LOG680
Introduction à l'approche DevOps

Créer les fondements de notre pipeline de déploiement

DevOps Handbook

Part III, Chap 9

Objectifs d'apprentissage

- Expliquer les conséquences de l'intégration tardive d'une application dans un environnement de production.
- Expliquer les avantages de permettre aux développeurs d'exécuter leurs applications sur des environnements de type production sur leurs propres postes de travail, créés à la demande et autogérés.
- Expliquer le rôle des systèmes de gestion des versions dans l'automatisation de la création "à la demande" des environnements de développement, de test et de production.
- Expliquer pourquoi il est important d'inclure les aspects autres que le développement dans le système de gestion des versions.
- Expliquer pourquoi l'utilisation du gestion de versions pour nos environnements permet de mieux prédire les performances informatiques et organisationnelles que l'utilisation du contrôle de versions pour notre code.
- Mutable vs Immutable Infrastructure. Quels sont les avantages et inconvénients de chacun.
- Expliquer comment devrait être définie la notion de "développement terminé" dans un contexte DevOps.

Sujets

- Introduction
- Environnements sur demande
- Gestion de versions
- Rendre l'infrastructure plus facile à reconstruire qu'à réparer
- Définition du développement "terminé"
- Conclusion

- **Introduction**
- Environnements sur demande
- Gestion de versions
- Rendre l'infrastructure plus facile à reconstruire qu'à réparer
- Définition du développement "terminé"
- Conclusion

Contenu du chapitre

Rappel -- chapitre 2 (LOG680-E21-06-First Way-Les principes de flux):

- Dans les transformations DevOps typiques, la contrainte suit généralement la progression suivante :
  - Création d'environnement
  - Déploiement de code
  - Configuration et exécution des tests
  - Architecture trop serrée

Dans ce chapitre, nous verrons comment créer les mécanismes qui nous permettront de:

- créer des environnements sur demande,
- étendre l'utilisation de la gestion de versions à tous les utilisateurs impliqués dans le flux de valeur,
- rendre l'infrastructure plus facile à reconstruire qu'à réparer et
- assurer que les développeurs exécutent leur code dans des environnements de type production tout au long du cycle de développement du logiciel.

Introduction

- Contexte
  - Trop souvent, nous ne découvrons le fonctionnement de nos applications dans un environnement de production que lors du déploiement en production

- Environnements de type production
  - Utilisation d'environnements de type production à toutes les étapes du flux de valeur.
  - Créés de manière automatisée
    - à la demande à partir de scripts et d'informations de configuration stockés dans le contrôle de version
    - entièrement auto-entretenus, sans aucun travail manuel requis de la part des équipes d'Ops
  - L'objectif est de pouvoir recréer l'environnement de production dans son intégralité à partir du contrôle de version.

Exemple : Enterprise Data Warehouse

- Contexte
  - Programme Enterprise Data Warehouse dirigé par Em Campbell-Pretty
  - Grande entreprise de télécommunications australienne en 2009 -- Programme de $200 millions
  - Dix flux ("streams") de travaux en cours, toutes utilisant des processus en cascade
    - Les dix flux accusaient un retard considérable
    - Un seul avait atteint avec succès l'étape de UAT conformément au calendrier imparti
    - Il a fallu six mois supplémentaires pour que ce flux complète l'UAT, avec une capacité résultante bien en deçà des attentes de l'entreprise.
- Rétrospective à l'échelle du programme :
  - Les équipes de développement **avaient besoin d'environnements provisionnés** pour pouvoir commencer à travailler et **attendaient souvent jusqu'à huit semaines**.
- Réponse:
  - Création une nouvelle équipe d'intégration et de construction chargée d'intégrer la qualité directement dans le processus, au lieu d'essayer de l'inspecter après coup.
  - Découverte surprenante: seulement **50% du code source de leurs environnements de développement et de test correspondait à ce qui était en production**.

Exemple : Enterprise Data Warehouse

- A permis de comprendre pourquoi les équipes faisaient face à autant de problèmes chaque fois qu'ils déployaient le code dans de nouveaux environnements.
- Actions prises
  - **Réingénierie** de toutes les modifications apportées aux différents environnements.
  - **Intégration** de tout le code dans le contrôle de version.
  - **Automatisation** du processus de création d'environnement.
- Résultat
  - Le temps nécessaire pour obtenir un environnement correct est passé de **huit semaines à un jour**.

