

Apprivoiser la complexité de l'exploitation

Jean Benoit

Université de Strasbourg

Direction Informatique

12/12/2013

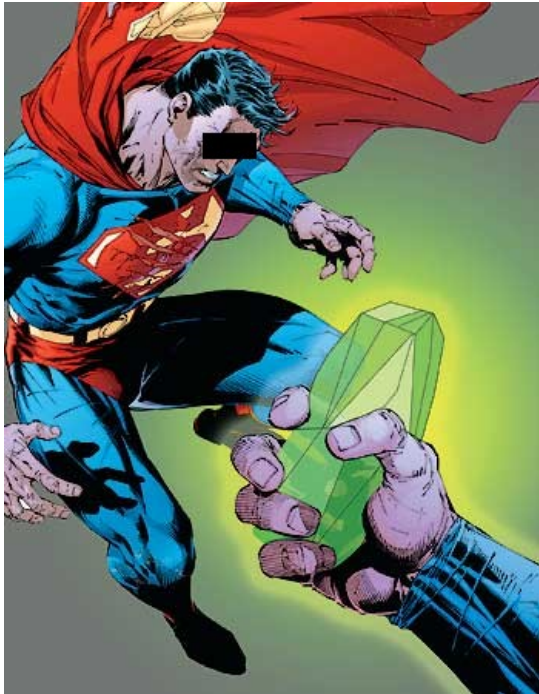
échanges
espiritualidad
insertion
perspectives
mutualisation
réussite
ouverture
fondation
CHEMISTRY
graduation
biology
 $E = mc^2$
RECHERCHE
SYNERGIES
COMPETENCES
pi
TECHNOLOGY
doctorat
enseignement supérieur
protechnologies
axiome
mécanique
management
droit
excellence
savoirs
wissenschaft
bibliothèques
médecine
tesis
théologie
gravitation
idéaux
connaissances
musica
langage
INTERNATIONAL
heuristique
partenariat
HISTOIRE
physique
mécanique quantique
insertion
PLURIDISCIPLINARITÉ
sciences
humain
ambition
quantique
MASTER
cultures
NETWORK

- ▶ Démarche
- ▶ ITIL et Visible Ops
- ▶ Implémentation
- ▶ Synthèse

- ▶ Objectif : fiabiliser le fonctionnement de l'infrastructure et des applications
- ▶ Combinaison de 3 pratiques complémentaires
 - Détection du changement
 - Inventaire
 - Déploiement reproductible
- ▶ Démarche résultant
 - de l'application de ces pratiques
 - d'une réflexion sur leur fonctionnement (pourquoi ça marche?)
- ▶ Que propose ITIL ?

- ▶ Démarche
- ▶ **ITIL et Visible Ops**
- ▶ Implémentation
- ▶ Synthèse

« ITIL manuals are like **kryptonite** to enthusiasts »



(BOFH, épisode 34)

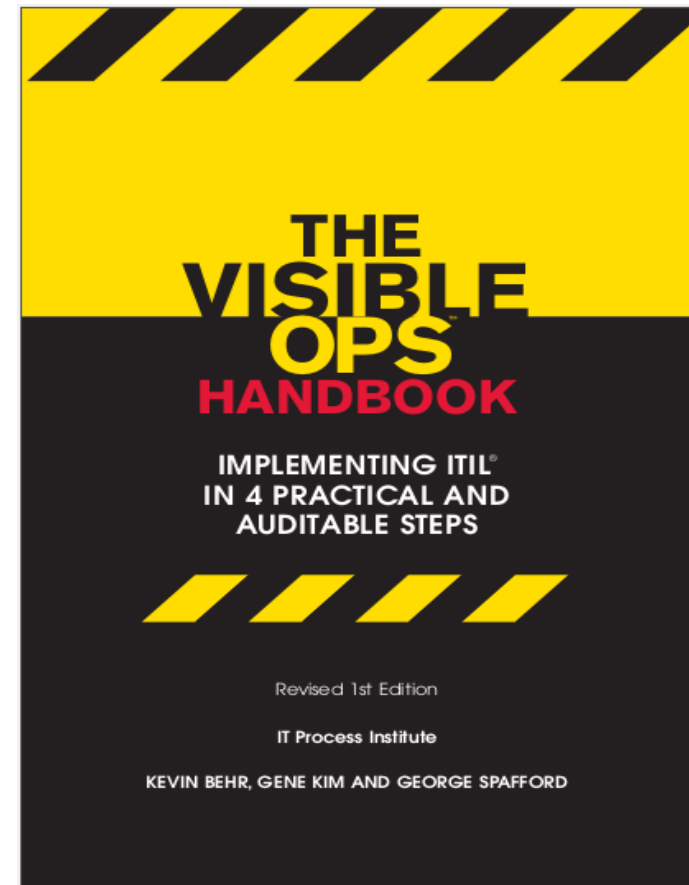
- ▶ Recueil de bonnes pratiques en 5 volumes
entre 180 et 400 pages chacun (>1500 pages au total)
- ▶ Beaucoup de concepts intéressants :
 - Gestion des configurations
 - Gestion des changements
 - Gestion de la capacité
 - Gestion des incidents etc.

