TRAVAUX DIRIGÉS
RAPPELS SUR LES CONTENEURS ET SÉCURITÉ

I. introduction

le but de ce TD est de revoir les commandes de base des conteneurs et introduire la notion de sécurité dans la conception et l'usage des conteneurs

II. Environnement

Nous allons travailler sur le système Debian des salles du plateau technique. Vous sauvegardez donc tous vos fichiers sur un projet Github afin de pouvoir redéployer rapidement vos infrastructures si nous changeons de salles

Vous installerez Docker et les éléments qui lui sont nécessaires. Prévoyez un script d'installation rapide permettant de reproduire l'installation dans une autre salle ou sur une autre machine. En terme de sécurité, il est préférable d'utiliser un autre user que root mais de lui octroyer les droits sudo.

III. rappel des principales commandes docker

III.1. **management docker.**

- **docker login** : permet de se logguer sur un docker registry. Nécessaire pour faire un push d'une image docker sur le registry en question
- **docker inspect #élément** : permet de donner des informations sur l'élément renseigné (image, conteneur, réseau, stockage, ...)
- **docker system prune** : permet de supprimer tous les conteneurs, images et réseaux non utilisés

III.2. **gestion des images.**

- **docker pull #nom_image** : permet de charger une image d'un hub vers le repository local.
- **docker images** : liste des images présentes
- **docker commit @id_container** : crée une image à partir d'un container en fonctionnement
- **docker history @id_image** : décrit les différentes couches de construction de l'image
- **docker rmi @id_image** ou **docker image rm @id_image** : supprime une image (uniquement si aucun container n'est lancé à partir de cette image, actif ou non)

III.3. **gestion des conteneurs.**

- **docker ps (-a)** : liste les conteneurs actifs ou avec l'option **-a** tous les conteneurs
- **docker rm (-f) @container_id** : supprime un conteneur inactif. avec l'option **-f** on supprime le conteneur même s'il est actif.
- **docker top @container_id** : liste les processus lancé dans le conteneur
- **docker port @container_id** : donne le mapping des ports d'un conteneur
- **docker cp ficher_local CONTAINER :path**

III.4. **run des conteneurs.**

- **docker run @image_name (commandes à exécuter)** : permet de démarrer un conteneur à partir d'une image et d'y lancer une commande spécifique si la commande est précisée
- **docker stop @container_id** : stop le conteneur actif
- **docker start @container_id** : démarre un conteneur stoppé
-

IV. DockerFile

Un Dockerfile permet de construire une image personnalisée. Il comprend à minima les instructions suivantes :

- **FROM** : qui indique l'image utilisée comme base de construction. L'ensemble des images hébergées sur le hub docker sont fournie avec le DockerFile qui a permis de la concevoir

— **RUN :** permet d'exécuter une commande pour dans l'image. Chaque commande RUN ajoute une couche non modifiable dans la constitution de l'image. Les commandes utilisées peuvent être la mise à jour du système, l'installation d'un applicatif, la création de répertoire, le déplacement dans le système de fichiers, ...
— **CMD :** permet d'indiquer la commande qui sera exécuté au démarrage d'une instance de cette image comme conteneur. Il existe deux mode d'écriture de CMD, le mode terminal et le mode exécution. Seule la dernière commande CMD sera utilisée.
— **ENTRYPOINT :** de la même façon que CMD, ENTRYPOINT indique la commande à exécuter dans le conteneur au démarrage. Si une instruction ENTRYPOINT est définie, suivant le mode d'écriture, la commande CMD sera ignorée (en mode terminal) ou sera utilisée comme paramétres de la commande ENTRYPOINT en mode exécution
— **ADD / COPY :** permet de copier un fichier du système hôte vers le conteneur. La différence entre les deux instructions porte sur le fait que pour ADD, les fichiers, au format archive, sont décompressés automatiquement si le conteneur le permet (outil de décompression disponible dans le conteneur
— **VOLUME :** permet de monter un volume local dans le conteneur, mais limité car le volume est défini pour le conteneur et disparait à al destruction de celui ci. Non pertinent pour de la sauvegarde de données. il faut préférer volumes nommés ou l'option -v du *docker run*

les modes d'écriture de CMD/ENTRYPOINT sont

mode	format	exemple
mode terminal	CMD commande	CMD ping 192.168.1.1
mode exécution	CMD ['commande','argument',...]	CMD ["ping","192.168.1.1"]

V. Sécurité des conteneurs

Les conteneurs avec Docker sont lancé avec les privilèges root sur le système hôte, nécessité lié à l'usage d'un démon par docker pour exécuter les conteneur. Cela entraine une potentielle faille de sécurité si un utilisateur réussi à sortir du conteneur.

V.1. outils pour l'analyse des conteneurs.

(1) **docker history @image_name** : cela permet de voir les différentes couches qui sont créées, leur taille et les commandes qui les ont générées. Ce premier point d'entrée permet d'envisager l'optimisation du Dockerfile pour limiter la taille de l'image. C'est important surtout dans le cadre d'une mise en production avec les notions de scalabilté et de réplicats

(2) **Hadolint** : logiciel permettant de tester un Docker file et de faire des préconisations en termes de sécurité et de bonnes pratiques : la version en ligne est https ://hadolint.github.io/hadolint/ . Ces bonnes pratiques sont aussi décrites dans la docs de docker (https ://docs.docker.com/build/building/best-practices/)

(3) **trivy** : permet de remonter l'ensemble des vulnérabilités présentes dans l'image et dans toutes ses couches.

(4) **dive** : Cet outil permet de voir l'ensemble des fichiers installés par couche d'une image docker. il permet ainsi d'optimiser la taille de l'image en reconstruisant une image où certains fichiers sont modifiés ou supprimés.

V.2. **usage de Podman.** Podman est une alternative à Docker, proposée par RedHat. Elle permet d'exécuter des conteneurs sans avoir les droits root, car il n'y a pas besoin de daemon qui tourne pour intéragir. Pour utiliser podman, il suffit de remplacer dans les commandes docker, *docker* par *podman*. Vous pouvez installer podman en suivant la procédure décrite sur le lien suivant (https ://podman.io/docs/installation)

VI. exercice

Dans une VM debian, procéder à l'installation de docker, refaites rapidement les exercices du TP2 sur le Dockerfile, puis tester la sécurité de vos différents fichiers avec les outils. Tester ensuite la solution podman avec les mêmes images.