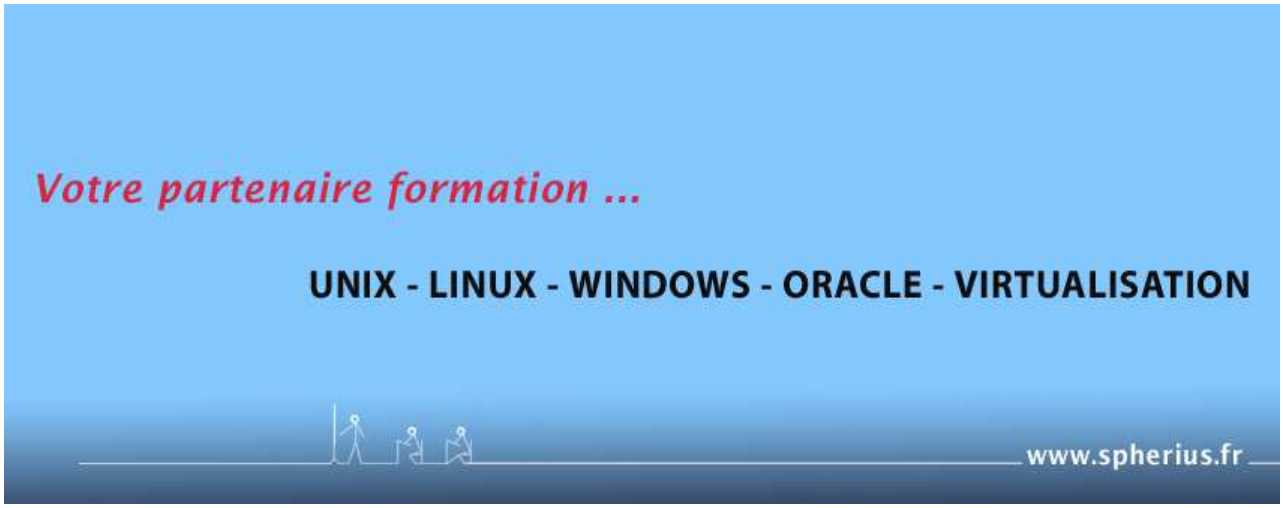
DOCKER



SOMMAIRE

Les auteurs de ce document sont :

- Monsieur Baranger Jean-Marc,
- Monsieur Schomaker Theo.

La version de Linux utilisée pour les commandes de ce support de cours est :

CentOS 7

Les références sont : les documents disponibles sur le site web de Docker.

Présentation de Docker

Dans ce chapitre, nous présentons Docker et ses concepts.

Présentation de Docker

- Le concept de Docker
- Image et conteneur
- L'intérêt de Docker et les besoins
- La virtualisation et docker
- Les conteneurs Linux : LXC , namespace et control-cgroups
- Les différentes éditions de Docker

Présentation de Docker

Le concept de Docker

- Flexible
- Léger
- Interchangeable
- Portable
- Évolutif
- Empilable

Le concept de Docker

Ci-dessous une présentation issue du site de Docker :

Docker est une plate-forme permettant aux développeurs et aux administrateurs système de développer, déployer et exécuter des applications avec des conteneurs. L'utilisation de conteneurs Linux pour déployer des applications s'appelle la conteneurisation. Les conteneurs ne sont pas nouveaux, mais leur utilisation pour déployer facilement des applications l'est.

La conteneurisation est de plus en plus populaire car les conteneurs sont :

Flexible:	même les applications les plus complexes peuvent être conteneurisées.
Léger:	Les conteneurs exploitent et partagent le noyau hôte.
Interchangeable:	vous pouvez déployer des mises à jour et des mises à niveau à la volée.
Portable:	vous pouvez les créer localement, les déployer sur le cloud et les exécuter n'importe où.
Évolutif:	vous pouvez augmenter et distribuer automatiquement les réplicas de conteneur.
Empilable:	vous pouvez empiler les services verticalement et à la volée.

Présentation de Docker

Image et conteneur

- Conteneur

  instance d'une image

- Image

  package exécutable

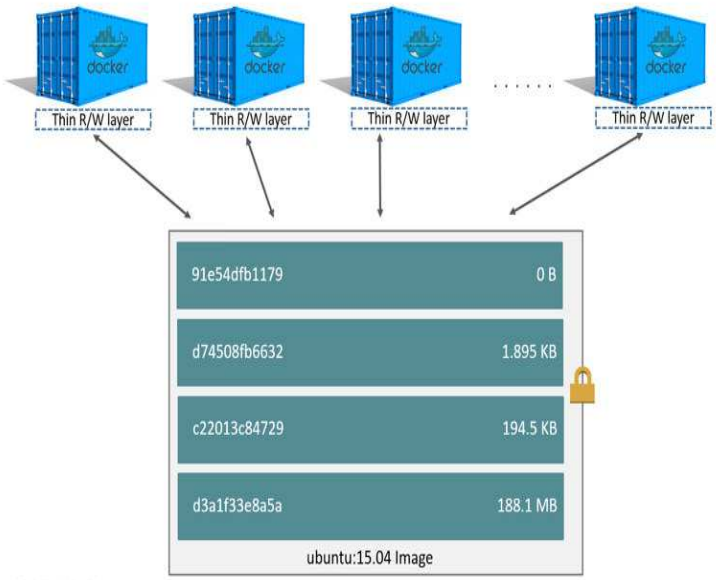


Image et conteneur

Un conteneur est lancé en exécutant une image.

Une image est un package exécutable qui inclut tout ce dont vous avez besoin pour exécuter une application: le code, une exécution, des bibliothèques, des variables d'environnement et des fichiers de configuration.
Une image est en lecture seule.

Un conteneur est une instance d'exécution active ou inactive d'une image.
C'est à dire : c'est ce que devient l'image en mémoire lorsqu'elle est exécutée.
Le conteneur est en lecture / écriture.

Vous pouvez voir une liste de vos conteneurs en cours d'exécution avec la commande, docker ps, comme vous le feriez sous Linux.

Présentation de Docker

L'intérêt de Docker et les besoins

- le packaging d'applications,
- des déploiements rapides,
- de pouvoir exécuter toute sorte d'applications,
- de faire coexister différents versions d'une même application sur un serveur,
- un accès immédiat à des applications pré-packagées,
- de faire transiter rapidement une application d'un environnement de développement à une infrastructure de production,
- de pouvoir exécuter des conteneurs sur n'importe quelle plate-forme : physique, virtuelle ou cloud,
- de conserver le serveur hôte « propre ».

L'intérêt de Docker et les besoins

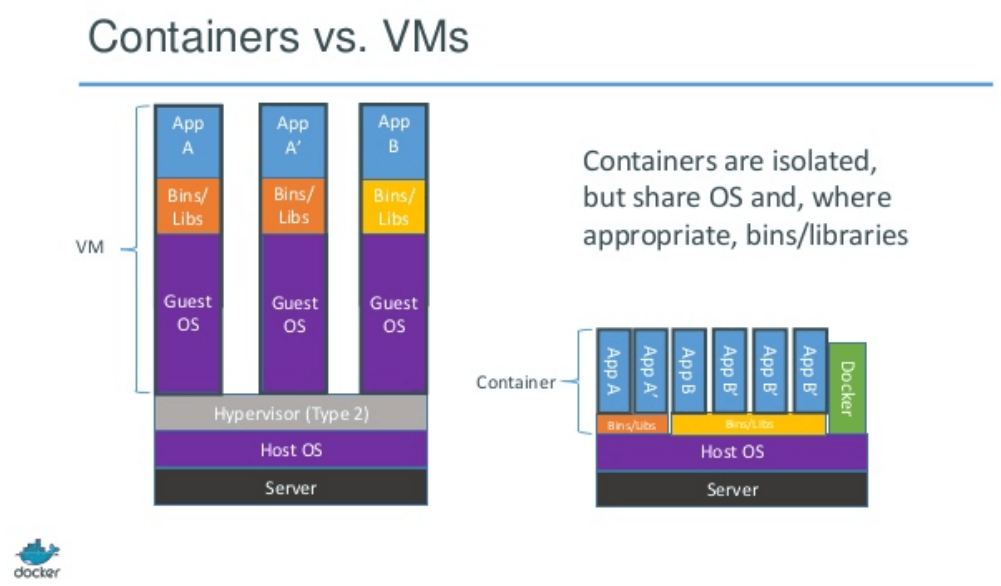
Docker répond aussi bien à des besoins et attentes des équipes de développement, de validation et de production.

Docker permet entre autres :

- le packaging d'applications,
- des déploiements rapides,
- de pouvoir exécuter toute sorte d'applications,
- de faire coexister différentes versions d'une même application sur un serveur,
- un accès immédiat à des applications pré-packagées,
- de faire transiter rapidement une application d'un environnement de développement à une infrastructure de production,
- de pouvoir exécuter des conteneurs sur n'importe quelle plate-forme : physique, virtuelle ou cloud,
- de conserver le serveur hôte « propre ».

Présentation de Docker

La virtualisation et Docker



La virtualisation et Docker

Un conteneur s'exécute nativement sur Linux et partage le noyau de la machine hôte avec d'autres conteneurs. Il exécute un processus discret, ne prenant pas plus de mémoire que tout autre exécutable, ce qui le rend léger.

En revanche, une machine virtuelle (VM) exécute un système d'exploitation «invité» à part entière avec un accès virtuel aux ressources hôte via un hyperviseur. En général, les machines virtuelles fournissent un environnement avec plus de ressources que la plupart des applications.

Présentation de Docker

Les conteneurs Linux : LXC , namespace et control-cgroups

- LXC
- Namespace
- Control-cgroups

Les conteneurs Linux : LXC, namespace et control-cgroups

Linux Containers : LXC

Le container Linux est une méthode de virtualisation différente des solutions de virtualisation applicatives comme KVM, Virtualbox ou VMware. LXC permet l'isolation des ressources comme la CPU, la mémoire, le réseau, les entrées/sorties disques, ...
Un container LXC crée un environnement virtualisé d'un système d'exploitation mais utilise le noyau du système hôte.

Les cgroups

Les Cgroups permettent de limiter l'accès aux ressources pour des services et des utilisateurs. Les ressources contrôlables sont la quantité de RAM, CPU, la bande passante en écriture ou en lecture, etc …

namespace

Le namespace est une fonctionnalité du noyau Linux qui partitionne les ressources du noyau de façon à ce qu'un ensemble de processus voit un ensemble de ressources. Un autre ensemble de ressources voyant un ensemble de ressources différent.

Les espaces de noms sont apparus avec le noyau 2.4.19 en 2002 qui fonctionne avec l'espace de nom de montage. Des espaces de noms ont été ajoutés au fur à mesure.
Il existe actuellement 7 types d'espaces de noms. La fonctionnalité de l'espace de noms est la même pour tous les types: chaque processus est associé à un espace de noms et ne peut voir ou utiliser que les ressources associées à cet espace de noms et les espaces de noms descendants le cas échéant. De cette façon, chaque processus (ou groupe de ceux-ci) peut avoir une vue unique sur la ressource. La ressource isolée dépend du type d'espace de noms créé pour un groupe de processus donné.

