Anto59290

Zeste de savoir

Déployer une application Django en production

9 mars 2020

Table des matières

Dans ce tuto nous allons voir comment déployer une application Django en production. En effet de nombreux articles couvrent la création d'une application Django et le développement à l'aide du serveur *runserver* de Django, mais il est plus difficile de trouver des informations sur les bonnes pratique de déploiement. Ce tuto couvre les paramètres d'une application pour la production (fichiers de *settings*), l'installation et la configuration de Gunicorn et l'installation et la configuration de Nginx. A la fin de ce tutoriel nous aurons une application Django en production prête pour un usage professionnel. Une seule approche technique sera présenté ici mais il existe plein d'autres logiciels et infrastructures permettant d'arriver au même résultat.

Prérequis : un minimum de connaissance de Django est requis pour une bonne compréhension. C'est encore mieux si vous avez une application et un serveur (une VM avec debian fait très bien l'affaire) sous la main. Un minium d'aisance avec la ligne de commande est préférable.

1. Paramétrons Django

1.1. Configurer vos *settings* Django

Dans un premier temps il est nécessaire de faire un tour du côté des *settings* de Django. En effet ceux ci permettent de paramétrer notre application pour qu'elle soit prête pour l'environnement de production.

1.1.1. Désactiver le mode Debug

Le paramètre le plus important est la valeur `DEBUG`.

Il est **impératif** que cette valeur soit fausse en production. En effet comme son nom l'indique ce paramètre permet de debugger l'application est convient a un environnement de développement,

mais **surtout pas à la production**. Quand ce paramètre est vrai de nombreuses informations sont affichées. Il faut s'assurer que ces informations ne seront jamais visibles par vos visiteurs.

Par exemple en cas d'erreur 500 vos visiteurs tomberont sur une page d'erreur (personnalisable) plutôt que sur la page d'erreur utilisée lors du debug présentant des données sur le comportement de votre application.

En désactivant le mode debug il devient impératif de remplir la valeur de la variable `ALLOWED_HOSTS`. Par défaut en mode debug, les connexions depuis le localhost sont automatiquement acceptées. Pour la production il faut configurer cette liste avec l'ensemble des noms d'hôtes auxquels le site doit répondre. Si cela n'est pas fait l'ensemble du site sera inaccessible et vous arriverez sur une page d'erreur.

FIGURE 1.1. – Erreur (en mode debug) précisant que l'application n'est pas configurée pour fonctionner sous ce nom d'hôte.

1.1.2. Paramétrer l'accès à base de données

Par défaut dans un nouveau projet, une base SQLite est utilisée. Cela est très pratique pour développer car c'est un moteur de base de données très simple à mettre en place. En revanche ce moteur n'est généralement pas utilisé en production pour des grosses applications web, on lui préfère en général des solutions comme MySQL, MariaDB, PostgreSQL, etc. Pour cela il est nécessaire de modifier la configuration des bases de données. Par défaut la configuration est la suivante :

```
```

On pourra modifier la configurer comme suit, par exemple pour PostgreSQL :

```
```

On voit ici qu'on précise le moteur de base de données utilisé ainsi que les informations de connexion. Il est important que garder les informations sensibles (utilisateur et mot de passe) aussi privées que possible. Ainsi **on évite en général de les ajouter en clair au fichier de réglages de l'application** et on se tourne vers l'utilisation de secrets (en savoir plus). Pour

en savoir plus sur la gestion des paramètres de base de données dans Django n'hésitez pas à consulter la documentation .

1.1.3. Désactiver la Django Debug Toolbar

Si vous utilisez le module Django Debug Toolbar, c'est le moment de vérifier que celui-ci est désactivé lorsque `Debug` est faux. C'est normalement le comportement par défaut... sauf si vous en avez décidé autrement. Django Debug Toolbar donne de nombreuses informations sur le fonctionnement de votre application et doit donc être désactivé dans tous vos environnements excepté votre machine de développement. Par exemple si on regarde le fichier de *settings* de Zeste de Savoir on trouvera le code suivant :

```
```

Ouf ! L'honneur est sauf, la debug toolbar n'est activé qu'en mode debug.

