softfluent

Livre blanc

DevOps : Implémentation d'un CI/CD

Contexte

S'adapter aux évolutions du marché et aux exigences du client n'est plus une option aujourd'hui et c'est ce qui explique en grande partie l'essor des méthodologies agiles et des pratiques DevOps.

Alors que les applications traditionnelles étaient pour l'essentiel des applications de back office, les applications web sont, au contraire, en frontal avec les clients. Pour être compétitives, les entreprises n'ont pas d'autre choix que de prendre le virage digital et d'optimiser l'expérience client.
C'est une véritable course contre la montre pour être le 1er sur le marché, être le plus performant et le rester.

Table des matières

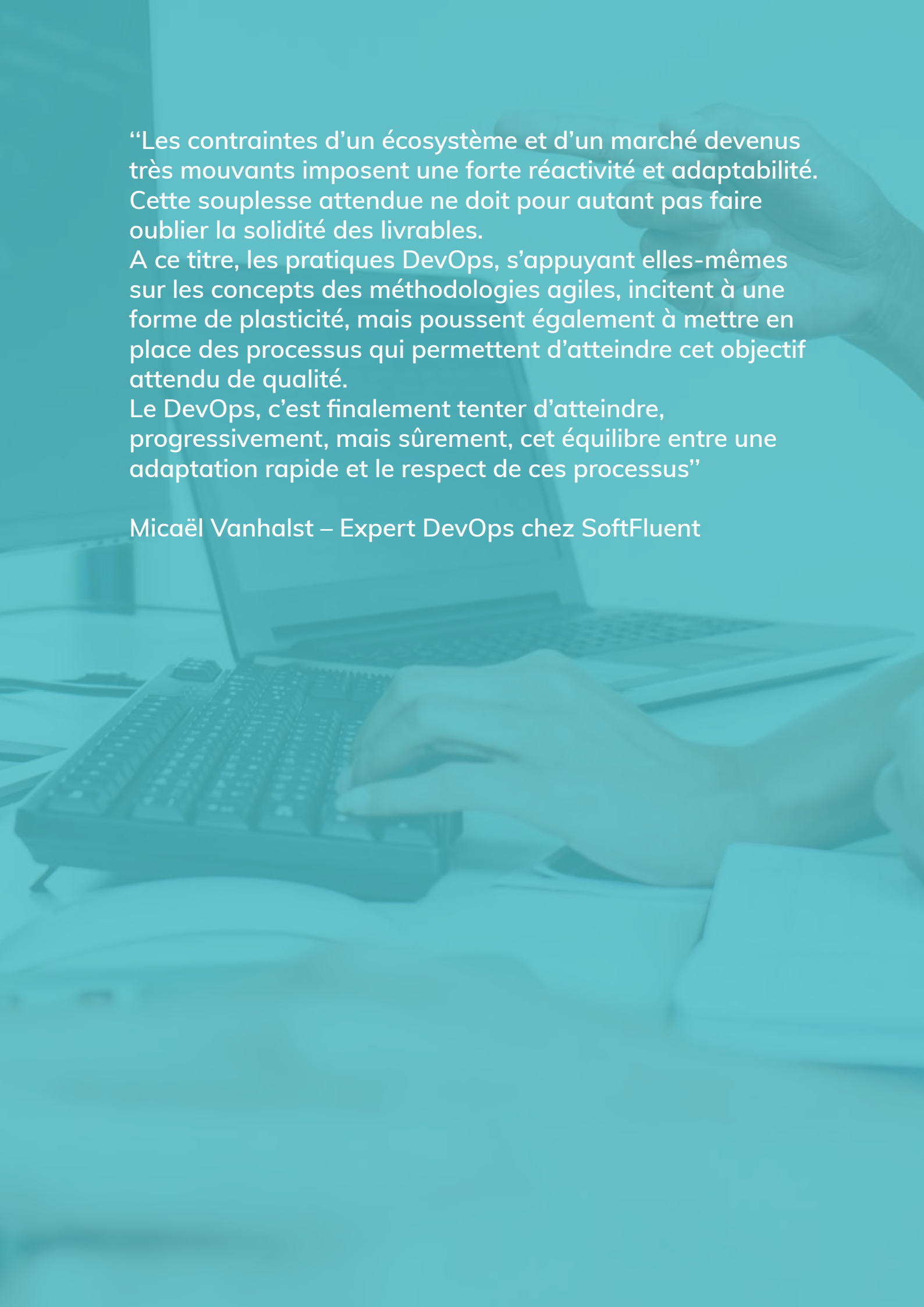
Introduction

Dans ce contexte de compétitivité exacerbée, de rythme effréné des évolutions technologiques et d'attentes croissantes des clients, l'organisation en silo s'avère totalement inadaptée. Il faut livrer rapidement les développements pour que le client puisse en bénéficier le plus vite possible.

La pression du 'Time to Market' nécessite un environnement capable de favoriser des releases fiables, de meilleure qualité, délivrées plus fréquemment, et plus rapidement, ce que permet la démarche DevOps...

La démarche DevOps permet de rapprocher les opérations de développement et de production en entreprise tout en alignant les équipes, les outils et les processus. L'objectif à la clé : améliorer le délai de mise en production, générer de la valeur, réduire les coûts informatiques et mieux répondre aux besoins métiers.

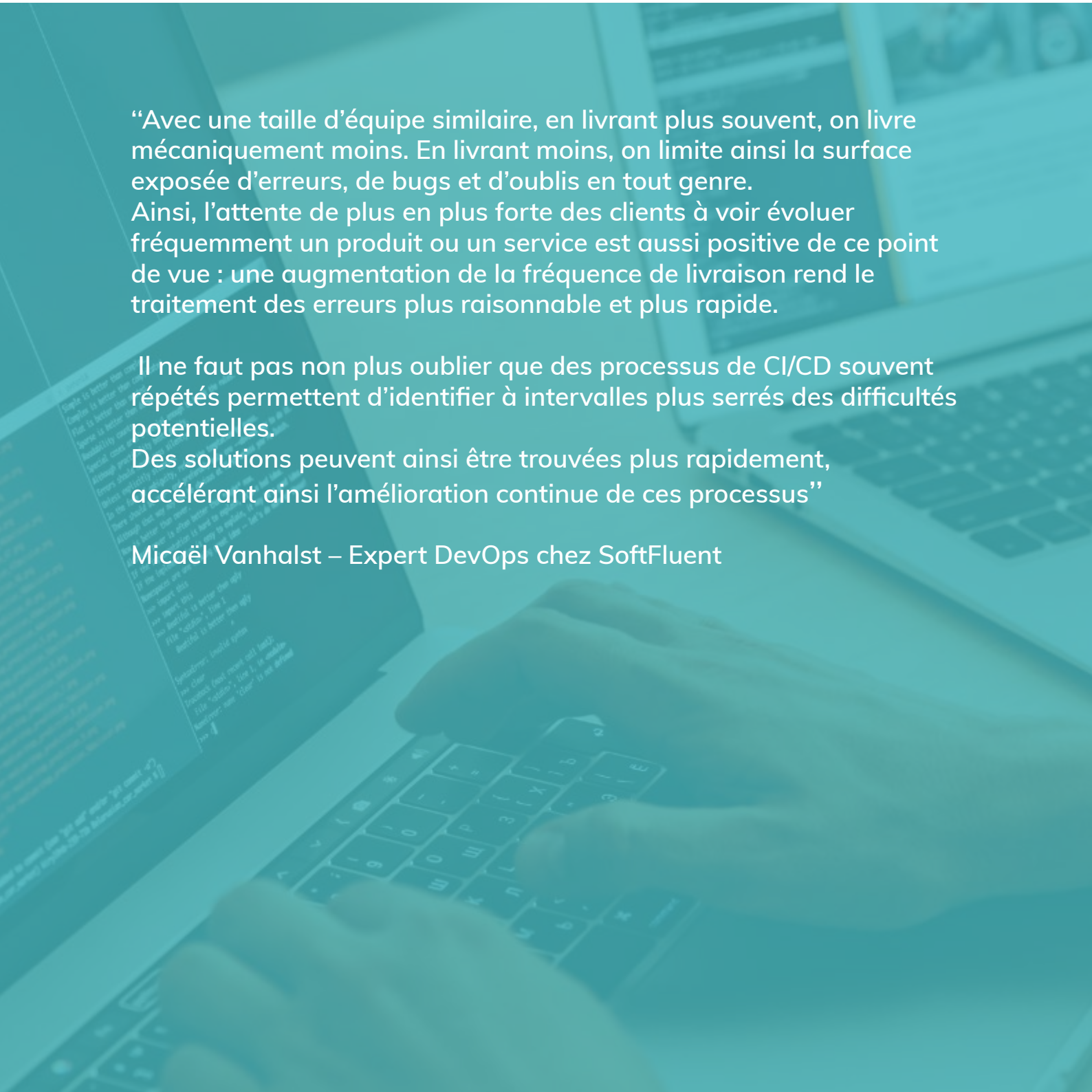
Pour s'assurer du maintien de cette qualité en permanence, la pratique de tests doit également être menée en continu. L'entreprise gagne en flexibilité et améliore la qualité de service pour ses clients en temps réel et de manière automatique.



