

De Flask a FastAPI: Migrando hacia un backend moderno con Python

Juan Rodríguez Monti

v1. Ago 2024

Sobre mi 🖐️



Juan Rodríguez Monti

- Software Engineering Manager at RELP (www.relp.ngo)
- Software developer, security, devops +18 años
- Python, Go, Linux, Cloud, Security, Node, Unix, etc.
- Conferencista y organizador de Conferencias
- Mi primera vez en Nerdearla
- Vivo en Argentina
- Me gusta viajar, leer, cocinar
- Papá de Feli 🤝

Conceptos **core** de la charla aka Agenda

Introducción

- Flask vs FastAPI: Una visión general

Comparación de Flask vs FastAPI

- Fortalezas y limitaciones
- Características clave y ventajas
- Comparación de rendimiento

FastAPI: Características detalladas

- Tipado estático
- Documentación automática
- Validación de datos
- Manejo de dependencias

Concurrencia y paralelismo en Python

- Async y await
- GIL
- Concurrencia y paralelismo
- Uso de asyncio

Migración de Flask a FastAPI

- Motivos para migrar
- Cuando Flask ya no cubre mis necesidades

Ejemplos prácticos

- Refactorización de una API Flask a FastAPI

Todo lo que necesito ?

- Técnicas avanzadas
- Consideraciones y herramientas

Ecosistema FastAPI

- Extensiones y bibliotecas complementarias
- Integración de herramientas de Python

Escalabilidad

ORM

Resilience Patterns

Seguridad

Pruebas y despliegue

Técnicas avanzadas

Ejemplos async/sync

Websockets

Server Side Events

Startup/Shutdown events

Son muchos temas y conceptos, trataremos de ver todo, en el tiempo disponible 🙌 🙌

Flask: Fortalezas y limitaciones



Armin Ronacher

Alguna info:

- Creado en Abril 2010.
- Simplicidad. Fácilmente se puede comenzar a hacer una API.
- Comunidad, battle-tested, ecosistema de librerías. Muy usado.
- Fácilmente desplegable.
- Bajo requerimiento de recursos.

Limitaciones:

- Puede parecer “viejo” respecto de propuestas superadoras más modernas. (Opinable).
- No está optimizado para, out of the box, tener un gran rendimiento a escala.
- Dificultad para realizar tareas básicas e indispensables para APIs en producción. Por ejemplo, validación de datos de forma nativa (marshmallow, otros).
- WSGI
- No brinda información de especificaciones de API automáticamente.
- Estándares.
- No tan apto para operaciones I/O que son muy frecuentes en API.

FastAPI: Fortalezas y limitaciones



Sebastián Ramírez

Características clave:

- Creado en Diciembre 2018.
- Basado en Starlette y Pydantic.
- ASGI.
- Asincronismo con async y await: Soporte nativo para operaciones asíncronas
- Documentación automática: Genera documentación interactiva

Ventajas:

- Alto rendimiento: Comparable con frameworks como Node.js y Go.
- Relativamente simple de aprender y usar: La sintaxis es intuitiva y aprovecha características modernas de Python.
- Autocompletado, gracias a los tipos.
- Seguridad
- Soporte para websockets
- Excelente para proyectos de Machine Learning

Desventajas (me cuesta encontrarlas) :

- Curva de aprendizaje no tan simple.
- No tiene un enfoque minimalista y simple como sí tiene flask.



WSGI / ASGI

WSGI es una especificación sincrónica para interfaces de servidores y aplicaciones web en Python. Fue desarrollada como una mejora sobre las prácticas existentes para garantizar una interoperabilidad más fácil entre diferentes servidores web y aplicaciones. La especificación WSGI permite que las aplicaciones web sean servidas por servidores compatibles con WSGI.

Flask está basado en WSGI.

ASGI es una especificación asincrónica que se creó como una evolución de WSGI para admitir aplicaciones web asincrónicas. ASGI no solo permite manejar solicitudes HTTP, sino también protocolos de comunicación en tiempo real como WebSockets, lo cual es útil para aplicaciones que requieren un manejo eficiente de conexiones concurrentes.

FastAPI está basado en ASGI.



Cuando Flask u otros frameworks empezaban a no ser suficientes

En nuestra experiencia :

- Cuando las API comienzan a crecer. Se hace difícil mantener la documentación inter-squads.
- Suceden cambios en las especificaciones que muchas veces no son documentados.
- Incluso si se hace esto, es tedioso y time-consuming.
- Cuando la performance comenzaba a ser un problema, y necesitábamos poder disponer de concurrencia en las API.
- La posibilidad de probar de forma simple, e integrada, y automática los endpoints.
- La validación de datos se hace a mano, o con herramientas como marshmallow. No escala bien.
- Django no era una opción dado que traía “demasiadas” cosas. FastAPI era un buen trade off entre simplicidad y cosas disponibles en forma nativa out-of-the-box.
- Migramos todas nuestras API a FastAPI y el resultado ha sido espectacular.



Ventajas



FastAPI

FastAPI: Fortalezas. Documentación.



The screenshot displays the FastAPI documentation website. The header is teal with the FastAPI logo, navigation links (FastAPI, Learn, Reference, FastAPI People, Resources, About, Release Notes), a search bar, and GitHub repository statistics for fastapi/fastapi (0.112.0, 74.3k stars, 6.3k forks). The left sidebar, titled 'Learn', lists various topics, with 'Tutorial - User Guide' selected and expanded. The main content area is titled 'Tutorial - User Guide' and includes an introduction paragraph, a 'Run the code' section, and a terminal window showing the command to run the FastAPI dev server.

