# La méthodologie DevOps

lundi 5 septembre 2022 10:25

Lien du cours : https://openclassrooms.com/fr/courses/6093671-decouvrez-la-methodologie-devops/6183233-decouvrez-les-origines-de-la-methodologie-devops

Les principaux avantages du DevOps sont :

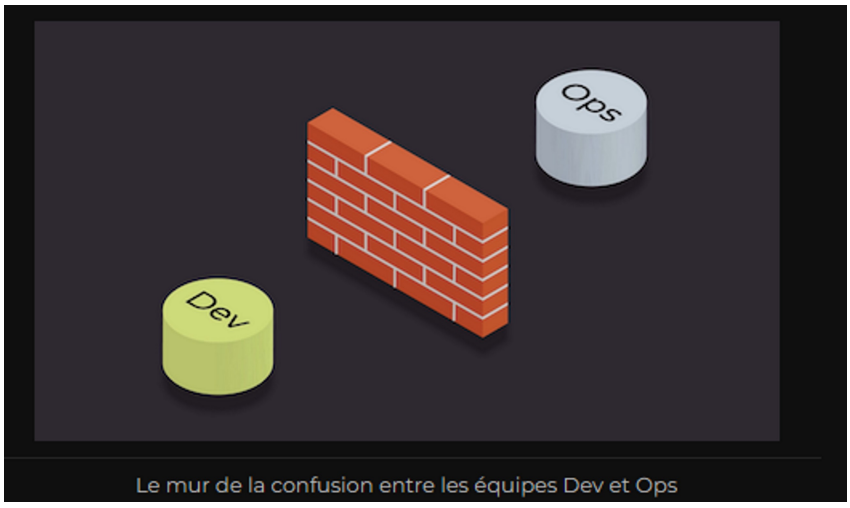
l'accélération des déploiements applicatifs

la réduction du **Time-to-Market**

## D'où vient le DevOps

*Un peu d'historique : le terme DevOps a été utilisé pour la première fois par Patrick Debois et Andrew Shafer dans leur conférence Agile infrastructure, lors de la 2008 Agile Toronto conference.*

Il faut savoir qu'**à cette époque, l'informatique d'entreprise a "siloté" les aspects dev et ops des applications**, en plaçant les responsabilités respectives dans des équipes séparées. Il s'est alors créé ce qu'on appelle le **mur de la confusion**. Ce mur est apparu car les deux équipes ont des objectifs respectifs antagonistes.



Le mur de la confusion entre les équipes Dev et Ops

La première, **l'équipe de développement**, a pour objectif principal de faire **évoluer l'application,** en ajoutant de nouvelles fonctionnalités ou en corrigeant des bugs, le plus rapidement possible et cela, dans un délai et un coût moindres, souvent au détriment de la qualité quand les contraintes de planning arrivent.

La seconde équipe, **l'équipe des ops,** a comme objectif de **maintenir l'application en conditions opérationnelles,** en garantissant la stabilité de l'application et la qualité de cette dernière, en sacrifiant souvent le coût et le temps nécessaires au déploiement d'une nouvelle version.

Les équipes de développement sont alors récompensées par le nombre de bugs qu'elles corrigent, ainsi que le nombre de nouvelles fonctionnalités livrées à chaque version. De leur côté, les équipes ops sont récompensées par le nombre de bugs critiques en production, ainsi que la disponibilité globale sur l'année de l'application.

*Le DevOps est un ensemble de pratiques qui met l'emphase sur l'automatisation des processus entre les équipes de développement, et les équipes en charge du maintien en conditions opérationnelles de l'application développée.*

La liste des avantages apportés par le DevOps est donc longue :

- gain de confiance des équipes entre elles
- accélération des livraisons et des déploiements
- résolution des tickets plus rapide
- gestion plus efficace des tâches non planifiées…

## Identifiez les caractéristiques de la méthodologie DevOps

Le DevOps est souvent associé à l'acronyme **CALMS** désignant les **5 piliers de cette méthodologie**, qui sont **:**

- **Culture**
- **Automatisation**
- **Lean**
- **Mesure**
- **Share**

### Culture

le DevOps est avant tout une **histoire de culture et de collaboration entre les développeurs et les opérationnels.** La culture compte pour plus de la moitié du DevOps. **Tous les outils mis en place ne servent à rien, s'il n'y a pas la volonté des développeurs et des opérations de travailler ensemble.**

Le DevOps résout, en premier lieu, **des problèmes humains, des problèmes de communication et des problèmes de responsabilités entre équipes ->** Dans ce sens, le

DevOps se rapproche du **modèle AGILE** tout en incluant **toute personne dont les aptitudes sont requises afin de délivrer un produit de qualité. L**e but étant de créer une **équipe orientée PRODUIT.**

*Un des moyens pour atteindre ce genre d'équipe est d'inclure par exemple **les opérationnels aux Daily Meetings**, les réunions quotidiennes et rapides de l'équipe, afin que ceux-ci puissent exposer leur problématiques, et que ces problématiques soient incluses dans les différents sprints.*

Un bon moyen de savoir si la culture DevOps est bien implémentée est de se poser des questions comme :

- Est-ce que les problèmes sont résolus par **TOUTE l'équipe produit** ?
- Est-ce que tous les membres de l'équipe partagent une **vision commune** ?
- Est-ce que les post-mortem sont dirigés vers la **résolution des problèmes** et non vers la détermination d'un responsable de l'erreur ?

