Introduction au DevOps

Ryan Lahfa

Déroulé du cours

Organisation du cours et son contenu

- Vous pouvez poser des questions à tout moment pendant le cours pour l'interrompre
- Les TPs se dérouleront avec un bloc de temps durant lequel, le début sera lancé par un étudiant [1] que je guiderais pendant tout le long.
- L'examination sera pratique et portera autour du déploiement essentiellement
- Tous les contenus et slides seront à disposition toujours à cette adresse <> avec des handouts pour les slides

[1] Volontaire préférablement! Ou on mettra en place un tirage au sort :)

Plan

- Quelques éléments d'histoire
- Concepts fondamentaux
- Tests en général, avec un TP de mise en place d'un CI avec test automatiques
- Déploiements ("Infrastructure as code"), avec un TP de déploiement automatique vers une machine
- Reproducibilité des environnements de production
- Architecture des services
- Haute-disponibilité et auto-scaling

Matériel et environnement requis

- Machine hôte: Linux, distros: NixOS préférée (sous VM ou non, si VM, paramétrer avec un réseau interne + un accès vers Internet)
- 2 machines cible: Linux, distros: Debian 9, NixOS avec un réseau interne avec l'hôte et/ou un accès vers Internet
- Un accès à SSH à la machine hôte ou un terminal avec un shell Bash.
- Avoir https://ngrok.com/ sous la main pour permettre un debuggage à distance pendant les TPs
- Ajouter toujours les clefs SSH publiques que vous trouverez à ce lien: <> pour permettre la connexion aux machines cibles.

Origines du DevOps

Détour par le Kaizen et le Japon d'après-guerre

Le Kaizen est un terme fusionné en japonais [2] qui se comprend comme « amélioration continue », le contexte est un Japon occupé par les États-Unis post second guerre mondiale, en pleine restructuration économique.

Le Kaizen est développé par des recherches aux États-Unis et est implémenté dans le cadre du plan Marshall afin de reconstruire le Japon.

[2] Kai et Zen, « changement » et « meilleur ».

Le cas de Toyota

Aujourd'hui, le Kaizen est énormément synonyme des techniques qui ont permis à Toyota un développement extrêmement efficace dans son début. Chaque personnel s'approprie sa tâche dans la chaîne de production et doit la réaliser à des niveaux de qualités de plus en plus haut.

- S'il y a une anomalie dans un élément de ligne, toute la chaîne est arrêtée, et il y a un « kaizen » qui commence pour déterminer une façon d'améliorer ce qui vient de se passer ;
- Chaque fin de projet entraîne un kaizen, peu importe s'il a fonctionné ou non ;
- Une culture du « Hansai » : auto-critique sincère et directe des résultats et détermination des points d'amélioration

Application par rapport à une DSI traditionnelle

Une DSI traditionnelle fonctionne comme un service à la disposition des autres pôles, elle est chargée des opérations de déploiement, de maintenance, de mise à jour de tous les services, y compris internes.

Le problème c'est qu'une entreprise a souvent plusieurs projets, de nature assez différentes, avec des technologies assez différentes, il est illusoire d'avoir une DSI capable de les gérer tous correctement, sachant que ça entraîne des frictions lors des passations et cela déresponsabilise les développeurs de la partie « déploiement » de leur code.

En effet, on voit que les DSI traditionnelles sont très lentes, ont des difficultés à répondre aux demandes des interlocuteurs, les personnes qui y travaillent souffrent de changements de contexte et d'interruptions intempestives, ce qui empêche de se concentrer et réaliser correctement une tâche.

Façon Kaizen / Kanban

On revient à une organisation en équipe plus simple, un ingénieur dit « DevOps », chargé de développer les opérations, i.e. de pérenniser le déploiement d'un projet, sa maintenance, sa surveillance sur les infrastructures de l'entreprise.

Chaque projet contenant un DevOps, la responsabilité est du bas vers le haut et rend le travail des DSI extrêmement simplifiés qui peuvent désormais se concentrer sur les déploiements de services partagés ou sur le travail d'infrastructure bas niveau.

