Laravel

Mise en production.

Auteur : ***Berthe Seydou***

24/08/2016

Table des matières

[Mise en production: 2](#_Toc459975500)

[1. Description du WorkFlow : 2](#_Toc459975501)

[2. Mise pratique : 4](#_Toc459975502)

Prérequis : GIT, Administration Système Linux et Connexion via SSH.

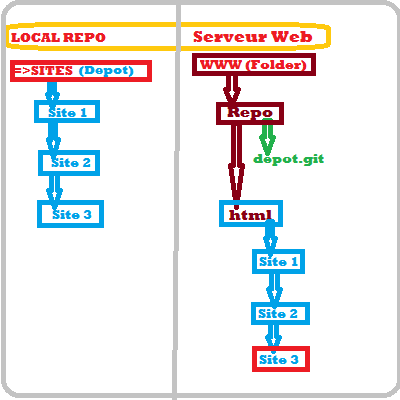
# Mise en production:

C’est bien de travailler en local sur un site mais à un moment donné il faudra bien l’héberger pour que le client puisse mettre le site en service. Pour faciliter le processus de mise en production, il faut avoir ce que l’on appelle **WorkFlow** qui permet structurer le processus de déploiement et de mise à jour. Dans ce document, nous allons traiter les points suivants :

1. Description et mise en place du **WorkFlow**.
2. Mise en pratique.

# Description du WorkFlow :

Dans notre cas, nous allons opter pour un **WorkFlow** qui sera sur basé sur l’automatisation de ce processus en utilisant GIT & **Laravel**. Avant de commencer, un bref aperçu de la structure de notre **WorkFlow** est la suivantes :



Comme présenté sur la figure ci-dessus, le **WorkFlow** sera divisé en deux parties une partie en local et une sur le serveur.

🡪La partie locale (LOCAL REPO) :

Cette partie contient les projets en cours de développement, c’est dans ce dépôts que l’on va faire le développement de notre applications.

🡪 La partie Serveur (Serveur Web) :

Cette partie contiendra essentiellement deux répertoires à partir du répertoire « /var/www » du serveur :

🡪 Le premier répertoire contiendra ce que l’on appelle un « dépôt partagé », sur la figure c’est celui nommé « Repo ». Ce répertoire est celui qui va permettre d’interagir avec le dépôt local sur notre machine et mettre à jour le site à jour.

🡪Le deuxième répertoire « html » qui est par défaut créer contiendra les sites en live.

A ce point la question qui se pose est comment ces deux (local et le serveur) sont reliés ?

En effet, lorsque l’on met notre application à jour en local et que l’on est prêt à passer en live, il suffit que l’on envoie ces modifications au niveau du dépôt distant sur le serveur « Repo » et à l’aide de ce que l’on appelle « **Hook** » on peut exécuter un script après une modification sur le répertoire « Repo » du serveur d’où l’utilité d’avoir ce répertoire comme partagé, le script en question consistera à mettre à jour le site en live.

Alors passons maintenant à la pratique, pour mettre en œuvre ce **WorkFlow** nous allons procéder comme suit :

1. Création du dépôt sur le serveur :

Nous commençons tout d’abord par se connecter au serveur par **ssh** (En ligne de commande) :

**ssh <nom utilisateur>@<IP Serveur>** (Le mot de passe sera demandé).

A la suite de la connections, on se déplace dans le répertoire « /var/www » du serveur et on crée le répertoire « Repo » (Ce répertoire contient déjà **html**).

**cd /var/www**

**mkdir Repo**

Nous allons maintenant créer le dépôt GIT partagé « **depot.git** » avec les commandes suivantes :

**cd Repo**

**mkdir depot.git**

**cd depot.git**

**git init --bare**

(Cette dernière commande initialise le dépôt distant partagé, remarqué surtout le répertoire **hooks**).

Dans le répertoire  « **hooks** » de « **depot.git** » on va éditer le contenu du fichier « post-update » par :

**#!/bin/sh**

**git --work-tree=/var/www/ --git-dir=/var/www/Repo/depot.git checkout –f**

L’option **« –work-tree & --git-dir**» représente respectivement le répertoire qui contenant les sites en live et celui dépôts partagé.

A ce point nous avons fini avec le coté serveur. Nous allons maintenant passer à la configuration côté local.

1. Configuration cote local :

Si vous êtes toujours connecté en au serveur déconnectez-vous avec la commande « exit » et passé dans le dépôt local contenant les sites à déployer.

Une fois dans ce répertoire, nous allons ajouter notre dépôt distant situé sur le serveur pour pouvoir y faire des mises à jour, on exécute la commande suivante :

**git remote add production ssh://<nom utilisateur>@<IP domaine>/var/www/Repo/depot.git**

Après l’exécution de cette commande, on aura notre dépôt distant sur le serveur parmi les dépôts distant local qui pourra être indexé par « production » comme spécifier dans la commande, vérifier avec commande : **git remote –v.**

1. Déploiement d’un projet :

Finalement pour déployer un projet, il suffit de faire un commit du projet en question et d’exécuter la commande suivante :

**git push production master**

Cette commande permet non seulement d’ajouter le projet en question en live s’il n’existe pas déjà mais elle met aussi le projet à jour avec les modifications apportées si le projet existe déjà sur le serveur.

