Docker

Présentation de l'environnement

Ce document est une introduction au système de conteneurisation Docker.
Il s'appuie sur les versions API 1.13 et ultérieures.

Après une présentation du concept, nous découvrons les commandes de base permettant de gérer des applications isolées dans leurs contextes.

Nous finirons les présentations avec une introduction à Kubernetes.

Pour des demandes de formations, contactez-moi :
https://pierreau.fr/Contact/index.php

Bonne lecture…

Pierre ROYER Manager | Architecte | Formateur #numérique

INDEX

PREAMBULE

I. Ce document

Informations

Nom du document	Docker-CentOS-8.docx	Référence	DEVOPS-DOCKER
Version	2020.02.02	Pages	71
Date de création	07/03/2017	Dernière modification :	19/07/2020
Auteur :	Pierre ROYER Tél : (+33) 614 672 909 https://www.linkedin.com/in/pierreau	Contributeur(s) :	
Mode de diffusion	☐ confidentiel ☐ restreint ☒ interne ☐ libre	Liste de diffusion	https://pierreau.fr
Annexes :			

II. Conventions

Les syntaxes utilisées dans ce document :

[root@CentOS ~]# représente un prompt bash en root sur un serveur CentOS
root@Debian:~ # représente un prompt bash en root sur un serveur Debian
[pierreau@CentOS ~]$ désigne un compte utilisateur local

Le contenu d'un fichier est encadré, les commandes sont en gras :

```
[CentOS@localhost ~]# vi /etc/ssh/sshd_config
```

```
PermitRootLogin yes
```

Les caractères en italique sont des exemples de paramètres :

```
192.168.100.100 ServeurA
192.168.100.101 ServeurB
```

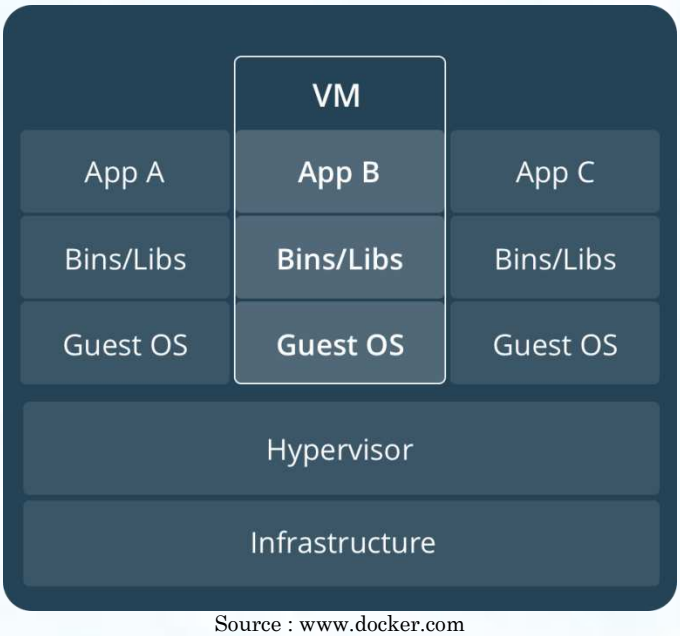
Information utile	Attention particulière	Risque important
		

Introduction

I. Virtualisation systèmes

La virtualisation de serveurs est un concept qui existe depuis 1972 : IBM le pratiquait depuis 1972 ; avec des mainframes VM/370.

Les machines virtuelles partagent un même serveur physique, alors que les conteneurs partagent le même système d'exploitation.



