Système minimal de gestion de conteneurs

Hind Malti

Informatique Microélectronique et Automatique

Mai 2019

Remerciements

Je tiens à remercier mes encadrants M. Redon Xavier et M. Vantroys Thomas pour leur patience, leur disponibilité, leurs encouragements et surtout leurs judicieux conseils, qui ont contribué à la réussite de ce projet.

J'adresse mes sincères remerciements à tous les professeurs et toutes les personnes qui par leurs paroles, leurs écrits, leurs conseils et leurs critiques ont guidé mes réflexions et ont accepté de répondre à mes questions durant mes recherches.

Table des matières

Remerciements **2**

Table des matières **3**

1 Introduction **5**
1.1 Description et objectifs 5
1.2 Les objectifs 7

2 Analyse du projet **7**
2.1 Positionnement par rapport à l'existant 7
2.2 Analyse du premier concurrent 8
2.3 Analyse du second concurrent 8
2.4 Docker comparé à Rkt 9
2.4.1 Sécurité de l'image du conteneur 9
2.4.2 Prévention des attaques d'élévation de privilèges «root» 9
2.4.3 Portabilité à d'autres systèmes de conteneurs 10
2.5 Différence avec Docker 10
2.6 Le matériel et les logiciels 10

3 État de l'art de la conteneurisation **11**
3.1 Définition de la conteneurisation 11
3.2 L'utilité de la conteneurisation 11
3.3 Évolution 11
3.3.1 Hello VMware 12
3.3.2 Les conteneurs 13
3.3.3 1979 : Unix V7 13
3.3.4 2000 : JBS FreeBSD 14
3.3.5 2001 : Linux VServer 14
3.3.6 2004 : conteneurs Solaris 15
3.3.7 2005 : Open VZ (Open Virtuzzo) 15
3.3.8 2006 : conteneurs de processus 15
3.3.9 2008 : LXC 15
3.3.10 2011 : Warden 16
3.3.11 2013 : LMCTFY 16
3.3.12 2013 : Docker 16
3.3.13 2017 : Les outils de conteneur deviennent matures . . . 17
3.3.14 Kubernetes grandit 18
3.3.15 Docker : un LXC augmenté 18
3.4 Différence avec la virtualisation 19

4 Les images Docker **20**
4.1 Les images sont généralement petites 20
4.2 Nom de l'image 21
4.2.1 Dépôts officiels et non officiels 22
4.2.2 Nom et marquage des images 22
4.2.3 Images et couches 23
4.2.4 Les hachages d'image (digests) 24
4.2.5 Images multi-architecturales 25
4.2.6 Supprimer des images 25

5 Travail réalisé **26**
5.1 Explication du code et sa structure 26
5.2 Fonctionnement de l'application 27
5.2.1 Images 27
5.2.2 Bridges 28
5.2.3 Conteneurs 29

6 Mandataire inverse **29**
6.1 Options ajoutées au projet 31
6.2 Améliorations possibles du projet 31

7 Annexes **32**
7.1 Ext4 32
7.2 Debootstrap 32
7.3 Cgroups : (control groups) 32
7.4 Unshare 33
7.5 chroot (change root) 33
7.6 Les Veth 34
7.7 Manifeste 35
7.8 API REST 35
7.9 Les SWAP 36
7.10 Les Namespace 36
7.10.1 Propriétés 36
7.11 Système de fichiers 37

1 Introduction

1.1 Description et objectifs

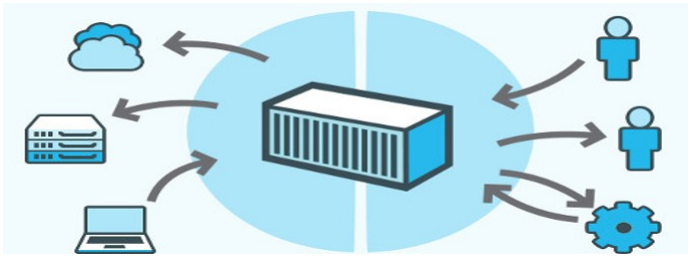
L'intitulé de mon projet est "Système minimal de gestion de conteneurs", il consiste à réaliser une solution logicielle permettant de créer, détruire, configurer et lister des conteneurs.

Un conteneur Linux est un ensemble de processus qui sont isolés du reste du système. Un conteneur s'exécute à partir d'une image distincte qui fournit tous les fichiers nécessaires à la prise en charge des processus qu'il contient :

- Les caractéristiques systèmes : taille mémoire maximale, nombre de CPUs utilisés, débit maximal etc.
- Une configuration réseau : liste de commutateurs virtuels de rattachement avec leurs adresses.

En fournissant une image qui contient toutes les dépendances d'une application, le conteneur assure donc la portabilité de l'application entre les divers environnements (développement, test puis production) ainsi que la mise à l'échelle face à la montée en charge.

Les logiciels doivent pouvoir fonctionner de manière fiable sur différents environnements, les conteneurs sont une solution novatrice pour résoudre ce problème, dorénavant on peut déployer de façon aisée en partant de l'ordinateur du développeur jusqu'aux serveurs de production en passant par les serveurs de développement. Cela pourrait être des machines physiques ou bien des machines virtuelles dans un Cloud privé ou public.



Les avantages majeurs de l'utilisation des conteneurs sont :

- La taille : En effet un conteneur prend quelques dizaines de mégaoctets de taille, contrairement à une VM qui peut prendre plusieurs gigaoctets à cause de son système d'exploitation. Ce qui rend l'utilisation des conteneurs très populaire sur les serveurs qui peuvent en contenir plusieurs. Et dans le cloud où l'on paie ce qu'on consomme, cela est très intéressant pour optimiser les coûts.

- Le temps : Une application conteneurisée peut être démarrée instantanément et quand il est nécessaire peut disparaître libérant ainsi des ressources sur son hôte tandis que les machines virtuelles peuvent prendre plusieurs minutes pour démarrer leurs systèmes d'exploitation et commencer à exécuter les applications qu'elles hébergent.
- La modularité : Plutôt que d'exécuter une application complexe entière dans un seul conteneur, l'application peut être divisée en modules (tels que la base de données, l'interface de l'application, etc.) Les applications créées de cette manière sont plus faciles à gérer car chaque module est relativement simple et des modifications peuvent être apportées aux modules sans avoir à reconstruire l'application entière. Étant donné que les conteneurs sont légers, les modules individuels (ou micro services) ne peuvent être instanciés que lorsqu'ils sont nécessaires et sont disponibles presque immédiatement.

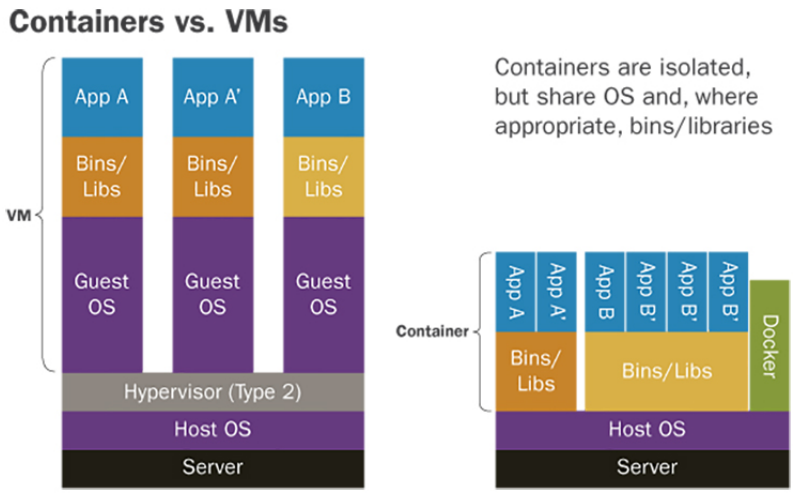
Avec une telle architecture, on suit la philosophie UNIX du "KISS"

- Les applications n'ont pas de dépendance système.
- Les mises à jours sont déployables facilement.
- La consommation de ressources est optimisée.

Les containers sont donc proches des machines virtuelles, mais présentent des avantages importants. Alors que la virtualisation consiste à exécuter de nombreux systèmes d'exploitation sur un seul et même système, les conteneurs se partagent le même noyau de système d'exploitation et isolent les processus de l'application du reste du système.

Plutôt que de virtualiser le hardware comme l'hyperviseur, le conteneur virtualise le système d'exploitation. Il est donc nettement plus efficient qu'un hyperviseur en termes de consommation des ressources système.

Concrètement, il est possible d'exécuter près de 4 à 6 fois plus d'instances d'applications avec un conteneur qu'avec des machines virtuelles comme Xen ou KVM sur le même hardware.



1.2 Les objectifs

L'objectif de ce projet est de réaliser un écosystème minimal permettant de gérer des images et des instances de conteneurs.

- Les commandes seront réalisées en shell et les informations stockées dans des fichiers textes. Une configuration réseau devra être mise en place.
- Il est demandé de réaliser un système de gestion de conteneurs comme Docker mais en plus simple.
- Il est interdit de lancer un démon ou de perturber le réseau de l'hôte avec des règles de filtrage de paquets.
- Un contrôle des instances se fera par "crontab" et les instances seront connectées via des commutateurs virtuels Linux classiques.
- La gestion sophistiquée des images à base de système de fichiers à couche n'est pas permise.
- Avant de lancer un conteneur le système de fichiers (un fichier lui-même) sera copié et le conteneur, sera démarré sur la copie.

