Apuntes de web2py

Sergio Alvariño [salvari@gmail.com](mailto:salvari@gmail.com)

Agosto-2019

Apuntes de web2py: Un framework para desarrollo de aplicaciones web

# Introducción

[web2py](http://www.web2py.com/) es un framework para facilitar el desarrollo de aplicaciones web escrito en Python.

***web2py*** funciona correctamente en Python 3. Su curva de aprendizaje no es tan empinada como la de Django y en muchos sentidos es más moderno que Django.

**web2py** está basado (no estrictamente) en el modelo [MVC](https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador)

**web2py** incorpora *Bootstrap 4*

## Referencias

* [Evolución del modelo MVC](https://martinfowler.com/eaaDev/uiArchs.html)
* [Fat models and thin controllers](https://nomadphp.com/blog/60/working-with-the-thin-controller-and-fat-model-concept-in-laravel)
* [Crítica del mantra](https://nomadphp.com/blog/60/working-with-the-thin-controller-and-fat-model-concept-in-laravel)

# Empezar rápido

## Instalación

Vamos a ver el proceso de instalación de una instancia de web2py en modo *standalone*. Normalmente uso web2py instalado de esta forma para entornos de desarrollo. Para un entorno de producción lo normal es instalar web2py tras un servidor web como [*Apache*](https://www.apache.org/) o [Nginx](https://www.nginx.com/), aunque dependiendo de la carga de trabajo y de como administres tus sistemas no tiene por que ser imprescindible y lo puedes poner en producción en modo *standalone*.

1. Creamos un entorno virtual

* Como ya hemos comentado ***web2py*** funciona ya en Python 3. Además con Python nunca está de mas encapsular nuestras pruebas y desarrollos en un entorno virtual.[[1]](#footnote-31) Así que creamos el virtualenv que llamaremos *web2py*:
* mkvirtualenv -p `which python3` web2py

1. Bajamos el programa de la web de Web2py y descomprimimos el framework:

* # creamos un directorio (cambia el path a tu gusto)  
  mkdir web2py\_test  
  cd web2py\_test  
    
  # bajamos el programa de la web y descomprimimos  
  wget https://mdipierro.pythonanywhere.com/examples/static/web2py\_src.zip  
    
  # opcionalmente borramos el zip, sería mejor guardarlo  
  # por si queremos hacer nuevas instalaciones  
  rm web2py\_src.zip

1. Generamos certificados para el protocolo *ssl*:

* Para usar con comodidad web2py conviene que nos generemos unos certificados para gestionar el ssl:
* # nos movemos al directorio de web2py  
  cd web2py  
    
  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]:A Coruna  
  Locality Name (eg, city) []:A Coruna  
  Organization Name (eg, company) [Internet Widgits Pty Ltd]:BricoLabs  
  Organizational Unit Name (eg, section) []:Division de Hackeo  
  Common Name (e.g. server FQDN or YOUR name) []:testServer@bricolabs.cc  
  Email Address []:contacto@bricolabs.cc  
    
  Please enter the following 'extra' attributes  
  to be sent with your certificate request  
  A challenge password []:secret1t05  
  An optional company name[]:Asociacion BricoLabs
* Y ahora ejecutamos:
* openssl x509 -req -days 365 -in server.csr \  
  -signkey server.key -out server.crt

1. Arrancamos el servidor:

* Ahora deberíamos tener los ficheros server.key, server.csr y server.crt en el directorio raiz de web2py, una vez generados estos ficheros podemos arrancar el servidor con los siguientes parámetros (recuerda activar el entorno virtual si no lo tienes activo):
* 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. Servidor de base de datos.

* Para usar ***web2py*** es imprescindible tener acceso a un servidor de base de datos. Podemos usar MySQL o MariaDB por ejemplo. Pero para empezar rápidamente vamos a tirar de [SQLite](https://www.sqlite.org/version3.html), un servidor fácil de instalar potente y versátil. Es importante usar la versión 3 que introduce grandes mejoras sobre el antiguo *SQLite*
* sudo apt install sqlite3

Y ahora si que ya tenemos todo listo para empezar a usar ***web2py***. Podemos crear nuestra primera aplicación.

### Los detalles tenebrosos

Si tienes mucha prisa por aprender web2py puedes saltarte esta sección e ir directamente a la sección [siguiente](#nuestra-primera-aplicación)

Si por el contrario quieres entender exactamente que hemos hecho para poder arrancar el ***web2py*** este puede ser el primer paso.

¿Qué es un *virtualenv*?

Python nos permite definir *virtualenv*. Un *virtualenv* es un entorno python aislado. Todos los *virtualenvs* están aislados entre si y mejor todavía son independientes del python del sistema. Esto te permite tener multiples entornos de desarrollo (o producción) cada uno con distintas versiones de python y diferentes librerias python instaladas en cada uno de ellos, o quizás diferentes versiones de las mismas librerias.

¿Que es *virtualenvwrapper*?

