A Comparison of Web Servers for Python Based Web Applications

Authored by: ***ASPHostServer Administrator*** *[asphostserver@gmail.com]*
Saved From: *http://faq.asphosthelpdesk.com/article.php?id=261*

In this article, we will talk about three main things: Python, Web Servers, and most importantly the bits and bobs in-between the two.

Jokes aside, this rather long article might seem dire for some looking for quick guidance or answers. Unfortunately, unlike most things in the world of Python, when it comes to choosing a production server stack to deploy your application on, there is not one-- and preferably only one --obvious way to do it.

However, this should not scare you. After completing this article, you will have a good knowledge of how different web servers work and handle the task of talking with Python based web applications. Upon gauging your needs and requirements, you will be able to decide which server to use.

Python Web Server Gateway Interface v1.0 (WSGI)

Understanding the Issues

Today, there exists web servers (or modules for servers) in ever growing numbers specifically designed (or adapted) to work with Python web applications interchangeably. However, this has not always been the case. In ye olde days, developers did not really have the possibility to switch web servers at will easily, and each switch came with a cost due to dependencies and limitations. Upon deciding on a framework to build on, you would have also decided, not always willingly nor consciously, on the server(s) you could use to serve the application as well. This was due to the lack of existence of a universally accepted *interface specification*: a common ground which applications (frameworks) and web servers alike would adapt and use to communicate, allowing interchangeability of components when necessary with possibly zero code change.

Birth of the Standard

At beginning of this century, efforts were made to finally solve the issue with the presentation of Python Enhancement Proposal (*PEP*) 333 to the community.

From the PEP (*Python Enhancement Proposal*) 333:

```
This document specifies a proposed standard
interface between web servers and Python web
applications or frameworks, to promote web
application portability across a variety of
web servers.
```

As explained in the PEP, this new standard was (and is) meant to allow portability across (and between) [web] servers and [Python web] applications. The standard's powerful features and its wide adoption compared to those from before lead the way for today: a world where many (perhaps too many) web servers willing to do the job for you exist.

The Comparison

In this comparison of web servers for Python based web applications, we will talk about some of the choices available and what makes them stand out. The aim here is for the reader to have a clearer vision and to provide help to match the servers against applications' custom needs to find *the one*. Due to the vast number of options (more popping up every day!), we will filter our way through and talk about those which are "special" in various ways: popularity, solidness, or doing something different (or better) compared to the rest.

Note: I would like to advise you, the reader, to be wary of biased and deceiving benchmarks which tend to *not* reflect the conditions of a real production environment. Unfortunately, those [articles] do not really mean much when it comes to choosing a web server for production, which is also highly unlikely to be the part causing the bottleneck. You are, therefore, advised to gauge and understand your own needs and then to try different options, refraining from speculative numbers to avoid real future disaster scenarios.

Web Servers (Alphabetical Order)

CherryPy WSGI Server

What is it?

CherryPy is actually a web framework. Yet it is a fully self-contained one -- meaning that it can run on its own, including in production scenarios without the need of additional software. This is achieved thanks to its own WSGI, HTTP/1.1-compliant web server. CherryPy project describes it as a "A high-speed, production ready, thread pooled, generic HTTP server". As it is a WSGI server, it can be used to serve any other WSGI Python application as well, without being bound to CherryPy's application development framework.

Why should you consider using it?

- It is compact and simple.
- It can serve any Python web applications running on WSGI.
- It can handle static files and it can just be used to serve files and folders alone.
- It is thread-pooled.
- It comes with support for SSL.
- It is an easy to adapt, easy to use pure-Python alternative which is robust and reliable.

Gunicorn

What is it?

Gunicorn is a stand-alone web server which offers quite a bit of functionality in a significantly easy to operate fashion. It uses the pre-fork model -- meaning that a central master process (Gunicorn) is tasked with managing the initiated worker processes (of differing types), which then handle and deal with the requests directly. And all this can be configured and adapted to suit your needs and diverse production scenarios.

Why should you consider using it?

- It supports WSGI and can be used with any WSGI running Python application and framework.
- It can also be used as a drop-in replacement for Paster (ex: Pyramid), Django's Development Server, web2py, et alia.
- Offers the choice of various worker types/configurations and automatic worker process management.
- HTTP/1.0 and HTTP/1.1 (Keep-Alive) support through synchronous and asynchronous workers.
- Comes with SSL support
- Extensible with hooks.
- It is transparent and has a clear architecture.
- Supports Python versions 2.6, 2.7, 3, 3.2, 3.3

Tornado (HTTP Server via wsgi.WSGIContainer)

What is it?

