Docker - Conclusion et évaluation

120 minutes Normal

DataScientest • com

Docker

Conclusion

Docker et les containers en général sont une technologie très en vogue et très recherchée, que ce soit chez les acteurs de la Data ou les développeurs plus généralement. Elle permet de développer très facilement des applications, de les empaqueter et de les isoler du serveur de production. Dans un processus de passage du développement à la production, elle permet donc de simplifier et d'accélérer les phases de tests et de déploiement.

Avec une communauté très active et de très nombreuses images pré-existantes, officielles ou non, Docker constitue un atout majeur pour les Data Scientist, Machine Learning Engineers et Data Engineers.

Exercice

Pour valider ce module, il faut réaliser l'exercice suivant. Merci de bien lire attentivement les consignes.

1. Présentation

Pour la correction de cet exercice, nous allons essayer de créer un pipeline de CI/CD pour tester une API. Nous allons nous placer dans la peau d'une équipe censé créer une batterie de test à appliquer automatiquement avant déploiement.

Dans notre scénario, une équipe a créé une application qui permet d'utiliser un algorithme de sentiment analysis: il permet de prédire si une phrase (en anglais) a plutôt un caractère positif ou négatif. Cette API va être déployée dans un container dont l'image est pour l'instant `datascientest/fastapi:1.0.0`.

Regardons les points d'entrée de notre API:

- `/status` renvoie 1 si l'API fonctionne
- `/permissions` renvoie les permissions d'un utilisateur
- `/v1/sentiment` renvoie l'analyse de sentiment en utilisant un vieux modèle
- `/v2/sentiment` renvoie l'analyse de sentiment en utilisant un nouveau modèle

Le point d'entrée `/status` permet simplement de vérifier que l'API fonctionne bien. Le point d'entrée `/permissions` permet à quelqu'un, identifié par un `username` et un `password` de voir à quelle version du modèle il a accès. Enfin les deux derniers prennent une phrase en entrée, vérifie que l'utilisateur est bien identifiée, vérifie que l'utilisateur a bien le droit d'utiliser ce modèle et si c'est le cas, renvoie le score de sentiment: -1 est négatif; +1 est positif.

> Pour télécharger l'image, lancez la commande suivante

Machine status

Ubuntu Server 18.04 LTS SSD Volume Type 64-bit x86

Online

34.242.166.235

Connect

Reset Stop

Pour tester l'API manuellement, lancez la commande

```
docker container run -p 8000:8000 datascientest/fastapi:1.0.0
```

L'API est disponible sur le port `8000` de la machine hôte. Au point d'entrée `/docs`, vous pouvez trouvez une description détaillée des points d'entrée.

Nous allons définir certains scénarios de tests qui se feront via des containers distincts.

2. Tests

Authentication

Dans ce premier test, nous allons vérifier que la logique d'identification fonctionne bien. Pour cela, il va falloir effectuer requêtes de type `GET` sur le point d'entrée `/permissions`. Nous savons que deux utilisateurs existent `alice` et `bob` et leurs mots de passes sont `wonderland` et `builder`. Nous allons essayer un 3e test avec un mot de passe qui ne fonctionne pas: `clementine` et `mandarine`.

Les deux premières requêtes devraient renvoyer un code d'erreur 200 alors que la troisième devrait renvoyer un code d'erreur 403.

Authorization

Dans ce deuxième test, nous allons vérifier que la logique de gestion des droits de nos utilisateurs fonctionne correctement. Nous savons que `bob` a accès uniquement à la `v1` alors que `alice` a accès aux deux versions. Pour chacun des utilisateurs, nous allons faire une requête sur les points d'entrée `/v1/sentiment` et `/v2/sentiment`: on doit alors fournir les arguments `username`, `password` et `sentence` qui contient la phrase à analyser.

Content

Dans ce dernier test, nous vérifions que l'API fonctionne comme elle doit fonctionner. Nous allons tester les phrases suivantes avec le compte d'`alice`:

- `life is beautiful`
- `that sucks`

Pour chacune des versions du modèle, on devrait récupérer un score positif pour la première phrase et un score négatif pour la deuxième phrase. Le test consistera à vérifier la positivité ou négativité du score.

3. Construction des tests

Pour chacun des tests, nous voulons créer un container séparé qui effectuera ces tests. L'idée d'avoir un container par test permet de ne pas changer tout le pipeline de test si jamais une des composantes seulement a changé.

Lorsqu'un test est effectué, si une variable d'environnement `LOG` vaut 1, alors il faut imprimer une trace dans un fichier `api_test.log`.

Vous êtes libre de choisir la technologie utilisée: les librairies de Python `requests` et `os` semblent des options abordables. Le coeur de cet exercice n'étant pas tellement la programmation Python, nous vous proposons un exemple de code possible pour une partie de test:

```
3
4  # définition de l'adresse de l'API
5  api_address = ''
6  # port de l'API
7  api_port = 8000
8
9  # requête
10 r = requests.get(
11     url='http//{address}:
12 {port}/permissions'.format(address=api_address, port=api_port),
13     params= {
14         'username': 'alice',
15         'password': 'wonderland'
16     }
17 )
18
19 output = '''
20 ============================
21     Authentication test
22 ============================
23
24 request done at "/permissions"
25 | username="alice"
26 | password="wonderland"
27
28 expected result = 200
29 actual restult = {status_code}
30
31 ==>  {test_status}
32
33 '''
34
35
36 # statut de la requête
37 status_code = r.status_code
38
39 # affichage des résultats
40 if status_code == 200:
41     test_status = 'SUCCESS'
42 else:
43     test_status = 'FAILURE'
44 print(output.format(status_code=status_code,
45 test_status=test_status))
46
47 # impression dans un fichier
48 if os.environ.get('LOG') == 1:
       with open('api_test.log', 'a') as file:
           file.write(output)
```

Il faut donc réaliser le code pour chacun des tests (Authentication, Authorization et Content). Ensuite, on construira des images Docker via des DockerFile pour lancer ces tests. Réfléchissez bien à la façon aux arguments que vous souhaitez passer au container (commande à lancer, variables d'environnement, ...). N'hésitez pas à tester directement votre code directement sur le container depuis une console iPython ou même un notebook Jupyter.

4. Docker Compose

On l'a vu plus tôt mais Docker Compose est un outil très utilisé pour les pipelines de CI/CD. Il nous permet de lancer nos différents tests d'un coup tout en facilitant le partage de données entre les différents tests. On vous demande ici de créer un fichier `docker-compose.yml` qui organise ce pipeline. Pensez notamment aux utilisations des noms de containers, des variables d'environnement ainsi que des réseaux (networks).

5. Rendus

Les attendus de cet exercice sont:

- un fichier `docker-compose.yml` qui contient l'enchaînement des tests à effectuer
- les fichiers Python utilisés dans les images Docker
- les fichiers Dockerfile utilisés pour construire ces images
- un fichier de appelé `setup.sh` contenant les commandes utilisées pour construire les images et lancer le docker-compose
- le résultat des logs dans un fichier `log.txt`
- éventuellement un fichier de remarques ou de justification des choix effectués

Validated