LOG680
Introduction à l'approche DevOps

Automatiser et activer les versions à faible risque

The Dev Ops Handbook

Part III, Chap 12

Objectifs d'apprentissage

- Expliquer en quoi consiste le découplage des déploiements des publications (releases). Pourquoi est-il important de découpler ces deux aspects?
- Expliquer les deux grandes catégories de modèles de publication. Expliquer les caractéristiques principales de chacune.
- Expliquer en quoi consiste la publication basée sur l'environnement.
- Expliquer en quoi consiste la publication basée sur les applications.
- Identifier et expliquer trois modèles de publications basées sur l'environnement.
- Expliquer le modèle/patron de déploiement bleu-vert (blue-green). Quels sont ses avantages et inconvénients.
- Expliquer en quoi consiste la publication canarie ("Canary release").
- Expliquer le principe de bascule de fonctionnalité ("feature toggle").
- Expliquer le principe de lancement sombre ("dark launch").

Sujets

- Introduction
- Automatisation du processus de déploiement
  - Déploiements automatisés en libre-service
  - Intégration dans le pipeline de déploiement
- Déploiements vs publications
  - Publications basées sur l'environnement
    - Déploiement bleu-vert (blue-green)
    - Publication canarie ("Canary release")
    - Système "Cluster Immune"
  - Publications basées sur les applications
    - Bascule de fonctionnalités
    - Lancements sombres ("dark launches")
- Conclusion

- **Introduction**
- Automatisation du processus de déploiement
  - Déploiements automatisés en libre-service
  - Intégration dans le pipeline de déploiement
- Déploiements vs publications
  - Publications basées sur l'environnement
    - Déploiement bleu-vert (blue-green)
    - Publication canarie ("Canary release")
    - Système "Cluster Immune"
  - Publications basées sur les applications
    - Bascule de fonctionnalités
    - Lancements sombres ("dark launches")
- Conclusion

Exemple Facebook

- En 2012, Chuck Rossi (Directeur de l'ingénierie de publication chez Facebook, responsable de superviser l'application quotidienne du code) a décrit leur processus de la manière suivante:

  «À partir de 13h00, je passe en mode "opérations" et je travaille avec mon équipe pour se préparer à lancer les modifications effectuées à Facebook.com du jour.

  C'est la partie la plus stressante du travail et elle dépend vraiment du jugement et de l'expérience de mon équipe.

  Nous travaillons pour faire en sorte que tous ceux qui ont effectué des changements soient pris en compte, et qu'ils testent et soutiennent activement leurs changements. »

- Juste avant la production, tous les développeurs dont les modifications ont été envoyées doivent être présents et se connecter à leur canal de discussion IRC.
  - Tous les développeurs non présents voient leurs modifications automatiquement supprimées du package de déploiement.

Exemple Facebook

- Rossi a ajouté:

  «Si tout semble bien aller et que nos tableaux de bord de test ("test dashboard) et tests Canarie sont verts,

  nous appuyons sur le gros bouton rouge et le parc entier de serveurs Facebook.com reçoit le nouveau code.

  En l'espace de vingt minutes, des milliers et des milliers de machines utilisent un nouveau code sans impact visible pour les utilisateurs du site. ”

- Plus tard cette année là, Rossi a doublé leur fréquence de publication de logiciel à deux fois par jour.
  - La seconde publication donnait aux ingénieurs, non résidant sur la côte ouest américaine, la possibilité de «travailler et d’expédier aussi rapidement que tout autre ingénieur de la société»,
  - Donnait également à chaque utilisateur une deuxième occasion chaque jour de publier du code et de lancer des fonctionnalités.

Exemple Facebook

- Kent Beck (créateur de la méthodologie Extreme Programming (XP), un des principaux promoteurs de TDD, et coach technique de Facebook),

  «Chuck Rossi a fait l'observation qu'il semble y avoir un nombre fixe de changements que Facebook peut gérer en un seul déploiement.

  Si nous voulons plus de changements, nous avons besoin de plus de déploiements.

  Cela a conduit à une augmentation régulière du rythme de déploiement au cours des cinq dernières années,

  passant de déploiements hebdomadaires à quotidiens à trois par jour de notre code PHP et de **cycles de six à quatre à deux semaines pour le déploiement de nos applications mobiles**.

  Cette amélioration a été principalement conduite par l'équipe d'**ingénierie de publication (Release Engineering**. »

- En utilisant l'intégration continue et en faisant du déploiement de code un processus à faible risque, Facebook a permis au déploiement de code de faire partie du travail quotidien de chacun et de maintenir la productivité des développeurs.
- **Cela nécessite que le déploiement du code soit automatisé, reproductible et prévisible.**

Exemple Facebook

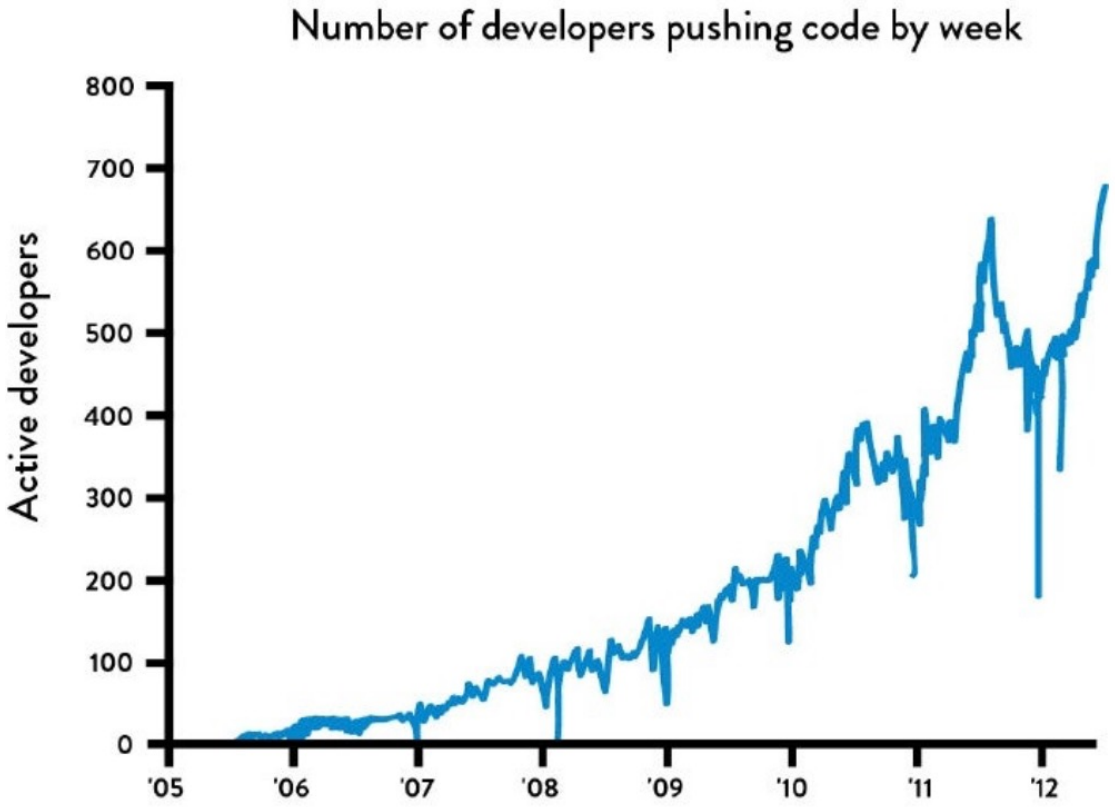

Figure 16 : *Number of developers deploying per week at Facebook*
(Source: Chuck Rossi, "Ship early and ship twice as often.")