## Automatisation

*Mettre en place un process de communication entre les personnes ne suffit généralement pas. C'est pour cela que les entreprises sont amenées à mettre en place un certain nombre d'**outils afin de favoriser et de fluidifier les déploiements**. Comme le principe du DevOps est de livrer plus souvent des produits de qualité, l'**automatisation** est un bon moyen d'arriver à ce résultat.*

Avec la méthodologie DevOps, Les applications sont **déployées plus souvent**. Cela amène à être rassuré sur la procédure de déploiement. Généralement, les entreprises sont amenées à ne déployer que quelques fois dans l'année, alors que des entreprises comme Google, Amazon et Facebook, qui sont souvent prises en exemple sur l'automatisation, peuvent déployer jusqu'à **17 fois par minute** en production.

Les configurations sont gérées **automatiquement par les outils**. **Dans les entreprises dites traditionnelles sans culture DevOps**, les développeurs rédigent souvent un manuel de déploiement sous Word, à destination des équipes d'exploitation, expliquant comment déployer les logiciels et les configurer. **Il est alors facile de se tromper lors du déploiement d'une application**, en ayant oublié une étape ou l'exécution d'un script, et de se retrouver avec un système fonctionnant mal ou ne fonctionnant pas du tout.

Tous ces problèmes conduisent à la situation citée précédemment, c'est-à-dire à ne **déployer que quelques fois dans l'année en HNO**, de peur de ne plus voir le système fonctionner.

**Avec la méthodologie DevOps, la création d'environnements se fait à la volée**. En utilisant des **outils d'Infrastructure-as-Code**, il est possible de créer **à la demande des environnements pour tester et déployer les applications**, ce qui réduit notamment les temps d'attente liés à la fourniture d'environnements hors production.

Comme les tests et l'infrastructure sont versionnés au même niveau que le code, tout changement de l'une de ces trois briques (tests, infrastructure et code) **relance automatiquement toute la chaîne dans un processus complet**. Dès lors, il est facile de rejouer des campagnes de tests de non-régression complète en quelques minutes, ou en quelques heures si la chaîne de bout en bout couvre une grande partie des fonctionnalités.

## Lean

**Le Lean**, ou Lean Management, vient de l'industrie, et plus particulièrement de Toyota dans les années 1990. **Lean sert à qualifier une gestion des ressources sans gaspillage**.

Les types de gaspillage sont au nombre de 7 :

- **surproduction**
- **attente**
- **transports**
- **étapes inutiles**
- **stocks**
- **mouvements inutiles**
- **corrections / retouches**

Dans le contexte du DevOps, **Lean va alors s'intéresser à délivrer de la valeur ajoutée au client final** (dans le cadre d'une application grand public, le public), **tout en minimisant les processus longs, coûteux, sans valeur ajoutée.**

**/!\ Mais attention à ne pas rogner sur la qualité de l'application, en sautant des étapes de l'intégration continue, afin de faire des économies sur les environnements /!**

Le **Value Stream Map** (ou **VSM**) est un outil emprunté du Lean, qui permet de **connaître les processus prenant du temps et n'apportant pas de valeur ajoutée**. Avec le VSM, c**haque étape d'un processus est décrite, qualifiée et quantifiée en performance**, sous la forme d'une chaîne de valeur (temps de traitement de chaque

étape, délai entre deux étapes, taux de transformation, etc.).

Il est alors important dans le DevOps de ne pas reprocher à quelqu'un le problème de déploiement, mais plutôt d'**encourager l'équipe à parler librement lorsqu'une erreur arrive**, afin d'identifier celle-ci, ses causes et **mettre en place un processus garantissant la non-reproduction de cette erreur**.

## Mesure

*Comme toute transformation d'entreprise, il est nécessaire d'avoir des **indicateurs de performance clés** (KPI ou Key Performance Indicator) afin de savoir si les efforts de transformation et d'amélioration continue transforment quelque chose.*

il existe une multitude d'outils sur le marché pour **juger de la performance d'une application**, ou d'une transformation. Les différents indicateurs utilisées peuvent être :

- Combien de temps la nouvelle fonctionnalité a pris pour p**asser du développement à la production** ?
- Combien de fois **un bug récurrent** apparaît ?
- Combien de personnes **utilisent le produit en temps réel** ?
- Combien **d'utilisateurs** a-t-on **gagnés ou perdus** en une semaine ?

## Sharing

*Le fameux **mur de la confusion** est largement dû à un **manque de savoir commun**. Mettre en place une politique de **responsabilité et de succès partagés** est déjà un premier pas vers le comblement de ce fossé entre les deux mondes. Le DevOps met en exergue l**e fait que la personne qui code l'application doit aussi la livrer et la maintenir en condition opérationnelle**.*

Les opérationnels et les développeurs **doivent travailler de concert** sur le cycle de vie complet de l'application.

les développeurs corrigent les problèmes des utilisateurs finaux, mais aussi **participent à la résolution des problèmes de production**. Ils apprennent également c**omment leur application est utilisée au quotidien** par les utilisateurs finaux. En se rendant disponibles auprès des équipes des opérations, les développeurs **construisent une relation de confiance** avec eux, ainsi qu'un respect mutuel.

## Résumé des caractéristiques de la méthodologie DevOps

- **Culture** : le DevOps est une affaire de **culture d'entreprise entre les dev et les ops**, avant d'être un ensemble d'outils ou même de pratiques métier. **Il faut que les dev et les ops travaillent ensemble**, aient une **vision commune**, soient tournés vers les mêmes objectifs.

- **Automatisation** : **tout ce qui peut être automatisé doit l'être !** Ceci se fait par des déploiements **programmés et fréquents**, des environnements **créés à la demande** et des **tests automatisés**.

