

Apprivoiser la complexité de l'exploitation

Jean Benoit

Direction Informatique / Université de Strasbourg
7 rue René Descartes
67000 Strasbourg

Résumé

ITIL est souvent mis en avant pour rationaliser l'exploitation. Il s'agit de montrer ici pourquoi une démarche ITIL formelle n'est pas nécessaire dans la plupart des cas, alors que la mise en œuvre combinée de trois pratiques spécifiques, la détection des changements, une approche fédérative de référentiels et de systèmes d'inventaire, et une utilisation ciblée du déploiement reproductible, apporte des améliorations cruciales à la qualité du service.

Mots-clefs

devops, déploiement, gestion des changements, ITIL, inventaire, automatisation

1 Introduction

À l'échelle d'un campus, l'exploitation des serveurs et des réseaux fait souvent face à des problèmes de stabilité et de cohérence. La dimension de l'infrastructure, les interactions complexes entre les différents composants et les actions des administrateurs système et réseau font naître des dysfonctionnements. À cela s'ajoute pour les administrateurs un manque de visibilité et des difficultés de compréhension des incidents, ce qui augmente leur fréquence et leur durée. Une démarche structurée est nécessaire pour organiser le bon fonctionnement des systèmes et des réseaux et en garantir la plus grande disponibilité possible. L'objet de cet article est de montrer comment la combinaison de trois pratiques particulières apporte des réponses à ces problèmes.

2 Pourquoi ?

"I know engineers. They love to change things." – Dr. McCoy

- Culture de la négligence : lorsqu'un incident survient, la cause n'est pas immédiatement connue. Il est dû le plus souvent à une erreur de manipulation causée par une intervention humaine (dans 60 à 80% des cas [1]).
- Absence de documentation : sur un grand nombre de serveurs, les interdépendances n'étant pas visibles, un changement à un endroit peut provoquer des dysfonctionnements en cascade, comme une chute de dominos¹.
- Manque de visibilité sur l'impact d'une modification : souvent, les personnes qui font une modification ne connaissent pas son effet en détail, du fait des dépendances et des interactions complexes entre les systèmes.
- Manque de visibilité sur l'origine du dysfonctionnement : la personne à l'origine d'un changement non-planifié n'est pas disponible au moment où le dysfonctionnement survient et elle n'a informé personne de son action. Les personnes chargées de rétablir le bon fonctionnement passent beaucoup de temps à en trouver la cause.
- Configurations non-reproductibles : certains services fonctionnent sans que l'on sache exactement comment². Suite à des modifications non-documentées, la plate-forme en fonctionnement ne peut être recrée à l'identique.

1. Des services critiques comme le DNS ou les bases de données peuvent bloquer des dizaines d'autres services s'ils ne fonctionnent plus.

2. Par exemple, il existe des applications faites d'éléments répartis sur plusieurs serveurs, et dont on ignore l'ordre correct de démarrage !

3 ITIL ?

Certaines recommandations d'ITIL, dont les trois pratiques détaillées ici font partie, étaient au départ des pratiques de terrain. Leur formalisation et leur unifications dans des processus les rend moins lisibles. L'histoire et le contexte sont absents. Et ITIL donne très peu de clés pour les mettre en œuvre car elle est non-prescriptive. Dans notre contexte, l'application conjointe de ces trois pratiques donne un résultat plus concret qu'une démarche ITIL formelle.

4 La détection des changements

"You can't defend. You can't prevent. The only thing you can do is detect and respond." –Bruce Schneier

4.1 Principes

La détection des changements vise à diffuser l'information sur ce qui a changé. Un processus tourne en permanence sur un système, et à intervalle régulier, il compare la configuration actuelle et la configuration dans sa version précédente. Si la configuration a changé, une notification est envoyée aux administrateurs concernés.

Attention : l'objectif de cet outil n'est pas de déterminer qui est coupable d'une modification, mais bien de comprendre ce qu'il s'est passé. Il est à l'administration système ce que la boîte noire est aux accidents aériens. L'observation attentive des notifications des différences donne aux administrateurs une vision précise des actions de configuration effectuées. Chaque action a du sens : la différence est interprétée et rapprochée d'une opération de maintenance planifiée. Si la différence ne correspond à rien de connu, l'administrateur qui s'en aperçoit peut aller voir la personne à l'origine du changement et lui demander des précisions, ou prendre toute mesure utile.

