

Apuntes web2py

Sergio Alvariño salvari@gmail.com

abril-2017

Resumen

Apuntes de web2py
Solo para referencia rápida y personal.

Índice general

1	Web2py	2
1.1	Instalación de web2py	2
1.1.1	Instalación standalone	2
1.1.2	Desplegar web2py con Nginx	3
1.1.3	Desplegar un repo de desarrollo con un web2py dedicado	7
1.2	Preparando una aplicación	7
1.2.1	Ejemplo de contenido del fichero <i>db_custom.py</i>	9
1.3	DAL	9
1.3.1	Union de tablas db.executesql	9
1.4	web2py y git	10
1.5	web2py y d3.js	12
2	Varios	13
2.1	executeSql	13
3	Tutoriales en la red	14
3.1	Killer Web Development por Marco Laspe	14
3.1.1	Cambios	14
4	Python	14
4.1	Decorators	14

1 Web2py

Toda la info [aquí](#)

1.1 Instalación de web2py

1.1.1 Instalación standalone

Bajamos el programa de la [web de Web2py](#)

Descomprimos el framework,

Preparamos los certificados

```
openssl genrsa -out server.key 2048
openssl req -new -key server.key -out server.csr
```

```
Country Name (2 letter code) [AU]:ES
State or Province Name (full name) [Some-State]:CORUNA
Locality Name (eg, city) []:CORUNA
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Vodafone
Organizational Unit Name (eg, section) []:TNO
Common Name (e.g. server FQDN or YOUR name) []:txfinder
Email Address []:sergio.alvarino@vodafone.com
```

```
Please enter the following 'extra' attributes
to be sent with your certificate request
```

```
A challenge password []:secret1t05
An optional company name []:Vodafone Spain
```

Ejecutamos:

```
openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
```

Ahora deberíamos tener los ficheros `server.key`, `server.csr` y `server.crt` en el directorio raíz de *web2py*, una vez generados estos ficheros tenemos que arrancar el servidor con los siguientes parámetros:

```
python web2py.py -a 'admin_password' -c server.crt -k server.key -i 0.0.0.0 -p 8000
```

Y ya podemos acceder nuestro server en la dirección `https://localhost:8000`

1.1.2 Desplegar web2py con Nginx

1.1.2.1 Instalación de *web2py*

Vamos a instalar web2py en */var/www* a nivel global. Será nuestro web2py para servir aplicaciones en producción.¹

```
cd
mkdir tmp
cd tmp
wget https://mdipierro.pythonanywhere.com/examples/static/web2py_src.zip
unzip web2py_src.zip
mv web2py /var/www
cd /var/www
chown -R www-data:www-data web2py/
```

Vamos a probar el *web2py*

Primero generamos una clave para tener acceso al interfaz de administración:

```
cd /var/www/web2py
openssl req -x509 -new -newkey rsa:4096 -days 3652 -nodes -keyout myweb2py.key -
out myweb2py.crt
.
.
.
Country Name (2 letter code) [AU]:ES
State or Province Name (full name) [Some-State]:Coruna
Locality Name (eg, city) []:A Coruna
Organization Name (eg, company) [Internet Widgits Pty Ltd]:s0j0
Organizational Unit Name (eg, section) []:Development
Common Name (e.g. server FQDN or YOUR name) []:51.255.37.47
Email Address []:salvari@gmail.com
```

Abrimos un puerto de desarrollo en el servidor (voy a abrir el 8080):

```
ufw allow 8080/tcp
```

Arrancamos el *web2py*:

```
python web2py.py -k myweb2py.key -c myweb2py.crt -i 0.0.0.0 -p 8080
```

Y visitamos en el navegador la dirección de nuestro servidor [<https://vps223560.ovh.net:8080>]

¹ Mejor comprobar donde está la última versión del fichero *sources* de web2py para hacer el wget

Si vemos un warning de que nuestro server no es seguro todo va bien. Nuestro navegador nos avisa por qué el certificado no está firmado por una CA que el conozca. Le decimos que siga y veremos la página de web2py.

