Cours Laravel 5.3 – plus loin – le déploiement

Une application se développe et se teste en local mais arrive un moment où il faut la mettre sur un serveur pour qu'elle devienne visible et accessible. Ce déploiement n'est pas forcément une tâche aisée selon le contexte. Dans ce chapitre nous allons faire un petit tour d'horizon de ce qu'il convient de faire sans pouvoir être exhaustif étant donné la multiplicité des configurations existantes.

L'environnement

Créer des environnements

Pour la gestion de l'environnement Laravel fait usage d'un package tiers : DotEnv. J'en ai déjà parlé dans ce cours mais on va un peu faire le point. Quand vous installez Laravel avec Composer vous trouvez à la racine le fichier **.env** avec ce contenu :

```
APP_ENV=local
APP_KEY=
APP_DEBUG=true
APP_LOG_LEVEL=debug
APP_URL=<a href="http://localhost">http://localhost</a>

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=homestead
DB_USERNAME=homestead
DB_PASSWORD=secret

BROADCAST_DRIVER=log
CACHE_DRIVER=file
SESSION_DRIVER=file
QUEUE_DRIVER=sync
```

```
REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379

MAIL_DRIVER=smtp
MAIL_HOST=mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=null
MAIL_PASSWORD=null
MAIL_ENCRYPTION=null

PUSHER_APP_ID=
PUSHER_KEY=
PUSHER_SECRET=
```

Ça se présente sous la forme clé/valeur. Par exemple on a déjà vu cela en oeuvre dans **config/app.php** :

```
'debug' => env('APP_DEBUG', false),
```

L'helper **env** permet d'aller lire la valeur de **APP_DEBUG** dans le fichier **.env** et d'affecter la valeur de la clé **debug** de la configuration de l'application.

On trouve aussi par exemple le réglage de MySQL dans **config/database.php** :

```
'host'      => env('DB_HOST', 'localhost'),
'port'      => env('DB_PORT', '3306'),
'database'  => env('DB_DATABASE', 'forge'),
'username'  => env('DB_USERNAME', 'forge'),
'password'  => env('DB_PASSWORD', ''),
```

Remarquez que l'helper accepte une valeur par défaut comme deuxième paramètre.

Que fait cet helper ?

Tout simplement il stocke les valeurs dans la super globale d'environnement **$_ENV** du serveur.

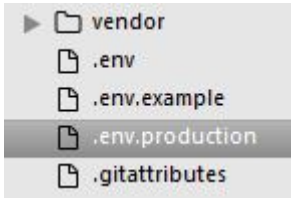
*Moralité : pour avoir plusieurs environnements il faut créer plusieurs fichiers **.env**.*

Le cas le plus classique sera d'avoir un environnement local de développement et un autre de déploiement. Il faut juste ne pas se mélanger les pinceaux, en particulier ne pas enlever le fichier **.gitignore** de la racine qui prévoit de ne pas envoyer le fichier **.env** qui peut contenir des données sensibles (si vous utilisez **git**).

La variable **APP_ENV** dans le fichier **.env** est justement destinée à connaître l'environnement actuel :

```
APP_ENV=local
```

Lorsque je crée une application je fais une copie du fichier d'environnement pour régler mes variables en situation production. Pour ne pas qu'il y ait de confusion je le nomme de façon explicite :



Dans ce fichier je fixe toutes les valeurs nécessaires :

```
APP_ENV=production
APP_DEBUG=false
APP_KEY=5avz0M3IGPu4GFxBBLPhQs00MNJREzoL
...
```

Il me suffit ensuite d'envoyer ce fichier là sur le serveur et de le renommer.

*Selon où vous hébergez votre application vous pouvez avoir la possibilité de définir les variables d'environnement sans avoir à utiliser le fichier **.env**. C'est le cas si vous utilisez par exemple **Forge**.*

Le déploiement

Le serveur

Laravel n’a pas besoin de grand chose mais quand même :

- PHP >= 5.6.4
- Extension PHP PDO
- Extension PHP OpenSSL
- Extension PHP Mbstring
- Extension PHP Tokenizer

D’autres part les dossiers **storage** et **bootstrap/cache** doivent avoir les droits d’écriture sur le serveur.

