Engineer

DevOps

∞ pratiques unissant développement logiciel et exploitation

Ce que c’est

Le DevOps organise le travail logiciel de façon que les personnes qui écrivent un système répondent aussi de sa construction, de sa mise à disposition et de son maintien en fonctionnement. Les étapes entre l’écriture du code et son exécution sont automatisées au lieu d’être transmises d’une équipe à l’autre. Le nom associe development et operations, soit le développement et l’exploitation, les deux groupes que ce terme visait à réunir.

Le NIST, l’Institut national des normes et de la technologie des États-Unis, le définit comme des pratiques d’automatisation des processus entre les équipes de développement et d’exploitation « afin qu’elles puissent construire, tester et mettre à disposition des logiciels plus vite et de façon plus fiable ». Cette définition est utile, car elle nomme la partie décisive : les processus entre les deux. Une entreprise peut avoir d’excellents développeurs et d’excellentes équipes d’exploitation, et perdre tout de même des jours au moment où les uns transmettent une version aux autres.

En pratique, le DevOps apparaît sous la forme de quelques habitudes qui se renforcent mutuellement :

- **Une responsabilité partagée de la production.** Quiconque modifie le code voit les effets de cette modification en fonctionnement, grâce à la supervision, aux journaux et aux alertes, et en répond.
- **Un chemin automatisé vers la mise à disposition.** Construction, tests et déploiement s’exécutent dans un pipeline, de la même façon chaque fois, généralement sur une base de CI.
- **Une infrastructure écrite dans des fichiers.** Serveurs, réseaux et bases de données sont décrits dans des modèles ou des playbooks conservés à côté du code, pour pouvoir reconstruire un environnement plutôt que devoir s’en souvenir.

- **Une construction, plusieurs environnements.** L'artefact ayant passé ses tests en préproduction est celui qui arrive en production ; seuls ses paramètres changent.
- **De petites mises à disposition fréquentes.** Chacune contient peu de changements, ce qui facilite la recherche d'un défaut et le retour arrière.

Le DevOps n'est pas un produit ; son emploi comme intitulé de poste reste imprécis. Des fournisseurs vendent des outils sous ce nom et des entreprises recrutent des « ingénieurs DevOps », mais ni l'un ni l'autre ne constitue la pratique elle-même. Un ingénieur DevOps construit et entretient généralement les pipelines et l'infrastructure qui permettent aux autres développeurs de livrer seuls. C'est un rôle au sein de la pratique, qui ne la remplace pas.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce que le DevOps ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Des changements testés, publiés et réversibles, pour livrer aux clients un mardi après-midi plutôt qu'un vendredi soir tard.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. Déploiement SaaS GIS automatisé : GitHub Actions — déploiement, données et reporting automatisés

Déploiement SaaS GIS automatisé : GitHub Actions Contactez-nous →

Pourquoi c'est important

Chaque passage de relais entre équipes entraîne une perte d'information. Le développeur sait pourquoi une modification a été faite et de quoi elle dépend ; l'équipe d'exploitation connaît l'état des serveurs et ce qui a cassé le mois dernier. Quand une version passe de l'un à l'autre sous forme de ticket et de paquet, les deux moitiés de ce savoir arrivent incomplètes. Les lacunes apparaissent pendant le déploiement, au moment où leur découverte coûte le plus cher.

La méthode Twelve-Factor App, consacrée à la construction d'applications proposées sous forme de service, nomme précisément trois de ces écarts. Un écart de temps : le code écrit aujourd'hui atteint la production des semaines plus tard. Un écart de personnes : un groupe écrit le code et un autre le déploie. Un écart d'outils : un portable utilise une base de données et la production une autre. Le DevOps cherche en grande partie à combler les trois : déployer quelques heures après l'écriture, laisser les auteurs déployer et garder les environnements aussi semblables que possible.

La taille des versions est le deuxième argument. Le livre de Google sur l'ingénierie de fiabilité des sites le dit simplement : des mises à disposition fréquentes signifient moins de modifications entre les versions. Une version contenant trois modifications est facile à analyser lorsqu'elle fonctionne mal. Une version en contenant trois cents impose une recherche.

Le troisième est le travail opérationnel répétitif, ou toil. Le même livre le définit comme un travail lié au fonctionnement d'un service, manuel, répétitif, automatisable, sans valeur durable et qui croît avec le service. Une équipe qui redémarre manuellement un processus tous les mardis, renouvelle des certificats à partir d'un rappel de calendrier et agrandit les disques lorsqu'une alerte sonne consacre son temps à un travail qu'un script ferait plus sûrement. L'automatisation permet à une petite équipe de suivre la croissance du système.

Les recommandations de sécurité ont adopté la même structure. Les recommandations du NIST sur le DevSecOps pour les applications natives du cloud partent d'un pipeline de construction, de test, d'empaquetage, de déploiement et d'exploitation. Elles comptent l'infrastructure, la politique appliquée à l'exécution et la supervision parmi les formes de code décrites par ce pipeline.

