Implantação

Implantar uma aplicação **FastAPI** é relativamente fácil.

Existem vários modos de realizar o *deploy* dependendo de seu caso de uso específico e as ferramentas que você utiliza.

Você verá mais sobre alguns modos de fazer o *deploy* nas próximas seções.

Versões do FastAPI

FastAPI já está sendo utilizado em produção em muitas aplicações e sistemas. E a cobertura de teste é mantida a 100%. Mas seu desenvolvimento continua andando rapidamente.

Novos recursos são adicionados frequentemente, *bugs* são corrigidos regularmente, e o código está continuamente melhorando.

É por isso que as versões atuais estão ainda no `0.x.x`, isso reflete que cada versão poderia ter potencialmente alterações que podem quebrar. Isso segue as convenções de Versionamento Semântico.

Você pode criar aplicações para produção com **FastAPI** bem agora (e você provavelmente já faça isso por um tempo), você tem que ter certeza de utilizar uma versão que funcione corretamente com o resto do seu código.

Anote sua versão `fastapi`

A primeira coisa que você deve fazer é "fixar" a versão do **FastAPI** que está utilizando para a última versão específica que você sabe que funciona corretamente para a sua aplicação.

Por exemplo, vamos dizer que você esteja utilizando a versão `0.45.0` no seu *app*.

Se você usa um arquivo `requirements.txt`, dá para especificar a versão assim:

```
fastapi==0.45.0
```

isso significa que você pode usar exatamente a versão `0.45.0`.

Ou você poderia fixar assim:

```
fastapi>=0.45.0,<0.46.0
```

o que significa que você pode usar as versões `0.45.0` ou acima, mas menor que `0.46.0`. Por exemplo, a versão `0.45.2` poderia ser aceita.

Se você usa qualquer outra ferramenta para gerenciar suas instalações, como Poetry, Pipenv ou outro, todos terão um modo que você possa usar para definir versões específicas para seus pacotes.

Versões disponíveis

Você pode ver as versões disponíveis (por exemplo, para verificar qual é a versão atual) nas Notas de Lançamento{.internal-link target=_blank}.

Sobre as versões

Seguindo as convenções do Versionamento Semântico, qualquer versão abaixo de `1.0.0` pode potencialmente adicionar mudanças que quebrem.

FastAPI também segue a convenção que qualquer versão de "*PATCH*" seja para ajustes de *bugs* e mudanças que não quebrem a aplicação.

!!! tip O "*PATCH*" é o último número, por exemplo, em `0.2.3`, a versão do *PATCH* é `3`.

Então, você poderia ser capaz de fixar para uma versão como:

```
fastapi>=0.45.0,<0.46.0
```

Mudanças que quebram e novos recursos são adicionados em versões "*MINOR*".

!!! tip O "*MINOR*" é o número do meio, por exemplo, em `0.2.3`, a versão *MINOR* é `2`.

Atualizando as versões FastAPI

Você pode adicionar testes em sua aplicação.

Com o **FastAPI** é muito fácil (graças ao Starlette), verifique a documentação: Testando{.internal-link target=_blank}

Após você ter os testes, então você pode fazer o *upgrade* da versão **FastAPI** para uma mais recente, e ter certeza que todo seu código esteja funcionando corretamente rodando seus testes.

Se tudo estiver funcionando, ou após você fazer as alterações necessárias, e todos seus testes estiverem passando, então você poderá fixar o `fastapi` para a versão mais recente.

Sobre Starlette

Você não deve fixar a versão do `starlette`.

Versões diferentes do **FastAPI** irão utilizar uma versão mais nova específica do Starlette.

Então, você pode deixar que o **FastAPI** use a versão correta do Starlette.

Sobre Pydantic

Pydantic inclui os testes para **FastAPI** em seus próprios testes, então novas versões do Pydantic (acima de `1.0.0`) são sempre compatíveis com FastAPI.

Você pode fixar o Pydantic para qualquer versão acima de `1.0.0` e abaixo de `2.0.0` que funcionará.

Por exemplo:

```
pydantic>=1.2.0,<2.0.0
```

Docker

Nessa seção você verá instruções e *links* para guias de saber como:

- Fazer uma imagem/container da sua aplicação **FastAPI** com máxima performance. Em aproximadamente **5 min**.
- (Opcionalmente) entender o que você, como desenvolvedor, precisa saber sobre HTTPS.

- Inicializar um *cluster* Docker Swarm Mode com HTTPS automático, mesmo em um simples servidor de $5 dólares/mês. Em aproximadamente **20 min**.
- Gere e implante uma aplicação **FastAPI** completa, usando seu *cluster* Docker Swarm, com HTTPS etc. Em aproxiamadamente **10 min**.