Source : www.docker.com

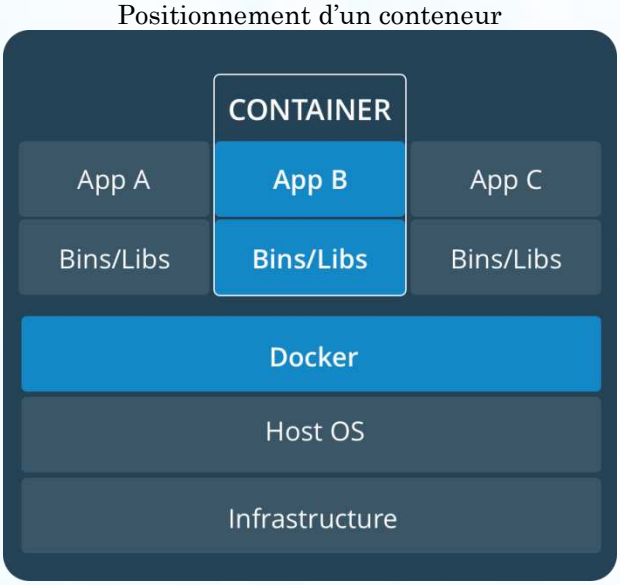
II. Virtualisation applicative

Contrairement aux machines virtuelles, les conteneurs Docker n'incluent pas de système d'exploitation, mais permettent d'isoler des applications dans des contextes.

Ce principe d'isolation existe depuis plusieurs années, et a été implémenté entre autres sur :

- Unix FreeBSD avec « BSD Jail » dès 1999
- Sun Solaris 10 (Oracle) en 2005. Les conteneurs Solaris sont appelés « zones »
- Linux en 2008 : LXC (LinuX Containers)

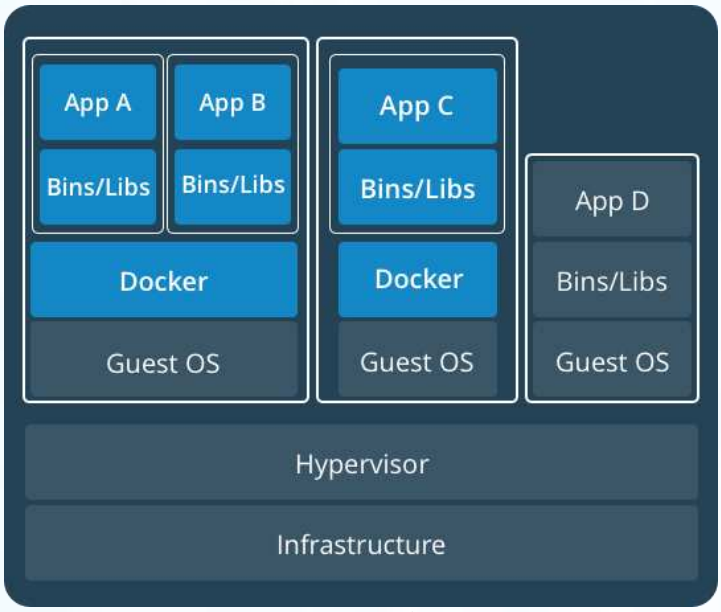
Positionnement d'un conteneur



Source : www.docker.com

III. Virtualisation virtuelle

La virtualisation système et applicative sont deux concepts différents, mais qui peuvent être complémentaires. Il est ainsi possible de contenir des conteneurs dans un système virtualisé :



Source : www.docker.com

Si l'on pousse le raisonnement, il est tout à fait envisageable d'exécuter Docker… dans un autre conteneur Docker, lancé au sein d'une machine virtuelle qui est hébergée sur un serveur physique…

IV. Présentation de Docker

Docker est un logiciel sous licence Apache 2.0, qui a été distribué en tant que projet open source à partir de mars 2013.

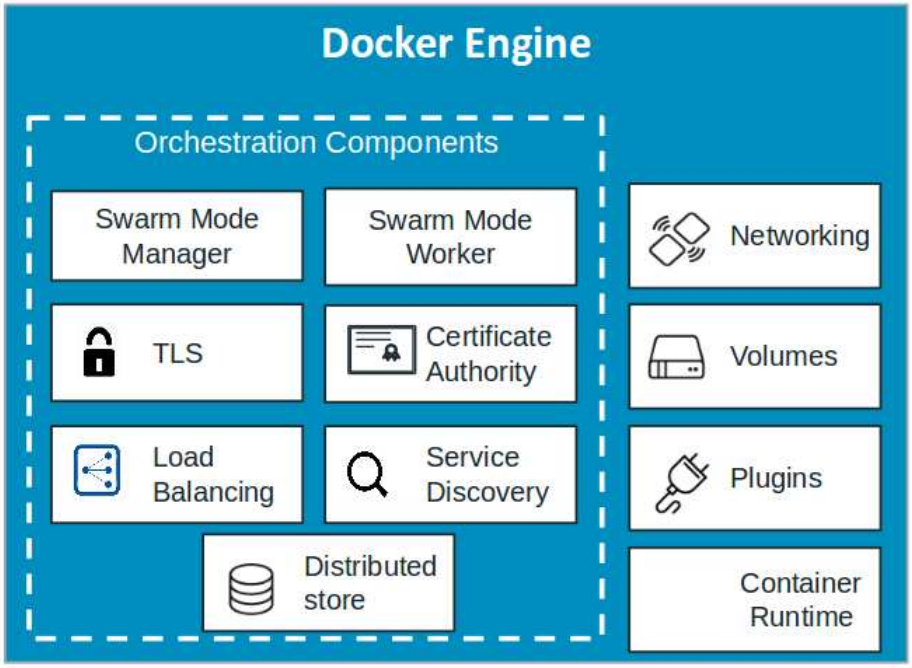
Docker permet d'automatiser le déploiement d'applications dans des conteneurs logiciels. Il peut empaqueter une application et ses dépendances dans un conteneur isolé, qui pourra être exécuté sur n'importe quel serveur.

