Dockerを使用したデプロイ

このセクションでは以下の使い方の紹介とガイドへのリンクが確認できます:

- **5分**程度で、**FastAPI** のアプリケーションを、パフォーマンスを最大限に発揮するDockerイメージ (コンテナ)にする。
- (オプション) 開発者として必要な範囲でHTTPSを理解する。
- **20分**程度で、自動的なHTTPS生成とともにDockerのSwarmモード クラスタをセットアップする (月5ドルのシンプルなサーバー上で)。
- **10分**程度で、DockerのSwarmモード クラスタを使って、HTTPSなどを使用した完全な**FastAPI** アプリケーションの作成とデプロイ。

デプロイのために、**Docker** を利用できます。セキュリティ、再現性、開発のシンプルさなどに利点があります。

Dockerを使う場合、公式のDockerイメージが利用できます:

tiangolo/uvicorn-gunicorn-fastapi

このイメージは「自動チューニング」機構を含んでいます。犠牲を払うことなく、ただコードを加えるだけで自動的に高パフォーマンスを実現できます。

ただし、環境変数や設定ファイルを使って全ての設定の変更や更新を行えます。

!!! tip "豆知識" 全ての設定とオプションを確認するには、Dockerイメージページを開いて下さい: tiangolo/uvicorn-gunicorn-fastapi.

`Dockerfile` の作成

- プロジェクトディレクトリへ移動。
- 以下の `Dockerfile` を作成:

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

COPY ./app /app
```

より大きなアプリケーション

Bigger Applications with Multiple Files{.internal-link target=_blank} セクションに倣う場合は、 `Dockerfile` は上記の代わりに、以下の様になるかもしれません:

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

COPY ./app /app/app
```

Raspberry Piなどのアーキテクチャ

Raspberry Pi (ARMプロセッサ搭載)やそれ以外のアーキテクチャでDockerが作動している場合、(マルチアーキテクチャである) Pythonベースイメージを使って、一から `Dockerfile` を作成し、Uvicornを単体で使用できます。

この場合、 `Dockerfile` は以下の様になるかもしれません:

```
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"]
```

FastAPI コードの作成

- `app` ディレクトリを作成し、移動。
- 以下の `main.py` ファイルを作成:

```
from typing import Optional

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: Optional[str] = None):
    return {"item_id": item_id, "q": q}
```

- ここでは、以下の様なディレクトリ構造になっているはずです:

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

Dockerイメージをビルド

- プロジェクトディレクトリ (`app` ディレクトリを含んだ、 `Dockerfile` のある場所) へ移動
- FastAPIイメージのビルド:

```
$ docker build -t myimage .

---> 100%
```

Dockerコンテナを起動

- 用意したイメージを基にしたコンテナの起動:

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

これで、Dockerコンテナ内に最適化されたFastAPIサーバが動作しています。使用しているサーバ (そしてCPUコア数) に沿った自動チューニングが行われています。

確認

DockerコンテナのURLで確認できるはずです。例えば: http://192.168.99.100/items/5?q=somequery や http://127.0.0.1/items/5?q=somequery (もしくはDockerホストを使用したこれらと同等のもの)。

