UNIVERSITÉ DE ROUEN	ANNÉE 2024-2025
FACULTÉ DES SCIENCES	ADMINISTRATION RÉSEAUX
MASTER 2 SSI	FICHE DE TD N° 2

Autour de Docker

Exercice 1 : Docker Volumes, Bind Mounts et Réseaux

Objectif

Comprendre l'utilisation des volumes, des bind mounts, et des réseaux dans Docker.

Instructions

1. Créez un répertoire local nommé `docker-volumes-test`.
— Dans ce répertoire, créez un fichier nommé `index.html` contenant le texte `"Bonjour de Docker Volume!"`.
2. **Bind Mount** :
— Démarrez un conteneur `Nginx` en utilisant le bind mount pour monter le répertoire `docker-volumes-test` dans le répertoire `/usr/share/nginx/html` du conteneur.
— Démarrer un conteneur de nom `monnginx` en mode détaché avec 1) un bind mount en lecture seule, du répertoire `docker-volumes-test` de la machine hôte sur le répertoire `/usr/share/nginx/htm` du conteneur 2) mapper le port 8080 de la machine hôte vers le port 80 du conteneur.
— Vérifiez le contenu du fichier en accédant à 'http ://localhost :8080' dans votre navigateur.
— Arrêtez puis supprimez le conteneur `monnginx`.
3. **Volume Docker** :
— Créez un volume Docker nommé `monvolume`.
— Démarrez un conteneur `Nginx` en utilisant ce volume pour stocker des fichiers dans le répertoire `/usr/share/nginx/html`.
— Copiez le fichier `index.html` dans le volume en démarrant un conteneur `busybox`.
— Montez le volume en lecture seule dans un nouveau conteneur Nginx de nom `monnginx` en mappant le port 8080 vers le port 80 du conteneur :
— Accédez à 'http ://localhost :8080' et vérifiez le contenu.
4. **Réseau Docker** :
— Créez un réseau Docker nommé `monrezo`.
— Démarrez deux conteneurs :
  — Un conteneur `Nginx`.
  — Un conteneur `BusyBox`.
— Connectez les deux conteneurs au réseau `monrezo` puis tester la connectivité entre eux.

Résultat attendu

Les conteneurs doivent être capables de partager des fichiers à l'aide de volumes et de bind mounts, et de communiquer entre eux sur le réseau Docker.

Exercice 2 : Dockerfile avec Multi-Stage Build

Objectif

Apprendre à utiliser les multi-stage builds pour optimiser la taille d'une image Docker.

Instructions

1. Créez un simple programme C `hello.c` qui affiche "Bonjour de Docker Multi-Stage Build !".
2. Créez un `Dockerfile` pour compiler ce programme avec une multi-stage build.
— Étape 1 : Utilisez l'image `gcc` pour compiler `hello.c`.
— Étape 2 : Utilisez l'image `alpine` pour créer une image finale minimale contenant uniquement l'exécutable.
3. Construisez l'image Docker avec le nom `hello-ssi`.
4. Exécutez un conteneur basé sur cette image et vérifiez la sortie.

Résultat attendu

Le conteneur doit afficher "Bonjour de Docker Multi-Stage Build !" et l'image finale doit être minimale.

Exercice 3 : Docker Compose

Objectif

Comprendre l'utilisation de Docker Compose pour orchestrer plusieurs conteneurs.

Instructions

1. Créez un répertoire nommé `docker-compose-test` puis créez dedans les fichiers suivants :
— `index.html` : Contient "Bonjour de Docker Compose !".
— `docker-compose.yml` : Définir les services `Nginx` et un serveur web simple (Apache).
2. Donnez le contenu du fichier `docker-compose.yml` permettant de lancer deux deux services `Nginx` et `Apache` qui doivent servir le contenu du fichier `index.html` sur différents ports.
3. Exécutez votre fichier `docker-compose.yml` en mode détaché.
4. Vérifiez les résultats :
— Accédez au site servi par le service `Nginx`.
— Accédez au site servi par le service `Httpd`.

Résultat attendu

Les deux services (`Nginx` et `Apache`) doivent servir le fichier `index.html` sur différents ports.

Objectif

Créer une image Docker minimale pour l'application `CMatrix` en utilisant un Dockerfile. Les sources du projet `CMatrix` sont disponibles sur le dépôt GitHub : https://github.com/abishekvashok/cmatrix. Cet exercice vous permettra de pratiquer les concepts de construction d'une image Docker optimisée et légère à partir des sources d'un projet open-source.

Instructions

1. **Cloner le Dépôt CMatrix**
   — Commencez par cloner le dépôt GitHub du projet `CMatrix` sur votre machine locale :
   — Familiarisez-vous avec le contenu du dépôt, en particulier le fichier `README.md` et le processus de compilation.
2. **Créer un Dockerfile Minimal**
   — Votre objectif est de créer une image Docker aussi petite que possible tout en incluant tout ce qui est nécessaire pour compiler et exécuter `CMatrix`.
   — Utilisez une **multi-stage build** pour séparer l'étape de compilation de l'étape d'exécution afin de réduire la taille de l'image finale.
   — Dans la première étape du Dockerfile, utilisez une image de base légère qui contient un compilateur C, comme `alpine` ou `debian-slim`, pour compiler `CMatrix`.
   — Dans la deuxième étape, utilisez une image de base encore plus petite, telle que `alpine`, pour exécuter uniquement le binaire compilé.
3. **Construire l'Image Docker**
   — Construisez l'image Docker en utilisant la commande suivante :
   — Vérifiez que l'image a été construite et notez sa taille :
4. **Tester l'Image Docker**
   — Lancez un conteneur basé sur l'image construite pour tester `CMatrix`.
   — Vous devriez voir l'animation de la matrice s'afficher dans le terminal.

Critères de Réussite

— L'image Docker doit être aussi petite que possible.
— L'application `CMatrix` doit s'exécuter correctement à l'intérieur du conteneur.
— Utilisez une multi-stage build pour optimiser la taille de l'image.
— L'image finale doit contenir uniquement les éléments nécessaires pour exécuter le binaire (pas les sources ou les outils de compilation).

Conseils

— Utilisez les images de base minimales comme `alpine` pour réduire la taille de l'image.
— Supprimez les fichiers inutiles dans l'image finale.
— Utilisez des couches Docker de manière efficace pour minimiser la taille.
— Ajouter le paquet `ncurses-terminfo-base` à l'image finale.