Présentation de Docker

Les différentes éditions de Docker

Docker Community Edition - Docker CE

Les éditions CE Stable et CE Edge

Docker Enterprise Edition - Docker EE

Les éditions EE Basic, EE Standard et EE Advanced

Les différentes éditions de Docker

Docker Community Edition (Docker CE) est disponible gratuitement. Il est idéal pour les développeurs et les petites équipes.
Docker CE est disponible pour de nombreuses plates-formes d'infrastructure.
La version Stable fournie des mises à jour fiables chaque trimestre.
La version Edge fournie de nouvelles fonctionnalités tous les mois.

Docker Enterprise Edition (Docker EE) est une solution payante proposée aux entreprises.
Docker EE est une plate-forme de conteneur prête à l'emploi. Elle est conçue pour le développement d'entreprise et les équipes informatiques qui construisent, expédient et exécutent des applications stratégiques en production et à grande échelle. Docker EE est intégré, certifié et pris en charge pour fournir aux entreprises une plate-forme de conteneurs sécurisées.
Il existe les éditions EE Basic, EE Standard et EE Advanced.

Notes

Installation et configuration de Docker

Dans ce chapitre, nous allons étudier le déploiement de Docker.

Installation et configuration de Docker

- Installation de Docker
- Configuration de Docker
- Syntaxe d'une commande Docker
- L'aide de Docker

Installation et configuration de Docker

Installation de Docker

Linux Windows MacOS

- Boot2Docker et Docker-toolbox

- Les pré-requis
- Les dépôts
- L'installation de Docker
- Vérification

Installation de Docker

L'installation de Docker peut se faire sous différents environnements systèmes Linux, MacOS et Windows.

Remarques pour Windows :
Docker est conçu pour fonctionner sous Linux, ainsi pour Windows, on peut déployer Boot2Docker qui est une VM Linux proposée par Docker.
On peut également télécharger docker-toolbox. Son installation déploie tout le nécessaire pour Docker, en autres : virtualbox avec une VM Boot2Docker, un client Docker, docker-machine et docker-compose. Si on le souhaite, Kitematic est une interface graphique pour installer des applications via Docker.

Veuillez vous référer au site de Docker pour les différents types d'installations. Nous détaillons ci-dessous le processus pour une installation sur Linux (CentOS).

Installation des pré-requis :

```
# yum  install  -y  yum-utils  \
  device-mapper-persistent-data  lvm2
```

Au minimum yum-utils, pour les deux autres packages cela concerne le driver de stockage devicemapper.

Configuration des dépôts :

Ajout d'un dépôt stable :

```
# yum-config-manager  --add-repo \
      https://download.docker.com/linux/centos/docker-ce.repo
Modules complémentaires chargés : fastestmirror, langpacks
adding repo from: https://download.docker.com/linux/centos/docker-ce.repo
grabbing file https://download.docker.com/linux/centos/docker-ce.repo to
/etc/yum.repos.d/docker-ce.repo
repo saved to /etc/yum.repos.d/docker-ce.repo
```

Activation du dépôt edge :

Il fait partie du dépôt docker.repo mais il est désactivé par défaut.

```
# yum-config-manager  --enable  docker-ce-edge
```

L'option --disable permet de le désactiver :

```
# yum-config-manager  --disable  docker-ce-edge
```

Mise à jour de l'index des paquetages de yum :

```
# yum  makecache  fast
```

Installation de Docker :

Lister les versions disponibles :

```
# yum  list  docker-ce.x86_64  --showduplicates  |  sort  -r
 * updates: distrib-coffee.ipsl.jussieu.fr
Paquets disponibles
Modules complémentaires chargés : fastestmirror, langpacks
Loading mirror speeds from cached hostfile
 * extras: mirror.in2p3.fr
docker-ce.x86_64            17.03.0.ce-1.el7.centos             docker-ce-stable

      base: ftp.rezopole.net
```

Installation de Docker :

```
# yum  install  docker-ce
```

Démarrage de Docker :

```
# systemctl  start  docker              {start|stop|restart|enable|disable}
```

```
# ps -e | grep dock
root      1125     1  1 10:49 ?        00:00:30 /usr/bin/dockerd
root      1577  1125  0 10:49 ?        00:00:19 docker-containerd
```

Vérifications :

```
# docker  version
Client:
 Version:    18.03.0-ce
 API version:1.37
 Go version: go1.9.4
 Git commit: 0520e24
 Built:      Wed Mar 21 23:09:15 2018
 OS/Arch:    linux/amd64
 Experimental:      false
 Orchestrator:      swarm

Server:
 Engine:
  Version:   18.03.0-ce
  API version:      1.37 (minimum version 1.12)
  Go version:go1.9.4
  Git commit:0520e24
  Built:     Wed Mar 21 23:13:03 2018
  OS/Arch:   linux/amd64
  Experimental:     false
```

```
# docker  run  hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
9bb5a5d4561a: Pull complete
Digest: sha256:f5233545e43561214ca4891fd1157e1c3c563316ed8e237750d59bde73361e77
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/engine/userguide/
```

Le groupe utilisateurs docker :

Pour exécuter des commandes Docker sans passer par le compte root, il faut que votre compte utilisateur soit associé au groupe 'docker'.

Installation et configuration de Docker

Configuration de Docker

- /etc/docker
- /var/lib/docker
- /usr/lib/systemd/system/docker.service

Configuration de Docker

Le répertoire /etc/docker :

```
[root@poste]# ls  -R  /etc/docker/
/etc/docker/:
key.json
```

Le fichier key.json contient la clé pour dockerd pour les connexions TLS.

Le contenu peut varier en fonction de la version de Docker :

```
# ls  -R  /etc/docker
/etc/docker/:
certs.d  daemon.json  key.json  seccomp.json

/etc/docker/certs.d:
redhat.com  redhat.io  registry.access.redhat.com

/etc/docker/certs.d/redhat.com:
redhat-ca.crt

/etc/docker/certs.d/redhat.io:
redhat-ca.crt

/etc/docker/certs.d/registry.access.redhat.com:
redhat-ca.crt
```

Le répertoire /var/lib/docker :

```
# cd  /var/lib/docker
# ls
builder     containers  network   plugins   swarm  trust
containerd  image       overlay2  runtimes  tmp    volumes
```

Le répertoire containers contient un répertoire par conteneur.

Le fichier de configuration du service Docker :

```
# cat /usr/lib/systemd/system/docker.service
[Unit]
Description=Docker Application Container Engine
Documentation=https://docs.docker.com
After=network-online.target firewalld.service
Wants=network-online.target

[Service]
Type=notify
# the default is not to use systemd for cgroups because the delegate issues stil
l
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker
ExecStart=/usr/bin/dockerd
ExecReload=/bin/kill -s HUP $MAINPID
# Having non-zero Limit*s causes performance problems due to accounting overhead
# in the kernel. We recommend using cgroups to do container-local accounting.
LimitNOFILE=infinity
LimitNPROC=infinity
LimitCORE=infinity
# Uncomment TasksMax if your systemd version supports it.
# Only systemd 226 and above support this version.
#TasksMax=infinity
TimeoutStartSec=0
# set delegate yes so that systemd does not reset the cgroups of docker containe
rs
Delegate=yes
# kill only the docker process, not all processes in the cgroup
KillMode=process
# restart the docker process if it exits prematurely
Restart=on-failure
StartLimitBurst=3
StartLimitInterval=60s

[Install]
WantedBy=multi-user.target
```

Installation et configuration de Docker

Syntaxe d'une commande Docker

```
# docker  COMMAND  OPTIONS  ARGUMENTS

# docker  run  -i
# docker  run  --interactive
# docker  run  -i=true

# docker  ps
# docker  ps  -a
# docker  search  hello-world
# docker  run  -it centos
# docker  run  -i  -t centos
# docker  run  -it  --name mon_conteneur  centos
```

Syntaxe d'une commande Docker

La syntaxe est :

```
# docker  COMMAND  OPTIONS  ARGUMENTS
```

Différents formats sont exploitables :

```
# docker  run  -i
# docker  run  --interactive
# docker  run  -i=true
```

L'auto-complétion est disponible :

```
# docker  r     suivi de 2 tabulations
```

Exemples :

```
# docker  ps
# docker  ps  -a
# docker  search  hello-world
# docker  run  -it centos
# docker  run  -i  -t centos
# docker  run  -it  --name mon_conteneur  centos
```

Installation et configuration de Docker

L'aide de Docker

Le site Les mans Le --help

```
# man  docker
# man  docker-run
# man  docker-network


# docker  --help
# docker  network  --help
# docker  network  connect  --help
```

L'aide de Docker

Le site de Docker, les mans et la syntaxe avec « --help » détaillent les commandes.

```
# man  docker
# man  docker-run
# man  docker-network
```

```
# docker  --help
# docker  network  --help
# docker  network  connect  --help
```

```
# docker  --help

Usage: docker COMMAND

A self-sufficient runtime for containers

Options:
      --config string      Location of client config files (default "/root/.docker")
  -D, --debug              Enable debug mode
      --help               Print usage
  -H, --host list          Daemon socket(s) to connect to (default [])
  -l, --log-level string   Set the logging level ("debug", "info", "warn", "error",
"fatal") (default "info")
      --tls                Use TLS; implied by --tlsverify
      --tlscacert string   Trust certs signed only by this CA (default
"/root/.docker/ca.pem")
      --tlscert string     Path to TLS certificate file (default
"/root/.docker/cert.pem")
      --tlskey string      Path to TLS key file (default "/root/.docker/key.pem")
```

```
      --tlsverify          Use TLS and verify the remote
  -v, --version            Print version information and quit

Management Commands:
  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
  volume      Manage volumes

Commands:
  attach      Attach 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 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

Run 'docker COMMAND --help' for more information on a command.
```

Notes

Mise en œuvre de Docker en ligne de commandes

Dans ce chapitre, nous allons étudier les lignes de commandes Docker.

Mise en œuvre de Docker en ligne de commandes

- Les ressources centralisées : Le Docker Hub
- Récupérer des informations : search, etc
- Fonctionnement d'un conteneur
- Arrêt et démarrage d'un conteneur
- Suppression
- Détail d'un conteneur : inspect
- Les différences entre deux conteneurs : diff
- Les logs, events, top et stats
- Les variables
- La copie de fichiers
- pause, unpause et wait
- Redémarrage automatique d'un conteneur
- Démarrer un conteneur au boot du serveur
- Conteneur et stockage
- Les ports réseaux : publication
- La gestion des ressources

Mise en œuvre de Docker en ligne de commandes

Les ressources centralisées : Le Docker Hub

- Portail d'hébergement de partage d'images Docker.
- Registry par défaut.
- Création d'un compte personnel.

Les ressources centralisées : Le Docker Hub

Docker Hub est un portail d'hébergement de partage d'images Docker pré-paramétrées.

