

Maxime Alexandre
Laurent Bello

<php@backelite.com>

Table des matières

1. Introduction	1
1.1. L'industrialisation de PHP	1
1.2. L'automatisation de tâches	2
2. Quid du déploiement automatisé de sites PHP	2
2.1. Déploiement des fichiers sources	2
2.2. Déploiement des ressources du site	3
2.3. Déploiement des fichiers externes contenant la liste des modifications sur la base de données	3
2.4. Déploiement des diverses autres actions du déploiement	3
3. Outils choisis	4
3.1. SSH	4
3.2. Subversion	4
3.3. Capistrano	4
3.4. Liquibase	4
4. Mise en place	4
4.1. Fonctionnement général	4
4.2. Gestion de la configuration	5
4.3. Mise en place avant déploiement	6
4.4. Déploiement des fichiers du site	7
4.5. Déploiement des fichiers externes de modifications de base de données	7
4.6. Déploiement des diverses autres actions du déploiement	8
5. Conclusion	8
5.1. Ce qu'il reste à améliorer	8
5.2. Inconvénients	8
5.3. Avantages	9

1. Introduction

1.1. L'industrialisation de PHP

PHP a aujourd'hui 15 ans et est utilisé sur plus de 50% des serveurs web. Avec 5 millions de développeurs, c'est la 3ème technologie après Java et C/C++. Les services conçus en PHP sont variés : petit site personnel, site de commerce en ligne, site de réseau social de taille mondiale, scripts de traitements complexes... tout est facilement réalisable avec cette technologie. Il faut désormais que les outils et processus évoluent pour devenir de plus en plus professionnels afin d'améliorer la qualité des livrables, de diminuer le temps de développement et de faciliter son utilisation par les développeurs.

Tous les développeurs et administrateurs connaissent le stress de la mise en production (MEP). Autant le dire, c'est le moment de vérité ; l'instant où l'on va enfin publier au grand jour le travail effectué par toute l'équipe projet et le rendre visible par le monde entier en temps réel. C'est une étape très importante du projet.

Les questions qui nous viennent lors des MEP sont souvent les mêmes : ai-je mis toutes les instructions dans le mode opératoire ? qui va s'occuper de traiter toutes les demandes ? mon site va-t-il encore

fonctionner en production ? comment allons-nous procéder s'il arrive un souci ? comment vérifier que tout va bien se dérouler ? ...

Afin de répondre en grande partie à ces questions, intéressons-nous à ce qu'on appelle l'industrialisation. Ce terme générique regroupe plusieurs étapes du génie logiciel (gestion de la configuration, gestion des sources, mise en production, ...) dont le résultat est de réussir à automatiser toutes les tâches répétitives - exit donc le fichier Word contenant les actions à opérer, relayé de services en services, imprécis et inexploitable. Nous allons nous intéresser plus particulièrement dans cet article à l'automatisation du déploiement de sites en PHP.

1.2. L'automatisation de tâches

Ce sujet est au coeur des entreprises cherchant à optimiser les performances et la qualité de leur production. Rendre automatique une tâche consiste à simplifier son utilisation tout en apportant une garantie sur son niveau de fonctionnement, certes parfois limité, mais indispensable dès qu'on est confronté à des tâches répétitives et/ou demandant une plus grande rigueur.

Cependant vu les progrès de l'informatique moderne, on peut dire que l'automatisation n'a actuellement pas de limite : chaque tâche pourrait se retrouver décrite dans un modèle informatique et traduite dans une suite d'actions automatisées. Oui, cependant il n'est pas forcément bon de "machinifier" toutes les métiers comme le montrent certaines questions permettant de mesurer le pour et le contre :

- Justifier l'intérêt dans le contexte du métier
- Justifier le coût de l'automatisation
- Construire un support durable
- Avoir l'aval des personnes concernées

Pour résumer, l'automatisation ne doit être choisie que si les apports à l'entreprise l'emportent sur les coûts de mises en place et de support. ¹

2. Quid du déploiement automatisé de sites PHP

Un site typique développé en PHP est généralement constitué de plusieurs éléments :

1. les fichiers sources au format texte (.php)
2. les ressources du site (images, documents...)
3. les modifications sur la base de données
4. les diverses actions spécifiques au déploiement

Etudions attentivement chacune des étapes afin d'en comprendre ses détails.