2 Analyse du projet

2.1 Positionnement par rapport à l'existant

Lorsqu'on parle aujourd'hui de gestion de conteneurs on parle systématiquement de la solution Docker. Je n'ai bien évidemment pas pour ambition de concurrencer Docker. Le but ici, est de réussir à reproduire un système de gestion minimal ressemblant à Docker en plus simple et de parvenir à le maîtriser.

2.2 Analyse du premier concurrent

Docker Enterprise Edition est sans doute la solution de gestion de conteneurs commerciale la plus connue. Il fournit une plate-forme intégrée, testée et certifiée pour les applications exécutées sur les systèmes d'exploitation Linux ou Windows et les fournisseurs de cloud. L'écosystème de Docker est riche et complet, cela va de la gestion de conteneurs, à l'orchestration (Docker Swarm ou son concurrent Google Kubernetes) tout en passant par la gestion du réseau, de l'intégration du cloud avec Docker machine, des applications multi conteneurs avec Docker Compose.

Aujourd'hui, selon les créateurs de Docker, plus de $3,5$ millions d'applications ont été containérisées en utilisant cette technologie, et plus de 37 milliards d'applications containérisées ont été téléchargées.

De même, d'après le système de monitoring cloud DataDog, $18,8\%$ des utilisateurs avaient adopté la plateforme en 2017. De son côté, RightScale estime que l'adoption de la plateforme dans l'industrie du Cloud a augmenté de 35% en 2017 à 49%en 2018. Des géants comme Oracle et Microsoft l'ont adopté, au même titre que presque toutes les entreprises du Cloud.

2.3 Analyse du second concurrent

Rocket (rkt) est un outil édité par CoreOS et est le concurrent de Docker. C'est un moteur de conteneur d'application développé pour les environnements de production modernes basés sur le cloud. Il présente une approche pod-native, un environnement d'exécution enfichable et une surface bien définie qui le rend idéal pour une intégration avec d'autres systèmes.

L'unité d'exécution principale de rkt est le "pod" : un ensemble d'une ou plusieurs applications s'exécutant dans un contexte partagé (les pods de rkt sont synonymes du concept du système d'orchestration Kubernetes). Rkt permet aux utilisateurs d'appliquer différentes configurations (telles que les paramètres d'isolation) au niveau du pod et au niveau plus granulaire par application. L'architecture de rkt signifie que chaque pod s'exécute directement dans le modèle de processus Unix classique (c'est-à-dire qu'il n'y a pas de démon central), dans un environnement isolé et autonome. rkt implémente un format de conteneur standard moderne et ouvert, la spécification App Container (appc), mais peut également exécuter d'autres images de conteneur, telles que celles créées avec Docker.

Depuis son introduction par CoreOS en décembre 2014, le projet rkt a considérablement mûri et est largement utilisé. Il est disponible pour la plu-

part des principales distributions Linux et chaque version de rkt construit des packages rpm / deb autonomes que les utilisateurs peuvent installer. Ces packages sont également disponibles dans le référentiel Kubernetes pour permettre de tester l'intégration rkt + Kubernetes. rkt joue également un rôle central dans la manière dont Google Container Image et CoreOS Container Linux exécutent Kubernetes. Les éditeurs se concentrent sur la sécurité (le plus gros point faible de Docker), la compatibilité et une intégration aux standards. Le but étant de fournir les mêmes fonctionnalités que docker et être complémentaire.

2.4 Docker comparé à Rkt

2.4.1 Sécurité de l'image du conteneur

L'un des avantages de Docker est qu'il existe un registre public à partir duquel tout le monde peut télécharger des images optimisées du serveur d'applications. Donc, si vous voulez un serveur Nginx optimisé pour une application Web Magento, vous en obtiendrez un du registre Docker. Cependant, il y a un danger caché à cela. Un attaquant peut remplacer une image de serveur par une autre infectée par un logiciel malveillant. Avant la version 1.8, Docker n'avait aucun moyen de vérifier l'authenticité d'une image de serveur. Mais dans la version 1.8, une nouvelle fonctionnalité appelée Docker Content Trust a été introduite pour signer et vérifier automatiquement la signature d'un éditeur. Dans rkt, la vérification de la signature est effectuée par défaut. Ainsi, dès qu'une image de serveur est téléchargée, elle est vérifiée avec la signature de l'éditeur pour voir si elle est falsifiée.

2.4.2 Prévention des attaques d'élévation de privilèges «root»

Docker s'exécute avec les privilèges de super-utilisateur «root» et crée de nouveaux conteneurs en tant que sous-processus. Le problème avec cela est qu'une vulnérabilité dans un conteneur ou un confinement médiocre peut donner à un attaquant un accès de niveau racine au serveur entier. Rkt a proposé une meilleure solution : les nouveaux conteneurs ne sont jamais créés à partir d'un processus root. De cette manière, même si une rupture de conteneur se produit, l'attaquant ne peut pas obtenir les privilèges root.

Flexibilité dans la publication ou le partage d'images. Lors du développement d'applications, il peut être nécessaire de partager des images de conteneur avec vos partenaires technologiques. Si vous souhaitez partager des conte-

neurs Docker, vous devez configurer un registre privé spécial sur vos serveurs ou l'héberger dans un compte payant Docker pour le partager avec vos partenaires. Pour rkt, vous n'avez besoin que de votre serveur Web. Rkt utilise le protocole HTTPS pour télécharger des images et utilise une méta-description sur le serveur Web pour pointer vers l'emplacement. C'est donc un serveur de moins à maintenir et plus facile à accéder pour les partenaires.

2.4.3 Portabilité à d'autres systèmes de conteneurs

De nouveaux systèmes de conteneurs sortent tout le temps. Avec Docker, la migration vers une nouvelle technologie de conteneur peut poser problème, car il utilise un format d'image propriétaire. En revanche, rkt utilise un format de conteneur open source appelé «appc». Ainsi, toute image de serveur créée à l'aide de rkt peut être facilement portée vers un autre système de conteneur à condition qu'elle suive le format ouvert «appc». En adhérant à un standard ouvert, rkt n'impose pas de verrouillage du fournisseur. Cela aide les propriétaires de système à migrer sans peine vers un autre système de conteneur mieux adapté à leurs besoins. [1]

2.5 Différence avec Docker

Docker est un logiciel modulaire qui comprend plusieurs couches. L'une de ses couches est le docker deamon (dockerd), le deamon est codé afin d'exposer une API REST. Ainsi, il est bind sur des sockets au sein de la machine.

L'autre inconvénient de Docker c'est qu'il va manipuler les Iptables [2] afin de pouvoir gérer le réseau des conteneurs, cela peut venir perturber les configurations réseaux déjà présentes sur la machine. Mon système n'utilisera que les commutateurs logiciels (bridges linux) standards. Ainsi, il n'y aura aucune altération du routage des paquets au sein de la machine.

De plus je n'utiliserai pas de gestion sophistiquée des images à base de système de fichiers à couche. Avant de lancer un conteneur le système de fichiers (un fichier lui-même) est copié, et le conteneur est démarré sur cette copie.

2.6 Le matériel et les logiciels

Ce projet ne m'a nécessité aucun matériel étant donné qu'il est purement informatique. J'aurai pu utiliser une machine virtuelle pour tester mes scripts

dessus afin de ne pas dégrader les configurations de la machine sur laquelle je travaille.

3 État de l'art de la conteneurisation

3.1 Définition de la conteneurisation

Un conteneur Linux est un processus ou un ensemble de processus isolés du reste du système. Tous les fichiers nécessaires à leur exécution sont fournis par une image distincte, ce qui signifie que les conteneurs Linux sont portables et fonctionnent de la même manière dans les environnements de développement, de test et de production. Ainsi, ils sont bien plus rapides que les pipelines de développement qui s'appuient sur la réplication d'environnements de test traditionnels.

3.2 L'utilité de la conteneurisation

L'incidence de la virtualisation sur l'informatique moderne est profonde. Elle permet aux entreprises d'améliorer considérablement la rentabilité et la flexibilité des ressources informatiques. Mais la virtualisation a un coût, notamment au niveau de l'hyperviseur et des systèmes d'exploitation invités, qui requièrent chacun de la mémoire et d'éventuelles licences coûteuses. Il en résulte une augmentation de la taille de chaque machine virtuelle, ce qui limite le nombre de VM qu'un serveur peut héberger. L'objectif de la conteneurisation vise à virtualiser les applications sans trop alourdir le système. L'idée n'est pas nouvelle : depuis plusieurs années déjà, des systèmes d'exploitation tels que OpenVZ, FreeBSD, Solaris Containers et Linux-VServer prennent cette fonctionnalité en charge. Mais c'est la récente introduction de plateformes ouvertes, telle que Docker, qui a remis sous les feux de la rampe la conteneurisation et son potentiel en matière d'applications distribuées évolutives.

3.3 Évolution

La plupart des applications tournent sur des serveurs. Dans le passé, nous ne pouvions exécuter qu'une seule application par serveur. Le monde des systèmes ouverts de Windows et Linux n'avait pas les technologies pour exécuter en toute sécurité plusieurs applications sur un même serveur. Bien sûr, chaque fois qu'une entreprise avait besoin d'une nouvelle application, le service informatique allait acheter un nouveau serveur. Et la plupart du

temps personne ne connaissait parfaitement les exigences de performance de la nouvelle application ! Cela signifie que le service informatique devait deviner souvent à la volée le modèle et la taille des serveurs à acheter.