Tornado is an application development framework and a networking library designed for handling asynchrnous operations, allowing servers to maintain a lot of open connections. It also comes with a WSGI server which other WSGI Python applications (and frameworks) can use to run.

Why should you consider using it?

- If you are building on top Tornado framework; or
- Your application needs asynchronous functionality.

Although under these circumstances you might want to choose Tornado's WSGI server for your project, you can also opt to use Gunicorn with Tornado [Asynchronous] workers.

Twisted Web

What is it?

Twisted Web is the web server that comes with the Twisted networking library. Whereas *Twisted* itself is "an event-driven networking engine", the*Twisted Web* server runs on WSGI and it is capable of powering other Python web applications.

Why should you consider using it?

- It is a simple to use, stable and mature product.
- It will run WSGI Python applications.
- It can act like a Python web server framework, allowing you to program it with the language for custom HTTP serving purposes.
- It offers simple and fast prototyping ability through`Python Scrips (.rpy)`which are executed upon HTTP requests.
- It comes with proxy and reverse-proxy capabilities.
- It supports Virtual Hosts.
- It can even serve Perl, PHP et cetera via twisted.web.twcgi API.

uWSGI

What is it?

Despite its *very* confusing naming conventions, uWSGI itself is a vast project with many components, aiming to provide a `full [software] stack forbuilding hosting services`. One of these components, the uWSGI server, runs Python WSGI applications. It is capable of using various protocols, including its own ***uwsgi*** wire protocol, which is quasi-identical to SCGI. In order to fulfil the (understandable) demand to use stand-alone HTTP servers in front of application servers, NGINX and Cherokee web servers are modularised to support **uWSGI**'s (best performing) ***uwsgi*** protocol to have direct control over its processes.

Why should you consider using it?

- uWSGI comes with a WSGI adapter and it fully supports Python applications running on WSGI.
- It links with libpython. It loads the application code on startup and acts like a Python interpreter. It parses the incoming requests and invokes the Python callable.
- It comes with direct support for popular NGINX web server (along with Cherokee* and lighttpd).
- It is written in C.
- Its various components can do much more than running an application, which might be handy for expansion.
- Currently (as of late 2013), it is actively developed and has fast release cycles.
- It has various engines for running applications (asynchronous and synchronous).
- It can mean lower memory footprint to run.

Waitress WSGI Server

What is it?

Waitress is a pure-Python WSGI server. At a first glance it might not appear to be that much different than many others; however, its development philosophy separates it from the rest. Its aim for easing the production (and development) burden caused by web servers for Python web-application developers. Waitress achieves this by neutralizing issues caused by platform (ex. Unix vs. Windows), interpreter (CPython vs. PyPy) and Python (version 2 vs. 3) differences.

Why should you consider using it?

- It is a very lean, pure-Python solution.
- It supports HTTP/1.0 and HTTP/1.1 (Keep-Alive).
- It comes ready to be deployed for production with a wide array of platform support.
- Unlike CherryPy, it actually is framework-independent in its nature.
- It runs on Windows and Unix, and on CPython interpreter and PyPy (Unix only).
- It supports Python versions 2 and 3.

Modules for Stand-Alone Servers

---

mod_python with a WSGI adapter (Apache) (Embedding Python)

What is it?

Simply put, mod_python is an Apache module that embeds Python within the server itself. Although not

recommended for various reasons (project is dead and outdated with only very recent intentions to continue the development by original author), it can be used to run WSGI applications on Apache via wrappers.

Why should you consider using it?

You might want to program and extend Apache using Python for a specific reason.

mod_wsgi (Apache) (Embedding Python)

What is it?

Being a WSGI compliant module, mod_wsgi allows you to run Python WSGI applications on Apache HTTP Server. It achieves this in two ways: the first being similar to how mod_python works, by embedding the code and executing it within the child process. The other method offers a daemon based operation mode whereby the WSGI application has its own distinct process, managed automatically by mod_wsgi.

Why should you consider using it?

- Existing experience with Apache can mean a stable production environment for your operations when it comes to running Python as well. This alone can save the day, making it worth it.
- If you are dependant on Apache or want to take advantage of its stable and rich extension modules, it will be the way to go.
- It can run applications under different system users for further security.
- It is a tried and tested, reliable software.
- World Wide Web contains*tonnes*of information and Q&A's related to it which can save you a lot of time when you encounter a real production problem.
- And it comes with all other functionality that Apache offers.