C'est est un référentiel basé sur le cloud dans lequel les utilisateurs et les partenaires Docker créent, stockent et distribuent des images de conteneur. Grâce à Docker Hub, un utilisateur peut accéder à des référentiels d'images publics et open source, et utiliser un espace pour créer ses propres référentiels privés.

Docker Hub est le registry par défaut.

On peut créer un compte gratuitement directement sur Docker Hub.

```
# docker  login                 pour s'authentifier avec un accès par login et mot de passe.
# docker  logout                pour se déconnecter.

# docker  push   image:tag      pour exporter son image vers le registry.
# docker  pull   image:tag      pour importer son image en local.
```

Mise en œuvre de Docker en ligne de commandes

Récupérer des informations : search, etc

```
# docker  search  debian

# docker  search  debian  --filter=stars=500
# docker  search  debian  -f=stars=500
# docker  search  debian  -f stars=500
# docker  search  debian  --filter="is-official=true"
# docker  search  debian  --filter="is-official=true"  --no-trunc

# docker  images
# docker  images  ubuntu*

# docker  ps  -a

# docker  info
```

Récupérer des informations : search, etc

La sous commande search permet de rechercher des images.

```
# docker  search  debian
NAME                DESCRIPTION                     STARS       OFFICIAL      AUTOMATED
ubuntu              Ubuntu is a Debian-based…       7702        [OK]
debian              Debian is a Linux distri…       2577        [OK]
google/debian                                       52                        [OK]
neurodebian         NeuroDebian provides neu…       50          [OK]
arm32v7/debian      Debian is a Linux distri…       35
armhf/debian        Debian is a Linux distri…       31
itscaro/debian-ssh  debian:jessie                   23                        [OK]
```

De la commande précédente, on peut filtrer les résultats :

```
# docker  search  debian --filter=stars=500
NAME      DESCRIPTION                                     STARS     OFFICIAL    AUTOMATED
ubuntu    Ubuntu is a Debian-based Linux operating sys…   7702      [OK]
debian    Debian is a Linux distribution that's compos…   2577      [OK]
```

Les options peuvent être utilisées au format long ou au format court.

```
# docker  search  debian  -f=stars=500
NAME      DESCRIPTION                                     STARS     OFFICIAL    AUTOMATED
ubuntu    Ubuntu is a Debian-based Linux operating sys…   7702      [OK]
debian    Debian is a Linux distribution that's compos…   2577      [OK]
```

```
# docker  search  debian  -f stars=500
NAME      DESCRIPTION                                     STARS     OFFICIAL    AUTOMATED
ubuntu    Ubuntu is a Debian-based Linux operating sys…   7702      [OK]
debian    Debian is a Linux distribution that's compos…   2577      [OK]
```

```
# docker  search  debian  --filter="is-official=true"
AME         DESCRIPTION                                     STARS     OFFICIAL  AUTOMATED
ubuntu       Ubuntu is a Debian-based Linux operating sys…   7702      [OK]
debian       Debian is a Linux distribution that's compos…   2577      [OK]
neurodebian  NeuroDebian provides neuroscience research s…   50        [OK]
```

Pour avoir le descriptif complet :

```
# docker  search  debian  --filter="is-official=true"  --no-trunc
NAME         DESCRIPTION
                    STARS              OFFICIAL            AUTOMATED
ubuntu       Ubuntu is a Debian-based Linux operating system based on free software.
                    7702                [OK]
debian       Debian is a Linux distribution that's composed entirely of free and open-
source software.    2577                [OK]
neurodebian  NeuroDebian provides neuroscience research software for Debian, Ubuntu, and
other derivatives.  50                  [OK]
```

Pour plus de détails sur les filtres:
https://docs.docker.com/engine/reference/commandline/search/

ou

```
# man  docker-search
```

Afficher les images Docker

```
# docker  images
REPOSITORY              TAG        IMAGE ID            CREATED             SIZE
docker.io/ubuntu        latest     f975c5035748        2 weeks ago         112 MB
docker.io/centos        latest     2d194b392dd1        2 weeks ago         195 MB
docker.io/hello-world   latest     f2a91732366c        4 months ago        1.85 kB
```

REPOSITORY: Le nom du dépôt et de l'image.

TAG: La version de l'image.
Cela peut-être un numéro de version ou un mot clef (exemple: latest).

IMAGE ID: Un numéro unique qui identifie l'image.

CREATED: La date de création de l'image.

SIZE: La taille de l'image.

```
# docker  images  ubuntu*
REPOSITORY          TAG                 IMAGE ID            CREATED              SIZE
ubuntu_theo         2.0                 fbc3f044bc57        About a minute ago   79.6MB
ubuntu_theo         1.0                 304662ebabcf        About a minute ago   79.6MB
ubuntu_theo         latest              076f47d3d2e3        2 minutes ago        79.6MB
ubuntu              latest              452a96d81c30        7 days ago           79.6MB
```

Afficher les conteneurs Docker

```
# docker  ps  -a
CONTAINER ID   IMAGE    COMMAND    CREATED         STATUS         PORTS  NAMES
de47694487db   centos   "bash"     19 hours ago    Up 3 minutes          inspiring_franklin
```

CONTAINER ID: numéro unique qui identifie le container.

IMAGE: Nom de l'image chargée au sein du container.

COMMAND: Commande s'exécutant au sein du container.

CREATE: Date de création du container.

STATUS: Le statut du container.

PORTS: Les ports accessibles du container.

NAMES: Le nom du conteneur. Le système en affecte un automatiquement.
Il est possible d'affecter un nom personnalisé au container.

```
# docker  ps  -a
CONTAINER ID        IMAGE       COMMAND         CREATED         STATUS
PORTS               NAMES
9d510aab7252  centos_version1  "/bin/bash"      32 minutes ago  Exited (0) 32 minutes ago
                    phase1_nginx
c930cb5bf5f0  centos_version1  "/bin/bash"      32 minutes ago      Up 34 seconds
                    kickass_goodall
41d567206949  centos_version1  "--name phase1_nginx"   33 minutes ago    Created
                    zen_swirles
```

Afficher des informations relatives à Docker

```
# docker  info
Containers: 5
 Running: 0
 Paused: 0
 Stopped: 5
Images: 51
Server Version: 18.03.0-ce
Storage Driver: overlay2
 Backing Filesystem: xfs
 Supports d_type: true
 Native Overlay Diff: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: bridge host macvlan null overlay
 Log: awslogs fluentd gcplogs gelf journald json-file logentries splunk syslog
Swarm: inactive
Runtimes: runc
Default Runtime: runc
Init Binary: docker-init
containerd version: cfd04396dc68220d1cecbe686a6cc3aa5ce3667c
runc version: 4fc53a81fb7c994640722ac585fa9ca548971871
init version: 949e6fa
Security Options:
 seccomp
  Profile: default
Kernel Version: 3.10.0-693.21.1.el7.x86_64
Operating System: CentOS Linux 7 (Core)
OSType: linux
Architecture: x86_64
CPUs: 2
Total Memory: 1.796GiB
Name: mars
ID: 74CL:QEUC:G6EF:2Q4V:S6NQ:KKNE:TLHV:XDDU:X7TC:QXIL:ULEL:X3IM
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): false
Registry: https://index.docker.io/v1/
Labels:
Experimental: false
Insecure Registries:
 127.0.0.0/8

Live Restore Enabled: false
```

Mise en œuvre de Docker en ligne de commandes

Fonctionnement d'un conteneur

```
# docker  run  hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
9bb5a5d4561a: Pull complete
Digest: sha256:f5233545e43561214ca4891fd1157e1c3c563316ed8e237750d59bde73361e77
Status: Downloaded newer image for hello-world:latest

# docker  run  alpine  cat /etc/hosts
# docker  run  -it  alpine     ou    docker  run  -i -t  alpine
# docker  run  --name container_theo  ubuntu

# docker  rename  container_theo  intranet

# docker  pull  centos:latest

# docker  create  --name container1  nginx

# docker  exec  -it container1  ls -l /usr/share/nginx
```

Fonctionnement d'un conteneur

L'instruction run permet de faire fonctionner un conteneur à partir d'une image.

Premier exemple :

```
# docker  run  hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
9bb5a5d4561a: Pull complete
Digest: sha256:f5233545e43561214ca4891fd1157e1c3c563316ed8e237750d59bde73361e77
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
. . .
```

Docker Engine a tenté de trouver une image nommée hello-world. C'est la première fois qu'on sollicite cette image, elle n'est donc pas stockée en local (Unable to find image 'hello-world:latest' locally).
Docker Engine va donc dans son registre Docker par défaut, Docker Hub (https://hub.docker.com), pour rechercher une image nommée "hello-world". Il trouve l'image sur le site et la transfère en local (Pulling from library/hello-world).
Puis Docker Engine exécute une instance de cette image dans un conteneur. La seule fonction de hello-world est de sortir le texte que vous voyez dans votre terminal, après quoi le conteneur s'arrête.
Le conteneur n'est pas supprimé, ni l'image.

Deuxième exemple :

```
# docker  run  alpine  cat /etc/hosts
127.0.0.1    localhost
::1    localhost ip6-localhost ip6-loopback
fe00::0      ip6-localnet
ff00::0      ip6-mcastprefix
ff02::1      ip6-allnodes
ff02::2      ip6-allrouters
172.17.0.2   6accff938f54
#
```

Alpine est une distribution Linux légère, couramment utilisée avec Docker.

Cette fois, l'image alpine n'a pas été transféré du Docker Hub. Ce qui signifie qu'elle a déjà été sollicité sur ce serveur.
Docker Engine a créé un conteneur pour exécuter une instance de cette image. La commande s'est exécutée au sein du container. Puis ce dernier s'est arrêté.

```
# docker  images
REPOSITORY            TAG                 IMAGE ID            CREATED             SIZE
hello-world           latest              e38bc07ac18e        5 weeks ago         1.85kB
alpine                latest              3fd9065eaf02        4 months ago        4.15MB

# docker ps -a
CONTAINER ID  IMAGE        COMMAND           CREATED        STATUS
PORTS        NAMES
6accff938f54  alpine       "cat /etc/hosts"  5 minutes ago  Exited (0) 5 minutes ago
             inspiring_mirzakhani
0532c72d8863  hello-world  "/hello"          9 hours ago    Exited (0) 9 hours ago
             objective_elion
```

Le mode interactif : -it ou -i -t

```
# docker  run  -it  alpine       ou  docker  run  -i -t  alpine
/ # ls
bin    dev    etc    home   lib    media  mnt    proc   root   run    sbin   srv    sys
tmp    usr    var
/ # hostname
00df68932957
/ # exit                          pour arrêter et quitter le conteneur
#
```

```
# docker ps -a
CONTAINER ID  IMAGE        COMMAND           CREATED         STATUS
PORTS        NAMES
00df68932957  alpine        "/bin/sh"        33 seconds ago  Exited (0) 7 seconds ago
             quirky_golick
6accff938f54  alpine       "cat /etc/hosts"  5 minutes ago   Exited (0) 5 minutes ago
             inspiring_mirzakhani
0532c72d8863  hello-world  "/hello"          9 hours ago     Exited (0) 9 hours ago
             objective_elion
```

Même si chaque commande d'exécution de conteneur docker utilise la même image alpine, chaque exécution est un conteneur **séparé et isolé**. Chaque conteneur possède un système de fichiers distinct et s'exécute dans un espace de noms différent. Par défaut, un conteneur n'a aucun moyen d'interagir avec d'autres conteneurs, même ceux provenant de la même image.

