Apuntes web2py

Sergio Alvariño salvari@gmail.com

abril-2017

Resumen

Apuntes de web2py
Solo para referencia rápida y personal.

Índice general

1 Web2py

Toda la info aquí

1.1 Instalación de web2py

1.1.1 Instalación standalone

Bajamos el programa de la web de Web2py

Descomprimimos 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 raiz 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]:sOjO
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 8000
```

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

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 **Crl+C**

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

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 inteface 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/wwww/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*
- 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
```

[2] Ver doc de uWSGI

Nota: Falló por que había un fichero en /etc/init.d/uwsgi. He movido este fichero a otra localización y ya funciona, depués 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 DAL

1.2.1 Union de tablas

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.nam
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])
```

2 Tutoriales en la red

2.1 Killer Web Development por Marco Laspe

Disponible aquí

2.1.1 Cambios

`Crud` debe ser importado explícitamente:

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

Despues 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)
```

3 Python

3.1 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.

Patrónes típicos:

- Argument checking
- Caching
- Proxy
- Context provider

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