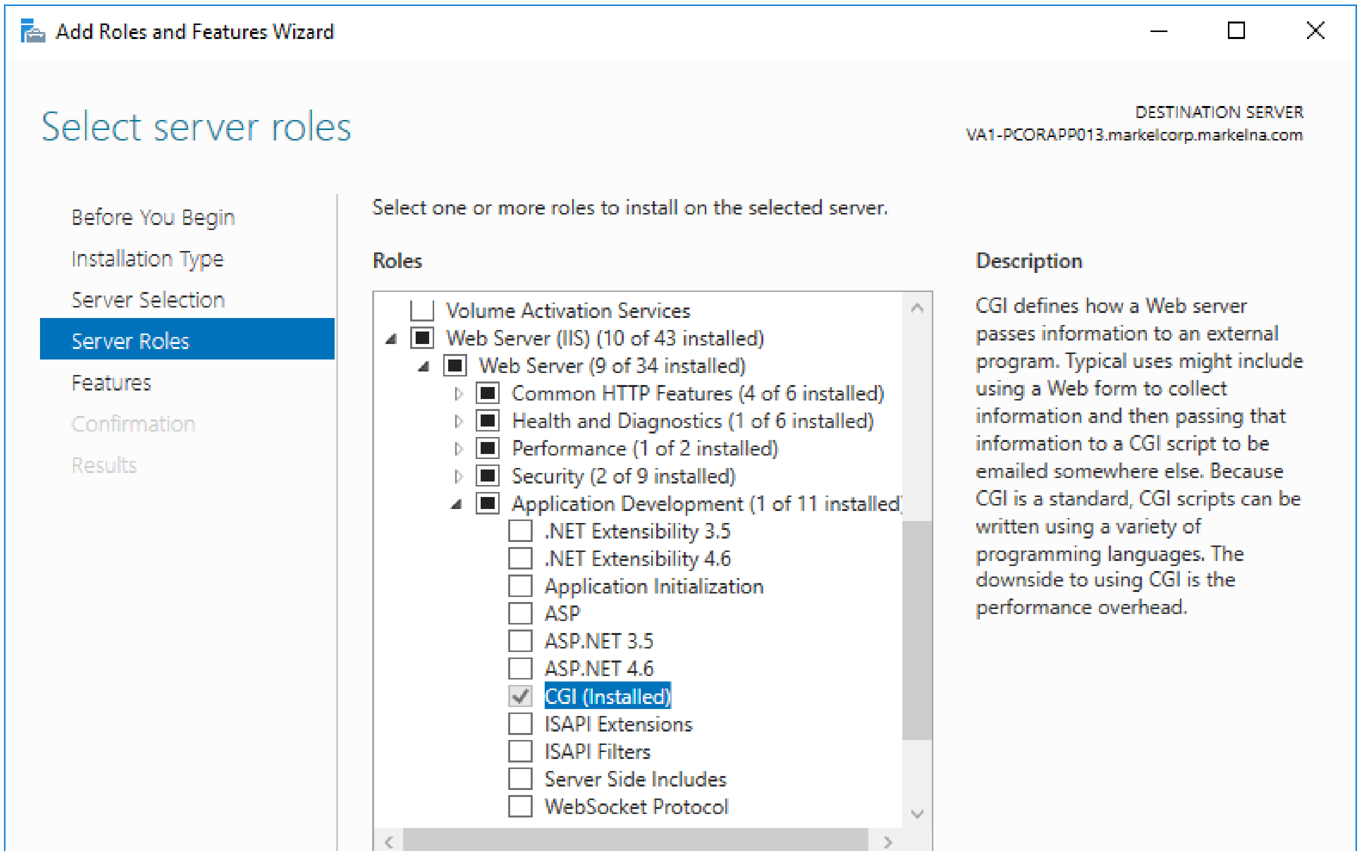
Deploy Django on IIS

Jonny Fuller Follow
May 16, 2019 · 8 min read

I needed to combine advice from quite a few blog posts detailing how to deploy Django to a classic IIS server. Many only contained one piece of the puzzle. Others, like this excellent post from Matt Woodward I could only access through waybackmachine. Therefore, I wanted to preserve my findings here for others and myself.

Enable IIS

First, make sure the host server's IIS is installed and has CGI enabled. To verify, from Windows search or the server management window click "Add Roles and Features". Skip through the the dialogues, make sure the target server is selected under Server Selection, then pause on the Server Roles tab. Make sure Web Server IIS is selected, and add the optional Application Development checkbox for CGI like shown.