À quoi cela ressemble en pratique

Les pages de ce site présentent moins le DevOps comme un intitulé de poste que comme un périmètre de responsabilité : la même personne conçoit un système, le met en production et l'y maintient.

Pour un produit cartographique, le travail de la fondatrice a couvert pendant deux ans l'infrastructure, le traitement des données et l'application cartographique destinée aux clients. Il englobait l'architecture, l'implémentation et l'assistance sur ces couches : concevoir, construire et maintenir l'environnement d'exploitation, les flux de traitement des données et l'application cartographique. Cette mission relie développement et exploitation. Le dossier établit le périmètre et la durée de ce travail d'ingénierie et de maintenance, sans établir de mesures de disponibilité ou de temps de réponse.

Sur un parc de 10 instances PostgreSQL et MS SQL Server sur des VPS Ubuntu Linux, l'exploitation a été automatisée progressivement : sauvegardes par `pg_dump` et tâches SQL Server Agent, supervision dans Prometheus et Grafana, correctifs réguliers, pare-feu UFW, contrôle d'accès fondé sur les rôles et reprise

après sinistre documentée. La page rapporte une disponibilité de 99,9 % sur les dix serveurs et une maintenance manuelle réduite de 50 % par l'automatisation : l'argument du travail répétitif mesuré sur un parc précis.

Sur une plateforme Azure avec des environnements de développement, de préproduction et de production, le déploiement était pris en charge de bout en bout selon un itinéraire fixe. Cette responsabilité portait sur chaque maillon : la CI/CD construit les images, les modèles Bicep assurent une infrastructure correspondante à chaque étape et les paramètres propres à chacune sont conservés à part des secrets. Un même artefact construit est donc promu d'une étape à l'autre au lieu d'être reconstruit pour chacune. Les modifications atteignent chaque environnement de manière prévisible, sans aucun déploiement manuel improvisé.

Où les problèmes surviennent

Le DevOps échoue discrètement, généralement en conservant le vocabulaire et en perdant les habitudes.

Le titre arrive, le mur reste. Une équipe d'exploitation est renommée, ou une « équipe DevOps » est recrutée pour s'intercaler entre développeurs et production. Les versions franchissent toujours un passage de relais ; seul le nom de l'autre côté change. Le test est simple : demandez qui peut livrer une modification et qui est réveillé lorsqu'elle casse. Si ce sont des personnes différentes disposant d'informations différentes, la pratique n'est pas encore là.

Déployer revient toujours à suivre une liste à la main. Le pipeline construit et teste, puis quelqu'un se connecte au serveur et exécute de mémoire les cinq dernières étapes, un peu différemment chaque fois. Ajoutez aussi ces dernières étapes au pipeline, donnez aux versions un seul itinéraire entre les environnements et traitez une publication manuelle en production comme un incident à expliquer.

Chaque environnement reçoit sa propre construction. Reconstruire pour la préproduction puis pour la production signifie que ce qui a été testé n'est pas ce qui a été livré. Une dépendance résolue différemment à la seconde construction devient un défaut que personne ne peut reproduire. Construisez une fois, étiquetez l'artefact avec son commit d'origine et promouvez cet artefact, en laissant l'environnement fournir les paramètres :

```
tag="$(git rev-parse --short HEAD)"
docker build -t "registry.example.com/app:$tag" .
docker push "registry.example.com/app:$tag"
# staging and production run this same tag;
# only the settings they start it with differ
docker run --env-file staging.env "registry.example.com/app:$tag"
```

La préproduction s'éloigne de la production. Un paramètre modifié à la main sur un serveur, une base mise à niveau d'abord en production, une version de bibliothèque fixée ici et flottante ailleurs : chacun de ces écarts rend la préproduction un peu moins utile comme répétition. Décrivez l'infrastructure dans des modèles, appliquez les mêmes partout et utilisez en développement les mêmes services d'appui qu'en production.

Le travail répétitif croît avec le système. Des tâches manuelles prenant dix minutes par semaine pour un serveur demandent des heures pour vingt. Sans budget explicite pour les automatiser, l'équipe finit par passer ses journées à maintenir le fonctionnement courant. Comptez le travail manuel récurrent et plafonnez la part de temps qu'il peut prendre.

Pour aller plus loin

- DevOps — la définition du NIST et les publications spéciales dont elle est tirée.
- Construction, préparation de version, exécution — pourquoi construire une seule fois puis associer la construction à la configuration pour former une version.
- Parité développement/production — les écarts de temps, de personnes et d'outils entre développement et production, et comment les combler.
- Configuration — garder hors du code, dans l'environnement, tout ce qui varie entre les déploiements.
- Éliminer le travail opérationnel répétitif — la définition du toil par Google et l'argumentaire pour le plafonner.
- Ingénierie des mises à disposition — constructions hermétiques, mises à disposition en libre-service et politiques imposées par les outils de livraison.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le parcours de déploiement, les risques opérationnels et les personnes d'astreinte. Nous pouvons définir un changement avec des vérifications, une procédure de reprise et une responsabilité claire.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce que le DevOps ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

welcome@engineer.company +45 50 10 61 84 Copenhague, Danemark

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/devops/