Você pode usar **Docker** para implantação. Ele tem várias vantagens como segurança, replicabilidade, desenvolvimento simplificado etc.

Se você está usando Docker, você pode utilizar a imagem Docker oficial:

tiangolo/uvicorn-gunicorn-fastapi

Essa imagem tem um mecanismo incluído de "auto-ajuste", para que você possa apenas adicionar seu código e ter uma alta performance automaticamente. E sem fazer sacrifícios.

Mas você pode ainda mudar e atualizar todas as configurações com variáveis de ambiente ou arquivos de configuração.

!!! tip Para ver todas as configurações e opções, vá para a página da imagem do Docker: tiangolo/uvicorn-gunicorn-fastapi.

Crie um `Dockerfile`

- Vá para o diretório de seu projeto.
- Crie um `Dockerfile` com:

```
FROM tiangolo/uvicorn-gunicorn-fastapi:python3.7

COPY ./app /app
```

Grandes aplicações

Se você seguiu a seção sobre criação de Grandes Aplicações com Múltiplos Arquivos{.internal-link target=_blank}, seu `Dockerfile` poderia parecer como:

```
FROM tiangolo/uvicorn-gunicorn-fastapi:python3.7

COPY ./app /app/app
```

Raspberry Pi e outras arquiteturas

Se você estiver rodando Docker em um Raspberry Pi (que possui um processador ARM) ou qualquer outra arquitetura, você pode criar um `Dockerfile` do zero, baseado em uma imagem base Python (que é multi-arquitetural) e utilizar Uvicorn sozinho.

Nesse caso, seu `Dockerfile` poderia parecer assim:

```
FROM python:3.7

RUN pip install fastapi uvicorn

EXPOSE 80
```

```
COPY ./app /app

CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "80"]
```

Crie o código FastAPI

- Crie um diretório `app` e entre nele.
- Crie um arquivo `main.py` com:

```
from fastapi import FastAPI

app = FastAPI()


@app.get("/")
def read_root():
    return {"Hello": "World"}


@app.get("/items/{item_id}")
def read_item(item_id: int, q: str = None):
    return {"item_id": item_id, "q": q}
```

- Você deve ter uma estrutura de diretórios assim:

```
.
├── app
│   └── main.py
└── Dockerfile
```

Construa a imagem Docker

- Vá para o diretório do projeto (onde seu `Dockerfile` está, contendo seu diretório `app`.
- Construa sua imagem FastAPI:

```
$ docker build -t myimage .

---> 100%
```

Inicie o container Docker

- Rode um container baseado em sua imagem:

```
$ docker run -d --name mycontainer -p 80:80 myimage
```

Agora você tem um servidor FastAPI otimizado em um container Docker. Auto-ajustado para seu servidor atual (e número de núcleos de CPU).

Verifique

Você deve ser capaz de verificar na URL de seu container Docker, por exemplo: http://192.168.99.100/items/5?q=somequery ou http://127.0.0.1/items/5?q=somequery (ou equivalente, usando seu *host* Docker).

