CONSTRUIRE SES PROPRES IMAGES

Exercice 18 – Nouveau script Python

Je vous donne un nouveau script Python « passwordGenerator.py ». Celui-ci permet de générer des mots de passe aléatoires de la taille que vous voulez (par défaut 16 caractères), et de l'afficher.

Tentez de le lancer à partir d'une image « pypy:slim-bullseye » comme nous l'avons fait dans l'exercice 13. Que remarquez-vous ?

Exercice 18 – Nouveau script Python

Si ça ne fonctionne pas, c'est complétement logique. Vous faites appel à l'image de quelqu'un d'autre, qui n'a pas installé la dépendance requise « pyperclip » pour lire le script. Il va être temps de créer votre première image !

1. Créez un répertoire dans lequel vous allez placer le fichier « passwordGenerator.py »
2. Ajoutez dans ce répertoire un fichier appelé « Dockerfile » (sans extension)
3. Modifiez ce fichier dans votre IDE préféré (Visual Studio Code par exemple)

Créer un DOCKERFILE

Un Dockerfile suit généralement une structure simple, avec une série d'instructions qui décrivent comment construire une image Docker :

- Déclaration de l'image de base avec FROM : Vous devez spécifier l'image de base à partir de laquelle vous allez construire votre image Docker. C'est la première instruction !
- RUN : Exécute une commande dans le conteneur lors de la construction de l'image.
- COPY : Copie des fichiers depuis le système hôte vers l'image Docker.
- WORKDIR : Définit le répertoire de travail par défaut pour les commandes suivantes.

Exercice 18 – Créer un DOCKERFILE

4. Modifiez le Dockerfile dans votre IDE préféré (Visual Studio Code par exemple) pour ajouter les lignes suivantes :

```
FROM pypy:slim-bullseye (utilise l'image de base)
RUN apt-get update && apt-get install -y \
    python3-pip (met à jour les libs du conteneur et installe pip)
COPY passwordGenerator.py /app/passwordGenerator.py (copie le fichier)
WORKDIR /app (définie le repertoire de travail)
RUN pip install pyperclip (installe l'extension pyperclip)
```

Exercice 18 – Construire son image

Pour créer une image Docker à partir de ce Dockerfile, exécutez la commande suivante (depuis le dossier) :

```
docker build -t generate-password-app .
```

Cela créera une image Docker nommée generate-password-app. Vous pouvez vérifier si elle existe bien en listant les images présentes sur votre poste de travail !

Exercice 18 – Créer le conteneur

Maintenant que votre image sur mesure existe, vous pouvez avancer :

- Créez un conteneur basé sur votre nouvelle image avec une consigne infinie
- Allez dans votre conteneur au moyen de la commande « docker exec » comme avant
- Lancez le programme pour générer des mots de passe aléatoires avec la commande :

```
python passwordGenerator.py --length 48
```

Exercice 19 – Un Paris / New York en PHP

Vous allez maintenant utiliser un script PHP « haversine.php », qui prend en paramètre 4 nombres :

- Paramètre 1 : latitude d’un point A
- Paramètre 2 : longitude d’un point A
- Paramètre 3 : latitude d’un point B
- Paramètre 4 : longitude d’un point B

Ce script vous renvoie la distance en km entre ces points, d’abord calculée par la formule mathématique de Haversine, puis par une librairie opensource GeoLocation

Exercice 19 – Un Paris / New York en PHP

Pour réussir à bien cette mission, vous aurez besoin des infos suivantes pour construire votre propre image via un Dockerfile :

- Basez-vous sur l'image **composer:latest**
- Utilisez **composer** (gestionnaire de dépendances PHP) pour installer la dépendance grâce à une instruction RUN :

```
composer require anthonymartin/geo-location
```

- Construisez votre image, puis lancer le conteneur basé dessus, et une fois dedans faites vos tests entre plusieurs villes

Exercice 19 – Un Paris / New York en PHP

Grandes villes du monde :

- Paris, France (Latitude : 48.856613, Longitude : 2.352222)
- New York, États-Unis (Latitude : 40.7128, Longitude : -74.0060)
- Tokyo, Japon (Latitude : 35.682839, Longitude : 139.759455)
- Pékin, Chine (Latitude : 39.904211, Longitude : 116.407395)
- Le Caire, Égypte (Latitude : 30.044420, Longitude : 31.235712)
- Berlin, Allemagne (Latitude : 52.520008, Longitude : 13.404954)
- Londres, Royaume-Uni (Latitude : 51.507351, Longitude : 0.127758)

Exercice 20 – Dockerfile multistage

Un Dockerfile multistage est un moyen efficace de créer des images Docker plus légères en utilisant plusieurs étapes de construction. Il permet de diviser le processus de construction en plusieurs phases distinctes, en éliminant les dépendances inutiles et en optimisant la taille de l'image finale.