- Introduction
- **Automatisation du processus de déploiement**
  - Déploiements automatisés en libre-service
  - Intégration dans le pipeline de déploiement
- Déploiements vs publications
  - Publications basées sur l'environnement
    - Déploiement bleu-vert (blue-green)
    - Publication canarie ("Canary release")
    - Système "Cluster Immune"
  - publications basées sur les applications
    - Bascule de fonctionnalités
    - Lancements sombres ("dark launches")
- Conclusion

Automatisation du processus de déploiement

- Dans les pratiques décrites jusqu'à présent dans le livre**, même si notre code et nos environnements ont été testés ensemble**, **il est fort probable que nous ne procédons pas au déploiement en production** car ceux-ci sont manuels, fastidieux, pénibles, et source d'erreurs**.**
  - Ils impliquent souvent un transfert incommode et peu fiable entre développement et opérations.
  - Parce que c'est douloureux, nous avons tendance à le faire de moins en moins souvent, ce qui entraîne une autre spirale descendante qui se renforce elle-même.
  - En reportant les déploiements en production, nous **accumulons des différences de plus en plus grandes** entre le code à déployer et ce qui est en cours d'exécution, ce qui **augmente la taille du lot de déploiement**.
  - **À mesure que la taille des lots de déploiement augmente**, **le risque de résultats inattendus associés au changement, ainsi que la difficulté à les corriger augmentent aussi**.
- Dans ce chapitre, nous **réduisons les frictions liées aux déploiements en production**, **en nous assurant qu'ils peuvent être effectués fréquemment et facilement**, que ce soit par rapport aux Opérations ou au Développement.
  - Nous faisons cela **en élargissant notre pipeline de déploiement**.

Automatisation du processus de déploiement

- Pour obtenir des résultats tels que ceux de Facebook, nous devons disposer d'un **mécanisme automatisé qui déploie notre code en production**.

- Surtout si nous avons un processus de déploiement qui existe depuis plusieurs années, **nous devons documenter complètement les étapes du processus de déploiement, comme un exercice de cartographie des flux de valeur**
  - Nous pouvons le faire en organisant un atelier (workshop) ou le documenter progressivement (e.g. dans un wiki).

Automatisation du processus de déploiement

- **Une fois le processus documenté, notre objectif est de simplifier et d'automatiser autant d'étapes manuelles que possible**, telles que:
  - Packaging du code de manière appropriée pour le déploiement
  - Création d'images ou de conteneurs de machine virtuelle préconfigurés
  - Automatisation du déploiement et de la configuration du middleware
  - Copie des packages ou des fichiers sur des serveurs de production
  - Redémarrage de serveurs, d'applications ou de services
  - Génération des fichiers de configuration à partir de modèles (templates)
  - Exécution de tests de fumée ("smoke test") automatisés pour s'assurer que le système fonctionne et est correctement configuré
  - Exécution de procédures de test
  - Script et automatisation des migrations de bases de données

- Si possible, nous ferons une ré-architecture pour supprimer des étapes, en particulier celles qui prennent beaucoup de temps.
  - Nous souhaitons également non seulement réduire nos délais, mais également le nombre de transferts afin de réduire les erreurs et les pertes de connaissances.

Automatisation du processus de déploiement

- Le fait que les développeurs se concentrent sur **l'automatisation et l'optimisation du processus de déploiement peut conduire à des améliorations significatives du flux de déploiement**.

- De nombreux outils fournissant une intégration et des tests continus permettent d'étendre le pipeline de déploiement de sorte que les versions validées puissent être promues en production, généralement après les tests d'acceptation de production
  - Exemples: plug-in Jenkins Build Pipeline, ThoughtWorks Go.cd et Snap CI, Microsoft Visual Studio Team Services, et Pivotal Concourse).

Exigences du processus de déploiement

- **Déployer de la même manière dans tous les environnements**
  - Utiliser le même mécanisme (pipeline) de déploiement pour tous les environnements (dév, test et production)
  - Permet d'améliorer la qualité des déploiements
- **Test de fumée de nos déploiements** - pendant le processus de déploiement
  - Vérifier que nous pouvons nous connecter à n'importe quel système (BD, bus de messages, services externes, etc.)
  - Exécuter un test de transaction sur l'ensemble du système pour s'assurer que notre système fonctionne comme prévu
  - Si l'un de ces tests échoue, le déploiement doit échouer
- **S'assurer de maintenir des environnements cohérents**:
  - Utiliser un processus de création d'environnement en une seule étape afin que les environnements soient générés à partir d'un mécanisme de génération commun
  - Continuellement veiller à ce que les environnements (dév, test et production) restent synchronisés.
- Lorsque des problèmes surviennent pendant le déploiement, nous tirons le cordon Andon et essaimons le problème jusqu'à ce que le problème soit résolu.
  - Tout comme lorsque notre pipeline de déploiement échoue à l'une des étapes précédentes.

accelerate business. anywhere.

Étude de cas : CSG International (2013)

- Contexte
  - CSG International est l'une des plus importantes opérations d'impression de billets aux États-Unis.
  - Scott Prugh – architecte en chef et vice-président du développement
  - Bien que les équipes de développement utilisaient l'intégration continue pour déployer leur code dans des environnements de test quotidiennement, les versions de production étaient exécutées par l'équipe des opérations.

- Décision
  - **Dans le but d'améliorer la prévisibilité et la fiabilité des versions de leurs logiciels**, Prugh a décidé de **doubler la fréquence de publication de deux à quatre par an** – réduction de moitié de leur intervalle de déploiement de vingt-huit à quatorze semaines.

Étude de cas : CSG International (2013)

- Leurs **résultats étaient étonnants**. En effectuant des déploiements quotidiens et en doublant la fréquence des publications ("publication releases"),
  - le **nombre d'incidents de production a diminué de 91%**,
  - le **délai moyen de déploiement de 80%**, et
  - le **délai de déploiement requis pour que le service fonctionne en production** de manière totalement autonome est passé **de quatorze jours à un jour**.

