Automatisation de l’administration système

Plan

- Problèmatique : trop de systèmes, trop de solutions
- Typage des solutions
- Puppet : gestion de configuration de systèmes
- Capistrano : déploiement d’applications
- Utilisation dans un service informatique

 pierre.gambarotto@  .fr

Problématique : pourquoi plus de systèmes à gérer ?

- plus de services à gérer
- applications web : transfert du coût de gestion du poste client au serveur
- complexité des infrastrucutres : environnements de prod, pré-prod, test, dev ...
- généralisation du poste client

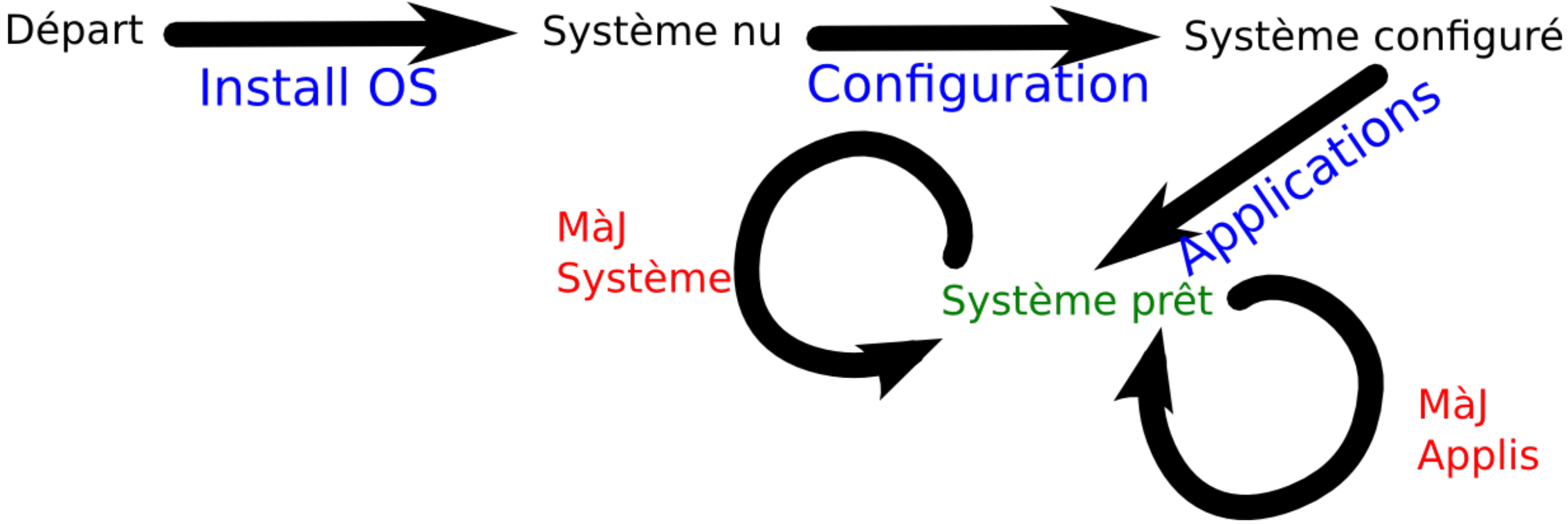
Les solutions triviales et leurs inconvénients :

- acheter des machines supplémentaires : coût en argent, mauvaise utilisation des ressources matérielles
- installer les nouveaux services sur des systèmes existant : adhérence des applications, frein à la mise à jour : coût maintenance/sécurité, mauvaise utilisation des ressources humaines

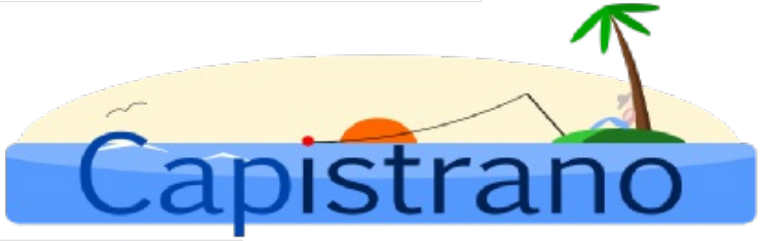
Rationalisation des architectures

- plus simple de gérer un système avec un nombre réduit de services (1 !)
- virtualisation des systèmes : meilleure utilisation des ressources matèrielles
- cloud/hébergement : plus de ressources matérielle