4.2 Effets

La détection du changement se base sur des techniques assez anciennes. Ce sont plutôt ses effets sur le fonctionnement du groupe des administrateurs qui sont intéressants. La mise en place de cette détection vise effectivement un changement culturel. Deux phénomènes nouveaux apparaissent lorsqu'elle est pratiquée :

- l'administrateur est très enthousiaste de voir quasiment en temps réel ce qui est modifié,
- il prend conscience que tous ses collègues voient ce qu'il modifie lui-même.

Dans le monde du développement open-source, la loi de Linus dit : *«avec suffisamment d'yeux, les bugs sont superficiels»*. Même si dans le contexte de l'administration système et réseau, le nombre de paire d'yeux est moins massif, la détection et la notification des changements rendent visible les causes potentielles d'un dysfonctionnement.

De plus, les administrateurs qui seraient tentés de faire un changement en production ne peuvent plus le faire en toute discrétion. Il leur faut, dans certains cas, justifier leur action. Ainsi, pour ne pas entamer leur réputation de professionnel de l'administration système et réseau, leurs interventions deviennent peu à peu des actions plus testées et donc mieux préparées. Il existe même des coutumes d'auto-dénigrement dans les groupes d'administrateurs conscients de ces problèmes. La personne responsable d'un changement impactant sans préparer et sans prévenir va elle-même afficher sa photo sur la page de l'intranet intitulée « dernière mise en production non-planifiée avec impact », ou bien payer une tournée de croissant, etc. Peu à peu, de nouvelles habitudes se développent, les changements non-planifiés ayant un impact sur le bon fonctionnement se réduisent, et le temps consacré à la préparation augmente.

4.3 Exemples

4.3.1 Détection des changements réseau avec Rancid

Rancid³ est un outil permettant de maintenir l'historique des changements sur un parc d'équipements réseaux. Il supporte de nombreux modèles d'équipements et il est facilement adaptable. En utilisant un système de gestion de version (CVS, SVN, Git), il conserve tous les changements de configuration des équipements et en notifie les administrateurs réseaux :

3. <http://www.shrubbery.net/rancid/>

```

Index: configs/ny-cc1.example.com
=====
retrieving revision 1.77
diff -u -4 -r1.77 ny-cc1.example.com
# ny-cc1> show configuration
- ## Last commit: 2012-01-01 14:40:21 CEST by joe
+ ## Last commit: 2012-01-02 11:25:36 CEST by fred
[...]
interfaces {
    ge-1/1/0 {
        vlan {
-             members [ 2 791 ];
+             members [ 2 791 425 ];
        }
    }
}

```

Pour déclencher la récupération de la configuration, Rancid peut être lancé à intervalles réguliers ou sur réception d'un événement via Syslog (cf. le module serveur de NETMAGIS⁴). De plus, disposer des archives des versions permettent de revenir en arrière à n'importe quel point de la configuration.

4.3.2 Git sur des serveurs UNIX

L'installation de git sur un serveur⁵ pour surveiller un répertoire en particulier rend les modifications traçables. Ce répertoire peut être un répertoire système ou le répertoire de configuration d'une application. Par exemple, on transforme le répertoire /etc en dépôt git local avec les commandes suivantes :

```
cd /etc ; git init ; git add . ; git commit -m « premier commit »
```

À partir de ce moment, chaque fois qu'un fichier est modifié, il est possible de le voir. Pour automatiser la détection, il suffit de programmer dans cron la commande suivante pour notifier les administrateurs des changements détectés :

```
( cd /etc ; git diff ) | mail sysadmins@example.com
```

Comme git est extrêmement rapide, la commande de détection et de notification se lance toute les minutes. La notification ne doit être faite que la première fois. Cela se règle facilement en emballant la commande dans un script, keepstate⁶, qui mémorise le dernier état et notifie uniquement lorsque l'état diffère.

