RECOMMANDATIONS DE SÉCURITÉ RELATIVES AUX DÉPLOIEMENTS DE CONTENEUR DOCKER

FICHE TECHNIQUE

ANSSI-FT-082
23/09/2020

PUBLIC VISÉ :

Développeur **Administrateur** RSSI **DSI** Utilisateur

Informations

Attention

Ce document rédigé par l'ANSSI présente les **« Recommandations de sécurité relatives aux déploiements de conteneur Docker »**. Il est téléchargeable sur le site `www.ssi.gouv.fr`.
Il constitue une production originale de l'ANSSI placée sous le régime de la « Licence ouverte v2.0 » publiée par la mission Etalab [3].
Conformément à la Licence Ouverte v2.0, le guide peut être réutilisé librement, sous réserve de mentionner sa paternité (source et date de la dernière mise à jour). La réutilisation s'entend du droit de communiquer, diffuser, redistribuer, publier, transmettre, reproduire, copier, adapter, modifier, extraire, transformer et exploiter, y compris à des fins commerciales.
Ce document est la synthèse d'une étude de cette technologie, réalisée en temps contraint, qui ne vise donc pas l'exhaustivité. Sauf disposition réglementaire contraire, ces recommandations n'ont pas de caractère normatif ; elles sont livrées en l'état et adaptées aux menaces au jour de leur publication. Au regard de la diversité des systèmes d'information, l'ANSSI ne peut garantir que ces informations puissent être reprises sans adaptation sur les systèmes d'information cibles. Dans tous les cas, la pertinence de l'implémentation des éléments proposés par l'ANSSI doit être soumise, au préalable, à la validation de l'administrateur du système et/ou des personnes en charge de la sécurité des systèmes d'information.

Évolutions du document :

VERSION	DATE	NATURE DES MODIFICATIONS
1.0	23/09/2020	Version initiale

Table des matières

1
Préambule

1.1 Qu'est ce que Docker ?

Docker est une plateforme ouverte pour le développement, le déploiement et l'exécution d'applications. Il permet d'embarquer et d'exécuter une application dans un environnement cloisonné appelé conteneur. Ce cloisonnement permet d'exécuter plusieurs conteneurs simultanément sur un hôte donné. À la différence de la virtualisation où la machine virtuelle contient un système d'exploitation complet, les outils associés et l'application hébergée, le conteneur ne contient que les bibliothèques et les outils nécessaires à l'exécution de l'application, et il fonctionne directement dans le noyau de la machine hôte.

Un conteneur est une unité standard de logiciel, qui embarque le code et toutes ses dépendances, afin que l'application « conteneurisée » fonctionne normalement et de façon fiable, quelle que soit la machine hôte. L'image d'un conteneur, présente sur le système, devient un conteneur au moment de l'exécution. Dans le cas d'un conteneur *Docker*, l'image devient un conteneur lorsqu'elle fonctionne sur le *Docker daemon*, ou *Docker Engine*. L'application « conteneurisée » fonctionnera toujours de la même manière, quel que soit l'hôte. *Docker* peut s'exécuter soit sur un serveur, soit dans un *Cloud* privé, ou public, apportant une capacité de portabilité et de flexibilité dans le déploiement et l'exécution de l'application.

Docker repose sur une architecture client-serveur (figure 1.1). Un client *Docker* communique avec le *Docker daemon* pour gérer la construction et le fonctionnement des conteneurs *Docker* ainsi que la distribution des images *Docker* issues d'un *Docker Registry*, public ou privé.

