Orsys - docker

Nicolas Rousset

Monday, 5th of december

1 TP - docker, CI et deploiement

1.1 Introduction

L'objectif de ce tp est d'exécuter le cycle complet de déploiement d'une application web, en intégrant des tests automatiques (CI - continuous integration) sous github action.

Ce tp est très guidé, et ne nécessite pas de connaissance en développement web. Une connaissance de `git` est préférable, mais non obligatoire.

Les utilisateurs avancés qui termineraient le tp en avance peuvent tenter leur chance en déployant d'autres images docker comme : - wordpress:6 - TODO - TODO

1.1.1 Rappel de git et github

`git` est le principal outil de gestion de version utilisée par les développeurs. Il permet de sauvegarder les différentes versions du code, et est souvent utilisé comme base pour le déploiement - le transfert des fichiers vers le serveur web distant se fait via git, souvent une `branche` nommée main, master ou prod.

`github` est le principal site d'hébergement public de repository `git` sur internet. On peut y avoir un compte gratuitement avec quelques limitations, notamment sur le nombre de repository privé.

`git` s'utilise principalement en ligne de commande, les principales lignes de commande sont :

- git clone git@github.com:Aenori/stunning-octo-robot.git => faire une copie locale d'un repository distant
- git commit -am "Some commit message" => sauvegarder localement les modifications de code
- git push => transférer les modifications locales vers un serveur distant

1.2 Mise en place

Etape 1: Si vous n'en avez pas, créez un compte gratuit sur github : https://github.com

Etape 2: Allez sur le repository du formateur : https://github.com/Aenori/stunning-octo-robot

Et forkez le repository sur votre compte en cliquant sur l'option fork en haut à droite :

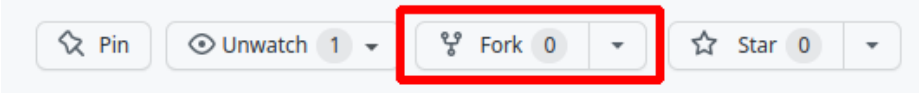

Figure 1: Github fork

Etpae 3: Allez sur votre compte et vérifiez que vous avez un repository qui s'appelle **stunning-octo-robot**

Etape 4: Clonez localement le repository localement :

```
git clone git@github.com:[NomDeVotreCompteGithub]/stunning-octo-robot.git
```

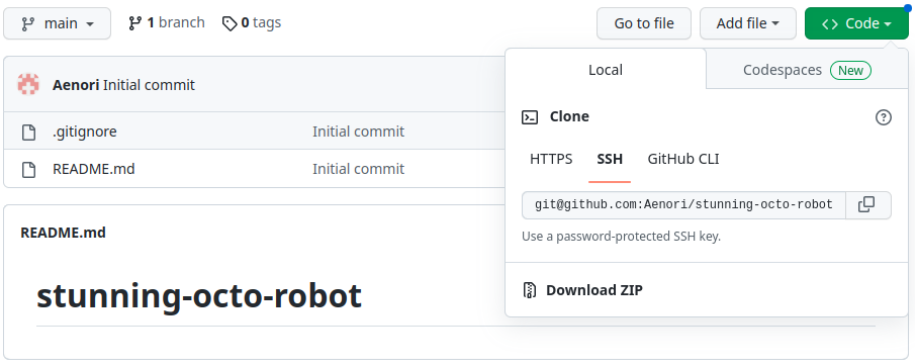

Figure 2: Github clone

La commande ci-dessus créera dans votre répertoire un sous répertoire `stunning-octo-robot` qui contiendra un fichier `application.py` ainsi que quelques sous-répertoires.

Parfait, vous êtes prêt à commencer le tp !

Remarque sur le site :

Il s'agit d'un petit projet utilisant python et flask. Cet environnement a été choisi car il nécessite peu de ligne de code et est simple à utiliser.

Le projet compte 4 routes (ie url) différentes, pour l'instant, seule la racine et hello fonctionneront, c'est normal.

1.3 Partie 1 : Ajout d'une image docker local

Etape 5: Ajout du Dockerfile

Placez vous à la racine de votre répertoire et créez un `Dockerfile` pour lancer le projet. Vous aurez besoin de 5 lignes :

- une image de base python 3.9
- la commande pour installer une dépendance python :

  RUN pip install flask
- la commande pour lancer l'application web python :

  CMD ["flask", "–app", "application.py", "run", "–host=0.0.0.0"]
- plus une ou deux lignes pour ajouter le code du répertoire à l'image

Etape 6: Test du dockerfile

Utiliser les deux commandes build et run pour lancer l'application dans un conteneur. Vous aurez besoin de lier le port 5000 du conteneur au port 5000 de la machine hôte

Maintenant connectez vous sur :

http://localhost:5000

Vous devriez voir indiquez Hello, world !. Si c'est le cas, félicitations !