- **Lean** : dans la méthodologie DevOps, concentrez-vous sur la création de valeur et limitez le **gaspillage** de ressources. Pour cela, utilisez la **Value Stream Map** pour maîtriser votre chaîne de valeur.

- **Mesure** : Afin de tout automatiser et tout optimiser, il faut mesurer tout ce que vous faites. Pour ça, mettez en place les **KPI** nécessaires pour superviser vos déploiements !

- **Sharing** : Dans le DevOps, les équipes de dev et d'ops **partagent des moments et les responsabilités**. C'est le cœur de l'objectif du DevOps !

# Mettez en place le DevOps en évitant les pièges

*Dans ce chapitre, je vous propose de découvrir plusieurs de ces patterns et anti-patterns, afin que vous puissiez les connaître avant d'implémenter la méthodologie DevOps dans votre entreprise, mais aussi savoir identifier les pièges courants dans lesquels ne pas tomber.*

*Ici, un pattern désigne une bonne pratique pour implémenter une méthode. Un anti-pattern est en quelque sorte un contre-exemple, un piège à éviter !*

Le principal but du DevOps dans une entreprise est d'abord d'**améliorer la livraison de valeur ajoutée pour le métier et l'utilisateur final**. Ce n'est pas nécessaire, ou pas forcément prioritaire de réduire les coûts, d'augmenter l'automatisation des livraisons, ou de tout faire depuis de la gestion de configuration.

Cela veut donc dire que plusieurs organisations auront potentiellement des **besoins différents pour structurer les équipes de développeurs et les équipes de production**, afin de permettre une collaboration **efficace** entre ces **deux métiers**.

**La structure d'une organisation DevOps va donc dépendre de plusieurs facteurs**,

comme :

- L'**ensemble des produits** de l'organisation. Une entreprise travaillant sur **moins de produits aura plus de facilités à la collaboration**, car il y aura **moins de silos naturels**, comme le prévoit la **loi de Conway**

- La **loi de Conway** est une observation sociologique qui stipule que "l**es organisations qui définissent des systèmes [...] sont contraintes de les produire sous des designs qui sont des copies de la structure de communication de leur organisation**." Ceci veut dire que la **structure des organisations** va automatiquement **contraindre les systèmes que cette organisation designe**, et leur structure.

- La portée, la force et l'efficacité du l**eadership technique**, et si les **développeurs** et la **production** ont un **objectif commun**, comme vu dans le chapitre précédent

- L'alignement des pratiques de l'**équipe Opérations informatiques** avec la chaîne de valeur. Si le département des opérations informatiques est organisé autour de la **configuration de serveurs** et de **hardware** et non autour des **fonctionnalités opérationnelles**, un **gros travail d'évolution sera à faire** sur ce point-là. Or, **toutes les entreprises n'ont pas la capacité ou la volonté d'évoluer** sur ce point

- L'**envie ou la capacité** d'une organisation à **changer son département des opérations informatiques**, de "**racking hardware**" et de "**configuration de serveurs**" à un **alignement réel** avec la chaîne de valeur, et à ce que **les fonctionnalités opérationnelles soient prises au sérieux par les équipes logicielles**

- la **capacité** ou les **compétences présentes** dans l'organisation qui sont nécessaires pour **prendre l'initiative sur les questions opérationnelles**.

## Exemple de patterns et d'anti-patterns


Le premier anti-pattern que l'on retrouve souvent en entreprise, et que le DevOps cherche à éliminer, est l**a séparation entre les équipes de développement et les équipes d'ops**. Ces équipes sont séparées par le fameux "**mur de la confusion**".

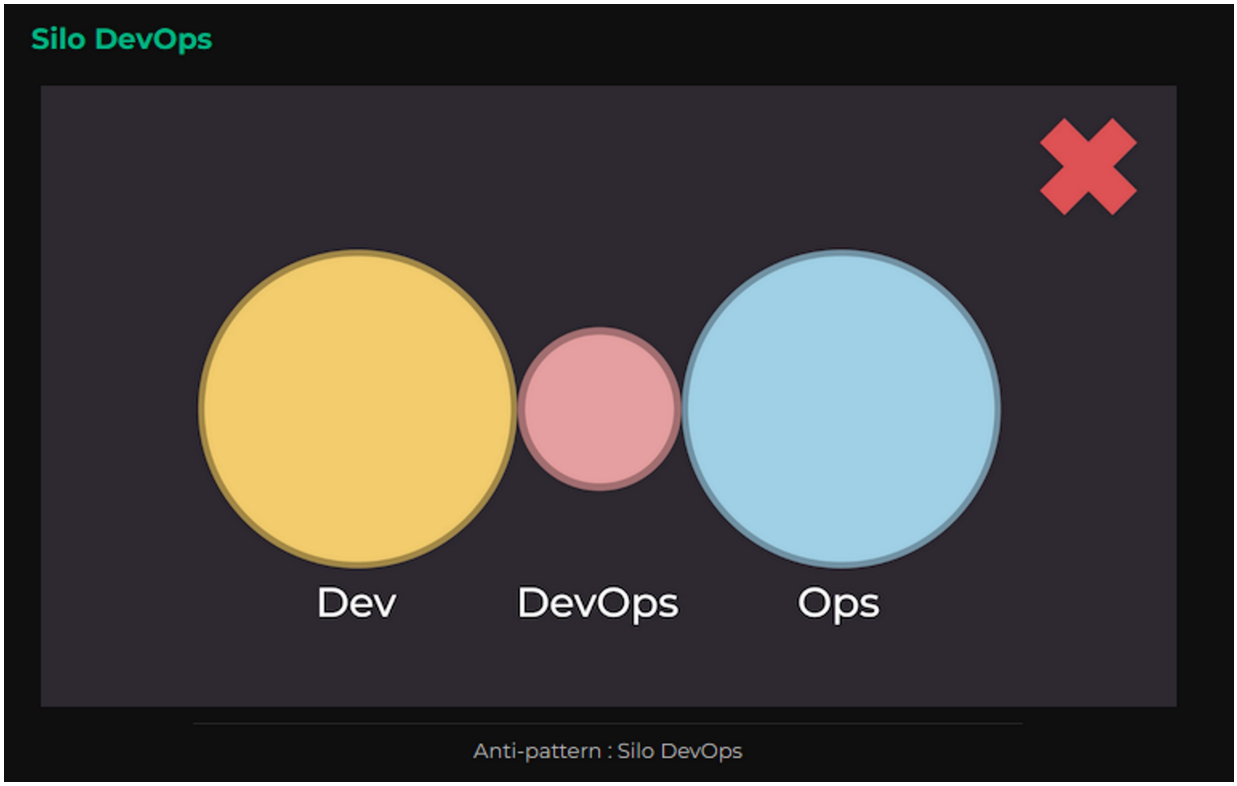
D'un côté, nous avons des **développeurs** dont le métier est de **développer des user stories, de faire évoluer l'application, d'implémenter de nouvelles features et de corriger les bugs**. Et de l'autre, nous avons des **ops** dont le métier est de **maintenir en condition opérationnelle l'application, de réparer les serveurs qui ne fonctionnent pas, d'appliquer des patchs de sécurité et de faire évoluer les produits.**

