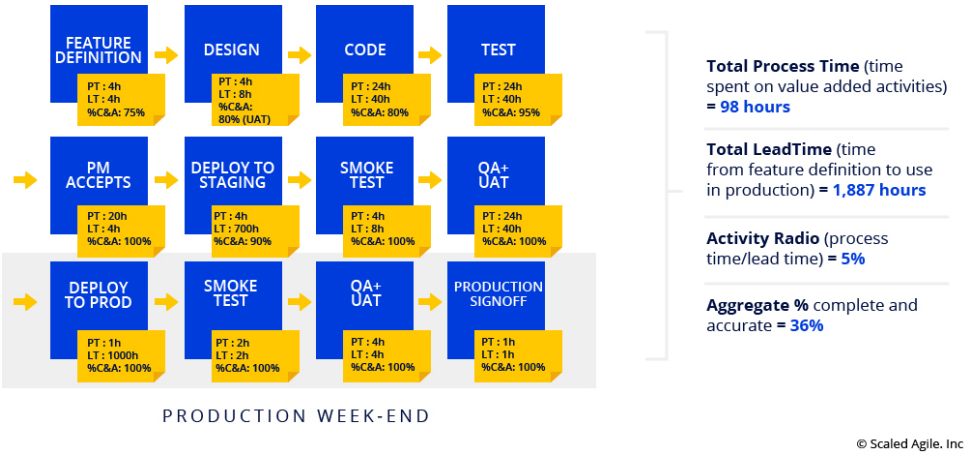
L'amélioration d'une métrique a souvent des effets positifs sur les autres métriques : une équipe qui augmente sa fréquence de déploiement (métrique Deployment Frequency) améliore aussi la qualité des déploiements (métrique Change Failure Rate).



2. VALUE STREAM MAPPING

Le Value stream mapping (VSM) est une pratique majeure dans la mise en place d'une démarche DevOps.

En effet, comme illustré par le diagramme suivant (extrait du site Web **SAFe**), le VSM permet de modéliser les étapes suivies de la définition d'une fonctionnalité jusqu'à sa mise en production.



Un diagramme de flux de valeur peut être annoté par des informations utiles à son analyse comme les durées des étapes ou les durées entre les étapes.

En tant qu'outil visuel, le VSM permet de partager la compréhension des flux de valeurs entre équipes. Cette compréhension permet d'améliorer la qualité des livrables entre équipes et réduire le gaspillage.

Le VSM permet de casser les silos : les équipes collaborent, indépendamment de la structure de l'entreprise, pour améliorer la qualité de leurs livrables (réduction de la dette technique, réduction du gaspillage…).

3. ARCHITECTURE MICROSERVICES

Les architectures microservices (vs architectures monolithiques) sont particulièrement adaptées aux contextes DevOps.

Ces architectures encouragent les équipes à déployer régulièrement de petits changements.

En effet, le développement d'applications microservices simplifie :

- La validation des changements obtenus grâce à un contrôle des impacts
- Le déploiement indépendant (et potentiellement le retour arrière) des services permis grâce à un couplage faible entre les services (rétrocompatibilité des APIs, architectures événementielles),
- La mise à l'échelle (scaling horizontal) des services grâce à une gestion dynamique des ressources (dimensionnement des ressources déterminé en fonction de la charge).

La mise en place d'architectures microservices engendre aussi de la complexité au niveau DevOps :

- Multiplication des processus d'automatisation,
- Complexité à interpréter les systèmes distribués,
- Complexité à gérer les environnements.