Add Roles and Features Wizard

Select server roles

DESTINATION SERVER
VA1-PCORAPP013.markelcorp.markelna.com

Before You Begin
Installation Type
Server Selection
Server Roles
Features
Confirmation
Results

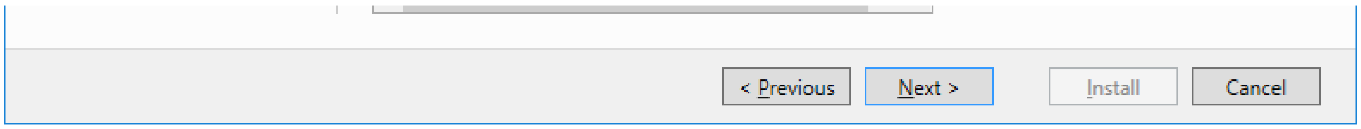
Select one or more roles to install on the selected server.

Roles

- [ ] Volume Activation Services
- [■] Web Server (IIS) (10 of 43 installed)
  - [■] Web Server (9 of 34 installed)
    - [■] Common HTTP Features (4 of 6 installed)
    - [■] Health and Diagnostics (1 of 6 installed)
    - [■] Performance (1 of 2 installed)
    - [■] Security (2 of 9 installed)
    - [■] Application Development (1 of 11 installed)
      - [ ] .NET Extensibility 3.5
      - [ ] .NET Extensibility 4.6
      - [ ] Application Initialization
      - [ ] ASP
      - [ ] ASP.NET 3.5
      - [ ] ASP.NET 4.6
      - [✓] CGI (Installed)
      - [ ] ISAPI Extensions
      - [ ] ISAPI Filters
      - [ ] Server Side Includes
      - [ ] WebSocket Protocol

Description

CGI defines how a Web server passes information to an external program. Typical uses might include using a Web form to collect information and then passing that information to a CGI script to be emailed somewhere else. Because CGI is a standard, CGI scripts can be written using a variety of programming languages. The downside to using CGI is the performance overhead.



(note yours may not say installed yet)

Click next to get through the screens then accept the changes. Once IIS is installed we'll need to set up our Python environment.

Python/Django Setup

Install Python, preferably as a user account with administrative privileges. The Python installer **hides important options behind a next** button. I recommend adding Python to the path and installing for all users.

Next we need to setup our Django environment. There's two ways to do this. Option one is we install our application dependencies to the system interpreter, but ideally one should install to a virtualenv as outlined below.

Virtualenv allows us to setup a copy of Python specifically for our Django application. From the command line ensure pip and setuptools are fully upgraded.

```
python -m pip install pip --upgrade && pip install setuptools --
upgrade
```

Then install the virtualenv tool

```
pip install virtualenv
```

Navigate to the default IIS sites folder "C:\inetpub\wwwroot", then git clone your repo so that your manage.py lives at "C:\inetpub\wwwroot\mydjangoproject\manage.py", where mydjangoproject is your repo-root and all its files. I like to put the virtualenv in the repo-root and ignore it with .gitignore. With the new repo-root as your current working directory use the virtualenv command to install the virtualenv.

```
virtualenv venv
```

This will add a “venv” folder with a copy of the Python interpreter and its own library to “C:\inetpub\wwwroot\mydjangoproject\”.

Activate the virtualenv so all our commands affect this interpreter.

```
C:\inetpub\wwwroot\mydjangoproject\> venv\scripts\activate
```

You will know the virtualenv is toggled because the terminal will be prefixed by (venv).

If wfastcgi is not already in your requirements.txt, add it. Now we can install all the project’s requirements to the virtualenv.

```
pip install -r requirements.txt
```

Django Settings

A Django project needs settings to run. And since Django is just Python, there’s a million various ways to configure the settings. They can live in any importable Python module in any location. In my humble opinion, a good best practice is to separate dev/prod settings, and then use a git-ignored environment module or an environmental variable switch between them.