Learn

- Python Types Intro
- Concurrency and async / await
- Tutorial - User Guide** ▾
 - First Steps
 - Path Parameters
 - Query Parameters
 - Request Body
 - Query Parameters and String Validations
 - Path Parameters and Numeric Validations
 - Body - Multiple Parameters
 - Body - Fields
 - Body - Nested Models
 - Declare Request Example Data
 - Extra Data Types
 - Cookie Parameters
 - Header Parameters
 - Response Model - Return Type
 - Extra Models
 - Response Status Code

Learn > Tutorial - User Guide

Tutorial - User Guide

This tutorial shows you how to use **FastAPI** with most of its features, step by step.

Each section gradually builds on the previous ones, but it's structured to separate topics, so that you can go directly to any specific one to solve your specific API needs.

It is also built to work as a future reference.

So you can come back and see exactly what you need.

Run the code

All the code blocks can be copied and used directly (they are actually tested Python files).

To run any of the examples, copy the code to a file `main.py`, and start `fastapi dev` with:

```
bash
$ fastapi dev main.py
```

Table of contents

- Run the code
- Install FastAPI
- Advanced User Guide

FastAPI: Quien lo usa ?



FastAPI: Fortalezas.

Documentación automática.



Fast API 0.10.0 OpenAPI 3.0
/openapi.json

default

GET /items/ Read Items Get

Parameters Cancel

Name	Description
q array[string] (query)	<input type="text" value="foo"/> -
	<input type="text" value="bar"/> -
<input type="button" value="Add Item"/>	

Responses

Curl

```
curl -X GET "http://localhost:8000/items/?q=foo&q=bar" -H "accept: application/json"
```

Request URL

```
http://localhost:8000/items/?q=foo&q=bar
```

Server response

Code	Details
------	---------



FastAPI: Fortalezas.
Performance.




Fast: Very high performance, on par with **NodeJS** and **Go**
(thanks to Starlette and Pydantic). *One of the fastest Python
frameworks available.*



FastAPI: Fortalezas.

Tipos.  

- Un enfoque moderno que usa las últimas características de Python
- Tipos
- Validar datos de entrada y salida
- Estándares
- Describir los datos de la API
- Generar documentación automática
- Correcciones y sugerencias en el IDE
- Posibilidad de escribir código más rápidamente y con menos errores



FastAPI: Fortalezas.
Concurrencia. Async. ✓ ✓

Versiones recientes de Python tienen soporte para código asíncrono, a través del uso de corutinas, `async`, `await`.



FastAPI: Fortalezas.

Casos de uso más complejos.



- Background tasks
- Integración simple con SQLAlchemy
- Se puede usar para APIs no REST (GraphQL, gRpc)
- Soporte para Websockets
- Al estar basado en estándares y generar especificaciones hay posibilidad de generación de clientes automáticos para las API
- Validación de tipos estándar en Python y otros más exóticos : URL, Email, UUID.
- Startup y shutdown events.
- CORS, Gzip, static files.
- Test client built with HTTPX



FastAPI: Sostenido en la espalda de gigantes

Starlette is a lightweight [ASGI](#) framework/toolkit, which is ideal for building async web services in Python.

Pydantic; Data validation using Python type hints.

Fast and extensible, Pydantic plays nicely with your linters/IDE/brain. Define how data should be in pure, canonical Python 3.8+; validate it with Pydantic.



Es mejor FastAPI que Flask?

Es mejor FastAPI que Flask?

ChatGPT 4o ▾

Es mejor FastAPI que Flask ?



Tanto FastAPI como Flask son frameworks populares para desarrollar aplicaciones web en Python, pero tienen diferentes enfoques y características que pueden hacer que uno sea más adecuado que el otro dependiendo de las necesidades del proyecto. A continuación, se presentan algunas comparaciones clave para ayudarte a decidir cuál es mejor para tu caso específico:

FastAPI

1. Rendimiento:

- FastAPI es conocido por su alto rendimiento gracias a su uso de Python asíncrono (asyncio) y la posibilidad de usar WebSockets.
- Puede manejar más solicitudes por segundo en comparación con Flask, lo que es útil para aplicaciones de alto tráfico.

2. Validación de datos:

- Utiliza Pydantic para la validación automática de datos y la creación de esquemas. Esto simplifica el manejo de la validación y la conversión de datos.

3. Documentación Automática:

- Genera documentación automática de la API (OpenAPI y Swagger UI) sin esfuerzo adicional.



DEPENDE



Es mejor FastAPI que Flask y otros ?

IMHO, como dije : SI.



Es mejor FastAPI que Flask?

PERO

Si tu aplicación :

- Es muy chica
- No necesitás documentación formal o automática
- Se busca tener una solución “micro”
- Se busca tener un scope de recursos bajo
- Se necesita desplegar lambdas
- [Inserte acá su necesidad]
- Para algunas personas Flask escala mejor. Esto no es necesariamente así, pero con FastAPI hay que ser cuidadoso por su característica asíncrona en lo que respecta a escalar una app.



Es mejor FastAPI que Flask?

FastAPI es mejor

- Documentación automática
- Asíncrono por defecto
- Validación de datos de entrada y salida con Pydantic
- Tipos
- Soporte nativo para Websockets
- Inyección de dependencias
- Amigo de los estándares (OpenAPI)
- Construido arriba de Starlette y Pydantic
- Integración simple con ORM
- Integración simple otro tipo de API: gRPC, GraphQL, etc.
- Escalable, aunque haciéndolo con cuidado.