Exécution en arrière plan : -d

```
# docker  run   nginx
      On perd la main sur le prompt,
      On est contraint de faire CTRL-C
```

```
# docker  run  -d  nginx
31f71151aed3fbd8797fc225a2477bfca82a9828913d1b313b6f9eebf8dc233d
#
```

Le prompt est récupéré et le conteneur fonctionne en arrière plan.

```
# docker  ps
CONTAINER ID  IMAGE   COMMAND    CREATED          STATUS         PORTS    NAMES
31f71151aed3  nginx  "nginx …"   11 seconds ago   Up 10 seconds  80/tcp   loving_dijkstra

# curl  http://172.17.0.2
<!DOCTYPE html>
<html>
<head>  <title>Welcome to nginx!</title>  </head>
<body>
<h1>Welcome to nginx!</h1>
  . . .
</body>
</html>

# docker  stop  loving_dijkstra  ;  docker  rm  loving_dijkstra
```

Nommer un conteneur :

L'option --name de la commande run permet de donner un nom spécifique à un container.

docker run --name=<nom_container> image

```
# docker  run  --name container1  ubuntu

# docker ps -a
CONTAINER ID  IMAGE   COMMAND      CREATED         STATUS                  PORTS  NAMES
fd7e85790271  ubuntu  "/bin/bash"  4 minutes ago   Exited (0) 17 seconds ago      container1
```

Renommer un conteneur : rename

```
# docker  rename  container1  intranet

# docker ps -a
CONTAINER ID    IMAGE    COMMAND       CREATED          STATUS          PORTS    NAMES
fd7e85790271    ubuntu   "/bin/bash"   40 minutes ago   Up 10 minutes            intranet
```

Transférer une image : pull

La sous commande pull transfère l'image du Docker Hub sans la démarrer.

```
# docker  images  centos
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE

# docker  pull  centos:latest
Trying to pull repository docker.io/library/centos ...
latest: Pulling from docker.io/library/centos
469cfcc7a4b3: Pull complete
Digest: sha256:989b936d56b1ace20ddf855a301741e52abca38286382cba7f44443210e96d16
Status: Downloaded newer image for docker.io/centos:latest

# docker  images  centos
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
docker.io/centos    latest              e934aafc2206        3 weeks ago         199 MB
```

Exploration d'un conteneur de l'image centos :

```
# docker  run  -it centos

[root@95462e4d6fed /]# pwd
/

[root@95462e4d6fed /]# ls
anaconda-post.log  bin  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run
sbin  srv  sys  tmp  usr  var

[root@95462e4d6fed /]# ps  -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 09:47 ?        00:00:00 /bin/bash
root        15     1  0 09:48 ?        00:00:00 ps -ef

[root@95462e4d6fed /]# cat  /etc/hosts
127.0.0.1    localhost
::1    localhost ip6-localhost ip6-loopback
fe00::0      ip6-localnet
ff00::0      ip6-mcastprefix
ff02::1      ip6-allnodes
ff02::2      ip6-allrouters
172.17.0.2   95462e4d6fed

[root@a95afd9b004c /]# df  -h
Filesystem                  Size  Used Avail Use% Mounted on
overlay                      27G  4.8G   23G  18% /
tmpfs                        64M     0   64M   0% /dev
tmpfs                       920M     0  920M   0% /sys/fs/cgroup
/dev/mapper/centos_mars-root   27G  4.8G   23G  18% /etc/hosts
shm                          64M     0   64M   0% /dev/shm
tmpfs                       920M     0  920M   0% /proc/scsi
tmpfs                       920M     0  920M   0% /sys/firmware

[root@95462e4d6fed /]# mkdir  /datas
[root@95462e4d6fed /]# rm  -rf  /home

[root@95462e4d6fed /]# ls
anaconda-post.log  bin  datas  dev  etc  lib  lib64  media  mnt  opt  proc  root  run
sbin  srv  sys  tmp  usr  var
```

D'une autre fenêtre terminale :

```
# docker  ps
CONTAINER ID  IMAGE   COMMAND      CREATED         STATUS         PORTS  NAMES
95462e4d6fed  centos  "/bin/bash"  3 minutes ago   Up 3 minutes          competent_goodall
```

```
# docker  ps  -a
CONTAINER ID   IMAGE   COMMAND      CREATED         STATUS         PORTS  NAMES
95462e4d6fed   centos  "/bin/bash"  4 minutes ago   Up 4 minutes          competent_goodall
```

Retour au conteneur :

```
[root@95462e4d6fed /]# exit                    pour arrêter et quitter le conteneur
exit
#
```

La création d'un conteneur sans le démarrer : create

La sous commande create crée le conteneur mais ne le démarre pas, contrairement à run.

```
[root@poste]# docker  create  --name container1  nginx
6b125c2bdfd18d15a3fe5526dd503c0763b2c61bfcb84398b2b1ae1843ca4c97

[root@poste]# docker  ps -a
CONTAINER ID   IMAGE   COMMAND                  CREATED          STATUS   PORTS NAMES
6b125c2bdfd1   nginx   "nginx -g 'daemon of…"   7 seconds ago    Created        container1

[root@poste]# docker  start  container1
container1

[root@poste]# docker  ps -a
CONTAINER ID  IMAGE COMMAND                CREATED         STATUS        PORTS  NAMES
6b125c2bdfd1  nginx "nginx -g 'daemon of…" 23 seconds ago  Up 4 seconds  80/tcp container1

[root@poste]# docker  ps
CONTAINER ID IMAGE COMMAND                CREATED         STATUS        PORTS  NAMES
6b125c2bdfd1 nginx "nginx -g 'daemon of…" 29 seconds ago  Up 10 seconds 80/tcp container1
```

Exécution d'une commande : exec

La sous commande exec permet d'exécuter une commande au sein d'un conteneur.

```
[root@poste]# docker  run  -d  --name container1  nginx
bba731072149f977176f76dcade73853bb47d125dc0b357327c2b3c6638fd19a

[root@poste]# docker  exec  -it container1  ls -l /usr/share/nginx/html
-rw-r--r--. 1 root root 537 Apr  9 16:01 50x.html
-rw-r--r--. 1 root root 612 Apr  9 16:01 index.html

[root@poste]# docker  exec  -it container1  /bin/bash
root@bba731072149:/# ls  -l  /usr/share/nginx/html
-rw-r--r--. 1 root root 537 Apr  9 16:01 50x.html
-rw-r--r--. 1 root root 612 Apr  9 16:01 index.html
root@bba731072149:/# exit
exit
```

Attacher et détacher un conteneur : attach ^P^Q

La sous commande attach permet de se connecter à un conteneur en cours de fonctionnement.

```
[root@poste]# docker  start  un_container
un_container
[root@poste]# docker  attach  un_container
[root@65fa2f995d85 /]# ls  /tmp
datas  fichier1
[root@65fa2f995d85 /]# exit
```

Pour se déconnecter d'un conteneur en cours de fonctionnement, on utilise la séquence de commandes suivantes : CTRL+P ET CTRL+Q.

Mise en œuvre de Docker en ligne de commandes

Arrêt et démarrage d'un conteneur

```
# docker  start  de47694487db
de47694487db
```

```
# docker  stop  de47694487db
de47694487db
```

```
# docker  restart  de47694487db
```

```
# docker  kill  de47694487db
# docker  kill  -s TERM  de47694487db
```

Arrêt et démarrage d'un conteneur

Syntaxe :

docker {start|stop|restart|kill} {ID_conteneur|Nom_conteneur}

La sous commande stop envoie le signal SIGTERM puis 10 secondes plus tard le signal SIGKILL.

```
# docker  start  de47694487db
de47694487db
```

```
# docker  stop  de47694487db
de47694487db
```

```
# docker  restart  de47694487db
```

```
# docker  kill  de47694487db
# docker  kill  -s TERM  de47694487db
```

Mise en œuvre de Docker en ligne de commandes

Suppression

```
# docker  rm  1e1cb8a74e4d

# docker  rm  inspiring_franklin

# docker  run  --rm  hello-world

# docker  rm  $(docker  ps  -aq)

# docker  rm  -f inspiring_franklin

# docker  rmi  docker.io/hello-world

# docker  system  prune

# docker  system  prune  --volumes
```

Suppression

Pour supprimer un conteneur :

docker rm {container_id | container_name} ...
Il ne doit pas être en cours d'utilisation.

```
# docker  rm  1e1cb8a74e4d
1e1cb8a74e4d
```

```
# docker  rm  inspiring_franklin
```

Suppression automatique à l'arrêt du conteneur : --rm

```
# docker  run  --rm  hello-world
```

Suppression de tous les conteneurs :

```
# docker  rm  $(docker  ps  -aq)
```

L'option -q renvoie le nom du conteneur. L'option -a renvoie tous les conteneurs même ceux qui ne sont pas en cours d'exécution.

```
# docker ps -aq
7df034cc1e9b
259ee8a6e1d5
. . .
```

Forcer la suppression d'un conteneur : -f

L'option -f permet d'arrêter et de supprimer un conteneur.

```
# docker  rm  -f  inspiring_franklin
```

Pour supprimer une image :

docker rmi image_1 image_2 ...

```
# docker  rmi  docker.io/hello-world
Untagged: docker.io/hello-world:latest
Untagged: docker.io/hello-
world@sha256:97ce6fa4b6cdc0790cda65fe7290b74cfebd9fa0c9b8c38e979330d547d22ce1
Deleted: sha256:f2a91732366c0332ccd7afd2a5c4ff2b9af81f549370f7a19acd460f87686bc7
Deleted: sha256:f999ae22f308fea973e5a25b57699b5daf6b0f1150ac2a5c2ea9d7fecee50fdf
```

Le nettoyage en profondeur :

docker system prune mais ne supprime pas les volumes.

```
# docker  system  prune
WARNING! This will remove:
        - all stopped containers
        - all networks not used by at least one container
        - all dangling images
        - all build cache
Are you sure you want to continue? [y/N] y
Deleted Containers:
75ff9d8a4de46c767e790010d9367c304e75c30c51bba20b40b4fbe4e287b71e
b81531e5c0636f2e3c890fce516781368776039a77504716055ac4d906981301
911e115627530905ad112e062da2d926c3d1709dd6a081191553d790a6925b3a
120d42ea14dc7a5edcd68e5089c9775e41798d41884c583506f25f371979622b

Total reclaimed space: 419.4 MB
```

A la fin de l'exécution de la commande, le système nous indique la place récupérée.