1.2. Gérer ses dépendances Python

En python on maintient généralement une liste de ces dépendances dans un fichier requirements.txt. Dans l'idéal il faut séparer les dépendances de manière plus fine, par exemple en utilisant trois fichiers différents :

— Un fichier requirements.txt qui contient toutes les dépendances de base de votre application.
— Un fichier pour le développement (requirements-dev.txt) qui contient tous les outils de développement et de débug tels que Django Debug Toolbar
— Un fichier requirements-prod.txt pour les dépendances propres à la pre-production et à la production

Dans le fichier de dépendances de dev ou de prod il est possible d'utiliser la syntaxe suivante pour importer les dépendances de base

```
```

Avec cette méthode on est sûr de ne pas avoir d'outils de débug installées sur la prod. Par exemple, toujours chez Zeste de savoir on trouve un fichier requirements-dev.txt contenant les dépendances qui ne doivent pas être installées en production :

```
```

Ce sont par exemple les dépendances permettant le debug, le *linting*, la documentation, les tests, etc.

1.3. Autres réglages

Avant de passer en production vérifiez bien que tous vos *settings* sont bons, voici quelques exemples de paramètres à vérifier :

— Paramètres d'authentification à la base de données : ceux-ci sont probablement différents entre votre environnement de dev et la production, vérifiez que les logins et les mots de passes soient corrects.
— Tokens d'API, comptes de tests : Si vous utilisez des services externes tel que Mandrill pour l'envoi de mail ou PayPal pour le paiement pensez à vérifier que vous utilisez bien vos comptes de production et non pas les comptes de test. Si vous oubliez cette étape les services externes ne se comporteront pas correctement : faux paiement dans la sandbox de PayPal, mail qui ne seront jamais envoyés avec mandrill, etc.

Si vous voulez être sûr de bien avoir fait le tour de tous les paramètres, il existe une liste des choses à vérifier avant de déployer dans la documentation de Django. On peut par exemple y trouver la commande suivante permettant de vérifier un certains nombres de points avant le déploiement :

```
```

Comme nous l'avons vu, il existe de nombreuses différences entre les *settings* de production et ceux de développement. Comme pour les fichiers *requirements* il peut être utile de maintenir deux fichiers, un pour votre machine locale et une pour votre serveur. On peut par exemple créer un module python contenant les *settings* adaptés aux différentes configurations.

```
```

2. Configurer Gunicorn et Supervisord

Nous allons maintenant passer à la partie «serveur» de ce cours. Dans un environnement de développement il suffit de lancer la commande *runserver* de Django pour que notre application soit accessible dans le navigateur. C'est un outil très simple et pratique qui permet de développer et de débugger très rapidement une application. En revanche ce serveur applicatif n'est pas du tout fait pour être utilisé en production : il est assez lent et ne tient pas du tout la charge. Il faut donc passer par une autre programme pour servir notre application.

2.1. Introduction

Pour faire tourner notre application en production nous allons nous reposer sur trois logiciels :

— Un serveur applicatif : il s'agit de Gunicorn dans ce tuto, c'est l'un des serveurs les plus connus dans le monde de Python. Son rôle est d'exécuter le code de notre application et de renvoyer les réponses aux requêtes qui lui sont faites.

— Supervisord : il sera en charge de lancer le serveur Gunicorn est d'assurer son bon fonctionnement.
— Un reverse proxy : nous utiliserons Nginx, il s'agit d'un des reverse proxy le plus connu, il est renommé pour sa fiabilité et sa rapidité. Son rôle sera de transmettre les requêtes venant des utilisateurs extérieurs à Gunicorn.

On considère dans la suite de ce tuto que le moteur de base de données que vous voulez utiliser est bien installé et configuré avec les bons paramètres afin de laisser l'application Django se connecter. Si ce n'est pas le cas vous obtiendrez une erreur applicative quand vous souhaiterez démarrer ou utiliser votre application dans la suite de ce chapitre.

Nous devons tout d'abord créer le virtualenv qui nous servira tout au long de ce tutoriel. Pour cela utilisez votre gestionnaire de paquet pour installer le virtualenv correspondant à la version de Python de votre projet.

```
```

Puis créons le répertoire et le virtualenv

```
```

Nous pouvons en profiter pour installer les dépendances et réaliser les migrations si nécessaires

```
```

On se retrouve à cette étape avec un projet Django fonctionnel, on pourrait lancer le *runserver* de Django pour vérifier que tout fonctionne correctement.

