Déployer avec Docker

Dans cette section, vous verrez des instructions et des liens vers des guides pour savoir comment :

- Faire de votre application **FastAPI** une image/conteneur Docker avec une performance maximale. En environ **5 min**.
- (Optionnellement) comprendre ce que vous, en tant que développeur, devez savoir sur HTTPS.
- Configurer un cluster en mode Docker Swarm avec HTTPS automatique, même sur un simple serveur à 5 dollars US/mois. En environ **20 min**.
- Générer et déployer une application **FastAPI** complète, en utilisant votre cluster Docker Swarm, avec HTTPS, etc. En environ **10 min**.

Vous pouvez utiliser **Docker** pour le déploiement. Il présente plusieurs avantages comme la sécurité, la réplicabilité, la simplicité de développement, etc.

Si vous utilisez Docker, vous pouvez utiliser l'image Docker officielle :

tiangolo/uvicorn-gunicorn-fastapi

Cette image est dotée d'un mécanisme d'"auto-tuning", de sorte qu'il vous suffit d'ajouter votre code pour obtenir automatiquement des performances très élevées. Et sans faire de sacrifices.

Mais vous pouvez toujours changer et mettre à jour toutes les configurations avec des variables d'environnement ou des fichiers de configuration.

!!! tip "Astuce" Pour voir toutes les configurations et options, rendez-vous sur la page de l'image Docker : tiangolo/uvicorn-gunicorn-fastapi.

Créer un `Dockerfile`

- Allez dans le répertoire de votre projet.
- Créez un `Dockerfile` avec :

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

COPY ./app /app
```

Applications plus larges

Si vous avez suivi la section sur la création d' Applications avec plusieurs fichiers{.internal-link target=_blank}, votre `Dockerfile` pourrait ressembler à ceci :

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

COPY ./app /app/app
```

Raspberry Pi et autres architectures

Si vous utilisez Docker sur un Raspberry Pi (qui a un processeur ARM) ou toute autre architecture, vous pouvez créer un `Dockerfile` à partir de zéro, basé sur une image de base Python (qui est multi-architecture) et utiliser Uvicorn seul.

Dans ce cas, votre `Dockerfile` pourrait ressembler à ceci :

```
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"]
```

Créer le code FastAPI.

- Créer un répertoire `app` et y entrer.
- Créez un fichier `main.py` avec :

```
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}
```

- Vous devriez maintenant avoir une structure de répertoire telle que :

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

Construire l'image Docker

- Allez dans le répertoire du projet (dans lequel se trouve votre `Dockerfile`, contenant votre répertoire `app`).
- Construisez votre image FastAPI :

```
$ docker build -t myimage .

---> 100%
```

Démarrer le conteneur Docker

- Exécutez un conteneur basé sur votre image :

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

Vous disposez maintenant d'un serveur FastAPI optimisé dans un conteneur Docker. Configuré automatiquement pour votre serveur actuel (et le nombre de cœurs du CPU).

Vérifier

Vous devriez pouvoir accéder à votre application via l'URL de votre conteneur Docker, par exemple : http://192.168.99.100/items/5?q=somequery ou http://127.0.0.1/items/5?q=somequery (ou équivalent, en utilisant votre hôte Docker).