- Introduction
- **Environnements sur demande**
- Gestion de versions
- Rendre l'infrastructure plus facile à reconstruire qu'à réparer
- Définition du développement "terminé"
- Conclusion

Questions – Environnements de production

- Quelles sont les conséquences de l'intégration tardive du logiciel (application) dans un environnement de production?
  - Donner des exemples concrets.
- Pourquoi est-il essentiel d'avoir accès à des environnements de production à toutes étapes du flux de valeur?
- En quoi le fait d'avoir accès à des environnements de production permet d'améliorer le flux de valeur de livraison de logiciel?
- Est-ce que l'accès à des environnements de production à toutes étapes du flux de valeur est suffisant pour garantir une livraison (aux utilisateurs) sans risque?
  - Quels sont les limites de l'utilisation d'environnements de production?

Questions – Environnements de test

- Quels sont les principaux défis associés à la production d'environnements de test?
  - Donner des exemples concrets.
- Est-il possible de développer des environnements de test pour les types de systèmes et domaines d'application?
- Est-il possible de tester/valider tous les aspects d'un logiciel avant son déploiement final?
  - Identifier des aspects du logiciel qui ne peuvent pas être testés/validés avant son déploiement final.

Environnements de dev, test, et production

- **Contexte**
  - Environnement de production – L'intégration tardive dans un environnement de production est l'une des principales causes des problèmes associés aux versions logicielles chaotiques, perturbantes et parfois catastrophiques
  - Environnements de test – Lorsque les opérations de livraison des environnements de test nécessitent de longs délais, les équipes peuvent ne pas les recevoir suffisamment tôt pour effectuer des tests adéquats.
    - Les environnements de test sont aussi souvent mal configurés ou très différents des environnements de production.
- **But:** Permettre aux développeurs d'exécuter leurs applications sur des environnements de type production sur leurs propres postes de travail, créés à la demande et autogérés.
- **Moyens:** Définir et automatiser la création de nos environnements
  - Peut servir à automatiser divers aspects, incluant: copie d'environnement virtualisé, utilisation d'outils de gestion de la configuration "infrastructure as code", et assemblage d'un environnement à partir d'un ensemble de conteneurs.

Environnements de dev, test, et production

- Au lieu que Ops génèrent et configurent manuellement l'environnement, nous pouvons utiliser l'automatisation pour les éléments suivants:
  - Copie d'un environnement virtualisé (par exemple, une image VMware, exécution d'un script Vagrant, démarrage d'un fichier Amazon Machine Image dans EC2)
  - Création d'un processus de création d'environnement automatisé qui commence à partir de «bare metal» (par exemple, installation PXE à partir d'une image de base)
  - Utilisation d'outils de gestion de la configuration «infrastructure as code» (par exemple, Puppet, Chef, Ansible, Salt, CFEngine, etc.)
  - Utilisation d'outils de configuration automatisés du système d'exploitation (e.g. Solaris Jumpstart, Red Hat Kickstart, Debian Preseed)
  - Assemblage d'un environnement à partir d'un ensemble d'images virtuelles ou de conteneurs (e.g. Vagrant, Docker)
  - Démarrage d'un nouvel environnement dans un cloud public (e.g. Amazon Web Services, Google App Engine, Microsoft Azure), un cloud privé ou une autre plate-forme PaaS (service, tel qu'OpenStack ou Cloud Foundry, etc.).

Environnements de dev, test, et production

- Ops profite de cette possibilité pour créer rapidement de nouveaux environnements
  - L'automatisation du processus de création d'environnement **renforce la cohérence** et **réduit le travail manuel fastidieux et sujet aux erreurs**.
  - Dev bénéficie de la possibilité de reproduire toutes les parties nécessaires de l'environnement de production pour pouvoir créer, exécuter et tester leur code sur leurs postes de travail.
  - Permet aux développeurs de trouver et de résoudre de nombreux problèmes, au tout début du projet, plutôt de le faire à la phase de tests d'intégration ou, pire, à la production.