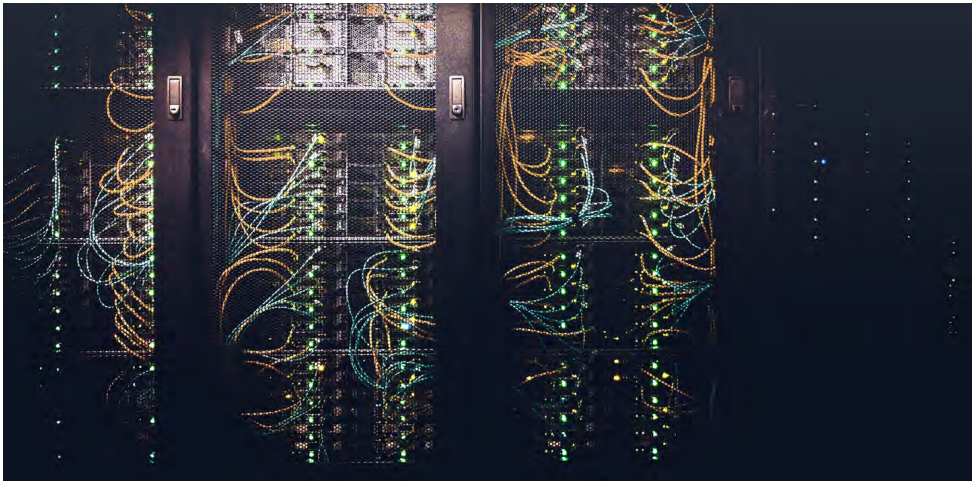
Dans la conception d'architecture microservices, pour éviter de gérer un trop grand nombre de services, de nombreux retours d'expérience recommandent de se baser sur une conception pilotée par domaine (DDD Domain Driven Design) et gérer le découpage des microservices par domaine (1 microservice = 1 domaine métier). La démarche DDD est décrite dans le livre Domain-driven design (Evans & Evans, 2004).

4. MOVE TO CLOUD

La migration vers le Cloud est aussi un facteur de succès pour les équipes DevOps.

L'utilisation de services managés permet aux exploitants de se consacrer sur les services métier. Par exemple, implémenter des solutions de sauvegarde de données ou de synchronisation de bases de données est une tâche compliquée et très consommatrice en temps. À l'exception du cas où ces éléments sont des facteurs concurrentiels pour l'entreprise, il est recommandé de déléguer ces tâches aux fournisseurs de services Cloud.

La panoplie des services Cloud permet aussi aux équipes de développement de choisir les services les plus adaptés. Ceci a un un impact très positif sur les temps de développement (notamment dans une démarche d'exploration où les équipes testent des services similaires pour trouver le plus adapté).



La migration vers le Cloud permet aussi une gestion dynamique des ressources : ce qui représente d'une part une économie et permet aussi aux équipes de mettre en place des environnements adaptés à des besoins différents (environnements de tests, tirs de performances, prototypes…).

05 — LES RISQUES

DANS CE CHAPITRE, NOUS ÉNUMÉRERONS CERTAINS RISQUES QUI PEUVENT ARRIVER DURANT LA MISE EN PLACE DE LA DÉMARCHE DEVOPS.

1. REFUS DU CHANGEMENT

Le refus de changement d'organisation (par exemple de suppression des silos) est un risque réel. Certaines équipes peuvent avoir peur du changement ou y voir une perte de leadership.

Pour remédier à ce risque, il est intéressant de réaliser la transformation par incréments :

- Intégrer en priorité les équipes les plus motivées,
- Expérimenter sur de petits périmètres (Minimal Viable Product ou MVP),
- Réaliser des retours d'expériences,
- Communiquer sur les réussites,
- Réaliser les actions de montées en compétences.

2. DÉGRADATION DES FLUX DE VALEUR

A chaque modification d'un flux de valeur, il est nécessaire de s'assurer que les modifications réalisées ne dégradent ni la qualité des produits ni la fréquence de livraison.

Modifier les processus existants sans suivi d'indicateurs est un vrai risque. En effet, sans indicateurs, il est impossible de s'assurer objectivement des apports positifs ou négatifs des changements.

En revanche, en mettant en place des indicateurs pertinents (comme les métriques DORA), il est possible de s'assurer des apports positifs des changements réalisés.

3. MANQUE DE CULTURE DEVOPS

La mise en place de pratiques DevOps comme "You build it, You run it" change le travail des équipes et nécessite des changements culturels, organisationnels, de processus et d'outillages.

Pour accompagner les équipes dans ces changements, il existe différents moyens : mise en place de formations, intégration d'experts dans les équipes, création de référentiels de bonnes pratiques...

4. MANQUE DE MOYENS

La mise en place d'une démarche DevOps nécessite des moyens matériels et des outils adaptés.

Concernant, par exemple, la validation des Merge Request (MR), une pratique intéressante est le déploiement des MR dans des environnements de test éphémères. Ceci implique d'allouer suffisamment de ressources pour gérer ces environnements éphémères (proportionnel au nombre de MR à valider).

Dans le cas où les équipes gèrent les ressources de manière dynamique, le coût supplémentaire est proportionnel à la durée de vie des environnements éphémères (coût faible si la durée de vie de ces environnements est faible).

