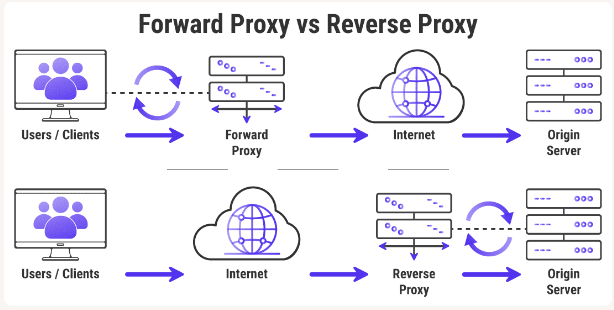
**PROXY**

Un proxy forward es un servidor que se emplea como intermediario entre las peticiones de recursos que realiza un cliente a otro servidor.

Un proxy inverso se ubica frente a un servidor web y recibe todas las solicitudes antes de que lleguen al servidor de origen. Funciona de manera similar a un proxy de reenvío, con la excepción de que en este caso es el servidor web el que utiliza el proxy y no el usuario o el cliente. Los proxies inversos se utilizan normalmente para mejorar el rendimiento, la seguridad y la fiabilidad del servidor web.

****

**¿Qué es un WSGI? (Web Server Gateway Interface)**

Empezando por [Zope](https://www.zope.org/), hace ya años que la comunidad empezó a pensar en el uso de Python con el objetivo de crear “*frameworks* para crear aplicaciones web”. Desde entonces vivimos una prolifereación de *frameworks* llegado a los más conocidos hoy en día: Flask o Django; los cuales hacen uso de WSGI.

Un servidor web tradicional no sabe, o no tiene forma, de ejecutar aplicaciones Python. A finales de los 90, el desarrollador Grisha Trubetskoy [creó un módulo para el servidor web Apache llamado mod\_python](https://grisha.org/blog/2013/10/25/mod-python-the-long-story/) con el que se podía ejecutar código Python. Sin embargo, *mod\_python* no era una especificación estándar. Digamos que no era más que una implementación que permitía correr código Python en un servidor. Pronto surgieron vulnerabilidades de seguridad y la comunidad Python empezó a implementar WSGI.

WSGI son las siglas de Web Server Gateway Interface. Es una especificación que describe cómo se comunica un servidor web con una aplicación web, y cómo se pueden llegar a encadenar diferentes aplicaciones web para procesar una solicitud/petición (o *request*).

WSGI es un estándar Python que está descrito en detalle en la especificación [PEP 3333](https://www.python.org/dev/peps/pep-3333/).

**¿porqué se necesitan los WSGI?, Intentemos explicarlo con el siguiente ejemplo.**

**Web server**

Imagina que eres un Web server y tu trabajo consiste en:

* Estar sentado tranquilamente esperando a recibir una petición (*request*) de un amable cliente.
* Cuando el cliente manda una petición, la recibes gustosamente.
* Entonces, tomas dicha petición y se la pasas a alguien llamado Aplicación Python (la web app) diciéndole:
* “*¡Ey! Despierta y mira esta petición que envía este cliente tan importante. ¡Por favor, haz algo con ella!*”
* Unos ms después, obtienes una respuesta de Aplicación Python y devuelves la respuesta al cliente raudo y veloz.

Como ves, tu única misión en la vida es esa: servir a clientes. No sabes nada más del contenido de esas peticiones. Sin embargo, eres muy muy bueno en lo que haces; si en un momento determinado recibes millones de peticiones, eres capaz de duplicarte y escalar para resolver todas esas peticiones sin problemas.

**Web App (Aplicación Python)**

Ahora imagina que eres la Aplicación Python. En este caso, sólo existes cuando el web server (que siempre está activo) te despierta. Es decir, sólo existes en tiempo de ejecución. Tu trabajo sería algo como esto:

* El web server te despierta y te manda la petición.
* Tomas la petición, la analizas y ejecutas ciertos comandos que tú sabes hacer muy bien.
* Devuelves el resultado de esas ejecuciones al web server y te vuelves a dormir.
* Finalmente, el web server entrega tus respuestas al ansioso cliente.