2.1. Déploiement des fichiers sources

Premièrement il est indispensable de savoir précisément la liste des fichiers nécessaires pour le déploiement demandé : nous utilisons donc un outil de version de sources identifiant atomiquement² les fichiers à mettre sur le serveur.

Subversion étant l'outil de gestion de version de référence, nous allons l'utiliser. La bonne pratique est donc de donner un nom à cette version dans le répertoire "tags" une fois finalisée et prête à être déployé.

Exemple : <http://subversion.mycompany.com/mysite/tags/version4.2>

¹Article de Joe McKendrick sur le sujet : <http://www.zdnet.com/blog/service-oriented/when-processes-are-beyond-the-reach-of-automation/5332>

²l'ensemble des sources correspond à une seule et unique version

PHP étant un langage dynamique interprété, une mise à jour simple des fichiers suffit à mettre à jour le site sans aucun redémarrage du serveur web ; il n'y a pas non plus d'étape de compilation. Il faut donc que toutes les modifications sur ces fichiers soient appliquées en un instant unique, traitées dans une seule et même action.

Pour ce faire, il existe plusieurs méthodes :

- i. demander à l'outil de mettre à jour les sources dans le même répertoire que celui de production (DocumentRoot)

```
svn update
```

- ii. utiliser l'outil de gestion de sources afin de télécharger les sources dans un autre dossier. Une fois le téléchargement terminé, simplement changer de répertoire (DocumentRoot).

```
svn export http://[...] version4.2
```

La seconde solution est préconisée car elle s'abstrait des problématiques du versionning de gestion de source inattendues et parfois non prévisibles (conflits, erreurs bloquantes...) et donc largement compromettantes pour la qualité de la MEP. De même, elle peut prendre un certain temps, ce qui corromperait l'unicité des sources.

Note

Etant donné que l'on récupère toutes les sources dans un nouveau répertoire, il ne faut effectuer aucune modification sur les fichiers dans l'ancien répertoire de production car elles seront perdues.

2.2. Déploiement des ressources du site

Un site web intègre des fichiers sources PHP, ainsi que des fichiers de type média (images, sons, vidéos) ou documents (PDF...). Ces fichiers ne sont pas versionnés car ils ne font pas partie des sources du projet en tant que tel - ils peuvent être déposés sur le serveur, téléchargés par un script externe... Il est nécessaire cependant de les conserver dans le site tout au long de sa vie - et de ses MEP.

Nous avons vu lors de la première étape que le dossier principal des sources est remplacé intégralement par un autre : ces ressources non versionnées sont donc retirées du site lors de cette étape. Nous devons donc créer un étape lors du déploiement qui va s'occuper de rendre ces fichiers disponibles, le plus rapidement possible.

2.3. Déploiement des fichiers externes contenant la liste des modifications sur la base de données

La mise à jour d'un site intègre souvent des modifications sur la base de données, que ce soit au niveau de la structure ou des données. Cette modification doit être effectuée au même moment que la mise à jour du code source du site. Il est donc indispensable de versionner ces modifications, tout autant que les fichiers sources PHP.

Chaque modification doit être identifiable par l'outil de versionning afin qu'il puisse mettre à jour la base de données, quelque soit son état initial. Il est nécessaire que le développeur consigne chaque modification dans un fichier texte et non pas appliquer une modification sur son propre environnement de développement, sans quoi cette modification serait perdue.

Tous les fichiers de modification de la base doivent donc faire partie intégrante des sources du projet. Ces fichiers sont donc mis à jour lors de la première étape ("Déploiement des fichiers sources"). Il ne reste plus qu'à appliquer les modifications sur la base via l'outil de versionning de base de données.

2.4. Déploiement des diverses autres actions du déploiement

Les 3 précédentes étapes se suffisent en elles-mêmes dans la plupart des besoins de déploiement. Il est cependant possible que votre MEP nécessite l'application d'actions de maintenance ainsi que des

traitements uniques ou récurrents. Vu le caractère spécifique de ces besoins, nous ne parlons donc pas d'automatisation de ces modifications. Il faut cependant garder en tête que s'il y en a, elles doivent apparaître de quelque façon que ce soit dans les sources du projet.

Exemple : si vous avez des actions récurrentes (type "crontab" à ajouter ou modifier), vous pouvez les indiquer dans un fichier texte reprenant ces tâches et utiliser un outil pour les appliquer pour vous (ou le faire "à la main" si vous n'avez pas d'outil, cependant cela exige une grande rigueur).