- En fournissant aux développeurs un environnement qu'ils maîtrisent pleinement, nous leur permettons de reproduire, diagnostiquer et réparer rapidement les défauts, en toute sécurité, isolés des services de production et des autres ressources partagées.
  - Ils peuvent également expérimenter avec des modifications des environnements, ainsi qu'avec le code d'infrastructure avec lequel elles sont crées (e.g. scripts de gestion de la configuration), créant ainsi des connaissances partagées entre Dev et Ops.

- Introduction
- Environnements sur demande
- **Gestion de versions**
- Rendre l'infrastructure plus facile à reconstruire qu'à réparer
- Définition du développement "terminé"
- Conclusion

Questions – Gestion de versions

- Qu'est-ce qu'un système de gestion de versions?
- À quoi servent les systèmes de gestion de versions?
  - Donner des exemples concrets.
- Quelles sont les fonctionnalités principales des systèmes de gestion de versions?

Système de gestion de versions

- **But**: **Créer une source unique de vérité (single source of thruth) pour l'ensemble du système.**

- Lors de l'étape précédente, nous avons discuté de la création "à la demande" des environnements de développement, de test et de production.
- Maintenant, nous devons nous assurer que toutes les parties de notre système logiciel sont inclues.

Système de gestion de versions

- Depuis des décennies, les systèmes de contrôle de versions sont devenus la norme pour les développeurs et les équipes de développement.
- Fonctionnalités
  - **Enregistre les modifications apportées au code** (fichiers ou ensembles de fichiers)
    - Incluant code source ou d'autres documents faisant partie d'un projet de développement logiciel
  - **Enregistre les métadonnées** (nom de l'auteur, moment de la modification)
  - **Permet de valider, de comparer, et de fusionner le code**
  - **Permet de restaurer des versions antérieures**

- Le système de gestion de versions doit être destiné à tous les membres de notre flux de valeur, y compris le contrôle qualité, les opérations, Infosec, ainsi que les développeurs.
- De plus, on doit également être en mesure de recréer l'ensemble des processus de pré-production et de construction.
  - Il faut aussi inclure les divers outils (e.g. compilateurs, outils de test) et les environnements dont ils dépendent

Système de gestion de versions

- Nous devons archiver les actifs suivants dans notre système de gestion de versions partagé:
  - tout les fichiers de code d'application et dépendances (e.g. bibliothèques, contenu statique, etc.)
  - script utilisé pour créer des schémas de base de données, données de référence d'application, etc.
  - outils et artefacts de création d'environnement décrits à l'étape précédente (e.g. images VMware ou AMI, recettes Puppet ou Chef, etc.)
  - fichier utilisé pour créer des conteneurs (e.g. fichiers de définition ou composition Docker ou Rocket)
  - tests automatisés associés et tous les scripts de test manuels
  - script prenant en charge le regroupement du code, le déploiement, la migration de la base de données et le provisionnement de l'environnement
  - artefacts du projet (e.g. documentation d'exigences, procédures de déploiement, notes de m-à-j, etc.)
  - fichiers de configuration du cloud (e.g. modèles AWS Cloudformation, fichiers DSC Microsoft Azure Stack, OpenStack HEAT)
  - tout autre script ou information de configuration requis pour créer une infrastructure prenant en charge plusieurs services (e.g. bus de service d'entreprise, systèmes de gestion de base de données, fichiers de zone DNS, règles de configuration pour les pare-feu et autres périphériques réseau)

Système de gestion de versions

- Dans le rapport 2014 sur l'état de DevOps de Puppet Labs, l'utilisation du contrôle de versions par Ops était le meilleur prédicteur des performances informatiques et organisationnelles.
  - En fait, l'utilisation de système de gestion de versions par Ops était un prédicteur plus important pour les performances organisationnelles et informatiques que si Dev utilisait la gestion de version.

- Les conclusions du rapport 2014 soulignent le rôle essentiel que joue la gestion de versions dans le processus de développement logiciel.
  - Le fait que toutes les modifications d'application et d'environnement sont enregistrées dans le contrôle de versions nous permet non seulement de voir rapidement toutes les modifications qui pourraient avoir contribué à un problème, mais également de revenir à un état antérieur connu en cours d'exécution, nous permettant de récupérer plus rapidement des échecs.