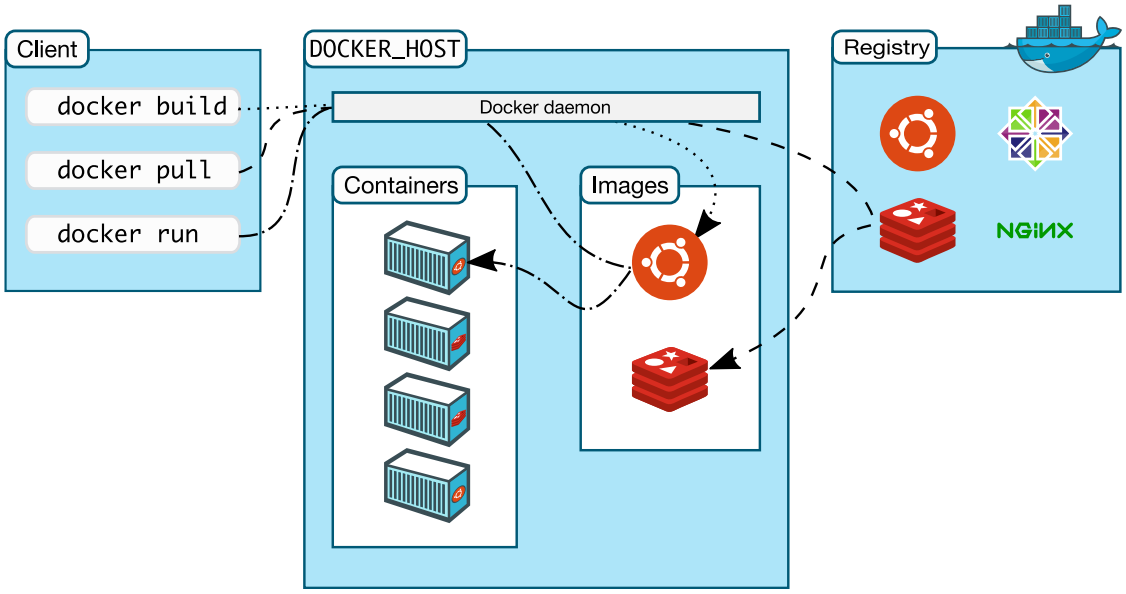

FIGURE 1.1 – Architecture Docker

1.2 Périmètre de l'étude

Ce document a pour objectif de présenter les bonnes pratiques de sécurité relatives au déploiement et à l'exécution de conteneur *Docker*. De ce fait, le *Docker daemon* et la gestion des images *Docker* sont hors périmètre de l'étude.

Bien que hors périmètre, les *Linux Security Modules* ou *LSM* (AppArmor [1], SELinux [2], TOMOYO [3], etc.) permettent de renforcer la sécurité des conteneurs [4] *Docker*. Dans un souci de défense en profondeur, l'étude de leur mise en oeuvre ne peut donc qu'être conseillée.

Les tests réalisés, qui ne visent pas l'exhaustivité du sujet, l'ont été sur la version 18.09.3, build 774a1f4, de *Docker Community Edition (CE)*.

1.3 Convention de lecture

Pour certaines recommandations, il est proposé plusieurs solutions techniques qui se distinguent par leur niveau de sécurité. Le lecteur a ainsi la possibilité de choisir une solution en adéquation avec ses besoins de sécurité.

En outre, dans une démarche itérative de sécurisation, les différents niveaux de sécurité proposés peuvent permettre de fixer une cible technique et d'identifier les étapes pour l'atteindre.

Ainsi, les recommandations sont présentées de la manière suivante :

1. https://docs.docker.com/engine/security/apparmor/.
2. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_atomic_host/7/html/container_security_guide/docker_selinux_security_policy.
3. https://www.kernel.org/doc/html/v4.15/admin-guide/LSM/tomoyo.html.
4. https://docs.docker.com/engine/security/security/.

- R*x* constitue une recommandation à l'état de l'art ;
- R*x* - et R*x* - - constituent des recommandations alternatives à R*x*, d'un niveau de sécurité moindre et respectivement décroissant.
- R*x* + constitue une recommandation alternative à R*x*, ayant des objectifs de sécurité plus élevés.

Par ailleurs, dans ce guide, l'utilisation du verbe « *devoir* » est volontairement plus prescriptive que la formulation « *il est recommandé* ».

1.4 Risques résiduels

Les risques résiduels suivants sont à prendre en compte :

- *Docker Community Edition (CE)* n'a pas été évalué par l'ANSSI et ne dispose donc pas de visa de sécurité. De ce fait, l'ANSSI ne garantit pas la robustesse de ce produit du point de vue de la sécurité ;
- les processus *Docker Community Edition (CE)* s'exécutent avec les droits de l'utilisateur *root* (*UID=0*). C'est en particulier le cas du *Docker daemon*, qui permet d'administrer et de configurer *Docker*, et qui est fortement exposé à des flux non maîtrisés. Un durcissement du système et des services *Docker* conformément au guide « Recommandations de configuration d'un système *GNU/Linux* » [1] publié par l'ANSSI doit être réalisé pour réduire la surface d'attaque et les risques de compromission de l'hôte ;
- *Docker Community Edition (CE)* présente peu de vulnérabilités connues. La plupart d'entre elles concernent la création de conteneur à partir d'une image malveillante. Cependant, la sécurité de *Docker* repose principalement sur les mécanismes de sécurité du noyau et sur des composants externes. Des vulnérabilités les affectant peuvent donc être exploitées pour compromettre *Docker* ou son hôte.

2
Recommandations

2.1 Cloisonnement du conteneur

Docker crée les systèmes de fichiers et les fichiers indispensables au fonctionnement de chaque conteneur et au cloisonnement des conteneurs entre eux, ainsi que par rapport à l'hôte :

- un système de fichiers racine (*/*) ou *root filesystem* ;
- un système de fichiers des périphériques (*/dev*) avec un nombre restreint de périphériques matériels ou pseudo-périphériques (ou *devices*). Par défaut, les conteneurs sont démarrés [5] sans accès au système de fichiers des périphériques (*/dev*) de l'hôte. L'option *–privileged* [6] permet cependant de rétablir cet accès ;
- un système de fichiers processus (*/proc*) ;
- un système de fichiers virtuel (*/sys*) ;
- un fichier */etc/resolv.conf* contenant la configuration DNS du conteneur ;
- un fichier */etc/hostname* contenant le nom du conteneur ;
- un fichier */etc/hosts* contenant la définition des adresses *loopback* et l'adresse IP attribuée au conteneur.