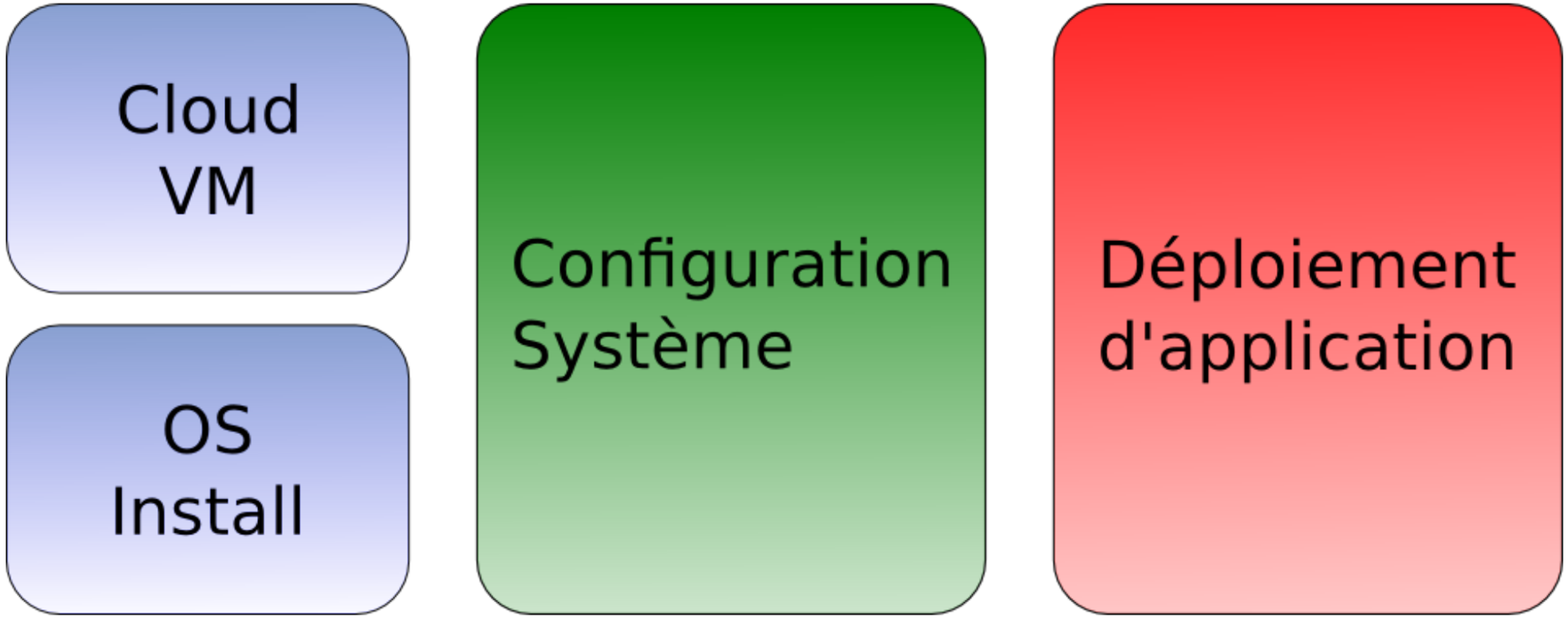
Chaque système …



Les solutions techniques pour automatiser



classification en 3 couches



Installer le système de base

2 grandes catégories :

- machine réelle
- machine virtuelle : at home et cloud

Prérequis :

- accès à la machine physique ou accès authentifié au socle (ou compte pour le cloud)

Résultat :

- un système configuré, administrable par accès réseau

Gestion par prototype : installer un nouveau système à partir de la sauvegarde d’un ancien

- ne gère pas les évolutions : dérive des configurations
- la configuration est à faire à chaque fois manuellement

Configurer/mettre à jour

scripter la configuration d’un système , une configuration = un script

Prérequis

- système configuré, accessible par le réseau
- un compte administrateur sur le système

