## this has been integrated into the Django doc

## About uWSGI, uwsgi, wsgi, and WSGI

The purpose of uWSGI is to link nginx up to a python ‘callable’ - a function or method. uWSGI is a gasket. In apache speak, it is a module. There are alternative programs that do the same thing while using the same front end protocol and back end API, one such program is *gunicorn*. uWSGI is the most commonly used and appears to be a favorite.

uWSGI was written in C and appears as a compiled binary in a python pip distribution. It is run as a server by using the uwsgi command.

The official website for uWSGI program is, according to wikipedia, <http://projects.unbit.it/uwsgi>. That link forwards to <https://uwsgi-docs.readthedocs.io/en/latest/>.

At the front side the uwsgi server listens to a port using the wsgi protocol, on the back side it calls a Python ‘callable’. The back side API definition is called the Python Web Server Gateway Interface, which is abbreviated as WSGI. Yes, anyone can see that this API should have been abbreviated PWSGI, but it wasn’t. The documentation calls the front side of a uWSGI server the ‘server side’ or ‘gateway side’, and the back side the ‘application side’, or the ‘framework’ side.

On the front side, the gateway side, the uwsgi server will accept input and produce responses on a socket while using exactly one of two protocols.

1. http
2. wsgi

Because uwsgi may be set up to accept http protocol on a socket, the uwsgi program may act as a web server. It is not a full featured web server, rather the web server mode is intended to be used for testing and possibly for internal use.

Nginx may communicate wtih the uwsgi server through either:

1. An IP socket
2. A unix socket

For communication over an IP socket a port number must be reserved for each uwsgi server. When using unix sockets a file name in the system must be reserved. Embedding a high numbered port into the launch script for uwsgi can lead to startup reliability problems if another IP user happens to have already taken that port. There is no way to permanently reserve port high number port numbers. Lower numbered port numbers follow a convention for services. Hence, the unix socket approach is preferable. The challenge with the unix socket is to work out the ownership permissions so that nginx and uwsgi may run as different users and groups and thus remain isolated.

The wsgi protocol is a binary protocol layer over tcp. It is documented here, <https://uwsgi-docs.readthedocs.io/en/latest/Protocol.html> The uwsgi protocol is an esoteric protocol spoken only by certain webservers. Nginx is one of them. The system is transaction based, so it seems reasonable to expect that the wsgi protocol just wraps http requests and responses on a one to one basis.

So why doesn’t django just implement the wsgi protocol? The answer is that performance python libraries are in fact often written in C. If we were to add a library component to speak the wsgi protocol, we would probably write it in C and call it *wsgi*.

On the back side, the application side, uwsgi may be configured to:

1. connect to files
2. call a python program that follows the WSGI API, i.e. the PEP333 spec.

When the front side of uwsgi is speaking http, and the back side is connected to files, uwsgi is acting as a classic web server. uwsgi can route static traffic to a different directory (just as nginx and django can).

The WSGI API is documented here, <https://www.python.org/dev/peps/pep-0333/>. The API describes calling a single ‘callable’, a function or a method and apparently some other forms. The name of this callable defaults to ‘application’, but the name can also be configured. Two parameters will be provided, one being the environment, the other being a ‘start\_response’ callable. Because we are transaction processing, there will be one call per ‘http request’, (Though that http request might have been wrapped in according to the wsgi protocol.)

## Summary of Nomenclature

Here is a summary of the official nomenclature.

* uWSGI - the uwsgi program project.
* uwsgi - the command for invoking the uWSGI server.
* wsgi - a protocol used for communicating through sockets
* WSGI - a Python API for talking about http requests and responses

It is apparent from reading through google pages that people are confused about what uWSGI does and the nomenclature that is used, and this can make it difficult to find information related to uwsgi the program, wsgi the protocol, and WSGI the API. Google searches for “wsgi protocol” invariably produce hits for pages about the WSGI API. (The python API seems to dominate the literature on the subject.)

All three of these, uwsgi the program, wsgi the protocol, and WSGI the API, have the same official website for documenting them. That being [readthedocs.io](http://readthedocs.io/). This site has two subdomains uwsgi and wsgi, but the partitioning doesn’t seem to create the expected separation of topics. For example, recall that the wsgi protocol is documented at the link, <https://uwsgi-docs.readthedocs.io/en/latest/Protocol.html>. Note two things about this link. Firstly, it is the same site that documents the uWSGI program. Secondly, at the subdomain, *uwsgi-docs* a person would expect to find docs on *uWSGI the program*, but instead, here is where we find docs on the *uwsgi protocol*.

The *official* links given in various places, [wsgi.org](http://wsgi.org/) (where one would expect documentation on the wsgi protocol and the WSGI API) and <http://projects.unbit.it/uwsgi> the official page for the uWSGI program, both lead to *the exact same page!*  <https://wsgi.readthedocs.io/en/latest/>.

## Installation, Control, and Configuration

uwsgi is installed via pip.

pip install uwsgi

uWSGI is started by calling with comman uwsgi. Options can be listed with --help. Here is a list: <https://uwsgi-docs.readthedocs.io/en/latest/Options.html>. Also

> uwsgi ini help.

uwsgi must be manually added to systemd: <https://uwsgi-docs.readthedocs.io/en/latest/Systemd.html>.