R1

Isoler les systèmes sensibles de fichiers de l'hôte

Chaque conteneur doit être démarré sans partager les systèmes de fichiers sensibles de l'hôte : le système de fichiers racine (*/*), des périphériques (*/dev*), des processus (*/proc*) ou virtuel (*/sys*). Par conséquent, l'option *–privileged* ne doit pas être utilisée.

Si besoin, *Docker* permet avec l'option *–device* [7] d'ajouter un périphérique de l'hôte au conteneur, par défaut, en lecture et écriture. Cette option permet de créer une FIFO, un fichier spécial en mode caractère, ou un fichier spécial en mode bloc, avec ce périphérique. Les options complémentaires *rwm* (*read/write/mknod* [8]) permettent de limiter l'accès à ce périphérique.

R2

Restreindre l'accès aux périphériques de l'hôte

Si un conteneur doit être démarré avec un périphérique de l'hôte, celui-ci doit être ajouté au conteneur avec l'option *–device* et les options complémentaires *rwm* spécifiées pour en limiter l'accès au strict minimum nécessaire.

5. https://docs.docker.com/engine/reference/run/.
6. https://docs.docker.com/engine/reference/run/#runtime-privilege-and-linux-capabilities.
7. https://docs.docker.com/engine/reference/commandline/run/.
8. http://man7.org/linux/man-pages/man2/mknod.2.html.

Par défaut, au démarrage de *Docker*, un réseau de type « pont réseau » ou *bridge*, *docker0*, est créé par le processus *Docker* pour séparer le réseau de l'hôte du réseau des conteneurs.

Lors de la création d'un conteneur, *Docker* le connecte, par défaut, au réseau *docker0*. Par conséquent, tous les conteneurs sont connectés au réseau *docker0* et sont en mesure de communiquer entre eux.

R3

Interdire la connexion du conteneur au réseau bridge docker0

Le service *Docker* doit être configuré pour démarrer un conteneur sans le connecter, par défaut, au réseau *bridge*, *docker0*, avec l'option *–bridge=none* (en ligne de commande ou dans le fichier de configuration).

L'option *–network host* [9] permet au conteneur de partager, lors de sa création, le réseau de l'hôte.

R4

Isoler l'interface réseau de l'hôte

Chaque conteneur doit être démarré sans partager l'interface réseau de l'hôte. L'option *–network host* ne doit pas être utilisée.

Docker permet de créer, à partir de la commande *docker network create [OPTIONS] NETWORK*, un réseau dédié pour « exposer » un port de l'interface réseau de l'hôte ou pour connecter les interfaces réseau de deux ou plusieurs conteneurs à ce réseau avec l'option *–network*.

R5

Créer un réseau dédié pour chaque connexion

Si le conteneur doit « exposer » un port de l'interface réseau de l'hôte ou doit être connecté à un ou plusieurs autres conteneurs, un réseau dédié doit être créé pour chaque connexion.

Par exemple sur la figure 2.1, pour la création d'un serveur Web « conteneurisé » exposant un port de l'interface réseau de l'hôte et utilisant un serveur de base de données « conteneurisé », *Docker* connecte :

- le réseau *bridge web* dédié à l'interface réseau du conteneur, et définit les règles adéquates *iptables* pour rediriger, via le réseau *web*, les flux entrants sur l'interface réseau de l'hôte (*IP* 192.168.0.2) à destination du serveur Web vers l'interface réseau du conteneur (*IP* 172.17.0.2) connectée sur le réseau *web*,
- le réseau *bridge db* dédié aux interfaces réseau des deux conteneurs pour permettre l'échange de données, via le réseau *db*, entre le serveur de base de données (*IP* 10.0.0.2) et le serveur Web (*IP* 10.0.0.1).

9. https://docs.docker.com/network/host/.