Comme précédemment mais également avec la suppression des volumes :

```
# docker  system  prune  --volumes
WARNING! This will remove:
        - all stopped containers
        - all networks not used by at least one container
        - all volumes not used by at least one container
        - all dangling images
        - all build cache
Are you sure you want to continue? [y/N] y
Deleted Volumes:
9f947dab7e74d29f9ca4e846e8961e3d93b0da0a309b78c2859c1dc36b7e3d01
f9a885600f12715ab3699682fa666b969a22e63f4597cd182c80292ddc11d3e8
portainer_data

Total reclaimed space: 512.24 MB
```

Mise en œuvre de Docker en ligne de commandes

Détail d'un conteneur : inspect

```
[root@poste]# docker  inspect  container1
[
    {
        "Id":
"953be4a0ede22836ad84d3967dbba55f8247975fe193f0a07508631ae5338298",
        "Created": "2018-05-04T21:47:03.504444074Z",
        "Path": "nginx",
        "Args": [
            "-g",
            "daemon off;"
        ],
        "State": {
            "Status": "running",

             .  .  .
```

Détail d'un conteneur : inspect

```
[root@poste]# docker  run  -d  --name container1  nginx
953be4a0ede22836ad84d3967dbba55f8247975fe193f0a07508631ae5338298
```

```
[root@poste]# docker  inspect  container1
[
    {
        "Id": "953be4a0ede22836ad84d3967dbba55f8247975fe193f0a07508631ae5338298",
        "Created": "2018-05-04T21:47:03.504444074Z",
        "Path": "nginx",
        "Args": [
            "-g",
            "daemon off;"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 5156,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2018-05-04T21:47:05.558617238Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
        "Image":
"sha256:ae513a47849c895a155ddfb868d6ba247f60240ec8495482eca74c4a2c13a881",
        "ResolvConfPath":
"/var/lib/docker/containers/953be4a0ede22836ad84d3967dbba55f8247975fe193f0a07508631ae5338
298/resolv.conf",
```

```
        "HostnamePath":
"/var/lib/docker/containers/953be4a0ede22836ad84d3967dbba55f8247975fe193f0a07508631ae5338
298/hostname",
        "HostsPath":
"/var/lib/docker/containers/953be4a0ede22836ad84d3967dbba55f8247975fe193f0a07508631ae5338
298/hosts",
        "LogPath":
"/var/lib/docker/containers/953be4a0ede22836ad84d3967dbba55f8247975fe193f0a07508631ae5338
298/953be4a0ede22836ad84d3967dbba55f8247975fe193f0a07508631ae5338298-json.log",
        "Name": "/container1",

. . .
        "Config": {
            "Hostname": "953be4a0ede2",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "ExposedPorts": {
                "80/tcp": {}
            },
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
                "NGINX_VERSION=1.13.12-1~stretch",
                "NJS_VERSION=1.13.12.0.2.0-1~stretch"
            ],
        },
        "NetworkSettings": {
            "Bridge": "",
            "SandboxID":
"9b4709e737a876ccd1b366a05280afe7c02925c3828f4e16c3070cb7a1a48377",
            "HairpinMode": false,
            "LinkLocalIPv6Address": "",
            "LinkLocalIPv6PrefixLen": 0,
            "Ports": {
                "80/tcp": null
            },
            "SandboxKey": "/var/run/docker/netns/9b4709e737a8",
            "SecondaryIPAddresses": null,
            "SecondaryIPv6Addresses": null,
            "EndpointID":
"202361eaaa3a59951ec9fd78eaf764a8cf9f8a54cf0c7710506c26557077c74c",
            "Gateway": "172.17.0.1",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "IPAddress": "172.17.0.2",
            "MacAddress": "02:42:ac:11:00:02",
            "Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "NetworkID":
"f930dda5a14004e6d2673dbfdd219738e0c6648f9745ec84d5d64d91467c149d",
                    "EndpointID":
"202361eaaa3a59951ec9fd78eaf764a8cf9f8a54cf0c7710506c26557077c74c",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.2",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:ac:11:00:02",
                    "DriverOpts": null
                }
            }
        }
    }
]
```

Mise en œuvre de Docker en ligne de commandes

Les différences entre le conteneur et l'image : diff

```
# docker  diff  competent_goodall
```

```
# docker  diff  95462e4d6fed
A /datas                    Add             Ajout
D /home                     Delete          Suppression
C /root                     Change          Modification
A /root/.bash_history
C /run
A /run/secrets
```

Les différences entre le conteneur et l'image : diff

```
# docker  run  -it centos
[root@95462e4d6fed /]# mkdir  /datas
[root@95462e4d6fed /]# rm  -rf  /home
[root@95462e4d6fed /]# ls
anaconda-post.log  bin  datas  dev  etc  lib  lib64  media  mnt  opt  proc  root  run
sbin  srv  sys  tmp  usr  var
[root@95462e4d6fed /]# exit

# docker  ps  -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
95462e4d6fed        centos              "/bin/bash"         4 minutes ago       Up 4
minutes                                      competent_goodall
a7268862874a        centos              "/bin/bash"         4 minutes ago       Exited
(0) 4 minutes ago
```

Pour connaître les différences entre le conteneur et l'image source :

```
# docker  diff  competent_goodall
```

ou

```
# docker  diff  95462e4d6fed
A /datas                    Add         Ajout
D /home                     Delete      Suppression
C /root                     Change      Modification
A /root/.bash_history
C /run
A /run/secrets
```

Mise en œuvre de Docker en ligne de commandes

Les logs, events, top et stats

```
# docker  logs  container1
# docker  logs  container1  --since  5m
# docker  logs  container1  --until  5m
# docker  logs  container1  --since  10m   -t
# docker  logs  container1  --tail 2
```

```
# docker  events

# docker  top  container1

# docker  stats  container1  container2  quirky_golick
```

Les logs, events, top et stats

Les logs

```
[root@poste]# docker  logs  container1
172.17.0.1 - - [04/May/2018:20:57:45 +0000] "GET / HTTP/1.1" 200 77 "-" "Mozilla/5.0
(X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0" "-"

. . .
```

Obtenir des logs récents avec since pour "moins de" :

```
[root@poste]# docker  logs  container1  --since  5m
172.17.0.1 - - [04/May/2018:21:04:43 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11;
Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0" "-"
172.17.0.1 - - [04/May/2018:21:04:44 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11;
Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0" "-"
```

Obtenir des logs plus anciens avec until pour "plus de" :

```
[root@poste]# docker  logs  container1  --until  5m
172.17.0.1 - - [04/May/2018:20:57:45 +0000] "GET / HTTP/1.1" 200 77 "-" "Mozilla/5.0
(X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0" "-"
172.17.0.1 - - [04/May/2018:21:02:43 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11;
Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0" "-"
```

avec : 5m pour 5 minutes
3h pour 3 heures
etc ...

Ajouter en début de ligne l'horodatage :

```
[root@poste]# docker  logs  container1  --since  10m   -t
2018-05-04T21:02:43.769162628Z 172.17.0.1 - - [04/May/2018:21:02:43 +0000] "GET /
HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101
Firefox/52.0" "-"
```

Afficher les N dernières lignes :

```
[root@poste]# docker  logs  container1  --tail 2
172.17.0.1 - - [04/May/2018:21:04:43 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11;
Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0" "-"
172.17.0.1 - - [04/May/2018:21:04:44 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11;
Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0" "-"
```

Option -t : pour ajouter l'horodatage.

Option -f : pour laisser le flux ouvert.

Les events

La sous commande events permet de suivre les événements au sein de Docker.

Avec cette commande, on perd la main sur la ligne de commande car elle affiche les événements en temps réel. Ce ne sont pas des logs.

```
[root@poste]# docker  events
2018-05-06T11:53:30.800611520+02:00 network connect
50f9f32f6f31667f7c7368b27b38d1058bed221a6a8e3e5672a4fc922cbfcf47
(container=50f3deb5589a62cedb7823d13af144ccba66fe59c2d03e24af496dbb477e3081, name=bridge,
type=bridge)
2018-05-06T11:53:31.590302766+02:00 container start
50f3deb5589a62cedb7823d13af144ccba66fe59c2d03e24af496dbb477e3081 (image=nginx,
maintainer=NGINX Docker Maintainers <docker-maint@nginx.com>, name=container1)
2018-05-06T11:54:24.513723202+02:00 container kill
50f3deb5589a62cedb7823d13af144ccba66fe59c2d03e24af496dbb477e3081 (image=nginx,
maintainer=NGINX Docker Maintainers <docker-maint@nginx.com>, name=container1, signal=15)
2018-05-06T11:54:24.734281020+02:00 container die
50f3deb5589a62cedb7823d13af144ccba66fe59c2d03e24af496dbb477e3081 (exitCode=0,
image=nginx, maintainer=NGINX Docker Maintainers <docker-maint@nginx.com>,
name=container1)
2018-05-06T11:54:25.063942210+02:00 network disconnect
50f9f32f6f31667f7c7368b27b38d1058bed221a6a8e3e5672a4fc922cbfcf47
(container=50f3deb5589a62cedb7823d13af144ccba66fe59c2d03e24af496dbb477e3081, name=bridge,
type=bridge)
2018-05-06T11:54:25.141121399+02:00 container stop
50f3deb5589a62cedb7823d13af144ccba66fe59c2d03e24af496dbb477e3081 (image=nginx,
maintainer=NGINX Docker Maintainers <docker-maint@nginx.com>, name=container1)
```

Au sein d'une autre fenêtre terminal, le conteneur 'container1' d'une image nginx a été démarré puis arrêté.

La sous commande top

La sous commande top affiche les processus d'un conteneur.

```
# docker  run  -d  --name container1  nginx
b461da8a3f636a56a052192cfe6e96572f65573bf637fc1882220f4174b71414

# docker  top  container1
UID    PID    PPID   C   STIME  TTY   TIME      CMD
root   4447   4433   0   22:40  pts/0 00:00:00  nginx: master process nginx -g daemon off
101    4477   4447   0   22:40  pts/0 00:00:00  nginx: worker process
```

On peut utiliser les options de la commande ps (exemple -aux ou -ef) :

docker top container1 -aux

Afficher les statistiques sur les conteneurs

La sous commande stats affiche les ressources utilisées par conteneur.

```
# docker  ps  -a
CONTAINER ID  IMAGE   COMMAND       CREATED          STATUS          PORTS      NAMES
2aea05e5f485  nginx   "nginx -g …"  4 minutes ago    Up 4 minutes    80/tcp     container2
b461da8a3f63  nginx   "nginx -g…"   8 minutes ago    Up 8 minutes    80/tcp     container1
00df68932957  alpine  "/bin/sh"     26 hours ago     Exited (0) 26 hours ago    quirky_gol
```

Affiche les ressources de tous les conteneurs :

```
# docker  stats
CONTAINER ID   NAME        CPU %   MEM USAGE / LIMIT    MEM %    NET I/O          BLOCK I/O      PIDS
b461da8a3f63   container1  0.46%   1.363MiB / 1.796GiB  0.07%    2.66kB / 2.32kB   1.99MB / 0B   2
2aea05e5f485   container2  0.23%   1.355MiB / 1.796GiB  0.07%    2.36kB / 2.08kB   0B / 0B       2
00df68932957   quirky_gol  0.00%   0B / 0B              0.00%    0B / 0B           0B / 0B       0
```

Faire CTRL-C pour reprendre la main.

