Behind a Proxy

In some situations, you might need to use a **proxy** server like Traefik or Nginx with a configuration that adds an extra path prefix that is not seen by your application.

In these cases you can use `root_path` to configure your application.

The `root_path` is a mechanism provided by the ASGI specification (that FastAPI is built on, through Starlette).

The `root_path` is used to handle these specific cases.

And it's also used internally when mounting sub-applications.

Proxy with a stripped path prefix

Having a proxy with a stripped path prefix, in this case, means that you could declare a path at `/app` in your code, but then, you add a layer on top (the proxy) that would put your **FastAPI** application under a path like `/api/v1`.

In this case, the original path `/app` would actually be served at `/api/v1/app`.

Even though all your code is written assuming there's just `/app`.

And the proxy would be **"stripping"** the **path prefix** on the fly before transmitting the request to Uvicorn, keep your application convinced that it is serving at `/app`, so that you don't have to update all your code to include the prefix `/api/v1`.

Up to here, everything would work as normally.

But then, when you open the integrated docs UI (the frontend), it would expect to get the OpenAPI schema at `/openapi.json`, instead of `/api/v1/openapi.json`.

So, the frontend (that runs in the browser) would try to reach `/openapi.json` and wouldn't be able to get the OpenAPI schema.

Because we have a proxy with a path prefix of `/api/v1` for our app, the frontend needs to fetch the OpenAPI schema at `/api/v1/openapi.json`.

```
graph LR

browser("Browser")
proxy["Proxy on http://0.0.0.0:9999/api/v1/app"]
server["Server on http://127.0.0.1:8000/app"]

browser --> proxy
proxy --> server
```

!!! tip The IP `0.0.0.0` is commonly used to mean that the program listens on all the IPs available in that machine/server.

The docs UI would also need the OpenAPI schema to declare that this API `server` is located at `/api/v1` (behind the proxy). For example:

```
{
    "openapi": "3.0.2",
    // More stuff here
    "servers": [
        {
            "url": "/api/v1"
        }
    ],
    "paths": {
            // More stuff here
    }
}
```

In this example, the "Proxy" could be something like **Traefik**. And the server would be something like **Uvicorn**, running your FastAPI application.

Providing the `root_path`

To achieve this, you can use the command line option `--root-path` like:

```
$ uvicorn main:app --root-path /api/v1

<span style="color: green;">INFO</span>:     Uvicorn running on
http://127.0.0.1:8000 (Press CTRL+C to quit)
```

If you use Hypercorn, it also has the option `--root-path`.

!!! note "Technical Details" The ASGI specification defines a `root_path` for this use case.

```
And the `--root-path` command line option provides that `root_path`.
```

Checking the current `root_path`

You can get the current `root_path` used by your application for each request, it is part of the `scope` dictionary (that's part of the ASGI spec).

Here we are including it in the message just for demonstration purposes.

```
{!../../../docs_src/behind_a_proxy/tutorial001.py!}
```

Then, if you start Uvicorn with:

```
$ uvicorn main:app --root-path /api/v1

<span style="color: green;">INFO</span>:     Uvicorn running on
http://127.0.0.1:8000 (Press CTRL+C to quit)
```

The response would be something like:

```
{
    "message": "Hello World",
    "root_path": "/api/v1"
}
```

Setting the `root_path` in the FastAPI app

Alternatively, if you don't have a way to provide a command line option like `--root-path` or equivalent, you can set the `root_path` parameter when creating your FastAPI app:

```
{!../../../docs_src/behind_a_proxy/tutorial002.py!}
```

Passing the `root_path` to `FastAPI` would be the equivalent of passing the `--root-path` command line option to Uvicorn or Hypercorn.

About `root_path`

Have in mind that the server (Uvicorn) won't use that `root_path` for anything else than passing it to the app.

But if you go with your browser to http://127.0.0.1:8000/app you will see the normal response:

```
{
    "message": "Hello World",
    "root_path": "/api/v1"
}
```

So, it won't expect to be accessed at `http://127.0.0.1:8000/api/v1/app`.

Uvicorn will expect the proxy to access Uvicorn at `http://127.0.0.1:8000/app`, and then it would be the proxy's responsibility to add the extra `/api/v1` prefix on top.

About proxies with a stripped path prefix

Have in mind that a proxy with stripped path prefix is only one of the ways to configure it.

Probably in many cases the default will be that the proxy doesn't have a stripped path prefix.

In a case like that (without a stripped path prefix), the proxy would listen on something like `https://myawesomeapp.com`, and then if the browser goes to `https://myawesomeapp.com/api/v1/app` and your server (e.g. Uvicorn) listens on `http://127.0.0.1:8000` the proxy (without a stripped path prefix) would access Uvicorn at the same path: `http://127.0.0.1:8000/api/v1/app`.

Testing locally with Traefik

You can easily run the experiment locally with a stripped path prefix using Traefik.

Download Traefik, it's a single binary, you can extract the compressed file and run it directly from the terminal.