Chaque modification doit être enregistrée par la commande "git commit". Une nouvelle habitude doit être prise par les administrateurs. L'avantage du commit est que l'on garde une trace précise de toutes les modifications. Chaque action est associée à une description, le "commit log". Par exemple, sur un firewall sous OpenBSD, le log des modifications portant sur le fichier /etc/pf.conf (règles de filtrage) ressemble à ceci :

```

commit 8821256cef04e848440c6dcedc5cb6fff2cf8b2d
Author: Charlie <root@lb.example.com>
Date: Wed Aug 14 16:20:53 2013 +0200

acces snmp pour sirius.example.com

```

Comme pour Rancid, l'intérêt de maintenir l'historique des modifications dans git avec une granularité fine, est aussi de pouvoir revenir en arrière facilement. Il est pertinent de centraliser ces éléments sur un dépôt git distant avec la commande "git push". C'est une sauvegarde centralisée mais aussi un élément de référentiel. Il ne faut pas négliger le risque de révéler des informations sensibles (mots de passe) si le dépôt git est mutualisé avec d'autres projets.

4.3.3 Génération de diff sur des images de machines virtuelles

Une autre approche, qui fonctionne uniquement sur un parc de machines virtuelles, consiste à générer des différences

4. <http://netmagis.org/>

5. <https://github.com/jeanl/changedetection>

6. <https://github.com/pdav/keepstate>

depuis la machine hôte. Elle consiste à monter en lecture seule l'image de la machine virtuelle en fonctionnement, à copier le répertoire surveillé et le comparer à la version précédente :

```
# exemple avec kvm, une image disque qcow2 et nbd
qemu-nbd --connect=/dev/nbd0 image.qcow2
mount -o ro /dev/nbd0p1 /mnt
cp /mnt/etc /versions/current
diff -r /versions/prev /versions/current | mail sysadmins@example.com
rm -fr /versions/prev
mv /versions/current /versions/prev
```

Il est recommandé de le faire uniquement sur des répertoires de configuration qui sont peu volumineux. Cette méthode est non-intrusive ; au niveau sécurité notamment elle révèle des compromissions sans modifier le système. Elle ne comporte pas de validation des modifications, contrairement à git. Et l'historique et les possibilités de retour en arrière sont moins immédiats.

4.4 Contexte d'application et pratiques associées

Il est recommandé d'appliquer la détection uniquement au sein d'un groupe d'administrateurs. Rappelons qu'il ne s'agit pas d'effectuer une surveillance visant à condamner les actions des administrateurs mais d'aider à comprendre les problèmes en augmentant la visibilité des modifications. Il est recommandé de ne pas notifier les personnes non-impliquées techniquement dans l'administration systèmes et réseaux. Évidemment, cela ne fonctionne qu'à condition de recruter des administrateurs compétents, d'entretenir leur motivation et de les responsabiliser par rapport à leurs actions.

D'autres pratiques comme l'organisation globale des changements avec un calendrier centralisé, et des circuits de communication pour informer les utilisateurs concernés sont tout à fait complémentaires. Cependant, mettre en place une gestion formelle des changements présente des risques importants, en particulier si elle est conduite dans l'objectif d'en faire un enjeu de pouvoir. La gestion des changements est un outil d'information. C'est un moyen d'éviter les collisions entre des modifications sur des systèmes interdépendants. C'est également une pratique qui par effet de bord amène une transformation culturelle dans les comportements des administrateurs en leur faisant prendre conscience de l'impact sur les utilisateurs. Mal implémentée, elle est capable de créer une bureaucratie inefficace qui ralentit les changements, et d'engendrer une lassitude de la part des administrateurs parce qu'ils ont l'impression de perdre une partie de leur autonomie. Il est possible d'alléger le fonctionnement de la gestion des changements. En réalité, une demande de changement formelle validée par un comité n'est pas nécessaire. Planifier des fenêtres de maintenance et pratiquer la détection des changements peut suffire. Si un changement non-standard et potentiellement impactant est observé en dehors des fenêtre de maintenance, les administrateurs qui le constatent réagissent rapidement.