Docker utilise des fonctionnalités natives de Linux comme :

- son noyau

- ses Cgroups (control groups) : fonctionnalités pour limiter, compter et isoler l'utilisation des ressources (processeur, mémoire, utilisation disque, etc.)
- une surcouche de LXC (LinuX Containers).

Outre le fait d'isoler un système de fichiers et des réseaux, Docker Engine intègre des outils d'orchestration, que nous aborderons dans le chapitre dédié à Swarm :



MAIS ALLONS DE CE PAS AU CŒUR DU SUJET…

INSTALLATION

Docker est distribué sous deux licences :

- Docker EE (Enterprise Edition) qui offre un support payant. Trois versions existent : Basic, Standard et Advanced.
- Docker CE (Community Edition), sans support.

Ce document se base sur Docker CE, et un système d'exploitation CentOS 8

I. Procédure

Paramétrer une IP fixe et une passerelle : **/etc/sysconfig/network-scripts/ifcfg-ens*33***
Faire correspondre le nom et l'adresse de l'interface réseau : **/etc/hosts**
Déclarer les DNS : **/etc/resolv.conf**
Nommer le serveur :

```
[root@CentOS ~]# hostnamectl set-hostname 'Docker'
```

Problème spécifique à CentOS 8 : pour permettre la résolution DNS des conteneurs dans Docker, nous devons désactiver le firewall :

```
[root@CentOS ~]# systemctl disable firewalld
```

Prévoir un espace disponible suffisant (plusieurs Go) sur le point de montage **/usr**
Les conteneurs seront stockés à cet endroit.

Paramétrage des dépôts de docker :

```
[root@CentOS ~]# dnf config-manager --add-repo=https://download.docker.com/linux/centos/docker-ce.repo
[root@CentOS ~]# dnf repolist -v
```

Recherche de la dernière version :

```
[root@CentOS ~]# dnf list docker-ce
Docker CE Stable - x86_64
Available Packages
docker-ce.x86_64                    3:19.03.5-3.el7                    docker-ce-
```

Installation :

La version de la dépendance par défaut du package **containerd.io** ne convient pas : il faut utiliser l'option **--nobest**

```
[root@CentOS ~]# dnf install docker-ce --nobest
```

```
[root@CentOS ~]# usermod -aG docker $USER
[root@CentOS ~]# id $USER
```

```
[root@CentOS ~]# systemctl start docker
[root@CentOS ~]# systemctl enable docker
```

II. Post-installation

Vérifications de l'installation :

```
[root@CentOS ~]# docker version
[root@CentOS ~]# docker info
```

Vérification des interfaces et réseaux virtuels :

Lors de l'installation, une interface virtuelle est créée :

```
[root@CentOS ~]# ifconfig -a
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 0.0.0.0
```

Docker met à disposition des réseaux virtuels, visibles via la commande suivante :

```
[root@CentOS ~]# docker network ls
NETWORK ID          NAME                DRIVER
7fca4eb8c647        bridge              bridge
9f904ee27bf5        none                null
cf03ee007fb4        host                host
```

Le réseau “bridge” utilise l’interface réseau « docker0 », qui est définie dans la machine hôte.

