How to use Django with Apache and `mod_wsgi`

Deploying Django with Apache and mod_wsgi is a tried and tested way to get Django into production.

mod_wsgi is an Apache module which can host any Python WSGI application, including Django. Django will work with any version of Apache which supports mod_wsgi.

The official mod_wsgi documentation is your source for all the details about how to use mod_wsgi. You'll probably want to start with the installation and configuration documentation.

Basic configuration

Once you've got mod_wsgi installed and activated, edit your Apache server's httpd.conf file and add the following.

```
WSGIScriptAlias / /path/to/mysite.com/mysite/wsgi.py
WSGIPythonHome /path/to/venv
WSGIPythonPath /path/to/mysite.com

<Directory /path/to/mysite.com/mysite>
<Files wsgi.py>
Require all granted
</Files>
</Directory>
```

The first bit in the **`WSGIScriptAlias`** line is the base URL path you want to serve your application at (**`/`** indicates the root url), and the second is the location of a "WSGI file" – see below – on your system, usually inside of your project package (**`mysite`** in this example). This tells Apache to serve any request below the given URL using the WSGI application defined in that file.

If you install your project's Python dependencies inside a **`virtual environment`**, add the path using **`WSGIPythonHome`**. See the mod_wsgi virtual environment guide for more details.

The **`WSGIPythonPath`** line ensures that your project package is available for import on the Python path; in other words, that **`import mysite`** works.

The **`<Directory>`** piece ensures that Apache can access your **`wsgi.py`** file.

Next we'll need to ensure this **`wsgi.py`** with a WSGI application object exists. As of Django version 1.4, **`startproject`** will have created one for you; otherwise, you'll need to create it. See the WSGI overview documentation for the default contents you should put in this file, and what else you can add to it.

Warning

If multiple Django sites are run in a single mod_wsgi process, all of them will use the settings of whichever one happens to run first. This can be solved by changing:

```
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "{{ project_name }}.settings")
```

in **`wsgi.py`**, to:

```
os.environ["DJANGO_SETTINGS_MODULE"] = "{{ project_name }}.settings"
```

or by using mod_wsgi daemon mode and ensuring that each site runs in its own daemon process.

Fixing `UnicodeEncodeError` for file uploads

If you get a **`UnicodeEncodeError`** when uploading files with file names that contain non-ASCII characters, make sure Apache is configured to accept non-ASCII file names:

```
export LANG='en_US.UTF-8'
export LC_ALL='en_US.UTF-8'
```

A common location to put this configuration is **`/etc/apache2/envvars`**.

See the Files section of the Unicode reference guide for details.

Using `mod_wsgi` daemon mode

"Daemon mode" is the recommended mode for running mod_wsgi (on non-Windows platforms). To create the required daemon process group and delegate the Django instance to run in it, you will need to add appropriate **`WSGIDaemonProcess`** and **`WSGIProcessGroup`** directives. A further change required to the above configuration if you use daemon mode is that you can't use **`WSGIPythonPath`**; instead you should use the **`python-path`** option to **`WSGIDaemonProcess`**, for example:

```
WSGIDaemonProcess example.com python-home=/path/to/venv python-path=/path/to/mysite.com
WSGIProcessGroup example.com
```

If you want to serve your project in a subdirectory (**`https://example.com/mysite`** in this example), you can add **`WSGIScriptAlias`** to the configuration above:

```
WSGIScriptAlias /mysite /path/to/mysite.com/mysite/wsgi.py process-group=example.com
```

See the official mod_wsgi documentation for details on setting up daemon mode.

Serving files

Django doesn't serve files itself; it leaves that job to whichever Web server you choose.

We recommend using a separate Web server – i.e., one that's not also running Django – for serving media. Here are some good choices:

- Nginx
- A stripped-down version of Apache

If, however, you have no option but to serve media files on the same Apache **`VirtualHost`** as Django, you can set up Apache to serve some URLs as static media, and others using the mod_wsgi interface to Django.

This example sets up Django at the site root, but serves **`robots.txt`**, **`favicon.ico`**, and anything in the **`/static/`** and **`/media/`** URL space as a static file. All other URLs will be served using mod_wsgi:

```
Alias /robots.txt /path/to/mysite.com/static/robots.txt
Alias /favicon.ico /path/to/mysite.com/static/favicon.ico

Alias /media/ /path/to/mysite.com/media/
Alias /static/ /path/to/mysite.com/static/

<Directory /path/to/mysite.com/static>
Require all granted
</Directory>

<Directory /path/to/mysite.com/media>
Require all granted
</Directory>

WSGIScriptAlias / /path/to/mysite.com/mysite/wsgi.py

<Directory /path/to/mysite.com/mysite>
<Files wsgi.py>
Require all granted
</Files>
</Directory>
```

Serving the admin files

When **`django.contrib.staticfiles`** is in **`INSTALLED_APPS`**, the Django development server automatically serves the static files of the admin app (and any other installed apps). This is however not the case when you use any other server arrangement. You're responsible for setting up Apache, or whichever Web server you're using, to serve the admin files.

The admin files live in (**`django/contrib/admin/static/admin`**) of the Django distribution.

We **strongly** recommend using **`django.contrib.staticfiles`** to handle the admin files (along with a Web server as outlined in the previous section; this means using the **`collectstatic`** management command to collect the static files in **`STATIC_ROOT`**, and then configuring your Web server to serve **`STATIC_ROOT`** at **`STATIC_URL`**), but here are three other approaches:

1. Create a symbolic link to the admin static files from within your document root (this may require **`+FollowSymLinks`** in your Apache configuration).
2. Use an **`Alias`** directive, as demonstrated above, to alias the appropriate URL (probably **`STATIC_URL`** + **`admin/`**) to the actual location of the admin files.
3. Copy the admin static files so that they live within your Apache document root.

Authenticating against Django's user database from Apache

Django provides a handler to allow Apache to authenticate users directly against Django's authentication backends. See the mod_wsgi authentication documentation.

« How to use Django with uWSGI

Authenticating against Django's user database from Apache »

Contents

Browse

- Prev: How to use Django with uWSGI
- Next: Authenticating against Django's user database from Apache
- Table of contents
- General Index
- Python Module Index

You are here:

Getting help

FAQ
Try the FAQ — it's got answers to many common questions.

Index, Module Index, or Table of Contents
Handy when looking for specific information.

django-users mailing list
Search for information in the archives of the django-users mailing list, or post a question.

#django IRC channel
Ask a question in the #django IRC channel, or search the IRC logs to see if it's been asked before.

Ticket tracker
Report bugs with Django or Django documentation in our ticket tracker.

Download:

Offline (Django 3.1): HTML | PDF | ePub
Provided by Read the Docs.