“Les contraintes d’un écosystème et d’un marché devenus très mouvants imposent une forte réactivité et adaptabilité. Cette souplesse attendue ne doit pour autant pas faire oublier la solidité des livrables.
A ce titre, les pratiques DevOps, s’appuyant elles-mêmes sur les concepts des méthodologies agiles, incitent à une forme de plasticité, mais poussent également à mettre en place des processus qui permettent d’atteindre cet objectif attendu de qualité.
Le DevOps, c’est finalement tenter d’atteindre, progressivement, mais sûrement, cet équilibre entre une adaptation rapide et le respect de ces processus”

Micaël Vanhalst – Expert DevOps chez SoftFluent

Avec DevOps, le cercle est vertueux. Les mises à jour fréquentes préviennent des petites erreurs, les équipes IT, ingénierie et qualité sont motivées pour travailler ensemble dans la même direction et sont donc plus efficaces. Le processus de prise de décision s'en trouve accéléré.



"Avec une taille d'équipe similaire, en livrant plus souvent, on livre mécaniquement moins. En livrant moins, on limite ainsi la surface exposée d'erreurs, de bugs et d'oublis en tout genre.
Ainsi, l'attente de plus en plus forte des clients à voir évoluer fréquemment un produit ou un service est aussi positive de ce point de vue : une augmentation de la fréquence de livraison rend le traitement des erreurs plus raisonnable et plus rapide.

Il ne faut pas non plus oublier que des processus de CI/CD souvent répétés permettent d'identifier à intervalles plus serrés des difficultés potentielles.
Des solutions peuvent ainsi être trouvées plus rapidement, accélérant ainsi l'amélioration continue de ces processus"

Micaël Vanhalst – Expert DevOps chez SoftFluent

Approche CI/CD (Intégration Continue/ Déploiement continu)

L'approche CI/CD permet d'augmenter la fréquence de distribution des applications grâce à l'introduction de l'automatisation au niveau des étapes de développement des applications.

Les bénéfices liés à l'industrialisation des développements et des processus associés sont multiples. On pense bien évidemment à la réduction des temps de déploiement et donc des coûts des projets de développement.
L'enjeu est également de réduire l'intervention humaine sur les tâches répétitives, de limiter les risques d'erreurs et, surtout, de permettre aux développeurs de se focaliser sur des tâches à plus haute valeur ajoutée.

L'automatisation contribue aussi à la réalisation d'un concept fort du DevOps, celui de la confiance. Avec une automatisation mature, bien maîtrisée, on ne devrait plus jamais craindre la mise en production d'un hotfix à déployer en urgence un vendredi soir, juste avant de partir en week-end...

Micaël Vanhalst – Expert DevOps chez SoftFluent

Les principaux concepts liés à l'approche CI/CD sont l'intégration continue, la distribution continue et le déploiement continu. L'approche CI/CD représente une solution aux problèmes posés par l'intégration de nouveaux segments de code pour les équipes de développement et d'exploitation.

Explication du CI, CD : différences

Le concept de développement d'applications modernes consiste à faire travailler plusieurs développeurs simultanément sur différentes fonctions d'une même application. Toutefois, lorsqu'un développeur travaille de son côté et apporte des modifications à une application, celle-ci peuvent entrer en conflit avec les modifications éventuelles d'autres développeurs et nécessiter des interventions manuelles.

Et ce problème se complexifie encore si ce même développeur travaille dans son propre environnement de développement intégré.
D'où l'importance du CI/CD.

Intégration continue

L'intégration continue (CI) permet aux développeurs de fusionner plus fréquemment leurs modifications de code dans une « branche » partagée, souvent critique et qui doit être protégée.
Les opérations réalisées par l'intégration continue doivent garantir cette protection. Ainsi, les développeurs peuvent intégrer leur code modifié sans problème, plusieurs fois par jour s'ils le souhaitent. Les modifications à fusionner sont automatiquement testées pour détecter le moindre conflit entre le code existant et le nouveau (à tous les niveaux, classes, fonctions, modules...).
Les dysfonctionnements éventuels sont ainsi résolus très tôt, plus fréquemment et plus rapidement.

Pour que l'intégration continue reste efficace, il est essentiel que les opérations qu'elle assure s'exécutent dans un temps raisonnable, de l'ordre de quelques minutes.

L'expérience montre qu'au-delà de 7 minutes, un développeur commence à s'impatienter. Pour éviter les manipulations parfois complexes ou simplement méconnues avec le gestionnaire de code source, le développeur préfère souvent attendre la complétion de l'intégration continue pour démarrer une nouvelle fonctionnalité à partir d'un code source à jour. Si l'attente devient trop longue, le risque est de voir un mécontentement naître qui pourrait aboutir à une désactivation de l'intégration continue. Ce serait alors un échec.

Micaël Vanhalst – Expert DevOps chez SoftFluent

“La livraison continue, pour être efficace, doit s’appuyer sur des référentiels capables de proposer une fonctionnalité de suivi de version. Cela est essentiel pour les raisons suivantes :
Empêcher de déposer un nouveau livrable avec un même numéro de version ; Donner une lecture simple de la version à déployer vers un environnement donné ; Accorder la possibilité de revenir (le plus automatiquement possible) à une version antérieure du livrable en cas de problème.
Des solutions comme « Nexus Repository » ou « Azure DevOps Artifacts » sont capables de fournir ce type de fonctionnalité.”

Micaël Vanhalst – Expert DevOps

Distribution (ou livraison) continue

Après l'automatisation des tests unitaires dans le cadre de l'intégration continue, la distribution continue automatise la publication du code validé dans un référentiel.

Aussi, pour garantir l'efficacité du processus de distribution continue, il faut d'abord introduire le processus d'intégration continue dans le pipeline de développement. La distribution continue permet de disposer d'une base de code toujours prête à être déployée dans un environnement de production.

Dans le cadre de la distribution continue, chaque étape (de la fusion des modifications de code jusqu'à la distribution des versions prêtes pour la production) implique l'automatisation des processus de test et de publication du code.

À la fin de ce processus, l'équipe d'exploitation est en mesure de déployer facilement et rapidement une application dans un environnement de production.

La mise en œuvre de la distribution ou livraison continue doit être optimisée. L'artefact de livraison ne doit être produit que lorsque c'est nécessaire.
Ainsi, si votre workflow Git prévoit une livraison qu'à partir de la branche principale, il est essentiel de conditionner la livraison continue en fonction de la branche de destination. Ceci évitera des opérations inutiles et l'utilisation de ressources qui pourraient être utiles par ailleurs, comme le serveur d'intégration.

Micaël Vanhalst – Expert DevOps

Déploiement continu

L'étape finale d'un pipeline CI/CD mature est le déploiement continu. En complément du processus de distribution continue, qui automatise la publication d'une version prête pour la production dans un référentiel, le déploiement continu automatise l'exécution d'une nouvelle version d'une application dans un environnement de production.