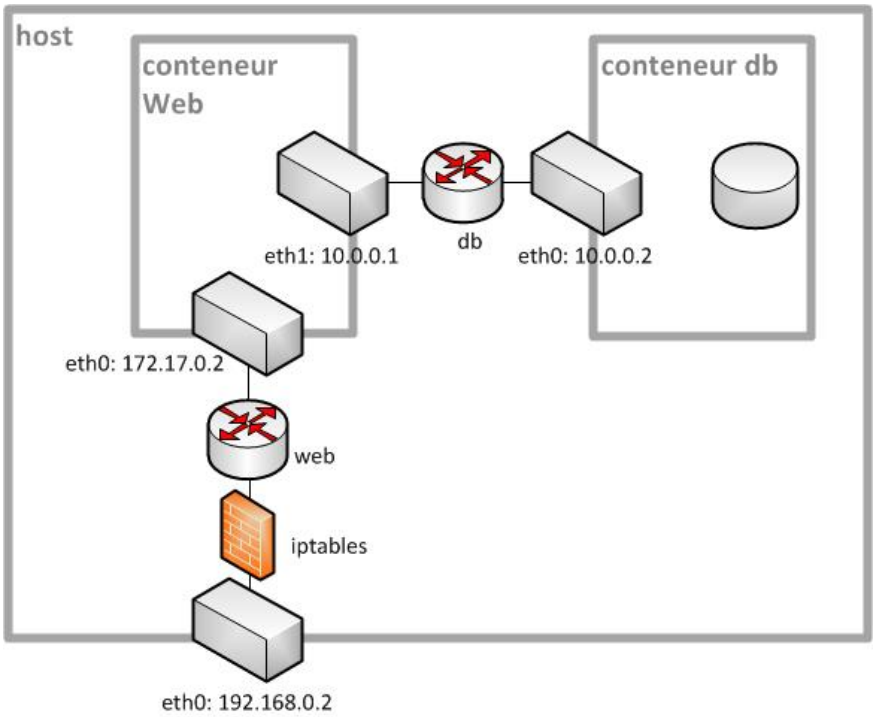

FIGURE 2.1 – Réseaux dédiés de type « pont réseau » ou *bridge*

2.2 Cloisonnement des ressources

Docker exploite la fonctionnalité *Kernel Namespaces* offerte par le noyau *Linux* pour cloisonner les conteneurs.

Lors de la création d'un conteneur, *Docker* lui associe, par défaut, l'ensemble des *namespaces* suivants pour cloisonner les ressources :

- **Namespace MNT** pour les points de montage du système de fichiers : le conteneur accède uniquement à ses propres points de montage et ne voit pas ceux de l'hôte ;
- **Namespace PID** pour les processus : le conteneur accède uniquement à ses propres processus, et n'a donc pas connaissance des processus s'exécutant sur l'hôte ou dans d'autres conteneurs. La numérotation de processus (PID) est réinitialisé et le processus principal du conteneur possède le PID 1 ;
- **Namespace NET** pour les interfaces réseau : le conteneur accède uniquement à ses propres interfaces réseau et ne voit pas celles de l'hôte ou des autres conteneurs ;
- **Namespace IPC** pour la communication inter-processus de type SysV : le conteneur bénéficie de canaux IPC dédiés (objets IPC System V, files de messages POSIX) et ne peut pas accéder aux autres canaux définis sur l'hôte ou dans un autre conteneur ;
- **Namespace UTS** pour le *hostname* et le nom de domaine *NIS* : le conteneur possède son propre *hostname* et son propre nom de domaine *NIS* distincts de ceux de l'hôte ou des autres conteneurs.

Docker permet cependant, avec les options au démarrage du conteneur, de partager un ou plusieurs *namespaces* avec l'hôte ou un autre conteneur :

- *–pid="home"* ou *–pid="container :<name|id>"* pour le *namespace PID* ;
- *–ipc="home"* ou *–ipc="container :<name|id>"* pour le *namespace IPC* ;
- *–uts="home"* pour le *namespace UTS*.

R6

Dédier des namespaces PID, IPC et UTS pour chaque conteneur

Chaque conteneur doit être démarré avec les *namespaces PID, IPC* et *UTS* distincts de l'hôte et des autres conteneurs et donc sans les options *–pid*, *–ipc* et *–uts*.

Par défaut, *Docker* n'associe pas de **namespace USER ID** distinct de celui de l'hôte à un conteneur pour cloisonner les utilisateurs. Le conteneur partage donc ses utilisateurs avec ceux de l'hôte et des autres conteneurs. En cas de compromission du conteneur, un attaquant ayant compromis le compte de l'utilisateur *root* (*UID=0*) du conteneur est aussi l'utilisateur *root* (*UID=0*) de l'hôte et des autres conteneurs.

Docker permet de démarrer un conteneur avec son propre *namespace USER ID* avec l'option *–userns-remap* [10].

Cependant, cette fonctionnalité n'emporte pas l'adhésion de l'ensemble de la communauté des professionnels de la sécurité [11] et la récente découverte de la vulnérabilité *CVE-2018-18955* [12] montre que la sécurité de cette fonctionnalité n'est pas encore mature. Son usage s'accompagne donc de quelques précautions.