En conséquence, le service informatique a fait la seule chose à faire : acheter de gros serveurs rapides avec beaucoup de résilience. Après tout, la dernière chose que quelqu'un souhaitait, y compris l'entreprise, était des serveurs surchargés. Les serveurs surchargés peuvent entraîner une perte de clients et de revenus. Alors, l'IT généralement achetait des serveurs plus gros que ce qui était réellement nécessaire. Cela a entraîné un énorme nombre de serveurs fonctionnant entre 5 et 10% de leur capacité potentielle. Un gaspillage tragique du capital et des ressources de l'entreprise !

3.3.1 Hello VMware

Pour tout cela, VMware, Inc. a fait un cadeau au monde : la machine virtuelle (VM). Et presque du jour au lendemain, le monde s'est transformé en un endroit bien meilleur ! Nous avons enfin acquis une technologie qui nous permettrait de gérer en toute sécurité plusieurs applications sur un seul serveur tout en étant isolées les unes des autres.

C'était une révolution, il n'était plus nécessaire de se procurer un tout nouveau serveur surdimensionné à chaque fois que l'entreprise demandait une nouvelle application.

Tout à coup, tout est devenu bien plus dynamique, lancer une machine virtuelle pour tester une application ou la lancer en production est rapide, contrairement à l'achat de nouveaux serveurs. De plus, la supervision et le management de machines virtuelles est aisé, on peut facilement dupliquer, supprimer, relancer une machine virtuelle. Ce nouveau dynamisme au sein de l'informatique a été un réel tournant à 180 !

Même si les machines virtuelles sont géniales, elles ne sont pas parfaites ! Le fait que chaque machine virtuelle nécessite son propre système d'exploitation dédié est un défaut majeur. Chaque OS consomme du CPU, de la RAM et du stockage qui pourraient autrement être utilisés pour alimenter davantage les applications. Chaque système d'exploitation a besoin de correctifs et de surveillance. Et dans certains cas, chaque système d'exploitation nécessite une licence. Le modèle de machine virtuelle présente également d'autres défis. Les machines virtuelles sont lentes à démarrer et à la portabilité pas terrible - migrer et déplacer des charges de travail de machine virtuelle entre hyper-

viseurs et les plates-formes cloud est plus difficile qu'il ne le faut.

3.3.2 Les conteneurs

Pendant longtemps, les grands acteurs du Web tels que Google utilisaient des conteneurs. C'est une technologie pour remédier à ces faiblesses du modèle de VM. Dans le modèle de conteneur, le conteneur est à peu près analogue à la VM. La différence est que chaque conteneur n'a pas besoin d'un système d'exploitation complet. En fait, tous les conteneurs d'un même hôte partagent un même système d'exploitation. Cela libère énormément de quantités de ressources système telles que le processeur, la RAM et le stockage. Il réduit également les coûts de licence potentiels et réduit les frais généraux liés aux correctifs de système d'exploitation et autres entretiens. Les conteneurs sont également rapides à démarrer et ultra-portables. Déplacement de charges de travail de conteneur depuis votre ordinateur portable vers le cloud, puis vers des ordinateurs virtuels.

3.3.3 1979 : Unix V7

UNIX®

Au cours de l'histoire unix du développement des conteneurs d'Unix V7 en 1979, l'appel système chroot a été introduit, modifiant le répertoire racine d'un processus et de ses fils dans un nouvel emplacement du système de fichiers. Cette avancée a marqué le début de l'isolation des processus : la séparation de l'accès aux fichiers pour chaque processus. Chroot a été ajouté à BSD en 1982. Un programme exécuté dans un tel environnement ne peut pas accéder aux fichiers et aux commandes situés en dehors de cette arborescence de répertoires environnementaux. Cet environnement modifié est appelé une " prison" chroot.

3.3.4 2000 : JBS FreeBSD

Près de deux décennies plus tard, en 2000, un petit fournisseur d'hébergement en environnement partagé est arrivé dans l'historique upBSD des conteneurs avec les FreeBSD jails afin de séparer clairement ses services de ceux de ses clients pour des raisons de sécurité et de facilité d'administration. Les jails FreeBSD permettent aux administrateurs de partitionner un système informatique FreeBSD en plusieurs systèmes indépendants plus petits, appelés jails, avec la possibilité d'attribuer une adresse IP à chaque système et une configuration.

3.3.5 2001 : Linux VServer

Comme les jails FreeBSD, Linux VServer est un mécanisme jail qui permet de partitionner les ressources des conteneurs (systèmes de fichiers, adresses réseau, mémoire) sur un système informatique. Introduite en 2001, cette virtualisation de système d'exploitation est mise en œuvre en appliquant un correctif au noyau Linux. Des correctifs expérimentaux sont toujours disponibles, mais le dernier correctif stable a été publié en 2006.

3.3.6 2004 : conteneurs Solaris

En 2004, la première version bêta publique de Solaris Containers, qui associe des contrôles des ressources système et une séparation des limites fournie par les zones, permettait de tirer parti de fonctionnalités telles que les instantanés et le clonage à partir de ZFS.

3.3.7 2005 : Open VZ (Open Virtuzzo)

Il s'agit d'une technologie de virtualisation au niveau du système d'exploitation pour l'historique de conteneursx de LinuopenVZ, qui utilise un noyau Linux corrigé pour la virtualisation, l'isolation, la gestion des ressources et les points de contrôle. Le code n'a pas été publié dans le noyau officiel de Linux.

3.3.8 2006 : conteneurs de processus

Process Containers (lancé par Google en 2006) a été conçu pour limiter, comptabiliser et isoler l'utilisation des ressources (CPU, mémoire, E / S de disque, réseau) d'un ensemble de processus. Il a été renommé «Control Groups (cgroups)» un an plus tard et a finalement été fusionné avec le noyau Linux 2.6.24.

3.3.9 2008 : LXC

LXC (LinuX Containers) a été la première et la plus complète implémentation du gestionnaire de conteneurs Linux. Il a été implémenté en 2008 à l'aide des "namespace " cgroups de Linux et fonctionne sur un seul noyau Linux sans nécessiter de correctifs.

Le noyau de Linux 2.6.24 intègre une prise en charge fondamentale de la conteneurisation pour assurer une virtualisation au niveau du système

d'exploitation et permettre à un même hôte d'exécuter plusieurs instances Linux isolées, baptisées « conteneurs Linux », ou LXC (LinuX Containers). LXC repose sur la notion de groupes de contrôle Linux, les cgroups. Ici, chaque groupe de contrôle offre aux applications une isolation totale des ressources (notamment processeur, mémoire et accès E/S), et ce sans recourir à des machines virtuelles à part entière. Les conteneurs Linux proposent également une isolation complète de leurs espaces de noms. Les fonctions telles que les systèmes de fichiers, les ID réseau et les ID utilisateur, ainsi que tout autre élément généralement associé aux systèmes d'exploitation, peuvent donc être considérés comme « uniques » du point de vue de chaque conteneur.

3.3.10 2011 : Warden

CloudFoundry a commencé Warden en 2011, en utilisant LXC dans le début et plus tard en le remplaçant par la mise en œuvre des conteneurs. Warden peut isoler des environnements sur n'importe quel système d'exploitation, s'exécutant en tant que démon et fournissant une API pour la gestion des conteneurs. Il a développé un modèle client-serveur pour gérer une collection de conteneurs sur plusieurs hôtes, et Warden inclut un service permettant de gérer les groupes de contrôle, les name space et le cycle de vie des processus.

3.3.11 2013 : LMCTFY

LMCTFY a démarré en 2013 en tant que version à source ouverte de la pile de conteneurs de Google, fournissant des conteneurs d'applications Linux. Les applications peuvent être configurées en «conteneur», en créant et en gérant leurs propres sous-conteneurs. Le déploiement actif dans LMCTFY a été arrêté en 2015 après que Google ait commencé à contribuer à libcontainer, qui fait désormais partie des concepts LMCTFY de base, qui fait maintenant partie de la Open Container Foundation.

3.3.12 2013 : Docker

Lorsque Docker est apparu en 2013, la popularité des conteneurs a explosé. Ce n'est pas une coïncidence : l'histoire croissante des conteneurs de docker a toujours été liée à l'utilisation de docker et des conteneurs.

Docker a explosé sur scène en 2013, et cela a causé de l'excitation dans les cercles informatiques. La technologie de conteneur d'application fournie par Docker promet de changer la façon dont les opérations informatiques sont réalisées de la même manière que la technologie de virtualisation quelques années auparavant.

Autres exemples, Docker propose des outils de génération automatisée de build. Ces outils aident les développeurs à passer plus facilement d'un code source à des applications conteneurisées, ou à travailler avec des outils de "configuration as a code", tels que Chef, Ansible, Puppet et autres, afin d'automatiser ou de rationaliser le processus de build.

La gestion des versions permet aux développeurs de suivre l'évolution des versions des conteneurs, de comprendre les différences, voire de revenir à des versions antérieures le cas échéant. Et sachant que tout conteneur peut servir d'image de base à un autre, il est d'autant plus facile de réutiliser des composants aisément partageables via un registre public (ou privé).

Tout comme Warden, Docker a également utilisé LXC à ses débuts et a par la suite remplacé ce gestionnaire de conteneurs par sa propre bibliothèque, libcontainer (plus tard apellée containerD). Mais il ne fait aucun doute que Docker s'est séparé du pack en offrant un écosystème complet pour la gestion des conteneurs.