5 Une approche fédérative de référentiels et d'inventaires

"If your policy is in a wiki or a document, it doesn't exist" –Dan Cohn.

5.1 Nécessité de construire un système d'information

La fragilité des services provient aussi de la méconnaissance des éléments qui les composent. Sans connaissance des éléments d'infrastructure, de leur configuration et de leurs interdépendance, appréhender les problèmes rationnellement est impossible. Un travail d'inventaire est indispensable. De plus, ce travail servira de base à l'étape suivante d'automatisation du déploiement. Comment procéder ? Le premier pas est d'identifier les serveurs les plus sensibles et de maintenir la liste de façon centralisée. Progressivement, ces informations sont consolidées et liées à d'autres référentiels (réseaux, applications etc.). Cette démarche va construire le système d'information de l'infrastructure. Ce dernier occupera une position centrale : de nombreux outils accéderont aux données qu'il contient.

5.2 Format du système d'information

Le système d'information de l'infrastructure est constitué de plusieurs référentiels et de nombreux outils. Cette situation

est normale. Il n'est pas possible d'avoir une base unique, ne serait-ce que parce que les éléments d'infrastructure sont de nature différentes (serveurs, équipements réseaux, configuration etc.). Avoir peu de types de données facilite grandement la maintenance des référentiels. Les éléments fondamentaux sur lesquels s'appuieront de nombreux outils sont les équipements réseaux et les serveurs. On peut y ajouter progressivement d'autres informations plus difficile à maintenir, comme les réseaux alloués (Vlans et blocs d'adresses), les entités (clients, institutions, etc.), les personnes responsables au sein des entités, les applications hébergées, etc. La maintenance des relations entre ces informations est certainement l'aspect le plus complexe à traiter.

Des outils existent : GLPI⁷ est un système d'inventaire pour gérer les serveurs. Netmagis est un système d'information réseau. Mais le DNS et des conventions de nommage sont également des outils intéressants (cf. 5.6.1). Intégrer comme référentiel les configurations collectées par Rancid et git est évidemment recommandé.

5.3 Faire une CMDB ?

Le modèle décrit dans ITIL[2] pour la gestion des données d'infrastructure, la CMDB (Configuration Management DataBase) est d'une grande complexité. Il propose de verser dans une base centrale le contenu de référentiels multiples, à l'aide de processus de synchronisation et de réconciliation des données. La CMDB stocke un maximum d'informations : les applications, les postes de travail, les releases, les serveurs etc., et toutes les relations entre elles.

Pourquoi cette centralisation exhaustive? Peut-être à cause de la notion de « release ». Traditionnellement, le déploiement d'un ensemble d'éléments était livré sous forme de « release » formelle et documentée. L'intervalle de temps entre les livraisons pouvait être de plusieurs mois, délais liés à des contraintes dépassées (la « release » correspondait à un média physique qu'il fallait diffuser puis installer). Dans ce contexte, la volonté de documenter chaque élément par rapport à une version et de pouvoir déclarer que la CMDB est à jour pour une version donnée est compréhensible. Actuellement, le rythme des changements est tel que le concept de « release » est difficilement applicable. Le fait de disposer de machines virtuelles, d'un mécanisme de distribution par le réseau et d'outils d'automatisation du déploiement a considérablement raccourci cet intervalle. Les cycles enchaînant l'intégration, les tests et le déploiement sont de plus en plus courts. Et l'exactitude des informations des référentiels est nécessairement variable dans le temps.

Cette centralisation exhaustive des données est discutable. De plus, ITIL ne précise pas comment initier la démarche de constitution d'une CMDB. Il semble plus praticable d'avoir une approche minimaliste, et pour chaque évolution du système d'information, de toujours viser un objectif précis⁸. La bonne question à se poser n'est donc pas : comment avoir une CMDB ? Ce serait plutôt : quel est le minimum de données de référence nécessaire pour fonctionner.

Sans centraliser les données, comment interconnecter les différents référentiels ? ITIL parle de fédérer des référentiels. En dehors d'ITIL, des frameworks standards de CMDB fédérés existent depuis quelques années⁹. À très grande échelle, cette approche peut se justifier. Mais dans la plupart des cas, les bénéfices sont hypothétiques et les efforts sont conséquents pour assembler et maintenir un tel système. Il est plus simple d'avoir quelques mécanismes simples pour lier des informations entre les différents référentiels (des web-services par exemple).

