Application OpenAI

Dans ce tutorial nous allons créer une application où l'utilisateur pose des questions à un personnage célèbre et ce personnage lui répond.

L'application repose sur l'envoi d'un prompt à l'API OpenAI.

D'un point de vue technique le stack est composé de

- python 3.12
- streamlit
- OpenAI

Ce tutorial nous permettra d'implémenter les points suivants

- gérer une clé API
- application python
- mounter le repertoire courant pour le code
- .dockerignore

Streamlit

Streamlit n'est pas directement lié à notre sujet de docker. Mais c'est un framework qui vaut la peine d'être connu.

Streamlit.io permet en quelques lignes de créer des applications et des dashboards tres efficaces, utiles, beau etc Streamlit est particulièrement adaptée aux projet de data science et aux dataframes pandas

Dans un terminal

```bash
pip install streamlit
streamlit hello
```

puis allez sur http://localhost:8501

Construire l'application

Tout d'abord la partie python

- `app.py` Le code de l'application tient dans un fichier app disponible sur https://github.com/SkatAI/ynov-docker/blob/master/apps/question-app/app.py
- `requirements.txt` liste les librairies requises par l'application. Le fichier contient 3 lignes : streamlit, openai et python-dotenv pour lire les variables d'environnement

```
streamlit
openai
python-dotenv
```

`python-dotenv` permet lire les variables d'environnement à partir d'un fichier .env où réside la clef API.

- `character_prompts.json` On a aussi besoin d'un fichier `json` pour la liste des personnages.

JavaScript

```
{
  "Louis de Funès": "Tu es Louis de Funès le comique français, répond e
  "Voltaire": "Tu es Voltaire le philosophe des lumières, répond à la q
  "Albert Einstein": "You are Albert Einstein, the famous physicist. An
  "Sherlock Holmes": "You are Sherlock Holmes, the fictional detective.
}
```

Changez les noms et instructions relatives aux personnages comme bon vous semble.

Les 3 fichiers sont sur https://github.com/SkatAI/ynov-docker/blob/master/apps/question-app/

Le Dockerfile

Le Dockerfile suit les instructions suivantes

- on part de l'image `python:3.12-slim`
- Définir le repertoire de travail `/app` dans le container (WORKDIR)

Installer les librairies python

- copier le fichier `requirements.txt` dans `/app`
- installer les librairies avec `pip install -r requirements.txt`

Le code

- copier les autres fichiers dans /app

Accessibilité

- exposer le port `8501` du container

Démarrer l'application

- avec `streamlit run app.py`

La solution se trouve sur le github mais il est plus intéressant pour vous d'essayer d'abord d'écrire le Dockerfile.

la clef API

La clef API ne doit pas être divulguée. Le service d'OpenAI étant payant, la facture finale pourrait être salée si la clef était rendu publique.

On va prendre l'approche du fichier .env pour la sécuriser. Et utiliser `.dockerignore` pour empêcher ce fichier d'arriver dans l'image.

- Dans le même repertoire créez un fichier `.env`
- récupérez la clef API dans le discord dans le channel #cle-API
- copiez la clef dans le fichier `.env`

Ensuite créez un fichier `.dockerignore` et ajoutez y les lignes suivantes

```
.env
dockerfile
```

Buildez l'image

Simplement en lui donnant le nom `questions` et le tag de version `01` : `questions:01`

```Bash
docker build -t questions:01 .
```

Notez que l'étape d'install des librairies de requirements prend le plus de temps.

Notez aussi que l'image commence a être de taille conséquente avec 533MB

```Bash
➜  question-app git:(master) ✗ docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
questions    01        fbad21d64f53   12 seconds ago   533MB
```

Exécutez le container

Quels sont les paramètres à passer au container ?

- le fichier .env avec le flag `-e`

et aussi

- le mapping des ports 8500:8500 avec `-p`
- le mode détaché avec `-d`
- le nom de l'image
- et le nom du container : `questions_app`

Ce qui donne

```Bash
docker run -d -p 8500:8500 -e ./.env questions:01 --name questions_app
```

Noms des images et containers

Le best practice est de ne pas donner le même nom au container et a son image. L'image est permanente tandis que le container est éphémère. son nom doit refléter le caractère opérationnel du container.

Toutefois il est utile que les 2 noms soient liées pour une meilleure organization. Le tout est donc de définir la bonne convention et de s'y tenir.

Ici ,l'image est le simple mot `questions` tandis que le container est nommé `questions_app` pour souligner qu'il s'agit là de l'application elle même.

Debugging et troubleshooting

Que vous voyez votre application sur `localhost:8501` ou non il est toujours intérèssant de surveiller ce qui se passe.

Voir les logs

Quand vous runnez Streamlit en local, les logs apparaissent dans le terminal.

Par l'intermédiaire d'un docker, les logs sont disponible via docker avec

```Bash
docker logs -n 20 -f 99ee30a14b5b
```

le `-f` permet de streamer les logs (equivalent de tail -f)

si tout va bien vous devriez voir

```Bash
2024-10-05 15:25:12.407 Starting server...
2024-10-05 15:25:12.407 Serving static content from /usr/local/lib/pyth
2024-10-05 15:25:12.411 Server started on port 8501
2024-10-05 15:25:12.411 Runtime state: RuntimeState.INITIAL -> RuntimeS

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://172.17.0.2:8501
  External URL: http://149.40.50.114:8501
```

accéder au container

L'autre moyen de debugguer et de voir que tout est en ordre est d'accéder au container avec

```Bash
docker exec -it 99ee30a14b5b /bin/sh
```

On utilise `/bin/sh` et non `/bin/bash` car le bash n'est pas installé sur les image slim.

Vous pouvez vérifier que le fichier `.env` n'est pas dans le container.

Modifier le code

On va maintenant modifier le code et voir la modification apparaître dans l'application

Pour cela il faut relancer le container en montant le repertoire courant au repertoire /app du container en ajoutant le paramètre suivant a commande de docker run.

```Bash
-v $(pwd):/app
```

Mais il faut d'abord stopper et supprimer le container

Puis relancer un nouveau container avec cette fois

```bash
docker run -d -p 8500:8500 \
-e ./.env \
-v $(pwd):/app \
questions:01 \
--name questions_app
```

Pour verifier que tout fonctionne, allez dans le fichier `app.py` et modifiez le code. Puis rechargez la page pour voir les modifications.

Vous pouvez aussi ajouter un personnage dans le fichier `character_prompts.json`. Il doit apparaitre dans le menu déroulant après avoir rechargé la page.

Rechargement automatique de la page

Streamlit dispose d'une fonction de reload automatique de la page. Pour l'activer il faut ajouter un fichier `config.toml` qui contient les lignes suivantes

Ajouter un fichier `config.toml` (optionnel) : Vous pouvez créer un fichier de configuration Streamlit dans le dossier de votre projet pour vous assurer que le rechargement automatique est activé.

Créez un fichier `.streamlit/config.toml` :

```
[server]
headless = true
reload = true
```

Cela garantira que Streamlit surveille les changements de fichiers et recharge automatiquement l'application.

Docker hub

Publiez votre image sur docker hub dans un nouveau repository