- A quick note. Django uses the environmental variable DJANGO_SETTINGS_MODULE to know which settings module to import. By default, manage.py sets this environmental variable. It’s probably a good idea to keep the manage.py script’s value and whatever value you configure for IIS in sync.
- Within the settings module is a very important key WSGI_APPLICATION, which should be the dot path to the application object. This key is how Django’s django.core.wsgi.get_wsgi_application() method provides wsfastcgi an application object. More on this in a bit…

Option 1) — Split settings and git-ignored environment.py — Within your Django project make a package called settings and then create a base.py, dev.py, prod.py, and environment.py. Copy your default settings.py in to base.py. Then add your development settings (ie DEBUG = True, your personal developer database connections, etc) to dev.py and your production settings to prod.py. Source control everything but ignore environment.py. Within environment.py import the settings suiting your target. Ie if your in production your environment.py should read:

```
from .base import *
from .prod import *
```

The last settings imported take precedence, so .prod will override any names in .base.

Now we need to tell Django environment.py is the appropriate settings file. Edit manage.py to reflect the path to environment. Example, if your settings package is in

```
"C:\inetpub\wwwroot\mydjangoproject\mydjangoproject\settings"
```

(Where the first mydjangoproject is the repo-root and second mydjangoproject contains all the importable Python packages for the project)

Then manage.py's django setting statement should read

```
os.environ.setdefault('DJANGO_SETTINGS_MODULE',
'mydjangoproject.settings.environment')
```

Where "mydjangoproject.settings.environment" is the importable dot path to the environment settings module.

The advantage of this strategy is you must explicitly add an environment.py module to your particular environment. As this file never becomes a part of source, we're protected from accidentally interacting with the wrong environment. The disadvantage is now an additional file must be maintained.

Option 2) — Source control everything but use environmental variables

Alternatively, we can just set our **DJANGO_SETTINGS_MODULE** environmental variable to point to either prod or dev. The disadvantage is that if this setting is accidentally configured for the wrong file, or a manage.py, or wsgi.py, sets a different value, we could accidentally interact with the wrong environment. Since both manage.py and wsgi.py modules will be source controlled, a mistake becomes possible. Ideally, because all these entry points use environ.setdefault, explicitly set settings would not get overwritten. Still, makes me nervous.

Configure IIS WFast CGI

By now, your Python environment should be setup and your Django settings should be configured. We need to configure IIS to use FastCGI.

From the IIS Manager select the server, then choose the FastCGI settings Icon.

(If you don't have this icon you may need to install FastCGI using "Get new Web Platform Components" on the right).

Set the full path to your target Python.exe (either the system interpreter or the virtual environment's). Set the arguments to the wfastcgi module (probably in your sitepackages).

here my project is "fraud_django"

Next we need to set three key environmental variables so Django knows where the settings live, wfastcgi knows how to obtain Django's callable application object, and Python knows how to import everything.

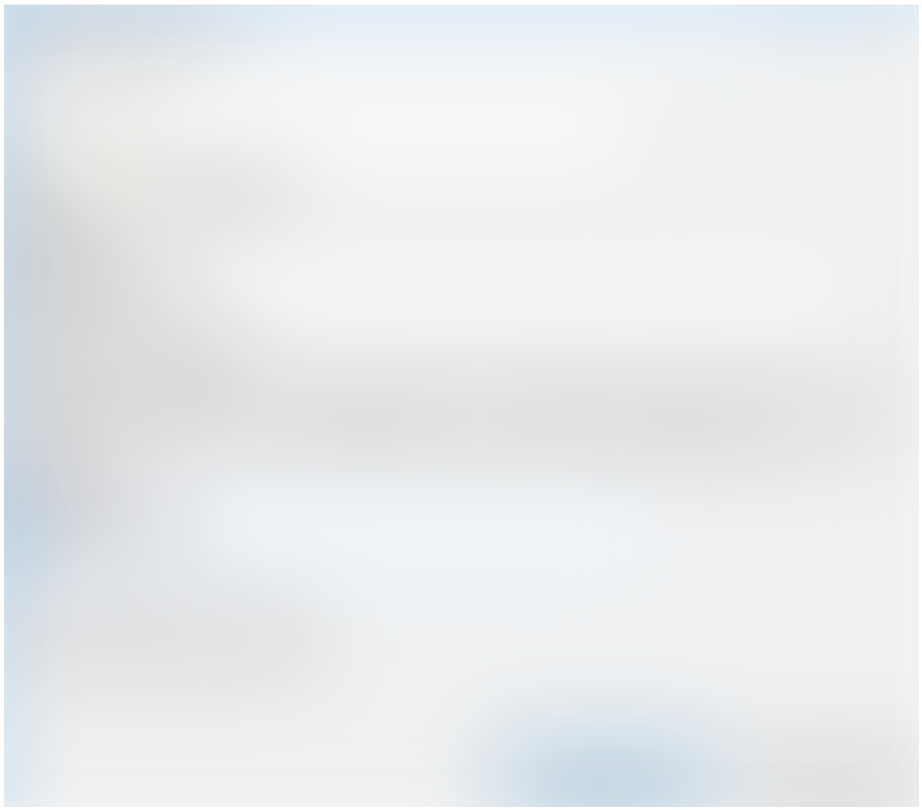
1. DJANGO_SETTINGS_MODULE = "path to your settings"
2. WSGI_HANDLER = django.core.wsgi.get_wsgi_application()
3. PYTHONPATH = "path to your repo-root ie in my case C:\inetpub\wwwroot\fraud_django"