Ces deux équipes ont donc **deux vitesses différentes** :

- Les développeurs voulant **aller vite** et avoir le **maximum de changements**

- Les opérationnels voulant garantir la **stabilité de l'application**.

**Ces deux mondes ne se comprennent pas** et finissent par **ralentir la cadence de livraison des applications en production**. Le DevOps est une approche visant à **casser ces silos** et **fluidifier les communications** entre personnes d'une part, et **mettre en place des outils d'automatisation de livraison** de l'autre.




Une des bonnes idées qu'une entreprise peut avoir, afin de pallier la problématique citée précédemment, est de mettre en place une **équipe DevOps en charge de l'implémentation des processus de livraison et de déploiement continus**. Cette équipe est généralement **transverse** à l'entreprise et est **sollicitée par toutes les applications**, afin de pouvoir mettre en place des **outils d'automatisation**.

Malheureusement, **la création de cette équipe transverse génère un troisième silo**, car elle va être confrontée à :

- des développeurs la sollicitant, **mais ne comprenant pas les fondamentaux du DevOps**
- des ops qui **veulent garder la main sur la mise en production, garder leurs outils**, et ne voient généralement pas d'un bon œil la création de cette équipe.

De plus, les développeurs et les ops consommateurs de cette équipe ne comprenaient pas les fondamentaux du DevOps, et se reposaient alors quasiment exclusivement sur cette équipe transverse. Nous avions alors créé un **troisième silo**, alors que le DevOps est **censé les casser et fluidifier la communication**.

## Patterns NoOps

Avec l'arrivée du **cloud computing**, et plus récemment du **serverless**, de plus en plus d'articles de blogs et de conférences parlent du **mouvement NoOps, ou la non-nécessité des ops dans un schéma traditionnel.** Effectivement, ce mouvement part du principe que la **plateforme Cloud, et donc le cloud provider, prend en charge toutes les opérations pour le maintien en condition opérationnelle de l'application**

(sauvegarde, plan de reprise d'activité, scalabilité, résilience, etc.).

**le métier de développeur et le métier d'ops sont bien deux métiers totalement différents.** Les développeurs ne mesurent pas la **complexité d'une application**, et savent **rarement** comment rétablir une infrastructure qui ne fonctionne pas.

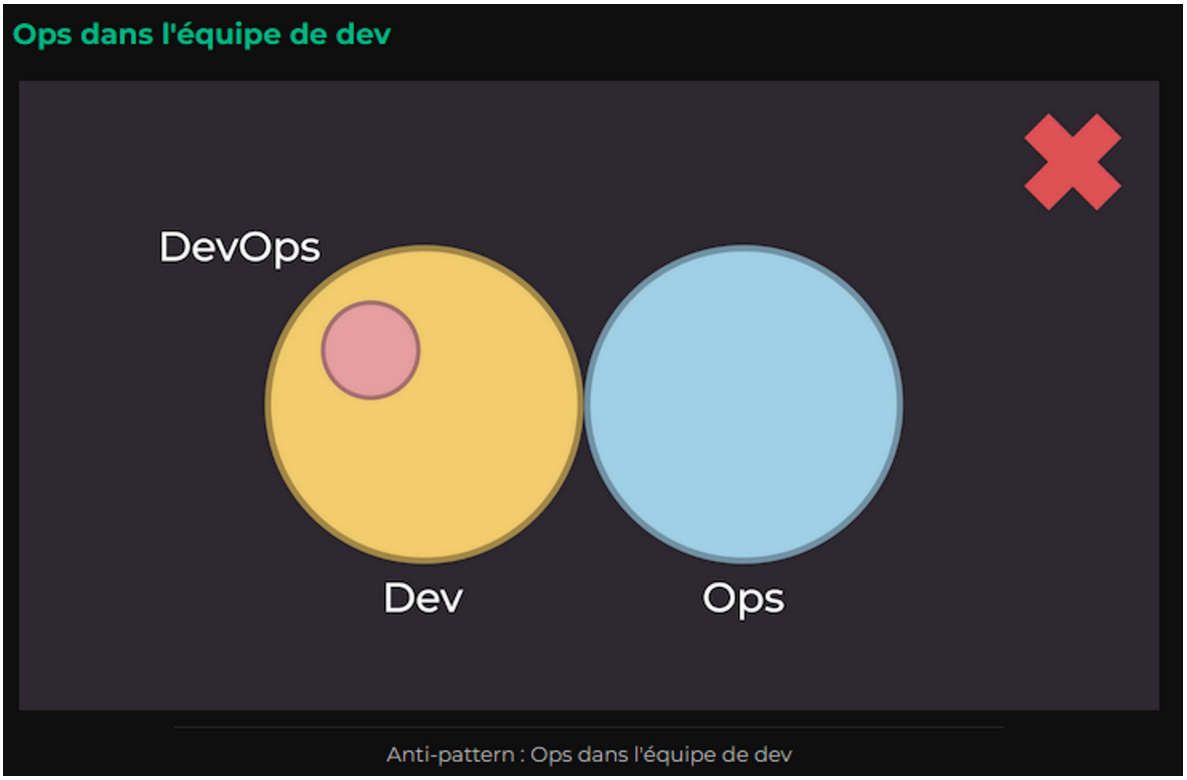
Une équipe constituée uniquement de développeurs finira **souvent** par avoir **besoin d'un ops**, au fur et à mesure de la montée en complexité de l'application. Dans ce cas-là, même si les développeurs gardent la main sur ce qu'ils ont mis en place, ils feront alors appel à une équipe **DevOps transverse**, ou à l'**équipe Cloud** d'une entreprise.