Enfin, on peut y ajouter l'envoi de mails à chaque MEP afin d'être notifié des différentes actions appliquées.

3. Outils choisis

3.1. SSH

Toutes les actions sont exécutées via l'outil SSH se connectant automatiquement aux différents serveurs, lançant des actions de façon sécurisée.

3.2. Subversion

Système de versionning centralisé simple et efficace qui répond au besoin de versionning standard (update, copy, commit...).

3.3. Capistrano

Outil développé en Ruby et standardisé pour le déploiement de sites développés avec RubyOnRails. Il intègre nativement :

1. La gestion de plusieurs environnements (dev, préprod, recette, prod) de déploiement et leur configuration associée.
2. La connexion automatique à ces environnements (en utilisant SSH)
3. Plusieurs commandes pré-configurées de déploiement (deploy, deploy:setup, ...)

Note

Afin de faciliter le déploiement sur plusieurs environnements (dev, preprod, prod...) il est nécessaire d'installer le module 'capistrano/ext/multistage'.

3.4. Liquibase

Outil développé en Java de versionning de base de données qui intègre des commandes décrites dans des fichiers XML. Il est compatible avec la plupart des bases de données telles que MySQL, MSSQL, Oracle, PGSQL...

4. Mise en place

4.1. Fonctionnement général

L'outil principal qui va permettre de lancer toutes les tâches est Capistrano. Ce dernier requiert une arborescence de dossiers organisée de telle manière :

```
alexandre@bucarest:/var/www/monsite$ ls -lah
total 24K
drwxrwxr-x  5 alexandre alexandre  4.0K 2009-12-01 20:14 .
drwxr-xr-x 56 alexandre root      4.0K 2010-08-11 15:21 ..
lrwxrwxrwx  1 alexandre alexandre   51 2009-12-01 20:14 current -> /
var/www/monsite/releases/20091201191257
drwxr-xr-x  2 alexandre alexandre  4.0K 2009-12-01 20:15 logs
drwxrwxr-x  3 alexandre alexandre  4.0K 2009-12-10 19:19 releases
-rw-r--r--  1 alexandre alexandre   605 2009-12-01 20:12 monsite
```

```
drwxrwxr-x 4 alexandre alexandre 4.0K 2009-12-01 20:12 shared
```

- `current` : lien symbolique pointant vers les sources du dernier déploiement du répertoire "releases"
- `logs` : répertoire contenant les différents logs des déploiements
- `releases` : répertoire contenant les sources copiées de chaque déploiement
- `shared` : répertoire de ressources partagées, conservées tout au long de la vie du projet
- `monsite` : fichier contenant la configuration Apache pour ce site

Capistrano va récupérer la liste des actions³ dans le fichier principal nommé "Capfile" en Ruby, copié de préférence à la racine du projet (`current`) :

Exemple 1. Fichier Capfile

```
load 'deploy' if respond_to?(:namespace) # cap2 differentiator
Dir['vendor/plugins/*/recipes/*.rb'].each { |plugin| load(plugin) }
load 'config/deploy/common'
load 'config/deploy/deploy.rb'
```

1. Le fichier `common` définit la configuration générique pour tous les environnements.
2. Le fichier `deploy.rb` contient donc la liste des recettes automatisées.

4.2. Gestion de la configuration

La configuration des environnements est contenue dans les fichiers du projet dans un répertoire précis : `config/deploy`. Cette dernière est composée de la configuration globale et d'une configuration spécifique à chaque environnement.

```
alexandre@bucarest:/var/www/monsite$ ls -l current/html/config/deploy/
total 40K
drwxrwxr-x 3 alexandre alexandre 4.0K 2010-07-30 11:08 .
drwxrwxr-x 5 alexandre alexandre 4.0K 2010-07-30 11:08 ..
-rw-r--r-- 1 alexandre alexandre 1.3K 2010-05-21 12:51 common
-rw-r--r-- 1 alexandre alexandre 129 2010-07-30 11:08 dev
-rw-r--r-- 1 alexandre alexandre 112 2010-04-28 17:59 prod1
-rw-r--r-- 1 alexandre alexandre 112 2010-04-28 17:59 prod2
-rw-r--r-- 1 alexandre alexandre 282 2009-12-09 09:55 staging
```