- Outre le fait que les déploiements se déroulent plus facilement pour les développeurs et les opérateurs,
  - **dans 50% des cas, le client a reçu la valeur deux fois plus rapidement**
- Ceci démontre que l'augmentation des fréquence des déploiements a des effets bénéfiques pour Dev, QA, Ops, ainsi que pour les clients/utilisateurs

Étude de cas : CSG International (2013)

Release Impact Improvements
Incidents by Release

Release	Incidents	Impact	Improvement
2013-Apr	201	455	0% (1x)
2014-Apr	67	153	66% (3x)
2015-May	41	97	79% (5x)
2015-Aug	18	45	90% (10x)

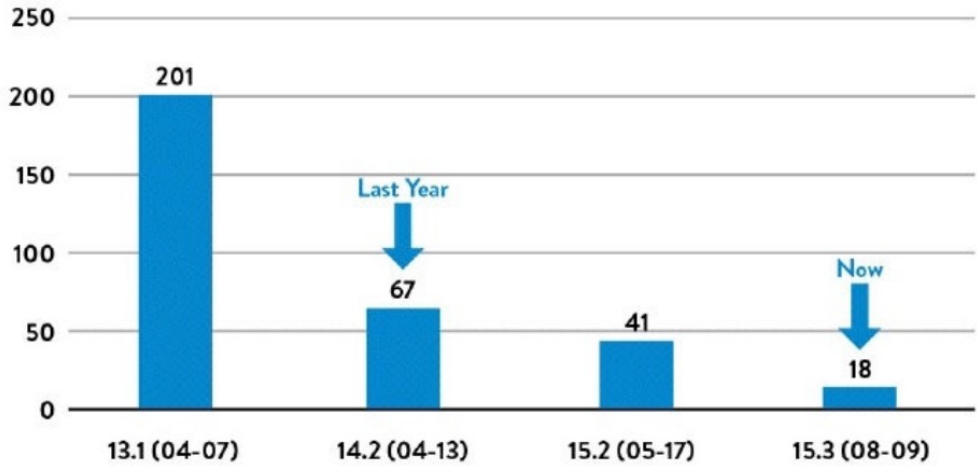

Figure 17: *Daily deployments and increasing release frequency resulted in decrease in # of production incidents and MTTR (Source: "DOES15 - Scott Prugh & Erica Morrison - Conway & Taylor Meet the Strangler (v2.0)," YouTube video, 29:39, posted by DevOps Enterprise Summit, November 5, 2015, https://www.youtube.com/watch?v=tKdIHCL0DUg .)*

Déploiements automatisés

Déploiements automatisés en libre-service

- Tim Tischler (directeur de l'automatisation des opérations chez Nike, Inc.) décrit l'expérience commune d'une génération de développeurs:

  «En tant que développeur, ma carrière n'a jamais été aussi satisfaisante que lorsque j'écrivait du code,

  lorsque j'appuyais sur le bouton pour le déployer,

  lorsque je pouvais voir les métriques de production confirmer que cela fonctionnait réellement en production, et

  quand je pouvais le réparer moi-même si le code ne fonctionnait pas. "

Déploiements automatisés en libre-service

- La capacité des développeurs à effectuer les tâches suivantes a été réduite de façon significative au cours de la dernière décennie.
  - déployer eux-mêmes du code en production,
  - à voir rapidement les clients satisfaits lorsque leur fonctionnalité est opérationnelle et
  - à résoudre rapidement les problèmes sans devoir ouvrir un ticket avec les Opérations

- La **pratique commune** qui en résulte est que
  - **Ops effectue les déploiements de codes**, car la séparation des tâches est une pratique largement acceptée pour réduire le risque de pannes de production et de fraude.

  - Cependant, **pour atteindre les résultats de DevOps**, notre **objectif est de nous fier davantage à d'autres mécanismes de contrôle permettant d'atténuer ces risques** de manière égale, voire plus efficace, tels que les **tests automatisés**, le **déploiement automatisé** et **l'examen par les pairs** des modifications.

Déploiements automatisés en libre-service

- *The Puppet Labs' 2013 State of devOps Report*, qui interrogeait plus de quatre mille professionnels de la technologie, a révélé que
  - **Il n'existait pas de différence statistiquement significative entre les taux de réussite des modifications entre les organisations dans lesquelles les équipes de Dev déployaient du code et celles dans lesquelles Ops le faisait.**

- En d'autres termes, quand il y a des objectifs communs qui couvrent Dev et Ops, et qu'il y a de la transparence, des responsabilités et une imputabilité pour les résultats du déploiement, le choix de l'équipe qui effectue le déploiement n'est pas important.

Déploiements automatisés en libre-service

- **Pour mieux permettre un flux rapide**, nous **souhaitons un processus de promotion du code pouvant être exécuté par les équipes de Dev ou d'Ops**, idéalement **sans étapes manuelles ni transferts**. Cela affecte les étapes suivantes:
  - **Construire (build)**: notre pipeline de déploiement doit créer des packages à partir du contrôle de versions pouvant être déployés dans n'importe quel environnement, y compris la production.
  - **Tester**: n'importe qui devrait pouvoir exécuter tout ou partie de notre suite de tests automatisés sur son poste de travail ou sur nos systèmes de test.
  - **Déployer**: tout le monde devrait pouvoir déployer ces packages dans tout environnement auquel ils ont accès, exécuté en exécutant des scripts également archivés dans le contrôle de version.

- Ce sont les **pratiques qui permettent aux déploiements d'être exécutés avec succès, indépendamment de la personne effectuant le déploiement**.

Intégration dans le pipeline de déploiement

- **But**: **Intégrer le déploiement de code dans le pipeline de déploiement**
  - Une fois le processus de déploiement de code automatisé, nous pouvons l'intégrer au pipeline de déploiement.
- Notre automatisation de déploiement doit fournir les fonctionnalités suivantes:
  - S'assurer que les packages créés au cours du processus d'intégration continue conviennent au déploiement en production.
  - Montrer la disponibilité des environnements de production en un coup d'œil.
  - Fournir une méthode de libre-service en mode "push-button" pour toute version appropriée du code empaqueté à déployer en production.
  - Enregistrer automatiquement, à des fins d'audit et de conformité, quelles commandes ont été exécutées sur quelles machines, à quel moment et qui les a autorisées, et quelle a été la sortie.
  - Exécuter un test de fumée ("smoke test") pour vous assurer que le système fonctionne correctement et que les paramètres de configuration, y compris les éléments tels que les chaînes de connexion à la base de données, sont corrects.
  - Fournir des informations rapides au déployeur afin qu'il puisse déterminer rapidement si son déploiement a réussi (e.g. le déploiement a-t-il réussi, l'application a-t-elle les performances attendues en production, etc.)?