Tipos (este y muchos otros temas podrían ser perfectamente una charla de ese tema)

Tipado Estático. FastAPI utiliza el tipado estático de Python para mejorar la autocompletación y verificación de tipos en el código.

Modelos Pydantic. Los modelos Pydantic permiten definir tipos de datos explícitos para las entradas y salidas de la API, garantizando la validación y conformidad de los datos.

Tipos de Parámetros. FastAPI usa las anotaciones de tipo de Python para identificar y validar automáticamente los parámetros de las rutas, como parámetros de consulta, ruta y cuerpo.

Dependencias Tipadas. Las dependencias en FastAPI pueden definirse utilizando anotaciones de tipo, facilitando la inyección y validación de dependencias en los endpoints de la API.

Tipado para Respuestas. FastAPI permite especificar el tipo de dato de la respuesta mediante anotaciones de tipo, mejorando la documentación automática y validación de la salida.



Tipado estático en FastAPI: Beneficios y ejemplos

```
3
4  app = FastAPI()
5
6  @app.get("/items/", response_model=List[str])
7  async def read_items() -> List[str]:
8      |   return ["item1", "item2"]
9
10 |
```



Tipado estático en FastAPI: Beneficios y ejemplos

```
juanrodriguezmonti@MacBook-M3-Pro-de-Juan code % curl -X GET "http://127.0.0.1:8000/items/"  
["item1","item2"]%
```




Endpoints.

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
async def root():
    return {"message": "Hello World"}
```



Endpoints. Path parameters.

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/items/{item_id}")
async def read_item(item_id):
    return {"item_id": item_id}
```



Endpoints. Path parameters con tipos.

```
from fastapi import FastAPI
```

```
app = FastAPI()
```

```
@app.get("/items/{item_id}")
```

```
async def read_item(item_id: int):
```

```
    return {"item_id": item_id}
```

Endpoints. Path parameters con tipos.

```
from fastapi import FastAPI
```



```
app = FastAPI()
```

```
fake_items_db = [{"item_name": "Foo"}, {"item_name": "Bar"}, {"item_name": "Baz"}]
```

```
@app.get("/items/")
```

```
async def read_item(skip: int = 0, limit: int = 10):
```

```
    return fake_items_db[skip : skip + limit]
```



Validación de datos con Pydantic en FastAPI

- **Validación de Datos**
- **Conversión Automática:** Convierte automáticamente los datos de entrada.
- **Declaración de Modelos:** Permite declarar modelos de datos utilizando clases de Python.
- **Errores Descriptivos:** Proporciona mensajes de error detallados cuando la validación de datos falla.
- **Compatibilidad con Anotaciones de Tipo:** Integra perfectamente con las anotaciones de tipo de Python, mejorando la claridad y mantenibilidad del código.
- **Documentación Automática:** Contribuye a la generación automática de documentación interactiva de la API
- **Manipulación de Datos:** Facilita la manipulación y transformación de datos a través de sus métodos y funcionalidades integradas.
- **Uso de Validadores Personalizados:** Permite definir validadores personalizados.
- **Soporte para Modelos Anidados:** Soporta la creación de modelos anidados, lo que permite estructurar datos complejos de manera eficiente.

Validación de datos con Pydantic en FastAPI

```
main.py x
main.py > obtener_articulos
1  from fastapi import FastAPI # type: ignore
2  from pydantic import BaseModel
3  from typing import List
4
5  app = FastAPI()
6
7  # Definir el modelo de datos utilizando Pydantic
8  class Artículo(BaseModel):
9      nombre: str
10     precio: float
11     en_stock: bool = None
12
13 @app.post("/articulos/", response_model=Artículo)
14 async def crear_articulo(articulo: Artículo):
15     return articulo
16
17 # Vamos a usar el response model para indicar que es una lista del modelo de datos "Artículo"
18 @app.get("/articulos/", response_model=List[Artículo])
19 async def obtener_articulos():
20     articulos = [
21         {"nombre": "Artículo 1", "precio": 10.0, "en_stock": True},
22         {"nombre": "Artículo 2", "precio": 20.0, "en_stock": False},
23         {"nombre": "Artículo 3", "precio": 30.0, "en_stock": True},
24         {"nombre": "Artículo 4", "precio": 40.0, "en_stock": False},
25         {"nombre": "Artículo 5", "precio": 50.0, "en_stock": True},
26     ]
27     return [Artículo(**articulo) for articulo in articulos]
28
```



Validación de datos con Pydantic en FastAPI


```
juanrodriguezmonti@MacBook-M3-Pro-de-Juan code % curl -X 'GET' \  
  'http://127.0.0.1:8000/articulos/' \  
  -H 'accept: application/json'
```

```
[{"nombre": "Artículo 1", "precio": 10.0, "en_stock": true}, {"nombre": "Artículo 2", "precio": 20.0, "en_stock": false}, {"nombre": "Artículo 3", "precio": 30.0, "en_stock": true}, {"nombre": "Artículo 4", "precio": 40.0, "en_stock": false}, {"nombre": "Artículo 5", "precio": 50.0, "en_stock": true}]
```