## Patterns Equipe Transverse Outils

La création d'une équipe transverse **uniquement basée sur les outils** est aussi une mauvaise idée lors de l'implémentation du DevOps. Comme nous avons pu le voir plus haut dans le cours, cette équipe va créer un troisième silo, là où le DevOps cherche à les faire disparaître.

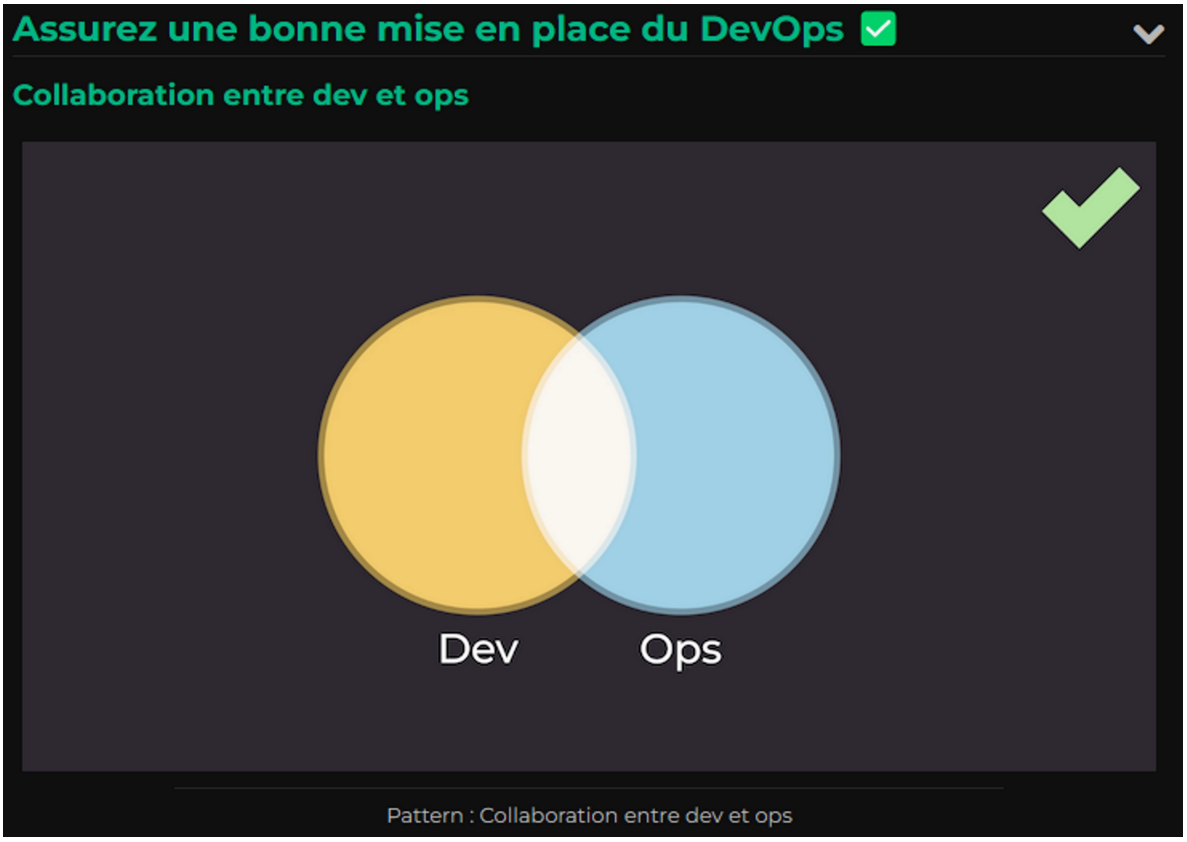
De plus, cette équipe est uniquement axée sur **l'implémentation des outils** afin de pouvoir mettre en place des **pipelines de déploiement de livraison continue**, de la **gestion de configuration**, de la **gestion d'environnement**, etc… Cette équipe n'a pas pour vocation de **changer la culture de l'entreprise**, et les deux équipes Dev et Ops continuent à travailler **sans changer leur process**, et en se r**eposant en grande partie sur cette équipe Outils transverse**, pour **livrer l'application**.