Intégration dans le pipeline de déploiement

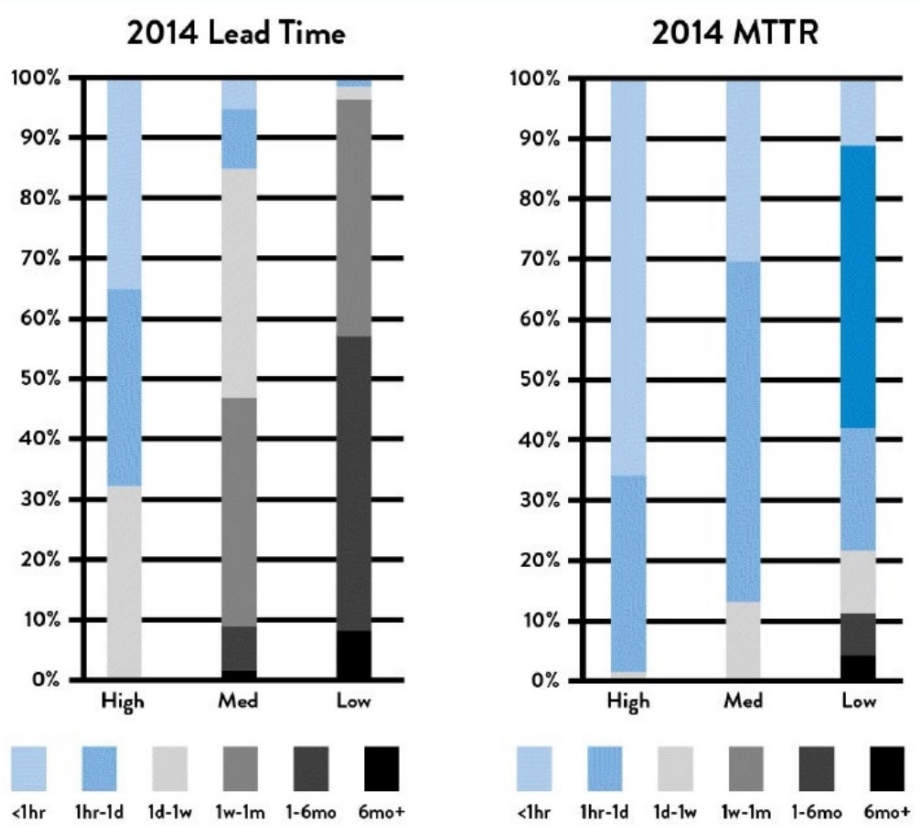

Figure 18: *High performers had much faster deployment lead times and much faster time to restore production service after incidents (Source: Puppet Labs,* 2014 State of DevOps Report.*)*

Etsy

Étude de cas : Etsy (2014)

- Etsy
  - Déploiement par les développeurs en libre-service, un exemple de déploiement continu (2014)
  - Contrairement à Facebook où les déploiements sont gérés par les ingénieurs de publication, **les déploiements chez Etsy sont effectués par toute personne souhaitant effectuer un déploiement**, telle que Développement, Opérations ou Infosec.
  - Le processus de déploiement chez Etsy est devenu tellement sûr et routinier que les nouveaux ingénieurs procéderont à un déploiement de production dès leur premier jour de travail.

- Noah Sussman (architecte de test chez Etsy), a écrit:

  «À 8 heures du matin, un jour ouvrable normal,
  environ 15 personnes commencent à faire la queue,
  tous s'attendant à déployer collectivement jusqu'à 25 changements avant la fin de journée.

Étude de cas : Etsy (2014)

- Les ingénieurs qui souhaitent déployer leur code
  - se rendent d'abord dans un "chat", où les ingénieurs s'ajoutent à la file d'attente de déploiement,
  - voient l'activité de déploiement en cours,
  - voient qui d'autre est dans la file d'attente,
  - diffusent leurs activités et
  - obtiennent l'aide d'autres ingénieurs lorsqu'il en ont besoin.
  - Lorsque le moment est arrivé pour un ingénieur d'effectuer son déploiement, il en est informé dans la salle de discussion.

- Chez Etsy, l'**objectif** était de **faciliter le déploiement en production en toute simplicité et en toute sécurité**, en utilisant le moins d'étapes possible et le moins de cérémonies possible.
  - **Avant même que le développeur soumettre son code**, le développeur **aura exécuté tous les 4,500 tests unitaires sur son poste de travail**, ce qui prend **moins d'une minute**.
  - Lors de ces tests, les appels vers des systèmes externes, tels que des bases de données, ont été supprimés ("stubbed out").

Étude de cas : Etsy (2014)

- **Après avoir soumis les modifications apportées au tronc** dans le contrôle de version, **plus de 7,000 tests de tronc automatisés sont instantanément exécutés sur leurs serveurs d'intégration continue (CI)**.
- Sussman écrit:

  «Par essais et erreurs, nous avons opté pour une durée d'environ 11 minutes, soit la plus longue durée possible de tests automatisés lors d'un push.

  Cela laisse du temps pour relancer les tests une fois au cours d'un déploiement [si quelqu'un casse quelque chose et a besoin de le réparer], sans dépasser la limite de 20 minutes ».

- Si tous les tests étaient exécutés de manière séquentielle, Sussman déclare que

  «les 7,000 tests de lignes réseau prendraient environ une demi-heure à exécuter.

  Nous avons donc divisé ces tests en sous-ensembles et les avons répartis sur les 10 machines de notre cluster Jenkins [CI] (...).

  La scission de notre suite de tests et l'exécution de nombreux tests en parallèle nous donnent la durée d'exécution souhaitée de 11 minutes. »

Étude de cas : Etsy (2014)

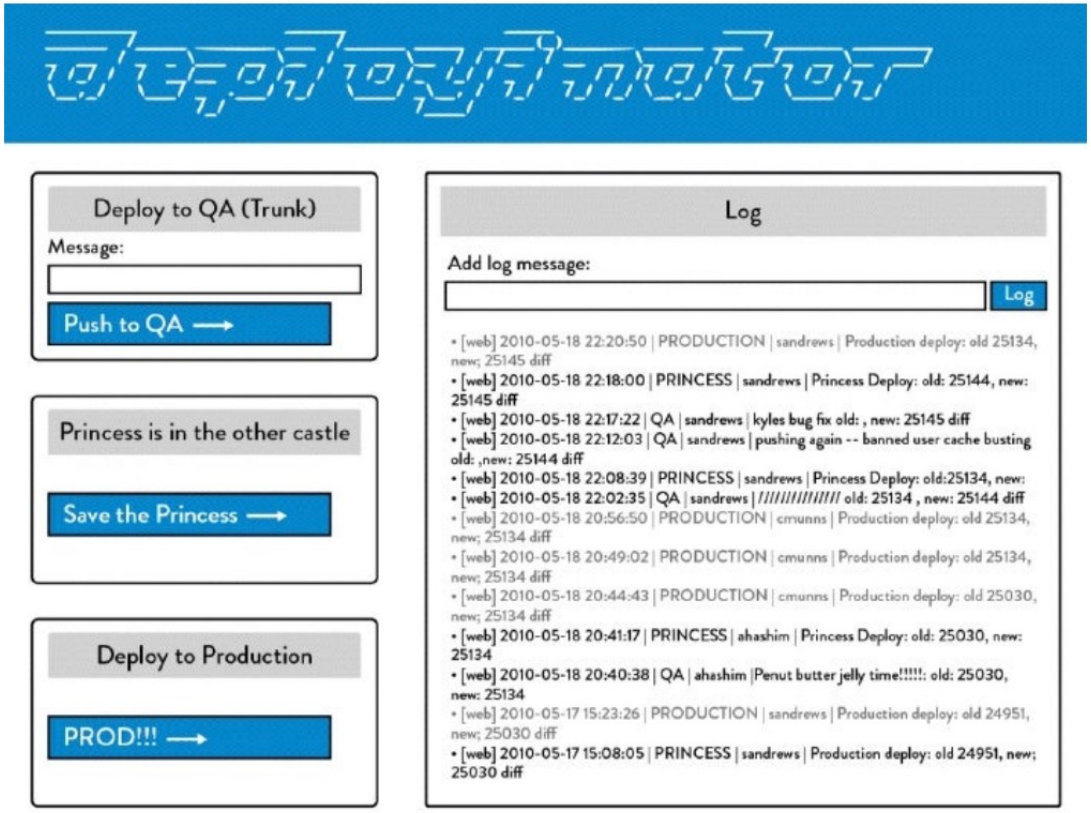

Figure 19: *The Deployinator console at Etsy (Source: Erik Kastner, "Quantum of Deployment," CodeasCraft.com, May 20, 2010, https://codeascraft.com/2010/05/20/quantum-of-deployment/.)*

