LOG680
Introduction à l'approche DevOps

Activer la rétroaction afin que le développement et les opérations puissent déployer le code en toute sécurité

The DevOps Handbook

Part IV, Chap 16

Francis Bordeleau, 2021

Objectifs d'apprentissage

- Expliquer en quoi consiste la technique de recherche contextuelle ("contextual inquiry") utilisée pour la conception d'interaction et d'expérience utilisateur (UX)
- Expliquer quelles sont les trois solutions qui peuvent être utilisées pour rétablir rapidement le service lorsque des problèmes sont détectés par la télémétrie de production
- Expliquer en quoi consiste le mécanisme de transfert de service ("service handback mechanism") chez Google
- Expliquer en consiste le concept de "Site Reliability Engineer" (SRE) chez Google. Quel est le rôle d'un(e) SRE?
- Expliquer en quoi consiste les deux listes de contrôle utilisé chez Google pour les deux étapes critiques de la mise en service de nouveaux services afin de garantir que ces équipes de produits autogérées puissent toujours bénéficier de l'expérience collective de l'organisation SRE

Sujets

- Introduction
- Utilisation de la télémétrie
- Partage des responsabilités de pagette
- Suivi des travaux en aval
- Autogestion des services de production
- Conclusion

- **Introduction**
- Utilisation de la télémétrie
- Partage des responsabilités de pagette
- Suivi des travaux en aval
- Autogestion des services de production
- Conclusion

Exemple Right Media

- En 2006, Nick Galbreath (VP ingénierie chez Right Media) était responsable des services de développement et des opérations pour une plate-forme de publicité en ligne affichant et diffusant plus de dix milliards d'impressions par jour. Il a décrit le paysage concurrentiel dans lequel elle opérait:

  « Dans notre secteur d'activité, les niveaux des inventaires d'annonces étaient extrêmement dynamiques, nous devions réagir aux conditions du marché en quelques minutes.

  Cela signifiait que **Dev devait être en mesure de modifier rapidement le code et de le mettre en production dès que possible**, faute de quoi nous perdrions face à des concurrents plus rapides.

  Nous avons constaté qu'**avoir un groupe séparé pour les tests, et même le déploiement, était tout simplement trop lent**.

  Nous **devions intégrer toutes ces fonctions dans un groupe, avec des responsabilités et des objectifs partagés**.

  Croyez-le ou non, notre plus grand défi consistait à **amener les développeurs à surmonter leur peur de déployer leur propre code**! »

Exemple Right Media

Crainte de déployer

- Il y a une ironie intéressante ici:
  - Dev se plaint souvent de ce que Ops ait peur de déployer du code
  - Mais dans ce cas, lorsqu'ils ont eu le pouvoir de déployer leur propre code, les développeurs ont tout aussi peur de procéder à des déploiements de code

- La **crainte de déployer du code partagé par les développeurs et les opérateurs** de Right Media n'est pas inhabituelle
  - Galbreath: « **fournir un retour plus rapide et plus fréquent** aux ingénieurs effectuant des déploiements (qu'il s'agisse de développeurs ou d'opérateurs), ainsi que de **réduire la taille des lots de travail**, **créait de la sécurité, puis de la confiance**. »

Exemple Right Media

- **Solution: Ajouter davantage de télémétrie de production à nos applications et à notre environnement**
  - Permet de réellement confirmer le nouveau code déployé fonctionne correctement
  - Permet de détecter les problèmes avant les clients
- Galbreath observe cette solution profite à tout le monde (Dev, Ops, et Infosec)

  « En tant que responsable de la sécurité, **il est rassurant de savoir que nous pouvons rapidement déployer des correctifs**, car nous devons constamment le faire. De plus, je suis toujours étonné de voir à quel point les ingénieurs s'intéressent à la sécurité lorsque nous trouvons des problèmes dans leur code et qu'ils peuvent rapidement les résoudre eux-mêmes. »
- L'histoire de Right Media montre qu'**il ne suffit pas d'automatiser le processus de déploiement**
- Nous devons également **intégrer le contrôle de la télémétrie de production** à notre travail de déploiement et **établir des normes culturelles** selon lesquelles tout le monde est également responsable de la santé de toute la chaine de valeur