NB : Ce **WorkFlow** doit être mise en place avant d’entamer la mise en pratique (La partie suivante).

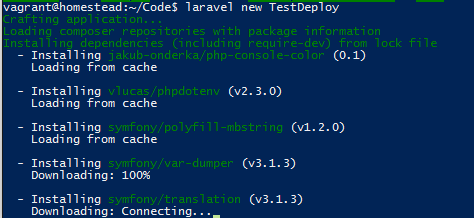
# Mise pratique :

A présent, passons à une partie plus pratique pour mettre en œuvre la création d’un projet jusqu’au déploiement du projet. Pour ce faire nous allons supposer que nous sommes une équipe composée de deux développeurs et un administrateur qui va se charger de déployer la dernière version de l’application en live.

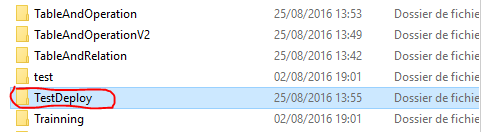
Nous allons suivre les étapes suivantes pour arriver à terme de cette mise en pratique.

1. Créer un projet et Committer sur Git.
2. Déploiement du projet sur le serveur Web.
3. Récupération du projet.
4. Modification et commit sur Git.
5. Redéploiement du projet en live.
6. Création du projet et Commit sur Git :

Créer un projet **Laravel** nommée **TestDeploy**, Dans mon cas ça donne l’illustration suivante :



Après la création du projet, mon répertoire de travail est la suivante :



Ce répertoire est aussi un dépôt Git, ainsi après des modifications sur un projet, je peux faire le commit facilement sur Serveur Web ou sur Git.

On va maintenant faire le Commit de ce projet sur Git avant de faire le déploiement.

On Cree un dépôt Git « Deploy », dans mon cas on a l’URL de ce dépôt est : <https://github.com/ArianeExtern/Deploy.git>, on va faire le commit comme suit :

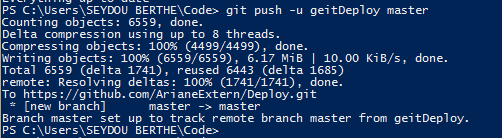
Ajout du dépôt git :



Ensuite on fait un « **git add TestDeploy** » pour que le projet soit traquer par git.

Après on fait un « **git commit –m '’Commit du projet TestDeploy’’** » pour faire le commit au dépôt local.

Finalement, on fait un « **git push –u geitDeploy master** », ça donne l’illustration suivante :



**NB : Tous l’intérêt de cette façon de faire est que les développeurs peuvent travailler et faire autant de commit que possible sur le projet et dès lorsque le projet est prêt à être déployer en live, ceci se ferait avec un commit sur le serveur Web.**

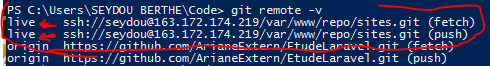
1. Déploiement du projet :

Comme expliquer dans la première partie de ce document, on va ajouter le dépôt distant situé sur le serveur Web dans la liste des dépôts distant de notre dépôt local avec la commande suivante :

git remote add live ssh://<Nom utilisateur>@<Adresse IP>/var/www/repo/depot.git

(Le mot de passe de l’utilisateur SSH vous sera demandé).

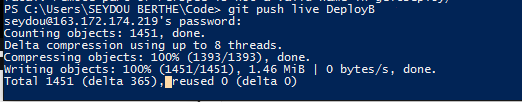
Si vous ne l’aviez pas encore fait dans le première partie de ce document, après l’exécution de cette commande, on a l’illustration suivante à la suite de l’exécution de la commande « **git remote –v** » pour lister les dépôts distants :



Comme on peut le voir sur l’image, il y’aura le dépôt distant situé sur le serveur Web qui sera ajouter.

Finalement pour déployer, il suffit de lancer la commande « **git push live master** » (Si notre projet est sur la branche **master** sinon il faudra spécifier la branche en question à la place de **master**) ; Cette commande va déployer notre projet si le projet ne l’est pas déjà sinon il va mettre à jour le dépôt distant.

Dans mon cas, ça donne l’illustration suivante :



Après l’exécution de cette commande, notre projet sera déployé sur le serveur distant.

Pour finir le déploiement, on va utiliser les outils en ligne de commande « Composer et php artisan » pour mettre à jour le projet de ces nouveaux fichiers.

On se connecte tous d’abord au serveur Web via SSH **« ssh <Nom Utilisateur>@<IP Adresse>** (Le mot de passe vous sera demandé)».

Ensuite on se déplace dans le répertoire « **/var/www/html**» :



A ce niveau, on doit s’assurer que le répertoire du projet appartient à l’utilisateur « www-data » sinon on fait : « **chown –R www-data <Projet>».** (Dans le cas échéant les utilisateurs ne pourront pas accéder au site).