Étude de cas : Etsy (2014)

- **Les prochains tests à exécuter sont les tests de fumée**, qui sont des tests au niveau du système qui exécutent cURL pour exécuter des scénarios de test unitaires PHPUnit.
- **Après ces tests, les tests fonctionnels sont exécutés**.
  - Exécutent des tests de bout en bout pilotés par une interface utilisateur graphique sur un serveur actif.
  - Ce serveur est soit leur environnement de contrôle qualité, soit leur environnement de transfert (surnommé «Princess»), qui est en fait un serveur de production qui a été retirée de la rotation, garantissant ainsi qu'elle correspond parfaitement à l'environnement de production.
- Erik Kastner écrit,
  - « Une fois que c'est au tour de l'ingénieur de déployer, il va dans Deployinator [un outil développé en interne, voir la figure 19] et appuie sur le bouton pour le faire passer à l'assurance qualité.
  - De là, il rend visite à Princess ....
  - Ensuite, quand il est prêt à fonctionner, il clique sur le bouton "Prod" et bientôt le code est en direct, et tout le monde sur IRC [canal de discussion] sait qui a envoyé quel code, avec un lien au diff.
  - Pour tous ceux qui ne sont pas sur IRC, il y a le courrier électronique que tout le monde reçoit avec les mêmes informations. "

Étude de cas : Etsy (2014)

- En 2009, le processus de déploiement chez Etsy a été une source de stress et de peur.
- En 2011, cette opération était devenue une opération courante, exécutée vingt-cinq à cinquante fois par jour, aidant les ingénieurs à mettre rapidement leur code en production, apportant ainsi de la valeur à leurs clients.

- Introduction
- Automatisation du processus de déploiement
  - Déploiements automatisés en libre-service
  - Intégration dans le pipeline de déploiement
- **Déploiements vs publications**
  - Publications basées sur l'environnement
    - Déploiement bleu-vert (blue-green)
    - Publication canarie ("Canary release")
    - Système "Cluster Immune"
  - Publications basées sur les applications
    - Bascule de fonctionnalités
    - Lancements sombres ("dark launches")
- Conclusion

Découpler les déploiements des publications

- Dans le **lancement traditionnel d'un projet logiciel**, **les publications (releases) sont déterminées par notre date de lancement marketing**.
  - La soirée précédente, nous avons déployé notre logiciel complet (ou aussi complet que possible) en production.
  - Le lendemain matin, nous annonçons nos nouvelles capacités au monde entier, commençons à prendre des commandes, fournissons la nouvelle fonctionnalité au client, etc.

- Cependant, **trop souvent, les choses ne se passent pas comme prévu**.
  - Nous pouvons rencontrer des charges de production pour lesquelles nous n'avons jamais testé ou conçu, ce qui entraîne une défaillance spectaculaire de notre service, tant pour nos clients que pour notre organisation.
  - Pire encore, la restauration du service peut nécessiter un processus de retour en arrière pénible ou une opération corrective tout aussi risquée, où nous apportons des modifications directement en production, ce peut être une expérience vraiment misérable pour les travailleurs.
  - Lorsque tout fonctionne enfin, tout le monde pousse un soupir de soulagement, reconnaissant que les déploiements et les publications (releases) ne se produisent pas plus souvent.

Découpler les déploiements des publications

- Bien sûr, nous savons que nous devons déployer plus fréquemment pour obtenir le résultat souhaité, à savoir un flux fluide et rapide, et non moins fréquemment.
  - Pour ce faire, nous devons **dissocier nos déploiements en** production **de nos nouvelles publications de fonctionnalités (feature releases)**.
  - **En pratique, les termes déploiement et publication (release) sont souvent utilisés de manière interchangeable.**
  - Cependant, ce sont deux actions distinctes qui servent deux objectifs très différents.

- **Déploiement**
  - Installation d'une version spécifiée du logiciel dans un environnement donné
    - Exemple: le déploiement de code dans un environnement de test d'intégration ou le déploiement de code en production.
  - Plus précisément, **un déploiement peut ou non être associé à la publication (release) d'une fonctionnalité (release) pour les clients**.

Découpler les déploiements des publications

- **Publication (release)**
  - La publication (release) **correspond au moment où nous mettons une fonctionnalité** (ou un ensemble de fonctionnalités) **à la disposition de tous nos clients ou d'un segment de clients**.
    - Exemple: nous permettons à la fonctionnalité d'être utilisée par 5% de notre clientèle.
  - **Notre code et nos environnements doivent être conçus de manière à ce que la publication des fonctionnalités ne nécessite pas de modification du code de notre application**.

- Le fait de confondre déploiement et publication rend difficile l'attribution de responsabilité par rapport à l'atteinte de résultats positifs.

- Le découplage de ces deux activités nous permet de
  - responsabiliser **Dev et Ops** quant au **succès des déploiements rapides et fréquents**,
  - tout en permettant aux **propriétaires de produits** d'être **responsable de la réussite commerciale de la publication (release) d'une nouvelle version**.

Découpler les déploiements des publications

- Les pratiques décrites jusqu'à présent dans ce livre garantissent que nous effectuons des déploiements de production rapides et fréquents tout au long du développement des fonctionnalités, dans le but de réduire le risque et l'impact des erreurs de déploiement.
- Le risque restant est le **risque de publication (release)**, qui consiste à **déterminer si les fonctionnalités que nous mettons en production atteignent les résultats souhaités pour le client et l'entreprise**.
- Si les délais de déploiement sont extrêmement longs, cela détermine la fréquence à laquelle nous pouvons commercialiser de nouvelles fonctionnalités.
- Cependant, **à mesure que nous parvenons à déployer à la demande, la rapidité avec laquelle nous exposons de nouvelles fonctionnalités aux clients devient une décision commerciale et marketing,** et non une décision technique**.**

Catégories de modèles de publication

- **Découpler** les **déploiements** de nos **publications ("releases")** change radicalement notre façon de travailler.
  - Nous ne sommes plus obligés d'effectuer des déploiements au milieu de la nuit ou le week-end pour réduire le risque d'impact négatif sur les clients.
  - Au lieu de cela, nous pouvons effectuer des déploiements pendant les heures de bureau habituelles, permettant ainsi à Ops de disposer enfin d'heures de travail normales, comme tout le monde.

- Nous pouvons utiliser **deux grandes catégories de modèles de publication**:
  - **Publications basées sur l'environnement**
  - **Publications basées sur les applications**

Publications basées sur l'environnement

- **Modèles de publications basées sur l'environnement**:
  - Nous déployons dans plusieurs environnements, mais un seul environnement reçoit un trafic client actif (par exemple, en configurant nos équilibreurs de charge).
  - Le nouveau code est déployé dans un environnement non actif et la publication effectue le transfert du trafic vers cet environnement.
  - Ce sont des modèles extrêmement puissants, car ils nécessitent généralement peu ou pas de modification de nos applications.
  - Ces schémas incluent des déploiements bleu-vert, des publications "canaris", et des systèmes immunitaires en grappe ("cluster immune systems").