Then create a file `traefik.toml` with:

```
[entryPoints]
  [entryPoints.http]
    address = ":9999"

[providers]
  [providers.file]
    filename = "routes.toml"
```

This tells Traefik to listen on port 9999 and to use another file `routes.toml`.

!!! tip We are using port 9999 instead of the standard HTTP port 80 so that you don't have to run it with admin (`sudo`) privileges.

Now create that other file `routes.toml`:

```
[http]
  [http.middlewares]

    [http.middlewares.api-stripprefix.stripPrefix]
      prefixes = ["/api/v1"]

  [http.routers]

    [http.routers.app-http]
      entryPoints = ["http"]
      service = "app"
      rule = "PathPrefix(`/api/v1`)"
      middlewares = ["api-stripprefix"]

  [http.services]

    [http.services.app]
      [http.services.app.loadBalancer]
        [[http.services.app.loadBalancer.servers]]
          url = "http://127.0.0.1:8000"
```

This file configures Traefik to use the path prefix `/api/v1`.

And then it will redirect its requests to your Uvicorn running on `http://127.0.0.1:8000`.

Now start Traefik:

```
$ ./traefik --configFile=traefik.toml

INFO[0000] Configuration loaded from file: /home/user/awesomeapi/traefik.toml
```

And now start your app with Uvicorn, using the `--root-path` option:

```
$ uvicorn main:app --root-path /api/v1
```

```
<span style="color: green;">INFO</span>:     Uvicorn running on
http://127.0.0.1:8000 (Press CTRL+C to quit)
```

Check the responses

Now, if you go to the URL with the port for Uvicorn: http://127.0.0.1:8000/app, you will see the normal response:

```
{
    "message": "Hello World",
    "root_path": "/api/v1"
}
```

!!! tip Notice that even though you are accessing it at `http://127.0.0.1:8000/app` it shows the `root_path` of `/api/v1`, taken from the option `--root-path`.

And now open the URL with the port for Traefik, including the path prefix: http://127.0.0.1:9999/api/v1/app.

We get the same response:

```
{
    "message": "Hello World",
    "root_path": "/api/v1"
}
```

but this time at the URL with the prefix path provided by the proxy: `/api/v1`.

Of course, the idea here is that everyone would access the app through the proxy, so the version with the path prefix `/app/v1` is the "correct" one.

And the version without the path prefix (`http://127.0.0.1:8000/app`), provided by Uvicorn directly, would be exclusively for the *proxy* (Traefik) to access it.

That demonstrates how the Proxy (Traefik) uses the path prefix and how the server (Uvicorn) uses the `root_path` from the option `--root-path`.

Check the docs UI

But here's the fun part. ✨

The "official" way to access the app would be through the proxy with the path prefix that we defined. So, as we would expect, if you try the docs UI served by Uvicorn directly, without the path prefix in the URL, it won't work, because it expects to be accessed through the proxy.

You can check it at http://127.0.0.1:8000/docs:

But if we access the docs UI at the "official" URL using the proxy with port `9999`, at `/api/v1/docs`, it works correctly! 🎉

You can check it at http://127.0.0.1:9999/api/v1/docs:

Right as we wanted it. ✔

This is because FastAPI uses this `root_path` to create the default `server` in OpenAPI with the URL provided by `root_path`.

Additional servers

!!! warning This is a more advanced use case. Feel free to skip it.

By default, **FastAPI** will create a `server` in the OpenAPI schema with the URL for the `root_path`.

But you can also provide other alternative `servers`, for example if you want *the same* docs UI to interact with a staging and production environments.

If you pass a custom list of `servers` and there's a `root_path` (because your API lives behind a proxy), **FastAPI** will insert a "server" with this `root_path` at the beginning of the list.

For example:

```
{!../../../docs_src/behind_a_proxy/tutorial003.py!}
```

Will generate an OpenAPI schema like:

```
{
    "openapi": "3.0.2",
    // More stuff here
    "servers": [
        {
            "url": "/api/v1"
        },
        {
            "url": "https://stag.example.com",
            "description": "Staging environment"
        },
        {
            "url": "https://prod.example.com",
            "description": "Production environment"
        }
    ],
    "paths": {
            // More stuff here
    }
}
```

!!! tip Notice the auto-generated server with a `url` value of `/api/v1`, taken from the `root_path`.

In the docs UI at http://127.0.0.1:9999/api/v1/docs it would look like:

!!! tip The docs UI will interact with the server that you select.

Disable automatic server from `root_path`

If you don't want **FastAPI** to include an automatic server using the `root_path`, you can use the parameter `root_path_in_servers=False`:

```
{!../../../docs_src/behind_a_proxy/tutorial004.py!}
```

and then it won't include it in the OpenAPI schema.

Mounting a sub-application

If you need to mount a sub-application (as described in Sub Applications - Mounts{.internal-link target=_blank}) while also using a proxy with `root_path`, you can do it normally, as you would expect.

FastAPI will internally use the `root_path` smartly, so it will just work. ✨