main.py X

main.py > obtener_articulos

```
1  from fastapi import FastAPI # type: ignore
2  from pydantic import BaseModel
3  from typing import List
4
5  app = FastAPI()
6
7  # Definir el modelo de datos utilizando Pydantic
8  class Artículo(BaseModel):
9      nombre: str
10     precio: float
11     en_stock: bool = None
12
13     @app.post("/articulos/", response_model=Articulo)
14     async def crear_articulo(articulo: Articulo):
15         return articulo
16
17     # Vamos a usar el response model para indicar que es una lista del modelo de datos "Articulo"
18     @app.get("/articulos/", response_model=List[Articulo])
19     async def obtener_articulos():
20         articulos = [
21             {"nombre": "Artículo 1", "precio": 10.0, "en_stock": True},
22             {"nombre": "Artículo 2", "precio": 20.0, "en_stock": False},
23             {"nombre": "Artículo 3", "precio": 30.0, "en_stock": True},
24             {"nombre": "Artículo 4", "precio": 40.0, "en_stock": False},
25             {"nombre": "Artículo 5", "precio": 50.0, "en_stock": True, "otro_dato": False},
26         ]
27         return [Articulo(**articulo) for articulo in articulos]
28
```

Como podemos ver, descarta el campo extra, para respetar la sanitización en la devolución de contenido respetando lo definido en el modelo de respuesta. Este tipo de features de FastAPI son geniales, brindan estabilidad al proceso de desarrollo, seguridad en la aplicación y permite avanzar de forma rápida eficiente en la creación de API. Esto junto a validación, conversión, detección temprana de errores, etc.

```
juanrodriguezmonti@MacBook-M3-Pro-de-Juan code % curl -X 'GET' \
'http://127.0.0.1:8000/articulos/' \
-H 'accept: application/json'
```

```
[{"nombre": "Artículo 1", "precio": 10.0, "en_stock": true}, {"nombre": "Artículo 2", "precio": 20.0, "en_stock": false}, {"nombre": "Artículo 3", "precio": 30.0, "en_stock": true}, {"nombre": "Artículo 4", "precio": 40.0, "en_stock": false}, {"nombre": "Artículo 5", "precio": 50.0, "en_stock": true}]
```



Manejo de dependencias en FastAPI

Gestión de Dependencias: FastAPI permite gestionar y resolver dependencias de manera automática y eficiente, facilitando la modularidad y la reutilización del código.


Declaración Sencilla: Las dependencias se declaran utilizando la función `Depends`, lo que permite definir claramente qué componentes necesita cada ruta o función.

Scope de las Dependencias: Se pueden definir dependencias a diferentes niveles de alcance, como por solicitud, por sesión o a nivel global, proporcionando flexibilidad en la gestión del estado.

Automatización de Tareas: Las dependencias pueden automatizar tareas comunes como la autenticación, autorización, conexión a bases de datos y validación de datos, simplificando el código de las rutas.

Pruebas Unitarias Facilitadas: La inyección de dependencias hace que el código sea más fácil de probar, permitiendo inyectar dependencias simuladas durante los tests unitarios.

Documentación Automática: Las dependencias definidas se incluyen automáticamente en la documentación generada por FastAPI.



```
29 from fastapi import Depends
30
31 def common_parameters(q: str = None, skip: int = 0, limit: int = 10):
32     return {"q": q, "skip": skip, "limit": limit}
33
34 @app.get("/items/")
35 async def read_items(common: dict = Depends(common_parameters)):
36     return common
37
```



Sin parámetros (valores default)

```
juanrodriguezmonti@MacBook-M3-Pro-de-Juan code % curl -X 'GET' \  
  'http://127.0.0.1:8000/items/' \  
  -H 'accept: application/json'
```

```
{"q":null,"skip":0,"limit":10}%
```



Con dos parámetros

```
juanrodriguezmonti@MacBook-M3-Pro-de-Juan code % curl -X 'GET' \  
  'http://127.0.0.1:8000/items/?q=example&skip=5&limit=20' \  
  -H 'accept: application/json'
```

```
{"q":"example","skip":5,"limit":20}%
```



Y mucho, mucho, mucho más

- Request Body
- Query Parameters and String Validations
- Path Parameters and Numeric Validations
- Body - Multiple Parameters
- Body - Fields
- Body - Nested Models
- Declare Request Example Data
- Extra Data Types
- Cookie Parameters
- Header Parameters
- Response Model - Return Type
- Extra Models
- Response Status Code
- Form Data
- Request Files
- Request Forms and Files
- Handling Errors
- Path Operation Configuration
- [...]

Posible desventaja antes de seguir. Pythonic?





Posible desventaja antes de seguir. Pythonic?

- A menudo es criticado FastAPI por puristas del lenguaje o expertos en Python por considerar que no tiene un enfoque “Pythonic”.
- Pythonic-metro ? No, no se puede medir de modo exacto, pero sabemos a qué se refiere.
- Quienes lo critican tienen puntos razonables al respecto.
- Uso “excesivo” de dependencias de tipos, automatización de cosas de modo no explícito, la capa adicional de abstracción de validación de pydantic, no es tan sencillo y directo como Flask u otros frameworks más simples, el uso de async-await, [...]
- Es opinable, muy opinable. Respeto la opinión a favor y en contra.
- FastAPI tiene cosas inspiradas en otros lenguajes, lo cual puede llevar incrementar esta idea.
- Para mí es genial FastAPI. Es cierto que a veces cuando se complejizan los endpoints o las funcionalidades el código puede perder expresividad y elegancia, pero me parece que es un punto menor considerando todo lo que FastAPI ofrece y la cantidad de cosas que soluciona.