Résultat

- gérer plusieurs systèmes similaires avec la même configuration
- automatiser certaines mises à jour
- rassembler dans un même endroit la description de tout le parc
- passage de gestion de machine à gestion de parc
- automatisation de l’intégration avec la supervision

Déploiement d’applications délégué à un tiers

Déléguer une partie de la gestion d’un système à un tiers

Cas typique : application web

- serveur http, serveur BD : installation, configuration et mises à jour gérées par une personne
- déploiements gérés par une autre personne

Prérequis

- serveur(s) installé(s) et configuré(s)
- séparation des comptes d’administrateur système et déploiement d’application

Résultat

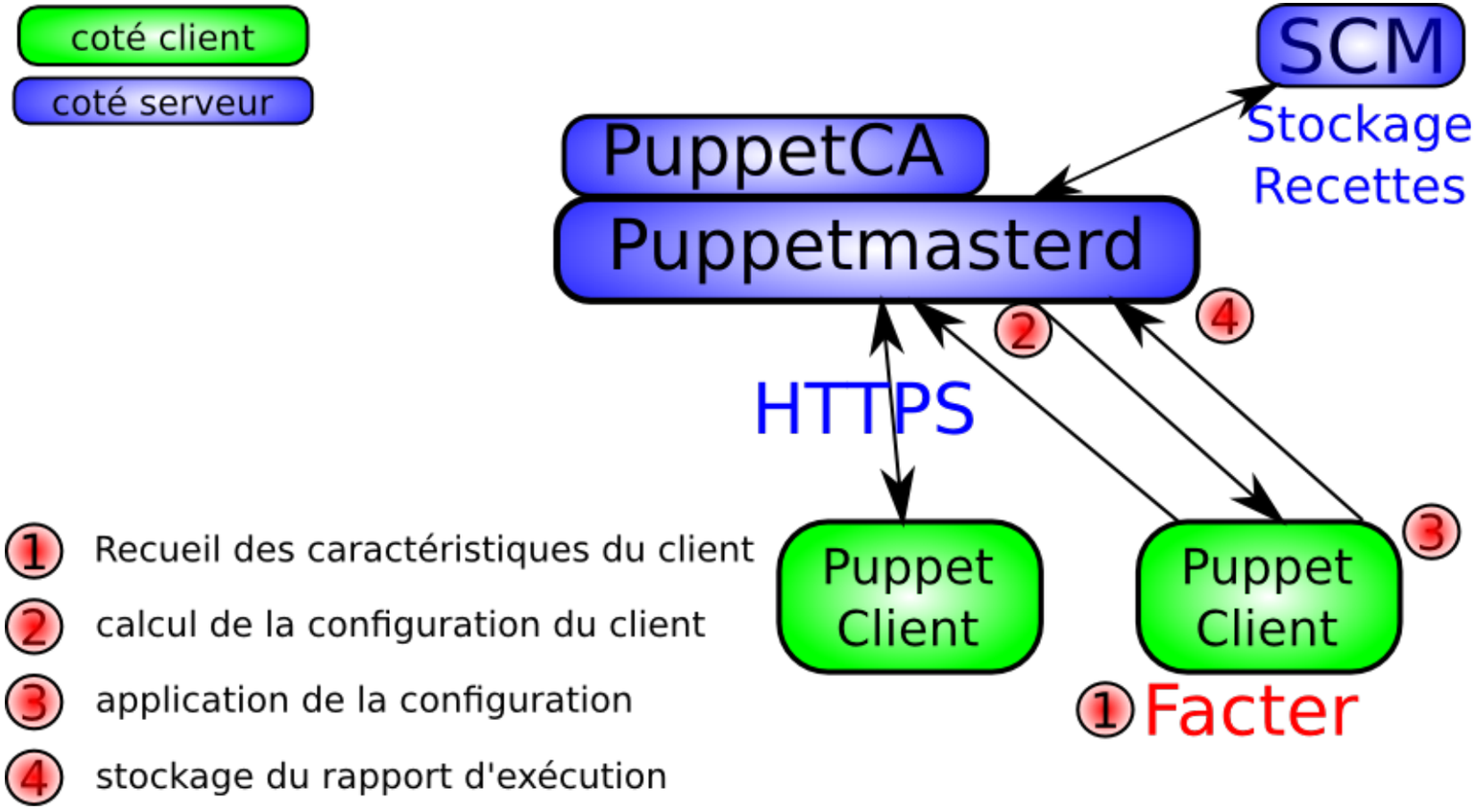
- bonne entente des Administrateurs Systèmes et des Développeurs
- gestion d’architecture de déploiement complexes

Études d’exemples

Puppet : gestion de configuration

- Capistrano : déploiement d’applications

Puppet: gestion de configuration



- architecture client/serveur
- configuration : décrite avec un DSL
- configuration : implémentation de ressources
- ressources : package, file, exec, service ...

