django-uwsgi Documentation

Release 0.2.0

Eugene Mechanism

November 10, 2016
## Contents

1 Features 3

2 Installation 5

3 Configuration 7
  3.1 Adding django-uwsgi to your project 7

4 Decorators 9
  4.1 Notes 9
  4.2 Example: a Django session cleaner and video encoder 9
  4.3 django_uwsgi.decorators API reference 10

5 Email Backend 15
  5.1 Usage 15
  5.2 Note 15
  5.3 Changing the backend 15
  5.4 django-configurations 16

6 Cache Backend 17
  6.1 Installation 17
  6.2 django-confy 17
  6.3 Settings 17

7 Management Command 19
  7.1 runuwsgi 19
  7.2 runuwsgi options: 19
  7.3 http 19
  7.4 socket 19
  7.5 Other options 19

8 Emperor 21

9 Integrations 23
  9.1 Django-Debug-Toolbar 23
  9.2 Wagtail 23

10 Screenshots 27

11 Todo 29
You can view the code of this project or fork it (please, send pull requests), at Github.
Features

- Admin page with uWSGI stats (options to reload/stop uWSGI, clear uWSGI cache)
- uWSGI Cache Backend for Django
- uWSGI Email Backend for Django (send emails via uWSGI’s spooler)
- Debug Panel for django-debug-toolbar (offers same functions as admin page)
- Django template loader for embedded into uWSGI files
- Django Management Command runuwsgi (with live autoreload when DEBUG is True)
- uWSGI config generator
- Django CBV Mixins based on uWSGI decorators
- Forms to send log messages and signals to uWSGI via admin interface

Some features are not added into repo or not yet implemented (See Todo)
django-uwsgi is easy installable via pip:
```
pip install django-uwsgi
```

or clone it from github:
```
git clone https://github.com/unbit/django-uwsgi.git
cd django-uwsgi
pip install .
```

# or for development
```
pip install -e .
```

By default `django-uwsgi` doesn't installed with uWSGI as requirement. And here are a few known reasons why:

- Django project installed into virtualenv and running in Emperor mode. In this case uWSGI is installed system-wide or into some other virtualenv.
- Some devs love to use system package managers like apt and prefer to install uwsgi other way.
- you need to build uWSGI with custom profile ex: `UWSGI_PROFILE=gevent pip install uwsgi`

You can install django-uwsgi with uWSGI by appending `[uwsgi]` to the install command:
```
pip install django-uwsgi[uwsgi]
```
3.1 Adding django-uwsgi to your project

Add ‘django_uwsgi’, to your INSTALLED_APPS in settings.py:

INSTALLED_APPS += ['django_uwsgi',]

Add django_uwsgi into urls.py:

urlpatterns += [url(r'^admin/uwsgi/', include('django_uwsgi.urls')),]
Chapter 4

Decorators

The uWSGI API is very low-level, as it must be language-independent.

That said, being too low-level is not a Good Thing for many languages, such as Python.

Decorators are, in our humble opinion, one of the more kick-ass features of Python, so in the uWSGI source tree you will find a module exporting a bunch of decorators that cover a good part of the uWSGI API.

4.1 Notes

Signal-based decorators execute the signal handler in the first available worker. If you have enabled the spooler you can execute the signal handlers in it, leaving workers free to manage normal requests. Simply pass `target='spooler'` to the decorator.

```
@timer(3, target='spooler')
def hello(signum):
    print("hello")
```

4.2 Example: a Django session cleaner and video encoder

Let’s define a `tasks.py` module and put it in the Django project directory.

```
import os
from django.contrib.sessions.models import Session
from django_uwsgi.decorators import cron, spool

@cron(40, 2, -1, -1, -1)
def clear_django_session(num):
    print("it’s 2:40 in the morning: clearing django sessions")
    Session.objects.all().delete()

@spool
def encode_video(arguments):
    os.system("ffmpeg -i \"%s\" image%%d.jpg" % arguments['filename'])
```

The session cleaner will be executed every day at 2:40, to enqueue a video encoding we simply need to spool it from somewhere else.
from tasks import encode_video

def index(request):
    # launching video encoding
    encode_video.spool(filename=request.POST['video_filename'])
    return render_to_response('enqueued.html')