La programación concurrente es
fácil

La programación concurrente 🖱️



Concurrencia, paralelismo, asíncrono, asyncio, promesas, threads, procesos, race conditions...






En ciencias de la computación, **conurrencia** se refiere a la habilidad de distintas partes de un programa, algoritmo, o problema de ser ejecutado en desorden o en orden parcial, sin afectar el resultado final.

Definición de Wikipedia.



En la **informática**, el **paralelismo** es la simple aplicación de múltiples **CPU** a un problema único.

Definición de Wikipedia.



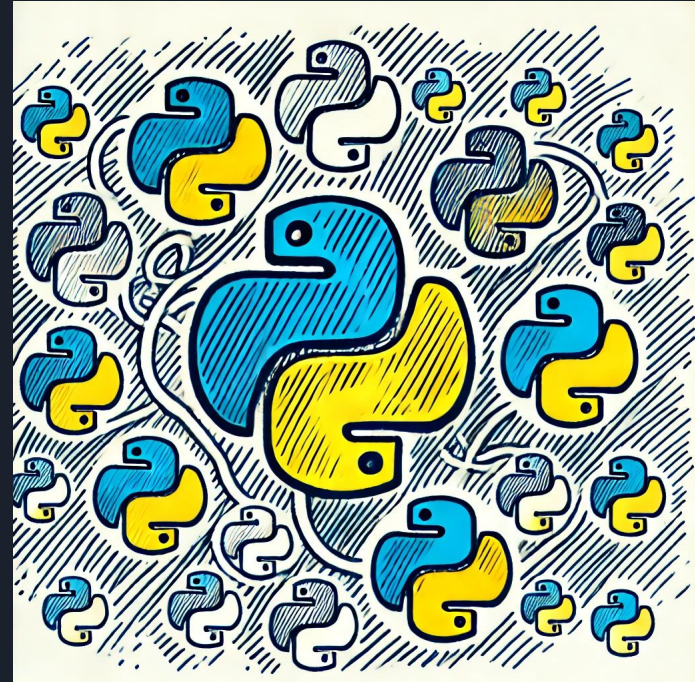
Asynchrony, in [computer programming](#), refers to the occurrence of events independent of the main [program flow](#) and ways to deal with such events.

Definición de Wikipedia.

Python

En Python tenemos varias maneras de buscar concurrencia:

- Threads
- Procesos
- Asyncio

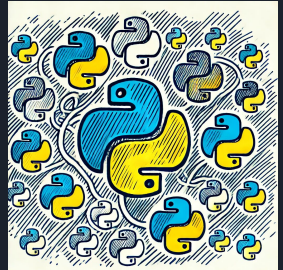



Cpython?

Es la implementación de Python oficial, y la más usada.

Hay otras? Claro que si.

- Jython
- IronPython
- PyPy
- ...

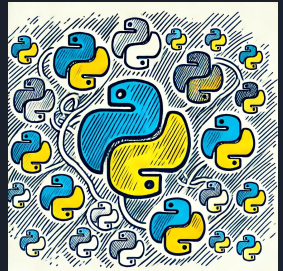




Threads. Un flujo de ejecución separado, que puede parecer se ejecuta al mismo tiempo.

CPython implementation detail: In CPython, due to the [Global Interpreter Lock](#), only one thread can execute Python code at once (even though certain performance-oriented libraries might overcome this limitation). If you want your application to make better use of the computational resources of multi-core machines, you are advised to use [multiprocessing](#) or [concurrent.futures.ProcessPoolExecutor](#). However, threading is still an appropriate model if you want to run multiple I/O-bound tasks simultaneously.

<https://docs.python.org/3/library/threading.html>



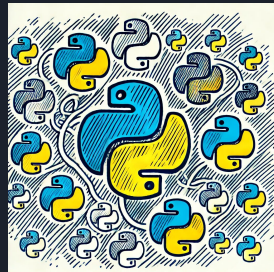


GIL? Global interpreter lock

A **global interpreter lock (GIL)** is a mechanism used in computer-language [interpreters](#) to synchronize the execution of [threads](#) so that only one native thread (per process) can execute basic operations (such as [memory allocation](#) and [reference counting](#)) at a time. ^[1]

https://en.wikipedia.org/wiki/Global_interpreter_lock

<https://docs.python.org/3/library/threading.html>

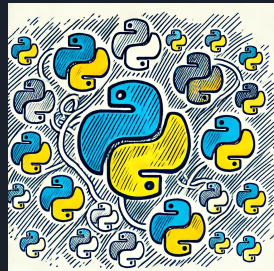


Multiprocessing

[multiprocessing](#) is a package that supports spawning processes using an API similar to the [threading](#) module. The [multiprocessing](#) package offers both local and remote concurrency, effectively side-stepping the [Global Interpreter Lock](#) by using subprocesses instead of threads. Due to this, the [multiprocessing](#) module allows the programmer to fully leverage multiple processors on a given machine. It runs on both POSIX and Windows.

Logra paralelismo en múltiples
cores.

<https://docs.python.org/3/library/multiprocessing.html>



Asyncio

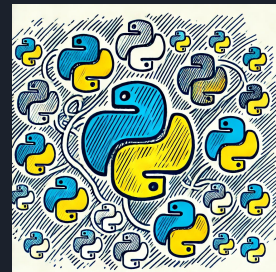
`asyncio` es una biblioteca para escribir código **concurrente** utilizando la sintaxis **`async/await`**.