Puppet: DSL

- DSL : une configuration est exprimée dans un langage spécifique

Ressource : brique de base

- type de la ressource : fixe les paramètres de configuration
- réaliser la ressource : donner une valeur aux paramètres

```
file { "/etc/passwd":  # nom unique donné à la ressource
  owner => root, # paramètre owner, valeur donnée : root
  group => root,
  mode => 644
}
```

Type de ressources

- `File` : gestion de fichier, le contenu d'un fichier peut être défini par un template. Le serveur `puppetmasterd` sert de serveur de fichier.
- `Exec` : exécution de script
- `Package` : installation de paquetage
- `User` : définition d'utilisateur
- bien d'autres disponibles
- extensible

Langage complet

```
1 $ssh = $operatingsystem ? { # $operatingsystem : remonté par facter
2   (redhat|fedora) => 'openssh-server',
3   (debian|ubuntu) => 'ssh',
4   default => 'openssh'
5 }
6
7 package { $ssh: # $ssh est la variable définie ci-dessus
8   ensure => installed,
9 }
```

- variables, conditionnelles, gestion de chaînes de caractères, tableaux de valeurs ...
- sur le client, `facter` collecte des données exposées sous forme de variables.

Gestion des dépendances entre les ressources

```
file { '/etc/ssh/sshd_config':
  source => 'puppet:///modules/sshd/sshd_config', # on récupère le contenu du fichier sur le serveur
  owner => 'root',
  group => 'root',
  mode => '640',
  notify => Service['sshd'], # redémarage de sshd après changement de ce fichier
  require => Package[$ssh], # on force l'ordre de résolution, le paquetage avant le fichier de configuration
}

service { 'sshd':
  ensure => running,
  enable => true, # actif au boot de la machine
  hasstatus => true,
  hasrestart => true,
}

```

Puppet : organisation des descriptions

par défaut : les ressource sont réalisées sur tous les clients puppet

```
1  node 'machine.domaine.tld' {
2    # restriction de la réalisation de
3    # ressources à un système spécifique
4  }
5
6  class NomClasse { # Définit un espace de nommage }
7  include NomClasse # réalisation des ressources définies dans la classe
8
9  define mon_type($var1, $var2) {
10   # définition d'une collection de ressources
11   # utilisant var1 et var2
12 }
13 mon_type { nom : # réalisation du type personnalisé mon_type
14   var1 => valeur,
15   var2 => valeur
16 }
```

Configuration d’un produit complet

- ensemble de classes et de types de ressources personnalisés
- définition d’une collection de ressources
- fichiers et template définissant le contenu des ressources `File`

Module

- distribution de la configuration d’un produit complet
- par exemple :
  - un serveur Apache et un site virtuel
  - un serveur de base de données et une base
  - un serveur ldap maître et un serveur ldap esclave

Kerberos

Serveur esclave :

```
node 'krb-slave.enseeiht.fr' {
  include kerberos

  $kerberos_realm="INP-TOULOUSE.FR"
  kerberos::server{$kerberos_realm:
    master => "krb-master.enseeiht.fr",
    password => "...",
    domains => ["enseeiht.fr", ".enseeiht.fr",  "ensat.fr", ".ensat.fr",
    "ensiacet.fr", ".ensiacet.fr","inp-toulouse.fr", ".inp-toulouse.fr"],
    default_domain => "inp-toulouse.fr"
  }
  # extrait, manque conf des principaux kerberos
}