Système de gestion de version

- Pourquoi **l'utilisation du contrôle de versions pour nos environnements** permet-elle de mieux prédire les performances informatiques et organisationnelles que l'utilisation du contrôle de versions pour notre code?
  - Parce que dans presque tous les cas, il existe beaucoup (ordres de grandeur) plus de paramètres configurables dans notre environnement que dans notre code.
  - Par conséquent, c'est l'environnement qui doit le plus être contrôlé par le système de gestion de versions.
  - Le contrôle de versions fournit également un moyen de communication à toutes les personnes travaillant dans la chaîne de valeur
    - Disposer des services de développement, d'assurance qualité, d'infosec et d'opérations permet de réduire les surprises, de créer une visibilité sur le travail de chacun et renforcer la confiance.

- Introduction
- Environnements sur demande
- Gestion de versions
- **Rendre l'infrastructure plus facile à reconstruire qu'à réparer**
- Définition du développement "terminé"
- Conclusion

Reconstruire l'infrastructure

- Bill Baker ("Distinguished Engineer" chez Microsoft) a expliqué que nous traitions les serveurs comme des animaux domestiques
  - «Vous les nommez et quand ils tombent malades, vous les soignez à nouveau.
  - [Maintenant] les serveurs sont [traités] comme du bétail. Vous les numérotez et quand ils tombent malades, vous les éliminez. »
- But: Reconstruire et recréer rapidement les infrastructures rapidement au lieu de les réparer en cas de problème.
- Infrastructure immuable *("Immutable Infrastructure")* –

  Références: The Benefits of Immutable Infrastructure, A Side-by-side Comparison of Immutable vs mutable Infrastructure
  - Cohérence des environnements
  - Systèmes de gestion de versions
  - Automatisation de la configuration
- La création d'environnement reproductible nous permet d'augmenter facilement la capacité en ajoutant davantage de serveurs (mise à l'échelle horizontale).
- On doit s'assurer de maintenir nos environnements de pré-production à jour.

- Introduction
- Environnements sur demande
- Gestion de versions
- Rendre l'infrastructure plus facile à reconstruire qu'à réparer
- **Définition du développement "terminé"**
- Conclusion

Définition du développement "terminé"

- Définition de "terminé" ("Done")
  - Scrum: « lorsque nous avons un code fonctionnel et potentiellement livrable ».
- Il faut modifier notre définition du développement "terminé"
  - But: Assurer que Dev et QA intègrent régulièrement le code avec des environnements de type production à des intervalles de plus en plus fréquents tout au long du projet.
  - On doit **élargir la définition de «terminé» au-delà de la fonctionnalité de code correcte**.
  - La définition **doit aussi inclure l'exécution dans des environnements de type production**.
    - À la fin de chaque intervalle de développement, nous avons un code intégré, testé, fonctionnel et potentiellement livrable, démontré dans un environnement de type production.
  - En d'autres termes, **nous n'accepterons le travail de développement comme "terminé" que lorsque celui-ci pourra être construit et déployé, et qu'il fonctionne comme prévu dans un environnement de production, plutôt que simplement lorsqu'un développeur pense que cela est fait**.
- En permettant aux équipes de Dev et Ops de maîtriser l'interaction du code et de l'environnement et en effectuant des déploiements rapides et fréquents, nous réduisons considérablement les risques de déploiement associés aux versions de code de production.

- Introduction
- Environnements sur demande
- Gestion de versions
- Rendre l'infrastructure plus facile à reconstruire qu'à réparer
- Définition du développement "terminé"
- **Conclusion**

Conclusion

- Le flux de travail rapide du développement aux opérations nécessite que tout le monde puisse obtenir des environnements de production à la demande.
  - En permettant aux développeurs d'utiliser des environnements de production même aux premières étapes d'un projet logiciel, nous réduisons considérablement le risque de problèmes de production.
- En plaçant tous les artefacts de production dans le système de gestion de versions, nous disposons d'une "source unique de vérité" ("single source of truth")
  - Permet de recréer l'environnement de production complet de manière rapide, en utilisant les mêmes pratiques pour le travail d'Ops que pour le travail de Dev.
- En rendant l'infrastructure de production plus facile à reconstruire qu'à réparer, nous facilitons et accélérons la résolution des problèmes, tout en facilitant l'extension de la capacité.
- La mise en place de ces pratiques ouvre la voie à une automatisation complète des tests – chapitre suivant.