En l'absence de passerelle manuelle entre la mise en production et l'étape précédente du pipeline, le déploiement continu dépend surtout de la conception de l'automatisation des processus de test.

Ensemble, ces trois pratiques CI/CD réduisent les risques liés au déploiement des applications, puisqu'il est plus simple de publier des modifications par petites touches qu'en un seul bloc. Cette approche nécessite néanmoins un investissement de départ considérable, car les tests automatisés devront être rédigés de manière à s'adapter à un large éventail d'étapes de test et de lancement dans le pipeline CI/CD.

Quelle est la différence entre CI, CD et l'autre CD ?

L'intégration continue (CI) consiste à apporter régulièrement des modifications au code d'une application, à les tester, puis à les fusionner dans un référentiel partagé.

“De nombreuses discussions sur le « CD » du CI/CD fourmillent autour du sens qu’il faut lui donner. Certains se refusent même à parler de livraison continue. D’autres confondent ou fusionnent volontairement les termes de « livraison continue » et de « déploiement continu ».

Ces grandes étapes de la CI/CD sont pourtant parfaitement identifiables et doivent être maintenues, ne serait-ce que parce que leur implémentation peut se faire progressivement, en commençant par l’intégration continue (CI), suivie de la livraison continue (CD) et, pour terminer, le déploiement continu (encore CD).

Certains de mes clients refusent, essentiellement pour des raisons de gouvernance et en attendant de trouver des solutions qui conviennent, d’aller jusqu’au déploiement continu, qui impose souvent de s’appuyer sur des grands opérateurs cloud comme AWS, Azure ou GCP. Ils s’arrêtent donc aujourd’hui à la livraison continue. Le terme reste donc important et doit continuer à être employé lorsqu’il convient.”

Micaël Vanhalst – Expert DevOps

Le « CD » de CI/CD désigne la « distribution (ou livraison) continue » et/ou le « déploiement continu », qui sont des concepts très proches, parfois utilisés de façon interchangeable. Les deux concepts concernent l'automatisation d'étapes plus avancées du pipeline, mais ils sont parfois dissociés pour illustrer le haut degré d'automatisation.

Dans le cadre de la distribution continue, les modifications apportées sont automatiquement testées et téléchargées dans un référentiel. Elles pourront ainsi être déployées, par l'équipe d'exploitation, dans un environnement de production actif. Le processus de distribution continue permet de :

- Résoudre les problèmes de visibilité et de communication entre l'équipe de développement et l'équipe
- Simplifier au maximum le déploiement de nouveau code.

Le déploiement continu (l'autre signification possible de « CD ») peut désigner le transfert automatique des modifications du développeur depuis le référentiel vers l'environnement de production. Ce processus permet de :

- Soulager les équipes d'exploitation
- Optimiser les mises en production

L’approche CI/CD se rapporte à un processus, souvent représenté sous forme de pipeline, qui consiste à introduire un haut degré d’automatisation et de surveillance continues dans le cycle de vie des applications.

Pipeline

C’est l’implémentation de l’approche CI CD.
Un pipeline de déploiement est une façon d’orchestrer vos builds au travers d’une série de passages garantissant la qualité, avec des approbations automatisées ou manuelles aux étapes stratégiques, culminant avec le déploiement en production.

Le pipeline de build est un ensemble de tâches pour, notamment, charger les sources du projet, restaurer les dépendances, générer le projet, exécuter les tests unitaires et, enfin, produire une version qui pourra être déployée dans l’environnement d’exécution.

“Comme évoqué plus haut, ce pipeline de build est généralement exploité dans le cadre de l’intégration continue, lorsqu’on a besoin d’assurer ces opérations de validation du code. Les grands opérateurs du marché, comme GitHub, GitLab ou Azure DevOps, proposent une configuration qui permet de déclencher automatiquement un pipeline de build lors des demandes de « Pull request ».
Cela rend systématique l’exécution de l’ensemble des actions nécessaires à garantir et maintenir une grande qualité de code.”

Micaël Vanhalst – Expert DevOps chez SoftFluent

Les étapes

Les étapes qui constituent un pipeline CI/CD sont des sous-ensembles distincts de tâches regroupés dans ce que nous appelons une phase de pipeline. Voici les phases de pipeline les plus courantes :

- Récupération : code source récupéré du gestionnaire
- Création : compilation de l'application
- Test : test du code et notamment tests automatisés
- Lancement : distribution de l'application au référentiel
- Déploiement : déploiement du code en production
- Validation et conformité : ces étapes de validation sont à adapter en fonction des besoins de l'entreprise

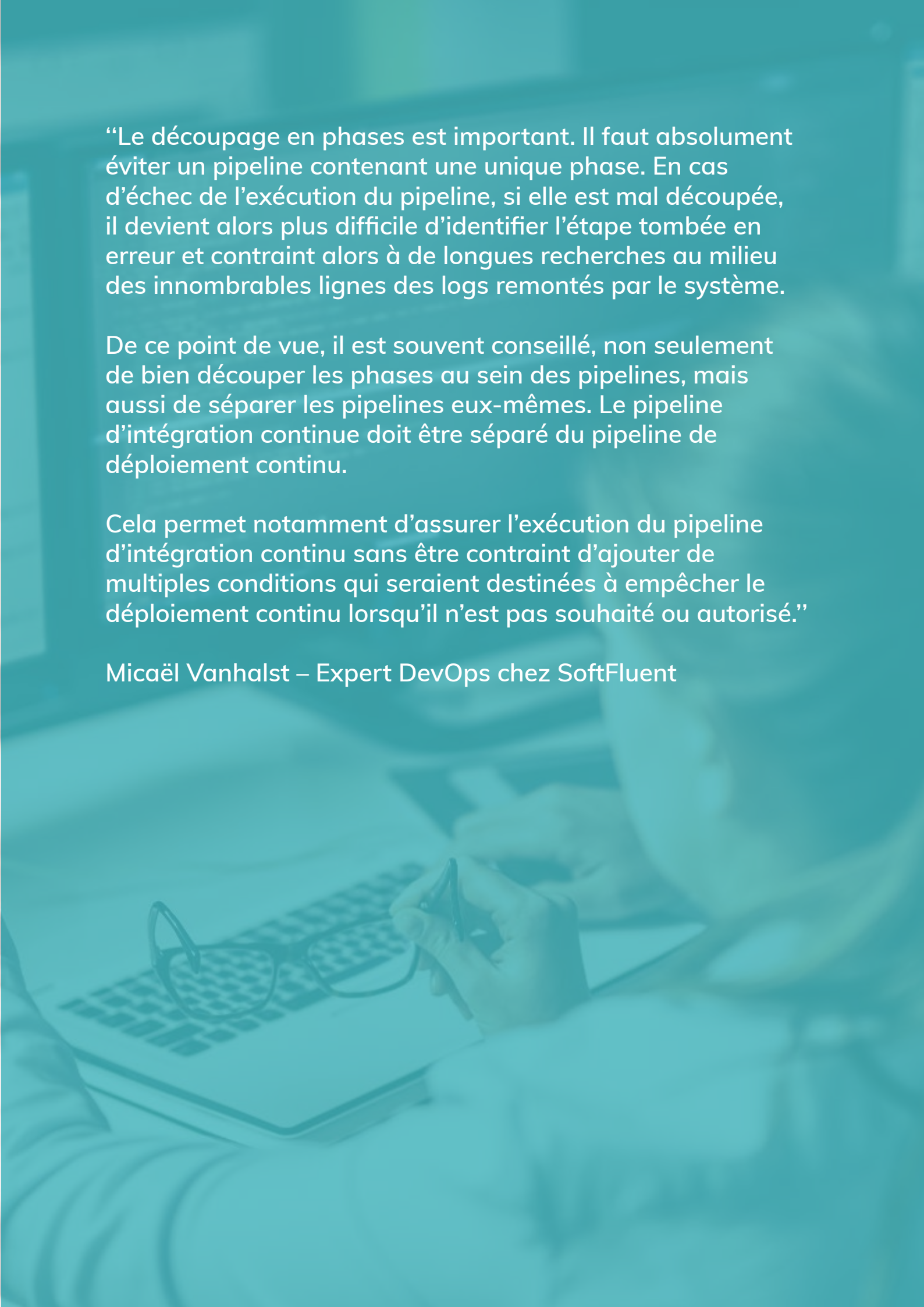
Cette liste de phases de pipeline n'est pas exhaustive. Votre pipeline devra se conformer aux exigences de votre entreprise

“Le découpage en phases est important. Il faut absolument éviter un pipeline contenant une unique phase. En cas d’échec de l’exécution du pipeline, si elle est mal découpée, il devient alors plus difficile d’identifier l’étape tombée en erreur et contraint alors à de longues recherches au milieu des innombrables lignes des logs remontés par le système.