```

Kerberos

Serveur maître :

```
node 'krb-master.enseeiht.fr' {
  include kerberos

  $kerberos_realm="INP-TOULOUSE.FR"
  kerberos::server{$kerberos_realm:
    password => "...",
    domains => ["enseeiht.fr", ".enseeiht.fr",  "ensat.fr", ".ensat.fr",
    "ensiacet.fr", ".ensiacet.fr","inp-toulouse.fr", ".inp-toulouse.fr"],
    default_domain => "inp-toulouse.fr"
  }
  # extrait, manque conf des principaux kerberos
}

```

Kerberos : export de ressource des esclaves vers le maître

```
1  class kerberos # extrait
2    define server($realm="", $password, $domains="",
3                  $master=true, $default_domain="")
4    {
5      case $master {
6        true: { # définition config pour le serveur maître }
7        default:{
8          # slave server with master defined
9          @@file{"/etc/krb5kdc/$realm_real/servers/$fqdn":
10           tag => "kerberos_$realm_real",
11           ensure => present,
12           owner => root, group => root, mode => 640,
13         }
14       ... other stuff
15     }
16   }
17 }
18
```

Gestion de parc

- `@@file{ ... }` : ressource définie sur un client puppet, réalisée sur un autre client puppet
- les constituants définissant la ressource (ici de type `File`) sont stockés sur le serveur

Kerberos : réalisation de la ressource définie sur les esclaves sur le maître :

```
 1 class kerberos # extrait
 2   define server($realm="",
 3                 $password,
 4                 $domains="",
 5                 $master=true,
 6                 $default_domain="")
 7   {
 8     case $master {
 9       true: {
10         File <<| tag == "kerberos_$realm_real" |>>{
11         }
12         # other stuff
13       }
14       default:{ # slave server with master defined }
15     }
16   }
17 }
18
```

Gestion de parc

- réalisation de ressources exportées : `File <<| ... filtre ... |>> {}`
- les informations stockées par un client puppet sur le serveur sont utilisées pour définir la configuration d'un autre client puppet

---

- Sur le maître, un fichier portant le nom de l'esclave est crée dans `/etc/krb5kdc/$realm_real/servers`.
- le script de propagation maître => esclaves peut maintenant utiliser cette information

Puppet : bilan

Puppet permet de rédiger des procédures d’installation et de configuration complètes:

Avantages

- la documentation de la procédure est la procédure elle même
- la mise à jour de la procédure permet la mise à jour de tous les serveurs sur lesquels elle est employée.
- caractère open-source : la communauté redistribue des définitions complètes de modules.
- gestion centralisée : la configuration d’un client peut être définie à partir de celles d’autres clients
- permet à un grand nombre de personnes de collaborer

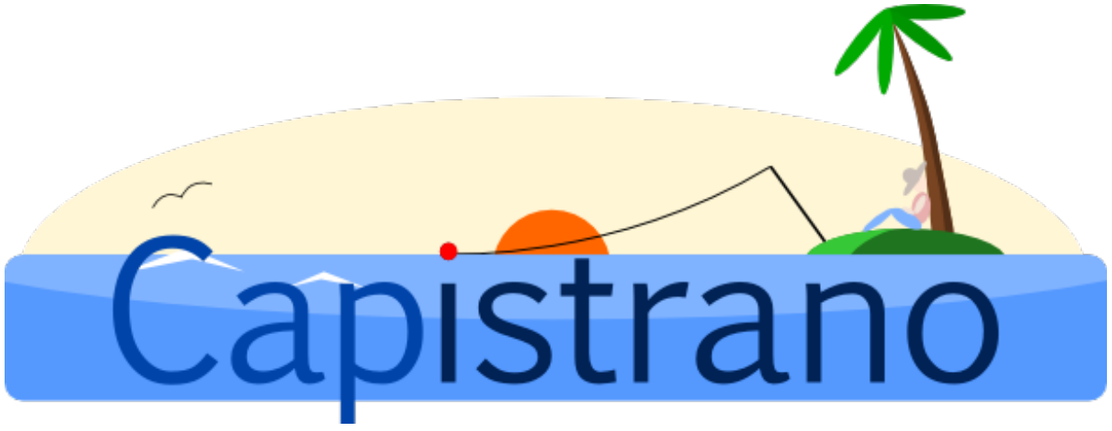
Inconvénients

- courbe d’apprentissage longue
- architecture structurante, complexe à mettre en oeuvre au niveau organisation