Remarque : le conteneur « quirky_gol » est arrêté.

On peut filtrer par conteneurs :

```
# docker  stats  container1  container2
CONTAINER ID   NAME        CPU %   MEM USAGE / LIMIT    MEM %    NET I/O          BLOCK I/O      PIDS
b461da8a3f63   container1  0.46%   1.363MiB / 1.796GiB  0.07%    2.66kB / 2.32kB   1.99MB / 0B   2
2aea05e5f485   container2  0.23%   1.355MiB / 1.796GiB  0.07%    2.36kB / 2.08kB   0B / 0B       2
```

Mise en œuvre de Docker en ligne de commandes

Les variables

```
# docker  run  -i  -t  -e=HOSTNAME  debian

# docker  run  -i  -t  -e=VILLE=Paris  debian

# docker  run  -i  -t  --env-file=fic_conf  debian

# docker  run  -i  -t  -h deb_theo  debian
```

Les variables

Injecter une variable existante dans un conteneur

docker run -e=<nom_variable> image

```
poste1# docker  run  -i  -t  -e=HOSTNAME  debian
root@545403db3f38:/# env
HOSTNAME=poste1
PWD=/
HOME=/root
TERM=xterm
SHLVL=1
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
_=/usr/bin/env
```

Injecter une nouvelle variable

A partir de la ligne de commande :

docker run -e <nom_variable=valeur> image
docker run -e=<nom_variable=valeur> image

```
# docker  run  -i  -t  -e VILLE=Paris  debian
root@e46f322c1064:/# echo $VILLE
Paris
```

A partir d'un fichier :

docker run --env-file=fichier_configuration image

```
# cat fic_conf
VILLE=Paris
prenom=theo
HOSTNAME=container1
```

```
# docker run -i -t --env-file=fic_conf debian
root@9277fd1ac312:/# echo $VILLE $prenom $HOSTNAME
Paris theo container1
```

Modifier le hostname du container lors du lancement

docker run -h <nom_machine> image

```
# docker run -i -t -h deb_theo debian
root@deb_theo:/# echo $HOSTNAME
deb_theo
```

Mise en œuvre de Docker en ligne de commandes

La copie de fichiers

```
# docker  cp  fichier1  container1:/tmp

# docker  cp  container1:/tmp/fichier1  fichier2
```

La copie de fichiers

La sous commande cp permet de copier un fichier entre un conteneur et le système local.

```
[root@poste]# ls
fichier1

[root@poste]# docker  cp  fichier1  container1:/tmp

[root@poste]# docker  exec  container1  ls /tmp
fichier1
```

```
[root@poste]# ls

[root@poste]# docker  cp  container1:/tmp/fichier1  fichier2

[root@poste]# ls
fichier2
```

Mise en œuvre de Docker en ligne de commandes

pause, unpause et wait

```
[root@poste]# docker  pause  container1
container1
[root@poste]# docker  ps
CONTAINER ID  IMAGE  COMMAND      CREATED       STATUS                   PORTS  NAMES
50f3deb5589a  nginx  "nginx -g…"  2 weeks ago   Up 11 seconds (Paused)   80/tcp
container1
[root@poste]# docker  unpause  container1



[root@poste]# docker   stop   container1
[root@poste]# docker   wait   container1
```

pause, unpause et wait

La sous commande pause permet de freezer un conteneur. Il est à noter qu'il n'est pas arrêté mais suspendu. Inversement, pour défreezer un conteneur, on utilise unpause.

```
[root@poste]# docker  start  container1
[root@poste]# docker  pause  container1
container1
[root@poste]# docker  ps
CONTAINER ID  IMAGE  COMMAND      CREATED       STATUS                   PORTS  NAMES
50f3deb5589a  nginx  "nginx -g…"  2 weeks ago   Up 11 seconds (Paused)   80/tcp container1
[root@poste]# docker  unpause  container1
container1
[root@poste]# docker  ps
CONTAINER ID  IMAGE   COMMAND      CREATED         STATUS           PORTS  NAMES
50f3deb5589a  nginx   "nginx -g…"  2 weeks ago     Up 25 seconds    80/tcp container1
```

Le mot clé wait permet de s'assurer qu'un conteneur est bien arrêté avant de continuer à exécuter des commandes.

```
# docker  stop  container1
# docker  wait  container1
```

Le stop rend la main immédiatement alors que le conteneur est en cours d'arrêt. Lorsque l'on tape ensuite la commande avec wait, on récupérera le prompt que lorsque le conteneur sera définitivement arrêté.
Ceci peut être utile pour des conteneurs qui sont long à s'arrêter.

Mise en œuvre de Docker en ligne de commandes

Redémarrage automatique d'un conteneur

```
# docker  run  -d  --name web  --restart on-failure  nginx
```

no

on-failure

unless-stopped

always

Redémarrage automatique d'un conteneur

On peut spécifier un redémarrage ou pas d'un conteneur avec le mot clé restart.

```
# docker  run  -d  --restart  unless-stopped  mon_image
```

Les arguments possibles :

no :	c'est la valeur par défaut.
on-failure :	redémarrer le conteneur s'il s'est arrêté avec un code de retour différent de 0 (erreur).
unless-stopped :	toujours redémarrer le conteneur, sauf s'il a été arrêté explicitement.
always :	démarrer le conteneur au démarrage du service docker.

Exemple :

```
[root@poste]# docker  run  -d  --name web  --restart on-failure  nginx
c90db4747eb7097c867508ecf89d3a58466900f7549734c02208ae5644d50cd0

[root@poste]# docker  ps  -a
CONTAINER ID  IMAGE  COMMAND      CREATED         STATUS          PORTS   NAMES
57d1a3968d6b  nginx  "nginx -g…"  4 minutes ago   Up 4 minutes    80/tcp  web

[root@poste]# pkill  -9  nginx

[root@poste]# docker  ps  -a
CONTAINER ID  IMAGE  COMMAND      CREATED         STATUS          PORTS   NAMES
57d1a3968d6b  nginx  "nginx -g…"  4 minutes ago   Up 2 seconds    80/tcp  web
```

Mise en œuvre de Docker en ligne de commandes

Démarrer un conteneur au boot du serveur

```
# docker  run  -d  --name webyes  --restart=always  nginx
```

Scripts de démarrage standards

Supervisor

Démarrer un conteneur au boot du serveur

1/ On utilise --restart always

docker run -d --name container1 --restart=always mon_image

```
[root@poste]# docker run -d --name webno  nginx
[root@poste]# docker run -d --name webyes --restart=always  nginx
[root@poste]# docker ps -a       (webno et webyes fonctionnent)
[root@poste]# shutdown -r now

[root@poste]# docker ps -a
CONTAINER ID  IMAGE COMMAND      CREATED        STATUS                    PORTS   NAMES
60b90448cc7c  nginx "nginx -gf…" 4 minutes ago  Up 2 minutes              80/tcp  webyes
793090984fd8  nginx  "nginx -g…" 5 minutes ago  Exited (0) 3 minutes ago          webno
```

2/ On intègre un service standard au système hôte. Il sollicite un script de démarrage (et d'arrêt) de vos conteneurs (docker start/stop).

3/ Vous pouvez exploiter « supervisor » : https://github.com/dockerfile/supervisor

Supervisor est un système client/serveur qui permet à ses utilisateurs de surveiller et de contrôler un certain nombre de processus sur des systèmes d'exploitation de type Linux. Il est destiné à être utilisé pour contrôler les processus liés à un projet ou un client, et est destiné à démarrer comme tout autre programme au moment du boot du système.

Mise en œuvre de Docker en ligne de commandes

Conteneur et stockage

- -v rep_du_hôte:rep_du_container
- --volume rep_du_hôte:rep_du_container

```
# docker  run  -d  --name container1  \
          -v  /root/Docker/nginx/site:/usr/share/nginx/html nginx

# docker  run  -it  --name un_container  \
          -v $(pwd):/tmp  -v $(pwd):/home  centos

# docker  run  -d  --name un_container  \
          -v /usr/share/nginx/html  nginx
```

Conteneur et stockage

Pour pérenniser les données d'un conteneur, il est possible de lier un répertoire du poste local à un répertoire du conteneur.

Syntaxe pour l'utilisation des volumes :

-v rep_du_hôte:rep_du_container
ou
--volume rep_du_hôte:rep_du_container

```
# docker  run  -d  --name container1  \
         -v  /root/Docker/nginx/site:/usr/share/nginx/html nginx
```

Le site qui apparaît correspond aux pages localisées sur le poste hôte au sein du répertoire :
/root/Docker/nginx/site

Démonstration :

```
[root@poste]# ls
datas  nginx

[root@poste]# docker  run  -it  --name un_container  \
                           -v $(pwd):/tmp  -v $(pwd):/home  centos

[root@65fa2f995d85 /]# ls  /tmp
datas  nginx
```

```
[root@65fa2f995d85 /]# cat  /etc/hosts
127.0.0.1    localhost
. . .
172.17.0.2   65fa2f995d85
[root@65fa2f995d85 /]# cat  /etc/hosts  >  /tmp/fichier1
[root@65fa2f995d85 /]# ls  /tmp
datas  fichier1  nginx
[root@65fa2f995d85 /]# useradd  -m  -d /home/user2  user2
[root@65fa2f995d85 /]# ls -a /home/user2
.  ..  .bash_logout  .bash_profile  .bashrc
[root@65fa2f995d85 /]# exit

[root@poste]# ls
datas  fichier1  nginx  user2
[root@poste]# ls  -a  user2
.  ..  .bash_logout  .bash_profile  .bashrc
[root@poste]# cat  fichier1
127.0.0.1    localhost
. . .
172.17.0.2   65fa2f995d85
```

```
[root@poste]# docker  rm  un_container
[root@poste]# docker  run  -it  --name autre_container  \
                           -v $(pwd):/tmp  -v $(pwd):/home  centos
[root@e46f322c1064 /]# ls  /tmp
datas  fichier1  nginx  user2
[root@e46f322c1064 /]# exit

[root@poste]# docker  start  -i  autre_container
[root@e46f322c1064 /]# ls  /tmp
datas  fichier1  nginx  user2
[root@e46f322c1064 /]# cat  /etc/passwd
root:x:0:0:root:/root:/bin/bash
. . .
user2:x:1000:1000::/home/user2:/bin/bash
[root@e46f322c1064 /]# userdel  -r  user2
[root@e46f322c1064 /]# ls  /home
datas  fichier1  nginx
[root@e46f322c1064 /]# rm  /tmp/fichier1
[root@e46f322c1064 /]# exit

[root@poste]# ls
datas  nginx
```

```
[root@poste]# docker  inspect  un_container
        "Mounts": [
            {
                "Type": "bind",
                "Source": "/root",
                "Destination": "/tmp",
                "Mode": "",
                "RW": true,
                "Propagation": "rprivate"
            },
            {
                "Type": "bind",
                "Source": "/root",
                "Destination": "/home",
                "Mode": "",
                "RW": true,
                "Propagation": "rprivate"
            }
        ],
```