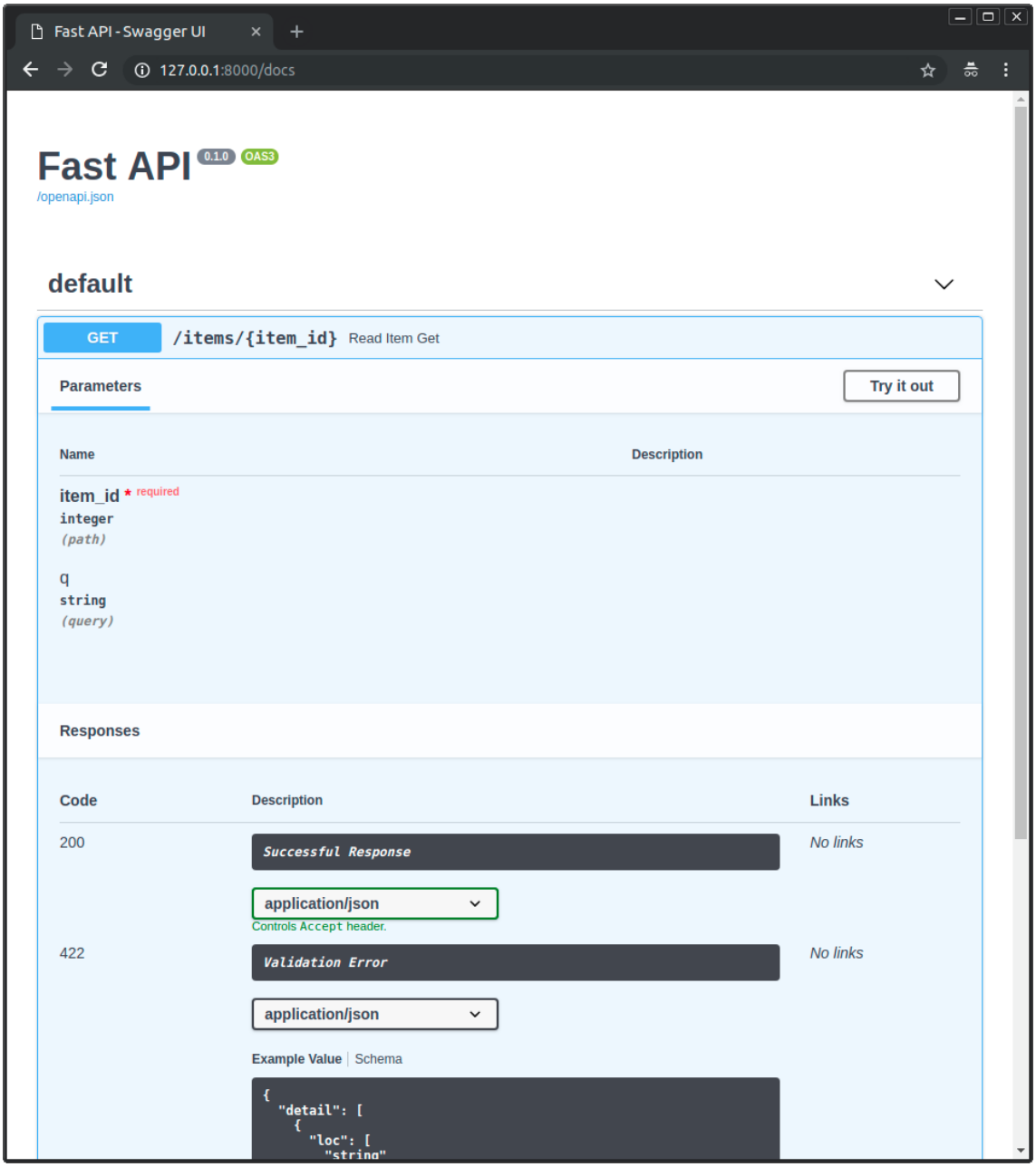
以下の様なものが返されます:

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

対話的APIドキュメント

ここで、http://192.168.99.100/docs や http://127.0.0.1/docs (もしくはDockerホストを使用したこれらと同等のもの) を開いて下さい。