Publications basées sur les applications

- **Modèles de publications basées sur les applications**:
  - C'est ici que nous modifions notre application afin de pouvoir sélectionner et exposer de manière sélective des fonctionnalités spécifiques de l'application en modifiant légèrement la configuration.
    - Par exemple, nous pouvons implémenter des indicateurs qui exposent progressivement de nouvelles fonctionnalités en production à l'équipe de développement, à tous les employés internes, à 1% de nos clients ou, lorsque nous sommes certains que la version fonctionnera comme prévu, à l'ensemble de notre clientèle.
  - Comme indiqué précédemment, cela permet d'utiliser une technique appelée «lancement sombre», dans laquelle toutes les fonctionnalités doivent être lancées en production et testées avec le trafic de production avant notre publication.
    - Par exemple, nous pouvons tester de manière invisible notre nouvelle fonctionnalité avec le trafic de production pendant des semaines avant notre lancement afin d'exposer les problèmes afin qu'ils puissent être résolus avant notre lancement réel.

- Introduction
- Automatisation du processus de déploiement
  - Déploiements automatisés en libre-service
  - Intégration dans le pipeline de déploiement
- Déploiements vs publications
  - **Publications basées sur l'environnement**
    - Déploiement bleu-vert (blue-green)
    - Publication canarie ("Canary release")
    - Système "Cluster Immune"
  - Publications basées sur les applications
    - Bascule de fonctionnalités
    - Lancements sombres ("dark launches")
- Conclusion

Publications basées sur l'environnement

- Cette section porte sur les modèles de publications basées sur l'environnement, qui ne nécessitent aucune modification du code de l'application.
  - Consiste à déployer dans plusieurs environnements, mais un seul d'entre eux reçoit le trafic client en direct.
  - Permet de réduire considérablement le risque associé aux versions de production et réduire le délai de déploiement.
- Trois patrons ("patterns") de publications basées sur l'environnement
  - **Déploiement bleu-vert ("blue-green deployment")**
  - **Publication canarie ("Canary release")**
  - **Système "Cluster Immune"**

Déploiement bleu-vert (blue-green)

- Le plus simple des trois patrons ("patterns") est appelé déploiement bleu-vert.
  - Dans ce modèle, nous avons deux environnements de production: bleu et vert.
  - À tout moment, un seul d'entre eux dessert le trafic client: dans la figure 20, l'environnement vert est actif.

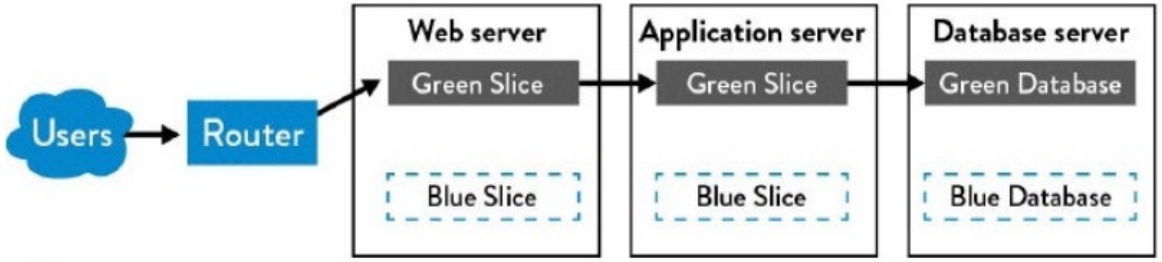

Figure 20: *Blue-green deployment patterns (Source: Humble and North,* Continuous Delivery, *261.)*

- Pour publier une nouvelle version de notre service, nous déployons initialement dans un environnement inactif où nous pouvons effectuer nos tests sans interrompre l'expérience utilisateur.
- Lorsque nous sommes convaincus que tout fonctionne comme prévu, nous exécutons notre publication en dirigeant le trafic vers l'environnement bleu.
  - Ainsi, le bleu devient vivant et le vert devient une mise en scène ("staging").
  - La restauration est effectuée en renvoyant le trafic client vers l'environnement vert.

Déploiement bleu-vert (blue-green)

- Avantages:
  - Modèle simple
  - Extrêmement facile de l'utiliser ("retrofit") sur des systèmes existants.
  - Présente également des avantages importants, tels que permettre à l'équipe d'effectuer des déploiements pendant les heures normales de travail et d'effectuer de simples changements (par exemple, changer un paramètre de routeur, changer un lien symbolique) pendant les heures creuses.
  - Permet de considérablement améliorer les conditions de travail de l'équipe chargée du déploiement.
- Désavantage:
  - Le fait d'avoir deux versions de notre application en production crée des problèmes lorsqu'elles dépendent d'une base de données commune.


DIXONS RETAIL
BRINGING LIFE TO TECHNOLOGY

Étude de cas : Dixons Retail (2008)

- Dixons Retail — Déploiement bleu-vert du système de point de vente (2008)
  - Dixons Retail: grand détaillant britannique utilisant des milliers de systèmes de point de vente résidant dans des centaines de magasins de détail et opérant sous plusieurs de différentes marques de clients.
  - Systèmes de point de vente: point-of-sale (POS) systems
- Dan North et Dave Farley, co-auteurs de Continuous Delivery, travaillaient sur un projet pour Dixons Retail.
  - Bien que les déploiements bleu-vert soient principalement associés à des services Web en ligne, North et Farley ont utilisé ce modèle pour réduire de manière significative les risques et les temps de basculement des mises à niveau de point de vente.
- La mise à niveau des systèmes POS est traditionnellement fait en utilisant une approche big-bang en cascade (waterfall)
  - Le logiciel POS-client et le serveur centralisé sont mis à niveau en même temps.
  - Nécessite un temps d'immobilisation important (souvent un week-end entier), ainsi qu'une bande passante réseau importante pour déployer le nouveau logiciel client à tous les magasins de détail.
  - Lorsque les choses ne se déroulent pas comme prévu, les opérations des magasins peuvent être fortement perturbées.

Étude de cas : Dixons Retail (2008)

- Pour cette mise à niveau, la bande passante réseau était insuffisante pour mettre à niveau simultanément tous les systèmes POS, ce qui rendait la stratégie traditionnelle impossible.
- Pour résoudre ce problème, ils ont utilisé la stratégie bleu-vert et créé deux versions de production du logiciel serveur centralisé.
  - Permet de prendre en charge simultanément les anciennes et les nouvelles versions du logiciel POS-client.
- Quelques semaines avant la mise à niveau prévue du système POS, ils ont commencé à envoyer de nouvelles versions des programmes d'installation du logiciel POS-client aux magasins via des liaisons réseau lentes
  - A permis de déployer le nouveau logiciel sur les systèmes POS dans un état inactif.
  - Pendant ce temps, l'ancienne version continuait à fonctionner normalement.