Autre syntaxe :

-v rep_du_container
Création implicite d'un volume Docker au sein du répertoire /var/lib/docker/volumes qui sera associé au répertoire du conteneur spécifié par l'option -v.

```
[root@poste]# docker  run  -d  --name un_container  \
                           -v /usr/share/nginx/html  nginx
5255e1c923655304566494328702edf721460a31d82b01294d4dab5d2a991120
```

```
[root@poste]# docker  inspect  un_container
        "Mounts": [
            {
                "Type": "volume",
                "Name":
"44bcfbee43f5684a34dc7a614093d298f27dbb59a2fd704fb5a302cb72831055",
                "Source":
"/var/lib/docker/volumes/44bcfbee43f5684a34dc7a614093d298f27dbb59a2fd704fb5a302cb72831055
/_data",
                "Destination": "/usr/share/nginx/html",
                "Driver": "local",
                "Mode": "",
                "RW": true,
                "Propagation": ""
            }
        ],
```

```
[root@poste]# ls  -l
/var/lib/docker/volumes/44bcfbee43f5684a34dc7a614093d298f27dbb59a2fd704fb
5a302cb72831055/_data/
total 8
-rw-r--r--. 1 root root 537  9 avril 18:01 50x.html
-rw-r--r--. 1 root root 612  9 avril 18:01 index.html
[root@poste]#
```

Mise en œuvre de Docker en ligne de commandes

Les ports réseaux : publication

```
# docker  run  -itd  -p 90:80  --name=container1  nginx
```

```
# docker  run  -d  --name container1  -p 90:80  -p 93:443  nginx
```

```
# docker  port  container1
443/tcp -> 0.0.0.0:93
80/tcp -> 0.0.0.0:90
```

```
[root@poste]#  docker  run  -itd  -P  nginx
```

Les ports réseaux : publication

Pour associer un port (mapper) du poste local avec un port du conteneur :

 # docker run -itd -p port_hôte:port_docker ... image

Utilisation standard :

```
[root@poste]# docker run -itd --name=container1 nginx
Unable to find image 'nginx:latest' locally
Trying to pull repository docker.io/library/nginx ...
latest: Pulling from docker.io/library/nginx
f2aa67a397c4: Pull complete
3c091c23e29d: Pull complete
4a99993b8636: Pull complete
Digest: sha256:0fb320e2a1b1620b4905facb3447e3d84ad36da0b2c8aa8fe3a5a81d1187b884
Status: Downloaded newer image for docker.io/nginx:latest
ce06d4a02c04bf7bd979c3d328aef1537a9192d3d8fc202785991bcc0dc23e0a

[root@poste]# docker ps -a
CONTAINER ID  IMAGE COMMAND      CREATED        STATUS              PORTS     NAMES
ce06d4a02c04  nginx "nginx ..."  4 minutes ago  Up About a minute   80/tcp    container1

[root@poste]# docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "d846b80431d3cadb9df03471cb44ef0a7e5c45baa25ccb05807ec92d132f1d1e",
        "Created": "2018-05-04T09:33:03.087493099+02:00",
        "Scope": "local",
        "Driver": "bridge",
```

```
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Containers": {
            "ce06d4a02c04bf7bd979c3d328aef1537a9192d3d8fc202785991bcc0dc23e0a": {
                "Name": "container1",
                "EndpointID":
"2bc2a15f8ce361a836fc876b7c81e8f34f6140ae26969b6a2ad6a4f410adbdb6",
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
            "com.docker.network.bridge.enable_icc": "true",
            "com.docker.network.bridge.enable_ip_masquerade": "true",
            "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
            "com.docker.network.bridge.name": "docker0",
            "com.docker.network.driver.mtu": "1500"
        },
        "Labels": {}
    }
]
```

nginx fonctionne sur le port 80.
du poste via un browser avec http://172.17.0.2 on obtient la page de garde nginx.
du poste via un browser avec http://localhost ça ne marche pas.

```
[root@poste]# docker stop container1
ce06d4a02c04bf7bd979c3d328aef1537a9192d3d8fc202785991bcc0dc23e0a
```

nginx ne répond plus !

```
[root@poste]# docker start container1
ce06d4a02c04bf7bd979c3d328aef1537a9192d3d8fc202785991bcc0dc23e0a
```

nginx fonctionne.

```
[root@poste]# docker stop container1
ce06d4a02c04bf7bd979c3d328aef1537a9192d3d8fc202785991bcc0dc23e0a
```

nginx ne répond plus !

```
[root@poste]# docker rm container1
container1
```

Mappage d'un port :

```
[root@poste]# docker  run  -itd  -p 90:80  --name=container1  nginx

[root@poste]# docker  ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS              PORTS               NAMES
6cd3945bc31b        nginx               "nginx -g 'daemon ..."   44 seconds ago
Up 43 seconds       0.0.0.0:90->80/tcp  container1

[root@poste]# docker  inspect  container1
            "PortBindings": {
                "80/tcp": [
                    {
                        "HostIp": "",
                        "HostPort": "90"
                    }
                ]
            },
```

du poste via un browser avec http://172.17.0.2 on obtient la page de garde nginx : port 80
du poste via un browser avec http://localhost:90 on obtient la page de garde nginx : port 90

```
[root@poste]# docker  stop  container1
[root@poste]# docker  rm  container1
```

Remarque : sans l'option d (le d pour mode le déconnecté).

```
[root@poste]# docker  run  -it  --name=container1  nginx
                  on n'a plus la main, on n'a pas récupéré le prompt.
                  Pendant ce temps nginx fonctionne (http://172.17.0.2)

                  CTRL-C
[root@poste]#     nginx ne marche plus ! (mais bon, s'il est arrêté : c'est normal!!)
```

Autre exemple :

```
[root@poste]# docker run -d --name container1  -p 90:80 -p 93:443  nginx
```

La sous commande port permet de lister les ports exposés d'un conteneur :

```
[root@poste]# docker  port  container1
443/tcp -> 0.0.0.0:93
80/tcp -> 0.0.0.0:90
```

L'option P :

L'option -P est utilisée pour mapper automatiquement n'importe quel port réseau du hôte Docker à un port du conteneur (une valeur aléatoire dans une plage des ports éphémères).

```
[root@poste]#  docker  run  -itd   -P   nginx
```

```
[root@poste]# docker  ps
CONTAINER ID        IMAGE           COMMAND                  CREATED         STATUS
PORTS                   NAMES
7056a3881be3        nginx          "nginx -g 'daemon of…"   6 seconds ago    Up 5 seconds
0.0.0.0:32768->80/tcp   happy_ardinghelli
```

Mise en œuvre de Docker en ligne de commandes

La gestion des ressources

- Mémoire
- CPU
- I/O disque

La gestion des ressources

La gestion de la mémoire :

Par défaut, un conteneur utilise autant de mémoire que nécessaire (RAM et swap).

-m :

Limiter la taille de la RAM à 500M pour un conteneur, à pour effet de plafonner la swap à 500M.
Par défaut, le conteneur peut utiliser la même quantité de swap que ce qui a été fixé pour la RAM.

```
# docker run -dt -m 500M --name cent1 centos
# docker inspect cent1 | grep -i mem
           "Memory": 524288000,              taille RAM en octets = 500M
           "MemorySwap": 1048576000,         taille RAM+swap
```

--memory-swap :

Pour limiter la taille de la RAM à 500M, mais désactiver la limitation de la swap.
Il sera donc possible d'utiliser autant de swap que disponible.

```
# docker run -it -m 500M --memory-swap -1 centos
```

En utilisant les deux paramètres (-m et --memory-swap), on peut définir une quantité de swap spécifique.
Pour limiter la taille de la RAM à 500M et la swap à 1,5G :

```
# docker run -it -m 500M --memory-swap 2G centos
```

--memory-reservation :

La réservation est une limite soft pour assurer un meilleur partage de la mémoire avec le hôte (et les autres conteneurs). Lorsque la mémoire est fortement sollicitée sur le système hôte, celui-ci peut contraindre les conteneurs à limiter leur consommation à leur limite de réservation.
Avec l'exemple, lorsque le conteneur consomme plus de 200M de mémoire (et moins de 500M), la prochaine récupération de mémoire système tente de réduire la mémoire du conteneur en dessous de 200M.
Par défaut (sans réservation), la réservation de mémoire est la même que la limite de mémoire.

```
# docker run -dt --name cent1 -m 500M  --memory-reservation 200M   centos
# docker inspect  cent1  |  grep  -i mem
          "Memory": 524288000,
          "MemoryReservation": 209715200,
          "MemorySwap": 1048576000,
```

On peut fixer une réservation sans limiter la mémoire :

```
# docker  run  -dt  --name cent1  --memory-reservation 200M  centos
# docker  inspect  cent1  |  grep  -i mem
            "Memory": 0,
            "MemoryReservation": 209715200,
            "MemorySwap": 0,
```

La gestion de la CPU :

Limitation du nombre de cpus utilisé par un conteneur : **--cpuset-cpus**

```
# docker  run  -it  --cpuset-cpus="0"  centos
     travaille qu'avec la cpu 0

# docker  run  -it  --cpuset-cpus="0-2"  centos
     travaille avec les cpus 0, 1 et 2

# docker  run  -it  --cpuset-cpus="0,2" centos
     travaille avec les cpus 0 et 2
```

Limitation du pourcentage cpu utilisable : **--cpus**

Pour une limitation à 0,5 cpu :

```
# docker  run  -ti  --cpus="0.5"  --rm  centos
[root@5fe2221e631b  /]# yes > /dev/null &
[root@5fe2221e631b  /]# yes > /dev/null &
[root@5fe2221e631b  /]# top
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
   16 root      20   0    4320    348    276 R  26.2  0.0   0:11.31 yes
   14 root      20   0    4320    352    276 R  23.9  0.0   0:19.96 yes
```

Pour une limitation à 1,5 cpu :

```
# docker  run  -ti  --cpus="1.5"  --rm  centos
[root@e8906aed974f /]# yes > /dev/null &
[root@e8906aed974f /]# yes > /dev/null &
[root@e8906aed974f /]# top
top - 14:13:46 up  2:58,  0 users,  load average: 1.37, 1.39, 1.51
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
   13 root      20   0    4320    348    276 R  75.4  0.0   0:04.38 yes
   15 root      20   0    4320    352    276 R  74.8  0.0   0:00.59 yes
```

Le partage de ressource cpu entre conteneurs : **--cpu-shares**

On peut positionner un share (poids) sur un conteneur pour favoriser (ou pénaliser) un container par rapports aux autres. Le share sera pris en compte que s'il y a une contention cpu sur le hôte. La valeur par défaut est 1024.