R7

Dédier un namespace USER ID pour chaque conteneur

Le service *Docker* doit être configuré pour démarrer tous les conteneurs avec un *namespace USER ID* distinct de l'hôte et des autres conteneurs avec l'option *–userns-remap* (en ligne de commande ou dans le fichier de configuration). Cette configuration doit s'accompagner d'une interdiction de créer de nouveau *namespace USER ID* à l'intérieur du conteneur, par exemple, avec le profil *Seccomp* [13] par défaut qui définit une politique de sécurité pour désactiver cette fonctionnalité à l'intérieur du conteneur.

Dans la version officielle du noyau *Linux* (*Vanilla* [14]), à la date de rédaction de cette fiche technique, la création d'un *Namespace USER ID* n'est pas restreinte à l'utilisateur *root* (*UID=0*). En l'absence de mécanisme de sécurité, n'importe quel utilisateur de l'hôte, et donc un utilisateur d'un conteneur, peut créer un *Namespace USER ID*. Le principe de défense en profondeur, recommande de mettre en œuvre, s'ils existent, les mécanismes de sécurité permettant de restreindre la création d'un *Namespace USER ID*, en complément de R7.

10. https://docs.docker.com/engine/security/userns-remap/.
11. https://lwn.net/Articles/673597/.
12. https://nvd.nist.gov/vuln/detail/CVE-2018-18955.
13. https://docs.docker.com/engine/security/seccomp/.
14. https://www.kernel.org/.

R7 +

Restreindre la création des Namespace USER ID à l'utilisateur root

Sur les systèmes *Linux* à base *Debian*, la configuration système *kernel.unprivileged_userns_clone* [15] doit être égale à 0 (valeur par défaut) pour restreindre la création d'un *Namespace USER ID* au seul utilisateur *root* (*UID=0*).

Le cloisonnement des ressources avec le *namespace USER ID* présente quelques limitations [16] connues.

R7 -

Restreindre le partage du Namespace USER ID de l'hôte

Lorsqu'un conteneur doit être démarré en partageant le *namespace USER ID* avec celui de l'hôte, et que le service *Docker* est configuré pour démarrer tous les conteneurs avec un *namespace USER ID* distinct de celui de l'hôte, il est possible d'utiliser l'option *–userns=host* au démarrage du conteneur. Dans ce cas, la menace de compromission de l'utilisateur *root* (*UID=0*) de l'hôte par le conteneur est à considérer.

2.3 Restriction des privilèges

Docker permet de démarrer les conteneurs (conteneur non privilégié) sans les privilèges de l'utilisateur *root* (*UID=0*), ou avec un ensemble restreint de *capabilities* [17].

Par défaut, les conteneurs sont démarrés avec les *capabilities* [18] suivantes, qui ne sont pas toutes nécessaires [5] et qui ne devraient donc pas être utilisées en production :

- **CAP_AUDIT_WRITE :** le sous-système audit du noyau ne gère pas actuellement les espaces de nommage ou *namespaces* ;
- **CAP_CHOWN** ;
- **CAP_DAC_OVERRIDE** ;
- **CAP_FOWNER :** les commandes nécessitant cette *capability* doivent être exécutées lors de la construction du conteneur dans le *Dockerfile* ;
- **CAP_FSETID** ;
- **CAP_KILL :** sauf si l'application « conteneurisée » envoie des signaux *kill* à des processus d'un autre utilisateur et qu'il n'est pas possible de modifier ce comportement à partir de la configuration ;
- **CAP_NET_BIND_SERVICE :** l'application « conteneurisée » doit être configurée sur des ports non priviligiés, et une redirection des ports priviligiés de l'hôte vers ces ports doit être réalisée avec l'option *–p* [19], par exemple pour un serveur Web « conteneurisé », une redirection du port priviligié 80 de l'hôte vers le port non priviligié 8080 du conteneur ;
- **CAP_NET_RAW** ;

15. https://security-tracker.debian.org/tracker/CVE-2016-8655.
16. https://docs.docker.com/engine/security/userns-remap/#user-namespace-known-restrictions.
17. https://manpages.ubuntu.com/manpages/trusty/fr/man7/capabilities.7.html.
18. https://github.com/moby/moby/blob/1308a3a99faa13ff279dcb4eb5ad23aee3ab5cdb/oci/caps/defaults.go.
19. https://docs.docker.com/engine/reference/run/#expose-incoming-ports.

- **CAP_MKNOD :** la création du périphérique doit être réalisée lors du démarrage du conteneur avec l'option *–device* ;
- **CAP_SETFCAP** ;
- **CAP_SETPCAP :** sauf si l'application « conteneurisée » modifie les *capability*, généralement en démarrant avec les *capability* nécessaires pour exécuter des commandes privilégiées avant de les supprimer. Certaines applications permettent, cependant, de modifier ce comportement à partir de la configuration ;
- **CAP_SETUID/CAP_SETGID :** sauf si l'application « conteneurisée » modifie les UID/GIDs, généralement en démarrant avec l'utilisateur *root* (*UID=0*) pour exécuter des commandes privilégiées avant de changer d'utilisateur/groupe d'utilisateurs. Certaines applications permettent, cependant, de modifier ce comportement à partir de la configuration ;
- **CAP_SYS_CHROOT :** sauf si l'application « conteneurisée » utilise la commande *chroot* et qu'il n'est pas possible de modifier ce comportement à partir de la configuration.