Étude de cas : Dixons Retail (2008)

- Lorsque tous les POS-clients ont été prêts pour la mise à niveau (la mise-à-jour des clients et des serveurs avaient été testés ensemble avec succès, et qu'un nouveau logiciel client avait été déployé sur tous les clients), les responsables de magasin avaient le pouvoir de décider quand publier la nouvelle version.
  - En fonction de leurs besoins, certains responsables souhaitaient procéder à la publication et utiliser les nouvelles fonctionnalités immédiatement, tandis que d'autres souhaitaient attendre.
  - Dans les deux cas, qu'il s'agisse de publier des fonctionnalités immédiatement ou d'attendre, les gestionnaires étaient nettement mieux traités que de laisser le service informatique centralisé les choisir au moment de la publication.

- **Résultat**:
  - **Publication (release) nettement plus fluide et plus rapide**
  - **Plus grande satisfaction de la part des directeurs de magasin**
  - **Beaucoup moins de perturbations dans les opérations du magasin**

Déploiement bleu-vert (blue-green)

- **Certaines variantes de ce modèle peuvent améliorer d'avantage la sécurité et les délais de déploiement grâce à l'automatisation**, mais en général implique un niveau de complexité supplémentaire, par exemple
  - **Publication canarie ("Canary release")**
  - **Système "Cluster Immune"**

Publication canarie ("Canary release")

- Le modèle de **publication Canarie automatise le processus de publication en promouvant des environnements de plus en plus importants et critiques**, au fur et à mesure que nous confirmons que le code fonctionne comme prévu.
- Le terme "Publication canarie" vient de la tradition des mineurs de charbon qui introduisaient des canaris en cage dans les mines pour permettre une détection précoce des niveaux toxiques de monoxyde de carbone.
  - S'il y avait trop de gaz dans la mine, les canaris mourraient, ce qui incitait les mineurs à évacuer.

- Dans ce modèle, lorsque nous effectuons une publication, nous surveillons les performances du logiciel dans chaque environnement.
- Lorsque quelque chose semble aller mal, nous reculons; sinon, nous déployons dans l'environnement suivant.

Publication canarie ("Canary release")

La figure 21 illustre les groupes d'environnements créés par Facebook pour prendre en charge ce modèle de version.

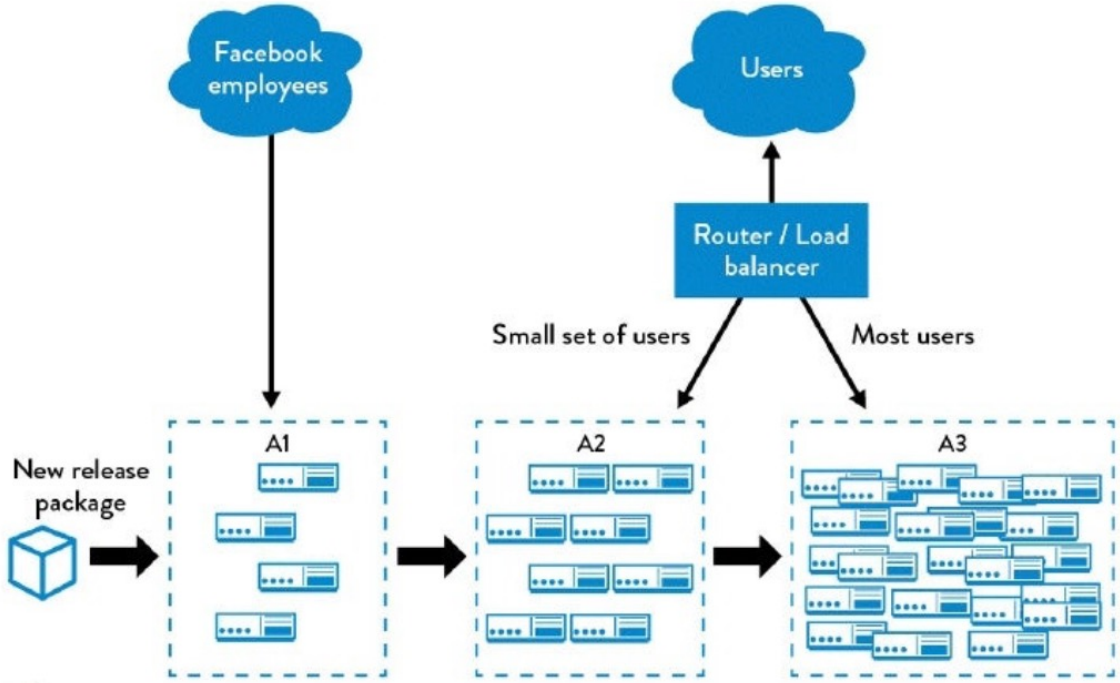

Figure 21: *The canary release pattern (Source: Humble and Farley,* Continuous Delivery, *263.)*

- Groupe A1: serveurs de production qui ne servent que des employés internes
- Groupe A2: serveurs de production qui ne desservent qu'un faible pourcentage de clients et sont déployés lorsque certains critères d'acceptation ont été satisfaits (qu'ils soient automatisés ou manuels)
- Groupe A3: le reste des serveurs de production, qui sont déployés après que le logiciel exécuté dans le cluster A2 a satisfait à certains critères d'acceptation.

Système "Cluster Immune"

- Le **système Cluster Immune étend le modèle de publications canaris** en
  - **associant notre système de surveillance de la production à notre processus de publication** et
  - **automatisant la restauration du code lorsque les performances du système de production qui s'affichent en regard de l'utilisateur s'écartent des limites d'une plage prédéfinie attendue**
    - Par exemple, lorsque les taux de conversion pour les nouveaux utilisateurs tombent en dessous de nos normes historiques de 15% à 20%.

- Ce type de protection présente **deux avantages importants**
  - Premièrement, nous **protégeons contre les défauts difficiles à trouver grâce aux tests automatisés**, tels que le changement de page Web qui rend invisible un élément de page essentiel (par exemple, le changement de CSS).
  - Deuxièmement, nous **réduisons le temps nécessaire pour détecter et réagir à la dégradation des performances créée par notre changement**.

- Introduction
- Automatisation du processus de déploiement
  - Déploiements automatisés en libre-service
  - Intégration dans le pipeline de déploiement
- Déploiements vs publications
  - Publications basées sur l'environnement
    - Déploiement bleu-vert (blue-green)
    - Publication canarie ("Canary release")
    - Système "Cluster Immune"
  - **Publications basées sur les applications**
    - Bascule de fonctionnalités
    - Lancements sombres ("dark launches")
- Conclusion

Publications basées sur les applications

- Les modèles basés sur l'environnement
  - Permettent de dissocier les déploiements des versions en utilisant plusieurs environnements et en basculant entre les environnements actifs
  - Peuvent être entièrement implémenté au niveau de l'infrastructure.

- Les **modèles de publications basées sur les applications**
  - **Peuvent être implémenter dans notre code**
  - **Permet une flexibilité encore plus grande** pour la diffusion en toute sécurité de nouvelles fonctionnalités à notre client, souvent par fonctionnalité.
  - Les modèles de publications basées sur l'application étant implémentés dans l'application, ils **nécessitent l'intervention de Dev**.

Bascule de fonctionnalités

- La principale manière d'activer les modèles de publications basées sur l'application consiste à implémenter des **bascules de fonctionnalités ("Feature Toggle")**
  - **Mécanisme permettant d'activer et de désactiver de manière sélective des fonctionnalités sans nécessiter de déploiement de code de production**.
  - **Permet également de contrôler les fonctionnalités visibles et disponibles pour des segments d'utilisateurs spécifiques** (e.g., les employés internes, les segments de clients).