3.3.13 2017 : Les outils de conteneur deviennent matures

Des centaines d'outils ont été développés pour faciliter la gestion des conteneurs. Alors que ces types d'outils existent depuis des années, 2017 est l'année où beaucoup d'entre eux ont gagné leurs galons. Il suffit de regarder Kubernetes ; Depuis son adoption dans la Cloud Native Computing Foundation (CNCF) en 2016, VMWare, Azure, AWS et même Docker ont annoncé leur soutien, en plus de leurs infrastructures.

Alors que le marché continue de croître, certains outils ont permis de définir certaines fonctions de la communauté des conteneurs. Ceph et REX-Ray établissent des normes pour le stockage de conteneurs, tandis que Flannel connecte des millions de conteneurs dans des centres de données. Et dans CI / CD, Jenkins change complètement la façon dont nous construisons et déployons des applications. Adoption de rkt et Containerd par la CNCF.

L'écosystème de conteneurs est unique dans la mesure où il repose sur un effort et un engagement de la communauté vis-à-vis de projets open source. Le don de Docker du projet Containerd à la CNCF en 2017 est emblématique de ce concept, ainsi que l'adoption de CNCF du conteneur KRT (prononcé

« fusée ») l'exécution dans le même temps. Cela a conduit à une plus grande collaboration entre les projets, à plus de choix pour les utilisateurs et à une communauté centrée sur l'amélioration de l'écosystème des conteneurs.

3.3.14 Kubernetes grandit

En 2017, le projet open-source a montré de grandes avancées pour devenir un k8s logo.pngtechnology plus mature. Kubernetes prend en charge des classes d'applications de plus en plus complexes, permettant ainsi la transition de l'entreprise vers le cloud hybride et les microservices. A DockerCon à Copenhague, Docker a annoncé qu'ils soutiendront le conteneur Kubernetes orchestrateur et Azure et AWS est tombé en ligne avec AKS (Azure Services Kubernetes) et EKS, un service Kubernetes pour rivaliser avec ECS propriétaire. Il s'agissait également du premier projet adopté par le CNCF et commandait une liste croissante de fournisseurs de services d'intégration de systèmes tiers. Kubernetes semble avoir un avenir prometteur en tant que plate-forme d'orchestration de facto.

3.3.15 Docker : un LXC augmenté

Les plateformes de conteneurisation d'applications, telles que Docker, ne remplacent pas les conteneurs Linux. L'idée consiste à utiliser LXC comme base, puis à ajouter des capacités de niveau supérieur. Par exemple, une plateforme comme Docker autorise la portabilité entre machines (qui exécutent aussi Docker) et permet ainsi à une application et à ses composants d'exister en tant qu'objet mobile unique. LXC seul permet la mobilité, mais la build est liée à la configuration du système. Donc la déplacer sur une autre machine peut introduire des différences susceptibles d'empêcher le conteneur de l'application de s'exécuter à l'identique (voire de s'exécuter tout court).

Mais le recours à LXC était un problème depuis le début. Tout d'abord, LXC est spécifique à Linux. C'était un problème pour un projet qui avait aspirations à être multi-plateforme. Deuxièmement, être dépendant d'un outil externe pour quelque chose d'aussi essentiel au projet était un risque énorme qui pourrait entraver le développement.

En conséquence, Docker. Inc. a développé son propre outil appelé libcontainer en tant que remplacement pour LXC. Le but de libcontainer était d'être une plate-forme agnostique outil permettant à Docker d'avoir accès à la plate-forme de base des conteneurs -blocs qui existent dans le système d'exploitation.

Libcontainer a remplacé LXC en tant que pilote d'exécution par défaut dans Docker 0.9.

Aujourd'hui, Dave Bartoletti, analyste chez Forrester, pense que seulement 10% des entreprises utilisent actuellement des conteneurs en production, mais près d'un tiers sont en phase de test. Docker a généré 762 millions de dollars de revenus en 2016. Les conteneurs ont transformé le monde de l'informatique car ils utilisent des systèmes d'exploitation partagés. Cette technologie permet à un data center ou à un fournisseur Cloud d'économiser des dizaines de millions de dollars par an, mais tout dépend bien sûr des risques qu'il serait prêt à prendre. Cependant, quelques préoccupations existent concernant l'assurance qu'auront les développeurs d'innover librement en utilisant des conteneurs. Les développeurs devraient pouvoir choisir les outils et les frameworks qu'ils souhaitent utiliser sans avoir à demander systématiquement la permission. L'utilisation d'un conteneur pourrait en effet étouffer la créativité d'un développeur... Le choix incombe, alors à l'entreprise d'investir ou non dans les conteneurs. Avec les avantages et les inconvénients qui se contrebalancent, tout peut se résumer au goût du risque.

3.4 Différence avec la virtualisation

Bien sûr, la conteneurisation a de nombreux avantages par rapport à la virtualisation mais on ne peut pas affirmer que cette technologie est 100% parfaite, elle a aussi son lot d'inconvénients. Ces deux schémas ci dessus, d'un environnement de virtualisation et d'un environnement de conteneurisation permettent de bien identifier les différences entre ces deux technologies.

Tout d'abord, les conteneurs partagent un seul et unique système d'exploitation, de ce fait, l'échange de données entre les conteneurs est plus simple et plus rapides que pour les VM. De plus comme chaque conteneur ne contient pas de système d'exploitation propre à lui, les conteneurs sont donc réduits et prennent moins de place et moins de ressource Serveur (environ 10 fois plus petit qu'un VM). Le temps de création et de suppression d'un conteneur est par la même occasion réduit. Les conteneurs facilitent l'évolution technique, par exemple si dans un environnement de VM, l'on souhaite faire évoluer les OS de plusieurs VM, il faut le faire manuellement sur chaque Machines. Ce problème n'est pas présent pour la conteneurisation car toute l'infrastructure repose sur un seul système d'exploitation. Mais la conteneurisation peut avoir quelques inconvénients, comme tous les conteneurs ne reposent que sur un seul système d'exploitation, la diversification des systèmes d'exploitation

n'est pas possible avec la conteneurisation, « ou est plus compliquée à mettre en place ». Les conteneurs sont isolés pour assurer la sécurité et empêcher les malwares de se transmettre entre les conteneurs, mais il est évident que les machines virtuelles seront toujours plus isolées que les conteneurs. Même si les conteneurs ont un grand nombre d'avantages, leur apparition ne sonne pas la fin de la virtualisation. Aujourd'hui, les machines virtuelles sont intégrées dans de nombreuses entreprises comportant des réseaux de grande taille. Pour utiliser entièrement la technologie de conteneurisation, ces entreprises devraient remanier tout leur système informatique ce qui est impensable. Mais de nouvelles entreprises ont vu le potentiel de la conteneurisation et on donc créer leur système informatique en fonction.

4 Les images Docker

La figure présente une vue de haut niveau de la relation entre les images et conteneurs. Nous utilisons les commandes : "docker container run" et "docker service create" pour démarrer un ou plusieurs conteneurs à partir d'une seule image. Cependant, une fois qu'on a démarré un conteneur à partir d'une image, les deux constructions deviennent dépendantes les unes des autres et on ne peut pas supprimer l'image avant que le dernier conteneur l'utilisant ait été arrêté et détruit. Tenter de supprimer une image sans s'arrêter et la destruction de tous les conteneurs l'utilisant entraînera l'erreur suivante :

docker image rm image-name
Error response from daemon : conflict : unable to remove repository reference
"image-name" (must force) - container container-id is using its referenc
ed image image-id

4.1 Les images sont généralement petites

Le but d'un conteneur est d'exécuter une application ou un service. Cela signifie que l'image à partir de laquelle le conteneur est créé doit contenir tous les systèmes d'exploitation et tous les fichiers requis pour exécuter l'application / service. Cependant, les conteneurs sont connus pour être rapide et léger. Cela veut dire que les images à partir desquelles ils sont construits sont généralement petites et dépouillées de toutes les parties non essentielles. Par exemple, les images Docker ne sont pas livrées avec 6 shells différents. Elles sont généralement livrées avec un seul shell minimaliste, ou aucun shell

du tout. Elles ne contiennent pas non plus de kernel - tous les conteneurs s'exécutant sur le Docker hôte partagent un accès à au kernel de l'hôte du kernel. Pour ces raisons, on dit parfois que les images ne contiennent pas suffisamment de système d'exploitation (généralement elles contiennent que des fichiers et des objets du système de fichiers liés au système d'exploitation).

L'image officielle d'Alpine Linux Docker fait environ 4 Mo de taille et représente un bon exemple de la petite taille des images Docker. Cependant, un exemple plus typique pourrait être quelque chose comme l'image officielle Ubuntu Docker qui fait actuellement environ 120 Mo. Ceux-ci sont clairement dépouillés de la plupart des parties non essentielles ! Les images Windows ont tendance à être plus volumineuses que les images Linux, à cause de la façon dont le système d'exploitation Windows fonctionne. Par exemple, la dernière version de .NET image (microsoft / dotnet :latest) est supérieure à 2 Go lorsqu'elle est extrait sans compression. L'image de Windows Server 2016 Nano Server dépasse légèrement 1 Go lorsqu'elle est tirée et non compressé.

