Обслуживание приложений Django с помощью Apache и mod_wsgi в Ubuntu 16.04

Июнь 27, 2016 11:56 дп 4 703 views | 4 Comments
Python, Ubuntu | Amber

Django — это производительный веб-фреймворк для разработки приложений или сайтов в Python.

Django также предоставляет простой сервер разработки для локального тестирования кода. К сожалению, для производства необходимо искать более безопасный и мощный веб-сервер, так как этот сервер совершенно не подходит.

Данное руководство поможет установить и настроить Django в виртуальном окружении Python, а также настроить Apache для поддержки приложения и обработки клиентских запросов (для этого используется модуль Apache под названием mod_wsgi, который взаимодействует с Django через интерфейс WSGI).

Требования

Для выполнения данного руководства нужно предварительно настроить сервер Ubuntu 16.04 и создать не-root пользователя с доступом к sudo. Подробные инструкции по начальной настройке сервера и создании пользователей можно найти здесь.

Фреймворк Django в данном руководстве будет установлен в виртуальное окружение Python, что позволит использовать индивидуальный набор пакетов для данного приложения.

После запуска приложения нужно настроить взаимодействие веб-сервера Apache с приложением Django. Для этого используется модуль mod_wsgi, который преобразует HTTP-запросы в понятный Django формат согласно спецификации WSGI.

Установка пакетов из репозитория Ubuntu

Для начала нужно установить из репозитория Ubuntu все необходимые пакеты: веб-сервер Apache, модуль mod_wsgi и pip (пакетный менеджер Python).

Обновите индекс пакетов и установите программы.

Если вы используете Python 2, введите:

```
sudo apt-get update
sudo apt-get install python-pip apache2 libapache2-mod-wsgi
```

В Python 3 используется другой пакет pip:

```
sudo apt-get update
sudo apt-get install python3-pip apache2 libapache2-mod-wsgi-py3
```

Настройка виртуальной среды

Подготовив сервер, можно приступать к работе над приложением Django. Сначала создайте виртуальное окружение Python для хранения инструментов проекта.

Для этого используйте команду virtualenv. Чтобы установить эту утилиту, введите:

```
Python 2:
sudo pip install virtualenv
Python 3:
sudo pip3 install virtualenv
```

Теперь утилита virtualenv установлена, и можно использовать её для создания окружения. Создайте каталог, в котором будет находиться виртуальное окружение, и перейдите в него:

```
mkdir ~/myproject
cd ~/myproject
```

В этом каталоге создайте виртуальную среду Python:

```
virtualenv myprojectenv
```

Эта команда создаст виртуальное окружение по имени myprojectenv в каталоге myproject, а также установит локальную версию Python и pip, которые можно использовать для создания изолированной среды разработки проекта.

Прежде чем приступить к установке зависимостей приложения, нужно включить виртуальное окружение.

```
source myprojectenv/bin/activate
```

После этого командная строка изменится, указывая, что текущей средой является виртуальная среда Python:

```
(myprojectenv)user@host:~/myproject$
```

Включив виртуальную среду, установите Django:

```
pip install django
```

Примечание: Вне зависимости от версии Python в активной виртуальной среде используется команда pip, а не pip3.

Создание и настройка Django-приложения

Установив Django в виртуальную среду, можно приступать к созданию файлов Django-проекта.

Создание Django-проекта

Установите файлы Django в ранее созданный каталог. Это создаст каталоги второго уровня, хранящие код и скрипты. Обратите внимание: команда должна заканчиваться символом точки.

```
django-admin.py startproject myproject .
```

Настройка Django-проекта

Первое, что нужно сделать после создания приложения, — отредактировать его настройки. Откройте файл settings в текстовом редакторе:

```
nano myproject/settings.py
```

В руководстве используется стандартная БД SQLite, потому настройка не займёт много времени.

Примечание: Для поддержки масштабных приложений рекомендуется настроить более надёжную СУБД.

В центре внимания будет настройка каталога статических файлов Django.

В конец файла добавьте директиву STATIC_ROOT, которая задаёт фреймворку Django местонахождение статических файлов.

```
. . .
STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'static/')
```

Сохраните и закройте файл.

После этого можно открыть исходную схему БД при помощи скрипта управления:

```
cd ~/myproject
./manage.py makemigrations
./manage.py migrate
```

Создайте аккаунт администратора проекта:

```
./manage.py createsuperuser
```

Укажите имя пользователя, адрес электронной почты и пароль.

Затем нужно собрать статический контент в один каталог:

```
./manage.py collectstatic
```

После подтверждения все статические файлы будут помещены в каталог static в каталоге проекта.

Теперь нужно разблокировать порт сервера разработки Django, 8000. На данный момент он закрыт брандмауэром UFW (если вы настроили сервер согласно этому руководству).