Now run uWSGI with the spooler enabled:

[uwsgi]
; a couple of placeholder
django_projects_dir = /var/www/apps
my_project = foobar
; chdir to app project dir and set pythonpath
chdir = %(django_projects_dir)/%(my_project)
pythonpath = %(django_projects_dir)
; load django
module = django.core.handlers:WSGIHandler()
env = DJANGO_SETTINGS_MODULE=%(my_project).settings
; enable master
master = true
; 4 processes should be enough
processes = 4
; enable the spooler (the mytasks dir must exist!)
spooler = %(chdir)/mytasks
; load the task.py module
import = task
; bind on a tcp socket
socket = 127.0.0.1:3031

The only especially relevant option is the import one. It works in the same way as module but skips the WSGI callable search. You can use it to preload modules before the loading of WSGI apps. You can specify an unlimited number of "import" directives.

4.3 django_uwsgi.decorators API reference

django_uwsgi.decorators.postfork(func)

uWSGI is a preforking (or “fork-abusing”) server, so you might need to execute a fixup task after each fork(). The postfork decorator is just the ticket. You can declare multiple postfork tasks. Each decorated function will be executed in sequence after each fork().

@postfork
def reconnect_to_db():
    myfoodb.connect()

@postfork
def hello_world():
    print("Hello World")

django_uwsgi.decorators.spool(func)

The uWSGI spooler can be very useful. Compared to Celery or other queues it is very “raw”. The spool decorator will help!

@spool
def a_long_long_task(arguments):
    print(arguments)
for i in xrange(0, 1000000):
    time.sleep(0.1)

@spool
def a_longer_task(args):
    print(args)
    for i in xrange(0, 1000000):
        time.sleep(0.5)

# enqueue the tasks
a_long_long_task.spool(foo='bar', hello='world')
a_longer_task.spool({'pippo': 'pluto'})

The functions will automatically return uwsgi.SPOOL_OK so they will be executed one time independently by their return status.

django_uwsgi.decorators.spoolforever(func)
Use spoolforever when you want to continuously execute a spool task. A @spoolforever task will always return uwsgi.SPOOL_RETRY.

@spoolforever
def a_longer_task(args):
    print(args)
    for i in xrange(0, 1000000):
        time.sleep(0.5)

# enqueue the task
a_longer_task.spool({'pippo': 'pluto'})

django_uwsgi.decorators.spoolraw(func)
Advanced users may want to control the return value of a task.

@spoolraw
def a_controlled_task(args):
    if args['foo'] == 'bar':
        return uwsgi.SPOOL_OK
    return uwsgi.SPOOL_RETRY

a_controlled_task.spool(foo='bar')

django_uwsgi.decorators.rpc("name", func)
uWSGI RPC is the fastest way to remotely call functions in applications hosted in uWSGI instances. You can easily define exported functions with the @rpc decorator.

@rpc('helloworld')
def ciao_mondo_function():
    return "Hello World"

django_uwsgi.decorators.signal(num)(func)
You can register signals for the signal framework in one shot.

@signal(17)
def my_signal(num):
    print("i am signal %d" % num)

django_uwsgi.decorators.timer(interval, func)
Execute a function at regular intervals.

4.3. django_uwsgi.decorators API reference
@timer

def three_seconds(num):
    print("3 seconds elapsed")

django_uwsgi.decorators.rbtimer(interval, func)
    Works like @timer but using red black timers.

django_uwsgi.decorators.cron(min, hour, day, mon, wday, func)
    Easily register functions for the CronInterface.
    @cron(59, 3, -1, -1, -1)
    def execute_me_at_three_and_fiftynine(num):
        print("it's 3:59 in the morning")

Since 1.2, a new syntax is supported to simulate crontab-like intervals (every Nth minute, etc.). */5 * * * * can be specified in uWSGI like thus:

@cron(-5, -1, -1, -1, -1)
def execute_me_every_five_min(num):
    print("5 minutes, what a long time!")

django_uwsgi.decorators.filemon(path, func)
    Execute a function every time a file/directory is modified.
    @filemon(\"/tmp\")
    def tmp_has_been_modified(num):
        print("/tmp directory has been modified. Great magic is afoot")

django_uwsgi.decorators.erlang(process_name, func)
    @erlang(\'foobar\')
def hello():
    return "Hello"

django_uwsgi.decorators.thread(func)
    Mark function to be executed in a separate thread.