R8

Interdire l'utilisation des capabilities

Chaque conteneur doit être démarré sans aucune *capability* avec l'option *–cap-drop=ALL*.

Brad Spengler (alias *Spender*) identifie les 19 *capabilities* « sensibles » suivantes parmi les 35 disponibles qui équivalent aux droits de l'utilisateur *root* (*UID=0*) (*full root*) [20] :

- CAP_AUDIT_CONTROL
- CAP_CHOWN
- CAP_DAC_OVERRIDE
- CAP_DAC_READ_SEARCH
- CAP_FOWNER
- CAP_FSETID
- CAP_IPC_OWNER
- CAP_MKNOD
- CAP_SETFCAP
- CAP_SETGID
- CAP_SETPCAP
- CAP_SETUID
- CAP_SYS_ADMIN
- CAP_SYS_CHROOT
- CAP_SYS_BOOT
- CAP_SYS_MODULE

20. https://forums.grsecurity.net/viewtopic.php?f=7&t=2522#p10271.

- CAP_SYS_PTRACE
- CAP_SYS_RAWIO
- CAP_SYS_TTY_CONFIG

R8 -

Limiter l'utilisation des capabilities

Un conteneur peut être démarré avec les *capabilities* strictement nécessaires, non sensibles, avec les options *–cap-drop=ALL* et *–cap-add={"capability A"}*.

2.4 Limitation des accès aux ressources

Docker exploite les *Control Groups* (ou *CGroups*) qui permettent de limiter, mesurer et prioriser l'accès aux ressources système (CPU, mémoire, I/O disque,etc.) pour chaque conteneur. Lors de la création d'un conteneur, *Docker* lui associe, par défaut, des *Control Groups* dédiés. L'option *–cgroup-parent* [21] permet cependant au conteneur de partager les *Control Groups* de l'hôte.

R9

Dédier les Control groups pour chaque conteneur

Chaque conteneur doit être démarré avec des *Control Groups* distincts de l'hôte et donc sans l'option *–cgroup-parent*.

Par défaut, un conteneur n'a pas de contraintes de ressources et peut utiliser toutes les ressources disponibles sur l'hôte.

R10

Limiter l'utilisation de la mémoire de l'hôte pour chaque conteneur

Chaque conteneur doit être démarré avec une limite maximale d'utilisation de la mémoire de l'hôte, avec l'option *–memory*, et une limite maximale d'utilisation de la mémoire *SWAP* de l'hôte avec l'option *–memory-swap* [22].

R11

Limiter l'utilisation du CPU de l'hôte pour chaque conteneur

Chaque conteneur doit être démarré avec une limite maximale d'utilisation du CPU de l'hôte avec l'option *–cpus*, ou avec les options *–cpu-period* et *–cpu-quota*.

L'espace disque des conteneurs est partagé avec l'espace disque de l'hôte. *Docker* exploite le mécanisme des systèmes de fichiers de type *union filesystem* (*aufs*, *btrfs*, *zfs*, etc.) pour construire le système de fichiers racine (*/*), ou *root filesystem*, du conteneur. Tous les calques ou couches (*layers*) sont en lecture seule, sauf la dernière couche (*Thin R/W layer*) qui est en lecture et écriture (figure 2.2).

Par défaut, le système de fichiers racine d'un conteneur est en lecture et écriture. Tous les fichiers présents peuvent être modifiés, et de nouveaux fichiers peuvent être créés à l'intérieur d'un conteneur. Les fichiers modifiés ou créés sont stockés sur la dernière couche (*Thin R/W layer*).

21. https://docs.docker.com/v17.12/engine/reference/commandline/dockerd/.
22. https://docs.docker.com/config/containers/resource_constraints/#--memory-swap-details.