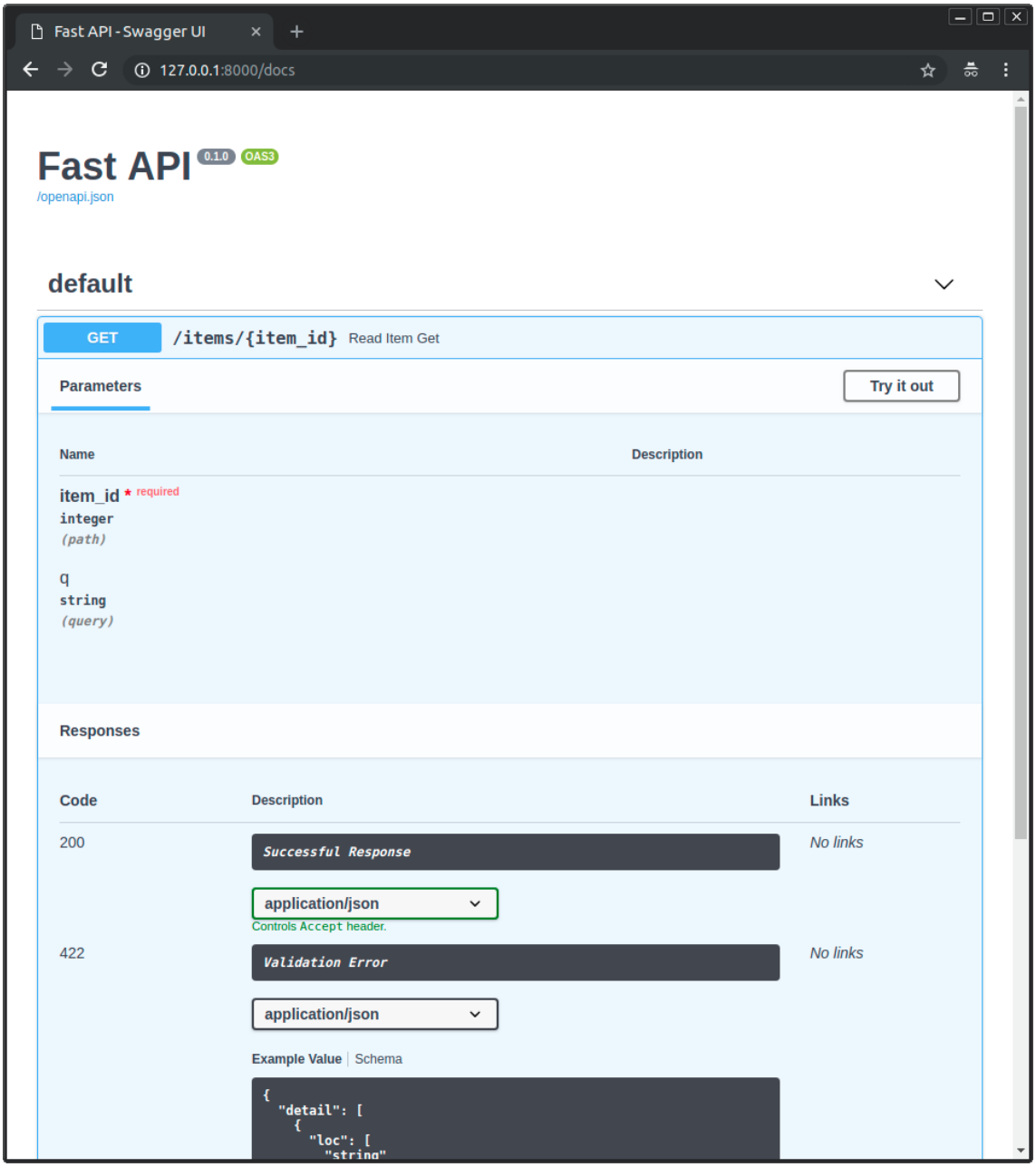
Você verá algo como:

```
{"item_id": 5, "q": "somequery"}
```

API interativa de documetação

Agora você pode ir para http://192.168.99.100/docs ou http://127.0.0.1/docs (ou equivalente, usando seu *host* Docker).