`asyncio` es utilizado como base en múltiples *frameworks* asíncronos de Python y provee un alto rendimiento en redes y servidores web, bibliotecas de conexión de base de datos, colas de tareas distribuidas, etc. `asyncio` suele encajar perfectamente para operaciones con límite de E/S y código de red **estructurado** de alto nivel.

`asyncio` provee un conjunto de *APIs* de **alto nivel** para:

- [ejecutar corutinas de Python](#) de manera concurrente y tener control total sobre su ejecución;
- realizar [redes E/S y comunicación entre procesos \(IPC\)](#);
- controlar [subprocesos](#);
- distribuir tareas a través de [colas](#);

<https://docs.python.org/es/3/library/asyncio.html>





Veamos algunos ejemplos.



Sin concurrencia, ni paralelismo.

Ejecuto varias cosas sin multithreading, multiproceso, o asyncio en Python en aplicaciones que no requieren esperas de I-O (red, disco, APIs, DDBB, etc) en un único core.





Con concurrencia, 1 único core.

Tareas que tienen esperas de I/O.



 Espera tarea amarilla



 Espera tarea amarilla

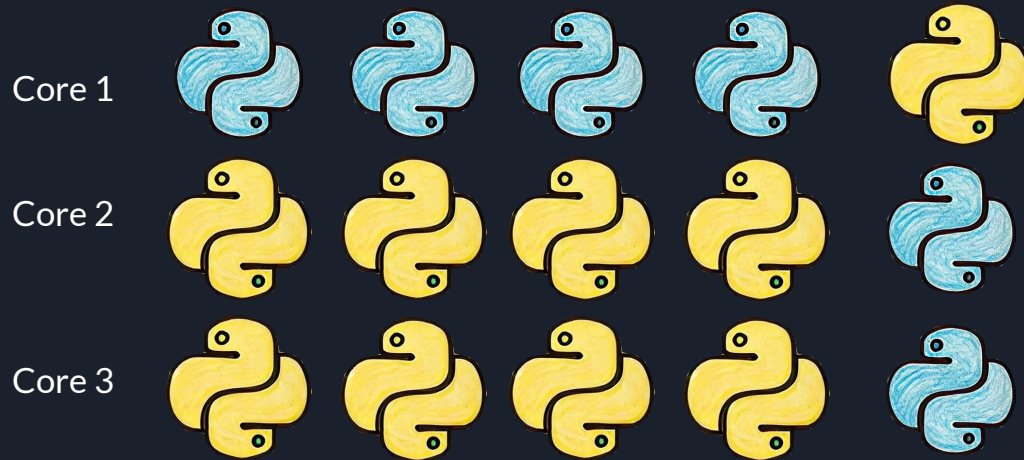
[...]

**Sirve. Mejoramos
el tiempo.**





Con concurrencia, y varios cores.



Con concurrencia y paralelismo en varios cores.






Paralelismo

Paralelismo es hacer varias tareas a la vez al mismo tiempo.



Paralelismo

Paralelismo requiere tener varios cores de CPU. En caso contrario no es posible.



Async. Qué ventajas voy a tener? Va a ser mucho más rápido.

Chess master Judit Polgár hosts a chess exhibition in which she plays multiple amateur players. She has two ways of conducting the exhibition: synchronously and asynchronously.

Assumptions:

- 24 opponents
- Judit makes each chess move in 5 seconds
- Opponents each take 55 seconds to make a move
- Games average 30 pair-moves (60 moves total)

Synchronous version: Judit plays one game at a time, never two at the same time, until the game is complete. Each game takes $(55 + 5) * 30 == 1800$ seconds, or 30 minutes. The entire exhibition takes $24 * 30 == 720$ minutes, or **12 hours**.

Asynchronous version: Judit moves from table to table, making one move at each table. She leaves the table and lets the opponent make their next move during the wait time. One move on all 24 games takes Judit $24 * 5 == 120$ seconds, or 2 minutes. The entire exhibition is now cut down to $120 * 30 == 3600$ seconds, or just **1 hour**.

[\(Source\)](#)

Wrap-up. Threads, Multi-processing, Asyncio.

Característica	Multithreading	Multiprocessing	Asyncio
Scope	Concurrente dentro de un solo proceso	Paralelismo verdadero con múltiples procesos	Concurrencia dentro de un solo hilo
Eficiencia	Limitado por GIL, bueno para I/O	Eficiente para CPU-bound tasks	Excelente para tareas I/O-bound
True Parallelism	No (debido al GIL)	Sí	No. Concurrente, no paralelo
Memoria Compartida	Compartida entre hilos	Aislada por proceso (no compartida)	Compartida en un solo hilo
Comunicación	Usar locks y condiciones	Usar multiprocessing. Queue o Pipe	Usar asyncio. Queue
Uso de CPU	No escala bien con múltiples núcleos	Escala bien con múltiples núcleos	No escala con múltiples núcleos
Context Switching	Ligero	Costoso (creación de procesos)	Ligero
Implementación	Fácil para I/O-bound, complicado para CPU-bound	Sencillo para CPU-bound, complicado para I/O-bound	Fácil para I/O-bound
Sobrecarga	Baja (dentro del mismo proceso)	Alta (creación y gestión de procesos)	Baja (dentro del mismo hilo)
Manejo de Excepciones	Complicado	Relativamente sencillo	Sencillo
Ejemplos de Uso	Operaciones de red, I/O disk	Procesamiento de imágenes, cálculos científicos	Servidores web, operaciones de red

Hagamos algunas llamadas a la API de Rick and Morty ->
<https://rickandmortyapi.com/>