De ce point de vue, il est souvent conseillé, non seulement de bien découper les phases au sein des pipelines, mais aussi de séparer les pipelines eux-mêmes. Le pipeline d’intégration continue doit être séparé du pipeline de déploiement continu.

Cela permet notamment d’assurer l’exécution du pipeline d’intégration continu sans être contraint d’ajouter de multiples conditions qui seraient destinées à empêcher le déploiement continu lorsqu’il n’est pas souhaité ou autorisé.”

Micaël Vanhalst – Expert DevOps chez SoftFluent



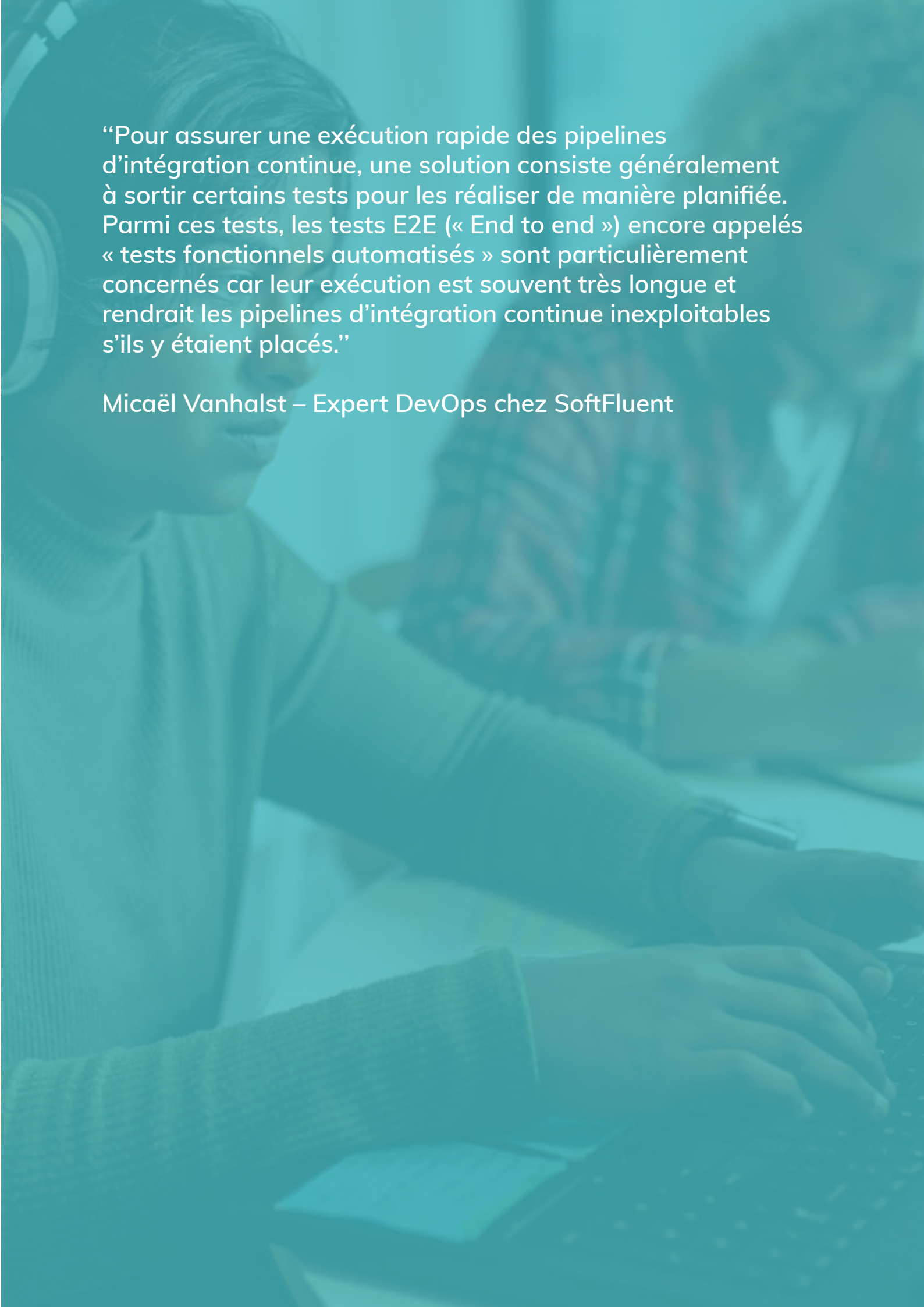
Lors de la mise en œuvre de pipelines, il est important de s'interroger sur leur déclenchement. Même s'il faut automatiser le plus possible, il faut aussi donner la liberté aux développeurs de tester, dans une mesure raisonnable, le code de leur branche de fonctionnalité en passant par l'exécution d'un pipeline de build.

Il est possible d'implémenter un déploiement continu quand les changements d'application sont exécutés dans le pipeline CI/CD et quand les versions successives sont déployées directement dans les environnements de production. Les équipes qui pratiquent la livraison continue choisissent de déployer en production selon un calendrier quotidien.

Calendrier de builds

La plupart des outils CI/CD permettent aux développeurs de publier des builds à la demande, soit déclenchées par des modifications de code dans le référentiel de contrôle de version, soit selon un calendrier défini. Les équipes doivent discuter du calendrier de builds qui convient le mieux à la taille de l'équipe, au nombre de modifications de code attendu chaque jour et à d'autres considérations relatives à l'application.

C'est une bonne pratique pour s'assurer que les modifications de code et les builds sont rapides. En effet, l'absence de calendrier pour les builds peut entraver la progression des équipes qui essaient de coder rapidement et de modifier du code fréquemment.



“Pour assurer une exécution rapide des pipelines d’intégration continue, une solution consiste généralement à sortir certains tests pour les réaliser de manière planifiée. Parmi ces tests, les tests E2E (« End to end ») encore appelés « tests fonctionnels automatisés » sont particulièrement concernés car leur exécution est souvent très longue et rendrait les pipelines d’intégration continue inexploitables s’ils y étaient placés.”

Micaël Vanhalst – Expert DevOps chez SoftFluent

Contrôle de version

Les équipes qui adoptent l'intégration continue commencent souvent par configurer le contrôle de version et par définir des pratiques. Même si le contrôle du code est effectué fréquemment, les fonctionnalités et les corrections sont appliquées sur des périodes courtes ou plus longues. Les équipes de développement pratiquant l'intégration continue utilisent différentes techniques pour contrôler les fonctionnalités et le code prêts à être déployés en production

- 'Feature Flags', autrement dit, des indicateurs de fonctionnalités. Ce mécanisme de configuration permet d'activer ou de désactiver des fonctionnalités et du code au moment de l'exécution. Les fonctionnalités encore en cours de développement sont balisées dans le code par des indicateurs de fonctionnalités déployées en production avec la branche principale, et désactivées jusqu'à ce qu'elles soient prêtes à être utilisées

- Une autre technique consiste à effectuer un contrôle de version par branche. Certaines équipes optent pour un workflow Git comme GitFlow, pour séparer, sur des branches distinctes, le code en cours de développement et le code validé et testé.