Ya podemos parar el *web2py* que hemos arrancado con un **Ctrl+C**

1.1.2.2 Instalación de uWSGI

Lo vamos a instalar globalmente, hay que asegurarse de que tenemos instalados:

```
apt install python-pip python-dev python3-dev python-setuptools
apt install build-essential
```

Podemos instalar *uWSGI* desde los repos de debian o mediante pip. Lo vamos a hacer con pip.

Y ahora instalamos *uWSGI* sin más que:

```
pip install wheel
pip install uwsgi
```

Por alguna razón nos falla. El *pip* deja el *uwsgi* instalado en */usr/local/bin* pero cuando lo ejecutamos nos responde que no existe el fichero */usr/bin/uwsgi*. Para salir del paso he hecho:

```
cd /usr/bin/
ln -s /usr/local/bin/uwsgi .
```

The uWSGI application container server interfaces with Python applications using the WSGI interface specification. The web2py framework includes a file designed to provide this interface within its handlers directory. To use the file, we need to move it out of the directory and into the main project directory:

```
cd /var/www/web2py
mv handlers/wsgihandler.py .
```

Hacemos una comprobación rápida de que *uWSGI* puede servir peticiones con:

```
uwsgi --http :8080 --chdir /var/www/web2py -w wsgihandler:application
```

Podremos visitar nuestro *web2py* en la dirección [http://vps223560.ovh.net:8080] ¡Ojo! Sin *SSL*.

Vale, una vez probado que todo funciona vamos a dejar configurado el servicio *uWSGI* en *systemd*.

Creamos un fichero */lib/systemd/system/uwsgi.service* que contenga:

```
[Unit]
Description=uWSGI Emperor
```

```
After=syslog.target
```

```
[Service]
```

```
ExecStart=/usr/local/bin/uwsgi --ini /etc/uwsgi/emperor.ini
```

```
Restart=always
```

```
KillSignal=SIGQUIT
```

```
Type=notify
```

```
StandardError=syslog
```

```
NotifyAccess=all
```

```
[Install]
```

```
WantedBy=multi-user.target
```

Tenemos que crear también el correspondiente fichero `/etc/uwsgi/emperor.ini` con el siguiente contenido:

```
[uwsgi]
```

```
emperor = /etc/uwsgi/apps-enabled
```

```
uid = www-data
```

```
gid = www-data
```

```
limit-as = 1024
```

```
logto = /tmp/uwsgi.log
```

Tenemos que crear un fichero de configuración para el “*vassal*”² correspondiente a *web2py*, el fichero `/etc/uwsgi/apps-available/web2py.ini`, con el contenido siguiente:

```
[uwsgi]
```

```
chdir = /var/www/web2py
```

```
module = wsgihandler:application
```

```
master = true
```

```
processes = 4
```

```
socket = /var/www/web2py/web2pyUwsgi.sock
```

```
chmod-socket = 660
```

```
vacuum = true
```

Basicamente le estamos diciendo al *uWSGI*:

- En que directorio y cual es el fichero de *handler* de nuestra aplicación.
- Que lance cuatro procesos (¿conviene mirar el número de cpu del server?)
- Que se comunique con *Nginx* a través de un *socket*

²Ver doc de uWSGI

- Cambiamos las propiedades del *socket*
- Y con la opción *vacuum* le decimos que limpie el *socket* cuando el proceso termine

Probamos el servicio que hemos configurado:

```
systemctl start uwsgi
systemctl uwsgi status uwsgi
```

Nota: Falló por que había un fichero en `/etc/init.d/uwsgi`. He movido este fichero a otra localización y ya funciona, después de ejecutar `systemctl daemon-reload`

Una vez que vemos que funciona, lo paramos con `systemctl stop uwsgi`

Enlazamos el fichero del *vassal* de *web2py* en el directorio `/etc/uwsgi/apps-enabled`:

```
cd /etc/uwsgi/apps-enabled
ln -s /etc/uwsgi/apps-available/web2py.ini
```