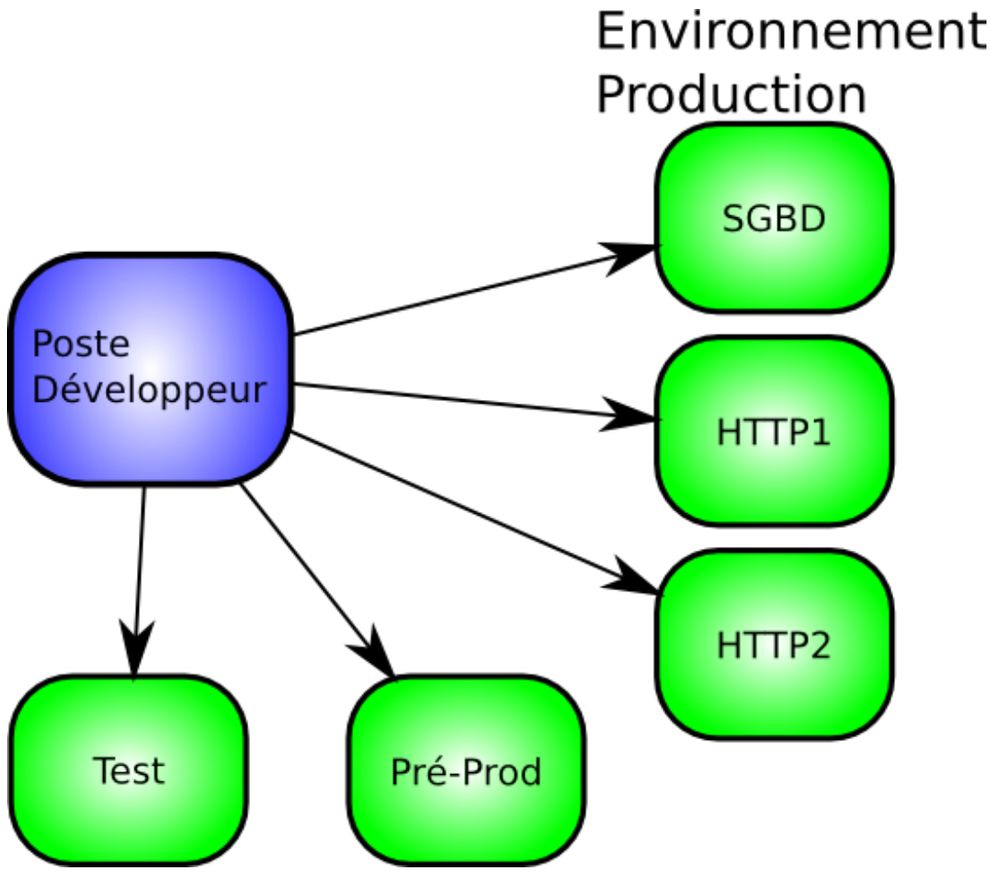
Études d’exemples

- Puppet : gestion de configuration

Capistrano : déploiement d’applications



Capistrano



- librairie ruby
- permet de décrire et d'exécuter des scripts sur plusieurs machines
- emploi typique : déployer une application web sur des systèmes déjà configurés
- gestion de tâches interdépendantes (à la Makefile)
- surcouche SSH pour la gestion des connexions
- gestion par transaction possible

Capistrano : syntaxe

fichier `Capfile` :

```
role :db, 'postgres.domain.tld' # définition de serveurs avec leur rôle associé
role :app, 'backend.domain.tld'
role :web, 'frontend.domain.tld'
set :user, "pierre" # définition de variable

desc "Description de la tâche" # définition d'une tâche
task :my_task, :roles =>:app,:web do # restriction de la tâche aux serveurs remplissant les rôles spécifiés
  run "echo #{user} fait des choses > /tmp/example"
end

after("another_task, "my_task") # ordonnancement des tâches
```

Invocation : `cap my_task`

L'action de la tâche `my_task` est exécuté sur les serveurs `:app`(`backend.domain.tld`) et `:web`(`frontend.domain.tld`) avec le login `pierre`.