“Des possibilités techniques existent pour exposer de manière différenciée des fonctionnalités. Il ne faut cependant pas minimiser le coût en termes de développement.
Comme pour tout le reste, il faut finalement proposer, discuter et choisir en étant toujours le plus réaliste et pragmatique possible.
Ce type de choix doit d’ailleurs impliquer largement les métiers, responsables des fonctionnalités à implémenter, en explorant des solutions plus organisationnelles, le temps de mettre en œuvre des solutions techniques”

Micaël Vanhalst – Expert DevOps chez SoftFluent

Les intérêts d'un pipeline

En optimisant et en standardisant les processus, les bénéfices sont multiples :

- Livrer des applications fiables en testant automatiquement chaque changement dans le code source pour réduire l'introduction de bogues.
- Livrer de nouvelles fonctionnalités plus fréquemment en optimisant vos processus et en supprimant les obstacles à la productivité.
- Limiter le stress et améliorez la productivité des développeurs.
- Réduire le code inutile
- Réduire la taille des branches de fonctionnalités pour faciliter l'examen par les pairs
- Augmenter la qualité de code
- Améliorer la gestion des produits et donc la satisfaction des utilisateurs

Résultats : moins de pannes difficiles à réparer et des ingénieurs heureux

En résumé l'approche CI / CD consiste en cycles de déploiement courts qui conduisent à des mises à jour moins risquées et plus fréquentes, et donc à un apprentissage plus rapide et à un feedback plus fréquent des utilisateurs.

Plus précisément, l'approche CI/CD garantit une automatisation et une surveillance continues tout au long du cycle de vie des applications, des phases d'intégration et de test jusqu'à la distribution et au déploiement. Ensemble, ces pratiques sont souvent désignées par l'expression « pipeline CI/CD » et elles reposent sur une collaboration agile entre les équipes de développement et d'exploitation.

KPI

Les indicateurs de performance doivent être concrets, mesurables et répondre aux particularités du DevOps :

- Etre évocateurs auprès des équipes de développement comme des équipes des opérations
- Permettre de mesurer la performance de toute la chaîne en termes de productivité, qualité, et satisfaction client

Les KPI sont le gage de rapidité et d'efficacité des équipes d'autant plus s'ils sont liés à un KPI métier. Non seulement cela permet de soigner le périmètre et l'efficacité des tests (les tests pour les tests ne sont pas forcément utiles et surtout il faut penser à archiver ceux devenus inutiles) mais cela permet surtout de stimuler et de motiver l'équipe.

"Ces KPI doivent être mis en œuvre très tôt dans le processus de mise en œuvre du CI/CD. Idéalement, certains devraient exister avant même l'implémentation du CI/CD. Ils permettent ainsi d'avoir une vue objective des améliorations que le CI/CD apportent. D'autres sont ensuite ajoutés pour mesurer d'autres éléments, accordant alors la possibilité de détecter des difficultés éventuelles, d'y apporter une réponse adaptée et finalement de progresser globalement"

Micaël Vanhalst – Expert DevOps chez SoftFluent

A lire sur le même sujet : Devops, quels indicateurs de performance et comment s'améliorer ?

Les modèles de déploiement

Le déploiement continu est l'étape finale d'un pipeline CI/CD mature et il dépend surtout de l'automatisation des processus de tests qui nécessite un investissement non négligeable pour que les tests soient rédigés de manière à s'adapter à un large éventail d'étapes.

Néanmoins sans les stratégies avancées de déploiement, il est difficile de mettre en place le déploiement continu. C'est la raison pour laquelle 99 % des DevOps ne font pas de CD au sens « Continuous Deployment », en réalité ils font du CD au sens « Continuous Delivery ».
Le déploiement continu implique de répondre à certaines questions :

Faut-il garder le site en production le temps qu'un correctif soit livré ? Faut-il faire un rollback vers la version précédente ?
Quoi qu'il en soit, le choix à adopter aura un impact sur la visibilité de la plateforme. Le déploiement d'un correctif peut entrainer un downtime (ou rupture de service), tout comme un rollback. Et, le risque est plus important pour un retour à la version précédente, d'autant plus que l'infrastructure, elle aussi, devra revenir à son état antérieur.

“Les tests sont fondamentaux. Il en existe de nombreux types : les tests unitaires, bien sûr, mais aussi les tests E2E, les tests d’intégration et les tests de montée en charge. Ce sont eux qui vont garantir que la livraison pourra s’effectuer à la fois sans régression et en conservant un niveau de performance suffisant. Ces tests sont généralement exécutés pendant la phase d’intégration continue qui précède le déploiement continu.

Il faut cependant rester vigilant sur leur durée d’exécution. L’intégration continue doit rester rapide, idéalement de l’ordre de quelques minutes, le temps de restaurer les dépendances, de compiler la solution et d’exécuter quelques tests unitaires de base. Il faut donc prévoir une mise en œuvre parallèle de l’exécution de l’ensemble de ces tests, soit de manière planifiée, ou encore déclenchée immédiatement après les opérations de l’intégration continue.

L’exécution de ces nombreux tests peut venir impacter le rythme des déploiements, en imposant une attente qui peut paraître longue, mais le bénéfice est supérieur au risque encouru par le déploiement d’une application ou d’un service mal testé.”

Micaël Vanhalst – Expert Devops chez SoftFluent

Comment adresser ces problèmes dans sa stratégie de déploiement ? La démarche DevOps, qui prône une livraison fréquente des évolutions aux utilisateurs, propose plusieurs moyens pour faire face à de telles situations en production.
Pour réussir le déploiement continu et surtout garantir la continuité des services, il existe des stratégies avancées de déploiement avec des pattern comme Blue-Green, Canary et A/B Testing...

"Le déploiement continu peut mettre en œuvre des processus complexes. Des modèles de déploiement continu permettent par exemple : Le déploiement sans interruption de service avec le modèle « Blue-Green Deployment » ; L'exposition progressive des nouvelles fonctionnalités à différents groupes d'utilisateurs avec des modèles comme le « Canary Release » ou sa version étendue, le « Ring-Based Deployment »."

Micaël Vanhalst – Expert DevOps

Qu'est-ce que le « Blue-Green Deployment » ?

Le « Blue Green Deployment » offre une approche de déploiement en production en sécurité tout en évitant que la plateforme ne soit hors ligne.

Le pattern « Blue-Green Deployment » est une implémentation d'un modèle plus générique appelé ZDD ou « Zero Downtime Deployment ». Son objectif est donc clairement de limiter au maximum la rupture de service lors du déploiement d'une nouvelle version d'une application ou d'un service.

Un déploiement Blue-Green utilise deux environnements de production configurés de manière identique sur lesquels vous pouvez déployer votre application et un routeur pour envoyer du trafic utilisateur vers l'un ou l'autre.
Lorsque que l'un des environnements (le Blue) sert activement les utilisateurs, l'autre (Green) est inactif. Lorsqu'une nouvelle version est prête, on la déploie sur l'environnement Green pour effectuer les dernières étapes de test.
Une fois que l'application fonctionne dans l'environnement Green, on modifie la configuration du routeur pour que tout le trafic utilisateur soit dirigé vers l'environnement Green, le Blue devient alors inactif.

Avec cette approche, pas de downtime. En cas de bogue on peut simplement switcher vers l'environnement de production précédent.
On peut effectuer de nombreux tests (Performance tests, Capacity tests, Stress tests, Load tests) pour évaluer les performances et la réactivité du site Web dans un environnement proche de la production.

Une fois l'environnement Green en production, l'environnement Blue garde la version précédente de la plateforme pour parer à toute éventualité, jusqu'à la livraison d'une nouvelle version qui sera déployée dans l'environnement Blue, puis « switché » en production. Le cycle peut être répété à chaque livraison.

On se retrouve en production avec une version assez mature et qui a quasiment déjà fait ses preuves, l'utilisateur ressentira quant à lui à peine une légère interruption en passant d'une version à l'autre.