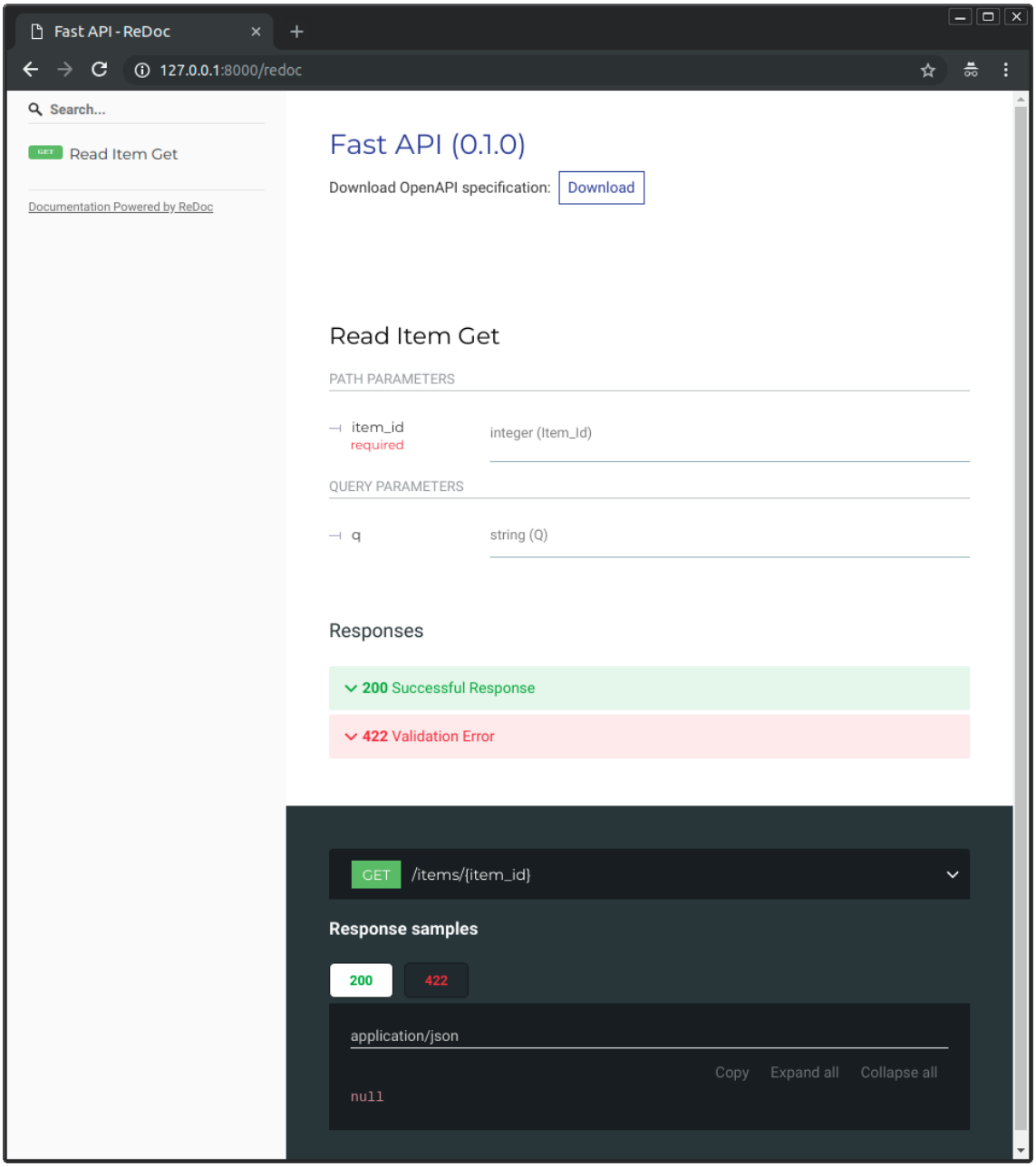
Você verá a API interativa de documentação (fornecida por Swagger UI):



APIs alternativas de documentação

E você pode também ir para http://192.168.99.100/redoc ou http://127.0.0.1/redoc (ou equivalente, usando seu *host* Docker).

Você verá a documentação automática alternativa (fornecida por ReDoc):



HTTPS

Sobre HTTPS

É fácil assumir que HTTPS seja algo que esteja apenas "habilitado" ou não.

Mas ele é um pouquinho mais complexo do que isso.

!!! tip Se você está com pressa ou não se importa, continue na próxima seção com instruções passo a passo para configurar tudo.

Para aprender o básico de HTTPS, pela perspectiva de um consumidor, verifique https://howhttps.works/.

Agora, pela perspectiva de um desenvolvedor, aqui estão algumas coisas para se ter em mente enquanto se pensa sobre HTTPS:

- Para HTTPS, o servidor precisa ter "certificados" gerados por terceiros.
    - Esses certificados são na verdade adquiridos por terceiros, não "gerados".
- Certificados tem um prazo de uso.
    - Eles expiram.
    - E então eles precisam ser renovados, adquiridos novamente por terceiros.
- A encriptação da conexão acontece no nível TCP.
    - TCP é uma camada abaixo do HTTP.
    - Então, o controle de certificado e encriptação é feito antes do HTTP.
- TCP não conhece nada sobre "domínios". Somente sobre endereços IP.
    - A informação sobre o domínio requisitado vai nos dados HTTP.
- Os certificados HTTPS "certificam" um certo domínio, mas o protocolo e a encriptação acontecem no nível TCP, antes de saber qual domínio está sendo lidado.
- Por padrão, isso significa que você pode ter somente um certificado HTTPS por endereço IP.
    - Não importa quão grande é seu servidor ou quão pequena cada aplicação que você tenha possar ser.
    - No entanto, existe uma solução para isso.
- Existe uma extensão para o protocolo TLS (o que controla a encriptação no nível TCP, antes do HTTP) chamada SNI.
    - Essa extensão SNI permite um único servidor (com um único endereço IP) a ter vários certificados HTTPS e servir múltiplas aplicações/domínios HTTPS.
    - Para que isso funcione, um único componente (programa) rodando no servidor, ouvindo no endereço IP público, deve ter todos os certificados HTTPS no servidor.
- Após obter uma conexão segura, o protocolo de comunicação ainda é HTTP.
    - O conteúdo está encriptado, mesmo embora ele esteja sendo enviado com o protocolo HTTP.