1.4 Partie 2 : premier déploiement web

Etape 7: Ajout de la directive EXPOSE

Ajoutez dans votre Dockerfile une ligne avec indiqué

```
EXPOSE 5000
```

Vous souvenez-vous de ce que fait cette directive (et de ce qu'elle ne fait pas ?)

Etape 8: Déploiement du site web

Maintenant allez sur render (render.com) et suivez les étapes :

- Créez un compte gratuit
- Linkez votre compte github, en n'ajoutant que le repository cloné 'stunning-octo-robot'
- Sélectionnez un nouveau "web service", gratuit, avec l'option Docker
- Sélectionnez le repository 'stunning-octo-robot'

Maintenant vous devriez avoir accès à votre webservice sur le web. Félicitations ! Comme vous pouvez le constater, docker vous permet de déployer rapidement votre application sur le web, sans avoir besoin de paramêtrer le serveur distant, car le conteneur est identique localement et sur le serveur distant.

1.5 Partie 3 : integration continue

A noter que cette partie n'est pas indispensable en terme de maîtrise de Docker, il s'agit plus d'illustrer une application très intéressante de Docker.

Etape 9: Ajout de l'integration continue

Allez sur votre compte Github, sur votre repository `stunning-octo-robot` et cliquez sur `actions`

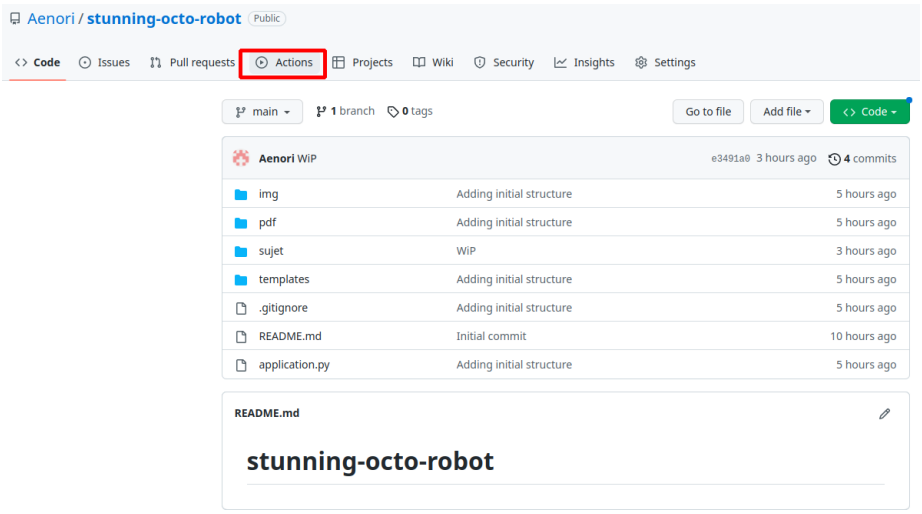

Figure 3: Github action

Ensuite sélectionnez l'option `python application` (à priori en bas à gauche)

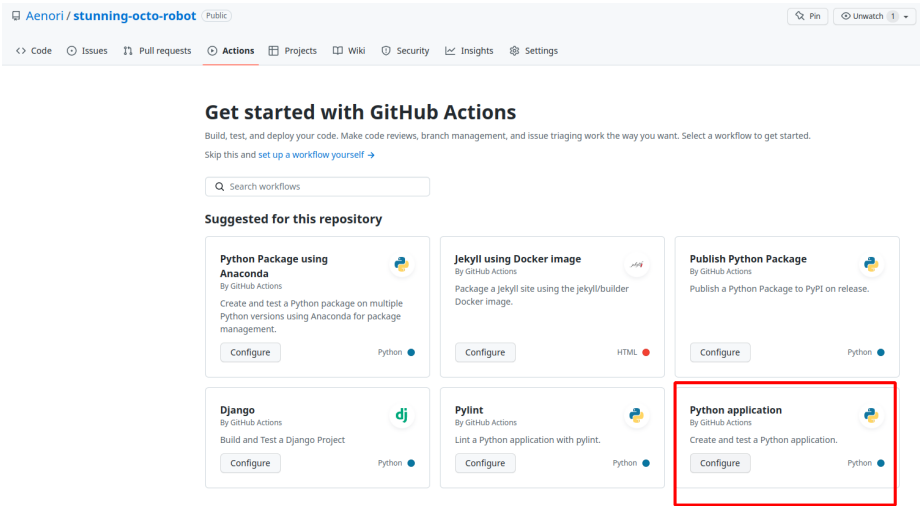

Figure 4: Python application

Et enfin valider le fichier de configuration tel quel en cliquant sur `start commit` (voir les 3 images suivantes)