Certes, cette équipe sera bénéfique à très court terme, car e**lle permet aux autres équipes de ne plus avoir à penser à tout ce qui ne touche pas à l'application**, mais son **impact est limité** car elle **ne changera pas les process**, qui parfois peuvent être **très lourds**.




L'organisation ne veut pas conserver une équipe d'exploitation distincte, de sorte que les équipes de développement assument la responsabilité de l'infrastructure, de la gestion des environnements, de la surveillance, etc. Toutefois, le fait de le faire en fonction du projet ou du produit signifie que ces éléments sont soumis à des **contraintes de ressources** et à des **redéfinitions des priorités**, qui conduisent à des approches partielles et à des solutions à demi perçues.

Dans cet anti-pattern, l'organisation montre un manque d'appréciation de l'importance et des compétences requises pour des opérations informatiques efficaces. En particulier, la valeur des opérations est diminuée parce qu'elle est traitée comme une gêne pour les développeurs (car les opérations sont gérées par un seul responsable d'équipe avec d'autres priorités).




La clé pour la réussite de l'implémentation du DevOps au sein d'une entreprise est la **communication entre les développeurs et les ops**.

Tout ceci doit évidemment passer par un **changement culturel de l'entreprise important** et soutenu par un sponsor fort, comme le DSI ou le directeur de production, par exemple.

## Patterns Dev et équipe Cloud

Pour les organisations disposant d'un département IT assez traditionnel qui ne peut pas ou ne veut pas changer (assez) rapidement, et pour les organisations qui exécutent toutes leurs applications dans le cloud public (Amazon EC2, Azure, etc.), il est probablement utile de traiter les **opérationnels comme une équipe qui fournit simplement l'infrastructure** sur laquelle les applications sont déployées et fonctionnent.

Une équipe (peut-être une équipe virtuelle) au sein des développeurs **agit alors comme une source d'expertise sur les fonctionnalités opérationnelles, les métriques, la surveillance, le provisionnement des serveurs, etc.**, et assure probablement la majeure partie de la **communication avec l'équipe Cloud**. Cette équipe est toujours une **équipe de développement** ; cependant, elle suit alors des pratiques standard comme le **TDD, le CI, le développement itératif, le coaching, etc.**

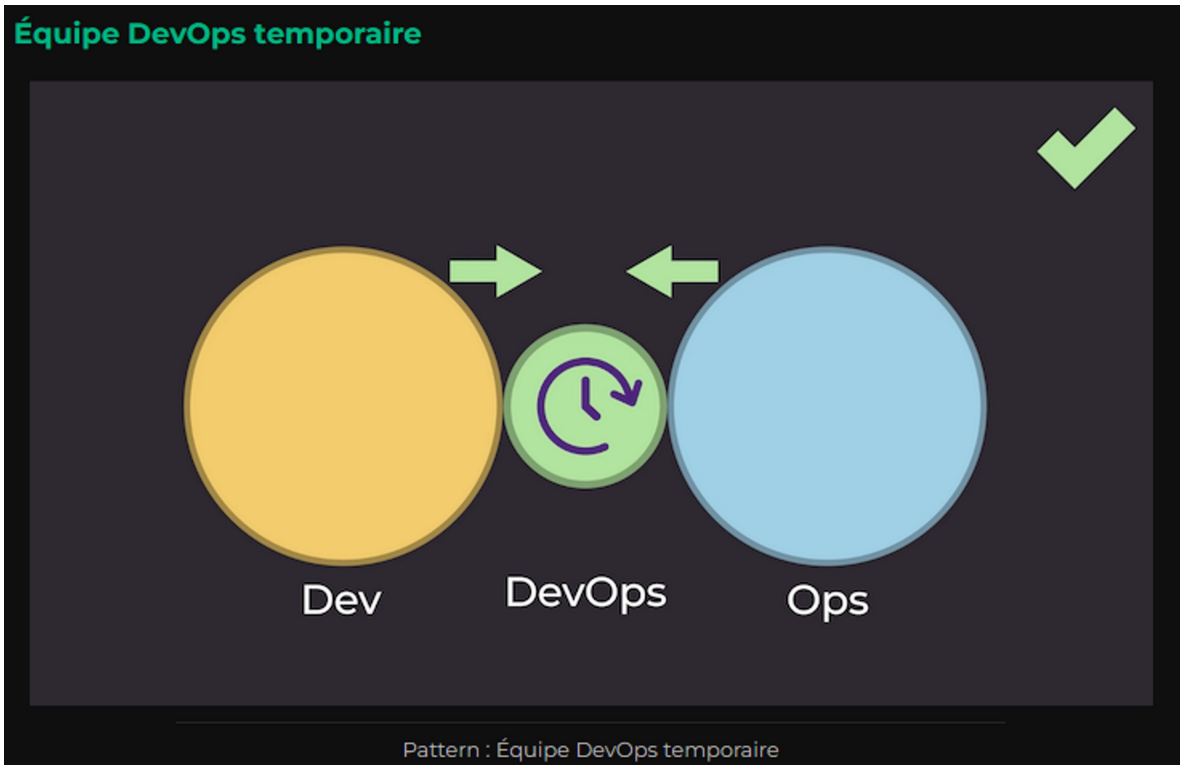
La topologie Cloud offre une certaine efficacité potentielle (perdant de la collaboration

directe avec les équipes d'exploitation) pour une **mise en œuvre plus facile**, peut-être en tirant de la valeur plus rapidement qu'en essayant le premier pattern **Collaboration entre dev et ops**, qui pourrait être tenté à une date ultérieure.