Capistrano : librairie de base

Actions possibles dans une tâche capistrano :

- exécution de commande sur un serveur distant : `run`, `parallel`
- transfert de fichiers : local vers distant : `put`, `upload`
- transfert de fichiers : distant vers local : `get`, `download`

Capistrano est fourni par défaut avec des tâches adaptées aux conventions suivantes :

- tout le code réside dans un SCM, type subversion ou git
- les serveurs visés : SGBD, HTTP

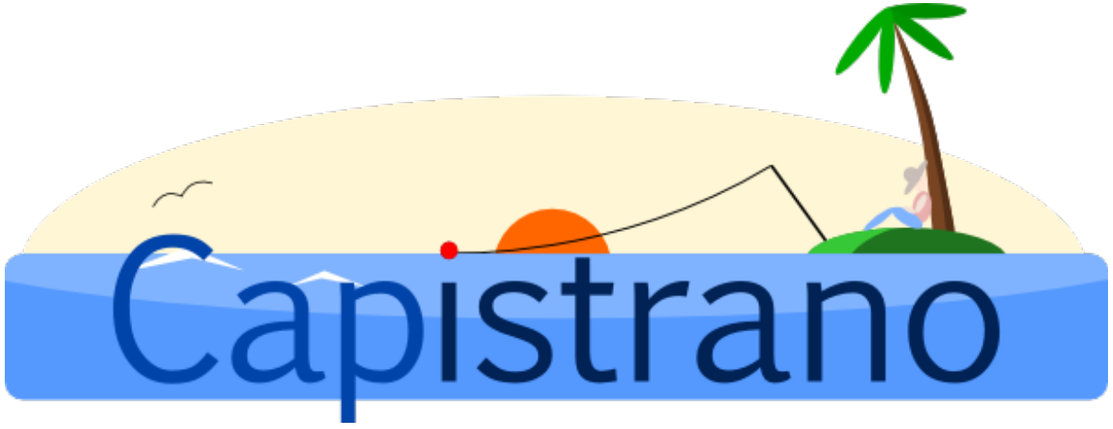
```
1 cap deploy:setup # initialisation : création de répertoire
2
3 cap deploy:update # déploiement du code à partir de la dernière version du SCM
4
5 cap deploy:start, deploy:stop, deploy:restart # contrôle de l'exécution de l'application
6
7 cap deploy:revert # retour à la version précédente
```

Capistrano : bilan

Capistrano permet d’automatiser le déploiement d’une application dans des environnements complexes (multi-serveurs).

Les plus

- simple : makefile+ssh
- fait gagner beaucoup de temps
- la configuration par défaut est utilisable sans comprendre le produit
- communauté : tâches spécifiques pour certains environnements/déploiements spécifiques



Les moins

- apprentissage pour étendre les tâches par défaut
- on peut gérer entièrement un serveur distant : ce n’est pas le but !

Récapitulatif des bénéfices de chaque type de solution

Installation initiale du système

- réduction du coût d’installation d’une machine
- utilisation optimisée des ressources matérielles
- 1 système == 1 service : réduction du couplage, facilité d’adminitration/évolution

Configuration des systèmes

- centralisation des descriptions : modulariser, réutiliser
- gestion de parc : automatiser l’inscription à la supervision

Délégation de déploiement

- séparation des rôles développeur/administrateur
- communication par spécification d’interface

Comment intégrer ces produits dans son service

Politique d’un service informatique

- nombre de personnes
- centralisation
- organisation (hiérarchie)

Méthodologies : COMMUNIQUER !

- approche de la base : faible nombre, peu centralisé, organisation lâche
- approche descendante : organisation forte, centralisée, nombre important
- la réalité est souvent entre les extrêmes ...

Faire intervenir les solutions

- collaboration développeur/sysadmin : produit type Capistrano
- approche technique
- nombreux retours d’expérience

ITIL® CMMI®

- approche institutionnelle
- approche formelle : décrire pour mieux gérer
- nécessite un engagement politique
- caractère structurant des produits type Puppet

Conclusion et perspective

Départ

- explosion du nombre de systèmes à gérer
- pléthore de solutions techniques

Arrivée

- des cases où mettre les solutions
- 2 exemples
- comment insérer ces solutions dans son quotidien

Vos questions ?