Чтобы разблокировать сервер разработки, введите:

```
sudo ufw allow 8000
```

Протестируйте проект, запустив сервер разработки Django:

```
./manage.py runserver 0.0.0.0:8000
```

Откройте браузер и посетите доменное имя или IP сервера на порте 8000:

```
http://server_domain_or_IP:8000
```

На экране должна появиться приветственная страница Django:

```
It worked!
Congratulations on your first Django-powered page.
```

Чтобы получить доступ к аккаунту администратора, добавьте секцию /admin к адресу, а затем введите имя и пароль.

Пройдя авторизацию, вы получите доступ к интерфейсу администратора.

Чтобы остановить сервер разработки, нажмите CTRL-C.

Чтобы отключить виртуальную среду, введите:

```
deactivate
```

Настройка Apache

Итак, проект Django готов, и можно приступать к настройке Apache на фронт-энде. Все клиентские подключения, полученные Apache, будут преобразованы в формат WSGI, необходимый Django, при помощи модуля mod_wsgi (который должен был включиться автоматически после установки).

Чтобы настроить WSGI, отредактируйте стандартный виртуальный хост.

```
sudo nano /etc/apache2/sites-available/000-default.conf
```

Все директивы, которые находятся в файле, можно оставить. Нужно только добавить несколько новых опций.

Сначала настройте статические файлы; после этого Apache будет направлять все запросы, начинающиеся с /static, в каталог static, который находится в каталоге проекта. Для этого нужно определить Alias и предоставить доступ к требуемому каталогу.

```
<VirtualHost *:80>
. . .
Alias /static /home/8host/myproject/static
<Directory /home/8host/myproject/static>
Require all granted
</Directory>
</VirtualHost>
```

Затем нужно настроить права доступа к файлу wsgi.py (второй уровень каталогов проекта), в котором хранится код проекта. Для этого используйте раздел Directory.

```
<VirtualHost *:80>
. . .
Alias /static /home/8host/myproject/static
<Directory /home/8host/myproject/static>
Require all granted
</Directory>
<Directory /home/8host/myproject/myproject>
<Files wsgi.py>
Require all granted
</Files>
</Directory>
</VirtualHost>
```

Теперь можно перейти к настройкам, обрабатывающим WSGI. Для запуска процессов WSGI используйте режим демона; для этого используется директива WSGIDaemonProcess. Эта директива принимает для процесса произвольное имя. В руководстве используется имя myproject.

Затем нужно указать путь к Python, где сервер Apache сможет найти все необходимые компоненты. Поскольку в настройке используется виртуальное окружение, нужно указать путь к каталогу виртуальной среды, а затем – путь Python к базовому каталогу проекта Django.

После этого нужно указать группу процессов; это значение должно совпадать со значением директивы WSGIDaemonProcess (в данном случае это myproject). В завершение нужно установить алиас скрипта, чтобы сервер Apache передавал запросы для корневого домена в файл wsgi.py:

```
<VirtualHost *:80>
. . .
Alias /static /home/8host/myproject/static
<Directory /home/8host/myproject/static>
Require all granted
</Directory>
<Directory /home/8host/myproject/myproject>
<Files wsgi.py>
Require all granted
</Files>
</Directory>
WSGIDaemonProcess myproject python-home=/home/8host/myproject/myprojectenv python-path=/home/8host/myproject
WSGIProcessGroup myproject
WSGIScriptAlias / /home/8host/myproject/myproject/wsgi.py
</VirtualHost>
```

Сохраните и закройте файл.

Права доступа

Используя базу данных SQLite (которая является стандартной), не забудьте предоставить веб-серверу Apache доступ к ней.

Для этого нужно предоставить группе-владельцу права на запись и чтение. По умолчанию файл БД называется db.sqlite3 и должен находиться в каталоге проекта:

```
chmod 664 ~/myproject/db.sqlite3
```

Передайте права на файл группе, в которую входит Apache; это группа www-data.

```
sudo chown :www-data ~/myproject/db.sqlite3
```

Чтобы иметь право на запись в файле, группа должна быть владельцем родительского каталога файла.

```
sudo chown :www-data ~/myproject
```

Теперь нужно снова отредактировать настройки брандмауэра. Закройте порт 8000 и добавьте исключение для трафика Apache:

```
sudo ufw delete allow 8000
sudo ufw allow 'Apache Full'
```

Проверьте синтаксис файлов Apache:

```
sudo apache2ctl configtest
```

Если в файлах нет ошибок, команда вернёт:

```
. . .
Syntax OK
```

Перезапустите сервис Apache, чтобы обновить настройки.

```
sudo systemctl restart apache2
```

Теперь можно получить доступ к сайту Django, открыв в браузере доменное имя или IP-адрес.

Примечание: Номер порта указывать не нужно.

Дальнейшие действия

Теперь приложение Django работает в индивидуальной виртуальной среде, а модуль mod_wsgi помогает Apache преобразовывать запросы в понятный Django формат.

Убедившись, что приложение работает, вы можете приступать к настройке безопасности. Если у приложения есть доменное имя, можно получить бесплатный SSL-сертификат от Let’s Encrypt.

Если у вас нет домена, вы можете создать сертификат самостоятельно.

Tags: Apache, Django, mod_wsgi, Python, Python 2, Python 3, SQLite, Ubuntu 16.04, virtualenv