Introduction

- Dans ce chapitre, nous créons les **mécanismes de retour d'information** qui nous permettent d'**améliorer la santé de la chaîne de valeur à chaque étape du cycle de vie du service**, de la conception du produit au développement et au déploiement, en passant par l'exploitation et, éventuellement, le retrait

- Ce faisant, nous nous assurons que nos services sont **« prêts pour la production »**, même aux premiers stades du projet, et nous **intégrons les enseignements tirés de chaque problème de "release" et de production dans nos travaux futurs**, ce qui se traduit par une sécurité et une productivité accrues pour tous

- Introduction
- **Utilisation de la télémétrie**
- Partage des responsabilités de pagette
- Suivi des travaux en aval
- Autogestion des services de production
- Conclusion

Utilisation de la télémétrie

- Objectif: **surveiller activement notre télémétrie de production lorsque quiconque effectue un déploiement de production**, comme illustré dans l'article de Right Media
  - **Permet à quiconque qui effectue un déploiement**, qu'il s'agisse de développeurs ou d'opérateurs, **de déterminer rapidement si les fonctionnalités fonctionnent comme prévu** après la publication du nouveau release en production
  - Nous ne devrions jamais considérer que le déploiement de notre code est complété tant qu'il ne fonctionne pas comme prévu dans l'environnement de production
- Pour ce faire, nous **surveillons activement les métriques associées à notre fonctionnalité** lors de notre déploiement afin de nous assurer que nous n'avons pas interrompu notre service par inadvertance ou, pire, que nous avons interrompu un autre service
  - Si notre modification casse ou altère une fonctionnalité, nous travaillons rapidement à la restauration du service, en faisant appel à toute autre personne chargée de diagnostiquer et de résoudre le problème

Utilisation de la télémétrie

- Notre objectif est de détecter les erreurs dans notre pipeline de déploiement avant leur mise en production
- Cependant, **il y aura toujours des erreurs que nous ne détectons pas** et **nous comptons sur la télémétrie de production pour rétablir rapidement le service**
- Si des problèmes sont détectés, nous pouvons choisir de
  - **Désactiver les fonctionnalités** endommagées avec des options – ce qui est souvent l'option la plus simple et la moins risquée car elle n'implique aucun déploiement en production, ou
  - **Corriger** – c'est-à-dire d'apporter des modifications de code pour corriger le défaut, qui sont ensuite mises en production via le pipeline de déploiement, ou
  - **Revenir en arrière ("roll back")** – par exemple, revenir à la version précédente en utilisant des options ou en supprimant la rotation des serveurs endommagés à l'aide des modèles de version bleu-vert ou canary, etc.
- Bien que réparer puisse s'avérer dangereux, **la réparation peut se faire de façon sécuritaire si on dispose de tests automatisés et de processus de déploiement rapides**, **ainsi que d'une télémétrie suffisante** pour nous permettre de confirmer rapidement si tout fonctionne correctement en production

Utilisation de la télémétrie

- La figure 37 illustre un déploiement de modification de code PHP chez Etsy qui a généré une pointe d'avertissements d'exécution PHP
- Dans ce cas, le développeur a rapidement remarqué le problème en quelques minutes, puis généré un correctif et l'a déployé en production, le résolvant en moins de dix minutes

