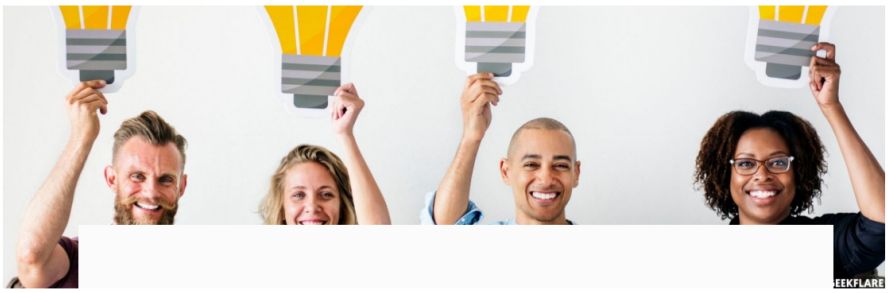


Top 5 Asynchronous Web Frameworks for Python

By Ankush Thakur on January 19, 2020

Posted in Development

> Asynchronous programming is a first-class citizen in Python now. If you're a web developer, there are amazing frameworks you can choose from!

As of writing, asynchronous is no more just a buzzword in the Python community. With the release of its asyncio library in 3.5 version, Python acknowledged the impact of Node.js on web development and introduce two new keywords into the language – `async` and `await` . This was a *very* big deal because the Python language is extremely wary of expanding the core syntax unless there's a pressing need, which only indicates how fundamentally important the Python developers considered the asynchronous capabilities.

As a result, floodgates of asynchronous programming were opened: libraries new and old started making use of the coroutines feature, asynchronous frameworks exploded in popularity, and new ones are still being written today. Performance at par with or better than Node.js's isn't unheard of, and unless your loading patterns involve plenty of CPU-heavy tasks, there's no reason why you can't make a few thousand requests per second.

But enough motivation!

Let's survey the current Python landscape and check out some of the top asynchronous frameworks.

1 Tornado

Surprisingly, Tornado isn't a new framework at all. Its initial release was in 2009 (exactly ten years ago, as of writing) and since then, its focus has been on providing rock-solid asynchronous programming with high concurrency.

Tornado

Tornado isn't a web framework fundamentally. It's a collection of asynchronous modules, which are also used to build the web framework module. More specifically, these modules are:

- Coroutines and other primitives (`tornado.gen` , `tornado.locks` , `tornado.queues` , etc.)
- Networking modules (`tornado.ioloop` , `tornado.iostream` , etc.)
- Asynchronous servers and clients (`tornado.httpserver` , `tornado.httpclient` , etc.)

These have been combined to produce the final framework modules: `tornado.web` , `tornado.routing` , `tornado.template` , etc.

```
import tornado.ioloop
import tornado.web

class MainHandler(tornado.web.RequestHandler):
    def get(self):
        self.write("Hello, world")

def make_app():
    return tornado.web.Application([
        (r"/", MainHandler),
    ])

if __name__ == "__main__":
    app = make_app()
    app.listen(8888)
    tornado.ioloop.IOLoop.current().start()
```

Tornado has a strong and committed following in the Python community and is used by experienced architects to build highly capable systems. It's a framework that has long had the answer to the problems of concurrency but perhaps didn't become mainstream as it doesn't support the WSGI standard and was too much of a buy-in (remember that the bulk of Python libraries are still synchronous).

2 Sanic

Sanic is a "modern" framework in the true sense of the word: it doesn't support Python version below 3.6, supports the simple and universal async/await syntax out of the box, and as a result, doesn't make you read loads of documentation and keep edge cases in your mind before you can write your first HTTP handler.