## Patterns Service externe DevOps

Certaines organisations, en particulier les plus petites, peuvent ne pas avoir les finances, l'expérience ou le personnel nécessaires pour prendre la direction des aspects opérationnels de l'application qu'elles produisent. L'équipe de développement peut alors faire appel à un **fournisseur de services** comme Google, Amazon ou Azure, pour les aider à créer des environnements de test, et à automatiser leur infrastructure et leur surveillance. Ce fournisseur externe pourra également les **conseiller** sur les types de fonctionnalités opérationnelles à mettre en œuvre pendant les cycles de développement logiciel.

Ce que l'on pourrait appeler **DevOps-as-a-Service** pourrait être un moyen utile et pragmatique pour une petite organisation ou une **petite équipe** d'en apprendre davantage sur l'automatisation, la surveillance et la gestion de la configuration, puis peut-être de passer à un modèle de type "Dev et équipe Cloud" ou même de type "Collaboration entre dev et ops", à mesure que le personnel s'accroît et se concentre sur les opérations.




L'équipe DevOps temporaire ressemble beaucoup à l'anti-pattern **Silo DevOps**, mais son intention et sa longévité sont très différentes. Cette équipe temporaire a pour mission de rapprocher les développeurs et les opérations, idéalement vers un modèle de type

**Collaboration entre dev et ops**, et éventuellement de se rendre obsolète.

Les membres de l'équipe temporaire parleront de problématiques propres à chaque équipe :

- en introduisant des idées comme les **stand-up meetings et le Kanban** pour les équipes Ops

- en pensant aux détails comme les l**oad-balancers, la gestion des cartes réseaux, et le SSL** pour les équipes Dev.

Si suffisamment de personnes commencent à voir l'intérêt de réunir dev et ops, alors l'équipe temporaire a une réelle chance d'atteindre son objectif. La **responsabilité** à long terme des déploiements et des diagnostics de production ne devrait donc pas être confiée à l'équipe temporaire, sinon elle risque de devenir un **silo DevOps**.

## Patterns Equipe d'évangélistes DevOps

Un des patterns qui pour moi fonctionne le mieux en entreprise, car je l'ai expérimenté et implémenté chez Contoso, est une équipe d'**évangélistes DevOp**s ou **coachs DevOps**. Mon rôle chez Contoso était un rôle de coach DevOps. J'aidais les équipes Devs et Ops des applications à implémenter le DevOps, mais aussi l'intégration et la livraison continues.

## Patterns Collaboration axée sur les conteneurs

Une des promesses des conteneurs (notamment avec Docker) est de pouvoir s'**exécuter de la même manière sur n'importe quel environnement exécutant un démon** pouvant exécuter ce conteneur. Effectivement, l'**application est encapsulée dans un conteneur avec tout ce qui est nécessaire pour la faire tourner (librairies, framework, middleware, etc.)**.

- D'un coté, les développeurs ne se chargent que de **créer l'image**, afin que l'application puisse tourner correctement.

- De l'autre, les ops **opèrent cette image** avec toutes les contraintes liées au maintien en condition opérationnelle.

De fait, le conteneur devient alors le l**ivrable universel** servant de passerelle entre les développeurs et les ops. Avec une **forte automatisation et beaucoup d'outillage**, ce pattern permet une **fluidification de livraison d'applications**, les

exigences de déploiement et de fonctionnement étant encapsulées dans le conteneur.

Malheureusement, si les développeurs commencent à ignorer les recommandations des ops pour le maintien en condition opérationnelle de l'application (par exemple le format d'envoi des logs), c**e pattern devient alors un anti-pattern similaire aux silo Dev et silo Ops**.

## Résumé

**les mauvaises pratiques** ❌ :

- **silo Dev et silo Ops** : Aucune méthodologie DevOps ici, modèle classique
- **silo DevOps** : LE piège, créer un troisième silo en essayant de briser les deux autres
- **NoOps** : Eh non ! Le rôle des ops est encore utile, même s'il évolue
- **équipe transverse outil** : Vous passez à côté de plein d'autres aspects du DevOps et... vous créez un troisième silo !
- **ops dans l'équipe de dev** : C'est montrer un manque d'appréciation du travail des ops !

**les bonnes pratiques** ✅ :

- **collaboration entre dev et ops** : La collaboration est la clé pour un bon fonctionnement de l'équipe de production en mode DevOps
- **dev et équipe Cloud** : Une équipe qui s'occupe simplement de fournir l'infrastructure de déploiement. Ceci peut être une étape intermédiaire vers une implémentation plus complète du DevOps
- **service externe** : Souvent une solution adaptée aux petites équipes, faire appel à un fournisseur externe d'expertise DevOps peut s'avérer une bonne idée
- **équipe DevOps temporaire** : Une équipe DevOps qui n'est présente que le temps de mettre en place les démarches DevOps et former les équipes Dev et Ops aux bonnes pratiques
- **évangélistes DevOps** : Proche du pattern précédent, c'est une équipe qui s'occupera uniquement de former les équipes au DevOps ;
- **collaboration axée sur les conteneurs** : les conteneurs permettent de fluidifier le

travail entre le développement et la mise en production.

## Présentation du métier Site Reliability Engineer - SRE

*Grâce au DevOps, un nouveau métier est apparu au début des années 2000 : le SRE, ou **Site Reliability Engineer**. Relativement peu d'entreprises l'implémentent aujourd'hui, mais il est toujours intéressant de connaître ce métier, ne serait-ce que pour mettre un nom sur l'implémentation concrète du DevOps dans une entreprise.*

Avant de présenter plus en détail ce métier, il est nécessaire de définir présenter certains termes :

- **Service Level Indicator** : Le **Service Level Indicator**, ou **SLI,** est une **mesure du niveau de service** rendu pour chaque client.