The Rick and Morty API



Big Head Morty

• Unknown - Human

Last known location:
Citadel of Ricks

First seen in:
The Rickshank Rickdemption



Tricia Lange

• Alive - Human

Last known location:
Earth (Replacement Dimension)

First seen in:
The Whirly Dirty Conspiracy



Flower Morty

• Alive - Human

Last known location:
Citadel of Ricks

First seen in:
The Ricklantis Mixup



Debrah's Partner

• Alive - Mythological Creature

Last known location:
Draygon

First seen in:
Claw and Hoarder: Special Ricktim's Morty



Mr. Nimbus' Squid

• Dead - Animal

Last known location:
Earth (Replacement Dimension)

First seen in:
Mort Dinner Rick Andre




Planetina

• Alive - Humanoid

Last known location:
Earth (Replacement Dimension)

First seen in:
A Rickconvenient Mort



```
fetch-sync.py × fetch-async.py 1
Users > juanrodriguezmonti > keys > fetch-sync.py > ...
1 import requests
2 import time
3
4 URL = "https://rickandmortyapi.com/api/character/"
5
6 def fetch_character(character_id):
7     response = requests.get(f"{URL}{character_id}", verify=False)
8     if response.status_code == 200:
9         return response.json()
10    else:
11        return None
12
13 def fetch_characters_sync():
14     start_time = time.time()
15     characters = []
16     for i in range(1, 501):
17         character = fetch_character(i)
18         if character:
19             characters.append(character)
20     end_time = time.time()
21     print(f"Tiempo total sincrónico: {end_time - start_time} segundos")
22     return characters
23
24 fetch_characters_sync()
25
```

PROBLEMAS 1 SALIDA CONSOLA DE DEPURACIÓN **TERMINAL** PUERTOS

Tiempo total sincrónico: 140.31176495552063 segundos
○ juanrodriguezmonti@MacBook-M3-Pro-de-Juan keys %

✖ 0 1 0

Sync version
140.31
segundos



The image shows a code editor with two tabs: 'fetch-sync.py' and 'fetch-async.py 1 X'. The active tab 'fetch-async.py' contains the following Python code:

```
Users > juanrodriguezmonti > keys > fetch-async.py > fetch_characters_async
1 import aiohttp
2 import asyncio
3 import time
4
5 URL = "https://rickandmortyapi.com/api/character/"
6
7 async def fetch_character(session, character_id):
8     async with session.get(f"{URL}{character_id}", ssl=False) as response:
9         if response.status == 200:
10             return await response.json()
11         else:
12             return None
13
14 async def fetch_characters_async():
15     start_time = time.time()
16     async with aiohttp.ClientSession() as session:
17         tasks = [fetch_character(session, i) for i in range(1, 501)]
18         characters = await asyncio.gather(*tasks)
19     end_time = time.time()
20     print(f"Tiempo total asincrónico: {end_time - start_time} segundos")
21     return characters
22
23 asyncio.run(fetch_characters_async())
24
```

The terminal at the bottom shows the command execution and the output:


```
● juanrodriguezmonti@MacBook-M3-Pro-de-Juan keys % python3 fetch-async.py
Tiempo total asincrónico: 6.702107906341553 segundos
○ juanrodriguezmonti@MacBook-M3-Pro-de-Juan keys %
```

Async version
6.70 segundos




**Casi 21 veces más
rápido!**

133 segundos menos



Qué más necesitamos tener y
analizar para tener una API que
pueda desplegarse en un entorno
de producción moderno con
FastAPI ?

- 
- Testing
 - Seguridad
 - Logs y observabilidad
 - Resilience Patterns
 - Escalabilidad
 - Deployment



Testing

Thanks to **Starlette**, testing **FastAPI** applications is easy and enjoyable.

It is based on **HTTPX**, which in turn is designed based on Requests, so it's very familiar and intuitive.

With it, you can use **pytest** directly with **FastAPI**.

<https://fastapi.tiangolo.com/tutorial/testing/>



Logs y Observabilidad



Podemos integrar en FastAPI

- Loguru (<https://github.com/Delgan/loguru>)
Logging con pilas incluidas
- Prometheus (<https://prometheus.io/>)
Solución de monitoreo
- OpenTelemetry (<https://opentelemetry.io/>)



Seguridad



Seguridad en FastAPI

Toda API que vaya a ser usada en un entorno productivo debe respetar altos estándares de seguridad.



Pydantic

- **Validación automática:** Garantiza que los datos entrantes cumplen con los tipos y restricciones definidos por nosotros.
- **Deserialización segura:** Reduce el riesgo de inyección de datos maliciosos.
- **Consistencia de datos:** Limpia y normaliza los datos, asegurando que solo se procesen datos válidos según las especificaciones.
- **Errores manejables:** Proporciona mensajes claros de error, facilitando la detección y corrección de problemas.
- **Protección contra datos inesperados:** Rechaza datos adicionales no definidos en los modelos.
- **Validación en profundidad:** Soporta modelos anidados y validaciones complejas.



Seguridad en FastAPI

- Algo que no está bueno es que no es posible definir un mecanismo de seguridad para la documentación automática de FastAPI.
- Si son entornos de desarrollo, podemos usarlo con tranquilidad.
- Si son entornos productivos, desaconsejo dejar eso allí, y tener dos ambientes al menos para poder trabajar con esto.
- Otra posibilidad, aunque débil desde el lado de la seguridad, es implementar un mecanismo de simple auth con `usr / pwd`.