Dans le cas où les équipes gèrent les ressources de manière statique (à éviter si possible), le coût est bien plus important : le coût est lié à la surallocation des ressources pour gérer ces environnements éphémères.

5. MANQUE D'ORGANISATION DEVOPS

Comme abordé dans la partie **Organisation des équipes**, la mise en place d'une démarche DevOps a des impacts sur l'organisation des équipes.

Lors de la mise en place d'une collaboration Dev et Ops, il est par exemple nécessaire de définir des objectifs communs et prévoir du temps pour la collaboration.

Les développeurs doivent avoir du temps pour prendre en compte les consignes des exploitants (exemple : observabilité, sécurité…).

Et les exploitants doivent avoir du temps pour la mise en place des services nécessaires (exemple : développement ou validation des composants d'Infrastructure As Code réutilisables…).

CONCLUSION

"UNE DESTINATION N'EST JAMAIS UN LIEU, MAIS UNE NOUVELLE FAÇON DE VOIR LES CHOSES."

Henry Miller, romancier et essayiste américain du 19ème siècle

POURQUOI METTRE EN PLACE CES PRATIQUES ?

La culture DevOps encourage à comprendre les flux de valeur de l'entreprise et à les optimiser globalement. Cette réflexion encourage à supprimer les silos entre les équipes, à améliorer la qualité des produits ainsi qu'à supprimer le gaspillage.

La mise en place des boucles de rétroaction permet d'améliorer en continu les flux de valeurs, d'accepter les erreurs, d'encourager le travail collaboratif et le partage de connaissances.

L'amélioration continue de la qualité de code, de l'architecture, de l'utilisation des services utilisés permet une généralisation des bonnes pratiques et donc une réduction importante de la dette technique.

Au-delà du besoin initial d'améliorer la qualité des produits, cette démarche permet d'améliorer l'empathie, la confiance et la communication entre les différents acteurs d'un projet.

BIBLIOGRAPHIE

Behr, K., Kim, G., & Spafford, G. (2018). The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win. IT Revolution Press.

Biographie Patrick Debois. Personal website of Patrick Debois – JEDI - Just Enough Documented Information Blog from https://www.jedi.be/bio/

Cohn, M. (2010). Succeeding with Agile: Software Development Using Scrum. Addison-Wesley.

Debois, P., Humble, J., Kim, G., & Willis, J. (2016). The DevOps Handbook: How to Create World-class Agility, Reliability, and Security in Technology Organizations. IT Revolution Press.

DevOps Topologies from https://web.DevOpstopologies.com/

DORA. DevOps from https://www.DevOps-research.com/research.html

Evans, E. J., & Evans, E. (2004). Domain-driven design. Addison-Wesley.

Explore DORA's research program. DevOps from https://www.DevOps-research.com/research.html#reports

Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures. ICS UCI from https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Kim, G., Willis, J., Debois, P., & Humble, J. (2016). The DevOps Handbook: How to Create World-class Agility, Reliability, and Security in Technology Organizations. IT Revolution Press.

Manifeste Agile. (2001). Manifeste pour le développement Agile de logiciels from https://agilemanifesto.org/iso/fr/manifesto.html

Monitoring de systèmes distribués. https://sre.google/sre-book/monitoring-distributed-systems

Outil d'évaluation DORA. DevOps
from https://www.DevOps-research.com/quickcheck.html

Outil d'évaluation SAFe. AgilityHealth
from https://agilityhealthradar.com/safe-DevOps-assessment

Royce, W. (1970). Managing the Development of Large Software Systems. Proceedings of IEEE.

SAFe 5 for Lean Enterprises from https://www.scaledagileframework.com/

Site Reliability Engineering. Site Reliability Engineering: Google from https://sre.google/

Skelton, M., & Pais, M. (2019). Team Topologies: Organizing Business and Technology

Teams for Fast Flow. IT Revolution.

The Three Ways: The Principles Underpinning DevOps. IT

Revolution
from https://itrevolution.com/the-three-ways-principles-underpinning-DevOps/

Willis, J. (2012, May 1). Culture DevOps. IT Revolution
from https://itrevolution.com/DevOps-culture-part-1/

CRÉDITS PHOTOS

François Delauney
altamirafilm.co
unsplash.com
pixabay.com
agilityhealthradar.com
devops-research.com
scaledagileframework.com

ippon

CONTACTEZ-NOUS !

fr.ippon.tech
blog.ippon.fr
medium.com/ippon

contact@ippon.fr
+33 1 46 12 48 48
@ippontech