Image Docker

Qu'est-ce qu'une Image Docker ?

Les images Docker sont des modèles immuables utilisés pour créer des conteneurs Docker. Elles constituent la base de la conteneurisation avec Docker, permettant de paquetiser le code, les outils, les bibliothèques, les dépendances, et tous les fichiers nécessaires à l'exécution d'une application dans un environnement isolé.

Les images Docker sont essentielles pour assurer la portabilité, la reproductibilité et la scalabilité des applications dans divers environnements de déploiement.

Caractéristiques des Images Docker

- **Immutabilité** : Une fois une image créée, elle ne change pas. Les modifications sont appliquées en créant une nouvelle image, ce qui favorise la consistance et la reproductibilité des environnements d'exécution.
- **Architecture en couches** : Les images Docker sont construites en utilisant un système de fichiers en couches. Chaque instruction dans un Dockerfile crée une nouvelle couche dans l'image. Les couches sont réutilisées entre les images, ce qui optimise l'utilisation de l'espace disque et le temps de téléchargement des images.
- **Stockage et partage** : Les images Docker peuvent être stockées et partagées à travers des registres d'images publics ou privés, comme Docker Hub, facilitant le partage et la distribution d'applications et de services conteneurisés.

Création et Utilisation des Images Docker

Création d'une Image

La création d'une image Docker commence généralement par l'écriture d'un Dockerfile, qui définit les étapes nécessaires pour assembler l'image, incluant la base à utiliser, les fichiers à copier, les commandes à exécuter, etc. Une fois le Dockerfile prêt, une image peut être créée en exécutant la commande :

```
docker build -t monimage:tag .
```

Cette commande construit une nouvelle image Docker à partir du Dockerfile dans le répertoire courant, en lui attribuant un nom (`monimage`) et un tag (`tag`).

Utilisation d'une Image

Pour exécuter un conteneur basé sur une image Docker, on utilise la commande :

```
docker run -d --name monconteneur monimage:tag
```

Cela crée et démarre un nouveau conteneur nommé `monconteneur` à partir de l'image `monimage:tag` . Le conteneur s'exécute en arrière-plan (`-d` pour "detached").

Gestion des Images Docker

Les images Docker peuvent être listées, modifiées, supprimées, et partagées à travers des commandes Docker. Par exemple :

- **Lister les images** : `docker images` ou `docker image ls`
- **Supprimer une image** : `docker rmi monimage:tag`
- **Télécharger une image** : `docker pull monimage:tag`
- **Envoyer une image** dans un registre : `docker push monimage:tag`

Exemple création d'une image docker :

Dockerfile

```
# Définir l'image de base. Ici, on utilise l'image officielle Python 3.8 en version "slim" pour une image finale plus légère.
FROM python:3.8-slim

# Définir le répertoire de travail dans le conteneur. Toutes les commandes qui suivent seront exécutées dans ce répertoire.
WORKDIR /app

# Copier le fichier requirements.txt dans le répertoire de travail (/app) du conteneur. Ce fichier liste les dépendances de l'application.
COPY requirements.txt .

# Installer les dépendances Python listées dans requirements.txt. L'option --no-cache-dir est utilisée pour minimiser la taille de l'image.
RUN pip install --no-cache-dir -r requirements.txt

# Copier les autres fichiers de l'application du répertoire courant sur l'hôte dans le répertoire de travail du conteneur.
COPY . .

# Exposer le port sur lequel l'application va s'exécuter. Cela ne publie pas le port, mais sert de documentation
```

```
entre celui qui déploie l'image et l'image elle-même.
EXPOSE 5000

# Définir la commande par défaut pour exécuter l'application. Ici, on lance l'application Flask en utilisant le serveur de développement de Flask.
CMD ["flask", "run", "--host=0.0.0.0"]
```

Pour cet exemple, supposons que vous avez une application Flask simple. Le fichier `requirements.txt` pourrait ressembler à ceci :

```
Flask==1.1.2
```

Et une application Flask de base, `app.py`, pourrait ressembler à :

Après avoir créé votre `Dockerfile` et placé `requirements.txt` et `app.py` dans le même dossier, vous pouvez construire l'image Docker avec la commande suivante (n'oubliez pas de remplacer `<tag>` par le nom et le tag de votre choix) :

```
docker build -t monapplicationflask:<tag> .
```

Cette commande construira une image Docker basée sur le `Dockerfile` du répertoire courant, en téléchargeant l'image Python, en installant les dépendances, et en copiant les fichiers de l'application dans l'image.

Pour exécuter un conteneur basé sur votre image nouvellement créée, utilisez :

```
docker run -p 5000:5000 monapplicationflask:<tag>
```

Cela démarrera un conteneur de votre image `monapplicationflask`, mappant le port 5000 du conteneur au port 5000 de votre hôte, permettant ainsi d'accéder à votre application Flask en naviguant vers `http://localhost:5000` dans votre navigateur.

Conclusion

Les images Docker sont au cœur de la plateforme Docker, permettant de créer des conteneurs qui exécutent des applications de manière isolée et reproductible.

Grâce à leur architecture en couches, leur immutabilité, et la facilité avec laquelle elles peuvent être partagées et stockées, les images Docker simplifient le déploiement et la gestion des applications dans des environnements de développement, de test, et de production.

Revision #1
Created 11 March 2024 11:07:47 by MASSON Romain
Updated 11 March 2024 12:31:02 by MASSON Romain