Y ahora dejamos *uWSGI* habilitado como servicio y arrancado:

```
systemctl enable uwsgi
systemctl start uwsgi
```

1.1.2.3 Nginx

Copiamos el certificado y la clave SSL al directorio `/etc/nginx/ssl`:

```
mkdir /etc/nginx/ssl
mv /var/www/web2py/myweb2py.crt /etc/nginx/ssl
mv /var/www/web2py/myweb2py.key /etc/nginx/ssl
```

Creamos un fichero `/etc/nginx/sites-available/web2py` con el siguiente contenido:

```
server {
    listen 80;
    listen [::]:80;

    server_name vps223560.ovh.net;

    root /var/www/html;
    index index.html;

    location ~* /(\w+)/static/ {
        root /var/www/web2py/applications/;
    }
    location / {
```

```

    include uwsgi_params;
    uwsgi_pass unix:/var/www/web2py/web2pyUwsgi.sock
}
}

server {
    listen 443;
    server_name vps223560.ovh.net;

    root html;
    index index.html index.htm;

    ssl on;
    ssl_certificate /etc/nginx/ssl/myweb2py.crt;
    ssl_certificate_key /etc/nginx/ssl/myweb2py.key;

    ssl_session_timeout 5m;

    #ssl_protocols SSLv3 TLSv1 TLSv1.1 TLSv1.2;
    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    ssl_ciphers "HIGH:!aNULL:!MD5 or HIGH:!aNULL:!MD5:!3DES";
    ssl_prefer_server_ciphers on;

    location / {
        include uwsgi_params;
        uwsgi_pass unix:/var/www/web2py/myweb2pyUwsgi.sock;
    }
}

```

1.1.3 Desplegar un repo de desarrollo con un web2py dedicado

rollo

1.2 Preparando una aplicación

1. Crea una aplicación desde el interfaz de administración, en nuestro caso la llamaremos **pyfinder**
2. Para usar *MySQL* como motor de base de datos: Editamos el fichero *applica-*

tions/alloer/private/appconfig.ini, tenemos que poner el uri que apunta a nuestra base de datos, sustituyendo *dbUser*, *dbPass* y *dbName* por valores reales.

```
; App configuration
[app]
name          = PyFinder
author        = Sergio Alvariño <sergio.alvarino@vodafone.com>
description    = TxFinder en Web2Py
keywords      = Thope, TxFinder, web2py, python, framework
generator     = Web2py Web Framework

; Host configuration
[host]
names = localhost:*, 127.0.0.1:*, *:*

; db configuration
[db]
; uri          = sqlite://storage.sqlite
uri           = mysql://dbUser:dbPass@localhost/dbName

migrate       = true
pool_size     = 10 ; ignored for sqlite

; smtp address and credentials
[smtp]
server        = smtp.gmail.com:587
sender        = salvari@gmail.com
login         = username:password
tls           = true
ssl           = true

; form styling
[forms]
formstyle     = bootstrap3_inline
separator     =
```

3. Editamos el fichero *applications/alloer/models/db.py* Tenemos que asegurarnos de editar esta sección para que no nos de problemas con palabras reservadas:

```
db = DAL(myconf.get('db.uri'),
         pool_size=myconf.get('db.pool_size'),
         migrate_enabled=myconf.get('db.migrate'),
```



```

        check_reserved=['mysql'])
#         check_reserved=['all'])

```

4. Creamos un fichero *db_custom.py* en el directorio: *applications/alloer/models* El fichero tiene que ser parecido al que figura a continuación.