Avantages

- L'implémentation est facile et rapide (sur un environnement type Cloud)
- Rollback possible car l'ancien déploiement est encore opérationnel
- Possibilité de faire des tests sur un environnement identique à la production

Inconvénients

- Besoin de deux fois plus de ressources (coûts et maintenance supérieurs)

Qu’est-ce que le « Canary Release » ?

Le principe du « Canary Release » est assez similaire au « Blue-Green Deployment », mais offre aux équipes projet une étape durant laquelle l’application est testée en production par un nombre restreint d’utilisateurs, sans impacter l’expérience utilisateur de la majeure partie des utilisateurs.

Ce mode est à privilégier, par rapport au « Blue Green Deployment », si vous n’êtes pas certain du fonctionnement en production de la nouvelle version. Cette approche permet de réduire énormément le risque de bogue et de rollback en production.

De façon pratique, l’approche « Canary Release » nécessite un second serveur en production. La nouvelle version d’une application est publiée en production sur le deuxième serveur et un petit nombre d’utilisateurs est dirigé vers cette dernière.
Durant cette phase, les feedbacks des utilisateurs peuvent être récupérés, les correctifs publiés pour stabiliser l’application, etc. Une fois l’application assez stable, celle-ci peut être déployée en production.

C’est une approche adéquate pour le « A/B Testing ». Les tests sont effectués en production par des utilisateurs finaux, avec possibilité de recueillir des métriques en situation d’utilisation normale de l’application.

Le « Canary Release » est un moyen de réduire les risques et de valider de nouvelles versions de logiciels en proposant des versions à un petit pourcentage d’utilisateurs. Son objectif est de détecter et de résoudre rapidement un problème avec une nouvelle version de logiciel avant qu’il ne dégrade l’expérience de chacun.

Comment fonctionne le « Canary Release » ?

Le Canary pourrait être considéré comme un cas spécial de « Test A/B », dans lequel le déploiement se produit de manière progressive. Le trafic est transféré lentement d’une version d’une application à une version plus récente d’une application à l’aide d’un routeur de trafic au niveau du serveur.

On commence par déployer une nouvelle version de service, qui ne reçoit aucun trafic. Si l’on observe que le service démarre normalement, on dirige un petit pourcentage de trafic vers ce service. Les erreurs sont surveillées en permanence.

L’augmentation du trafic s’effectue par étape jusqu’à ce que le nouveau service reçoive 100% du trafic. Ce qui permet d’arrêter progressivement les anciens services. La mise en place de cette stratégie de déploiement repose sur des contrôles fiables et précis.

Qu’est ce que le « Ring Based Deployment » ?

Le « Ring based Deployment » est la version étendue du « Canary Release ». Il peut lui-même être combiné avec l’exposition différenciée des fonctionnalités « Features Flags » générant un niveau de complexité encore plus important. Cette combinaison produit un nouveau modèle le « Dark Launch ».

Le modèle du « Ring Based Deployment » étend le « Canary Release » car il identifie de manière plus précise les différentes cibles. Alors que le « Canary Release » redirige progressivement le trafic réseau, de manière indifférenciée, vers une cible générale, le « Ring Based Deployment » procède par étapes en ciblant des publics déterminés à l’avance.

Feature Flags

Ce mécanisme de configuration permet d’activer ou de désactiver des fonctionnalités au moment de l’exécution. Les fonctionnalités encore en cours de développement sont balisées dans le code par des indicateurs de fonctionnalités déployées en production à partir de la branche principale et désactivées jusqu’à ce qu’elles soient prêtes à être utilisées.

“Ce modèle a un impact qui peut être fort en termes de coût de développement. S’il n’a pas été initialement prévu, il peut être difficile à mettre en œuvre. Si possible, il est donc conseillé d’y penser très tôt, dès la conception. Sinon, il faut avoir conscience que cette implémentation demandera un effort conséquent et devra faire l’objet d’un vrai projet.

La version simple de ce modèle s’appuie sur un simple fichier de configuration consommé par un service à développer qui, en fonction du contenu de ce fichier de configuration, a la charge de l’activation ou de la désactivation d’une fonctionnalité. Cette version a l’inconvénient d’imposer un nouveau déploiement de l’application chaque fois qu’on souhaite modifier le fichier de configuration.

Des versions plus avancées vont jusqu’à présenter une interface utilisateur permettant d’activer ou de désactiver dynamiquement des fonctionnalités, sans qu’un déploiement ne soit nécessaire. Il existe même des services hébergés, s’appuyant sur des systèmes d’abonnement et proposant ce type de fonctionnalité, avec une intégration facilitée et permettant de limiter l’effort de développement.”

Micaël Vanhalst – Expert DevOps chez SoftFluent

Dark launch

C'est la combinaison du « Ring-based Deployment » et des « Features Flags ». Concrètement, la nouvelle fonctionnalité lorsqu'elle est encore au stade de préversion est livrée en production avec les autres mises à jour et correctifs de la version en production. Celle-ci est ensuite activée pour un nombre réduit d'utilisateurs.
Ceci peut se faire suivant un certain nombre de critères : zone géographique, sexe ou en proposant un message invitant des utilisateurs à tester cette nouvelle fonctionnalité. Les équipes projet seront alors en mesure de :

- Obtenir les retours des utilisateurs sur la fonctionnalité ;
- Analyser le comportement de ceux-ci face à cette dernière ;
- Evaluer le degré de satisfaction, évaluer l'impact de la fonctionnalité sur les performances du système, etc.

Au fil des déploiements, la fonctionnalité est améliorée en tenant compte du feedback des utilisateurs et des différents indicateurs, jusqu'à son déploiement à l'ensemble des utilisateurs.

"Ce modèle permet une très grande souplesse, en proposant d'activer des fonctionnalités en fonction d'un groupe d'utilisateurs cible, d'un environnement ou de tout autre paramètre. Pour implémenter ce modèle, si possible, il est généralement conseillé de s'appuyer sur des solutions existantes comme « LaunchDarkly »."

Micaël Vanhalst – Expert DevOps chez SoftFluent

Qu’est-ce que le « Test A/B » ?

Le « Test A/B » est une méthode connue de comparaison de deux versions qui dépasse le cadre du déploiement. Deux variantes sont présentées aux utilisateurs de manière aléatoire pour déterminer celle qui fonctionne le mieux.

Les cas d’utilisation pour les « Tests A/B » de microservices sont souvent l’essai de nouvelles fonctionnalités pour une partie d’utilisateurs ou de régions géographiques, ou le test d’une mise à jour à une échelle réduite avant le déploiement complet.

Dans cette stratégie de déploiement, si un nombre significatif d’erreurs est détecté sur la nouvelle version le déploiement peut alors être annulé et tout le trafic est envoyé au service ancien.

Comment fonctionne un « A/B Testing » ?

Dans un « Test A/B », nous modifions une page Web d’une application pour créer une deuxième version de la même page par exemple. Ça peut être un titre H2, la position d’un bouton, ou une refonte complète de la page. Une partie du trafic affiche la version originale de la page et l’autre partie, identifiée par un paramètre spécifique, la version modifiée avec des statistiques pour évaluer la performance de chacune.

Et avec Azure

Voyons ici un exemple d'implémentation concrète du « Blue-Green Deployment ».
Lorsque vous créez une Web Application sur Azure, vous disposez par défaut d'un slot de production.

Chaque fois que vous déployez votre application, cette dernière est déployée par défaut sur ce slot. Azure Web Apps vous offre cependant la possibilité de créer un autre emplacement de déploiement (staging) dans le même service plan. Lorsque la nouvelle version est prête à entrer en production, vous la déployez dans un premier temps sur le slot de staging.

A partir du portail Azure, vous pouvez définir le pourcentage d'utilisateurs qui seront redirigés vers cette nouvelle version et effectuer vos tests de performance en production (A/B Testing). Une fois cette dernière prête à entrer en utilisation, en un clic, vous effectuez un swap entre les deux versions.

La version sur le slot staging devient la version en production et celle qui était en production précédemment passe en staging. En cas d'un bogue important le retour à la version précédente est tout aussi simple.