► Mais...

- Lourd : nombreux rôles, processus, risque de création de silos etc.
- ITIL n'est pas prescriptif
 - Il n'indique pas par où commencer
 - Il n'indique pas ce qui est important
- Certaines pratiques proposées ont été mises en œuvre bien avant qu'ITIL n'ait été conçu
 - Aucun historique, pas de contexte d'application
- Peu d'exemples d'implémentation

Une autre approche : The Visible Ops Handbook

- ▶ 100 pages
- ▶ 38 pages essentielles
- ▶ Implémentations
 - Exemples concrets
 - Témoignages, retours d'expériences
- ▶ Décomposé en 4 phases
- ▶ Ce livre a été écrit à cause d'un usage particulier de Tripwire





- ▶ Outil de sécurité : vérifie l'intégrité d'un système
 - Calcule la signature de tous les fichiers
 - Compare aux valeurs de référence de l'installation
 - Détecte automatiquement les changements
- ▶ S'il y a un changement, Tripwire lève une alerte

Tripwire → Visible Ops

Modified object name: /etc/passwd-

Property:	Expected	Observed
* Inode Number	2099877	2101089
* Size	2714	2772
* Modify Time	Sun Nov 18 13:32:14 2012	Thu Dec 13 20:40:28 2012
* CRC32	C20ayv	AdbhBT
* MD5	B4D5f3WjCmExJ+c1nPNB5u	B8wE3wV3ojnztY7Vp4XUn6

Modified object name: /etc/shadow-

Property:	Expected	Observed
* Size	1762	1887
* Modify Time	Thu Nov 1 19:23:18 2012	Sun Nov 18 13:32:22 2012
* CRC32	BHLRTD	DsS9ba
* MD5	Df9YtwPJbrd2TLZcajWpVY	BU9hvSUSEUj39LgpRQGmfj

Rule Name: Security Control (/etc/passwd)
Severity Level: 66

Modified Objects: 1

Modified object name: /etc/passwd

Property:	Expected	Observed
* Inode Number	2101057	2101094

- ▶ Gene Kim, l'auteur de Tripwire, s'étonne de voir que...
 - Tripwire n'est pas utilisé comme outil de sécurité dans certaines entreprises...
 - ... mais qu'il est utilisé pour augmenter la fiabilité des services
- ▶ Il constate aussi, en comparant les organisations, que...
... certaines sont plus efficaces que d'autres
- ▶ Les plus efficaces auraient des pratiques spécifiques
 - Détection du changement, inventaire, déploiement automatisé
- ▶ Ces constats ont conduit à l'écriture de « Visible Ops »

1. Stabiliser le patient
2. Inventaire / attraper les artefacts fragiles
3. Construire une bibliothèque de déploiement reproductible
4. Amélioration continue

Phase 1 : stabiliser le patient



- ▶ Lister les systèmes critiques générant le plus d'interventions non-planifiées
 - Périmètre restreint
- ▶ Mise en place d'une « clôture » autour
 - Plus de changement sur ces systèmes sans autorisation
 - Publication de la politique :
Post-it, /etc/motd (bannière système)