Ensuite lançons la commande « **composer install** » pour installer les dépendances. (Intéressant après une modification).

Après on régénère les fichiers auto-générer avec la commande « **composer dump-autoload** ».

Finalement, Si c’est la première que l’on déploie l’application, on crée la base de données et on lance la commande « **php artisan migrate** (Cette commande sera lancé aussi après une modification sur les tables)».

NB : Les fichiers placés dans le répertoire « **public** (**index.php, .htaccess**)» du projet doit être déplacé à la racine du projet et faire modification suivantes dans « index.php » :

🡪 **require \_\_DIR\_\_.'/../bootstrap/autoload.php';** => require \_\_DIR\_\_.'/bootstrap/autoload.php';

🡪 **$app = require\_once \_\_DIR\_\_.'/../bootstrap/app.php';** =>

$app = require\_once \_\_DIR\_\_.'/bootstrap/app.php';

Dans le fichier « .htaccess » doit aussi avoir la ligne suivante : « RewriteBase /<Racine Projet>  dans notre cas RewriteBase /TestDeploy» juste en dessous de la ligne « RewriteEngine On ».

1. Récupération du projet :

Comme expliquer ci-haut la dernière version du projet (live) se trouve sur GitHub (La branche **Master**),

Donc si un développeur doit faire une nouvelle fonctionnalité, il procède par :

🡪Créer une nouvelle branche contenant la dernière version :

**git checkout –b <ma nouvelle fonction> master**

Cette commande récupère la dernière version du code depuis le dépôt central (Pas du Serveur Web, on fait seulement des commit sur le dépôt du serveur quand l’application est prête à passer en live) et crée une branche (Si elle n’existe pas) contenant la dernière version.

🡪Si cette branche existe déjà, tous développeurs voulant travailler dessus doit commencer par récupérer la dernière version (Comme ci-dessus) avant d’effectuer toute modification.

1. Modification et Commit sur Git :

Après la récupération de la dernière version sur une nouvelle branche à part, le développeur peut effectuer ces modifications sur cette branche sans impacter la version sur la branche **master** (Branche Principale) en procédant comme suit :

🡪 Après une modification,

🡪Il fait un « **git status** » pour voir les fichiers modifiés,

🡪Ensuite un « **git add <fichier modifié>** » pour ajouter les fichiers parmi ceux traqués par Git.

🡪Et un « **git commit** » pour préparer les fichiers traquer à être déployer sur le serveur.

🡪Finalement s’il finit les modifications et qu’il a déjà fait le commit des fichiers modifiés, Il suffit qu’il fasse un « **git push –u origin <ma nouvelle fonction>** » pour envoyer les modifications sur le dépôt central (GitHub). (L’option –u spécifie que c’est une mise à jour de la branche en question).

Par la suite le développeur doit faire ce que l’on appelle un « **Pull Request** (Voir image ci-dessous)» pour notifier les autres développeurs qu’il a fini la nouvelle fonction, ceci permettra aux autres développeurs d’évaluer le travail, éventuellement faire une discussion autour de cette branche et même suivre les nouvelles modifications apporté à cette branche au cas où les développeurs ne valide pas le travail. Si le travail est validé on va donc ajouter cette branche « **ma nouvelle fonction** » avec la branche principale « **master** » comme suit (Par l’administrateur):

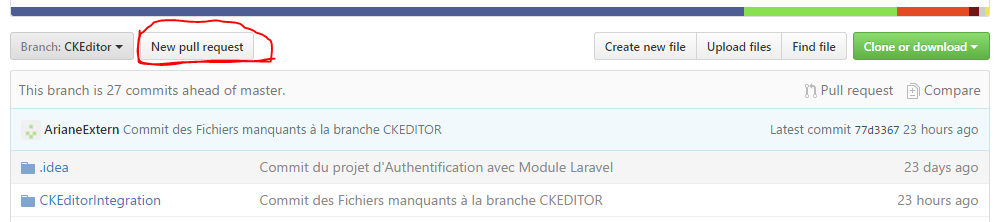
🡪**git checkout master** (Se déplacer sur la branche principale).

🡪**git pull** (Mettre à jour la branche principale).

🡪**git pull origin <ma nouvelle fonction>** (Mettre à jour la branche principale par « ma nouvelle fonction » en local).

🡪Finalement **git push** (Pour mettre à jour la branche principale distante).

Tous ceci sans surtout touché à la version en live d’où l’intérêt de ce Workflow, nous allons maintenant voir comment faire un redéploiement.



1. Redéploiement en live :

Finalement après avoir fait la fusion de la nouvelle fonction avec la branche principale, l’administrateur peut décider de mettre le projet en live.

Dans ce cas, il récupère la dernière version (Branche principale ou Master) du dépôt central avec la commande : « **git checkout origin master** » (Noter que **origin** est l’alias de notre dépôts central).

Finalement il fait un « **git push live master** » (Noter que live est l’alias de notre dépôt sur le serveur Web ssh://…) et on met à jour le projet sur le serveur Web (Voir 2).