How To Use the Web2py Framework to Quickly Build Your Python App

Authored by: ***ASPHostServer Administrator*** *[asphostserver@gmail.com]*
Saved From: *http://faq.asphosthelpdesk.com/article.php?id=269*

The Python programming language is a versatile, high-level language that can be used easily for web programming, among other tasks. Building web applications can be difficult and toilsome on your own. Web frameworks preform much of the heavy lifting and can set up a template to build your web application easily.

Python has many different web frameworks. One of the most interesting is **web2py**. Web2py is a full-stack web framework that can be used to completely develop your web app. It has SQL database integration, a multi-threaded web server, and comes with a complete web IDE for designing your program.

This allows you to easily program with a unified interface from anywhere you can access a web browser. You can easily create and test your application in the same interface.

We will be installing and working with this framework on an Ubuntu 12.04.

Install the Web2py Software

Your Ubuntu server instance should already come with Python installed by default. This takes care of one of the only things that web2py needs to run successfully.

The only other software that we need to install is the`unzip`package, so that we can extract the web2py files from the zip file we will be downloading:

```
sudo apt-get update
sudo apt-get install unzip
```

Now, we can get the framework from the project's website. We will download this to our home folder:

```
cd ~
wget http://www.web2py.com/examples/static/web2py_src.zip
```

Now, we can unzip the file we just downloaded and move inside:

```
unzip web2py_src.zip
cd web2py
```

Now that we are inside the web2py directory, how do we install it? Well, one of the great things about web2py is that you do not install it. You can run it right from this folder by typing:

```
python web2py.py
```

However, this will only launch the web interface that is accessible on the local machine. This is a security feature, but it doesn't help us since our framework is being hosted on a remote.

To stop the server, typing "CTRL-C" in the terminal. We will sort out the web access issue momentarily.

Create SSL Certificates to Allow Remote Access

To allow remote access, we must start the web framework with SSL. We need to create our certificates before we can do that. Luckily, the`openssl`package is already installed.

We can create an RSA key to use for certificate generation with the following command:

```
openssl genrsa -out server.key 2048
```

Using this key, we can then generate the`.csr`file:

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

Finally, we can use these two pieces to create an SSL certificate:

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

You should now have a`server.key`,`server.csr`, and`server.crt`file in your web2py directory. We can use these to start up the interface in a secure manner by passing some parameters to web2py when we call it:

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

You can select whichever password you would like to use to log into your framework web interface. The "0.0.0.0" allows this to be accessible to remote systems.

You can access the interface by visiting:

```
https://your_ip:8000
```

You should get a warning that says that the SSL certificate is not signed by a known certificate authority.

This is normal, since we signed the key itself. Click the button that allows you to proceed anyways.

How To Use the Web Interface to Design Your Program

Upon visiting the site, you should be presented with the default web2py application:

If you click on the "Administrative Interface" button, you can sign in with the password you just chose when starting the server.

When you're done, you should be taken to the admin interface:

On the left, you can see three folders. These are actually included sample applications. If you click on the folder titles, you will be taken to the live apps. One is the "admin" interface that you are using now. The "welcome" app is the basic application that you saw before you signed in.

You can edit all of these applications (except the admin) from within this interface by clicking on the "Manage" drop-down menu and selecting "Edit":

If you did this, get back to the main interface by clicking the "Site" link in the top navigation bar.

Let's create a new app through the "New simple application" field on the right side of the application. We will name it "sample_app":

This takes us to the editing interface of our new "sample_app" application. It lists the standard MVC (model, view, and controller) files, as well as a languages directory, a static pages directory, and directories for modules, plugins, and private files.

This is actually just a graphical representation of what is going on in the filesystem. If you were to stop the web server with CTRL-C, and navigate within the web2py directory to the "applications/sample_app" folder, you will see folders that match these categories.

If we visit the site by going to:

```
https://your_ip:8000/sample_app
```

We can see that it looks almost exactly like the welcome app that was included by default (the only difference being the title).

Exploring the Web2py's MVC Implementation

Web2py takes care of a lot of the details necessary to form a fully functioning app. However, you must know how it interacts in order to effectively develop within this system. Web2py implements a policy of "coding-by-convention".

Coding-by-convention is a design choice selected in order to reduce the number of decisions a developer has to make. This means that a developer should only have to specify when they are moving away from a convention.

For instance, if the controller for an application is called "image_blog.py" and it has a function called "main", web2py will try to serve this page with a view called "image_blog/main.html". The ".html" part is the default if no other extension is specified in the controller. If we break this convention, we would have to specify the alternative. Otherwise, this is automatic.

We can see that there is a controller called`default.py`. This is the default controller that is used when no other is specified. If you click on the "Edit" button to the left of this controller, you can see the functions it defines:

If we return back to the main editing interface (by clicking the "Edit" link in the top navigation bar), we can see

that the functions that define user-facing actions have a matching view:

Create A Sample Controller and View

Let's try this out by creating our own controller first. Under the "Controllers" section, click "Create" to make a new controller:

If we edit this file, we will see that it defines an "index" function that will be called if no other is requested. This serves as a default. This function simply returns a dictionary that stores a variable called`message`with a value of "hello from hello.py".

```
# coding: utf8
# try something like
def index(): return dict(message="hello from hello.py")
```

You can edit this and click on the disk icon to save your work. For our purposes, this does exactly what we want it to do.

Next, we need to create the corresponding view to render the information that is being passed back (the dictionary that defines message). Click on the "Edit" button or the "<<back" button to return to the app directory.

Under the "Views" section create a new view. We need to match the controller and function for this view to be automatically applied:

Edit the file by clicking the button to the left of it. You should see the default view, which contains something like:

```
{{extend 'layout.html'}}
<h1>This is the hello/index.html template</h1>
{{=BEAUTIFY(response._vars)}}
```

This file extends, or builds off of the`layout.html`view. This is a useful method of keeping a consistent look between all of your pages. If we visit the page in our browser, we can see how it renders.

We can get to this page by visiting:

```
https://your_ip:8000/sample_app/hello
```

You can see that it rendered the results in our dictionary below the heading. We can print this directly by editing our view.

If we remove the second two lines, we can replace them with our message text, since it forms the complete message we want to show:

```
{{extend 'layout.html'}}
<h1>{{=message}}</h1>
```

Now, we should just see the message that the controller passed, rendered as an h1 header:

If we would like to get rid of the styling, we can remove the top line.

The`{{=message}}`is a way that we can embed Python code within our files. This allows us to dynamically generate content that is not necessarily available at the time the program is written.

Conclusion

You should now have a very basic understanding of the web2py framework. More importantly, you should be able to see how easy it is to develop on this framework. The flexibility of being able to develop through text files or through a web interface means that you can work well in different environments.

The web interface also comes with a great number of tools for working with your growing application. You can see stack traces when a bug is introduced, you can easily package your application for deployment, and you view a log of all of the errors your application has encountered while developing.

Web2py is a framework, an IDE, and a development suite all in one package.