Selon la méthode d’installation que vous allez employer vérifiez que vous avez bien une clé de cryptage dans votre environnement.

L’envoi des fichiers de l’application

Il existe de nombreux outils pour envoyer ses fichiers sur le serveur de production mais c’est un sujet assez vaste et pas forcément facile. Dans le cadre de ce cours je me contenterai de parler de l’envoi avec **FTP** qui est la méthode la plus simple et classique.

Mais avant d’envoyer les fichiers il faut être sûr de ne pas avoir des choses inutiles.

La première chose à faire est de vider le cache qui peut être volumineux :

```
php artisan cache:clear
```

La seconde chose est d’envoyer le bon fichier de configuration dont je vous ai parlé ci-dessus.

Et le dossier vendor ?

Là ça dépend si vous disposez de **SSH** sur le serveur, ce qui vous permet d’utiliser **composer** (même s’il n’est pas installé globalement vous pouvez toujours envoyer le fichier **phar**) et ainsi d’éviter l’envoi du dossier **vendor** parce que vous pourrez exécuter

un **composer install**.

Si vous ne disposez pas de **SSH** sur votre serveur vous pouvez soit changer d'hébergeur, soit envoyer le dossier **vendor** et tout son contenu.

L'installation

Une fois que les fichiers de l'application sont sur le serveur si vous avez **SSH** (et donc vous n'avez pas envoyé le dossier **vendor**) vous devez procéder à l'installation.

Soit vous avez **composer** installé globalement sur le serveur

```
composer install
```

Soit vous avez envoyé le fichier phar :

```
php composer.phar install
```

Et là le dossier vendor va se créer et se remplir.

Le cache

Pour gagner en performance vous pouvez mettre les routes en cache pour en disposer plus rapidement :

```
route:cache           Create a route cache file for faster route registration
```

Attention ! si vous avez les routes en cache et que vous modifiez le fichier de routes il faudra régénérer le cache !
De la même manière vous pouvez mettre en cache les fichiers de configuration :

```
config:cache          Create a cache file for faster configuration loading
```

La base de données

Il vous faut également créer la base de données sur le serveur et bien renseigner les paramètres dans votre fichier .env de production. Ensuite si vous dispose de **SSH** lancez les migrations

et la population (si nécessaire) sur le serveur :

```
php artisan migrate --seed
```

Si vous ne disposez pas de **SSH** l'affaire se complique un peu. Vous devez exporter votre base sur le serveur MySQL local et avec le fichier SQL généré créer votre base sur le serveur MySQL distant.

Avec tout ça votre application devrait fonctionner !

Les mises à jour ultérieures

Mise à jour du framework

Pour effectuer les mises à jours ultérieures du framework il faudra utiliser :

```
composer update
```

*Et évidemment si vous ne disposez pas de **SSH** vous devez faire la mise à jour en local et transférer ensuite les fichiers ajoutés et modifiés du dossier vendor.*

Mise à jour de l'application

Quand vous mettez à jour votre application il peut y avoir des moments où il vaut mieux que personne ne se connecte. Laravel propose un mode maintenance facile à mettre en oeuvre avec Artisan :

```
php artisan down
```

Et bien sûr il y a la commande inverse :

```
php artisan up
```

Évidemment encore faut-il que vous ayez accès à Artisan sur votre serveur…

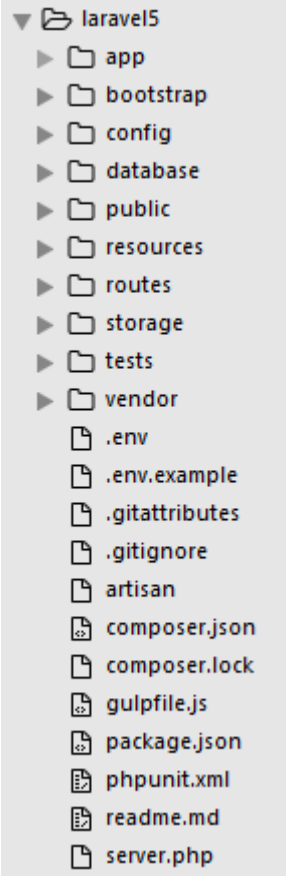
*En mode maintenance la vue affichée est celle située ici : **resources/views/errors/503.blade.php***