2.2. Gunicorn

Gunicorn sera notre serveur applicatif, si Gunicorn fait parti des dépendances de votre application Django il a été installé à l'étape précédente. Si ce n'est pas le cas, on l'installe :

```
```

Dans ce cas, pensez à ajouter la dépendance vers Gunicorn dans votre fichier requirements destiné à la production. A ce stade, on peut déjà lancer notre serveur Gunicorn pour vérifier que tout fonctionne bien :

```
```

Ici on demande juste à Gunicorn de lancer notre application sur la machine en servant le port 8001. Depuis votre serveur, dans un terminal, vous devriez pouvoir atteindre votre application

Vous pouvez aussi essayer de charger la page depuis votre navigateur, tout fonctionne normalement. C'est bien beau tout ça, par contre dès que vous allez fermer votre terminal, votre application ne sera plus disponible. Pas très pratique ! Pour corriger ce problème nous allons avoir besoin d'un autre outil : supervisord.

2.3. Supervisord

Supervisord est un outil extrêmement pratique permettant de lancer des applications et de suivre leur état. En effet il est possible que votre programme plante et quitte de manière inopinée. Une des possibilités de Supervisord est de pouvoir le relancer de manière automatique. Supervisord permet aussi en une commande de connaître le statut des différentes applications que vous avez lancées, plus besoin d'aller chercher l'état du service ou le PID du programme à la main.

Nous allons ici nous servir de Supervisord pour lancer Gunicorn et le relancer en cas de besoin. Pour cela la configuration va être assez simple. Supervisord utilise le format de fichier INI pour paramétrer les différents programmes à gérer. Commençons par installer le paquet

Les fichiers de configuration sont stockés dans le dossier /etc/supervisor/conf.d/, je vous propose de créer le fichier /etc/supervisor/conf.d/hello_world.conf avec le contenu suivant :

Listing 1 – configuration de supervisord

Cette configuration assez simple fonctionne de la manière suivante :
On commence par définir le nom de notre programme afin de pouvoir le repérer facilement. `[program:helloworld]`. Essayer de bien nommer vos programmes et vos fichiers afin de faciliter la maintenance. On définit ensuite les paramètres d'environnement nécessaire à l'exécution de notre programme : ici les fichiers de *settings* à utiliser. Si vos fichiers de *settings* sont dans un module il est possible de le préciser sous la forme `settings.nom_du_fichier`. Au besoin il est possible de passer d'autres paramètres d'environnement sous la forme d'un dictionnaire (`KEY="val",KEY2="val2"`). Le paramètre `directory` permet quant à lui de donner le répertoire ou se trouve notre application, ce qui permet par exemple d'utiliser un chemin relatif vers le fichier wsgi dans la commande Gunicorn. C'est aussi pratique quand vous utilisez beaucoup de configurations différentes pour retrouver en un clin d'œil le répertoire d'une application.
On définit ensuite la commande à exécuter pour lancer le programme. On voit ici que supervisord lance notre serveur Gunicorn avec des paramètres pour :

— fichier de *socket* : le premier paramètre *bind* permet de définir un fichier de *socket* que Gunicorn va utiliser pour communiquer avec Nginx, nous verrons ce point plus en détails dans la partie suivante.

— *bind* : le second *bind* permet de lier Gunicorn à une adresse, dans cet exemple votre application sera disponible sur le port 8001 du serveur.
— *workers* : précise le nombre de processus qui vont traiter en parallèle des requêtes. Par défaut la valeur est de 1, ce qui est assez faible. La valeur recommandée est en général de deux fois le nombre de cœurs du serveur. Cette valeur est assez généraliste est dépend du serveur et de votre application. N'hésitez pas à faire des essais pour tout en surveillant la charge de votre serveur pour trouver le meilleur réglage
— *log-file* : ce sont les chemins vers les fichiers logs. Veillez bien que ces dossiers existent, et que votre utilisateur lançant le programme ait bien les droits d'écriture dans ces fichiers. C'est une source d'erreur fréquente.

On définit ensuite si notre programme doit démarrer automatiquement lors du démarrage de Supervisord et si il doit redémarrer en cas d'arrêt. Ce sont les parametres `autostart` et `autorestart` du fichier de configuration. Enfin on précise où enregistrer les logs Supervisord liés à cette application. Vérifiez bien que ces dossiers existent et que les droits sont corrects . Ces logs vont être très précieux pour vous aider à débugger si tout ne se passe pas comme prévu. Ces deux derniers réglages sont optionnels et vous pouvez laisser Supervisord choisir le dossier d'enregistrement des logs. Dans ce cas veillez à bien savoir où ils se trouvent !
N'hésitez pas à vous plonger plus en détails dans les paramètres de Supervisord pour découvrir d'autres paramètres. Voyons maintenant comment utiliser Supervisord pour démarrer notre application Nous allons commencer par recharger les fichiers de configuration à l'aide de la commande

Si vous avez activé l'autostart (comme dans l'exemple ci-dessus) votre application devrait être maintenant démarré. Sinon nous pouvons lancer un programme à l'aide de la commande

Il est possible de la même façon de stopper un programme ou de le redémarrer.