- ▶ Principe : « faire confiance, mais vérifier »
 - Les admin peuvent faire des changements
 - Outil de détection
- ▶ Électrification de la clôture : détection des changements
 - Est-ce un changement autorisé ?
 - non : investigation
- ▶ Créer une culture du changement
 - Détection : **catalyseur** pour la **gestion des problèmes**
- ▶ Organiser les changements
 - Fenêtres de maintenance
 - Communication

- ▶ Processus de gestion des changements formel ou pas ?
 - Stricte application des fenêtres de maintenance
 - Changement détecté en dehors = changement non autorisé
 - Il n'est pas nécessaire de rapprocher les changements détectés et les changements non autorisés
 - → le processus formel de gestion des changements est inutile (pas de formulaire, de comité, de processus d'approbation etc.)
 - Gestion formelle piloté par une équipe des changements
 - Suivi des changements majeurs :
demande initiale, décisions → tickets
 - Calendrier des changements
 - Communication aux personnes concernées

Phase 2 : “catch & release”



Attrapez les tous !

- ▶ Les gardes-forestiers dans les parcs naturels attrapent tous les animaux, les pèsent, les répertorient, les baguent, et les relâchent
- ▶ Idem avec les serveurs, les applications etc.
- ▶ Questions : à quoi sert-il ? Que se passe-t-il s’il crashe ? Est-il sauvegardé ? Peut-on le reconstruire ? Quelles sont ses dépendances ? Etc.

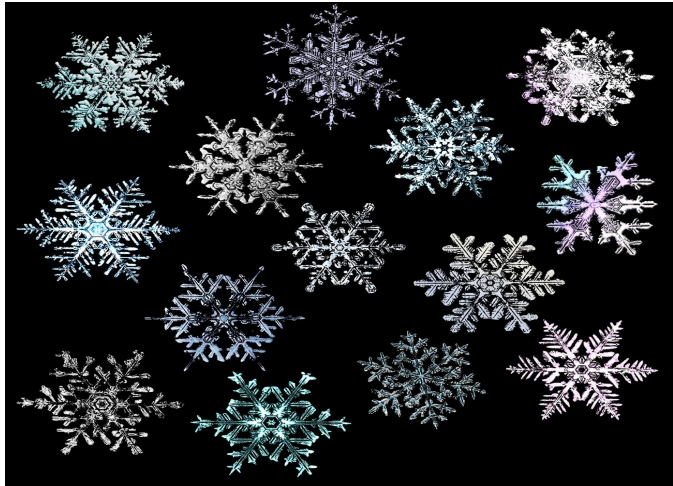
→ Inventaire détaillé

Objet : Configuration Items (CI)

Base d’inventaire : Configuration Management Database (CMDB)

Phase 3 : bibliothèque de déploiements reproductibles





Serveur en production

≈ œuvre d'art unique

- Dérive de la configuration
- Application de patches
→ Non-reproductibilité

- ▶ Principe directeur : re-cr er est plus simple que r parer
 - Automatisation du d ploiement
 - “Bare-metal build” : d ploiement complet sur un serveur nu
- ▶ Changement dans la fa on de fonctionner
 - On passe d’un mode r actif   un mode pro-actif
 - Davantage de temps pass  sur la conception du d ploiement
 - D ploiement : assemblage de composants standards
 - Serveur Web : composant Linux + composant Apache
 - Serveur BDD : composant Linux + composant Mysql
 - D ploiement = code
 - Factorisation, Version, Tests etc.

- ▶ Mesurer
- ▶ Étendre le périmètre



- ▶ Démarche
- ▶ ITIL et Visible Ops
- ▶ **Implémentation**
- ▶ Synthèse

Comment faire dans mon université / labo ?