Important: Threading must be enabled in uWSGI with the enable-threads or threads <n> option.

@thread
def a_running_thread():
    while True:
        time.sleep(2)
        print("i am a no-args thread")

@thread
def a_running_thread_with_args(who):
    while True:
        time.sleep(2)
        print("Hello $s (from arged-thread)" % who)

a_running_thread()
a_running_thread_with_args("uWSGI")

You may also combine @thread with @postfork to spawn the postfork handler in a new thread in the freshly spawned worker.
@@postfork@@
@@thread@@
def a_post_fork_thread():
    while True:
        time.sleep(3)
        print("Hello from a thread in worker \d" % uwsgi.worker_id())

django_uwsgi.decorators.lock(func)
    This decorator will execute a function in fully locked environment, making it impossible for other workers
    or threads (or the master, if you’re foolish or brave enough) to run it simultaneously. Obviously this may be
    combined with @postfork.

@lock
def dangerous_op():
    print("Concurrency is for fools!")

django_uwsgi.decorators.mulefunc([mulespec],func)
    Offload the execution of the function to a mule <http://uwsgi-docs.readthedocs.org/en/latest/Mules.html>. When
    the offloaded function is called, it will return immediately and execution is delegated to a mule.

@mulefunc
def i_am_an_offloaded_function(argument1, argument2):
    print argument1,argument2
    You may also specify a mule ID or mule farm to run the function on. Please remember to register your function
    with a uwsgi import configuration option.

@mulefunc(3)
def on_three():
    print "I’m running on mule 3."

@mulefunc('old_mcdonalds_farm')
def on_mcd():
    print "I’m running on a mule on Old McDonalds' farm."

django_uwsgi.decorators.harakiri(time,func)
    Starting from uWSGI 1.3-dev, a customizable secondary harakiri subsystem has been added. You can use this
    decorator to kill a worker if the given call is taking too long.

@harakiri(10)
def slow_function(foo, bar):
    for i in range(0, 10000):
        for y in range(0, 10000):
            pass
            # or the alternative lower level api
uwsgi.set_user_harakiri(30) # you have 30 seconds. fight!
slow_func()
uwsgi.set_user_harakiri(0) # clear the timer, all is well
Email Backend

A Django backend for e-mail delivery using uWSGI Spool to queue deliveries.

### 5.1 Usage

First, add uWSGI backend in your settings file.

```python
EMAIL_BACKEND = 'django_uwsgi.mail.EmailBackend'
```

And send your e-mails normally.

```python
from django.core.mail import send_mail

send_mail('Subject here', 'Here is the message.', 'from@example.com', ['to@example.com'], fail_silently=False)
```

### 5.2 Note

You must setup uwsgi spooler. Example ini:

```ini
plugin = spooler
spooler = /tmp
spooler-import = django_uwsgi.tasks
```

or use built in management command `runuwsgi`

### 5.3 Changing the backend

By default the `django.core.mail.backends.smtp.EmailBackend` is used for the real e-mail delivery. You can change that using:

```python
UWSGI_EMAIL_BACKEND = 'your.backend.EmailBackend'
```
5.4 django-configurations

If you’re using django-configurations in your project, you must setup importer as mentioned in django-configurations docs for celery
6.1 Installation

change settings to:

```python
CACHES = {
    'default': {
        'BACKEND': 'django_uwsgi.cache.UwsgiCache',
        # and optionally, if you used a different cache name
        'LOCATION': 'foobar'
    }
}
```

6.2 django-confy

if you're using django-confy, you can use url like:

```
CACHE_URL=uwsgi://foobar
```

6.3 Settings

UWSGI_CACHE_FALLBACK

- **False** - raises Exception if uwsgi cannot be imported.
- **True** (default) - if uwsgi is not importable this cache backend will alias to LocMemCache. Note that south or other management commands might try to load the cache backend so this is why it’s the default.
Management Command

7.1 runuwsgi

```
python manage.py runuwsgi
```

7.2 runuwsgi options:

7.3 http

```
python manage.py runuwsgi http=8000
```

7.4 socket

```
python manage.py runuwsgi socket=/tmp/projectname-uwsgi.sock
```

7.5 Other options

Any other options can be passed via environment variables, prefixed with `UWSGI_`
you can use `django_uwsgi.emperor` module if you want to store vassals configs in PostgreSQL database. Simply add `django_uwsgi.emperor`, into `INSTALLED_APPS`

```
INSTALLED_APPS += ['django_uwsgi.emperor',]
```

Populate vassals via django admin interface and start uwsgi with command like:

```
uwsgi --plugin emperor_pg --emperor "pg://host=127.0.0.1 user=foobar dbname=emperor;SELECT name,config,ts FROM vassals WHERE enabled = True"
```

Each time vassal added, removed, updated, enabled or disabled - uwsgi will start/stop it or reload.
9.1 Django-Debug-Toolbar

If you’re using django-debug-toolbar, you can add:

```python
DEBUG_TOOLBAR_PANELS += ['django_uwsgi.panels.UwsgiPanel',]
```

9.2 Wagtail

If you’re using Wagtail:

There is wagtail_hooks.py file available and Wagtail will read it automatically.

And you don’t have to add django_uwsgi into urls.py.

Wagtail admin interface:
# Chapter 9. Integrations

## Django-uwsgi Documentation, Release 0.2.0

### UWSGI Status

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>loop</td>
<td>None</td>
</tr>
<tr>
<td>masterpid</td>
<td>32169</td>
</tr>
<tr>
<td>started_on</td>
<td>Nov 19, 2016, 10:03 a.m.</td>
</tr>
<tr>
<td>now</td>
<td>Nov 19, 2016, 10:03 a.m.</td>
</tr>
<tr>
<td>buffer_size</td>
<td>65535</td>
</tr>
<tr>
<td>total requests</td>
<td>25</td>
</tr>
<tr>
<td>numproc</td>
<td>4</td>
</tr>
<tr>
<td>cores</td>
<td>1</td>
</tr>
<tr>
<td>cwd</td>
<td>/home/linus/MYPROJECTs/mom</td>
</tr>
</tbody>
</table>
### 9.2. Wagtail

<table>
<thead>
<tr>
<th>Extension</th>
<th>Updated</th>
<th>Created At</th>
<th>Enabled</th>
<th>Link Template</th>
</tr>
</thead>
<tbody>
<tr>
<td>txt</td>
<td>Oct 24, 2016, 1:53 am</td>
<td>Oct 24, 2016, 1:40 am</td>
<td>✔️</td>
<td>14717436563.21471</td>
</tr>
</tbody>
</table>

Page 1 of 1.
CHAPTER 10

Screenshots

django-debug-toolbar panel

Wagtail admin interface:
Emperor’s Vassal Admin Panel
django.contrib.admin interface
Todo

• Tests
• uWSGI config generator
• Improve Docs
• Translations?
• Good cache panel
• Ability to add cronjobs/filemonitors via admin interface
• Options for sendfile if uwsgi serving files

Some code is borrowed from projects I did earlier and some code is still not added yet, but does exists in my projects.
Changelog
CHAPTER 13

Contributing

13.1 Code Style
13.2 Docs
13.3 Tests
CHAPTER 14

Indices and tables

- genindex
- modindex
- search
d

django_uwsgi.decorators, 10
Index

C
cron() (in module django_uwsgi.decorators), 12

D
django_uwsgi.decorators (module), 10

E
erlang() (in module django_uwsgi.decorators), 12

F
filemon() (in module django_uwsgi.decorators), 12

H
harakiri() (in module django_uwsgi.decorators), 13

L
lock() (in module django_uwsgi.decorators), 13

M
mulefunc() (in module django_uwsgi.decorators), 13

P
postfork() (in module django_uwsgi.decorators), 10

R
rbtimer() (in module django_uwsgi.decorators), 12
rpc() (in module django_uwsgi.decorators), 11

S
signal() (in module django_uwsgi.decorators), 11
spool() (in module django_uwsgi.decorators), 10
spoolforever() (in module django_uwsgi.decorators), 11
spoolraw() (in module django_uwsgi.decorators), 11

T
thread() (in module django_uwsgi.decorators), 12
timer() (in module django_uwsgi.decorators), 11