Test de l’installation :

```
[root@CentOS ~]# docker run hello-world
```

PREMIERS PAS

I. Apperçu des commandes

```
[root@CentOS ~]# docker --help

Management Commands:
  config      Manage Docker configs
  container   Manage containers
  image       Manage images
  network     Manage networks
  node        Manage Swarm nodes
  plugin      Manage plugins
  secret      Manage Docker secrets
  service     Manage services
  stack       Manage Docker stacks
  swarm       Manage Swarm
  system      Manage Docker
  trust       Manage trust on Docker images
  volume      Manage volumes

Commands:
  attach      Attach local standard input, output, and error streams to a running container
  build       Build an image from a Dockerfile
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  diff        Inspect changes to files or directories on a container's filesystem
  events      Get real time events from the server
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
  history     Show the history of an image
  images      List images
  import      Import the contents from a tarball to create a filesystem image
  info        Display system-wide information
  inspect     Return low-level information on Docker objects
  kill        Kill one or more running containers
  load        Load an image from a tar archive or STDIN
  login       Log in to a Docker registry
  logout      Log out from a Docker registry
  logs        Fetch the logs of a container
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  ps          List containers
  pull        Pull an image or a repository from a registry
  push        Push an image or a repository to a registry
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  rmi         Remove one or more images
  run         Run a command in a new container
  save        Save one or more images to a tar archive (streamed to STDOUT by default)
  search      Search the Docker Hub for images
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop one or more running containers
  tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
  top         Display the running processes of a container
  unpause     Unpause all processes within one or more containers
  update      Update configuration of one or more containers
  version     Show the Docker version information
  wait        Block until one or more containers stop, then print their exit codes
```

II. Installation d'un conteneur

A. Récupération de l'image

Les images de conteneurs peuvent contenir aussi bien des applications, que d'autres distributions Linux. Les exemples suivants vont se baser sur une image Debian, ayant une réputation d'au moins 10 étoiles :

```
[root@CentOS ~]# docker search --filter stars=10 debian
[root@CentOS ~]# docker pull debian
```

Liste des images locales :

```
[root@CentOS ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
debian              latest              978d85d02b87        2 weeks ago         123 MB
```

B. Création d'un conteneur

Créer un conteneur à partir de l'image Debian, nommé « DebCont0 », accessible avec un terminal (-t) interactif (-i)

```
[root@CentOS ~]# docker create --name DebCont0 debian
```

Vérification, en visualisant la liste des conteneurs :

```
[root@CentOS ~]# docker ps -a
CONTAINER ID   IMAGE    COMMAND       CREATED             STATUS          PORTS       NAMES
2e2d14b89c2b   debian   "/bin/bash"   4 seconds ago       Created                     DebCont0
```

C. Démarrage de conteneurs

Démarrer un conteneur déjà existant en attachant les signaux STDIN/OUT ERR en mode interactif :

```
[root@CentOS ~]# docker start -a -i DebCont0
```

Renommer un conteneur :

```
[root@CentOS ~]# docker rename DebCont0 ContDeb0
```

Démarrer un conteneur nommé « ContDeb1 » en tache de fond (option -d), depuis l'image « debian » avec un terminal en mode interactif :

```
[root@CentOS ~]# docker run -dti --name ContDeb1 debian
```

L'option **--rm** permet de détruire le conteneur une fois utilisé et stoppé

```
[root@CentOS ~]# docker run --rm --name ContDeb2 debian df -h
```

Exécution d'une commande dans le conteneur Debian déjà démarré

```
[root@CentOS ~]# docker exec -ti ContDeb1 cat /etc/issue
Debian GNU/Linux 8 \n \l
```

Démarrer un conteneur et accéder à son terminal

```
[root@CentOS ~]# docker run -ti debian
root@8fcceab47ae2:/# cat /etc/issue
Debian GNU/Linux 8 \n \l
```

Accéder à un conteneur démarré

```
[root@CentOS ~]# docker attach ContDeb1
```

Quitter un conteneur

```
root@8fcceab47ae2:/# <CTRL>p <CTRL>q
```

D. Arrêt / suppression de conteneurs / images

Accéder au shell d'un conteneur déjà démarré

```
[root@CentOS ~]# docker exec -ti ContDeb1 /bin/bash
```