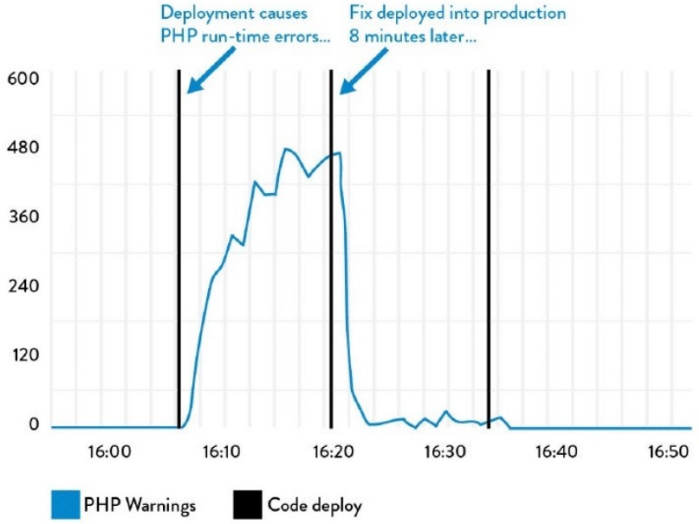

Figure 37: *Deployment to Etsy.com causes PHP run-time warnings and is quickly fixed (Source: Mike Brittain, "Tracking Every Release.")*

- Les déploiements en production étant l'une des principales causes des problèmes de production, **chaque événement de déploiement et de modification est superposé à nos graphiques de mesures** afin de garantir que tous les utilisateurs de la chaîne de valeur sont au courant de l'activité
  - Permet une meilleure communication et coordination, ainsi qu'une détection et récupération plus rapides

- Introduction
- Utilisation de la télémétrie
- **Partage des responsabilités de pagette**
- Suivi des travaux en aval
- Autogestion des services de production
- Conclusion

Partage des responsabilités de pagette

- Même lorsque nos déploiements et versions de production ("releases") se déroulent sans faille, dans tout environnement de services complexes, il y aura toujours des problèmes inattendus, tels que des incidents et des pannes survenant à un moment peu opportun (e.g. 02h00)
  - Si ils ne sont pas corrigées, ils peuvent être à l'origine de problèmes récurrents et de difficultés pour les ingénieurs d'Ops en aval, notamment lorsque ces problèmes ne sont pas visibles pour les ingénieurs en amont, qui sont responsables de la création du problème
- Même si le problème résulte en un défaut attribué à l'équipe technique, il peut être priorisé à un niveau inférieur au développement de nouvelles fonctions
  - Le problème peut persister pendant des semaines, des mois, voire des années, provoquant un chaos continu et des perturbations dans les opérations
  - Ceci est un exemple de la façon dont les centres de travail en amont peuvent optimiser localement pour eux-mêmes mais dégrader les performances pour l'ensemble du flux de valeur

Partage des responsabilités de pagette

- Pour éviter cela, tous les acteurs de la chaîne de valeur se partageront les responsabilités en aval du traitement des incidents opérationnels
  - Nous pouvons y parvenir en mettant les développeurs, les responsables du développement ("development managers") et les architectes en rotation de pagettes, comme l'a fait Pedro Canahuati (directeur de l'ingénierie de la production de Facebook) en 2009
  - Cela permet à tous les acteurs de la chaîne de valeur d'obtenir un retour d'information viscéral sur leurs décisions en amont en matière d'architecture et de codage
- En agissant de la sorte, les opérations ne sont pas seules et isolées face aux problèmes de production liés au code
- Chacun cherche plutôt à **trouver le juste équilibre entre la résolution des problèmes de production et le développement de nouvelles fonctionnalités**, quel que soit notre rôle dans la chaîne de valeur
  - Comme l'observait Patrick Lightbody (vice-président directeur de la gestion des produits chez New Relic) en 2011,

    «nous avons constaté que lorsque nous réveillons les développeurs à 2 heures du matin, les défauts étaient corrigés plus rapidement que jamais»