IMPORTANTE: en cada tabla crear el campo *id* de tipo *integer*, es para uso interno de *web2py*

IMPORTANTE: especificar *migrate* *FALSE* al final en todas las tablas externas

1.2.1 Ejemplo de contenido del fichero *db_custom.py*

```

db.define_table('afoxtfo',
    Field('id', 'integer'),
    Field('opti_of_connection_id' , 'string'),
    Field('afo' , 'string'),
    Field('afo_fiber' , 'string'),
    Field('opti_cable_id' , 'string'),
    Field('tfo' , 'string'),
    Field('tfo_fiber' , 'string'),
    Field('cable_endpoint' , 'string'),
    Field('side' , 'string'),
    Field('state' , 'string'),
    Field('loaddate' , 'string'),
    migrate = False);

```

1.3 DAL

1.3.1 Union de tablas *db.executesql*

El código de abajo funciona correctamente, el método *executesql* necesita que le pasemos una referencia a un campo real de la base de datos, no he sido capaz de hacerlo funcionar de ninguna otra forma.

```

sql = """
SELECT origination_node AS node
FROM site s,
      segment g
WHERE s.id = {0}
      AND s.site_name = g.origination_site_name

```

```

UNION
SELECT destination_node AS node
  FROM site s,
       segment g
 WHERE s.id = {0}
       AND s.site_name = g.destination_site_name"".format(myid)

```

```
coloc_nodes = db.executesql(sql, fields=[db.segment.origination_node], colnames=['node'])
```

Ejemplos, sacados de Google code:

```

db.define_table('person', Field('name'), Field('email'))
db.define_table('dog', Field('name'), Field('owner', 'reference person'))
db.executesql([SQL code returning person.name and dog.name fields], fields=[db.person.name, db.dog.name])
db.executesql([SQL code returning all fields from db.person], fields=db.person)
db.executesql([SQL code returning all fields from both tables], fields=[db.person, db.dog])
db.executesql([SQL code returning person.name and all db.dog fields], fields=[db.person.name, db.dog])

```

1.4 web2py y git

Hay varias formas de hacer el desarrollo. Yo he optado por tener un web2py

Es importante tener el siguiente fichero *.gitignore* en el directorio *web2py*

```

.*
!.gitignore
*.pyc
*.pyo
*~
#*
*.1
*.bak
*.bak2
*.svn
*.w2p
*.class
*.rej
*.orig
Thumbs.db
.DS_Store
*.DS_Store
# index.yaml

```

```
# routes.py
# logging.conf
# gluon/tests/VERSION
# gluon/tests/sql.log
httpserver.log
httpserver.pid
parameters*.py
# ./deposit
# ./benchmark
# ./build
# ./dist*
# ./dummy_tests
# ./optional_contrib
# ./ssl
# ./docs
# ./logs
# /*.zip
# ./gluon/*.1
# ./gluon/*.txt
# ./admin.w2p
# ./examples.w2p
CHANGELOG
LICENSE
MANIFEST.in
README.markdown
VERSION
anyserver.py
web2py.py
examples/*
handlers/*
extras/*
gluon/*
scripts/*
site-packages/*
applications/welcome
applications/examples
applications/admin
applications/*/databases/*
applications/*/sessions/*
applications/*/errors/*
```

```

applications/*/cache/*
applications/*/uploads/*
applications/*/private/*
applications/*/.*.py[oc]
applications/*/static/temp
applications/*/progress.log
# applications/examples/static/epydoc
# applications/examples/static/sphinx
# applications/admin/cron/cron.master
# HOWTO-web2py-devel
# logs/
# cron.master

```

1.5 web2py y d3.js

1. Created a new app with the wizard (default layout, name: TestD3). Views: index,error and visualizations where I want to have the d3 stuff.
2. Put the d3 javascript file in static/js
3. In View TestD3/views/default/visualizations.html:

```

{{response.files.append(URL(r=request,c='static',f='/js/d3.js'))}}
{{extend 'layout.html'}}

```

```

<p>Here we would like to have some d3.js plots</p>
<script type="text/javascript">

```

```

    d3.select('body').append('svg').append('circle').style("stroke", "gray").style("fi
</script>

```

This produced a red circle but the circle below the copyright 2013 – powered by web2py line. Of course I have to select properly because I want to have the circle inside:

WRITE THE APPROPRIATE CONTROLLER I ll take some time to dive into web2py and then I will post whatever worked with d3.