The configuration file is identical to the command line options. It is loaded via the --ini option. <https://uwsgi-docs.readthedocs.io/en/latest/Configuration.html>

E.g.

# uwsgi --http :8000 --wsgi-file customer\_gateway/wsgi.py

Is the same as

# uwsgi --ini uwsgi.ini

# cat uwsgi.ini

[uwsgi]

http = :8000

wsgi-file = customer\_gateway/wsgi.py

Config files start with the string “[uwsgi]”. When assigning to options there are no quotes around string values.

I think it is best to explicitly provide home and chdir, otherwise uwsgi must be launched in the django project directory.

> uwsgi --http :8000 --home /var/www/html/customer\_gateway/env --chdir /var/www/html/customer\_gateway/customer\_gateway --module customer\_gateway.wsgi

Or

> uwsgi --ini uwsgi.parms

> cat uwsgi.parms

[uwsgi]

http = :8000 # runs http protocol on this tcp port

# http-socket = :8000 #runs http protocol on this tcp port

# socket = :8000 # runs wsgi protocol on this tcp

# socket = /home/nginx\_customer\_gateway\_mediator/socket # runs wsgi protocol on this unix socket

home = /var/www/html/customer\_gateway/env

chdir = /var/www/html/customer\_gateway/customer\_gateway

module = customer\_gateway.wsgi

uwsgi picks one of the two front side protocols based on the parameter used for specifying the front side location. If it is ‘http’ or ‘http-socket’ uwsgi is going to speak http. If it is ‘socket’ then it is going to speak wsgi. You can open a browser on the port when it speaks http, but not when it speaks wsgi.

The problem with hard configuring a tcp socket is that conceivably the port will not be available when the uwsgi server is run.

The problem with configuring a unix socket is that those are in the file system and have owner, group, and permissions. uwsgi makes creates the socket when it is run, and deletes the socket when the process terminates. Because it makes the socket, uwsgi sets the permissions. There are options for this. For example,

[uwsgi]

...

uid=myuser

gid=mygroup

chown-socket = %(uid):www-data

chmod-socket = 660

This is the first test passing version of the customer\_gateway ini file:

/var/www/html/customer\_gateway/uwsgi\_wsgi\_unix\_socket.ini:

[uwsgi]

chdir = /var/www/html/customer\_gateway/customer\_gateway

master = True

vacuum = True

uid = customer\_gateway

gid = customer\_gateway

socket = /home/nginx\_customer\_gateway\_mediary/socket

chown-socket = %(uid):nginx\_customer\_gateway\_mediary

chmod-socket = 660

home = /var/www/html/customer\_gateway/env

module = customer\_gateway.wsgi

At startup uwsgi reads the wsgi.py file and the settings.py file in the django settings directory (a subdirectory with the same name as the project). If you use specify only the wsgi.py file explicitly, and there is a settings.py file in the same directory, uwsgi will find it.

The settings.py file has a variable called ‘ALLOWED\_HOSTS’. The server hostname and all means for accessing the site must be listed there, e.g. also the IP. These values are strings and must be quoted. When the variable is not set, and all else is working, there will be a helpful message suggesting to set it. After any changes uwsgi must be restarted to coerce it to re-read the setting file.

## Nginx Support for wsgi and uwsgi

In the nginx.conf file there is an http directive block, a server conf file in /etc/nginx/sites-available will be included into that block. Inside that server conf file there will be a server directive block. Inside of that server block there will be a location block. Inside the location block nginx supports the uwsgi\_pass directive. This directive is built into nginx.

The uwsgi\_pass directive is set to the name of an upstream directive block. This is typically found in the same file. The upstream block then describes the tcp or unix socket that is to be used for wsgi protocol communication between nginx and uwsgi. As an example, here is the ‘as of this writing’ server conf file for customer\_gateway, customer\_gateway.conf:

upstream wsgi\_server\_location{

server unix://home/nginx\_customer\_gateway\_mediary/socket;

# server 127.0.0.1:8001;

}

server {

listen 8080;

server\_name 35.194.71.194;

charset utf-8;

client\_max\_body\_size 75M;

location / {

uwsgi\_pass wsgi\_server\_location;

include uwsgi\_params;

}

}

Note that the name of the directive is ‘uwsgi\_pass’, yet any wsgi protocol conforming server may be used. Perhaps this directive should have been called ‘wsgi\_pass’ without the ‘u’.

Note the include line for uwsgi\_parms. This included file contains nginx ‘magic’ variable assignments for tuning the communication between uwsgi. <https://uwsgi-docs.readthedocs.io/en/latest/Vars.html>

Perhaps they are magic because it is so hard to find documentation on what they do. There is an ‘example’ uwsgi\_params file in the /etc/nginx directory. Many of the examples I see just include this directly, as can be seen in the example just above. Since /etc/nginx.conf is also in /etc/nginx, and other conf files were included into it, there was no need to add a path to uwsgi\_params for include to find it..

## Spawning multiple uWSGI processes to meet load demands

There is a program, the Emperor, that launches multiple uWSGI processes as needed. Apparently any high bandwidth, i.e. enterprise public, site will need this. <https://uwsgi-docs.readthedocs.io/en/latest/Emperor.html>

The systemd configuration link mentioned above is for setting up the ‘emperor’.