Title	WHAT IS WSGI AND WHY IS IT NECESSARY?
Description	Intro
Date	2022 18 Augus
Author	Arashtad
Author URI	https://Arashtad.com

WSGI is a specification for how a server and application communicate. Both the server interface, as well as the application interface, are specified in PEP 3333. A WSGI application (or framework or toolkit) can be stacked on any server that conforms to the WSGI specification. These applications are compatible with each other. A middleware piece must implement both sides of the WSGI interface: the application on top, and the application below. For a middleware piece to be a middleware piece, it must implement both sides of the WSGI interface, application, and server.

WSGI OVERVIEW:

There were about 14 million users on the web in 1993, and 100 websites. Pages were static at the time, but dynamic content, such as news and data, was already necessary. Rob McCool and others implemented the Common Gateway Interface (CGI) for the HTTPd web server of the National Center for Supercomputing Applications (NCSA). Since then, the number of Internet users has exploded, and dynamic websites have become ubiquitous. This was the first web server capable of serving content generated by a separate application. When learning a new language or even at the time of learning to code, developers, soon enough, want to know how to hook their code into the web.

There have been many changes since the advent of CGI. The CGI approach became impractical because it required creating a new process on each request, wasting CPU and memory. Some other low-level approaches have emerged, such as FastCGI](1996) and mod_python (2000), which provide different interfaces between Python web frameworks and the web server. The proliferation of approaches led to limitations in developers' choices of frameworks and web servers.

As a solution to this problem, Phillip J. Eby proposed PEP-0333, the Python Web Server Gateway Interface (WSGI) in 2003. The idea was to provide a high-level, universal interface between Python applications and web servers. PEP-3333 came out in 2003 to add Python 3 support to the WSGI interface. As of today, almost all Python frameworks rely on WSGI for communication with their web servers. This is how Django, Flask, and many other popular frameworks do it.

HOW DOES WSGI WORK?

When a WSGI server receives a client request, passes it on to the application, and then sends the response back to the client. It does nothing else; All the gory details must be supplied by the application or middleware. WSGI spec is not needed to build applications on top of frameworks or toolkits. If the middleware is not already integrated into the framework or the framework provides some kind of wrapper to integrate those that are not, one must understand how to stack it with the application or framework.

WHY DO WE NEED WSGI?

Python applications cannot be run on a traditional web server. Grisha Trubetskoy, a developer, designed the Apache module mod_python in the late 1990s in order to execute arbitrary Python code. For several years in the late 1990s and early 2000s, Apache configured with mod_python ran most Python web applications. However, mod_python wasn't a standard specification, it was only an implementation that allowed Python code to run on a server. As a result, the community became aware that a consistent way to execute Python code for web applications was needed as mod_python's development slowed and security vulnerabilities were discovered. The Python community came up with WSGI as a standard interface that modules and containers could implement. WSGI is now accepted for running Python web applications.

PURPOSES OF WSGI:

WSGI PROVIDES FLEXIBILITY:

In applications, developers can replace web stack components with others. For example, they can switch from Green Unicorn to uWSGI without modifying the application. PEP 3333 states that if such an API is available and widely used in Python web servers, users will be able to choose a framework and web server that suits them, while framework and server developers can concentrate on their preferred aIn applications, developers can replace web stack components with others. For example, they can switch from Green Unicorn to uWSGI without modifying the application. PEP 3333 states that if such an API is available and widely used in Python web servers, users will be able to choose a framework and web server that suits them, while framework and server developers can concentrate on their preferred areas of specialization.
reas of specialization.

WSGI SERVERS PROMOTE SCALING:

A WSGI server is responsible for serving thousands of requests for dynamic content at once, not a framework. WSGI servers handle web server requests and decide how to forward those requests to an application framework. To scale web traffic efficiently, segregation of responsibilities is essential.

DIFFERENT IMPLEMENTATIONS OF WSGI:

There is a long list of WSGI implementations out there which you can find on the WSGI Read the Docs page. The following four implementations are among the most recommended and popular ones:

1. GREEN UNICORN

The Gunicorn web application server is an enterprise-class WSGI web server that offers a lot of functionality. It natively supports various frameworks through its adapters, making it an excellent drop-in replacement for many development servers. Gunicorn works in much the same way as Unicorn's successful Ruby web server. Both use the pre-fork model. In essence, this means that the central [Gunicorn] master process manages workers, creates sockets, and bindings, etc.

2. UWSGI

UWSGI is gaining steam as a high-performance WSGI server implementation.

3. MOD_WSGI

Mod_wsgi is an Apache HTTP Server module developed by Graham Dumpleton for hosting Python-based web applications with WSGI extensions and includes Python 2 and 3 (as of versions 2.6 and 3.2).

4. CHERRYPY

This framework uses the Python programming language to build object-oriented web applications. It wraps the HTTP protocol, but stays at a low level and does not offer much more than what exists in RFC 7231. There is no support for tasks like templating for output rendering or backend access in CherryPy. You can use it as a web server or launch it from any WSGI-compatible environment. You can extend the framework with filters, which are called at certain points during request/response processing.

WHICH FRAMEWORKS USE WSGI?

TCP/IP:
Flask and Django are both popular python frameworks that implement web Applications and both of them use WSGI. There are of course other python frameworks that use WSGI, but Django and Flask are the two most common. We will have a brief introduction to both frameworks in the following.

DJANGO

With Django, you can create secure and maintainable websites quickly and easily. The experienced developers have built it and it can handle much of the hassle of web development, so you can focus on writing your app without reinventing the wheel. The software is free and open source, has an active community, great documentation, and many free and paid support options.

FLASK

This lightweight WSGI web application framework allows users to create simple and fast web applications while scaling up to complex applications with ease. Originally designed as a wrapper for Werkzeug and Jinja, Flask has gained popularity as a web application framework for Python. Developers decide which tools and libraries they want to use. There are several extensions available from the community that makes adding new functionality easy. Flask offers suggestions but does not enforce dependencies. Developers choose the tools and libraries they want to use.

WRAPPING UP

In this article, you got familiar with WSGI, what it is, why we use it, why is it important, and how to implement it. In summary, WSGI is the Web Server Gateway Interface. It is a specification that describes how a web server communicates with web applications, and how web applications can be chained together to process one request.