5.4 Alimentation des référentiels

L'alimentation des référentiels exige de trouver un compromis entre la difficulté de maintenance manuelle des informations (elles seront rarement mises à jour) et la difficulté d'automatiser la collecte (il faut maintenir le code de collecte et il fonctionnera rarement parfaitement).

Une première approche consiste à documenter tous les éléments déployés (serveurs, applications, etc.). Mais il est très difficile de garantir la pérennité de ces informations. On saisit souvent dans un wiki la documentation d'installation, la liste de dépendances entre serveurs, etc. Cette documentation doit exister, mais certaines informations sont incohérentes entre elles et vite périmées. La dérive est très rapide : plusieurs modifications de l'infrastructure sont faites dans une journée. La révision de la documentation se fait dans les semaines qui suivent, quand elle n'est pas purement et simplement oubliée. Il peut se passer de nombreux événements de faible ampleur qui sont renseignés très tard (ou jamais) dans la documentation : un changement de serveur pour les bases de données, une mise à jour de paquet, etc.

7. <http://www.glpi-project.org/>

8. <http://www.nanog.org/meetings/nanog26/presentations/stephen.pdf>

9. <http://www.dmtf.org/standards/cmdbf>

Il est recommandé d'utiliser des processus automatisés pour maintenir à jour les référentiels là où c'est possible. On peut utiliser des agents d'inventaire et certains protocoles ou API (cf. 5.6.3). Une approche possible consiste à s'appuyer sur les recettes de déploiement automatisée pour extraire des informations à jour (cf. exemple 5.6.1). En effet, ces recettes sont plus proches de la réalité des services qui tournent. Mais cette méthode fonctionne à condition d'avoir étendu le déploiement automatisé sur un nombre significatif d'éléments d'infrastructure.

5.5 Interface de programmation du référentiel

Il faut doter les référentiels d'interface de programmation, sous forme de web-service ou de librairie. En effet, l'API va permettre d'automatiser de nombreuses actions. Elle doit proposer au minimum les accès en lecture au référentiel. L'action la plus courante est l'énumération des objets tels que les serveurs et les équipements réseau. De plus, il faut pouvoir mettre des tags sur les objets pour les regrouper et les filtrer (exemples : tous les serveurs critiques, tous les serveurs qui jouent le rôle de firewall, tous les commutateurs d'extrémité, etc.). Si le référentiel ne supporte pas les tags, il est alors possible de faire figurer au moins un tag dans le nom de l'élément par l'application d'une convention de nommage (biblio-fw, pharma-sw etc.).

5.6 Exemples d'interaction avec des référentiels

5.6.1 Déploiement du mécanisme de détection des changement

La liste des serveurs (filtrée par des tags ou via le schéma de nommage) est également utilisés comme source par le système de déploiement pour y installer le mécanisme de détection des changements. Dans cet exemple, le référentiel des serveurs est inclus au système de déploiement (Opscode Chef¹⁰). La commande "knife ssh" recherche tous les serveurs dont le nom commence par "fw-" et lance la recette d'installation de l'outil de détection de changement :

```
knife ssh "name:fw-*" "chef-client -o 'recipe[changedetection::install]'"
```

5.6.2 Alimentation du référentiel par l'outil de déploiement

Le paramétrage des recettes est le seul qui soit fidèle à la réalité des serveurs déployés. Il contient toutes les dépendance à des serveurs externes : serveur de base de données, serveur d'authentification, serveur de log, etc. Le référentiel peut être complété par ces informations. Le système de déploiement doit proposer une API d'accès. Il est recommandé de structurer ces informations de manière homogène (toujours les mêmes variables quelles que soient les recettes de déploiement) et d'appliquer un schéma de nommage homogène (notamment s'il y a plusieurs bases de données). Exemple d'extraction avec Opscode Chef en ligne de commande :

```
# knife data bag show myapp
dbhost: db1.example.com
dbname: appdb
dbpass: XXXXXXXX
dbuser: mysql
ldaphost: directory.example.com
ldapuser: cn=user1,o=example
ldappass: YYYYYYYY
```