Un hôte Docker correctement installé n'a aucune image dans son référentiel local. Le référentiel d'images local sur un hôte Docker basé sur Linux est généralement situé à l'emplacement suivant :

/ var / lib / docker / storage-driver.

On peut vérifier si notre hôte Docker contient des images dans son référentiel local avec la commande suivante :

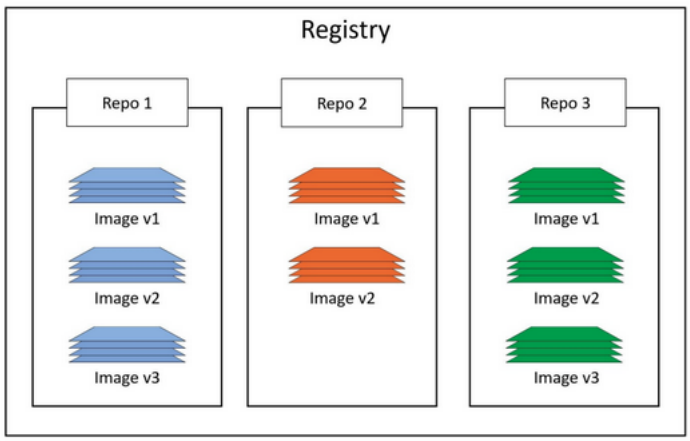
docker image ls

REPOSITORY TAG IMAGE ID CREATED SIZE

4.2 Nom de l'image

Dans le cadre de chaque commande, nous devions spécifier quelle image extraire. Pour ce faire, nous avons besoin d'un peu de contexte sur la manière de stocker les images. Registres d'images : Les images Docker sont stockées en ligne dans des registres d'images. Le registre le plus courant est Docker Hub [1]. Il existe d'autres registres, y compris des registres tiers . Cependant, le client Docker est avisé et utilise par défaut Docker Hub.

Les registres d'images contiennent plusieurs référentiels d'images. À leur tour, les référentiels d'images peuvent contenir plusieurs images. Cela pourrait être un peu déroutant, alors la figure montre une image d'un registre d'images contenant 3 référentiels, et chaque référentiel contient une ou plusieurs images.



4.2.1 Dépôts officiels et non officiels

Docker Hub a également le concept de référentiels officiels et non officiels des dépôts.

Comme le nom l'indique, les référentiels officiels contiennent des images qui ont été vérifiées par Docker, Inc. Cela signifie qu'ils doivent contenir un code à jour et de haute qualité, sécurisé, bien documenté et conforme aux meilleures pratiques. Les référentiels non officiels sont tout le contraire.

4.2.2 Nom et marquage des images

Le format pour docker : image pull quand on travaille avec une image d'un dépôt officiel, c'est :

docker image pull repository : tag

Dans les exemples Linux précédents, nous avons téléchargé des images Alpine et Ubuntu. avec les deux commandes suivantes :

docker image pull alpine :latest and docker image pull ubuntu :latest

Les commandes suivantes montrent comment téléchargé différentes images d'un référentiel officiel :

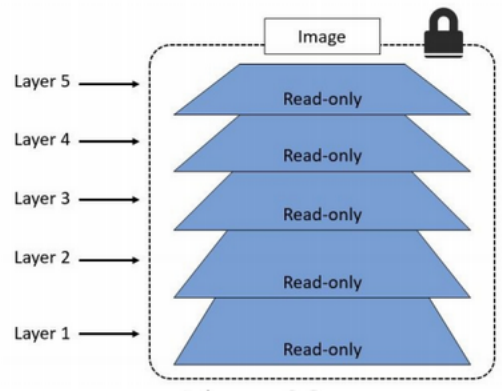
docker image pull mongo :3.3.11
//This will pull the image tagged as '3.3.11'
//from the official 'mongo' repository.
docker image pull redis :latest
//This will pull the image tagged as 'latest'
//from the official 'redis' repository.
docker image pull alpine
//This will pull the image tagged as 'latest'
//from the official 'alpine' repository.

Quelques points à noter sur les commandes ci-dessus : Tout d'abord, si on ne spécifie pas de balise d'image après le nom du référentiel, Docker suppose que nous nous référons à la dernière image.

Deuxièmement, la dernière balise n'a aucun pouvoir magique ! Juste parce qu'une image est étiquetée comme dernière ne garantit pas que c'est l'image est la plus récente dans un dépôt !

4.2.3 Images et couches

Une image Docker est juste un groupe de couches en lecture seule faiblement connectés. Docker s'occupe d'empiler ces couches et de les représenter comme un seul objet unifié. Il y a plusieurs façons de voir et d'inspecter les couches qui composent une image :



Toutes les images Docker commencent par une couche de base, et au fur et à mesure que des modifications sont apportées, le contenu est ajouté, les nouvelles couches sont ajoutées en haut.

4.2.4 Les hachages d'image (digests)

L'image n'est en réalité qu'un objet de configuration répertoriant les couches et ainsi que certaines métadonnées (telle que son nom par exemple). Les couches constituant une image sont totalement indépendantes et n'ont aucune notion de faire partie d'une image collective.

Chaque image est identifiée par un identifiant de cryptage qui est un hachage de l'objet config. Chaque couche est identifiée par un identifiant de cryptage qui est un hachage du contenu qu'elle contient.

Cela signifie que changer le contenu de l'image, ou de l'une de ses couches, provoquera le changement des cryptages associés. En conséquence, les images et les couches sont immuables.

Lorsqu'on push et pull les images, nous compressons leurs couches pour économiser la bande passante. Mais la compression d'une couche change son contenu ! Cela signifie que le contenu du hachage ne correspond plus après l'opération de push et pull ! Ceci est évidemment un problème. Par exemple, lorsqu'on place une couche d'image sur Docker Hub, Docker Hub tente de vérifier que l'image est arrivée sans être altérée en route. Pour ce faire, il exécute un hachage sur la couche et vérifie s'il correspond au hash qui a été envoyé avec la couche. Parce que la couche a été compressée (modifiée) la vérification du hachage échouera.

Pour résoudre ce problème, chaque couche reçoit également un élément appelé hachage de distribution. C'est un hachage de la version compressée de la couche. Quand une couche est push et pull du registre, son hash de distribution est inclus, et c'est ce qui est utilisé pour vérifier que la couche est arrivée sans être altérée. Ce modèle de stockage à contenu adressable améliore considérablement la sécurité en nous offrant un moyen d'imprimer et de superposer des données après des opérations de push et pull. Cela évite aussi les collisions d'identifiants pouvant survenir si les identifiants d'image et de couche étaient générés de manière aléatoire.

4.2.5 Images multi-architecturales

Docker prend désormais en charge les images multi-plateformes et multi-architectures. Cela signifie qu'un référentiel d'images et une balise doivent avoir une image pour Linux sur x64 et Linux sur PowerPC, etc. Pour ce faire, l'API de registre prend en charge un fat manifest ainsi qu'une image manifest. Les fat manifests listent les architectures supportées par une image particulière, tandis que l'image manifest, liste les couches qui composent une image particulière.

Supposons qu'on utilise Docker sous Linux x64. Quand on pull une image depuis Docker hub, notre client Docker adresse les demandes d'API pertinentes à Docker API tournant sur Docker Hub. Si un fat manifest existe pour cette image, il sera analysé pour voir s'il existe une entrée pour Linux sur x64. Si elle existe, l'image manifest pour cette image est récupérée et analysée pour les couches réelles qui composent l'image. Les couches sont identifiées par leurs identifiants de cryptage et sont extraites du Registre "blob".

4.2.6 Supprimer des images

Lorsqu'on n'a plus besoin d'une image, on peut la supprimer de notre hôte Docker avec la commande :

docker image rm. rm

Supprimer les images "pull" dans les étapes précédentes avec la commande :

docker image rm

Si l'image qu'on essaye de supprimer est utilisée par un conteneur en cours d'exécution, on ne pourra pas la supprimer. Il faudra Arrêter et supprimer tous les conteneurs avant d'essayer la suppression à nouveau. Un raccourci pratique pour nettoyer un système et supprimer toutes les images sur un Docker hôte est d'exécuter la commande docker image rm et lui transmettre une liste de tous les ID d'image sur le système en appelant l'image de menu fixe ls avec l'indicateur -q.

docker image rm (docker image ls -q) -f

5 Travail réalisé

5.1 Explication du code et sa structure

Le plus difficile pour moi fut de trouver la bonne structure et l'architecture qui répondait correctement au cahier des charges. L'erreur que j'ai commise est de ne pas avoir fait un schéma papier et ne pas avoir pris le temps de structurer et généraliser mon script pour qu'il s'adapte à tout utilisateur. Le plus important est d'avoir compris l'utilité et l'importance de cette technique dans un projet et d'en tirer une leçon.

Il y a une vingtaine de scripts, j'ai choisi de les diviser comme ce qui suit :

— help : indique toutes les fonctionnalités, les commandes et flags associés : baleine help

— create image : La commande pour simplement créer une image est : image create -i “nom image” -s “taille image” -r “chemin image” -p “proxy” car on pourrait vouloir créer une image mais ne pas vouloir lancer un conteneur de dessus, d'où la séparation de la création du conteneur et de l'image.. Il faut noter que dans ce script, il crée l'image dans le PATH BALEINE : /var/lib/baleine/images.
Cependant, la copie de l'image sur laquelle le conteneur est lancé se fait dans le PATH : /var/lib/baleine/containers/nom container Il crée un manifeste contenant le nom de l'image, sa taille, son chemin. Ainsi, on établit un ordre logique dans la structure. Et l'utilisation sera généralisée à tout utilisateur qui se sert de l'application. Ce script crée aussi un manifeste (voir la définition en annexe) de l'image, contenant : son nom, sa taille et son chemin.