**Vienen los problemas**

De repente un día llegan los problemas. Las conversaciones entre el web server y la “Aplicación Python” siempre han fluido perfectamente, porque son en Español:

*“*¡Ey! Despierta y mira esta petición que envía este cliente tan importante. ¡Por favor, haz algo con ella!*”*

Sin embargo, ese día te encuentras con:

*“Эй! Проснись и посмотри на этот запрос, отправленный этим важным клиентом. Пожалуйста, сделай что-нибудь с ним!”*

Y las cosas empiezan a ir fatal porque no hay forma de entenderse ¿ves por donde voy?

El web server se puede comunicar con la Aplicación Web de muchísimas formas distintas, así que ésta tiene que aprender cientos de idiomas diferentes para entender lo que le llegue del web server.

Antes de los WSGI, los desarrolladores de las “Aplicaciones Python”, tenían que adaptar el código para adaptarse a los requerimientos del web server. Además, había que escribir diferentes códigos para cada clase de web server. Los desarrolladores están para escribir código, pero no para estar peleando con este tipo de cosas.

La solución **WSGI** !!

WSGI se pensó para solucionar este tipo de problemas. ¿Cómo solucionarlo? Fácil, igual que hacemos los humanos para comunicarnos. Si se definen una serie de reglas, todo queda claro y no encontraremos problemas.

Si quieres hablar con la “Aplicación Python”, aprende a hablar con *estas palabras*; si algo va mal, comunícalo de *esta otra forma*. Y de la misma manera, el web server tendrá que aprender *esas palabras*, saber cómo comunicarse en caso de que *algo vaya mal*, etc.

El unicornio verde

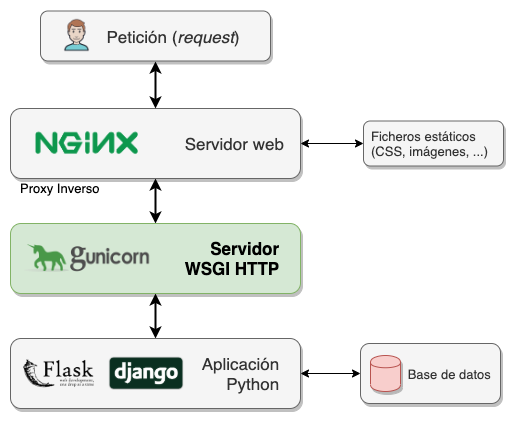
Actualmente [Gunicorn](https://gunicorn.org/) es el servidor HTTP WSGI más usado por los Pythonistas. El nombre viene de la abreviatura Green Unicorn. Está escrito en Python y su uso es extremadamente simple.



**A producción**

Para poner tu aplicación Python Django/Flask/(*framework* que use WSGI) a dar servicio tienes varias formas de hacerlo, por ejemplo uso de **stacks**:

Pese a que tanto Django como Flask tienen servidores web integrados que se suelen usar para pruebas, sólo se recomienda el uso de estos precisamente para eso: entornos de pruebas. Cuando queramos poner nuestra aplicación en producción, generalmente se usa Nginx para hacer de proxy inverso y Gunicorn como servidor WSGI. Gunicorn pude escalar con varios workers trabajando en paralelo y conectándose con nuestra aplicación Django/Flask. Quedaría así:



**Por tanto, un WSGI nos ayuda a…**

Darnos flexibilidad: los desarrolladores pueden cambiar entre Gunicorn, uWSGI,… sin cambiar el código.

Escalabilidad: gracias al uso de Gunicorn podemos servir miles de peticiones resolviendo contenido dinámico.

**OTROS CONCEPTOS**

**Application:**  
Una Application Java es un programa hecho en Java y que generalmente se ejecuta en la máquina donde se encuentra al inicio del código. Tiene en mismo sentido que un programa C o otro lenguaje.

**Applet:**  
El Applet es recuperado a distancia a través de un navegador Web y se ejecuta en el lado cliente. Devido a esta descarga, las reglas de seguridad referentes a los Applets son muy estrictas. Un ejemplo de esta seguridad es que un Applet, por ejemplo, no permite la escritura en disco ni la impresión.  
  
**Servlet:**  
Un Servlet se puede sonsiderar como un Applet pero del lado del servidor. Un Servlet dá funcionalidades a un servidor Web.

**Enlaces:**

<https://medium.com/@nachoad/que-es-wsgi-be7359c6e001#:~:text=WSGI%20son%20las%20siglas%20de,%2Fpetici%C3%B3n%20(o%20request)>.

<https://peps.python.org/pep-3333/>