- Les bascules de fonctionnalités sont **généralement implémentés en encapsulant des éléments de logique d'application ou d'interface utilisateur avec une instruction conditionnelle**.
  - La fonctionnalité étant **activée ou désactivée en fonction d'un paramètre de configuration**.
  - Cela peut être aussi simple qu'un fichier de configuration d'application (fichiers de configuration en JSON, XML, par exemple), ou via un service de répertoire ou même un service Web spécialement conçu pour gérer le basculement de fonctionnalité.

Bascule de fonctionnalités

- Les bascules de fonctionnalités permettent également d'effectuer les opérations suivantes:
  - **Rétrogradez facilement**:
    - Les fonctions qui créent des problèmes ou des interruptions de production peuvent être désactivées rapidement et en toute sécurité en modifiant simplement le paramètre de bascule des fonctions.
    - Particulièrement utile lorsque les déploiements sont peu fréquents: il est généralement beaucoup plus facile de désactiver les fonctionnalités d'un intervenant que de restaurer une version complète.
  - **Diminuez gracieusement les performances**:
    - Lorsque notre service supporte des charges extrêmement élevées qui nous obligeraient normalement à augmenter notre capacité ou, pire, risqueriez que notre service échoue en production, nous pouvons utiliser des options pour réduire la qualité du service.
    - En d'autres termes, nous pouvons augmenter le nombre d'utilisateurs que nous servons en réduisant le niveau de fonctionnalités fournies (par exemple, réduire le nombre de clients pouvant accéder à une fonction donnée, désactiver les fonctions gourmandes en ressources processeur, telles que les recommandations, etc.).
  - **Augmenter notre résilience grâce à une architecture orientée services**:
    - si nous avons une fonctionnalité qui repose sur un autre service qui n'est pas encore complet, nous pouvons toujours déployer notre fonctionnalité en production mais la cacher derrière une bascule de fonctionnalités.
    - Lorsque ce service devient enfin disponible, nous pouvons activer la fonctionnalité.
    - De même, lorsqu'un service sur lequel nous comptons échoue, nous pouvons désactiver la fonctionnalité pour empêcher les appels vers le service en aval tout en maintenant le reste de l'application en cours d'exécution.

Bascule de fonctionnalités

- Pour s'assurer que les erreurs contenues dans des fonctions associées à des bascules de fonctionnalités soient soient détectées, nos tests d'acceptation automatisés doivent s'exécuter avec toutes les fonctions activées.
  - Nous devrions également vérifier que notre fonctionnalité de bascule fonctionne correctement aussi!

- Les bascules de fonctionnalités **permettent de découpler les déploiements de code et les publications de fonctionnalités**.
- Dans les chapitres suivants, nous **utiliserons les bascules de fonctionnalités pour permettre le développement basé sur des hypothèses et les tests A/B**, renforçant ainsi notre capacité à atteindre les résultats commerciaux souhaités.

Lancements sombres ("dark launches")

- Les bascules de fonctionnalités nous **permettent de déployer des fonctionnalités en production sans les rendre accessibles aux utilisateurs** => permet d'utiliser une technique appelée **lancement sombre**.
  - Consiste à déployer toutes les fonctionnalités en production, et à tester ces fonctionnalités alors qu'elles sont toujours invisibles pour les clients.
  - Pour les modifications importantes ou risquées, nous le faisons souvent des semaines avant le lancement de la production, ce qui nous permet de tester en toute sécurité les charges prévues similaires à la production.

- Par exemple, supposons que nous ne lancions pas une nouvelle fonctionnalité qui présente un risque significatif pour la publication, telle que de nouvelles fonctionnalités de recherche, des processus de création de compte ou de nouvelles requêtes de base de données.
  - Lorsque tout le code est en production et que la nouvelle fonctionnalité est désactivée, nous pouvons modifier le code de session de l'utilisateur pour appeler des nouvelles fonctions.
  - Au lieu d'afficher les résultats à l'utilisateur, nous enregistrons ou ignorons simplement les résultats.

Lancements sombres ("dark launches")

- Exemple
  - il est possible que 1% de nos utilisateurs en ligne passent des appels invisibles vers une nouvelle fonctionnalité dont le lancement est prévu pour voir comment notre nouvelle fonctionnalité se comporte sous charge.
  - Une fois tous les problèmes identifiés et résolus, nous augmentons progressivement la charge simulée en augmentant le nombre et la fréquence d'utilisation du nouvel utilisateur.
  - Ce faisant, nous sommes en mesure de simuler en toute sécurité des charges similaires à celles de la production, ce qui nous donne l'assurance que notre service fonctionnera comme il se doit.

- De plus, lorsque nous lançons une fonctionnalité, nous pouvons la déployer progressivement vers de petits segments de clients, en arrêtant la publication si un problème est détecté.
  - De cette manière, nous minimisons le nombre de clients auxquels une fonction est attribuée
  - Nous la supprimons uniquement si nous trouvons un défaut ou que nous ne sommes pas en mesure de maintenir les performances requises.

Lancements sombres ("dark launches")

- Plus tard, lorsque mettons en place une télémétrie de production adéquate dans nos applications et nos environnements, nous pouvons également permettre des cycles de retour plus rapides pour valider nos hypothèses et résultats commerciaux immédiatement après le déploiement de la fonctionnalité en production.

- En faisant cela, nous n'attendons plus la publication d'une version Big-bang pour déterminer si les clients souhaitent utiliser les fonctionnalités que nous développons.
- Au lieu de cela, **lorsque nous annonçons et publions une nouvelle fonctionnalité majeure, nous avons déjà testé nos hypothèses et mené d'innombrables expériences pour améliorer notre produit de façon itérative avec de vrais clients**
  - Aide à valider le fait que ces fonctionnalités permettront d'obtenir les résultats souhaités.

- Introduction
- Automatisation du processus de déploiement
  - Déploiements automatisés en libre-service
  - Intégration dans le pipeline de déploiement
- Déploiements vs publications
  - Publications basées sur l'environnement
    - Déploiement bleu-vert (blue-green)
    - Publication canarie ("Canary release")
    - Système "Cluster Immune"
  - Publications basées sur les applications
    - Bascule de fonctionnalités
    - Lancements sombres ("dark launches")
- **Conclusion**

Conclusion

- Comme le montrent les exemples de Facebook, Etsy et CSG, **les publications (releases) et les déploiements ne doivent pas nécessairement être des affaires à haut risque et dramatiques** qui nécessitent des dizaines ou des centaines d'ingénieurs à travailler en permanence.
- Au lieu de cela, ils **peuvent être entièrement routiniers et faire partie du travail quotidien** de chacun.

- Ce faisant, nous **pouvons réduire nos délais de déploiement de plusieurs mois à quelques minutes**, ce qui **permet à nos organisations de fournir rapidement de la valeur à nos clients** sans causer de chaos ni de perturbation.
- De plus, en associant Dev et Ops, nous pouvons enfin faire en sorte que les opérations fonctionnent avec humanité.