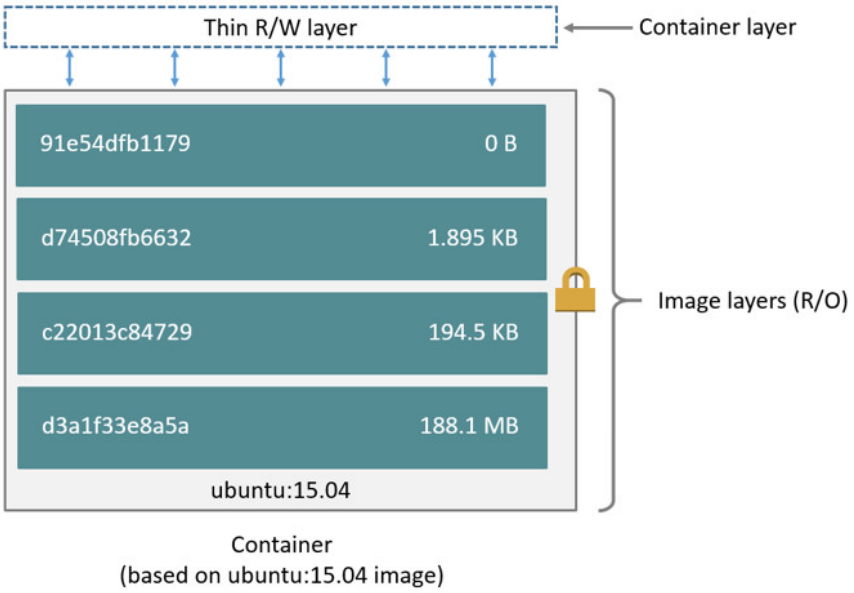

FIGURE 2.2 – Conteneur et couches

En cas de compromission du conteneur, et en l'absence de restriction d'usage ou de cloisonnement de l'espace de stockage du conteneur, la menace d'un déni de service d'un conteneur vis-à-vis de l'hôte et des autres conteneurs est à considérer, notamment avec la saturation de l'espace de stockage du conteneur compromis.

R12

Restreindre en lecture le système de fichiers racine de chaque conteneur

Chaque conteneur doit être démarré avec son système de fichiers racine, ou *root filesystem*, en lecture seule avec l'option *–read-only*.

R12 -

Limiter l'écriture de l'espace de stockage de chaque conteneur

Un conteneur peut être démarré avec une limite maximale d'utilisation de l'espace disque de l'hôte pour son système de fichiers racine, ou *root filesystem*, en lecture et écriture avec l'option *–storage-opt*.

R12 - -

Limiter l'écriture de l'espace de stockage de l'ensemble des conteneurs

Un conteneur peut être démarré avec son système de fichiers racine, ou *root filesystem* en lecture et écriture, dès lors que la zone de stockage local de *Docker*, par défaut */var/lib/docker/* sur *Linux*, est une partition dédiée et distincte des partitions de l'hôte. Dans ce cas, la menace d'un déni de service d'un conteneur vis-à-vis d'un autre conteneur est à considérer.

Les applications « conteneurisées » produisent au plus trois types de données :

- les **données non persistantes ou temporaires** pouvant être supprimées à chaque arrêt du conteneur ;

- les **données persistantes** devant être conservées après chaque arrêt ou destruction du conteneur ;
- les **données partagées** avec l'hôte ou avec un ou plusieurs autres conteneurs.

R13

Créer un système de stockage pour les données non persistantes

Les répertoires contenant des données non persistantes ou temporaires doivent être montés à l'aide d'un montage *bind tmpfs* avec l'option *–mount type=tmpfs* et les options de *tmpfs* [23], *tmpfs-size* et *tmpfs-mode*.

Information

Par défaut, le type de montage *bind tmpfs* est en lecture et écriture pour tout le monde avec le *sticky bit* positionné (mode 1777) et sans limite de taille.

R14

Créer un système de stockage pour les données persistantes ou partagées

Les répertoires contenant des données persistantes ou partagées avec l'hôte, ou avec d'autres conteneurs, doivent être montés à l'aide de :

- un montage *bind mount* [24], avec une limitation de l'espace disque utilisable (option *–mount type=bind,o=size*), ou,
- un montage *bind volume* [25] d'une partition dédiée et distincte des partitions utilisées par l'hôte, ou d'un volume *Docker*, avec une limitation spécifiée de l'espace disque (option *–mount type=volume*).

Si les données persistantes ou partagées ne doivent pas être modifiées par le conteneur, les répertoires doivent être montés en lecture seule avec l'option *read-only* [26] [27].

R15

Restreindre l'accès aux répertoires et aux fichiers sensibles

Si un conteneur doit partager des répertoires ou des fichiers sensibles avec l'hôte, ou un autre conteneur, il doit être démarré en restreignant ses accès au strict minimum nécessaire.

2.5 Journalisation du conteneur

Conformément à la recommandation R5 de la note technique « Recommandations de sécurité pour la mise en œuvre d'un système de journalisation » [2] publiée par l'ANSSI, les journaux doivent être automatiquement exportés sur une machine physique différente de celle qui les a générés. *Docker* permet d'exporter les flux standard *STDERR* et *STDOUT* du conteneur.

23. https://docs.docker.com/storage/tmpfs/#tmpfs-options.
24. https://docs.docker.com/storage/bind-mounts/.
25. https://docs.docker.com/storage/volumes/.
26. https://docs.docker.com/storage/bind-mounts/#use-a-read-only-bind-mount.
27. https://docs.docker.com/storage/volumes/#use-a-read-only-volume.