5.6.3 Référentiel minimal et mapping dynamique

Il est très difficile de maintenir tous les référentiels à jour. Cependant, il est possible de partir de très peu de données de référence et en faisant des requêtes sur les éléments d'infrastructure de récupérer dynamiquement de nombreuses informations. Par exemple, on peut énumérer les VM hébergées sur une machine physique, mais on peut aussi localiser un serveur ou un équipement sur le réseau¹¹. Cette localisation fonctionne comme la commande traceroute mais au niveau Ethernet. Elle reçoit en entrée une adresse IP et interroge dynamiquement les serveurs et les équipements de proche en proche, et notamment la table ARP, la table des voisins LLDP ou CDP, la table de bridging etc. La

10. <http://www.opscode.com/chef/>

11. <https://github.com/jeanl/where>

localisation a besoin de très peu données de référence : la passerelle associée à une adresse (qui peut simplement être une convention, par exemple la première ou la dernière adresse utilisable du réseau), et les communautés SNMP des équipements réseaux. Cet exemple montre aussi que stocker une grande quantité d'information dans une CMDB n'est pas pertinent quand quelques éléments et quelques secondes d'attente permettent de répondre à des requêtes usuelles.

6 Les déploiements reproductibles et automatisés

"You are in a maze of twisty little passages, all different"–Don Woods

L'image souvent utilisée pour décrire la variabilité des configurations des serveurs est le flocon de neige : observé de près, chaque cristal de glace possède une structure unique. Cette non-reproductibilité n'est pas un état souhaitable. En s'appuyant sur l'inventaire, il s'agit de réduire la variabilité des configurations en automatisant le déploiement.

6.1 Principes

L'idée du déploiement reproductible et automatisé est de formaliser chaque action d'administration pour reconstruire totalement un éléments d'infrastructure en partant de zéro. Ce système offre d'énormes avantages :

- le déploiement prend un temps déterminé (contrairement à l'intervention en urgence suite à un crash),
- la reproductibilité de l'opération facilite les tests en amont et leur résultat est plus conforme à la réalité,
- l'automatisation libère les administrateurs des tâches manuelles et fastidieuses en réduisant le risque d'erreur,
- un cluster devient « scalable », par le déploiement de serveurs supplémentaires tous conformes au modèle,
- la formalisation de chaque action constitue une documentation pour les autres administrateurs,
- l'action d'administration devient du code dont les fonctions sont réutilisables et les modifications sont traçables.

Concevoir un déploiement comme un programme conduit à modifier fondamentalement la façon de travailler des administrateurs, amenant un changement de mentalité du même ordre que la détection du changement. Bien sûr, formaliser et automatiser le déploiement est plus long que réaliser les mêmes actions d'administration à la main. Il ne faut pas tout automatiser mais privilégier, en s'appuyant sur l'inventaire, le déploiement de serveurs critiques ou des groupes de serveurs identiques. Néanmoins, il faut considérer le délai comme intrinsèque à cette transformation radicale : le temps n'est plus consacré, en aval, à réparer des dysfonctionnements en exploitation, mais, en amont, à la conception d'éléments d'infrastructure robustes.

6.2 Fonctionnement

Les systèmes de déploiement centralisent les configurations. On effectue une installation minimum sur un serveur nu : seule la connectivité réseau et la présence de l'agent de déploiement sont nécessaires. Puis le système de déploiement va lancer un ensemble de programmes d'installation, parfois appelés « recettes », avec les paramètres appropriés. Chaque recette abstrait les actions sous forme de ressource. Cela peut être un fichier de configuration (statique ou sous forme de template), une commande à exécuter, une entrée dans la crontab, un package à installer, un service à démarrer, etc.

La possibilité actuelle de créer des machines virtuelles à la demande facilite grandement l'ensemble des aspects du déploiement : les tests, la bascule du service sur un nouveau serveur, l'ajout de nouveaux serveurs en cas d'augmentations de la charge, etc.