Arrêt normal d'un conteneur

```
[root@CentOS ~]# docker stop ContDeb1
```

Arrêt forcé d'un conteneur

```
[root@CentOS ~]# docker kill ContDeb1
```

Effacer un conteneur

```
[root@CentOS ~]# docker rm ContDeb1
```

Effacement forcé d'un conteneur (même démarré)

```
[root@CentOS ~]# docker rm -f ContDeb1
```

Suppression de tous les conteneurs

```
[root@CentOS ~]# docker rm $(docker ps -a -q)
```

Effacer une image

```
[root@CentOS ~]# docker rmi -f debian
```

Suppression de toutes les images

```
[root@CentOS ~]# docker rmi $(docker images -q)
```

E. Paramètres de démarrage

Création d'un conteneur nginx nommé ContNgx1, limité à 1Go de swap, 512 Mo de RAM, avec mappage du dossier host local vers le dossier du conteneur /usr/share/nginx/html, et mappage du port externe 8080 vers le port interne 80

```
[root@CentOS ~]# docker run -dti --name ContNgx1 --memory-swap 1024m -m 512m -v $(pwd):/usr/share/nginx/html -p 8080:80 nginx
```

Nous pouvons vérifier que nginx répond bien à partir du serveur hôte (http://localhost:8080/)

Modification de quelques paramètres, à la volée, une fois le conteneur démarré :

```
[root@CentOS ~]# docker update --memory-swap 2048m -m 1024m ContNgx1
[root@CentOS ~]# docker update --cpuset-cpus 3 ContNgx1
```

F. Copie de fichiers

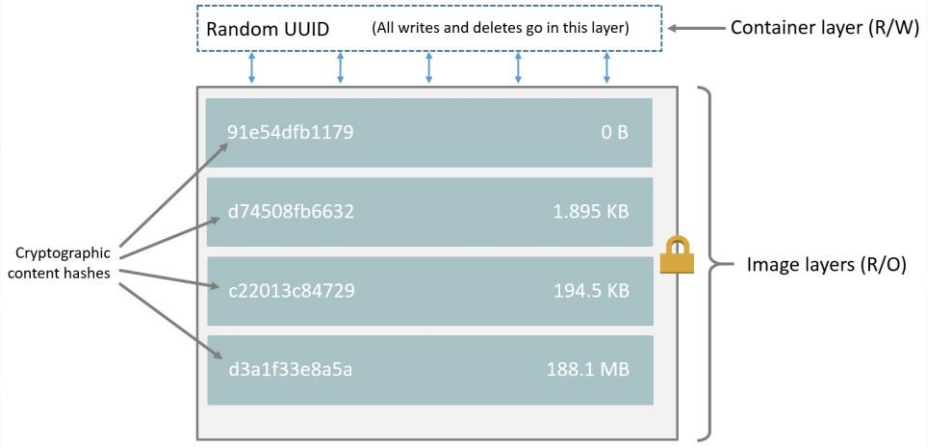
La copie de fichiers locaux dans un conteneur nommé « apache » s'effectue de cette manière :

```
[root@CentOS ~]# docker cp Docker/Apache/index.html apache:/var/www/localhost/htdocs/
```

IMAGES & LAYERS & CONTAINER

I. Modèle en couches

Un conteneur Docker est en réalité une couche (layer) contenant un système de fichiers modifiable, s'appuyant sur plusieurs autres couches en lecture seule, qui se superposent.