— create bridge : On crée un bridge grâce à la commande : bridge create -b “nom bridge” -a “ADDR IPV4”

— create container : ce script permet de lancer une instance et crée des interfaces réseau virtuelles relié au commutateur/bridge cité plus tôt. Et ce avec la commande : container create -i “nom image” -c “nom container” -b “nom bridge” -a “ADDR_IPV4” -r “repertoire” -p ”programme”
Mais avant ceci, il copie l'image dans : /var/lib/baleine/containers/-nom container
Puis il la monte dans : /mnt/baleine/nom container
De même, il crée un manifeste contenant le nom du conteneur, l'image depuis laquelle il a été monté, le nom de son bridge, son PID ainsi que son : starting time” (pour savoir depuis quand le conteneur tourne).

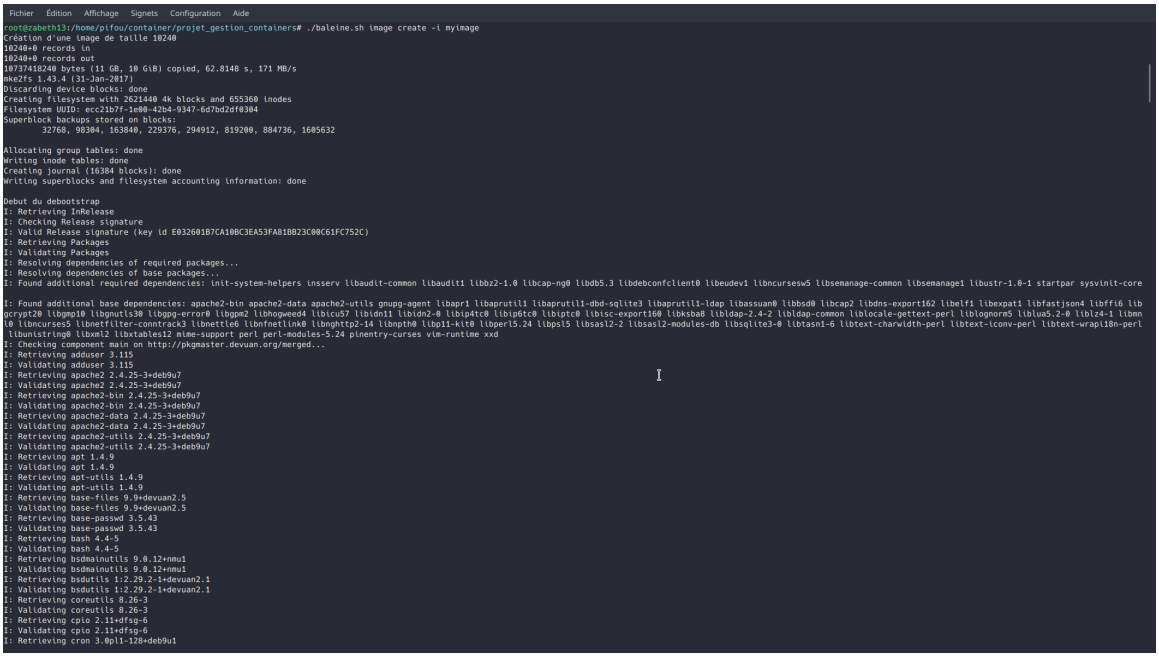
— remove image : Ce script permet de détruire complètement une image et supprimer son manifest avec la commande : image remove “NAME

IMAGE TO REMOVE"
- remove container : ce script éradique l'instance et détruit son image disque et de ses caractéristiques, avec : container remove "NAME CONTAINER TO REMOVE"
- remove bridge : Ici, on supprime simplement un bridge spécifique (en entrant le nom de ce dernier en paramètre). bridge remove "NAME BRIDGE TO REMOVE"
- stop container : arrête une instance, et détruit naturellement ses ressources réseau. Mais son image disque et ses caractéristiques restent conservées dans un état stable pour qu'il puisse faire se lancer à nouveau depuis. container stop -c "NAME CONTAINER TO STOP"
- list images : liste toutes les images qui ont été créées (nom, taille , chemin) , ceci se fait à partir du manifeste : image list
- list containers : de même, ici on liste tous les conteneurs qui tournent (nom du conteneur , le nom de l'image qui l'a lancé, PID , temps d'exécution) : container list
- list bridges : idem, on liste tous les bridges bridge list
- up/down bridge : ici nous pouvons mettre en up ou down un bridge
  bridge -b "NAME BRIDGE" up
  bridge -b "NAME BRIDGE" down

5.2 Fonctionnement de l'application

5.2.1 Images

Création d'image :



Liste d'images :

```
root@zabeth13:/home/pifou/container/projet_gestion_containers# ./baleine.sh image list
----------------------
Nom image: myimage
Taille: 10240
Chemin: /var/lib/baleine/images
----------------------
```

Suppression d'image :

```
root@zabeth13:/home/pifou/container/projet_gestion_containers# bash baleine.sh image remove -i apache2_reverse
root@zabeth13:/home/pifou/container/projet_gestion_containers# bash baleine.sh image list
```

5.2.2 Bridges

Création de bridge :

```
root@zabeth13:/home/pifou/container/projet_gestion_containers# ./baleine.sh bridge create -b mybridge -a 192.168.42.1/24
Création du bridge mybridge
Attribution de l'adresse ip 192.168.42.1/24 à mybridge
Demarrage de mybridge
root@zabeth13:/home/pifou/container/projet_gestion_containers# ./baleine.sh bridge list
----------------------
bridge name     bridge id               STP enabled     interfaces
mybridge                8000.000000000000       no
----------------------
```

Liste des bridges :

```
root@zabeth13:/home/pifou/container/projet_gestion_containers# ./baleine.sh bridge create -b mybridge -a 192.168.42.1/24
Création du bridge mybridge
Attribution de l'adresse ip 192.168.42.1/24 à mybridge
Demarrage de mybridge
root@zabeth13:/home/pifou/container/projet_gestion_containers# ./baleine.sh bridge list
----------------------
bridge name     bridge id               STP enabled     interfaces
mybridge                8000.000000000000       no
----------------------
```

Suppression des bridges :

```
root@zabeth13:/home/pifou/container/projet_gestion_containers# ./baleine.sh bridge remove -b mybridge
root@zabeth13:/home/pifou/container/projet_gestion_containers# ./baleine.sh bridge list
```

5.2.3 Conteneurs

Création de conteneur :

```
root@zabeth13:/home/pifou/container/projet_gestion_containers# ./baleine.sh container create -l myimage -c mycontainer -b mybridge -a 192.168.42.2/24 -p "/usr/sbin/apache2ctl start"
Creation du container mycontainer basé sur l'image myimage lancant le programme /usr/sbin/apache2ctl start connecté au(x) bridge(s) mybridge avec les adresses 192.168.42.2/24
Montage de l'image
PID Unshare :27563
nohup: appending output to 'nohup.out'
Configuration réseau au sein du conteneur
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
19: eth0@if20: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether aa:be:b1:e7:65:95 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.42.2/24 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::a8be:b1ff:fee7:6595/64 scope link tentative
       valid_lft forever preferred_lft forever
```

Liste des conteneurs :

```
root@zabeth13:/home/pifou/container/projet_gestion_containers# bash baleine.sh container list
----------------------
Nom container: mycontainer
Nom image: apache2_reverse
PID: 3088
Starting time:  2019-05-09-21h-50m-05s
Interfaces: vifc866_0
----------------------
```

Suppression de conteneurs :

```
root@zabeth13:/home/pifou/container/projet_gestion_containers# ./baleine.sh container remove -c mycontainer
Kill du container
Démontage de /mnt/baleine/mycontainer
Suppression de /mnt/baleine/mycontainer
Suppression de /var/lib/baleine/containers/mycontainer
Suppression des interfaces réseau :
Suppression de vifc866_0
Suppression du manifeste du container
```

6 Mandataire inverse

Pour réaliser l'architecture avec un mandataire inverse apache2, on va mettre en place deux bridges, un sera considéré comme public, et l'autre comme étant privé. Le mandataire inverse sera présent sur le bridge considéré comme public, tandis que l'autre serveur apache2 (qui fera office de serveur HTTP) sera présent sur le bridge privé.

Le mandataire inverse sera également présent sur le bridge privé, ainsi il aura deux adresses ip. Une publique, et une autre privée. Lorsque l'on accèdera au mandataire inverse via son adresse ip publique, il va transmettre la requête au serveur HTTP non accessible.