“Après l’échange des slots de staging et de production, il ne faut pas oublier de modifier la configuration de la redirection du trafic réseau pour que l’ensemble des utilisateurs ait accès à la production.

Toutes ces opérations peuvent être accomplies en exploitant le portail Azure, mais il est recommandé de les mettre en œuvre de la manière la plus automatisée possible. Elles peuvent évidemment être réalisées dans le cadre du déploiement continu.”

Micaël Vanhalst – Expert DevOps chez SoftFluent

Rappel des bonnes pratiques de déploiement Cloud

Chaque équipe de développement a des exigences uniques qui peuvent plus ou moins complexifier l'implémentation d'un pipeline de déploiement sur un service Cloud et avoir une incidence sur son efficacité. Les trois principaux composants du déploiement sur App Service sont : les sources de déploiement, les pipelines de génération et les mécanismes de déploiement.

Source de déploiement

Une source de déploiement correspond à l'emplacement de votre code d'application. Pour les applications de production, la source de déploiement est généralement un référentiel hébergé par un logiciel de gestion de version comme Git et en s'appuyant sur des opérateurs comme GitHub, GitLab, BitBucket ou les repos d'Azure DevOps.

Pipeline de build

Une fois que vous avez choisi une source de déploiement, l'étape suivante consiste à choisir un pipeline de build. Un pipeline de build lit votre code source à partir de la source de déploiement et exécute une série d'étapes (telles que la compilation de code, la minimisation du HTML et du JavaScript, l'exécution de tests et l'empaquetage de composants) pour rendre l'application exécutable.

Ces opérations peuvent être exécutées sur un serveur de builds, comme Jenkins, TeamCity ou en exploitant les pipelines d'Azure DevOps.

"Le pipeline de build généralement exécuté dans le cadre de l'intégration continue, va produire en sortie le package applicatif qu'on appelle souvent artefact de livraison. Il est important de respecter la stabilité de cet artefact. En d'autres termes, il faut que ce soit le même livrable issu de la CI qui soit transporté vers les différents environnements (Dev, Test, UAT, Prod par exemple) au risque de se retrouver avec des versions différentes de l'application au sein des différents environnements.

Dans ce cas, si une erreur est détectée, on n'a alors plus de point de comparaison et il devient très complexe d'en identifier la source. C'est pourtant un cas assez fréquent qui au-delà du temps perdu, présente un risque majeur. La stabilité de l'artefact de livraison est donc essentielle. La seule chose qu'on puisse autoriser, c'est l'adaptation de la configuration afin qu'elle s'adapte aux environnements cibles successifs."

Micaël Vanhalst – Expert DevOps chez SoftFluent

Mécanisme de déploiement

Le mécanisme de déploiement est l'action utilisée pour placer votre application intégrée dans le répertoire « /home/site/wwwroot » de votre application web. Le répertoire « wwwroot » est un emplacement de stockage monté partagé par toutes les instances de votre application web.

Lorsque le mécanisme de déploiement place votre application dans ce répertoire, vos instances reçoivent une notification pour synchroniser les nouveaux fichiers. App Service prend en charge les mécanismes de déploiement suivants :

- Points de terminaison Kudu : Kudu est l'outil de productivité de développement open source qui gère les déploiements continus et fournit des points de terminaison HTTP pour le déploiement, comme « Zip Deploy » (déploiement d'une archive ZIP ou WAR) ;

- FTP et WebDeploy : à l'aide des informations d'identification de votre site ou de votre utilisateur, vous pouvez aussi téléverser des fichiers via FTP ou WebDeploy.

La plupart des outils de déploiement, tels que les pipelines d'Azure DevOps, Jenkins et les plug-ins d'éditeur utilisent un de ces mécanismes de déploiement.

Utiliser des emplacements de déploiement

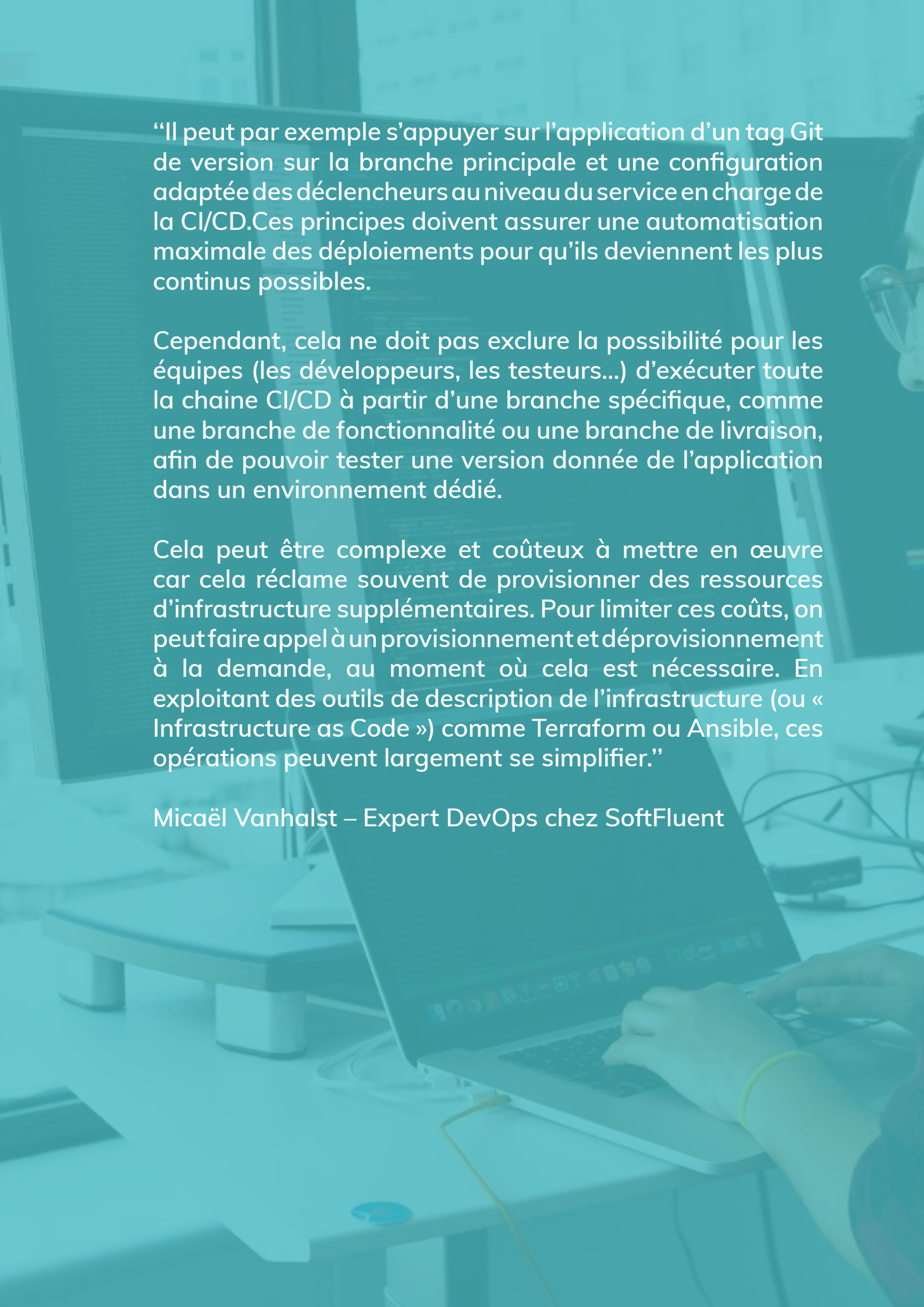
Dans la mesure du possible :

- Utilisez des emplacements de déploiement comme un plan App Service Standard par exemple. Cela permet de déployer dans un environnement intermédiaire et d'effectuer les tests de vérification de Build.

- Déployez votre branche de production principale sur un emplacement dit de 'non-production' et lorsque vous êtes prêt, échangez-la dans l'emplacement de production. Un échange en production, plutôt qu'un déploiement en production permet d'éviter les temps d'arrêt et de restaurer les modifications en les échangeant à nouveau.