- ▶ Dans notre contexte, les équipes sont souvent plus petites,
Il faut une approche plus pragmatique
- ▶ Détection des changements participative
- ▶ Fédération des systèmes d'inventaires
- ▶ Déploiement reproductibles ciblés

Exemple : implémentation de la détection sur des serveurs

▶ Mise en place :

```
cd /etc
```

```
git init
```

```
git add .
```

```
git commit -m 'premier commit'
```

▶ Toutes les minutes en cron :

```
(cd /etc; git status; git diff) | keepstate | mail sysadmins
```

▶ Obligation de faire des commits (mais automatisable)

Résultat : changement détecté

```
Date: Fri, 20 Sep 2013 16:31:04 +0200 (CEST)
From: Cron Daemon
To: joe@example.com
Subject: fw2 change detection report
```

```
* git status for /etc :
```

```
M pf.conf
```

```
* git diff for /local :
```

```
diff --git a/pf.conf b/pf.conf
```

```
-# Acces APPLI1 sur base APPLI1TEST
```

```
-pass inet proto tcp from { $appli1 } to { $bdd1 }
```

```
+# Acces APPLI1 sur base APPLI1TEST et APPLI1
```

```
+pass inet proto tcp from { $appli1 } to { $bdd1 $bdd3 }
```

- ▶ Participative = au sein de la même communauté
- ▶ Seules les personnes impliquées techniquement sont notifiées (administrateurs système, réseau, administrateurs d'applications ...)
- ▶ La question n'est pas : « qui ? »...
... mais : « Quoi ? » Qu'est-ce qui a été modifié ?
- ▶ Donner de la visibilité aux administrateurs
- ▶ Comprendre ce qui s'est passé...
... pour trouver la cause d'un dysfonctionnement



- ▶ « Je vois ce que les autres changent »
- ▶ « Les autres voient ce que je change »
- ▶ Les tournées de croissants
- ▶ Cela conduit chacun à prévenir et préparer davantage
 - Fenêtre de maintenance
 - Confiance plus grande
- ▶ **Responsabilisation**
- ▶ **Cultiver l'autonomie et la compétence**

- ▶ Il y aura toujours **plusieurs référentiels** :
 - Réseau
 - Serveurs...
- ▶ Une CMDB centraliserait tous ces référentiels ?
 - Coûteux à mettre en place
 - La qualité des données est difficile à maintenir
- ▶ Principe minimaliste : quel est le **minimum** de données de référence nécessaire ?

- ▶ Fonctions nécessaires : énumération, tags
- ▶ S'appuyer sur des outils existants
 - GLPI, framework de déploiement, etc.
- ▶ Utiliser **toutes** les sources (par ex. schéma de nommage DNS)
- ▶ L'ensemble de ces outils constitue le **système d'information de l'infrastructure**
- ▶ Interactions entre les outils : couplage lâche
 - Mécanismes simples pour lier les informations entre référentiels (web service par exemple)
- ▶ Évolution du référentiel : scripts développés en fonction des besoins

Alimentation de l'inventaire manuelle ou automatique ?

- ▶ Automatiser quand c'est possible : agent d'inventaire etc.
- ▶ Saisie manuelle des éléments abstraits : lien serveur – service
- ▶ Au départ : documenter l'infra dans le wiki
 - La doc est nécessaire mais pas suffisante
 - Il y a une dérive rapide entre la doc et la réalité
- ▶ Idée : consulter la recette de déploiement automatisé
 - Les informations y sont forcément à jour :
 - Serveur de base de données utilisé
 - Port TCP...

Exemple d'alimentation du référentiel par l'outil de déploiement

```
# 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
```

- ▶ En priorité
 - Les serveurs critiques
 - Les serveurs en cluster
- ▶ Écrire une recette de déploiement
 - Un effort coûteux mais rentable
 - Résultat reproductible et testable
 - Formalisation d'actions autrefois manuelles, documentation
 - Traçabilité grâce à la gestion de version
- ▶ Exemple : le cluster des 8 relais de messagerie Osiris

- ▶ Prendre un outil existant avec un langage spécifique (DSL) :
Chef, Puppet etc.
- ▶ Outil structurant : notion de ressource
 - Fichier
 - Template
 - Package
 - Entrée dans la crontab...
- ▶ Recette = fonction
 - ré-utilisabilité des composants

- ▶ Étendre progressivement le déploiement automatisé
- ▶ Peer-programming des recettes
- ▶ Écrire des tests d'infrastructure
 - Scripts
 - Frameworks (Cucumber)
- ▶ Rapprocher les développeurs et les administrateurs
 - Techniques commune
 - Intégration continue
- ▶ Déployer l'agent de déploiement sur tous les serveurs
 - Même si le service n'est pas déployé automatiquement
 - Services de base uniquement (SSH, syslog, supervision...)