Si l'application « conteneurisée » produit des journaux d'évènements sous la forme de fichiers dédiés, souvent stockés dans le répertoire */var/log*, ou l'un de ses sous-répertoires. Une redirection de ces fichiers vers les flux standard, *STDERR* et *STDOUT*, peut être mise en œuvre lors de la création du conteneur et ainsi permettre l'utilisation de la journalisation des conteneurs par *Docker*. C'est ce qui est mis en place par défaut sur l'image officielle *Apache* [28].

R16

Exporter les journaux avec Docker

Le service *Docker* doit être configuré pour collecter les journaux d'évènements issus de l'application « conteneurisée » et les exporter vers un serveur centralisé avec l'option *–log-driver=<logging driver>* (en ligne de commande ou dans le fichier de configuration).

Information

Docker supporte plusieurs *drivers* de logs [29] et permet aussi d'en créer de nouveaux [30].

Si l'application « conteneurisée » ne produit pas de journaux sur les flux standard, *STDERR* ou *STDOUT*, et ne permet pas une redirection des journaux vers ces flux, la mise en œuvre d'une solution distincte de *Docker* à l'intérieur du conteneur doit être envisagée.

R16 -

Exporter les journaux depuis l'intérieur du conteneur

L'export des journaux d'évènements issus de l'application « conteneurisée » peut être mise en œuvre à l'intérieur du conteneur par une solution distincte de *Docker*.

28. https://github.com/docker-library/php/blob/cf02ea0f79cf7cff3853f6d4254b95b65d44cc12/7.2/stretch/apache/Dockerfile.
29. https://docs.docker.com/config/containers/logging/configure/.
30. https://docs.docker.com/engine/extend/plugins_logging.

A
Correspondance avec les recommandations du Center for Internet Security

Le *Center for Internet Security*[31] (*CIS*) a publié un document de référence [4] de recommandations conçues pour aider les administrateurs système, les professionnels de la sécurité et de l'audit, etc. à établir une configuration de base sécurisée pour *Docker CE*.

La correspondance entre les recommandations au chapitre 2 et ces recommandations est définie dans le tableau suivant :

Recommandations	*CIS Docker Community Edition Benchmark v1.1.0*
R16	2.12 *Ensure centralized and remote logging is configured*
R1	5.4 *Ensure privileged containers are not used* 5.31 *Ensure the Docker socket is not mounted inside any containers*
R2	5.17 *Ensure host devices are not directly exposed to containers*
R3	5.29 *Ensure Docker's default bridge docker0 is not used*
R4	5.9 *Ensure the host's network namespace is not shared*
R6	5.15 *Ensure the host's process namespace is not shared* 5.16 *Ensure the host's IPC namespace is not shared* 5.20 *Ensure the host's UTS namespace is not shared*
R7	2.8 *Enable user namespace support* 5.30 *Ensure the host's user namespaces is not shared*
R8	5.3 *Ensure Linux Kernel Capabilities are restricted within containers*
R9	5.24 *Ensure cgroup usage is confirmed*
R10	5.10 *Ensure memory usage for container is limited*
R12	5.12 *Ensure the container's root filesystem is mounted as read only*
R15	5.5 *Ensure sensitive host system directories are not mounted on containers*

Un conteneur *Docker*[32] est disponible pour réaliser une analyse de sécurité de l'environnement *Docker* d'un hôte au regard de ces recommandations.

31. https://www.cisecurity.org.
32. https://www.cisecurity.org/benchmark/docker/.

Liste des recommandations

Bibliographie

[1] *Recommandations de configuration d'un système GNU/Linux*.
Guide Version 1.2, ANSSI, février 2019.
https://www.ssi.gouv.fr/reco-securite-systeme-linux.

[2] *Recommandations de sécurité pour la mise en œuvre d'un système de journalisation*.
Note technique DAT-NT-012/ANSSI/SDE/NP v1.0, ANSSI, décembre 2013.
https://www.ssi.gouv.fr/journalisation.

[3] *Licence ouverte / Open Licence v2.0*.
Page web, Mission Etalab, 2017.
https://www.etalab.gouv.fr/licence-ouverte-open-licence.

[4] Center for Internet Security.
CIS Docker Community Edition Benchmark v1.1.0, June 2017.
https://www.cisecurity.org/benchmark/docker/.

[5] Dan Walsh.
Secure Your Containers with this One Weird Trick, October 2016.
https://rhelblog.redhat.com/2016/10/17/secure-your-containers-with-this-one-weird-trick/.

ANSSI-FT-082
Version 1.0 - 23/09/2020

AGENCE NATIONALE DE LA SÉCURITÉ DES SYSTÈMES D'INFORMATION

ANSSI - 51, boulevard de La Tour-Maubourg, 75700 PARIS 07 SP
www.ssi.gouv.fr / conseil.technique@ssi.gouv.fr

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE
Premier ministre
SGDSN