Es un frontend para usar *virtualenv*, la herramienta nativa de python para gestionar *virtualenvs*. Es completamente opcional, aunque a mi me parece muy cómoda.

¿Qué es todo eso de los certificados?

***web2py*** viene preparado para usar *https* (estas siglas tienen varias interpretaciones: *HTTP over TLS*, *HTTP over SSL* o *HTTP Secure*). *https* usa comunicaciones cifradas entre tu navegador y el servidor web para garantizar dos cosas: que estás accediendo al auténtico servidor y que nadie este interceptando la comunicación entre navegador y servidor.

Para usar *https* hay que hacer varias cosas:

* Generar un CSR (Certificate Signing Request)
* Obtener con ese CSR un certificado SSL de una autoridad certificadora (CA)
* O alternativamente generar nosotros un certificado a partir del CSR

Lo que hemos hecho con los comandos *openssl* ha sido:

* Generar un par de claves (privada y pública) para nuestro servidor (server.key)
* Generar con esa clave un CSR (el CSR lleva la información que le hemos metido de nuestro servidor y la clave pública)
* Generar un certificado firmándolo nosotros mismos con esa misma clave como si fueramos la autoridad certificadora.

Esto nos vale para arrancar ***web2py*** aunque nuestro navegador nos dará una alerta de riesgo de seguridad por que no reconoce a la CA.

[Más info de *openssl*](https://www.digitalocean.com/community/tutorials/openssl-essentials-working-with-ssl-certificates-private-keys-and-csrs)

## Nuestra primera aplicación

Vamos a crear nuestra primera aplicación en web2py.

Si has seguido los pasos de la [sección anterior](#instalación) ya tienes el ***web2py*** funcionando y puedes seguir cualquiera de los tutoriales que hay en la red para aprender.

En esta guía vamos a ver la creación de una aplicación paso a paso. Crearemos una aplicación de inventario para el material de la Asociación BricoLabs, empezando por una funcionalidad sencilla y añadiendo cosas según se nos ocurran.

Crea una aplicación desde el interfaz de administración, en nuestro caso la llamaremos ***cornucopia***.

Nuestro ***web2py*** nace con algunas aplicaciones de ejemplo creadas, de hecho la pantalla inicial es una de ellas la aplicación “Welcome” o “Bienvenido” (dependerá del lenguaje por defecto de tu navegador).

Para crear nuestra aplicación ***cornucopia***:

* Vamos al botón **admin** en la pantalla principal.
* Metemos la password con la que hemos arrancado el ***web2py*** en la linea de comandos.
* Desde la ventana de administración creamos nuestra nueva aplicación

Inmediatamente nos encontraremos en la ventana de diseño de nuestra nueva aplicación. ***web2py*** nos permite diseñar completamente nuestra aplicación desde aquí, ni siquiera necesitaremos un editor de texto (aunque nada impide usar uno, desde luego).

### private/appconfig.ini

El primer fichero que vamos a examinar es private/appconfig.ini La sección private debería estar abajo de todo en la ventana de diseño.

En la sección [app] del fichero podemos configurar el nombre de la aplicación y los datos del desarrollador.

En la sección [db] fichero configuramos el motor de base de datos que vamos a usar en nuestra aplicación. Por defecto viene configurado *sqlite* así que no vamos a tener que cambiar nada en este sentido.

En la seccion [smtp] podemos configurar el gateway de correo que usará la aplicación para enviar correos a los usuarios. Por defecto viene viene la configuración para usar una cuenta de gmail como gateway, solo tenemos que cubrir los valores de usuario y password y la dirección de correo.

### El Modelo

En la parte superior de la ventana de diseño (o edición) de nuestra aplicación tenemos la sección Models



Menú Modelos

***web2py*** se encarga de crear las tablas necesarias en la base de datos que le hayamos indicado que use.

Al crear la aplicación \_**web2py** ha creado en la base de datos todas las tablas relacionadas con la gestión de usuarios y sus privilegios.

Si echamos un ojo al modelo gráfico (*Graphs Models*) veremos las tablas que ***web2py*** ha creado por defecto y las relaciones entre ellas.

Si vemos el log de comandos de sql (*sql.log*) veremos los comandos que ***web2py*** ha ejecutado en el motor de base de datos.

Y por último si vemos *database administration* podremos ver las tablas creadas en la base de datos, e incluso crear nuevos registros en esas tablas.

También podemos echar un ojo al contenido del fichero db.py o menu.py pero por el momento **no** vamos a modificar nada en esos ficheros.