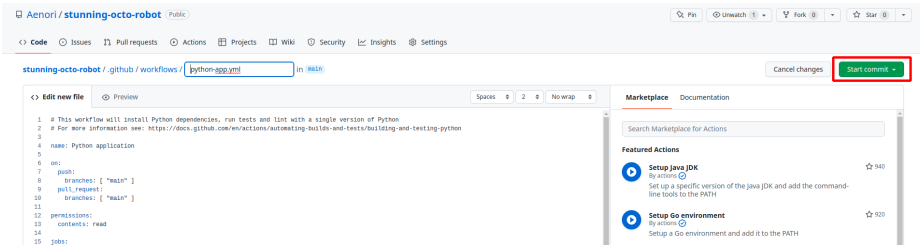

Figure 5: Validate

Félicitations, vous venez d'activer l'intégration continue ! A noter qu'il peut être nécessaire d'utiliser `git pull` pour récupérer le fichier créé par github dans le repository (sous le chemin .github/workflows/python-app.yml)

Normalement quand vous faites un push, vous devriez voir apparaître dans l'onglet "actions" du repository github un item de workflow.

Vous pouvez essayer par exemple de modifier le fichier templates/hello.html en remplaçant par exemple Hello, World! par juste Hello, et vous constaterez que github détecte automatiquement que quelque chose ne va pas.

Mais où intervient Docker ici ? En fait si vous ouvrez le fichier de configuration

`.github/workflows/python-app.yml`

Vous constaterez que Github fait tourner les tests unitaires sous une image docker ubuntu:latest et que le test de l'application passe par des steps identiques à ceux d'un Dockerfile.

Il est ainsi possible de customiser cette image, y compris en utilisant un Dockerfile customisé. Nous ne développerons pas cette possibilité, par manque de temps et car cela n'est pas forcément possible avec les options gratuites de github.

1.6 Partie 4 : ajout d'une dépendance système

Dans cette partie nous allons ajouter une dépendance système, ie une programme externe à python et qui n'est donc pas gérée par la gestion des dépendances de python. Cela permet de vraiment utiliser le potentiel de docker, puisque cette fonctionnalité ne sera simplement gérée par un hébergement python classique, ou au prix de beaucoup de configuration, spécifique à l'hébergement et coûteuses.

Pour cela nous allons utiliser le package ubuntu imagemagick, qui contient notamment des fonctionnalités de redimensionnement d'image.

Etape 10: Ajout d'imagemagick

Ajouter une ligne dans le docker file pour installer le package imagemagick

```
apt update && apt install -y imagemagick
```

A noter que l'on utilisera pas ce type de ligne dans un cas réel sans ajouter une partie pour purger les fichiers téléchargés par `apt update`

**Etape 11: Test de l'ajout d'imagemagick

Lancez l'application localement et vérifiez que la route /image fonctionne (elle appelle la librairie imagemagick pour redimensionner aléatoirement une image)

A chaque fois que vous rechargez la page, la taille de l'image doit changer.

Etape 12: Vérifiez que cela fonctionne sur le web

Déployez votre application sur le web. Comme votre application est lié à votre compte github, il suffit de faire un push sur le repository pour que l'application soit déployé sur render.

1.7 Partie 5 : ajout d'une base de donnée

Dans cette partie nous allons ajouter une base de donnée. Cela se fera localement en ajoutant un fichier `docker-compose.yml` à la racine du repository. Pour information le fichier docker-compose.yml de la correction fait 27 lignes.

Pour tester la base de donnée, il faudra utiliser les routes :

```
localhost:5000/db
localhost:5000/init_db
```

Pour cela, nous aurons de besoin de : - un fichier docker-compose - deux services, le service flask et une base de donnée - pour le service base de donnée, vous pouvez utiliser l'image postgres:15-alpine - rediriger le port du service flask (vous pouvez aussi rediriger le port 5432 de la base de donnée, mais cela n'est pas obligatoire) - un volume pour la base de donnée pour l'emplacement disque /var/lib/postgresql/data - un volume pour le service flask et le répertoire courant peut être utile, mais pas obligatoire - des variables d'environnement : - pour la db : POSTGRES_PASSWORD: postgres - pour flask : DB_CONNECTION: postgresql://postgres:postgres@postgres:5432/postgres (a noter que le postgres de postgres:5432 correspond à l'hôte de la base de donnée, qui est également le nom du service dans docker-compose.yml)

Etape 13: docker-compose sans base de donnée

Commencez par configurer le fichier docker-compose.yml

1.8 Partie 6 : déploiement web complet

Maintenant pour que notre application complète tourne sur le web, il suffit de configurer render. Pour cela il faut : - ajouter une base de donnée - ajouter dans l'application web la valeur de la variable d'environnement pour la connection à la base de donnée