Seguridad en FastAPI

- Soporte para mecanismos estándar de seguridad (OAuth2, OAuth1, OpenID)
- Soporte para trabajar con JWT web tokens
- Manejo de CORS
- Manejo de cabeceras custom
- Pydantic
- Middlewares
- Sanitización y definición de esquemas de entrada y output
- Integración simple con Auth0 (<https://auth0.com/blog/build-and-secure-fastapi-server-with-auth0/>)




Rate Limiter

- Cuando vamos a desplegar una API, tenemos muchas capas y formas de protegerla. Desde la infra, hasta el deploy con herramientas y configuraciones.
- Sin embargo, cuando queremos tener control de rate limiting (es decir, que alguien no pueda hacer 600 requests malintencionadamente en un tiempo corto, tenemos herramientas para lidiar con esto.



Rate Limiter

- SlowAPI.
- A rate limiting library for Starlette and FastAPI adapted from [flask-limiter](#). This package is used in various production setups, handling millions of requests per month, and seems to behave as expected. There might be some API changes when changing the code to be fully `async`, but we will notify users via appropriate semver version changes.



CI-CD con análisis de dependencias, linting y análisis estático y de seguridad

- Para avanzar en la seguridad de nuestro proyecto FastAPI, podemos armar un flujo automático, por ejemplo, con Github Actions que incluya :
 - Análisis de dependencias con Dependabot
 - Análisis de dependencias con Bandit y Safety
 - Flake8 y Black (PEP8, calidad de código, etc)
 - MyPy



Resilience Patterns



Resilience patterns

- Retry : <https://tenacity.readthedocs.io/en/latest/>
- Circuit Breaker: <https://github.com/fabfuel/circuitbreaker>
- Timeout Pattern: Se puede implementar con asyncio con lo que ya trae incluido FastAPI.
- Fallback Pattern: También lo podemos implementar con FastAPI.
- Bulkhead Pattern: ídem a lo anterior.



Escalabilidad



Escalabilidad

- Implementación de caché
- Escalabilidad horizontal y vertical en los microservicios
- Uso de mayor cantidad de workers (con sus implicancias)
- Diseño de soluciones con colas de mensajería (rabbitmq) donde hay productores y consumidores de información
- Load balancers
- Correcto uso de la concurrencia. Librerías asíncronas
- Multiprocessing para tareas donde se necesite mucha cpu
- Identificación de cuellos de botella a través de monitoreo y observabilidad



Deployment



EC2

NGINX

- Podemos desplegar nuestra aplicación FastAPI utilizando EC2 si usamos AWS.
- Un enfoque tradicional sería utilizar Nginx como proxy reverso.
- Podemos correr la aplicación, en nuestra instancia, por ejemplo, Linux de EC2, en un puerto 8xxx.
- Nginx podemos configurarlo para que sólo permita HTTPS (puerto 443) y se comuniquen con nuestra aplicación.



EC2

NGINX

- Seguridad
- Grupos de seguridad restrictivos en la instancia ec2
- Firewall configurado debidamente en la instancia
- Podemos hardenizar Nginx. Podemos hacer varias cosas desde la forma en que lo configuramos (restringir paths, usar expresiones regulares, rechazar ciertos User-Agents, etc.)
- Mod-Security (reached end of life) o aplicaciones similares
- fail2ban
- Esconder los banners que informan qué servidor estamos corriendo para prevenir problemas de seguridad
- Headers custom
- Ejecutar entornos restringidos y con permisos de usuario muy limitados



EC2


NGINX

- Vamos a necesitar un certificado si deseamos poder ejecutar nuestra aplicación FastAPI con HTTPs.
- Podemos usar certbot en Linux (<https://certbot.eff.org/>) para certificados emitidos por una entidad conocida y respetada.
- Podemos usar, si disponemos de una cuenta de AWS, la herramienta de emisión de certificados de AWS, o cualquier cloud provider.
- Si usamos certbot, vamos a necesitar probar el ownership de nuestro dominio, y podemos hacer con un cron que sea un proceso automático la renovación, además de ser gratuito.



FastAPI - Otras formas de deploy

- Docker / Kubernetes
- Lambda ; Mangum :
<https://github.com/jordaneremieff/mangum?tab=readme-ov-file>
- Plataformas que resuelven el despliegue : Platform.sh ,
Porter, Coherence (sugeridos en la web de FastAPI)



Wrap-up. Qué aprendí luego de más de un año de uso.

- La validación de inputs y outputs y la conversión de datos es **FABULOSA**
- Se hace muy fácil mantener la interacción entre squads de integración de API, Frontend-Backend, gracias a la documentación automática siempre actualizada y con la posibilidad de probar endpoints desde la misma documentación.
- Tener **especificaciones disponibles** gracias a OpenAPI
- Escalabilidad. Si es necesario se integra muy bien con Redis, RabbitMQ para trabajar con colas de mensajes, generadores y procesadores
- Tipado con las ventajas que eso implica
- Inyección de dependencias. Hace que se haga muy escalable el desarrollo
- La experiencia ha sido genial, no he tenido bugs raros o cuellos de botella que no se han podido solucionar
- Posibilidad de integrar tecnologías complejas como Server Sent Events
- Performance 🧡🧡🧡



Muchas gracias!



Contacto

Blog: <https://juanrodriguezmonti.github.io/>

Linkedin: <https://www.linkedin.com/in/juanrodriguezmonti/>

Email: rmontijuan@gmail.com