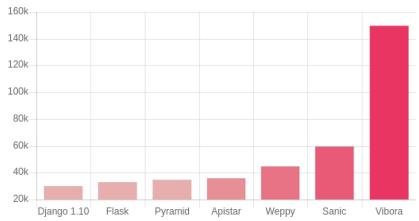
As a result, the resulting syntax is quite pleasant (in my opinion, at least); it resembles code you'd write with any other microframework (Flask, CherryPy, for example) with just a few `async` sprinkled in:

```
from sanic import Sanic
from sanic.response import json

app = Sanic()

@app.route("/")
async def test(request):
    return json({"hello": "world"})

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8000)
```

Sanic is arguably the most popular and most loved async framework in the Python world. It has almost all features that you'd want for your projects — routing, middleware, cookies, versioning, blueprints, class-based views, static files, streaming, sockets, etc. — and what it doesn't offer out of the box — templating, database support, file I/O, queues — can be added as there are just enough async libraries for these as of today.

3 Vibora

Vibora is a close cousin of Sanic, except that it's fixated on becoming the fastest Python web server out there. In fact, the very first visit of its website greets you with a framework comparison:



As you can see, Vibora claims to be several times faster than the classic frameworks and being more than twice as fast as Sanic, its nearest competitor. Of course, benchmarks are to be taken with a grain of salt. 🙂

Although in syntax and features, Vibora is comparable to Sanic (or maybe even slightly better as it bundles popular libraries and things like templating are available out of the box), I'd consider Sanic to be more mature as it's been around longer and has a bigger community.

```
from vibora import Vibora, JsonResponse

app = Vibora()

@app.route('/')
async def home():
    return JsonResponse({'hello': 'world'})

if __name__ == '__main__':
    app.run(host="0.0.0.0", port=8000)
```

If you're a performance junkie, though, Vibora might float your boat. That said, as of writing Vibora is underdoing a complete rewrite to become even faster, and the link to its performance version says it's under "heavy development." It's going to be a let down for those who picked up Vibora earlier and soon must face breaking changes, but hey, it's early days in Python async world, and nobody expects things to be stable.

4 Quart

If you enjoy developing in Flask but rue the lack of async support, you'll enjoy Quart a lot.

Quart is compliant with the ASGI standard, which is a successor to the famous WSGI standard and offers async support. The interesting thing about Quart is that it's not only similar to Flask but is actually compliant with the Flask API! The author of this framework wanted to preserve the Flask feel and just add async, WebSockets, and HTTP 2 support to it. As a result, you can learn Quart right from the Flask documentation, just keeping in mind that functions in Quart are asynchronous.

```
from quart import Quart

app = Quart(__name__)

@app.route('/')
async def hello():
    return 'hello'

app.run()
```

Feels (almost) exactly like Flask, doesn't it?!

Since Quart is an evolution of Flask, all the features inside Flask are available: routing, middleware, sessions, templating, blueprints, and so on. In fact, you can even use Flask extensions directly inside Quart. One catch is that Python 3.7+ is only supported, but, then, if you're not running the latest version of Python, maybe async isn't the right path. 😀

The documentation is really wanting if you don't have earlier experience with Flask, but I can recommend Quart as it's probably the only async framework nearing its 1.0 release soon.

5 FastAPI

The last (but most impressive) framework on this list is FastAPI. No, it's not an API-only framework; in fact, FastAPI seems to be the most feature-rich and documentation-rich framework that I came across when researching async Python frameworks.

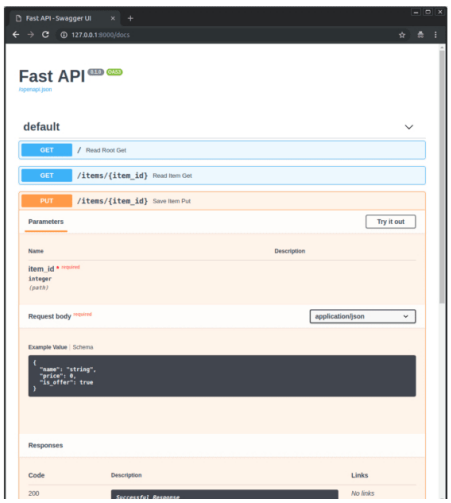
It's interesting to note that the framework author studied several other frameworks in-depth, from the contemporary ones like Django to modern ones like Sanic, as well as looking across technologies into NestJS (a Node.js, Typescript web framework). Their development philosophy and extensive comparisons can be read here.