Il est également utile de faire appel à une logique multistage si vous utilisez un **langage de programmation compilé**. Vous pourrez gérer la compilation, les tests dans 2 phases distinctes, puis une 3ème phase pour l’image finale !

Exercice 20 – Dockerfile multistage

Je vous transmets un programme écrit en Rust, qui est langage qui nécessite d’abord d’être compilé, puis c’est un petit fichier binaire qui est lancé. C’est parfait, nous allons rédiger un Dockerfile multistage pour gérer d’abord la phase de compilation, puis le runtime sur un conteneur plus petit (sans besoin de dépendance).

1. Téléchargez et décompresser le dossier zippé « rust-example-helloWorld »
2. Ajouter votre Dockerfile à la raçine du projet

Exercice 20 – Le Dockerfile multistage

```
FROM rust:slim-buster AS build # Étape 1 : Stage de build
WORKDIR /app # Créez un répertoire de travail
COPY Cargo.toml ./ # Copiez les fichiers Cargo.toml et Cargo.lock pour gérer les dépendances
COPY src/ ./src/ # Copiez tous les fichiers sources
RUN rustup target add x86_64-unknown-linux-musl
RUN cargo build --release --target x86_64-unknown-linux-musl # Compilation de l'application

FROM scratch # Étape 2 : Stage final
# Copiez l'exécutable binaire depuis le stage de build
COPY --from=build /app/target/x86_64-unknown-linux-musl/release/rust-example /rust-example
```

Exercice 20 – Dockerfile multistage

Si vous analyser le Dockerfile, plusieurs choses doivent vous sauter aux yeux :

- Pour générer plusieurs étapes, il suffit de déclarer les étapes intermédiaires comme **From <nom de l'image> AS <nom de l'étape>**
- Ici, nous avons utilisé une image **rust** pour la compilation, ajouté des dépendances et une cible de compilation spécifique
- L'étape finale utilise une image **scratch**, la plus légère qu'on peut trouver. On copiera simplement le binaire depuis l'étape **build**

Exercice 20 – Dockerfile multistage

Faites marcher cette image en construisant d’abord l’image, puis en lançant un conteneur en direct avec la commande :

docker run -it <nom de l’image> ./rust-example

Vous devriez voir dans votre console le message Hello World! Notez comme nous n’avons pas eu besoin d’aller dans le conteneur, on peut également directement lancer un programme dans un conteneur grâce à la commande « docker run » !

Exercice 20 – Dockerfile multistage

Au passage, notez également le poids final de l’image que vous avez monté… et comparez-le avec les autres images en PHP, JS !

Cela doit vous faire comprendre que le choix dans le montage des images, surtout avec la stratégie multistage et un langage compilé, peut faire gagner beaucoup en termes de taille finale. Et quand l’image est plus petite, cela veut dire aussi une réduction des coûts d’hébergement !

Exercice 21 – Multistage pour la météo

C’est à vous de jouer avec un autre cas d’usage en Rust, une application de météo en direct ! J’ai construit un programme qui prends 3 paramètres :

- Un nom de ville (pour l’affichage)
- Sa latitude
- Sa longitude

Le programme vous renverra la température et le taux d’humidité qu’il y fait en temps réel (et pour de vrai !)

Exercice 21 – Multistage pour la météo

Comme pour l’exercice 20, vous allez construire un Dockerfile multistage, avec une étape de build et une étape finale sur une image scratch.

Attention, ajoutez plusieurs dépendances avant de lancer le build :

```
RUN apt-get update && apt-get install -y \
    libssl-dev \
    pkg-config \
    musl-tools
ENV OPENSSL_DIR=/usr
```

Exercice 21 – Multistage pour la météo

Comme pour l’exercice 20, vous n’avez pas besoin de rentrer dans le conteneur, vous pouvez le lançer directement grâce à la commande suivante (qui donnera la météo de Paris) :

docker run -it <nom de l’image> ./rust-weather Paris 48.856613 2.352222

Exercice 21 – Multistage pour la météo

Faites marcher cette ligne de commande, puis tester d’autres villes. Le nom est juste pour l’affichage, c’est vraiment la latitude et la longitude qui comptent.

Encore une fois, notez la taille finale de cette image, qui est ridiculement petite compte tenu du fait que vous avez des librairies de requêtages HTTP avec des appels à une API publique. Imbattable !

Récapitulatif

Prenez maintenant le temps de noter toutes les commandes nouvellement apprises, ce qu'elles font voire les options que vous pouvez passer en paramètre. Cette prise de notes vous sera utile !