Installation sur serveur mutualisé

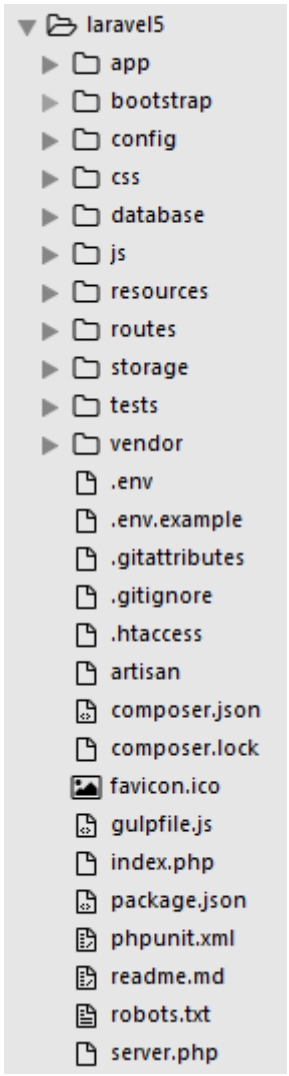
Sur un serveur mutualisé peut se présenter une difficulté. On a vu que le seul dossier de Laravel qui doit être accessible est **public**, les autres ne doivent pas l'être. En général on fait pointer son domaine (ou sous-domaine) sur le dossier **public** et le tour est joué.

Mais parfois on ne dispose que d'un dossier genre **public_html** et on doit tout mettre dedans. Laravel n'a pas été conçu pour ça mais on peut quand même le faire fonctionner ainsi. Voici la procédure :

Commencez par installer normalement Laravel. Vous devez avoir cette architecture :



On va commencer par transférer tout ce qui se trouve dans le dossier **public** à la racine et supprimer ce dossier. Vous devez avoir maintenant cette situation :



Evidemment maintenant plus rien ne fonctionne !

Ouvrez le fichier index.php et trouvez cette ligne :

```
require __DIR__.'/../bootstrap/autoload.php';
```

Remplacez-la par celle-ci :

```
require __DIR__.'/bootstrap/autoload.php';
```

Dans le même fichier trouvez cette ligne :

```
$app = require_once __DIR__.'/../bootstrap/app.php';
```

Remplacez-la par celle-ci :

```
$app = require_once __DIR__.'/bootstrap/app.php';
```

Ces modifications sont nécessaires à cause du déplacement des fichiers et de la modification des emplacements relatifs.

Maintenant la page d'accueil de Laravel va s'afficher

correctement.

Le souci c'est que maintenant tout devient accessible et il faut prendre des précautions !

Par exemple en prévoyant dans les dossiers que vous voulez inaccessibles (app, bootstrap…) un **.htaccess** avec :

Deny from all

Les seuls fichiers accessibles restent ceux qui sont présents à la racine et il y a encore des choses sensibles comme votre fichier de configuration. il faut donc aussi prévoir quelque chose au niveau du **.htaccess** à ce niveau !

Créer une installation comme WordPress

Si vous avez besoin d'une installation assistée genre celle proposée par WordPress j'ai créé un package qui peut vous aider.

Pour voir comment ça se présente comencez par créer une installation fraîche de Laravel.

Ajoutez l'authentification avec Artisan :

php artisan make:auth

Vérifiez que tout fonctionne correctement.

Créez aussi une base de donnée MySQL sur le serveur et notez son nom et les paramètres pour y accéder.

Ensuite installez le package :

composer require bestmomo/laravel5-3-installer

Attendez que l'installation soit terminée.

Ajoutez la référence du package dans **config/app.php** :

Bestmomo\Installer\InstallerServiceProvider::class,

Pour terminez publiez la configuration, les langues, les vues et les routes :

php artisan vendor:publish --tag=laravel-installer

C'est pratiquement terminé !