Partage des responsabilités de pagette

- L'un des effets secondaires de cette pratique est qu'elle aide la direction du développement à réaliser que **les objectifs commerciaux ne sont pas atteints simplement parce que les fonctionnalités sont marquées comme «terminées»**
  - Au lieu de cela, elles ne le sont que lorsqu'elles fonctionnent comme prévu en production, sans provoquer d'escalades excessives ou travail non planifié pour le développement ou les opérations
- Cette pratique s'applique également aux équipes orientées marché, chargées du développement de la fonction et de son exploitation en production, ainsi qu'aux équipes axées sur les fonctions
  - Comme l'a observé Arup Chakrabarti (responsable de l'ingénierie des opérations chez PagerDuty) lors d'une présentation en 2014, «il est de moins en moins courant que les entreprises disposent d'équipes sur appel; à la place, tous ceux qui sont impliqués dans le code de production et les environnements devraient être joignables en cas de temps mort. »
- Quelle que soit la manière dont nous avons organisé nos équipes, les principes sous-jacents restent les mêmes: **lorsque les développeurs reçoivent des informations sur les performances de leurs applications en production**, incluant le fait d'avoir à faire des corrections lorsque des problèmes se produisent, **ils se rapprochent des clients**, ce qui crée un intérêt dont tout le monde dans le flux de valeur bénéficie

- Introduction
- Utilisation de la télémétrie
- Partage des responsabilités de pagette
- **Suivi des travaux en aval**
- Autogestion des services de production
- Conclusion

Suivi des travaux en aval

- L'une des techniques les plus puissantes en matière de conception d'interaction et d'expérience utilisateur (UX) est la **recherche contextuelle ("contextual inquiry")**.
  - **Permet à l'équipe produit d'observer l'utilisation de l'application par les utilisateurs dans leur environnement naturel**, i.e. travaillant à leur poste de travail
  - **Permet de découvrir et comprendre des difficultés rencontrées par les utilisateurs** (lors de l'utilisation de l'application), par exemple en exigeant des clics pour effectuer des tâches simples dans leur travail quotidien, couper et coller du texte à partir de plusieurs écrans ou écrire des notes sur papier

- La réaction la plus courante chez les développeurs après avoir participé à une observation client est la consternation, affirmant souvent à quel point «il était affreux de voir les nombreuses façons dont nous infligions des souffrances à nos clients»
  - Ces observations des clients aboutissent presque toujours à un apprentissage important et à un vif désir de améliorer la situation pour le client

Suivi des travaux en aval

- **Notre objectif est d'utiliser cette même technique pour observer l'impact de notre travail sur nos clients internes**
  - Les développeurs doivent suivre leur travail en aval afin de pouvoir voir comment les centres de travail en aval doivent interagir avec leur produit pour le rendre opérationnel
- Les développeurs doivent suivre leur travail en aval
  - En voyant les difficultés rencontrées par les clients, ils prennent de meilleures décisions, en toute connaissance de cause, dans leur travail quotidien
- **Permet de créer une rétroaction sur les aspects non fonctionnels de notre code** et nous permet d'identifier des moyens d'améliorer la déployabilité, la gérabilité, l'opérabilité, etc.

Suivi des travaux en aval

- L'observation UX a souvent un impact puissant sur les observateurs
- Gene Kim (fondateur et CTO de Tripwire pendant treize ans et co-auteur de ce livre):

  « L'un des pires moments de ma carrière professionnelle a été l'année 2006, lorsque j'ai passé toute la matinée à regarder un de nos clients utiliser notre produit.

  Je le regardais effectuer une opération que, selon notre modèle d'utilisation, nos clients devaient effectuer toutes les semaines et, à ma plus grande surprise (horreur!), nous avons découvert qu'il fallait soixante-trois clics.