- **Service Level Objective** : Le **Service Level Objective**, ou **SLO**, est un **objectif que doit atteindre le SLI** défini par l'équipe. Le SLO est défini entre le **fournisseur du service et le consommateur**.

- **Service Level Agreement** : **Le Service Level Agreement**, ou **SLA**, est un contrat passé entre un f**ournisseur de service** et le **consommateur** de ce service. Le non-respect de ce SLA aboutit souvent à des **pénalités financières**.

### Le rôle du SRE

Le SRE a deux rôles :

- il passe 50 % de son temps à faire des tâches "**ops**", comme de l'**astreinte**, **corriger des problèmes en production** ou faire des **interventions manuelles**

- il passe les autres 50 % de son temps à **automatiser** tout ou partie de ses tâches, et à **développer de nouvelles fonctionnalités**.

### Les piliers du SRE

Le métier du SRE repose sur **5 piliers** issus des piliers **CALMS :**

- **Réduire les silos organisationnels (Share) :**

    - Le SRE partage la possession des applications avec les développeurs, pour créer des responsabilités partagées
    - Le SRE **utilise les mêmes outils** que les développeurs, et vice versa.

- **Accepter les pannes comme normales (Culture)**
    - Le SRE **accepte le risque**
    - Le SRE **quantifie les pannes et la disponibilité d'une manière normée**, en utilisant le **SLI** et le **SLO**
    - Le SRE mandate des post-mortem irréprochables.

- **Implémenter des changements graduels (Lean)**
    - Le SRE encourage les développeurs et les Product Owners à avancer rapidement en réduisant les coûts de panne.

- **S'appuyer sur des outils et de l'automatisation (Automatisation)**
    - Le SRE dispose d'une charte pour automatiser les tâches subalternes (appelées "labeurs"), à l'extérieur.

- **Tout mesurer (Mesure)**
    - Le SRE **définit des moyens normatifs** pour **mesurer les valeurs**
    - Le SRE croit fondamentalement que **le fonctionnement des systèmes est un problème logiciel.**

## Les 4 signaux dorés

Le SRE définit **quatre signaux** principaux à observer constamment. Ces quatre signaux sont appelés les **quatre signaux dorés** : **latence, trafic, erreurs** et **saturation**.

- **Latence** : La **latence** est le **temps** qu'il faut pour **envoyer une demande et recevoir une réponse**. La latence est généralement **mesurée du côté du serveur**, mais elle peut aussi être **mesurée du côté du client pour tenir compte des différences de**

**vitesse du réseau**. L'équipe d'exploitation a le plus de maîtrise sur la latence côté serveur, mais la latence côté client sera plus pertinente pour les clients finaux.

Le seuil cible que vous choisissez peut varier en fonction du type d'application. Un système automatisé, comme une API ou un serveur, peut nécessiter des temps de réponse beaucoup plus rapides qu'un humain sur un téléphone mobile. Vous devez également **suivre séparément la latence des requêtes réussies et des requêtes échouées**, car les requêtes échouées échouent souvent rapidement, sans traitement supplémentaire.

- **Trafic** : Le **trafic** est une mesure du **nombre de demandes qui circulent sur le réseau.** Il peut s'agir de requêtes HTTP vers votre serveur web ou votre API, ou de messages envoyés à une file d'attente de traitement. Les périodes de pointe de trafic peuvent exercer une pression supplémentaire sur votre infrastructure et la pousser jusqu'à ses limites, ce qui peut avoir des effets en aval.

  C'est un signal clé, parce qu'il vous aide à différencier les problèmes de **capacité des configurations système** inappropriées, qui peuvent causer des problèmes, même en période de faible trafic. Pour les systèmes distribués, il peut également vous aider à **planifier la capacité** à l'avance, pour répondre à la demande à venir.

- **Erreurs :** Les **erreurs** peuvent vous **renseigner sur les erreurs de configuration de votre infrastructure**, **les bugs dans votre code ou les dépendances non résolues**. Par exemple, un pic du taux d'erreur pourrait indiquer la défaillance d'une base de données, ou une panne de réseau.

  Suite à un déploiement de code, il peut indiquer des bugs dans le code qui ont survécu aux tests, ou qui n'ont fait surface que dans votre environnement de production. **Le message d'erreur vous donnera plus d'informations sur le problème exact**. Les erreurs peuvent également affecter les autres métriques, en réduisant artificiellement la latence, ou les tentatives répétées qui finissent par saturer d'autres systèmes distribués.

- **Saturation** : La **saturation** définit la **charge sur votre réseau** et les r**essources de votre serveur**. Chaque ressource a une **limite** au-delà de laquelle les performances se dégradent ou deviennent indisponibles. Ceci s'applique aux ressources telles que l'utilisation du CPU, l'utilisation de la mémoire, la capacité du disque et les opérations par seconde. Il faut comprendre la conception et l'expérience de votre système distribué, pour savoir quelles parties de votre service pourraient devenir saturées en premier. Souvent, ces mesures sont des indicateurs avancés, de sorte que vous pouvez ajuster la capacité avant que la performance ne se dégrade.

  Atteindre la limite de saturation peut affecter votre service de différentes manières. Par exemple, lorsque le CPU est plein, cela peut entraîner des réponses retardées, l'espace de stockage plein peut entraîner une défaillance des écritures

sur le disque, et la saturation du réseau peut entraîner la chute de paquets.