Simple Python HTTP(S) Server — Example

Mar 20, 2016 13:17 · 889 Words · 5 Minute Read

PYTHON

The standard Python library has a built-in module that can be used as minimalistic HTTP/HTTPS web server. It provides support of the protocol and allows you to extend capabilities by subclassing.

Serve static HTML/CSS files to outside world can be very helpful and handy in many real life situations. For example, to show a client HTML pages you've created or stub an API by creating a static file.

Example Of Static HTTP Web Server

Yet another purpose that static web server can serve is to create a dummy API by creating json or/and xml files. The structure of resources organized in sub-folders will provide RESTful-like URLs. E.g. /users/all.json.json may contain dummy records of users. This approach even faster then creating, for instance, a Flask application. No database required, works everywhere. To download data from a remote server. Let's say there are some difficulties with scp command. It is possible to run simple server on the remote machine and download necessary contents via HTTP.

Python 3.X

```
python3 -m http.server 8000 --bind 127.0.0.1
```

Both port and bind address are optional. For more details, please read the official docs.

Python 2.X

```
python -m SimpleHTTPServer 8000
```

Python 2.x can only accept port as a parameter Bind address parameter is not available. Python 2.x Docs.

In both cases contents of the current folder will be accessible via http://127.0.0.1:8000

Example With SSL Support

To run secure HTTPs server create a following module:

Python 3.X

```
from http.server import HTTPServer, BaseHTTPRequestHandler
import ssl


httpd = HTTPServer(('localhost', 4443), BaseHTTPRequestHandler)

httpd.socket = ssl.wrap_socket (httpd.socket,
        keyfile="path/to/key.pem",
        certfile='path/to/cert.pem', server_side=True)

httpd.serve_forever()
```

Python 2.X

```
import BaseHTTPServer, SimpleHTTPServer
import ssl
```

```
httpd = BaseHTTPServer.HTTPServer(('localhost', 4443),
        SimpleHTTPServer.SimpleHTTPRequestHandler)

httpd.socket = ssl.wrap_socket (httpd.socket,
        keyfile="path/tp/key.pem",
        certfile='path/to/cert.pem', server_side=True)

httpd.serve_forever()
```

To generate key and cert files with OpenSSL use following command

```
openssl req -x509 -newkey rsa:2048 -keyout key.pem -out cert.pem -days 365
```

Further examples will assume Python 3.5+ as an interpreter.

Advanced Python HTTP Server

Let’s make our web server a little more advanced by handling requests.

Do_GET

Consider the following code:

```
from http.server import HTTPServer, BaseHTTPRequestHandler


class SimpleHTTPRequestHandler(BaseHTTPRequestHandler):

    def do_GET(self):
        self.send_response(200)
        self.end_headers()
        self.wfile.write(b'Hello, world!')
```

```
httpd = HTTPServer(('localhost', 8000), SimpleHTTPRequestHandler)
httpd.serve_forever()
```

This is a very trivial HTTP server that responds *Hello, world!* to the requester. Note, that `self.send_response(200)` and `self.end_headers()` are mandatory, otherwise the response wont be considered as valid. We can check that it actually works by sending a request using HTTPie:

```
$ http http://127.0.0.1:8000

HTTP/1.0 200 OK
Date: Sun, 25 Feb 2018 17:26:20 GMT
Server: BaseHTTP/0.6 Python/3.6.1

Hello, world!
```

Note, that `self.wfile` is a file like object, thus expects a byte-like objects to the `write` function. Another way of feeding the `wfile` is by using BytesIO object (see example below).

Do_POST

Let's handle a POST request now. Full example:

```
from http.server import HTTPServer, BaseHTTPRequestHandler

from io import BytesIO


class SimpleHTTPRequestHandler(BaseHTTPRequestHandler):

    def do_GET(self):
        self.send_response(200)
        self.end_headers()
        self.wfile.write(b'Hello, world!')

    def do_POST(self):
        content_length = int(self.headers['Content-Length'])
        body = self.rfile.read(content_length)
        self.send_response(200)
        self.end_headers()
        response = BytesIO()
        response.write(b'This is POST request. ')
        response.write(b'Received: ')
        response.write(body)
        self.wfile.write(response.getvalue())


httpd = HTTPServer(('localhost', 8000), SimpleHTTPRequestHandler)
httpd.serve_forever()
```

The request body can be accessed via `self.rfile`. It is a BufferedReader so `read([size])` method should be executed in order to get the contents. Note, that `size` should be explicitly passed to the function, otherwise the request will hang and never end.

This is why obtaining `content_length` is necessary. It could be retrieved via `self.headers` and converted into an integer. An example above just prints back whatever he receives, like follows:

```
http http://127.0.0.1:8000 key=value
HTTP/1.0 200 OK
Date: Sun, 25 Feb 2018 17:46:06 GMT
Server: BaseHTTP/0.6 Python/3.6.1

This is POST request. Received: {"key": "value"}
```

You may consider to parse the JSON if you like.

Twisted As A Simple Web HTTP(S) Server

Another great example of a web server is Twisted. Clearly, it is much faster than one built in Python and provides lots of features out of the box. It supports SSL without a need to write a single line of code. It supports both Python 3.x and 2.x.

Installation

```
pip install twisted
```

Usage

To run a twisted as a web server to serve current directory:

```
twistd -no web --path
```

You will see the output like follows:

```
(.venv) andrey@work$ ~/Projects/test_app   twistd -no web --path=.
2016-10-23T19:05:02+0300 [twisted.scripts._twistd_unix.UnixAppLogger#info] twis
2016-10-23T19:05:02+0300 [twisted.scripts._twistd_unix.UnixAppLogger#info] reac
2016-10-23T19:05:02+0300 [-] Site starting on 8080
2016-10-23T19:05:02+0300 [twisted.web.server.Site#info] Starting factory <twist
```

Options

-n, –nodaemon don't daemonize, don't use default umask of 0077

-o, –no_save do not save state on shutdown

–path= is either a specific file or a directory to be set as the root of the web server. Use this if you have a directory full of HTML, cgi, epy, or rpy files or any other files that you want to be

Commands

web A general-purpose web server which can serve from a filesystem or application resource.

If you are looking for HTTPS and SSL support, consider the following options:

–https= Port to listen on for Secure HTTP.

-c, –certificate= SSL certificate to use for HTTPS. [default: server.pem]

-k, –privkey= SSL certificate to use for HTTPS. [default: server.pem]

Docker Example

Here are an example of Dockerfile I use to serve simple html pages to outside world.

```
FROM python:3.5


VOLUME ["/code"]
ADD . /code
WORKDIR /code


EXPOSE 5000
CMD ["python", "-m", "http.server", "5000"]
```

It is possible to write custom handlers and extend the basic functionality. Including creating HTTPS server etc. Find official documentation for python 3 http server is here. Python 2 documentation is here

17
SHARES

ALSO ON **ANVILEIGHT**

Review of "Succeeding With Agile" by Mike …

4 years ago • 1 comment

This book is exactly what a company like ours requires at this point of transition. …

Sending Emails With Django Templates …

4 years ago • 8 comments

Let's explore how to store email template in Django database and manage …

Behavior Driven Development …

6 years ago • 13 comments

Setting up BDD with Django is an easy task. Especially, using PyCharm IDE. This …

Gra and

4 yea

An e appr using