  Cette personne ne cessait de s'excuser en disant des choses comme: «Désolé, il existe probablement une meilleure façon de procéder. »
- Malheureusement, il n'y avait pas de meilleure façon de faire cette opération

Suivi des travaux en aval

- Un autre client a décrit la configuration initiale du produit en 1 300 étapes
  - Tout à coup, j'ai compris pourquoi le travail de gestion de notre produit était toujours attribué au dernier ingénieur de l'équipe - personne ne voulait le travail de gestion de notre produit
  - C'est l'une des raisons pour lesquelles j'ai aidé à créer la pratique UX dans mon entreprise, pour aider à atténuer la douleur que nous causions à nos clients

- **L'observation UX permet de créer de la qualité à la source et génère une empathie bien plus grande pour les membres de l'équipe dans le flux de valeur**
  - Dans l'idéal, l'observation UX nous aide dans la création d'exigences non fonctionnelles codifiées qui viennent s'ajouter à notre arriéré de travail commun, nous permettant par la suite de les intégrer de manière proactive à chaque service que nous construisons, ce qui constitue un élément important de la création d'une culture de travail DevOps

- Introduction
- Utilisation de la télémétrie
- Partage des responsabilités de pagette
- Suivi des travaux en aval
- **Autogestion des services de production**
- Conclusion

Autogestion des services de production

- **Même lorsque les développeurs écrivent et exécutent leur code dans des environnements de production au quotidien, les opérations peuvent toujours connaître des "releases" (en production) désastreuses**, car **c'est la première fois que nous voyons comment notre code se comporte lors d'un publication et dans de véritables conditions de production**
  - Ce résultat est dû au fait que les apprentissages opérationnels ont souvent lieu trop tard dans le cycle de vie du logiciel
- Si rien n'est fait, il en résulte souvent des logiciels de production difficiles à utiliser

Autogestion des services de production

- Une solution possible consiste à faire ce que **Google** fait, à savoir que les **groupes de développement gèrent eux-mêmes leurs services en production avant de pouvoir prétendre à la gestion d'un groupe d'opérations centralisé**
  - Si les développeurs sont responsables du déploiement et de la prise en charge de la production, nous sommes beaucoup plus susceptibles d'effectuer une transition en douceur vers les opérations
- **LLR: Revue de la préparation au lancement – "Launch Readiness Review"**
- **HRR: Revue de la préparation au transfert – "Hand-off Readiness Review"**

Étude de cas : Google (2010)

- Afin de garantir que ces équipes de produits autogérées puissent toujours bénéficier de l'expérience collective de l'organisation SRE, Google a créé deux listes de contrôles pour les deux étapes critiques de la mise en service de nouveaux services:

- LLR: Revue de la préparation au lancement – "Launch Readiness Review"
  - Le LRR doit être exécuté et approuvé avant que tout nouveau service Google ne soit rendu public et reçoive un trafic de production en direct
- HRR: Revue de la préparation au transfert – "Hand-off Readiness Review"
  - Le HRR est exécuté lorsque le service passe à un état géré par Ops, généralement plusieurs mois après le LRR

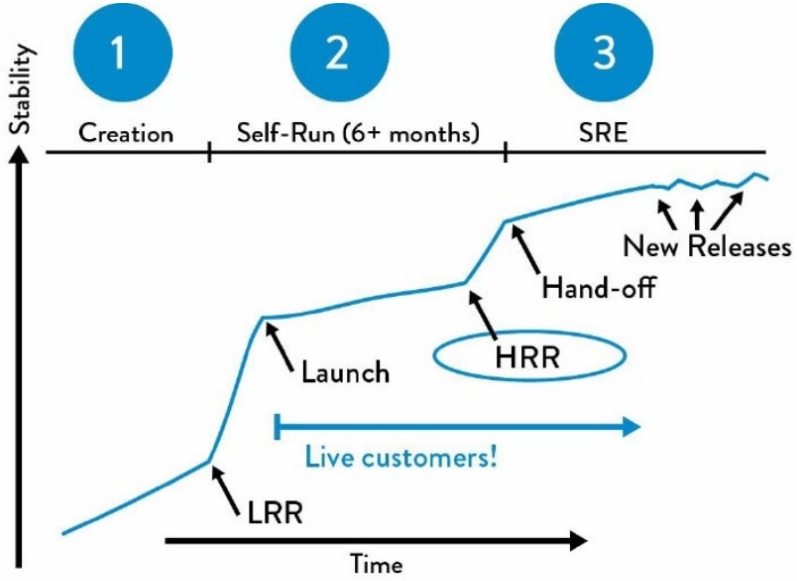

Figure 39: *The "Launch readiness review and hand-offs readiness review" at Google (Source: "SRE@Google: Thousands of DevOps Since 2004," YouTube video, 45:57, posted by USENIX, January 12, 2012, https://www.youtube.com/watch?v=iIuTnhdTzK0 .)*