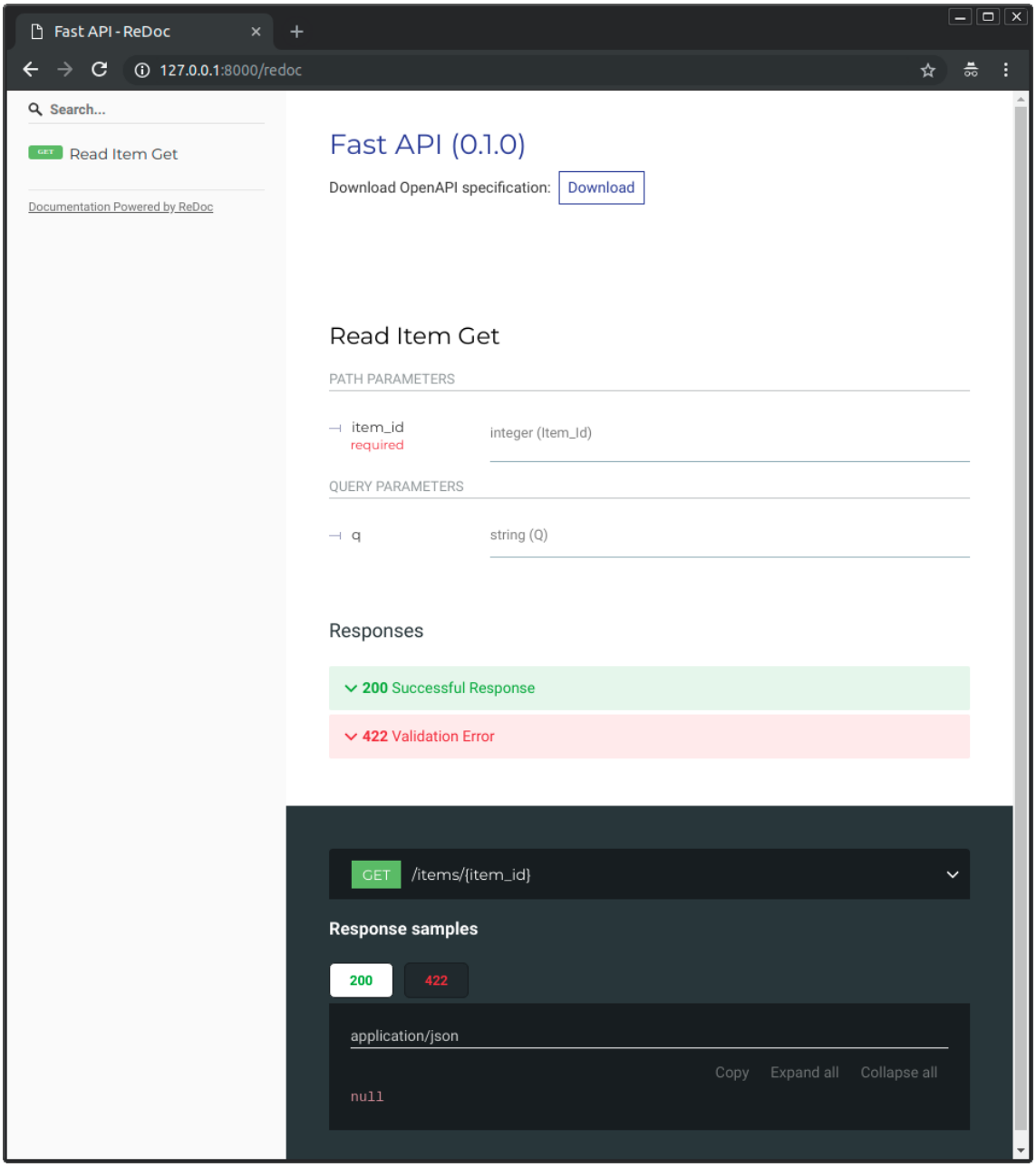
自動生成された対話的APIドキュメントが確認できます (Swagger UIによって提供されます):



その他のAPIドキュメント

また同様に、http://192.168.99.100/redoc や http://127.0.0.1/redoc (もしくはDockerホストを使用したこれらと同等のもの) を開いて下さい。

他の自動生成された対話的なAPIドキュメントが確認できます (ReDocによって提供されます):



Traefik

Traefikは、高性能なリバースプロキシ/ロードバランサーです。「TLSターミネーションプロキシ」ジョブを実行できます（他の機能と切り離して）。

Let's Encryptと統合されています。そのため、証明書の取得と更新を含むHTTPSに関するすべての処理を実行できます。

また、Dockerとも統合されています。したがって、各アプリケーション構成でドメインを宣言し、それらの構成を読み取って、HTTPS証明書を生成し、構成に変更を加えることなく、アプリケーションにHTTPSを自動的に提供できます。

次のセクションに進み、この情報とツールを使用して、すべてを組み合わせます。

TraefikとHTTPSを使用したDocker Swarmモードのクラスタ

HTTPSを処理する（証明書の取得と更新を含む）Traefikを使用して、Docker Swarmモードのクラスタを数分（20分程度）でセットアップできます。

Docker Swarmモードを使用することで、1台のマシンの「クラスタ」から開始でき（1か月あたり5ドルのサーバーでもできます）、後から必要なだけサーバーを拡張できます。

TraefikおよびHTTPS処理を備えたDocker Swarm Modeクラスターをセットアップするには、次のガイドに従います:

Docker Swarm Mode and Traefik for an HTTPS cluster

FastAPIアプリケーションのデプロイ

すべてを設定するための最も簡単な方法は、**FastAPI** Project Generators{.internal-link target=_blank}を使用することでしょう。

上述したTraefikとHTTPSを備えたDocker Swarm クラスタが統合されるように設計されています。

2分程度でプロジェクトが生成されます。

生成されたプロジェクトはデプロイの指示がありますが、それを実行するとさらに2分かかります。