Run a Server Manually - Uvicorn

The main thing you need to run a **FastAPI** application in a remote server machine is an ASGI server program like **Uvicorn**.

There are 3 main alternatives:

- Uvicorn: a high performance ASGI server.
- Hypercorn: an ASGI server compatible with HTTP/2 and Trio among other features.
- Daphne: the ASGI server built for Django Channels.

Server Machine and Server Program

There's a small detail about names to have in mind. 💡

The word "**server**" is commonly used to refer to both the remote/cloud computer (the physical or virtual machine) and also the program that is running on that machine (e.g. Uvicorn).

Just have that in mind when you read "server" in general, it could refer to one of those two things.

When referring to the remote machine, it's common to call it **server**, but also **machine**, **VM** (virtual machine), **node**. Those all refer to some type of remote machine, normally running Linux, where you run programs.

Install the Server Program

You can install an ASGI compatible server with:

=== "Uvicorn"

```
* <a href="https://www.uvicorn.org/" class="external-link"
target="_blank">Uvicorn</a>, a lightning-fast ASGI server, built on uvloop and
httptools.

<div class="termy">

```console
$ pip install "uvicorn[standard]"

---> 100%
```

</div>

!!! tip
    By adding the `standard`, Uvicorn will install and use some recommended extra
dependencies.

    That including `uvloop`, the high-performance drop-in replacement for `asyncio`,
that provides the big concurrency performance boost.
```

=== "Hypercorn"

```
* <a href="https://gitlab.com/pgjones/hypercorn" class="external-link"
target="_blank">Hypercorn</a>, an ASGI server also compatible with HTTP/2.

<div class="termy">

```console
$ pip install hypercorn

---> 100%
```

</div>

...or any other ASGI server.
```

Run the Server Program

You can then your application the same way you have done in the tutorials, but without the `--reload` option, e.g.:

=== "Uvicorn"

```
<div class="termy">

```console
$ uvicorn main:app --host 0.0.0.0 --port 80

<span style="color: green;">INFO</span>:     Uvicorn running on http://0.0.0.0:80
(Press CTRL+C to quit)
```

</div>
```

=== "Hypercorn"

```
<div class="termy">

```console
$ hypercorn main:app --bind 0.0.0.0:80

Running on 0.0.0.0:8080 over http (CTRL + C to quit)
```

</div>
```

!!! warning Remember to remove the `--reload` option if you were using it.

```
The `--reload` option consumes much more resources, is more unstable, etc.

It helps a lot during **development**, but you **shouldn't** use it in **production**.
```

Hypercorn with Trio

Starlette and **FastAPI** are based on AnyIO, which makes them compatible with both Python's standard library asyncio and Trio.

Nevertheless, Uvicorn is currently only compatible with asyncio, and it normally uses `uvloop`, the high-performance drop-in replacement for `asyncio`.

But if you want to directly use **Trio**, then you can use **Hypercorn** as it supports it. ✨

Install Hypercorn with Trio

First you need to install Hypercorn with Trio support:

```
$ pip install "hypercorn[trio]"
---> 100%
```

Run with Trio

Then you can pass the command line option `--worker-class` with the value `trio`:

```
$ hypercorn main:app --worker-class trio
```

And that will start Hypercorn with your app using Trio as the backend.

Now you can use Trio internally in your app. Or even better, you can use AnyIO, to keep your code compatible with both Trio and asyncio. 🎉

Deployment Concepts

These examples run the server program (e.g Uvicorn), starting **a single process**, listening on all the IPs (`0.0.0.0`) on a predefined port (e.g. `80`).

This is the basic idea. But you will probably want to take care of some additional things, like:

- Security - HTTPS
- Running on startup
- Restarts
- Replication (the number of processes running)
- Memory
- Previous steps before starting

I'll tell you more about each of these concepts, how to think about them, and some concrete examples with strategies to handle them in the next chapters. 🚀