Réalisons cela grâce à nos conteneurs, tout d'abord nous allons créer deux bridges :

```
root@zabeth13:/home/pifou/container/projet_gestion_containers# bash baleine.sh bridge create -b private -a 192.168.42.1/24
Création du bridge private
Attribution de l'adresse ip 192.168.42.1/24 à private
Demarrage de private
root@zabeth13:/home/pifou/container/projet_gestion_containers# bash baleine.sh bridge create -b public -a 10.0.0.1/8
Création du bridge public
Attribution de l'adresse ip 10.0.0.1/8 à public
Demarrage de public
```

On vérifie que les deux bridges sont correctement faits :

```
21: private: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default qlen 1000
    link/ether 4e:3d:56:aa:2c:c7 brd ff:ff:ff:ff:ff:ff
    inet 192.168.42.1/24 scope global private
       valid_lft forever preferred_lft forever
    inet6 fe80::4c3d:56ff:feaa:2cc7/64 scope link
       valid_lft forever preferred_lft forever
22: public: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default qlen 1000
    link/ether 6e:fd:b1:da:70:80 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.1/8 scope global public
       valid_lft forever preferred_lft forever
    inet6 fe80::6cfd:b1ff:feda:7080/64 scope link
       valid_lft forever preferred_lft forever
```

Ensuite, nous allons créer deux images, une sera apache2 reverse et l'autre apache2. Puis nous allons monter l'image apache2 reverse dans le but de modifier sa configuration.

```
root@zabeth13:/home/pifou/container/projet_gestion_containers# bash baleine.sh image list
----------------------
Nom image: apache2
Taille: 10240
Chemin: /var/lib/baleine/images
----------------------
----------------------
Nom image: apache2_reverse
Taille: 10240
Chemin: /var/lib/baleine/images
----------------------
```

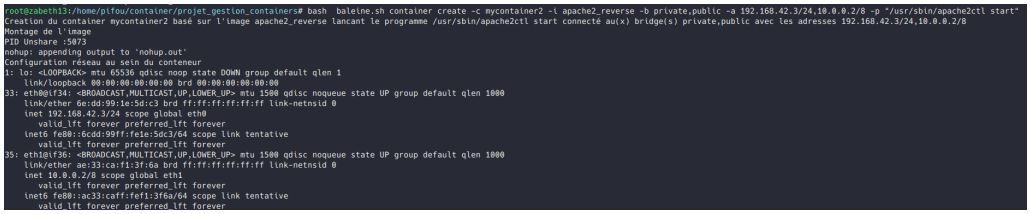
```
root@zabeth13:/home/pifou/container/projet_gestion_containers# mount -t ext4 -o loop /var/lib/baleine/images/apache2_reverse /mnt/apache2_reverse
root@zabeth13:/home/pifou/container/projet_gestion_containers# vim /mnt/apache2_reverse/etc/apache2/sites-enabled/000-default.conf
```

On modifie le fichier /etc/apache2/sites-enabled/000-default afin d'y rajouter les lignes suivantes :
ProxyPass / 192.168.42.2
ProxyPassReverse / 192.168.42.2

De cette manière, toutes les requêtes réalisées à 10.0.0.2 (l'ip du mandataire inverse) sera transmis au conteneur possédant le serveur HTTP à l'ip 192.168.42.2.

Maintenant que la configuration est terminée, on peut demarrer nos conteneurs :

```
root@zabeth13:/home/pifou/container/projet_gestion_containers# bash  baleine.sh container create -c mycontainer -i apache2_reverse -b private -a 192.168.42.2/24 -p "/usr/sbin/apache2ctl start"
Creation du container mycontainer basé sur l'image apache2_reverse lancant le programme /usr/sbin/apache2ctl start connecté au(x) bridge(s) private avec les adresses 192.168.42.2/24
Montage de l'image
PID Unshare :4489
nohup: appending output to 'nohup.out'
Configuration réseau au sein du conteneur
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
31: eth0@if32: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether e2:9d:15:ac:b8:33 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.42.2/24 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::e09d:15ff:feac:b833/64 scope link tentative
       valid_lft forever preferred_lft forever
```



6.1 Options ajoutées au projet

— Pour faciliter l'utilisation, un help équivalent à un manuel a été mis en place, que l'on peut visualiser en exécutant la commande suivante : baleine.sh help

— Une option a été mise en place concernant les interfaces réseau lors de la création du conteneur. En effet, si l'on souhaite créer plusieurs interfaces sur un même conteneur avec plusieurs bridges, on arrive à gérer ceci, grâce à la création d'une arraylist en shell (découverte très intéressante et surprenante).

— J'ai rajouté une option qui permet à l'utilisateur de rentrer ses arguments dans n'importe quel ordre du moment où il met les drapeaux (Flags) correspondants devant. (L'utilitaire Help) l'aidera à les connaître.

— De même, il m'a été imposé de ne pas faire une gestion sophistiquée des images à base de système de fichiers à couche. Pour ce, j'utilise le système de fichier ext4, qui ne gère pas les couches

6.2 Améliorations possibles du projet

— J'avais essayé pendant un moment de gérer des versions d'images en leur accordant un nombre aléatoire pour les différencier, mais ça m'a paru compliqué à gérer par la suite. Cependant, ça pourrait être une idée d'amélioration du script et de gestion de bordure et généralisation.

— Gérer une architecture différente de Debian (une architecture Redhat par exemple).

7 Annexes

7.1 Ext4

Est une évolution du système de fichier ext3, qui est actuellement le système de fichier le plus utilisé sous Linux. Il présente de nombreux avantages et optimisations par rapport à l'ancienne version, tout en assurant une rétro-compatibilité. Ext4 est stable et est le système de fichier par défaut sous 9.10. Outre le fait qu'il puisse gérer les volumes d'une taille allant jusqu'à un exbioctet (260 octets), la fonctionnalité majeure de ext4 est l'allocation par extent qui permettent la pré-allocation d'une zone contiguë pour un fichier, pour minimiser la fragmentation. L'option extent est activée par défaut depuis le noyau Linux 2.6.23 ; avant cela, elle devait être explicitement indiquée lors du montage de la partition.

7.2 Debootstrap

Est un outil permettant d'installer un système de base Debian dans un sous-répertoire d'un autre système déjà installé. Il ne nécessite pas de CD d'installation, mais un accès à un référentiel Debian. Il peut également être installé et exécuté à partir d'un autre système d'exploitation. Ainsi, on peut par exemple utiliser debootstrap pour installer Debian sur une partition non utilisée d'un système Gentoo en cours d'exécution. Il peut également être utilisé pour créer un rootfs pour une machine d'une architecture différente, appelée "cross-debootstrapping". Il existe également une version largement équivalente écrite en C : cdebootstrap, qui est plus petite. Debootstrap ne peut utiliser qu'un seul référentiel pour ses packages. Si l'on doit fusionner des paquets provenant de différents référentiels (comme le fait apt) pour créer un rootfs, ou si l'on doit personnaliser automatiquement le rootfs, on utilise Multistrap.

7.3 Cgroups : (control groups)

Est une fonctionnalité du noyau Linux pour limiter, compter et isoler l'utilisation des ressources (processeur, mémoire, utilisation disque, etc).
Un groupe de contrôle est une suite de processus qui sont liés par le même critère. Ces groupes peuvent être organisés hiérarchiquement, de façon que chaque groupe hérite des limites de son groupe parent. Le noyau fournit l'accès à plusieurs contrôleurs (sous-systèmes) à travers l'interface cgroup2. Par exemple, le contrôleur « memory » limite l'utilisation de la mémoire, « cpuacct » comptabilise l'utilisation du processeur, etc.

Les Control groups peuvent être utilisés de plusieurs façons :
— En accédant au système de fichier virtuel cgroup manuellement
— En créant et gérant des groupes à la volée en utilisant des outils tels que cgcreate, cgexec, cgclassify (de libcgroup)
— Le « démon de moteur de règles » qui peut automatiquement déplacer les processus de certains utilisateurs, groupes ou commandes vers un cgroup en respectant ce qui est spécifié dans la configuration.

— Indirectement à travers d'autres logiciels qui utilisent cgroups, tels que la virtualisation LXC8, libvirt, systemd, Open Grid Scheduler/Grid Engine9.

7.4 Unshare

Unshare() permet à un processus de dissocier les parties de son contexte d'exécution qui sont actuellement partagées avec d'autres processus. Une partie du contexte d'exécution, comme l'espace de noms, est implicitement partagée lorsqu'un processus est créé avec fork(2) ou vfork(2), pendant que d'autres parties, comme la mémoire virtuelle, peuvent être partagées par une demande explicite lors de la création d'un processus avec clone(2).
Le principal intérêt de unshare() est de permettre à un processus de contrôler son contexte d'exécution partagé sans avoir à créer un nouveau processus.

7.5 chroot (change root)

Est un appel système qui a également donné son nom à une commande des systèmes d'exploitation Unix permettant de changer le répertoire racine d'un processus de la machine hôte.
En d'autres termes, la commande "chroot" permet de changer le répertoire racine vers un nouvel emplacement. Chroot peut être utilisé dans deux cas :

— En tant que bascule d'environnement pour prendre le contrôle d'une installation Linux depuis un autre système.
— En tant que prison pour empêcher un utilisateur de remonter dans l'arborescence pour l'emprisonner dans un répertoire spécifique (ce qui peut être utilisé avec un serveur FTP pour que les utilisateurs ne remontent pas dans l'arborescence du système).

7.6 Les Veth

Virtual Ethernet permet aux partitions logiques de communiquer entre elles sans avoir à affecter de matériel physique aux partitions logiques.

On peut créer des adaptateurs Ethernet virtuels sur chaque partition logique et les connecter à des réseaux locaux virtuels. Les communications TCP / IP sur ces réseaux locaux virtuels sont acheminées via le microprogramme du serveur.