(<https://localhost:8000/testD3/default/visualizations>)

More on this subject <https://groups.google.com/forum/#!msg/web2py/lngBXrQlh1g/DqEmW8FkkoEJ>

A nice one: <https://stackoverflow.com/questions/34326343/embedding-d3-js-graph-in-a-web2py-bootstrap-page>

https://github.com/willimoa/welcome_d3

Really important: https://github.com/monotasker/plugin_d3

Interesting: <https://realpython.com/blog/python/web-development-with-flask-fetching-data-with-requests/> <http://grokbase.com/t/gg/d3-js/14425gneaf/web2py-d3-json-where-should-i-put-the-json-file> <http://www.web2pyslices.com/slice/show/1689/animations-of-svg-images-and-paths-with-d3js>

Pure D3 <https://github.com/d3/d3/wiki/tutorials> <http://alignedleft.com/tutorials/d3>
<http://gcoch.github.io/D3-tutorial/> <https://bl.ocks.org/mbostock/3883245> <https://bl.ocks.org/basilesimon/29efb0>
<https://bl.ocks.org/d3noob/402dd382a51a4f6eea487f9a35566de0> <https://leanpub.com/D3-Tips-and-Tricks/read#leanpub-auto-crossfilter-dcjs-and-d3js-for-data-discovery> <http://bl.ocks.org/cpbotha/507371>

Google charts

<http://www.web2pyslices.com/slice/show/1721/google-charts-plugin>

2 Varios

2.1 executeSql

Hay que estudiar esto con calma y aprender a usarlo con precisión:

```
coloc_node_orig = db( (db.site.id == myid)
                      & (db.site.site_name == db.segment.origination_site_name)
                      ).select(db.segment.origination_node.with_alias('node'),
                              distinct=True)
coloc_node_dest = db( (db.site.id == myid)
                      & (db.site.site_name == db.segment.destination_site_name)
                      ).select(db.segment.destination_node.with_alias('node'),
                              distinct=True)
```

```
sql = """
SELECT origination_node AS node
  FROM site s,
       segment g
 WHERE s.id = {0}
       AND s.site_name = g.origination_site_name
UNION
SELECT destination_node AS node
  FROM site s,
```

```

        segment g
WHERE s.id = {0}
      AND s.site_name = g.destination_site_name"".format(myid)

coloc_nodes = db.executesql(sql, fields=[db.segment.origination_node], colnames=['node'], as_dict=False)

```

3 Tutoriales en la red

3.1 Killer Web Development por Marco Laspe

Disponible [aquí](#)

3.1.1 Cambios

Crud debe ser importado explícitamente:

```

from gluon.tools import Crud
crud = Crud(db)

```

Después ya podemos seguir el tutorial haciendo por ejemplo:

```

def entry_post():
    """returns a form where the user can entry a post"""
    form = crud.create(db.post)
    return dict(form=form)

```

4 Python

4.1 Decorators

Decorators añadidos desde Python 2.4 para permitir que el *function and method wrapping* fuese más fácil de leer y entender. *function and method wrapping* consiste en implementar una función (o método) que recibe como parámetro una función (¿o método?) y devuelve una función mejorada.

El caso de uso original era definir los métodos como métodos de Clase o métodos estáticos en la cabecera de su definición.

La receta general:

```
def mydecorator(function):  
    def _mydecorator(*args, **kw):  
        # do some stuff before the real  
        # function gets called  
        res = function(*args, **kw)  
        # do some stuff after  
        return res  
    # returns the sub-function  
    return _mydecorator
```

El intérprete carga los *decorators* cuando se lee el módulo la primera vez, debe limitarse su uso a *wrappers* que puedan aplicarse de forma genérica. Si el *decorator* está fuertemente acoplado con la clase o función que decora debería reescribirse y convertirlo en un invocable regular para evitar la complejidad.

Patrones típicos:

- Argument checking
- Caching
- Proxy
- Context provider

Nota: En web2py parece que los *decorators* se usan típicamente como proveedores de contexto. Hay que ver como funciona la sentencia `with` de Python 2.5 que se crea con el mismo propósito.