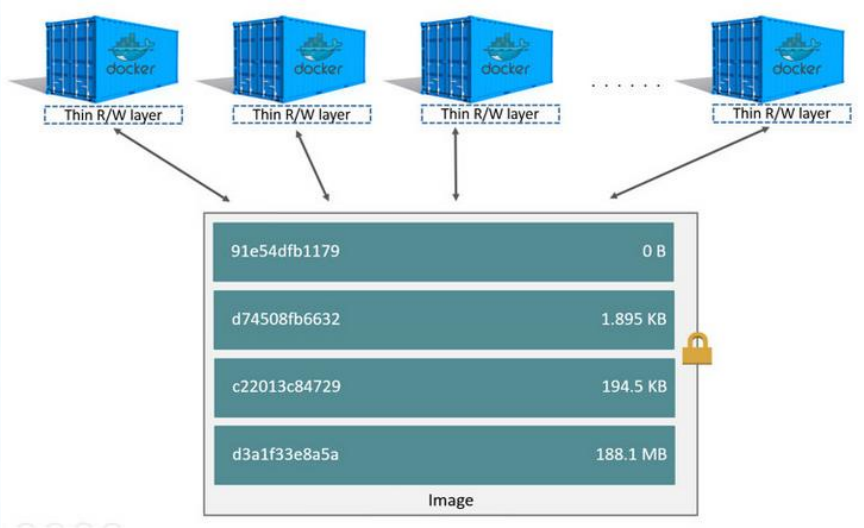


Source : www.docker.com

La liste de l'ensemble des couches formant une image est affichée via

```
[root@CentOS ~]# docker history <image>
```

A chaque création d'un nouveau conteneur basé sur une même instance d'image, seule la dernière couche en lecture/écriture est créée, ce qui optimise ainsi les ressources en stockage, et la rapidité d'instanciation d'un nouveau conteneur :



Source : www.docker.com

II. Pilotes et systèmes de fichiers

La gestion de ces couches est assurée par un driver Docker, qui est associé à un type de filesystem.

Il existe plusieurs types de drivers, qui sont pris en charge pas différents systèmes hôtes. Parmi les plus fréquents :

Driver	(sup)porté par
overlay / overlay2 (Linux kernel >= 4.0) / AUFS	ext4 / xfs

Btrfs	Btrfs
devicemapper	LVM
zfs	zfs

Le choix de ces pilotes peut être spécifié soit :

1/ dans la variable DOCKER_OPTS définie dans le fichier /etc/default/docker

2/ avec la commande :

```
[root@CentOS ~]# dockerd --storage-driver=devicemapper &
```

III. Conversion d'un conteneur en image

Imaginons que :

- l'on récupère une image Debian
- on créé un conteneur nommé « debian_nginx.updated »
- on le mette à jour, en installant Nginx
- on veuille créer une image nommée « debian/nginx:updated ».

La séquence sera la suivante :

```
[root@CentOS ~]# docker pull debian:latest
[root@CentOS ~]# docker run -dti --name debian_nginx.updated debian:latest
[root@CentOS ~]# docker exec -ti debian_nginx.updated apt-get -y update
[root@CentOS ~]# docker exec -ti debian_nginx.updated apt-get install -y nginx
```

On contrôle l'existence d'un différentiel entre l'image d'origine et le conteneur :

```
[root@CentOS ~]# docker diff debian_nginx.updated |head -10
...
[root@CentOS ~]# docker commit debian_nginx.updated  debian/nginx:updated
```

On contrôle la création de l'image :

```
[root@CentOS ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
debian/nginx        updated             f3234c725502        6 seconds ago       195 MB
debian              latest              8cedef9d7368        6 weeks ago         123 MB
```

Il est possible de renommer une image :

```
[root@CentOS ~]# docker tag debian/nginx:updated debian/nginx:latest
```

IV. Options de conversion

Il est possible d'intégrer des options de création d'image :

```
[root@CentOS ~]# docker commit -c "EXPOSE 80" debian_nginx.updated  debian/nginx:updated
```

…lors du lancement du conteneur construit à partir de cette image, le port 80 sera exposé.

V. Archivage d'une image

Les images peuvent être archivées (et compressées) :

```
[root@CentOS ~]# docker save debian/nginx:updated | gzip > debian_nginx:updated.tar.gz
```

…puis restaurées :

```
[root@CentOS ~]# docker load < debian_nginx:updated.tar.gz
```

DOCKERFILE

Les Dockerfiles sont des scripts permettant d'automatiser et d'industrialiser la création d'images personnalisées.