6.3 Déploiement = code

Le déploiement automatisé introduit des pratiques courantes du monde du développement : le fractionnement en modules réutilisables, la séparation du fond et de la forme avec des templates, la gestion de version, les tests, etc.

Les tests deviennent élémentaires: cela revient à effectuer un déploiement sur une machine virtuelle de test.

L'administrateur observe le résultat et corrige sa recette. Il est évidemment judicieux d'automatiser la batterie de tests¹².

Il est important de standardiser au maximum les différents composants pour chaque classe de serveur. Au niveau de l'environnement, cela inclut les serveurs physiques, le système d'exploitation, l'outil de déploiement, etc. Au niveau des composants déployés, cela peut être les modules d'authentification, la configuration d'un frontal web, etc. L'objectif est de développer une librairie de fonctions afin d'en réutiliser des composants (cf. exemple 6.4.1) et de capitaliser sur ceux déjà débogués. L'utilisation de templates de configuration participe à cet objectif en garantissant ainsi une régularité des configuration sur l'ensemble des éléments déployés¹³.

La gestion de version des configurations déployées est vraiment complémentaire avec la détection des changements. Elle permet de mesurer la dérive de la configuration. En effet, pour des questions de périmètre ou de temps, tout n'est pas automatisé : il reste des interventions en urgence qui ne passent pas par le déploiement. La détection du changement rend cela plus robuste : elle permet de ne rien perdre en facilitant l'intégration dans la déploiement de ces modifications, et elle permet de réparer en redéployant si une modification manuelle vient perturber le fonctionnement.

6.4 Exemples

6.4.1 Composant réutilisable

La gestions des certificats est une recette réutilisable pour de nombreux types de déploiement (web, imap, ldap, etc.). Les certificats SSL expirent régulièrement. Une recette géant les certificats va simplifier leur mise à jour. Elle standardise la localisation des certificats dans un répertoire, installe l'autorité de certification, appliquer les droits corrects (protection en lecture de la clef privée), et déploie un certificat "wildcard" convenant à tous les serveurs d'un domaine.

6.4.2 Mutualisation du test

La plate-forme Shibboleth IDP¹⁴ est déployée de façon automatisée sur un serveur de pré-production avec une recette d'installation qui enchaîne un test.

Pour illustrer le principe de réutilisation des composants, le test¹⁵ validant le déploiement est le même que celui utilisé dans la plate-forme de supervision pour vérifier que l>IDP fonctionne. La plate-forme est intéressante à tester car elle rencontre plusieurs types de dysfonctionnement possibles : elle ne démarre pas suite à une erreur de syntaxe dans un fichier XML de configuration, elle démarre mais elle ne fonctionne pas correctement en raisons de mauvaises permissions sur des fichiers, elle démarre et fonctionne mais s'arrête dès qu'une requête particulière est effectuée (ce dernier cas provient de la grande souplesse de configuration de Shibboleth : une partie de la configuration est un programme Javascript uniquement évalué à l'exécution).

7 Conclusion

L'adoption combinée de ces trois pratiques donne aux administrateurs plus de contrôle sur leur infrastructure. Une fois la démarche amorcée sur les systèmes critiques, on peut l'étendre progressivement aux autres systèmes sans obligatoirement s'appuyer sur une démarche ITIL formelle. En intégrant la détection des changements, les référentiels et l'automatisation du déploiement dans les actions d'administration, et en améliorant continuellement ces pratiques, les comportements changent peu à peu, et le fonctionnement de l'infrastructure se stabilise.

Bibliographie

- [1] Kevin Behr, Gene Kim et George Spafford. The Visible Ops Handbook: Implementing ITIL in 4 Practical and Auditable Steps. Information Technology Process Institute, 2005
- [2] Great Britain Cabinet Office. ITIL Service Transition. The Stationery Office, 2011

12. <http://auxesis.github.io/cucumber-nagios/>

13. Comme protéger 10.000 serveurs d'un coup en factorisant la sécurisation de la configuration du service SSH

14. <http://shibboleth.net/products/identity-provider.html>

15. <https://github.com/jean1/shibtest>