

DÉPLOIEMENT D'UN SERVICE AVEC DOCKER

ACTIVITÉ 4 – DÉPLOIEMENT DE LA DISTRIBUTION KALI-LINUX

Lien :

- Lien vers une formation sur Docker complète mais très simple : <https://blog.microlinux.fr/formation-docker/>

Les documents suivants viennent en complément :

- récapitulatif des commandes Docker ;
- les Instructions Dockerfile.

SOMMAIRE

A. Personnalisation de l'image.....	3
B. Création d'une première image et déploiement du conteneur.....	5
C. Optimisation du déploiement.....	7
1. Gestion des volumes.....	7
2. Gestion du réseau.....	8

La distribution Kali-Linux est utilisée tant par les administrateurs réseaux et système que les développeurs dans leurs tests de vulnérabilité sur les infrastructures et les applications. L'administrateur réseau vous demande de préparer des images de cette distribution.

L'objectif est de :

- créer des images personnalisées de la distribution Kali-Linux intégrant une interface graphique ;
- différencier les images (toutes basées sur l'image officielle de kali « [kalilinux/kali-rolling](https://kalilinux.com) ») en fonction des méta-paquets installés de manière à disposer de distributions plus ou moins « légères » ;
- publier ces images de Kali-Linux sur le registre officiel ;
- déployer les distributions en tant que de besoin.

Nous allons, pour cela, utiliser (et personnaliser) une solution Docker :

- <https://github.com/onemarcifty/kali-linux-docker>

La solution s'appuie elle-même sur l'image officielle de kali « [kalilinux/kali-rolling](https://kalilinux.com) ».

L'accès à la distribution sera réalisé via le protocole RDP et la distribution sera également accessible en SSH.

Cette activité sera aussi l'occasion de :

- de consolider les notions sur les volumes ;
- d'appréhender des notions sur le réseau avec Docker

Vous devez faire toutes les manipulations, notamment celles précédées par :



A. PERSONNALISATION DE L'IMAGE



Pour comprendre le fonctionnement de la solution, lire l'intégralité de la documentation et visualiser la vidéo. Vous pouvez également consulter les différents fichiers.

Nous allons utiliser l'outil « git » (à installer si cela n'est pas déjà fait) pour récupérer les fichiers nécessaires dans le dossier « contexte ».

```
user@servDockerAR:~$ cd contexte
user@servDockerAR:~/contexte$ git clone https://github.com/onemarcfifty/kali-linux-docker.git
user@servDockerAR:~/contexte$ cd kali-linux-docker
user@servDockerAR:~/contexte/kali-linux-docker$ cp env_template env
```

Il est possible ensuite de supprimer le dossier « .git » et le fichier « .gitignore ».

Le projet est constitué de trois fichiers principaux :

- **le fichier « env »** contenant les variables qui vont permettre de :
 - personnaliser l'image (comme l'environnement du bureau, les méta-paquets que l'on veut installer ou le logiciel d'accès à distance que l'on veut utiliser),
 - personnaliser la création du conteneur (comme les ports exposés ou le volume destiné aux données persistantes) ;
- **le dockerfile** à partir duquel l'image va être créée qui installe tous les éléments nécessaires en fonction notamment des choix précédemment effectués via les variables d'environnement ;
- **le script build** qui :
 - récapitule la configuration choisie via les variables d'environnement (si une variable n'est pas définie dans le fichier « env », le script demande toutes les options),
 - construit l'image Docker s'appuyant sur le fichier dockerfile qui prend en compte les variables d'environnement,
 - crée le conteneur à partir de l'image créée et de certains variables (comme les ports exposés),
 - démarre le nouveau conteneur.

Il est nécessaire de procéder à quelques modifications pour adapter ce projet à notre contexte.

Le fichier Dockerfile ne prévoit pas l'utilisation des locales françaises et le timezone Europe/Paris.

➤ Remplacer les deux lignes 83 et 84 par celles qui le permettent.

Certaines variables d'environnement doivent être personnalisées :

- L'accès à l'interface graphique ne sera réalisé que via le protocole RDP, tous les éléments relatifs à VNC peuvent être désactivés.
- Les paquets kali à installer : « core » (on réalise une image légère dans un premier temps, aucun paquet supplémentaire n'est installé – voir [ici](#)).
- Le nom de l'image docker : `votre_id_docker/kalilinux:core` (on aura l'occasion de réaliser d'autres images de kali, on les distinguera via le tag en fonction des méta-paquets installés).
- Le nom du conteneur généré : `kali-linux-core` (on pourra ainsi en utiliser plusieurs en parallèle avec des noms différents personnalisés en fonction des méta-paquets installés).
- Le répertoire hôte à monter en tant que volume : `votre_répertoire_personnel/kali-linux-core`.
- Le répertoire du conteneur associé (on persiste ici les données de l'utilisateur créé dans le conteneur) : `/home/userkali`.
- Le nom d'utilisateur : « userkali ».
- le mot de passe de l'utilisateur à renforcer.