# Secciones en el futuro

## web2py y git

## Instalación con nginx

## Certificados let’s encrypt

Para usar MySQL como motor de base de datos: Editamos el fichero applications/pyfinder/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 =  
  
Editamos el fichero applications/pyfinder/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'])  
  
Creamos un fichero db\_custom.py en el directorio: applications/pyfinder/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

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);

db.define\_table(‘physical’, Field(‘id’, ‘integer’), Field(‘circuit\_id’, ‘string’), Field(‘circuit\_name’, ‘string’), Field(‘line\_type\_code’, ‘string’), Field(‘mux\_type\_code’, ‘string’), Field(‘vendor\_code’, ‘string’), Field(‘line\_of\_sight\_distance’, ‘string’), Field(‘distance’, ‘string’), Field(‘carrier\_circuit\_name’, ‘string’), Field(‘circuit\_activation\_date’, ‘string’), Field(‘circuit\_deactivation\_date’, ‘string’), Field(‘circuit\_desconection\_date’, ‘string’), Field(‘origination\_tributary’, ‘string’), Field(‘destination\_tributary’, ‘string’), Field(‘circuit\_medium\_type’, ‘string’), Field(‘phys\_conn\_bandwidth’, ‘string’), Field(‘circuit\_state\_code’, ‘string’), Field(‘from\_user\_site\_id’, ‘string’), Field(‘from\_map\_site\_id’, ‘string’), Field(‘from\_site\_name’, ‘string’), Field(‘from\_zone’, ‘string’), Field(‘from\_node\_id’, ‘string’), Field(‘from\_user\_node\_id’, ‘string’), Field(‘from\_node\_class\_code’, ‘string’), Field(‘from\_node\_technology\_code’, ‘string’), Field(‘from\_node\_state\_code’, ‘string’), Field(‘from\_phys\_node\_id’, ‘string’), Field(‘from\_node\_type\_code’, ‘string’), Field(‘from\_shelf\_no’, ‘string’), Field(‘from\_user\_shelf\_id’, ‘string’), Field(‘from\_slot\_no’, ‘string’), Field(‘from\_slot\_num\_text’, ‘string’), Field(‘from\_card\_id’, ‘string’), Field(‘from\_card\_type\_code’, ‘string’), Field(‘from\_user\_card\_id’, ‘string’), Field(‘from\_port\_no’, ‘string’), Field(‘from\_port\_id’, ‘string’), Field(‘to\_user\_site\_id’, ‘string’), Field(‘to\_map\_site\_id’, ‘string’), Field(‘to\_site\_name’, ‘string’), Field(‘to\_zone’, ‘string’), Field(‘to\_node\_id’, ‘string’), Field(‘to\_user\_node\_id’, ‘string’), Field(‘to\_node\_class\_code’, ‘string’), Field(‘to\_node\_tech\_code’, ‘string’), Field(‘to\_node\_state\_code’, ‘string’), Field(‘to\_phys\_node\_id’, ‘string’), Field(‘to\_node\_type\_code’, ‘string’), Field(‘to\_shelf\_no’, ‘string’), Field(‘to\_user\_shelf\_id’, ‘string’), Field(‘to\_slot\_no’, ‘string’), Field(‘to\_slot\_num\_text’, ‘string’), Field(‘to\_card\_id’, ‘string’), Field(‘to\_card\_type\_code’, ‘string’), Field(‘to\_user\_card\_id’, ‘string’), Field(‘to\_port\_no’, ‘string’), Field(‘to\_port\_id’, ‘string’), Field(‘loaddate’, ‘string’), Field(‘create\_user\_id’, ‘string’), Field(‘create\_date’, ‘string’), Field(‘modified\_user\_id’, ‘string’), Field(‘modified\_date’, ‘string’), migrate = False) db.define\_table(‘segment’, Field(‘id’, ‘integer’), Field(‘virtual\_link\_id’, ‘string’), Field(‘path\_name’, ‘string’), Field(‘origination\_route\_path’, ‘string’), Field(‘destination\_route\_path’, ‘string’), Field(‘path\_type’, ‘string’), Field(‘protection\_type’, ‘string’), Field(‘hop\_no’, ‘string’), Field(‘physical\_conn’, ‘string’), Field(‘physical\_conn\_id’, ‘string’), Field(‘physical\_conn\_media’, ‘string’), Field(‘physical\_conn\_bandwidth’, ‘string’), Field(‘logical\_conn\_id’, ‘string’), Field(‘logical\_conn’, ‘string’), Field(‘logical\_conn\_type’, ‘string’), Field(‘origination\_node\_id’, ‘string’), Field(‘origination\_node’, ‘string’), Field(‘origination\_node\_name’, ‘string’), Field(‘origination\_card’, ‘string’), Field(‘origination\_site\_id’, ‘string’), Field(‘origination\_site’, ‘string’), Field(‘origination\_site\_name’, ‘string’), Field(‘origination\_site\_latitude’, ‘string’), Field(‘origination\_site\_longitude’, ‘string’),

Receta para instalar desde cero bajando el repo de github Checlist

Crear bases de datos en MySQL  
 txdb  
 txdbnew  
 txdbold  
Dar privilegios a

1. Los siguientes comandos asumen que tienes instalado *virtualenvwrapper* como recomendamos en la guía de postinstalación de Linux Mint, si no lo tienes tendrás que crear un virtualenv con los comandos tradicionales [↑](#footnote-ref-31)