Right click sites and select add. Give it a name of your choosing and set the physical path the the wwwroot. Next we need to map the FastCGI handler to the site. Select your new site and choose handler mappings. Click "Add Module Mapping…" from the actions selection on the right.

Set request path to * and Module to FastCgiModule. However, the executable section **must be set manually** as we need to select the Python executable and pipe to wfastcgi, which the UI's … browse button prohibits. For example, my value uses the virtualenv paths:

```
C:\inetpub\wwwroot\fraud_django\venv\Scripts\python.exe|C:\inetpub\ww
wroot\fraud_django\venv\Lib\site-packages\wfastcgi.py
```

(Note the pipe with no whitespace) You can give it whatever name you see fit. Your mapping should look something like below.



(fraud_django is my project's name, your repo-root can be anything)

We're not done yet. Be sure to click Request Restrictions and uncheck "Invoke handler only if request is mapped to". When prompted "Do you want to create a FastCGI application for this executable" click "No" because we previously created one. What's the difference between adding the executable for the site via the above versus the original configuration we performed on the FastCGI server settings? I have no idea and will update if I find out.

Serving Static Files

We only want Django to serve dynamic content. All our static files will be served by IIS without going through FastCGI. Two steps are required to accomplish this task.

1. Configure the **STATIC_URL** setting so Django's reverse url functionality knows where the static end point begins. I suggest using the default value of '/static/'.

2. Create a static folder and serve it as a virtual directory. I like to use a folder called "static" in the repo-root. For example, for my project this folder lives at "C:\inetpub\wwwroot\fraud_django\static". Next, set the **STATIC_ROOT** setting to point to this folder so Django's manage.py collectstatic command knows where to copy all the content. You can use Python's pathlib Path class and __file__ to easily set this location dynamically like in this example from base.py within my settings module.

```
from pathlib import Path

BASE_DIR = Path(__file__).parents[2]

STATIC_ROOT = BASE_DIR / Path('static')
```

Once the folder is created, and the path set to STATIC_ROOT, mount the folder as a virtual directory by right clicking on your site and choosing the option. In the proceeding dialog sure to set the alias to whatever you named in STATIC_URL, ie "static".

By default IIS will carry the previously configured fastcgi handler mapping to this folder, **causing your site to not work** and log 404s. Click on static and be sure to remove the handler you configured for Django.

Now your site should work! Below are some helpful trouble shooting tips.

Trouble Shooting/Common Gotch yas

1. Check the application pools, and make sure one of them contains your application (either DefaultAppPool or you may have created one for Django). If the site still does not load click the application pool containing your Django app and try changing the identity to a different user like LocalSystem.

2. Make sure you removed the django handle from the static virtual directory's handler mappings.

3. Make sure you remembered to uncheck "Invoker handler only if request is mapped to" by clicking the Request Restrictions button on the module mapping dialog.

4. After configuring the server the first time I then deleted the configuration to start from scratch to write this guide. During that time, somehow I managed to mess up the configuration because the FastCGI "app" was defined at the site level. The correct configuration appears to be making sure FastCGI settings are set at the server level, and application is defined in the site, but rather the site is configured to use the FastCgiModule in the edit module mapping dialog. I still do not know what the

difference is between setting FastCGI Settings at the server level, then redeclaring them again during the module mapping process, but it now works.

5. Remember to add your hostname to your Django setting's ALLOWED_HOSTS key. I cannot tell you how many times I forgot to do this.

)

Python Django Deployment

About Help Legal