Pour connaître le statut de toutes vos applications en un clin d'œil utilisez la commande de status :

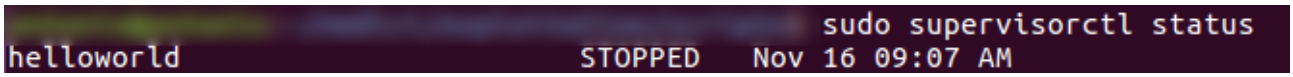

FIGURE 2.2. – Indication de l'état d'une tache supervisord, ici elle est arrêtée

Notre application tourne maintenant correctement et Supervisord se charge de la surveiller. En cas de plantage ou de redémarrage de la machine, Surpervisord relancera votre application pour qu'elle soit toujours disponible. Il ne nous reste plus qu'à rendre notre site accessible depuis l'ensemble des ordinateurs du réseau.

3. Configurer NGinx

3.1. Nginx

Nous allons utiliser la fonctionnalité de *reverse proxy* de Nginx, c'est à dire qu'il fera le lien entre une URL demandée et un serveur applicatif capable de renvoyer le contenu associé a cette URL. Si la configuration le permet, Nginx demandera la ressource associée à cette URL à Gunicorn et renverra la réponse à l'utilisateur.
Si Nginx n'est pas installé, il suffit de lancer la commande suivante :

```
```

On peut vérifier que le paquet est bien installé en affichant la version à l'aide la commande :

```
```

La configuration par défaut de Nginx se trouve dans le fichier `/etc/nginx/nginx.conf`, nous n'allons pas la modifier mais je vous invite à la lire pour en apprendre plus sur le fonctionnement de Nginx.
La configuration de Nginx se déroule dans deux dossiers :

- `/etc/nginx/site-available` qui contient les fichiers de configuration des "sites" qui peuvent être servis depuis notre serveur. C'est un dossier de dépôt, les configurations placées dans ce dossier ne sont pas utilisées directement.
- `/etc/nginx/site-enabled` qui contient la configuration des "sites" qui seront servis depuis notre serveur. Seuls les sites configurés dans ce dossier seront effectivement disponibles par les utilisateurs depuis l'extérieur.

On stocke donc nos configurations Nginx dans le dossier site-available et on les active en créant un lien symbolique dans le dossier site-enabled pointant vers site-available. Ainsi pour désactiver temporairement un site on peut simplement supprimer le lien symbolique et garder la configuration complète dans le dossier de dépôt.
En général on essaye de garder ces deux répertoires aussi organisés que possible. Pour cela une bonne pratique est de créer un fichier de configuration par site disponible. Les fichiers du dossier site-enabled sont chargés depuis le fichier `/etc/nginx/nginx.conf`.
Nginx est un outil très puissant et complexe, décrire l'ensemble des possibilités demanderait un cours à part entière, nous allons donc nous concentrer ici sur un exemple minimal, mais fonctionnel. Une configuration possible serait la suivante :

```
```

Arrêtons-nous quelques instants sur cette configuration et analysons la en détails.

- Le paramètre `upstream` permet de dire à Nginx de transmettre les requêtes des utilisateurs à un autre serveur. Dans notre cas on «redirige» la requête vers gunicorn *via* une *socket*. On précise une durée maximum de 30 secondes pendant laquelle Nginx tente à nouveau de contacter Gunicorn en cas d'échec. Au-delà de 30 secondes la requête se termine en échec.
- On ouvre ensuite un block `server` qui regroupe toute la configuration de notre site. Le découpage de la configuration en différents blocs `server` permet une très grande granularité dans les directives de configurations.
- La directive `listen` permet d'indiquer sur quel port va écouter et répondre Nginx, ici le port 80 (HTTP) et le port 443 (HTTPS). N'oubliez pas que pour que votre HTTPS

fonctionne correctement il faut établir un certificat et le préciser dans la configuration Nginx.
— Le server_name correspond au nom de domaine auquel votre serveur doit répondre. Notez bien que cela doit coller avec votre paramètre ALLOWED_HOSTS, sinon ça ne fonctionnera pas.
— Les deux lignes suivantes permettent de définir les fichiers d'enregistrements des *logs* d'accès et d'erreur. Ces fichiers sont importants en cas de problème afin de mieux comprendre ce qui se passe sur votre serveur.
— Le bloc suivant permet de définir le dossier où se trouvent les fichiers statiques de Django. Ils seront servis quand une url de type nom_de_domaine.com/static/nom_de_fichier sera appelée.