É uma prática comum ter um servidor HTTP/programa rodando no servidor (a máquina, *host* etc.) e gerenciar todas as partes HTTP: enviar as requisições HTTP decriptadas para a aplicação HTTP rodando no mesmo servidor (a aplicação **FastAPI**, nesse caso), pega a resposta HTTP da aplicação, encripta utilizando o certificado apropriado e enviando de volta para o cliente usando HTTPS. Esse servidor é frequentemente chamado TLS *Termination Proxy*.

Vamos encriptar

Antes de encriptar, esses certificados HTTPS foram vendidos por terceiros de confiança.

O processo para adquirir um desses certificados costumava ser chato, exigia muita papelada e eram bem caros.

Mas então *Let's Encrypt* foi criado.

É um projeto da Fundação Linux.Ele fornece certificados HTTPS de graça. De um jeito automatizado. Esses certificados utilizam todos os padrões de segurança criptográfica, e tem vida curta (cerca de 3 meses), para que a segurança seja melhor devido ao seu curto período de vida.

Os domínios são seguramente verificados e os certificados são gerados automaticamente. Isso também permite automatizar a renovação desses certificados.

A idéia é automatizar a aquisição e renovação desses certificados, para que você possa ter um HTTPS seguro, grátis, para sempre.

Traefik

Traefik é um *proxy* reverso / *load balancer* de alta performance. Ele pode fazer o trabalho do "*TLS Termination Proxy*" (à parte de outros recursos).

Ele tem integração com *Let's Encrypt*. Assim, ele pode controlar todas as partes HTTPS, incluindo a aquisição e renovação de certificados.

Ele também tem integrações com Docker. Assim, você pode declarar seus domínios em cada configuração de aplicação e leitura dessas configurações, gerando os certificados HTTPS e servindo o HTTPS para sua aplicação automaticamente, sem exigir qualquer mudança em sua configuração.

Com essas ferramentas e informações, continue com a próxima seção para combinar tudo.

Cluster de Docker Swarm Mode com Traefik e HTTPS

Você pode ter um *cluster* de Docker Swarm Mode configurado em minutos (cerca de 20) com o Traefik controlando HTTPS (incluindo aquisição e renovação de certificados).

Utilizando o Docker Swarm Mode, você pode iniciar com um "*cluster*" de apenas uma máquina (que pode até ser um servidor por 5 dólares / mês) e então você pode aumentar conforme a necessidade adicionando mais servidores.

Para configurar um *cluster* Docker Swarm Mode com Traefik controlando HTTPS, siga essa orientação:

Docker Swarm Mode and Traefik for an HTTPS cluster

Faça o *deploy* de uma aplicação FastAPI

O jeito mais fácil de configurar tudo pode ser utilizando o Gerador de Projetos **FastAPI**{.internal-link target=_blank}.

Ele é designado para ser integrado com esse *cluster* Docker Swarm com Traefik e HTTPS descrito acima.

Você pode gerar um projeto em cerca de 2 minutos.

O projeto gerado tem instruções para fazer o *deploy*, fazendo isso leva outros 2 minutos.

Alternativamente, faça o *deploy* FastAPI sem Docker

Você pode fazer o *deploy* do **FastAPI** diretamente sem o Docker também.

Você apenas precisa instalar um servidor ASGI compatível como:

=== "Uvicorn"

```
* <a href="https://www.uvicorn.org/" class="external-link"
target="_blank">Uvicorn</a>, um servidor ASGI peso leve, construído sobre uvloop e
httptools.

<div class="termy">

```console
$ pip install uvicorn[standard]

---> 100%
```
```

```
</div>
```

=== "Hypercorn"

```
* <a href="https://gitlab.com/pgjones/hypercorn" class="external-link"
target="_blank">Hypercorn</a>, um servidor ASGI também compatível com HTTP/2.

<div class="termy">

```console
$ pip install hypercorn

---> 100%
```

</div>

...ou qualquer outro servidor ASGI.
```

E rode sua applicação do mesmo modo que você tem feito nos tutoriais, mas sem a opção `--reload`, por exemplo:

=== "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>
```

Você deve querer configurar mais algumas ferramentas para ter certeza que ele seja reinicializado automaticamante se ele parar.

Você também deve querer instalar Gunicorn e utilizar ele como um gerenciador para o Uvicorn, ou usar Hypercorn com múltiplos *workers*.

Tenha certeza de ajustar o número de *workers* etc.

Mas se você estiver fazendo tudo isso, você pode apenas usar uma imagem Docker que fará isso automaticamente.