The syntax is quite pleasant; one can even argue it's much more enjoyable than the other frameworks we've come across:

```
rom fastapi import FastAPI

app = FastAPI()

@app.get("/users/me")
async def read_user_me():
    return {"user_id": "the current user"}

@app.get("/users/{user_id}")
async def read_user(user_id: str):
    return {"user_id": user_id}
```

And now, the list of killer features that make FastAPI outshine other frameworks:

Automatic API doc generation: As soon as your endpoints have been written, you can play with the API using a standards-compliant UI. SwaggerUI, ReDoc, and others are supported.



The framework also does automatic data model documentation with JSON Schema.

Modern development: Yes, the word "modern" gets thrown around a lot, but I found FastAPI actually to walk its talk. Dependency Injection and type hinting are first-class citizens, enforcing not just good coding principles but preventing bugs and confusion in the long run.

Extensive documentation: I don't know about you, but I'm a total sucker for good documentation. And in this area, FastAPI wins hands-down. It has pages upon pages of docs explaining almost every little subtlety and "watch out!" moments for developers of all levels. I sense a clear "heart and soul" in the docs here, and the only comparison I can find is the Django docs (yes, FastAPI docs are that good!).

Beyond the basics: FastAPI has support for WebSockets, Streaming, as well as GraphQL, besides having all the traditional helpers like CORS, sessions, cookies, and so forth.

And what about the performance? Well, FastAPI is built on the amazing Starlette library, resulting in performance that matches Node, and in some cases, even Go! All in all, I really have the feeling that FastAPI is going to race ahead as the top async framework for Python.

Conclusion

A lot is going on in the Python async landscape these days. New frameworks are popping up, old ones are being rewritten, and libraries are being evolved to match async behavior. While Python has built-in support for an event loop and it's possible to make parts of your application async, you can choose to go all-in and build on one of the frameworks here. Just be sure to keep the long-term in mind: several of the Python async frameworks out there are in early stages and are being rapidly evolved, which is going to hurt your development process and raise business costs. Caution is key!

But all said and done; Python is production-ready to deliver light-out performance when it comes to web frameworks. If for so long you've been thinking of migrating to Node, now you don't need to! 🙂

Sounds cool? Master Python today!

TAGS: Python

More Great Reading on Geekflare

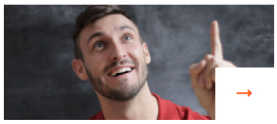
Introduction to YAML in Python for Beginners

A Guide to Flatten List & List of Lists in Python

Convert List to Dictionary in Python

How to Remove Last Character from Python String?

Check If List is Empty in Python With These 3 Easy Techniques



5 Cool things You can do with Python

Power Your Business

Choosing the right product and service is essential to run an online business. Here are some of the tools and services to help your business grow.

Netsparker

Netsparker uses the Proof-Based Scanning™ to automatically verify the identified vulnerabilities with proof of exploit, thus making it possible to scan thousands of web applications and generate actionable results within just hours.

Try Netsparker

Kinsta

Probably the best managed WordPress cloud platform to host small to enterprise sites. Kinsta leverages Google's low latency network infrastructure to deliver content faster. Free SSL, CDN, backup and a lot more with outstanding support. You'll love it.

Try Kinsta

Sucuri

A global CDN and cloud-based web application firewall for your website to supercharge the performance and secure from online threats. SUCURI WAF protects from OWASP top 10 vulnerabilities, brute force, DDoS, malware, and more.

Try Sucuri

+65 More Awesome Resources

Explore site tools

Browse articles

Discover trusted resources