- ▶ Présentation « Automatisation de l'administration de 700 serveurs avec Chef »
- ▶ Par Christophe PALANCHE et Alain HEINRICH
- ▶ Jeudi 12 Décembre 2013
- ▶ Salle Berlioz
- ▶ 16h00

- ▶ Déploiement de Shibboleth : 3 possibilités d'erreur
 - Shibboleth ne démarre pas suite à une erreur de syntaxe dans le fichier de configuration XML
 - Shibboleth démarre mais dysfonctionne à cause de droits d'accès incorrects sur certains fichiers
 - Shibboleth démarre mais s'arrête dès qu'une requête est effectuée en raison d'un morceau de configuration en Javascript évalué à l'exécution !
- ▶ Environnement de test : compte, application authentifiée
- ▶ Le script enchaîne la connexion à l'application, les redirections Shibboleth et CAS, et retourne un état (vrai ou faux)
- ▶ Script mutualisé : test du déploiement, supervision

Exemple de déploiement piloté par l'inventaire

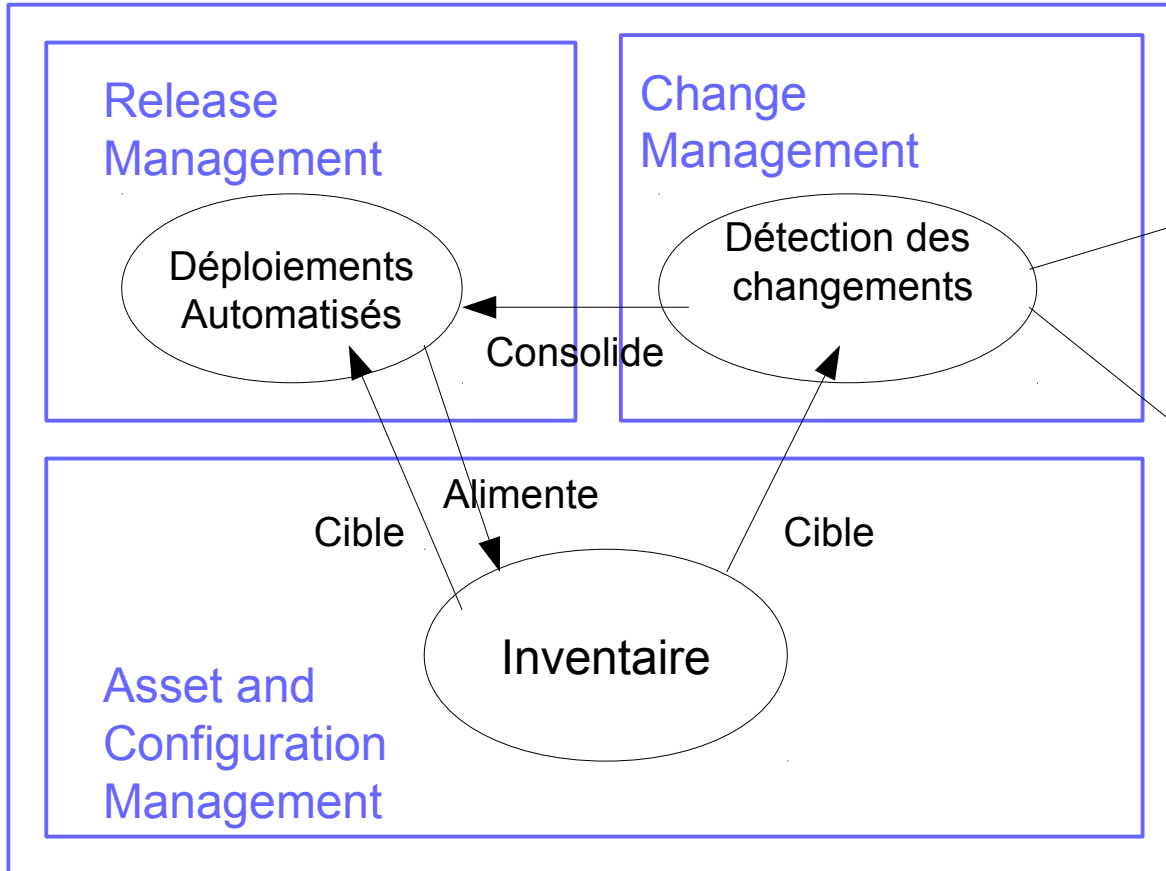
```
knife ssh \  
  "name:fw-*" \  
  "chef-client -o 'recipe[changedetection::install]'"
```