Vous pouvez constater que le `dockerfile` modifie les ports d'écoute des services `ssh` et `xrdp` même à l'intérieur du container. Pour les utilisations courantes que nous avons l'habitude de faire (par exemple sur les laboratoires du bloc3), cela n'est pas utile :

- les services du conteneur écoute sur leurs ports habituels ;
- on y accède via un mappage de port.

Mais nous allons plus tard utiliser ce conteneur en l'intégrant complètement au réseau de l'hôte et il sera donc nécessaire à ce moment-là d'utiliser des ports (même les ports internes au conteneur) qui ne sont pas utilisés sur l'hôte : voir partie 2.3.

➤ Modifier, dans le fichier « `env` » les variables d'environnement correspondantes.

Vous ferez référence autant que possible à des noms de variable de manière à ce qu'il y ait le moins de chose à modifier lorsque l'on créera une nouvelle image basée par exemple sur le méta-paquet « `default` ».

➤ Supprimer l'exposition du port VNC dans le `Dockerfile`.

➤ Modifier la création du conteneur dans le script « `build` » pour supprimer le port d'écoute de VNC.

➤ Créer le `$HOSTDIR` (si vous ne le créez pas, la commande de création de conteneur va le faire mais attribuera ce dossier à l'utilisateur `root`).

B. CRÉATION D'UNE PREMIÈRE IMAGE ET DÉPLOIEMENT DU CONTENEUR

Il s'agit maintenant d'utiliser le script « build » pour créer l'image et déployer le conteneur.

- Se positionner dans le contexte : **cd ~/contexte/kali-linux.**
- Exécuter le script « build » (l'exécution prend un certain temps notamment au niveau de l'étape qui installe les paquets – Pour ne pas perdre de temps, répondre aux questions qui suivent et continuer la documentation) : **./build :**

Configuration:

Desktop environment: xfce
Remote Access: rdp
...

Hit enter to start building the container

Pour info, voici un exemple de récapitulatif quand la commande a été réalisée avec succès :

```
[+] Building 721.7s (26/26) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile                0.1s
=> => transferring dockerfile: 7.85kB                             0.0s
=> [internal] load .dockerignore                                  0.0s
=> => transferring context: 2B                                     0.0s
=> [internal] load metadata for docker.io/kalilinux/kali-rolling:latest 2.5s
=> [auth] kalilinux/kali-rolling:pull token for registry-1.docker.io 0.0s
=> CACHED [ 1/21] FROM docker.io/kalilinux/kali-rolling@sha256:e8dc477b5 0.0s
=> [ 2/21] RUN apt update -q --fix-missing                        12.0s
=> [ 3/21] RUN apt upgrade -y                                    27.3s
=> [ 4/21] RUN apt -y install --no-install-recommends sudo wget curl d 399.8s
=> [ 5/21] RUN apt -y install locales                             28.6s
=> [ 6/21] RUN sed -i s/^# en_US.UTF-8 UTF-8/en_US.UTF-8 UTF-8/ /etc/loc 0.5s
=> [ 7/21] RUN locale-gen                                         4.3s
=> [ 8/21] RUN echo "#!/bin/bash" > /startkali.sh               5.1s
=> [ 9/21] RUN echo "/etc/init.d/ssh start" >> /startkali.sh     5.9s
=> [10/21] RUN chmod 755 /startkali.sh                            6.0s
=> [11/21] RUN apt -y install --no-install-recommends kali-linux-core 38.4s
=> [12/21] RUN useradd -m -s /bin/bash -G sudo userkali          6.5s
=> [13/21] RUN echo "userkali:userKaliPass0" | chpasswd          1.1s
=> [14/21] RUN echo "Port 20022" >> /etc/ssh/sshd_config          6.0s
=> [15/21] RUN rm /etc/xdg/autostart/xfce4-power-manager.desktop >/dev/n 5.1s
=> [16/21] RUN if [ -e /etc/xdg/xfce4/panel/default.xml ] ;      5.4s
=> [17/21] RUN if [ "xx2go" = "xrdp" ] ; then apt -y instal      4.9s
=> [18/21] RUN if [ "xrdp" = "xrdp" ] ; then apt -y ins         65.7s
=> [19/21] RUN if [ "xvnc" = "xrdp" ] ; then apt -y install     5.1s
=> [20/21] RUN echo "/bin/bash" >> /startkali.sh                6.1s
=> [21/21] WORKDIR /root                                         5.2s
=> exporting to image                                           79.5s
=> => exporting layers                                           79.5s
=> => writing image sha256:86b31d65fb8775c7f14aaf23a0dd46dabf7ddd9d1c606 0.0s
=> => naming to docker.io/aporaf/kalilinux:core                 0.0s
14b9c305e50eb33b92fac6b6efe214d3db1ef4020cc25ff541a6e76be7961654
Image (aporaf/kalilinux:core) and container (kali-linux) build successful.
kali-linux will now start.
kali-linux
```