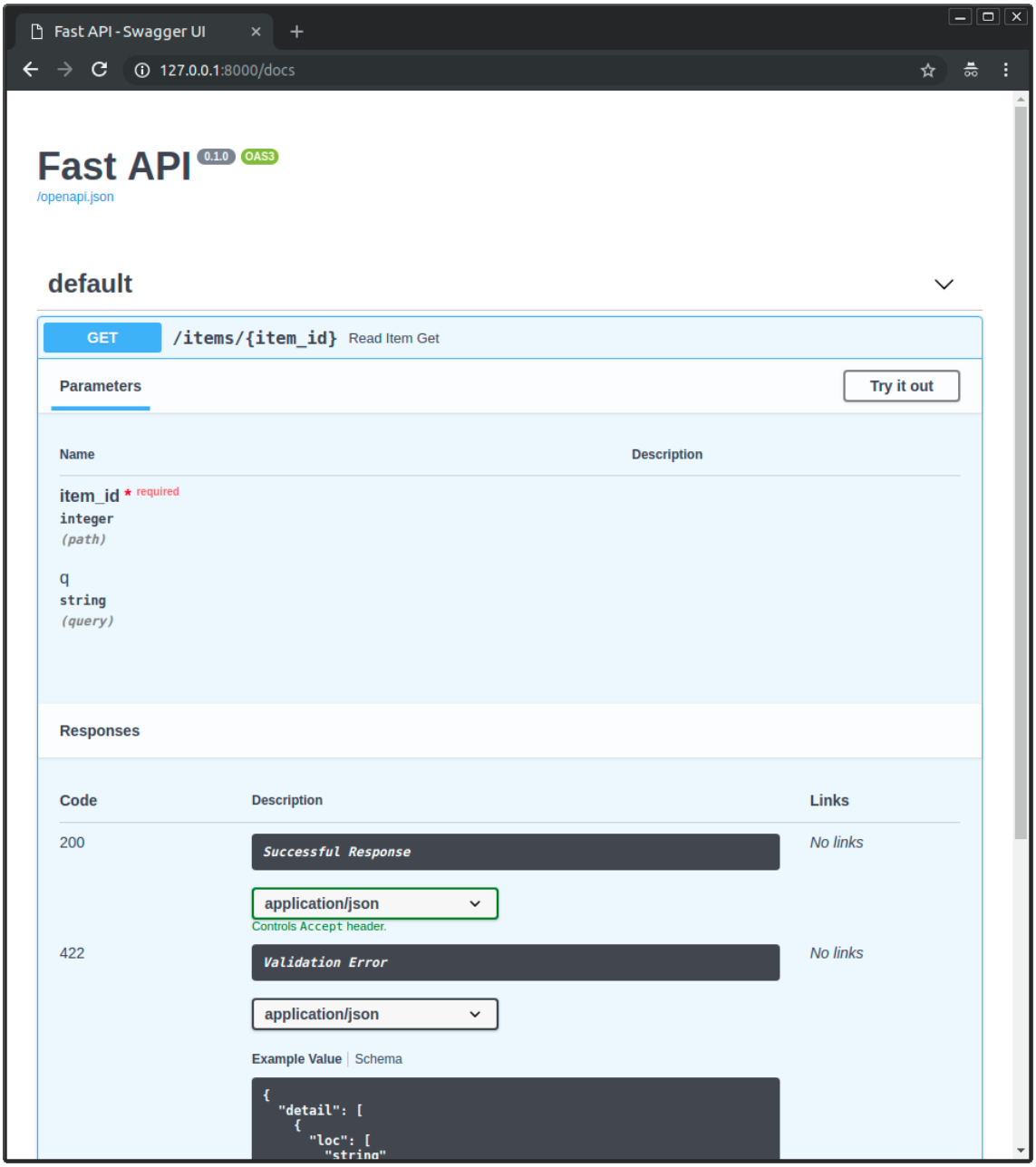
Vous verrez quelque chose comme :

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

Documentation interactive de l'API

Vous pouvez maintenant visiter http://192.168.99.100/docs ou http://127.0.0.1/docs (ou équivalent, en utilisant votre hôte Docker).