- ▶ Commande centralisée sur le serveur de déploiement
- ▶ S'exécute sur tous les firewalls (machines nommées fw-*)
- ▶ Lance l'installation du logiciel de détection des changements

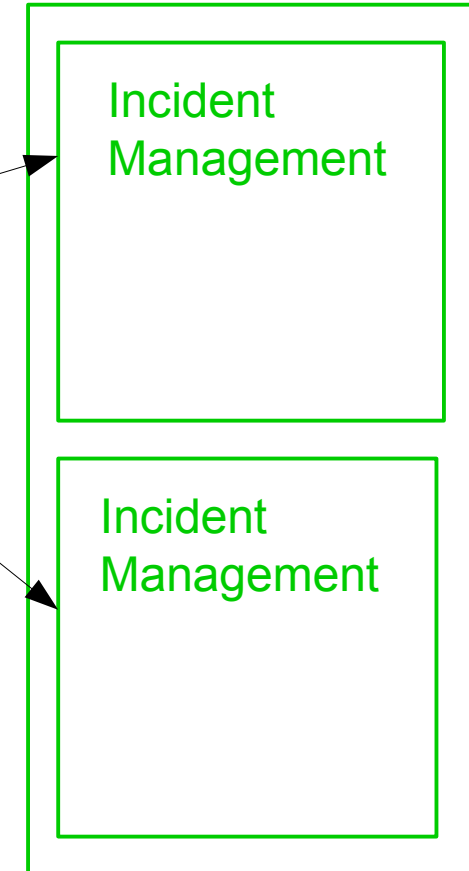
- ▶ SI infra = socle des nombreuses automatisations
- ▶ L'inventaire fournit une liste de serveurs suivant un critère
 - Pour cibler le déploiement automatisé
 - La détection du changement est-elle déployée sur les serveurs ?
- ▶ Le déploiement alimente l'inventaire avec des données à jour
- ▶ La détection du changement :
 - Détecte une dérive par rapport à la recette
 - Permet l'intégration des changements détectés au déploiement
- ▶ Détection + SI infra : facilite la résolution des problèmes
- ▶ Détection + déploiement : facilite la restauration du service

Une carte

Service Transition



Service Operation



Facilite

Facilite

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

- ▶ Commande centralisée sur le serveur de déploiement
- ▶ S'exécute sur tous les firewalls (machines nommées fw-*)
- ▶ Lance l'installation du logiciel de détection des changements

- ▶ Démarche
- ▶ ITIL et Visible Ops
- ▶ Implémentation
- ▶ Synthèse

- ▶ Pour plus de fiabilité des services...
- ▶ Appréhender les concepts importants
 - Lire ITIL, Visible Ops etc.
 - Confronter les concepts à la réalité = pratiquer
- ▶ Maintenir un Système d'Information de l'infrastructure
 - C'est le socle
 - Le faire évoluer progressivement
- ▶ Faire de la détection des changements
 - Pour comprendre ce qui se passe
- ▶ Automatiser les déploiements
 - Pour stabiliser l'infrastructure