 Vous serez potentiellement amené à supprimer le conteneur créé et en créer un autre avec la même image. Il ne sera pas nécessaire de recommencer tout le processus.

Le script « build » utilise deux commandes pour lancer le conteneur en lieu et place de la commande « docker run » :

- docker create
- docker start

➤ Écrire la commande de création du conteneur en remplaçant chaque variable par sa valeur.

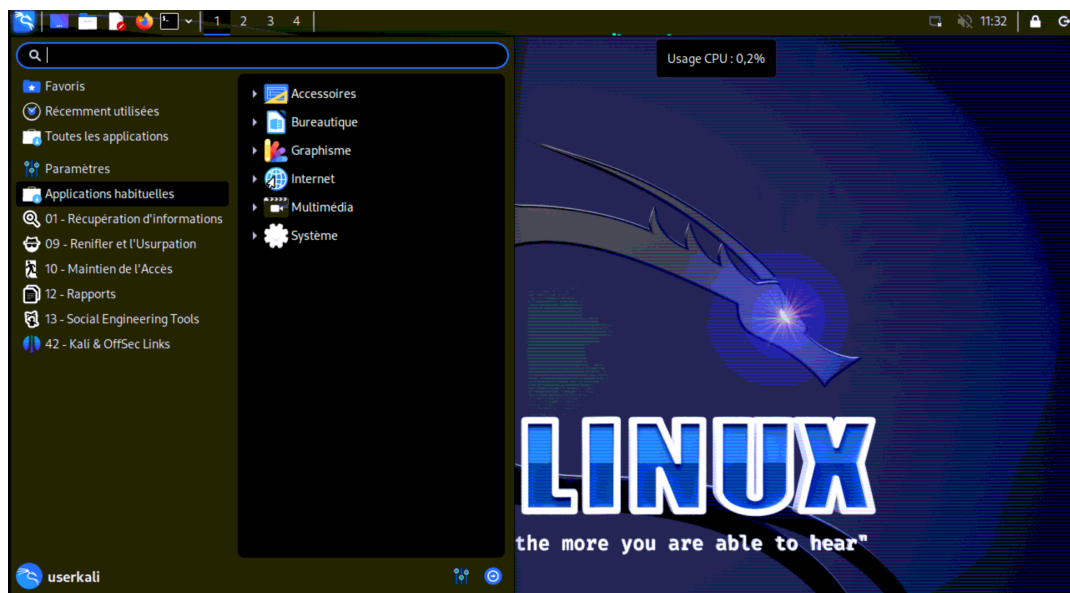
➤ Écrire la commande de démarrage du conteneur.

➤ Écrire la commande qui permettra d'accéder à la distribution en SSH.

Vous vérifierez cet accès une fois le conteneur créé.

➤ Configurer le client bureau à distance pour accéder à l'interface graphique de Kali.

i Pour les utilisateurs de Remmina, ouvrez les paramètres de connexion pour le profil de connexion qui rencontre des problèmes, accédez à Avancé et sélectionnez « Cache de glyphes » et « Détendez les vérifications de commande ».



➤ Publier l'image sur le registre officiel.

➤ Donner la taille de votre image.

➤ Vérifier que le conteneur se soit bien créé.

C. OPTIMISATION DU DÉPLOIEMENT

1. GESTION DES VOLUMES

Le volume nécessaire à la persistance des données a été créé manuellement, mais il est préférable de laisser Docker gérer complètement les volumes pour plusieurs raisons principales :

1. **Portabilité** : Docker garantit que les données associées à un conteneur sont stockées de manière indépendante de l'hôte. Cela signifie qu'il est facile de déplacer les conteneurs d'un hôte Docker à un autre sans avoir à se soucier du stockage sous-jacent. Les volumes Docker sont portables et transférables.
2. **Simplicité** : Docker offre des commandes simples et cohérentes pour créer, gérer et inspecter les volumes. Vous pouvez les créer avec « `docker volume create` », les attacher à des conteneurs avec « `docker run -v` », et les supprimer avec « `docker volume rm` ». Cela simplifie la gestion du stockage associé aux conteneurs.