Transfert de service ("handback")

- Sert de soupape d'échappement de pression, garantissant que nous ne mettons jamais les Opérations dans une situation où elles sont prises au piège de la gestion d'un service fragile alors qu'une dette technique toujours croissante les enterre et amplifie un problème local en un problème global
- Permet de garantir que Ops dispose de capacité suffisante pour travailler sur des travaux d'amélioration et des projets préventifs
- Constitue peut-être l'une des meilleures démonstrations du respect mutuel entre les ingénieurs de Dev et d'Ops
  - De cette manière, Dev est en mesure de générer rapidement de nouveaux services
  - Les ingénieurs d'Ops rejoignent l'équipe lorsque les services revêtent une importance stratégique pour la société
  - Dans de rares cas, ils les retournent lorsque la gestion de la production devient trop compliquée
- Pratique de longue date chez Google

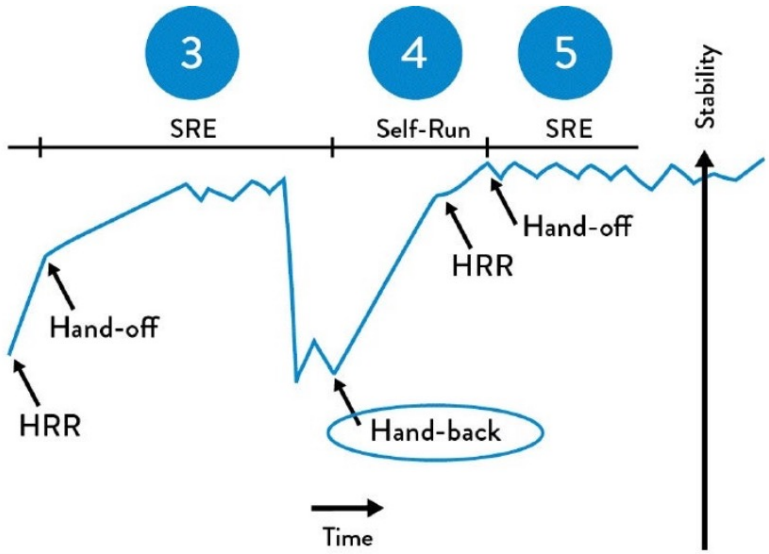

Figure 38: *The "Service Handback" at Google (Source: "SRE@Google: Thousands of DevOps Since 2004," YouTube video, 45:57, posted by USENIX, January 12, 2012, https://www.youtube.com/watch?v=iIuTnhdTzK0.)*

- Introduction
- Utilisation de la télémétrie
- Partage des responsabilités de pagette
- Suivi des travaux en aval
- Autogestion des services de production
- **Conclusion**

Conclusion

- Dans ce chapitre, nous avons discuté des **mécanismes de retour qui nous permettent d'améliorer notre service à chaque étape de notre travail quotidien** en **laissant les développeurs suivre leur travail en aval** et en **créant des exigences non fonctionnelles** qui aident les équipes de développement à écrire davantage de code prêt pour la production, voire en remettant des services problématiques à l'autogestion par Dev

- **En créant ces boucles de rétroaction, nous rendons les déploiements de production plus sûrs**, **augmentons la capacité de production du code créé par Dev** et **aidons à créer une meilleure relation de travail entre Dev et Ops** en renforçant des objectifs, des responsabilités et une empathie communs