Déployer du code en continu

Que votre équipe projet ait sélectionné un workflow Git avancé comme « GitFlow » ou ait décidé d'aller vers une stratégie de gestion des branches Git plus simple, comme le TBD ou « Trunk Based Development », il faut qu'un choix sur le processus de mise en production ait été fait.



“Il peut par exemple s’appuyer sur l’application d’un tag Git de version sur la branche principale et une configuration adaptée des déclencheurs au niveau du service en charge de la CI/CD.Ces principes doivent assurer une automatisation maximale des déploiements pour qu’ils deviennent les plus continus possibles.

Cependant, cela ne doit pas exclure la possibilité pour les équipes (les développeurs, les testeurs…) d’exécuter toute la chaine CI/CD à partir d’une branche spécifique, comme une branche de fonctionnalité ou une branche de livraison, afin de pouvoir tester une version donnée de l’application dans un environnement dédié.

Cela peut être complexe et coûteux à mettre en œuvre car cela réclame souvent de provisionner des ressources d’infrastructure supplémentaires. Pour limiter ces coûts, on peut faire appel à un provisionnement et déprovisionnement à la demande, au moment où cela est nécessaire. En exploitant des outils de description de l’infrastructure (ou « Infrastructure as Code ») comme Terraform ou Ansible, ces opérations peuvent largement se simplifier.”

Micaël Vanhalst – Expert DevOps chez SoftFluent

Déployer des conteneurs en continu

La conteneurisation a déjà pris une grande part au sein des pratiques DevOps, mais elle ne peut encore que progresser. En effet, la conteneurisation apporte un nombre important d'avantages, parmi lesquels :

- La standardisation du packaging d'une application ou d'un service
- Une très grande stabilité de la description des pipelines (la gestion des dépendances, d'une grande partie de la configuration étant gérées au sein du conteneur lui-même)
- La possibilité d'exécuter sur le poste du développeur un environnement qui devient très proche de la production (il suffit d'installer le runtime Docker)
- L'exploitation d'orchestrateurs tels que Kubernetes pour améliorer le processus de déploiement (en évitant notamment les ruptures de service)

A lire aussi https://www.softfluent.fr/blog/devops-docker-net-core-optimiser-taille-image-docker/

Le DevOps n'a pas inventé les modèles de déploiement présentés ci-dessus. Le DevOps définit de manière assez rationnelle et pragmatique, un certain nombre de concepts.
Il se trouve simplement que ces modèles de déploiement correspondent à ces concepts. C'est pour cette raison qu'ils sont souvent mis en avant quand on aborde le sujet du DevOps. D'autres modèles existent, certains sauront répondre à vos besoins.

Il sera parfois nécessaire d'adapter ou de créer un modèle qui vienne s'aligner à vos contraintes.

CI/CD as a service

L'intégration et la livraison continues sont des composantes fondamentales de la démarche DevOps. Toutefois, alors que les pipelines doivent prendre en compte les nouvelles architectures – conteneurs notamment –, il devient nécessaire d'abstraire un peu plus ces étapes du cycle de vie d'une application : c'est le sens d'une approche 'as a Service' du CI/CD.

'As a Service' désigne des services fournis par le biais du cloud, qu'il s'agisse de SaaS, de IaaS ou de PaaS. Facile d'en conclure que CI/CD 'as a Service' signifie alors que le pipeline, managé ou non, s'appuie sur une infrastructure cloud quelle qu'elle soit.

Ce modèle permet aux équipes de s'affranchir des problèmes habituels et des frais liés à la gestion et au maintien d'une infrastructure CI/CD. Il s'agit d'un SaaS entièrement géré, qui dispense l'équipe du maintien d'une infrastructure CI/CD interne.

Automatiser un pipeline CI/CD permet de tester automatiquement chaque changement qui entre en production et limite considérablement les risques à toutes les étapes jusqu'à l'entrée en production.

Côté fonctionnement, CI/CD ‘as a Service’ s’intègre au service de référentiel de votre choix (GitHub, Bitbucket, GitLab, Azure DevOps, etc.).

À chaque fois que les développeurs poussent du code sur ce référentiel, la solution prend le relais, exécute les pipelines CI/CD et exécute les outils de build et de test choisis avant de déployer les modifications de code en production.

Contrairement à ce qu’on peut entendre ici ou là, il est donc primordial d’intégrer totalement et au plus tôt toutes les équipes dans un projet de migration vers une plateforme managée, des équipes de sécurité aux finances, en passant par la gouvernance et les équipes d’exploitation.

Chacune a sa place, peut y apprendre de nouveaux concepts, de nouvelles méthodes de travail et collaborer avec les autres pour mener un projet qui doit toujours impliquer toute l’entreprise.

“Faire usage de plateforme proposant des services managés de CI/CD est une bonne chose : l’ensemble des parties prenantes n’a plus alors qu’à se concentrer sur ce qui apporte le plus de valeur, c’est-à-dire livrer le plus vite possible et le mieux possible une nouvelle version. Il faut cependant rester prudent à plusieurs titres.

L’emploi de ce type de plateforme peut rapidement faire oublier les notions de sécurité et de coûts. Lorsqu’une plateforme d’intégration et de déploiement continus est gérée en interne par des équipes d’exploitation, ces sujets sont souvent mieux maîtrisés.

Dès lors qu’on transfert ces services à des plateformes managés, les équipes peuvent avoir tendance à omettre ou délaisser, au moins dans un premier temps, jusqu’à ce que la réalité de la production les rattrape, ces points pourtant essentiels.”

Micaël Vanhalst – Expert DevOps chez SoftFluent

Les auteurs

Christine Moronval

Responsable Communication chez Volkswagen puis Directrice de Clientèle dans le groupe DDB, Christine rejoint SoftFluent avec plus de 20ans d'expérience en marketing opérationnel et Communication. Responsable des opérations et de la communication Corporate, elle contribue également au Marketing de contenu de SoftFluent mais aussi des filiales SoftFluent Digital et SoftFluent Software.

Micaël Vanhalst

Après s'être impliqué dans un projet de Startup comme développeur Backend, où il a pu développer son intérêt et ses compétences sur Azure DevOps, Micaël Vanhalst a intégré la société SoftFluent.
Consultant expérimenté et expert DevOps, Micaël accompagne aujourd'hui nos clients dans la mise en place de leur démarche DevOps.

Fondée il y a 16 ans par d'anciens consultants Microsoft, SoftFluent c'est une équipe d'experts reconnus sur les technologies Microsoft, la qualité logicielle et le développement d'applications métiers. SoftFluent intervient auprès de grands comptes, ETI, éditeurs ou start-up sur des prestations de services pointues : logiciels sur mesure, migration vers le Cloud Azure, audit applicatif, méthodologies agiles, processus DevOps.

SoftFluent a également à cœur de s'engager pour des causes humanitaires et de mettre ses compétences au service de la santé et s'engage auprès d'acteurs importants dans le secteur comme Stago ou EIG. SoftFluent a également collaboré avec le Professeur Pierre Kalfon des hôpitaux de Chartres pour développer un outil numérique permettant d'améliorer durablement le confort en réanimation.

Grâce à l'expérience de ses consultants en architecture applicative et en développement sur de nombreuses solutions multi-composants de type web/webservices, mobiles ou client-riche connectés dans des environnements complexes tant en fonctionnement On Premise que Microsoft Azure, SoftFluent peut vous accompagner pour l'ensemble de vos projets de développement dans notre centre de services ou en assistance technique pour étoffer votre équipe.

Au-delà de son activité d'ESN innovante, SoftFluent a enrichi son offre d'un savoir-faire en transformation digitale et en édition de logiciels de productivité, avec notamment 2 offres novatrices portées par ses filiales : RowShare, un tableau collaboratif en ligne et le Mur Digital interactif, une solution matérielle et logicielle clé en main.

NOUS CONTACTER

Vous avez un projet ?

Contactez-nous

projet@softfluent.com

www.softfluent.fr
5, rue de la Renaissance 92160 Antony - +33 1 75 60 04 45