Benchmarks

Independent TechEmpower benchmarks show **FastAPI** applications running under Uvicorn as one of the fastest Python frameworks available, only below Starlette and Uvicorn themselves (used internally by FastAPI). (*)

But when checking benchmarks and comparisons you should have the following in mind.

Benchmarks and speed

When you check the benchmarks, it is common to see several tools of different types compared as equivalent.

Specifically, to see Uvicorn, Starlette and FastAPI compared together (among many other tools).

The simpler the problem solved by the tool, the better performance it will get. And most of the benchmarks don't test the additional features provided by the tool.

The hierarchy is like:

- **Uvicorn**: an ASGI server
  - **Starlette**: (uses Uvicorn) a web microframework
    - **FastAPI**: (uses Starlette) an API microframework with several additional features for building APIs, with data validation, etc.
- **Uvicorn**:
  - Will have the best performance, as it doesn't have much extra code apart from the server itself.
  - You wouldn't write an application in Uvicorn directly. That would mean that your code would have to include more or less, at least, all the code provided by Starlette (or **FastAPI**). And if you did that, your final application would have the same overhead as having used a framework and minimizing your app code and bugs.
  - If you are comparing Uvicorn, compare it against Daphne, Hypercorn, uWSGI, etc. Application servers.
- **Starlette**:
  - Will have the next best performance, after Uvicorn. In fact, Starlette uses Uvicorn to run. So, it probably can only get "slower" than Uvicorn by having to execute more code.
  - But it provides you the tools to build simple web applications, with routing based on paths, etc.
  - If you are comparing Starlette, compare it against Sanic, Flask, Django, etc. Web frameworks (or microframeworks).
- **FastAPI**:
  - The same way that Starlette uses Uvicorn and cannot be faster than it, **FastAPI** uses Starlette, so it cannot be faster than it.

- FastAPI provides more features on top of Starlette. Features that you almost always need when building APIs, like data validation and serialization. And by using it, you get automatic documentation for free (the automatic documentation doesn't even add overhead to running applications, it is generated on startup).
- If you didn't use FastAPI and used Starlette directly (or another tool, like Sanic, Flask, Responder, etc) you would have to implement all the data validation and serialization yourself. So, your final application would still have the same overhead as if it was built using FastAPI. And in many cases, this data validation and serialization is the biggest amount of code written in applications.
- So, by using FastAPI you are saving development time, bugs, lines of code, and you would probably get the same performance (or better) you would if you didn't use it (as you would have to implement it all in your code).
- If you are comparing FastAPI, compare it against a web application framework (or set of tools) that provides data validation, serialization and documentation, like Flask-apispec, NestJS, Molten, etc. Frameworks with integrated automatic data validation, serialization and documentation.