Un adaptateur Ethernet virtuel fournit une fonction similaire à celle d'un adaptateur Ethernet 1 Gb. Une partition logique peut utiliser des adaptateurs Ethernet virtuels pour établir plusieurs connexions interpartition haut débit au sein d'un même système géré. Les partitions logiques AIX, IBM® i, Linux et Virtual I / O Server et les environnements Windows intégrés à la plate-forme System i peuvent communiquer entre eux via TCP / IP via les ports de communication Ethernet virtuels.

Les cartes Ethernet virtuelles sont connectées à un commutateur Ethernet virtuel de type IEEE 802.1q (VLAN). Grâce à cette fonction de commutateur, les partitions logiques peuvent communiquer entre elles à l'aide d'adaptateurs Ethernet virtuels et en affectant des ID de VLAN leur permettant de partager un réseau logique commun. Les cartes Ethernet virtuelles sont créées et les attributions d'ID de VLAN sont effectuées à l'aide de la console HMC. Le système transmet les paquets en les copiant directement de la mémoire de la partition logique émettrice vers les mémoires tampons de réception de la partition logique réceptrice sans aucune mise en mémoire tampon intermédiaire du paquet.

Le nombre d'adaptateurs Ethernet virtuels autorisés pour chaque partition logique varie en fonction du système d'exploitation.

AIX 5.3 et versions ultérieures prennent en charge jusqu'à 256 cartes Ethernet virtuelles pour chaque partition logique.
La version 2.6 du noyau Linux prend en charge jusqu'à 32 768 cartes Ethernet virtuelles pour chaque partition logique. Chaque partition logique Linux peut appartenir à un maximum de 4 094 réseaux locaux virtuels.

7.7 Manifeste

Un manifeste contient des informations sur une image, telles que les couches, la taille et le digest (hash). La commande "docker manifest" fournit également aux utilisateurs des informations supplémentaires telles que le système d'exploitation et l'architecture pour lesquelles une image a été créée.

Une liste de manifestes est une liste de couches d'image créée en spécifiant un ou plusieurs noms d'image (idéalement plusieurs). Il peut ensuite être utilisé de la même manière qu'un nom d'image dans les commandes docker pull et docker run, par exemple.

Idéalement, une liste de manifestes est créée à partir d'images de fonctions identiques pour différentes combinaisons os / arch. Pour cette raison, les listes de manifestes sont souvent appelées «images multi-arch». Cependant, un utilisateur peut créer une liste de manifestes pointant vers deux images : une pour Windows sur amd64 et une pour Darwin sur amd64.

7.8 API REST

Une API compatible REST, ou « RESTful », est une interface de programmation qui fait appel à des requêtes HTTP pour obtenir (GET), mettre à jour PUT), publier (POST) et supprimer (DELETE) des données ou bien simplement interagir avec des services.

Aujourd'hui, l'utilisation croissante du Cloud et des architectures microservices Web font que l'API REST est devenu un standard de facto. Pour donner un exemple, la plupart des grandes entreprises du Web exposent une API REST afin de pouvoir avoir accès à leurs données et/ou interagir avec leurs services.

Dans le cas spécifique de Docker. Le moteur expose une API REST, cette dernière est utilisée par l'interface en ligne de commande. Ce qui est intéressant, c'est qu'il suffit maintenant de bind le moteur Docker sur une adresse IP et un port, et on peut interagir avec le moteur à distance. En pratique, c'est que qui est fait quand on orchestre Docker avec Kubernetes par exemple.

La plupart des commandes du clien ont un équivalent direct en point de terminaison d'API (par exemple, la commande docker ps va faire un GET /containers au moteur Docker).

7.9 Les SWAP

L'espace d'échange, aussi appelé par son terme anglais swap space ou simplement swap, est une zone d'un disque dur faisant partie de la mémoire virtuelle de l'ordinateur. Il est utilisé pour décharger la mémoire vive physique (RAM) de l'ordinateur lorsque celle-ci arrive à saturation.

L'espace d'échange, se trouve généralement sous une forme de partition de disque dur – on parle alors de partition d'échange. Il peut aussi se présenter sous forme de fichier – on parle alors de fichier d'échange.

Par défaut, la plupart des distributions calculent et s'attribue automatiquement un espace d'échange suffisant lors de leurs installation (sa taille est régulièrement égale au double de la taille de la RAM).

7.10 Les Namespace

Le terme espace de noms (namespace) désigne en informatique un lieu abstrait conçu pour accueillir des ensembles de termes appartenant à un même répertoire, comme dans l'exemple suivant où les espaces de noms sont nommés « Jean-Paul » et « Jean-Pierre » :
Jean-Paul
Jean-Pierre
Mes livres
Mes BD
Mes CD
Mes CD

7.10.1 Propriétés

Un espace de noms peut être vu comme une fonction F qui, à un ensemble de symboles S, associe un ensemble O d'objets (à prendre au sens large). Ces objets peuvent être des entiers, des réels, des objets informatiques, des lieux, des personnes, etc.
En programmation informatique, les espaces de noms sont généralement des injections seulement car s'il y a deux objets nommés distincts alors leur nom est différent et un objet nommé peut avoir plusieurs noms.
Dans le cas d'un réseau Ethernet, les espaces de noms sont des bijections car chaque carte réseau a une adresse Ethernet unique et, à partir d'une telle carte, il est possible de retrouver l'adresse Ethernet correspondante. Enfin, s'il

y a un ensemble d'écoliers dans une classe et considérons leur prénom comme ensemble de symboles, l'espace de noms est surjectif seulement. En effet, tous les écoliers ont un prénom, mais plusieurs peuvent avoir le même prénom. Ce dernier cas, arrive parfois en programmation dans plusieurs situations (ex. : table de hachage), ce qui produit des collisions. Pour distinguer les objets nommés, on peut étendre les noms par des préfixes (dans le cas de la programmation) ou ajouter le nom de la personne ou son adresse (dans le cas des écoliers). Dans ce cas, on parle d'extension de l'espace de noms. On peut aussi remplacer les noms ambigus par des pseudonymes ou des alias. Dans ce cas, on parle de modifier l'espace de noms.

7.11 Système de fichiers

Le système de fichiers (par exemple : zfs) permet l'existence de volumes virtuels (datasets sous zfs), qui partagent un même espace physique, mais peuvent être montés à des endroits différents d'une même hiérarchie de fichiers, ce qui permet la dé-duplication entre des volumes différents et a l'avantage de ne pas imposer à chaque volume virtuel de réserver un espace fixe difficile à modifier après coup.

Conclusion

Ce projet purement informatique en autonomie était une première pour moi. En effet, je ne pensais pas au début de l'intégration de la filière Informatique micro-électronique et automatique que j'allais autant apprécier faire un projet informatique aussi pointu.

Les cours que j'ai reçu au premier semestre en système réseau m'ont passionné et donné envie d'en apprendre plus. Ce qui a naturellement orienté mon choix vers ce projet que je ne regrette point.
J'ai acquis énormément de culture générale sur ce monde, eu l'occasion d'anticiper des cours de 5ème année, et orienter mes sujets de stages.

En outre, j'ai énormément progressé, sur l'écriture des scripts shell, sur la recherche de solutions liées au code, la bonne utilisation de google (La rapidité et la sélection du bon site, savoir poser ses questions sur stack overflow est une qualité à acquérir absolument pour un développeur). Aussi, donner une structure à son projet, subdiviser les tâches et le gérer le temps était un excellent exercice.

Concernant le travail réalisé, je me suis impliquée autant que je pouvais et essayé de remettre un code "propre" et fonctionnel répondant à une grande majorité du cahier des charges. J'espérais faire mieux, mais faute de temps et rythme soutenu du semestre j'ai dû sacrifier plus de temps sur d'autres modules.

En prenant du recul sur le projet, j'apprécie finalement les différentes difficultés que j'ai rencontrées et réussi à surmonter, il n'y a pas plus formateur que les erreurs que l'on commet, et je pense que c'est tout l'objectif des projets de 4ème année.

Pour finir, je remercie encore une fois mes encadrants de m'avoir donné l'opportunité de découvrir Docker dans un contexte de projet, de m'avoir soutenu et encouragé à aller plus loin et réussir.

Références

En plus de Stack overflow, stack exchange ainsi que le livre "Docker deep dive" de Nigel Poulton, voici quelques ressources qui m'ont été très utiles :

https://kubernetes.io/docs/concepts/workloads/pods/pod/#what-is-a-pod
https://coreos.com/rkt/docs/1.29.0/app-container.html#pods
https://phelepjeremy.wordpress.com/2017/06/21/la-conteneurisation/
https://bobcares.com/blog/docker-vs-rkt-rocket/
https://www.redhat.com/fr/topics/containers/whats-a-linux-container
https://www.lebigdata.fr/docker-definition
https://www.supinfo.com/articles/single/5473-conteneurs-pourquoi-t-on-besoin
https://www.lemondeinformatique.fr/actualites/lire-containers-ou-vm-comment-fair
html
https://linuxcontainers.org/fr/?fbclid=IwAR0-9f_H4jZ9Tj2dpZV94YACulzj5QHmZ0nOXXo
https://linuxcontainers.org/fr/lxd/getting-started-cli/?fbclid=
IwAR2EGS__hV962B7Bcw7TSJ54nN22Isj1UvORuTZ_0vexACAjfrJve0v6AZA
https://linuxcontainers.org/fr/lxc/introduction/
https://fralef.me/docker-and-iptables.html https://docs.docker.
com/storage/storagedriver/#images-and-layers
https://docs.docker.com/network/iptables/