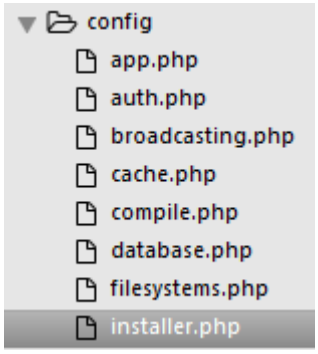
Voyons comment ça fonctionne…

Etape 1 : vérification du serveur

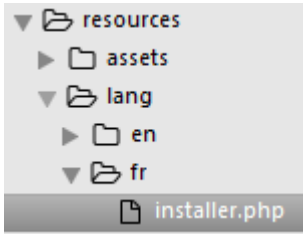
Votre page d'accueil a changé pour celle-ci :

L'aspect ici est avec la locale en français, il y a aussi les textes en anglais.

Si vous arrivez sur cette page c'est que le serveur est correct pour Laravel (version et extensions de PHP, permissions sur les dossiers). Pour changer des éléments à ce niveau ça se passe dans le fichier de configuration :



Si vous voulez modifier les textes, en particulier le titre et le numéro de version, vous pouvez le faire dans le fichier des langues, par exemple pour le français :



Etape 2 : la base de données

Dans l'étape suivante on demande les informations pour la connexion à la base :

Modifiez les informations en conséquence. S’il y a une erreur vous recevez ce message :

Si tout se passe bien les migrations et les populations

éventuelles sont lancées.

Etape 3 (optionnelle) : création d'un administrateur

Cette étape a lieu par défaut, vous pouvez changer ce comportement dans le fichier **config/installer.php** :

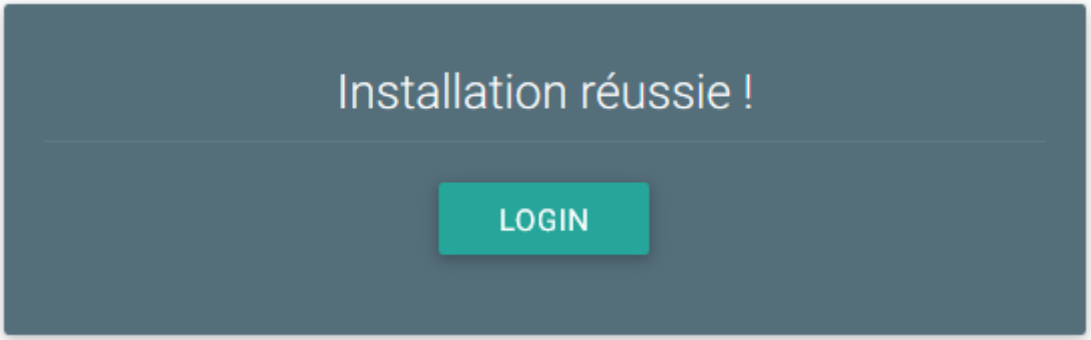
```
'administrator' => false,
```

Voici le formulaire qui apparaît :

Les éléments du formulaire sont aussi paramétrables dans le fichier de configuration.

Si tout se passe bien l'administrateur est créé et une nouvelle clé de cryptage est enregistrée dans le fichier **.env**.

Vous devez voir apparaître ce message :



A la fin de l'installation la référence au fournisseur de service de l'installeur est supprimée dans le fichier config/app.php, ce qui supprime les accès à cet installeur.

Vous avez ainsi tout ce qu'il faut pour créer une belle application et la distribuer :

- créez l'application complète avec le vendor,
- ajoutez le package,
- ajustez les paramètres dans la configuration du package

En résumé

- Il faut créer des environnements pour répondre aux différentes situations : local, distant…
- Le déploiement est facile et rapide si on dispose du **SSH**.
- On peut supprimer le dossier **public** tout en ayant un Laravel fonctionnel, il faut juste prendre des mesures pour assurer la sécurité.
- Le package **bestmomo/laravel5-3-installer** permet de créer une installation dans le style de WordPress.