Le fichier `common`, chargé par défaut par Capistrano, permet d'initialiser la configuration :

```
set :application, "monsite"
set :stages, %w(dev staging prod1 prod2)
set :repository_root, "http://subversion.monsite.com/monsite/"

# configure your shared directory (elements that aren't static, like
# downloaded elements)
set :bk_shared_directories, %w(public/img/sites)
set :bk_writing_directories, %w(var/cache var/log var/tmp var/smarty/
templates_c)

set :supportlist, "alexandre@monsite.com,support@monsite.com"

set :deploy_to, "/usr/local/web/htdocs/#{application}"

set :liquibase_path, ""
set :liquibase_options, ""
```

³des "recettes" dans le jargon Capistrano

Prenons l'exemple de la configuration d'un environnement qui sera automatiquement choisi via la ligne de commande :

Exemple 2. Exemple de configuration sur l'environnement de production "prod1" contenu dans le fichier `config/deploy/prod1`

```
set :url, "m.monsite.fr"  
role :web, "monserveur"  
set :user, "webadmin"
```

- url : nom de mon site
- web : serveur sur lequel le site va être déployé
- user : nom utilisé par SSH pour se connecter sur le serveur

4.3. Mise en place avant déploiement

La recette permettant de mettre en place cette architecture est "deploy:setup".

Exemple :

```
alexandre@bucarest:/var/www/monsite/current$ cap staging deploy:setup
```

va mettre en place le site sur l'environnement de recette

Une étape importante dans l'installation d'un nouveau site sur un environnement est la configuration Apache de ce site. Une nouvelle recette est ajoutée dans le fichier `deploy.rb` juste après l'étape "deploy:setup" :

```
after "deploy:setup", "backelite:create_vhost"
```

La création de ces règles Apache est dictée par l'utilisation d'un "template" de "vhost" Apache :

```
<VirtualHost *:80>  
  DocumentRoot <%= deploy_to %>/current/html  
  ServerName <%= url %>  
  ErrorLog /usr/local/web/logs/<%= application %>/error_log  
  CustomLog /usr/local/web/logs/<%= application %>/access_log  
  combined  
</VirtualHost>
```

Ce "template" utilise les informations de configuration dans la recette "backelite:create_vhost" :

```
desc <<-EOD  
  Create vhost and send an email with all the actions to apply  
EOD  
task :create_vhost do  
  file = File.join(File.dirname(__FILE__), "templates",  
    "vhost.template")  
  template = File.read(file)  
  buffer = ERB.new(template).result(binding)  
  put buffer, "#{deploy_to}/#{application}"  
  
  run "echo #{try_sudo} cp #{deploy_to}/#{application} #{webconfig_to}/  
sites-available/#{application} >> #{deploy_to}/#{log_file}"  
  run "echo #{try_sudo} ln -sf #{webconfig_to}/sites-available/  
#{application} #{webconfig_to}/sites-enabled/ >> #{deploy_to}/  
#{log_file}"  
  run "echo #{try_sudo} mkdir /usr/local/web/logs/#{application} >>  
#{deploy_to}/#{log_file}"  
  run "echo #{try_sudo} chown webadmin.web /usr/local/web/logs/  
#{application} >> #{deploy_to}/#{log_file}"
```

```
run "echo #{try_sudo} /usr/sbin/apache2ctl graceful >> #{deploy_to}/  
#{log_file}"  
run "echo #{try_sudo} /usr/sbin/apache2ctl status >> #{deploy_to}/  
#{log_file}"  
run "echo 'Do not forget to create the database before the first  
deploy.' >> #{deploy_to}/#{log_file}"  
run "echo ' ' >> #{deploy_to}/#{log_file}"  
run "echo 'Please do or check the printed actions.' >> #{deploy_to}/  
#{log_file}"  
end
```

Une fois exécutée, cette recette génère un fichier texte au nom du projet - le fameux monsite dans l'arborescence - contenant la définition des règles Apache :

```
<VirtualHost *:80>  
DocumentRoot /usr/local/htdocs/monsite/current/html  
ServerName m.monsite.fr  
ErrorLog /usr/local/web/logs/monsite/error.log  
CustomLog /usr/local/web/logs/monsite/access.log combined  
</VirtualHost>
```

Ce fichier sert ensuite lors de la création du site sur le serveur Web. Cette action reste très sensible (droits root, configuration réseau, redémarrage du serveur web...), une manipulation manuelle est donc préférée. Cependant nous allons aider notre cher administrateur système en lui indiquant la configuration Apache à installer pour ce site.