Pour rendre le site accessible nous allons créer un fichier dans le dossier sites-available et y coller la configuration analysée précédemment. Puis pour activer cette configuration on crée un lien symbolique depuis le répertoire site-enabled.

Nginx met à notre disposition un outil permettant de tester la configuration, pour cela il suffit de taper la commande suivante

Le résultat obtenu est le suivant :

En cas d'échec on obtient un message d'erreur précisant l'erreur à régler

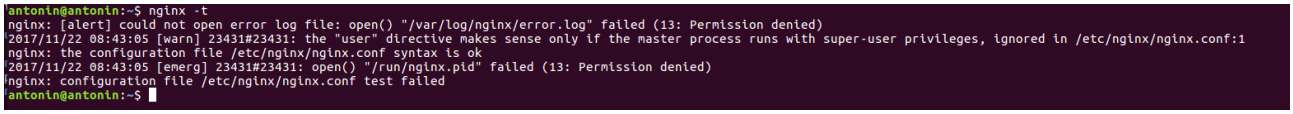

FIGURE 3.3. – Exemple d'erreur Nginx. Ici mon utilisateur ne possède pas les bons droits pour écrire dans le répertoire de log.

La dernière chose à faire est de redémarrer Nginx pour qu'il prenne en compte les modifications de la configuration. Pour cela on utilise la commande :

Et voila ! Votre application est accessible à l'adresse définie dans le serveur Nginx. Si vous avez configuré un vrai serveur il faudra ensuite configurer votre nom de domaine pour qu'il pointe vers votre serveur. Si vous êtes dans une machine virtuelle ou en locale il est possible de simuler le nom de domaine en éditant le fichier hosts de votre ordinateur . Ainsi j'ai attribué dans le fichier host le domaine hello.com à l'adresse localhost, je pourrais accéder à mon application en utilisant l'url hello.com:8010. Voici comment tester rapidement en ligne de commande :

On se retrouve dans la conclusion pour quelques idées d'améliorations

Ce tutoriel touche à sa fin. Nous aurons vu ensemble les bases pour déployer une application web codée en Python en production. Les logiciels utilisés sont à la fois puissants et très robustes, votre site web pourra encaisser des dizaines de connexions sans problème (sauf si vous avez des problèmes de performance ailleurs) !
Si vous rencontrez des problèmes de configurations n'hésitez pas à regarder les fichiers log (Nginx, Supervisor et Gunicorn) pour savoir ce qui coince .

3.1.1. En savoir plus

Pour approfondir je vous conseille de regarder de plus près les sujets suivants :

— Sentry : Un logiciel permettant de connaître les erreurs que vos utilisateurs rencontrent en production. Il est très utilisé dans l'écosystème Python et permet d'avoir plus d'informations sur les utilisateurs ayant rencontrés un bug (connectés ou non, état de l'application, etc.).
— Nginx : Ici nous avons couvert une utilisation très basique de ce logiciel, il est capable de faire beaucoup plus de choses : servir du contenu en HTTPS, autoriser l'accès à certaines adresses IP uniquement, etc.
— Automatiser le déploiement : Ici le lancement et la configuration de Nginx ou de Supervisord a été réalisé à la main, mais il est possible à l'aide d'outils de déploiement d'automatiser toutes ces actions. Envie d'en savoir plus, ça se passe ici .
— Gestion des dépendances : Ici nous avons utilisé l'outil le plus simple pour gérer les dépendances : pip. Il existe des outils plus complets tels que pipenv ou encore pip-tools qui permettent de gérer plus finement les dépendances de votre application.

3.1.2. Quelques liens

— Documentation de Gunicorn, example de configuration de Nginx
— La liste des *settings* de Gunicorn
— Un article en anglais sur le même sujet
— Mieux comprendre Nginx (EN)

Merci à artragis pour la validation de ce tutoriel.