Exemple de script
```
[root@CentOS ~]# vi Dockerfile
```

```
# Only the instructions RUN, COPY, ADD create layers

# Image de référence
 FROM          debian:latest

# Données méta
 LABEL  fr.pierreau.dept="R&D" \
               fr.pierreau.version="1.0" \
               fr.pierreau.environment="production" \
               fr.pierreau.released="November 28, 2016"

# Commandes exécuteés pendant la création de l'image
 RUN           apt update && apt upgrade -y
 RUN           apt-get install -y nginx \
                                 vim \
                                 wget \
                                 zabbix-agent

# Copie de fichiers du host vers le conteneur
 COPY   nginx.conf /etc/nginx/nginx.conf
 COPY   service_start.sh /home/docker/script/service_start.sh
 RUN           chmod 744 /home/docker/script/service_start.sh

# Points de montage créé dans le conteneur
 VOLUME         ["/etc/nginx/sites-enabled", "/etc/nginx/certs", "/etc/nginx/conf.d",
"/var/log/nginx", "/var/www/html"]

# Configuration Apache
 RUN    echo 'ServerName apache.cloud.pierreau.fr:80' >> /etc/apache2/apache2.conf

# Ajout d'un compte de déploiement Ansible
 RUN   useradd -m -p deploy deploy

# Commande ENTRYPOINT (non surchargeable par docker run) lancée au démarrage du conteneur
 COPY  docker-entrypoint.sh /usr/sbin/
 RUN   chmod +x /usr/sbin/docker-entrypoint.sh
 ENTRYPOINT ["/usr/sbin/docker-entrypoint.sh"]

# Commande (surchargeable par docker run) lancée au démarrage du conteneur
 CMD           ping pierreau.fr

# Affectation d'une valeur à une variable d'environnement
 ENV           PS1="\u@\h\w \s-\v\$ "

# Changer de répertoire courant
 WORKDIR       /home/docker

# Ports réseau ouverts à l'extérieur du conteneur
 EXPOSE 80 443 123/udp
```

ENTRYPOINT :

```
[root@CentOS ~]# vi docker-entrypoint.sh
```

```
#!/bin/bash
set -e

/usr/sbin/apache2ctl -D FOREGROUND
/usr/sbin/zabbix_agentd -f

exec "$@"
```

En considérant que le fichier Dockerfile est dans le répertoire courant, la création de l'image nommée « debian:nginx » via ce fichier s'effectue avec la commande

```
[root@CentOS ~]# docker build -t 'debian:nginx' .
```

On peut spécifier le chemin du dossier d'un autre dockerfile de cette manière

```
[root@CentOS ~]# docker build -f /[path]/Dockerfile .
```

L'image créée sera constituée de plusieurs couches (layers), visibles avec la commande

```
[root@CentOS ~]# docker history debian:nginx
```

VOLUMES

Un volume permet à des conteneurs de stocker des informations persistantes. Il peut être partagé entre plusieurs containers.

I. Création d'un volume

Création d'un volume :

```
[root@CentOS ~]# docker volume create DataVolume1
```

Vérifications :

```
[root@CentOS ~]# docker volume ls
DRIVER              VOLUME_NAME
local               DataVolume1

[root@CentOS ~]# docker volume inspect DataVolume1
```

Partage de ce volume dans deux dossiers « ExtVolume » montés dans 2 conteneurs « DebCont1 » (effacé automatiquement lors de son arrêt), et « DebCont2 » :

```
[root@CentOS ~]# docker run -ti --rm --name DebCont1 -v DataVolume1:/ExtVolume debian
root@6c780805a52c:/# ls ExtVolume
root@6c780805a52c:/# touch ExtVolume/test
root@6c780805a52c:/# ls ExtVolume
test
<CTRL P> <CTRL Q>
[root@CentOS ~]# docker stop DebCont1

[root@CentOS ~]# docker run -ti --name DebCont2 -v DataVolume1:/ExtVolume debian
root@9825b015920a:/# ls ExtVolume
test
```

II. Création à la volée

Il est possible de créer un volume persistant dès la création ou le démarrage d'un conteneur :

```
[root@CentOS ~]# docker create --name DebCont0 -t -i -v DataVolume1:/ExtVolume debian

[root@CentOS ~]# docker run -ti -name DebCont1 -v DataVolume1:/ExtVolume debian
```

III. Droits sur les conteneurs

La lecture seule sur un volume est assurée par le paramètre read-only **:ro**

```
[root@CentOS ~]# docker run -ti -name DebCont1 -v DataVolume1:/ExtVolume:ro debian
```

IV. Gestion des volumes

Supprimer un volume :

```
[root@CentOS ~]# docker volume rm DataVolume1
```

Suppression des volumes non utilisés par des conteneurs :

```
[root@CentOS ~]# docker volume prune
```