Enfin, un email est envoyé automatiquement à la fin du traitement. Il contient la liste des différentes actions et vérifications manuelles à faire par l'administrateur, dont la création de la base de données sur le serveur.

Désormais le site répond. Il faut désormais lui assigner les sources fichiers et données.

4.4. Déploiement des fichiers du site

La prochaine recette à utiliser est "deploy", qui va déployer les sources et mises à jour sur la base de données.

Exemple :

```
alexandre@bucarest:/var/www/monsite/current$ cap staging deploy
```

va déployer le site sur l'environnement de recette

La recette "deploy" est celle de Capistrano par défaut.

Ensuite, Capistrano nous demande quelle version (branche) nous voulons déployer sur cet environnement :

```
Tag to deploy (or type 'trunk' to deploy from trunk):
```

Il faut indiquer "trunk" ou tout autre répertoire copié du répertoire "tags".

Ainsi Capistrano va récupérer les sources du projet et les placer dans le répertoire "releases". Il s'occupe ensuite de créer les liens symboliques entre les répertoires partagés dans "shared".

La dernière étape effectuée par Capistrano consiste à remplacer le lien symbolique "current" qui pointe alors sur la dernière version présente dans "releases".

4.5. Déploiement des fichiers externes de modifications de base de données

Toujours dans l'étape "deploy", après avoir récupéré toutes les sources du site avec la version spécifiée, Capistrano va intégrer la mise à jour de la base de données dans le déploiement.

```
desc <<-EOD
after "deploy", "backelite:update_db"
```

Cette étape consiste à lancer l'outil "liquibase" avec comme arguments le fichier principal décrivant les changements à opérer config/model/master.xml ainsi que le fichier de configuration de l'environnement config/model/liquibase-staging.properties

```
desc <<-EOD
  Update DB
EOD
task :update_db do
  run "echo 'try to update_db.' >> #{deploy_to}/#{log_file}"
  set :liquibase_bin, "cd #{current_path} && #{liquibase_path}liquibase
#{liquibase_options} --logLevel=warning --defaultsFile=config/model/
liquibase-#{stage}.properties --changeLogFile=config/model/master.xml"
  on_rollback {
    run "#{liquibase_bin} rollback"
  }
  run "#{liquibase_bin} update"
  run "echo 'update_db done.' >> #{deploy_to}/#{log_file}"
end
```

4.6. Déploiement des diverses autres actions du déploiement

Après chaque déploiement, nous souhaitons recevoir un mail nous indiquant l'état du déploiement et mettre ainsi au courant toute l'équipe projet.

```
desc <<-EOD
after "deploy", "backelite:send_log"
desc "Send log"
task :send_log do
  run "cat #{deploy_to}/#{log_file} | mail #{supportlist} -s
'#[application] BkCapistrano action done'"
end
```

Il peut exister d'autres tâches spécifiques à réaliser après le déploiement d'un projet, notamment la mise à jour de la "crontab".

Note

Pour faciliter le travail de l'administrateur système, il est conseillé de renseigner chaque ligne de la "crontab" dans un fichier texte versionné que le développeur doit mettre à jour. L'administrateur n'a alors plus qu'à copier/coller les lignes.

5. Conclusion

5.1. Ce qu'il reste à améliorer

- Automatisation de l'intégralité des tâches (ajout du vhost, redémarrage Apache, mise à jour de la crontab...)
- Retour en arrière ("rollback") validé sur chacune des actions du déploiement. En cas d'erreurs de migration de base de données (même si les erreurs sont rares) les fichiers du site ne sont pas remis dans leur état initial et cela est fait manuellement.
- Améliorer le système d'alertes de déploiement

5.2. Inconvénients

- Seulement pour les OS compatibles avec les liens symboliques
- Utilise beaucoup de technologies différentes (Ruby, Java, SSH...)

- Mise en place assez compliquée et prenant du temps

5.3. Avantages

- C'est le développeur qui réalise les mises en production, en accord avec l'administrateur système & réseau
- Déploiements rapides
- Déploiements de meilleure qualité (moins de rollbacks, moins de temps d'indisponibilité...)
- Capacité rapide de rollback
- Personnalisation facile à mettre en place