3. **Sécurité** : Docker gère les permissions et les propriétaires des volumes, ce qui renforce la sécurité. Les conteneurs s'exécutent dans un environnement isolé, et Docker veille à ce que les données stockées dans les volumes ne soient pas facilement accessibles depuis l'extérieur. Par ailleurs, à tout moment, tous les volumes peuvent être répertoriés avec « `docker volume ls` » sans risque d'en oublier sur l'espace disque avec des données personnelles par exemple.
4. **Sauvegarde et restauration** : Les volumes Docker peuvent être sauvegardés et restaurés facilement, car toutes au même « endroit », ce qui facilite la gestion des données persistantes dans les conteneurs. Vous pouvez également automatiser ce processus à l'aide d'outils tiers ou de scripts.
5. **Évolutivité** : Docker prend en charge la gestion des volumes sur des clusters de conteneurs, ce qui vous permet de mettre à l'échelle vos applications et de gérer efficacement le stockage partagé entre plusieurs conteneurs sur différentes machines.

➤ Supprimer le conteneur créé à l'étape précédente.

➤ Supprimer l'espace disque actuellement utilisé (`/home/$USERNAME/kali-linux-core`)

➤ Modifier la création du conteneur dans le script « build » de manière à intégrer cette nouvelle gestion de volumes (le volume associé à un conteneur se nommera « `home_<nom du conteneur>` »).

Il est rappelé qu'il n'est pas nécessaire de créer le volume avant, s'il n'existe pas, Docker le créera automatiquement lors de l'exécution de la commande `docker create` ou `docker run`.

2. GESTION DU RÉSEAU

Pour approfondir les concepts de réseau sur Docker : <https://blog.microlinux.fr/formation-docker-11-reseaux/>.


Le driver « bridge » (ce que nous utilisons par défaut)

Lorsque que l'on installe Docker, il crée automatiquement un réseau de type bridge (pont) nommé `bridge` connecté à l'interface réseau `docker0` (consultable avec la commande `ip addr show docker0`) d'adresse IP `172.17.0.0/16` et de passerelle `172.17.0.1` (correspondant à l'interface `docker0`). Chaque nouveau conteneur Docker est automatiquement connecté à ce réseau, sauf si un réseau personnalisé est spécifié.

Tous les conteneurs connectés à ce réseau peuvent communiquer via leur adresse IP locale (distribuée automatiquement par Docker) et chaque conteneur peut communiquer en dehors du réseau via cette passerelle. Docker utilise le pare-feu `Netfilter` intégré au noyau Linux pour toutes les opérations de pare-feu et de routage.

Lorsque l'on active l'option -p (par exemple -p 8080:80), Docker ouvre le port 8080 sur le système hôte pour le faire pointer vers le port 80 du conteneur auquel il applique une adresse IP locale (dans le réseau spécifié, 172.17.0.0/16 par défaut donc).

Le routage interne de Docker fait transiter le trafic de manière transparente.


 Lorsque l'on déploie une application ou plusieurs conteneurs doivent être liés mais aussi isolés des autres conteneurs, il est d'usage de créer un réseau isolé de type bridge (voir activité suivante).

Le driver « none »

C'est le type de réseau à utiliser si on souhaite interdire toute communication interne et externe avec le conteneur, car ce dernier sera dépourvu de toute interface réseau (sauf l'interface loopback).

Le driver « host » (ce que nous allons mettre en œuvre)

Ce type de réseau permet aux conteneurs d'utiliser la même interface que l'hôte. Il supprime donc l'isolation réseau entre les conteneurs et seront par défaut accessibles de l'extérieur. De ce fait, le conteneur prendra la même IP que votre machine hôte et tous les ports exposés le seront également sur l'hôte.

 Cette manière de procéder est peu commune, étant donné qu'elle n'est pas du tout sécurisée mais c'est celle qui nous intéresse ici si l'on veut utiliser le conteneur Kali comme s'il faisait partie intégrante de notre réseau pour par exemple tester des attaques sur le réseau de production et/ou pour faire une capture wireshark sur ce même réseau,

Vous pouvez activer cette fonctionnalité en utilisant l'option --network host dans la commande docker run. Cette option permet au conteneur de partager le réseau de l'hôte. Le conteneur n'a pas d'adresse IP qui lui est propre, il utilise celle de l'hôte.

 Supprimer le conteneur

 Modifier la création du conteneur dans le script « build » de manière à intégrer cette nouvelle gestion du réseau



Pour déployer les autres images et conteneurs associés en tant que de besoin, il suffit de

modifier :

- la variable \$XKALI_PKG ;
- les variables relatives aux ports (uniquement si les conteneurs ont vocation à démarrer en même temps).