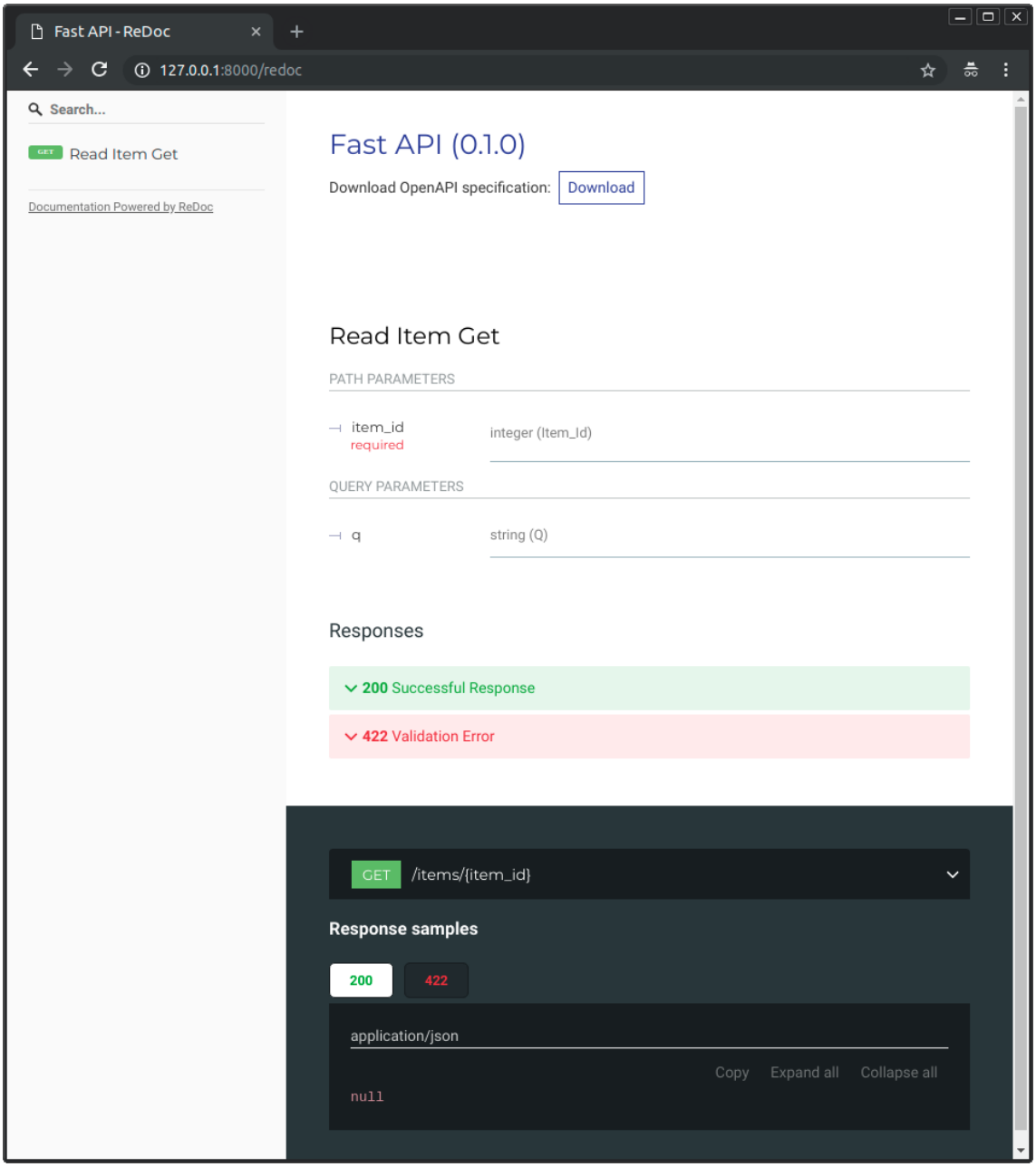
Vous verrez la documentation interactive automatique de l'API (fournie par Swagger UI) :



Documentation de l'API alternative

Et vous pouvez également aller sur http://192.168.99.100/redoc ou http://127.0.0.1/redoc (ou équivalent, en utilisant votre hôte Docker).

Vous verrez la documentation automatique alternative (fournie par ReDoc) :



Traefik

Traefik est un reverse proxy/load balancer haute performance. Il peut faire office de "Proxy de terminaison TLS" (entre autres fonctionnalités).

Il est intégré à Let's Encrypt. Ainsi, il peut gérer toutes les parties HTTPS, y compris l'acquisition et le renouvellement des certificats.

Il est également intégré à Docker. Ainsi, vous pouvez déclarer vos domaines dans les configurations de chaque application et faire en sorte qu'elles lisent ces configurations, génèrent les certificats HTTPS et servent via HTTPS à votre application automatiquement, sans nécessiter aucune modification de leurs configurations.

Avec ces informations et ces outils, passez à la section suivante pour tout combiner.

Cluster en mode Docker Swarm avec Traefik et HTTPS

Vous pouvez avoir un cluster en mode Docker Swarm configuré en quelques minutes (environ 20 min) avec un processus Traefik principal gérant HTTPS (y compris l'acquisition et le renouvellement des certificats).

En utilisant le mode Docker Swarm, vous pouvez commencer par un "cluster" d'une seule machine (il peut même s'agir d'un serveur à 5 USD/mois) et ensuite vous pouvez vous développer autant que vous le souhaitez en ajoutant d'autres serveurs.

Pour configurer un cluster en mode Docker Swarm avec Traefik et la gestion de HTTPS, suivez ce guide :

Docker Swarm Mode et Traefik pour un cluster HTTPS

Déployer une application FastAPI

La façon la plus simple de tout mettre en place, serait d'utiliser les **Générateurs de projet FastAPI**{.internal-link target=_blank}.

Le générateur de projet adéquat est conçu pour être intégré à ce cluster Docker Swarm avec Traefik et HTTPS décrit ci-dessus.

Vous pouvez générer un projet en 2 min environ.

Le projet généré a des instructions pour le déployer et le faire prend 2 min de plus.