Aussi, la version déployé sert de version de test finale pendant tout le long projet par opposition à être disponible seulement à la fin, cela permet de détecter immédiatement les problèmes de performance ou déjà de commencer des tests de profondeur et d'adopter l'application tôt.

Quelques éléments de DevOps

Dettes inversées

Dans une DSI traditionnelle, c'est la DSI qui est endetté de projets.

Dans une DSI avec une approche « Lean » [3], c'est le projet qui est responsable de son déploiement.

[3] Terme assez générique, qui décrit surtout une méthodologie simplifiée des processus métiers.

Gestionnaires de versions

Un gestionnaire de version est certainement la pièce maîtresse de tout projet, indispensable.

Au delà de Git, certaines entreprises rencontrent le besoin d'aller plus loin:

- Bazaar: https://bazaar.canonical.com/en/ : workflow avancés et performance dans les monorepos
- Darcs: http://darcs.net/ : ne requiert pas de serveur central, marche offline, basé sur des théories mathématiques un peu plus avancées ;
- Pijul (expérimental): https://pijul.org/ : basé sur Darcs mais avec une théorie encore plus poussée

Serveur de jobs

Un serveur de job pour exécuter des tâches est très utile, de nos jours, on peut avoir des versions faibles, comme:

- Travis CI
- Circle CI
- GitHub Actions
- GitLab CI

Qui permettent de faire tourner des tâches liés à un référentiel avec quelques limitations.

Limites des CI/CD gratuits

- Pas de virtualisation en général, impossible de tester un déploiement dans une machine virtuelle dans un CI
- Limites de ressources, les tests systèmes sont difficiles à réaliser lorsqu'ils demandent beaucoup de services
- Un noeud, impossible de tester un système distribué ou son comportement
- Pas souvent possible de SSH dans la machine pour comprendre ce qui se passe lorsqu'un job échoue
- Sécurité discutable: les secrets finissent temporairement sur la machine d'un tiers
- Dépend de l'infrastructure des autres!
- YAML.

Construction automatique

La plupart des projets se compilent ou s'interprète, dès lors, il est intéressant de lancer la confection ou alors des linters sur le code en continue.

Chaque commit déclenche une confection, on en tire des artéfacts qu'on peut passer entre collègues afin de pouvoir récupérer un binaire ou une tarball fonctionnelle.

Intégration continue

Après la construction automatique, il est désirable de tester son code à chaque commit, de voir quel commit bloque et d'empêcher les propositions de changements d'être fusionné dans les branches stables si les tests ne passent pas avec elle.

Exemple classique: https://bors.tech

Livraison continue

Après la livraison continue, naturellement, on étend l'automatisation au déploiement du projet, chaque commit pourrait techniquement donner lieu à un artéfact (binaire), mais dans le cadre d'un projet web ou réseau, on a envie d'orchestrer un déploiement sur les infrastructures d'entreprise et de rediriger des DNS vers le système basé sur le commit.

Souvent, en réalité, la granularité est par branche,
e.g. `feature-fairececours` → `fairececours.company.example.com`

Voire, juste sur une branche d'accumulation / staging.

Déploiement continue en production

Enfin, on n'automatise pas toute de suite le déploiement en production, c'est une opération qui requiert toujours un niveau de maturité très élevé: a-t-on envie que des commits cassés soient déployés à 4 h du matin ? Non, pas vraiment.

Cependant, il y a un autre challenge: le déploiement sans coupure ou page de maintenance, « zero downtime deployments ».

Déploiement sans coupure

La technique la plus simple et classique [4] c'est de dupliquer la production et de router progressivement les anciens utilisateurs vers la nouvelle, bien sûr, il y a des subtilités au niveau de la base de données pour les synchroniser.

En procédant ainsi, cela permet notamment de faire du A/B testing au niveau des versions de production ou alors de faire du progressive rollout, en déployant des fonctionnalités progressivement à une partie des utilisateurs.

[4] Blue Green deployments.

Mise au point de l'environnement

Mise au point de l'environnement

Vous trouverez des fiches techniques à cette URL sur comment mettre en place les outils que nous utiliserons.