DOCKER COMPOSE

Docker Compose est un outil pour définir et exécuter des applications Docker multi-conteneurs interdépendants. Il utilise :

- La définition de l'environnement via un Dockerfile
- la définition des services contenus dans un fichier « docker-compose.yml »
- la commande « docker-compose up » pour démarrer l'environnement isolé

I. Installation

Identifier la dernière version en date sur l'URL https://github.com/docker/compose/releases/

Définir la variable d'environnement en fonction de la dernière version :

```
[root@CentOS ~]# dockerComposeVersion="1.25.1"
```

Installer Docker Compose :

```
[root@CentOS ~]# dnf install curl
[root@CentOS ~]# curl -L
https://github.com/docker/compose/releases/download/$dockerComposeVersion/docker-compose-
`uname -s`-`uname -m` > /usr/local/bin/docker-compose
```

Définir les droits d'execution :

```
[root@CentOS ~]# chmod +x /usr/local/bin/docker-compose
```

II. Définition de l'environnement

A. Dockerfile

```
[root@CentOS ~]# vi Dockerfile
```

```
# Image de référence
 FROM          nginx:alpine

# Copie de fichiers du host vers le conteneur
 COPY   nginx.conf /etc/nginx/nginx.conf
```

B. docker-compose.yml

```
[root@CentOS ~]# vi docker-compose.yml
```

```
version: "3.5"
services:
    reverseproxy:
        image: reverseproxy
        container_name: reverseproxy
        ports:
            - "8080:8080"
            - "8081:8081"
        environment:
            - PGID=133
        restart: always

    nginx:
        depends_on:
            - reverseproxy
        image: nginx:alpine
```

```
        container_name: nginx
        restart: always

    apache:
        depends_on:
            - reverseproxy
        image: httpd:alpine
        container_name: apache

        volumes:
            - apache:/var/www/html/

# Les fichiers /etc/hosts et /etc/resolv.conf  sont regénérés à chaque démarrage du
conteneur

        extra_hosts:
            - "ns1.cloud.pierreau.fr: 172.16.137.204"
        dns:
            - 172.16.137.204
            - 1.1.1.1
        restart: always

volumes:
    apache:
        name: apache-http

# Utilisation d'un réseau existant
networks:
    default:
        external:
            name: Pierreau
```

Le nombre d'instances de l'application est fixé à 4 :

```
[root@CentOS ~]# docker-compose scale app=4
```

Liste des conteneurs :

```
[root@CentOS ~]# docker-compose ps
```

Liste des images utilisées :

```
[root@CentOS ~]# docker-compose images
```

Arrêt des conteneurs gérés par Docker-Compose :

```
[root@CentOS ~]# docker-compose stop
```

Supprimer l'ensemble des conteneurs crées avec docker-compose :

```
[root@CentOS ~]# docker-compose rm
```

DOCKER MACHINE

Contactez-moi
pour planifier vos formations,
et aborder la suite du programme ci-dessous.

https://pierreau.fr/Contact/index.php
Tél : (+33) 614 672 909

A bientôt…

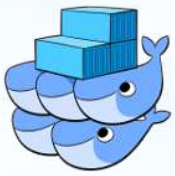
I. Installation

II. Création d'un cluster VirtualBox

A. Installer virtualbox

B. Création de 3 machines

SWARM



I. Vue d'ensemble

II. Briques techniques

III. Réseau virtuel et sécurité

A. Certificats

B. Docker network

IV. Mise en oeuvre

A. Ouverture des ports

B. Initialisation d'un noeud

C. Création d'un service

D. Scalabilité

SUPERVISION

I. Commandes

II. Outils GUI

Plusieurs environnements graphiques peuvent être installés afin de contrôler l'environnement Docker :

A. Shipyard

B. Portainer

C. Rancher

Interface élaborée permettent de travailler avec des instances dans les clouds (Amazon, Azure…)

Kubernetes

I. Fonctionnement

II. Prérequis

A. Stockage

B. Virtualisation

C. Compte de service

D. Docker

E. SeLinux

III. Installations

A. Virtualbox

B. Kubernetes (k8s)

C. Minikube

D. k3s

E. Microk8s

IV. Ajout de nodes en cluster

V. Ajout de pods répliqués

VI. Exposition d'un service en haute disponibilité