Avec contention sur le hôte :

cas1 : conteneur1 avec share à 512, conteneur2 à 512
conteneur1: 50%, conteneur2: 50%

cas2 : conteneur1 avec share à 512, conteneur2 à 512, conteneur3 avec share à 1024
conteneur1: 25%, conteneur2: 25% et conteneur3: 50%

La gestion des I/O disques :

Limitation de la bande passante en écriture (byte) : **--device-write-bps**
Ce paramètre limite la bande passante en écriture en bytes par seconde.

```
# docker run -it --rm  --device-write-bps /dev/sda:1mb centos
[root@4cdb1ca7268f /]# time dd if=/dev/zero of=/tmp/fic  \
                            bs=1M count=10 oflag=direct
10485760 bytes (10 MB) copied, 10.0185 s, 1.0 MB/s
```

Remarque, sans --device-write-bps :

```
[root@4bb1281af0cd /]# time dd if=/dev/zero of=/tmp/fic  \
                            bs=1M count=10  oflag=direct
10485760 bytes (10 MB) copied, 0.0274967 s, 381 MB/s
```

Limitation de la bande passante en écriture (IO) : **--device-write-iops**
Ce paramètre limite la bande passante en écriture en nombre d'IO par seconde.

Limitation de la bande passante en lecture (byte) : **--device-read-bps**
Ce paramètre limite la bande passante en lecture en bytes par seconde.

Limitation de la bande passante en lecture (IO) : **--device-read-iops**
Ce paramètre limite la bande passante en lecture en nombre d'IO par seconde.

Le partage de la bande passante I/O entre conteneurs : **--blkio-weight**
on peut positionner un share (poids) sur un conteneur pour favoriser (ou pénaliser) un container par rapport aux autres. Le share ne sera pris en compte que s'il y a une contention I/O sur le hôte. La valeur par défaut est 500, on fixe une valeur entre 10 et 1000.

Opérations lancées en même temps sur 3 conteneurs :

```
# docker run -it --rm  --blkio-weight 100  centos
[root@4bb1281af0cd /]# time dd if=/dev/zero of=/tmp/fic  \
                            bs=1M count=1024 oflag=direct
1073741824 bytes (1.1 GB) copied, 27.2039 s, 39.5 MB/s
# docker run -it --rm  --blkio-weight 100  centos
[root@03f62c89926b /]# time dd if=/dev/zero of=/tmp/fic  \
                            bs=1M count=1024 oflag=direct
1073741824 bytes (1.1 GB) copied, 26.0709 s, 41.2 MB/s
# docker run -it --rm  --blkio-weight 800  centos
[root@b2a586d1396e /]# time dd if=/dev/zero of=/tmp/fic  \
                            bs=1M count=1024 oflag=direct
1073741824 bytes (1.1 GB) copied, 14.2063 s, 75.6 MB/s
```

Le partage de la bande passante I/O entre conteneurs pour un device : --blkio-weight-device
Comme précédemment mais pour un périphérique donné.

```
# docker run -it --rm  --blkio-weight-device /dev/sda:800  centos
```

Notes

Création d'une image personnalisée

Dans ce chapitre nous allons étudier la création d'images personnalisées.

Création d'une image personnalisée

- Produire une image de l'état d'un conteneur - commit
- Le fichier Dockerfile
- Le Dockerfile - ENTRYPOINT
- Le Dockerfile – les mots clefs
- Le Dockerfile – les bonnes pratiques
- Sauvegarde et restauration d'une image

Création d'une image personnalisée

Produire une image de l'état d'un conteneur - commit

- docker commit nom_container nom_nouvelle_image
- docker commit container_id nom_nouvelle_image
- man docker-image
- docker tag ubuntu_theo:latest ubuntu_theo:0.1

Produire une image de l'état d'un conteneur - commit

La sous commande commit permet de générer une image à partir d'un conteneur existant. Il est préconisé d'arrêter le conteneur au préalable. Le contenu des volumes ne sont pas pris en compte.

Syntaxe :

docker commit nom_container nom_nouvelle_image
docker commit container_id nom_nouvelle_image

man docker-image

```
[root@poste1]# docker run -it ubuntu
root@e54bc6b747a7:/# rm -rf /opt
root@e54bc6b747a7:/# mkdir /rep
root@e54bc6b747a7:/# touch fic1
root@e54bc6b747a7:/# exit
exit

[root@poste1]# docker commit e54bc6b747a7 ubuntu_theo
sha256:5776f6168cdb775d2c304eb5e69a471335de67217be4283b65fc4db40fbc5ff1

[root@poste1]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu_theo         latest              5776f6168cdb        26 seconds ago      79.6MB
ubuntu              latest              452a96d81c30        3 weeks ago         79.6MB

[root@poste1]# docker run -i -t ubuntu_theo
root@b754da9a37fd:/# ls /
bin  boot  dev  etc  fic1  home  lib  lib64  media  mnt  proc  rep  root  run  sbin  srv
sys  tmp  usr  var
```

Utilisation des tags : docker tag ...

Un tag correspond à un alias pour image.

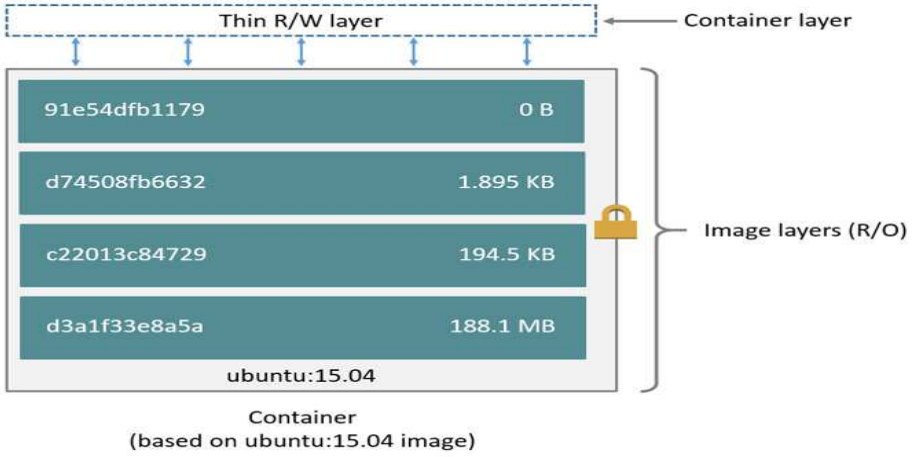
```
[root@poste1]# docker run -it --name version1 ubuntu_theo
root@e11c98a8120f:/# ... manipulations
root@e11c98a8120f:/# exit
exit

[root@poste1]# docker stop version1
version1

[root@poste1]# docker commit version1 ubuntu_theo:1.0
sha256:633d14acead517b8002f4ca1564173216955d5ba308942e02601c740681351a9

[root@poste1]# docker images ubuntu_theo
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu_theo         1.0                 633d14acead5        9 seconds ago       79.6MB
ubuntu_theo         latest              5776f6168cdb        7 minutes ago       79.6MB


[root@poste1]# docker run -it --name version2 ubuntu_theo:1.0
root@57fae319d176:/# ... manipulations
root@57fae319d176:/# exit
exit

[root@poste1]# docker stop version2
version2

[root@poste1]# docker commit version2 ubuntu_theo:1.1
sha256:292ebc700bd95cd9f7a8edb7843e2f2039db39d776b313c0a0ee8f8c94fc13cc

[root@poste1]# docker images ubuntu_theo
REPOSITORY          TAG                 IMAGE ID            CREATED              SIZE
ubuntu_theo         1.1                 292ebc700bd9        7 seconds ago        79.6MB
ubuntu_theo         1.0                 633d14acead5        About a minute ago   79.6MB
ubuntu_theo         latest              5776f6168cdb        8 minutes ago        79.6MB

[root@poste1]# docker tag ubuntu_theo:latest ubuntu_theo:0.1

[root@poste1]# docker images ubuntu_theo
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu_theo         1.1                 292ebc700bd9        5 minutes ago       79.6MB
ubuntu_theo         1.0                 633d14acead5        6 minutes ago       79.6MB
ubuntu_theo         0.1                 5776f6168cdb        14 minutes ago      79.6MB
ubuntu_theo         latest              5776f6168cdb        14 minutes ago      79.6MB

[root@poste1]# docker rmi ubuntu_theo:latest
Untagged: ubuntu_theo:latest

[root@poste]# docker images ubuntu_theo
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu_theo         1.1                 292ebc700bd9        6 minutes ago       79.6MB
ubuntu_theo         1.0                 633d14acead5        7 minutes ago       79.6MB
ubuntu_theo         0.1                 5776f6168cdb        15 minutes ago      79.6MB
[root@poste1]#
```

Pour lister les tags disponibles pour une image donnée disponible sur un registry (dépôt) :

curl https://<nom_du_registry>/v1/repositories/<Nom_de_l_image>/tags

Attention à l'espace de stockage :

Supprimer des fichiers au sein d'un conteneur, puis générer une image ne réduit pas sa taille par rapport à l'image de base. C'est toujours le principe des layers.




```
[root@poste]# docker  images
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
docker.io/centos        latest              e934aafc2206        3 weeks ago         199 MB

[root@poste]# docker ps -a | grep  95462e4d6fed
95462e4d6fed centos "/bin/bash"  3 hours ago  Exited (0) 4 minutes ago  competent_goodall

[root@poste]# docker start -i 95462e4d6fed
[root@95462e4d6fed /]# dd if=/dev/zero of=/grosfic1 bs=1M count=100
100+0 records in
100+0 records out
104857600 bytes (105 MB) copied, 0.0919759 s, 1.1 GB/s
[root@95462e4d6fed /]# exit
exit

[root@poste]# docker  stop  95462e4d6fed
95462e4d6fed

[root@poste]# docker  commit  95462e4d6fed  centos_version1
sha256:45c879fd8f3a4c7013faa00ddaedc93ddb4da1af45bd05882ed6b9c7ccabc0e3

[root@poste]# docker  images
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
centos_version1         latest              45c879fd8f3a        12 seconds ago      303 MB
docker.io/centos        latest              e934aafc2206        3 weeks ago         199 MB

[root@poste]# docker  run  -it  centos_version1
[root@7df034cc1e9b /]# ls
anaconda-post.log  bin  datas  dev  etc  grosfic1  lib  lib64  media  mnt  opt  proc
root  run  sbin  srv  sys  tmp  usr  var
[root@7df034cc1e9b /]# rm  grosfic1
rm: remove regular file 'grosfic1'? y
[root@7df034cc1e9b /]# exit
exit

[root@poste]# docker  commit  7df034cc1e9b  centos_version2
sha256:c0324d47fc2cfe3dc4eb5a8421f34a0a820c86927bef987420ae72ca2fdf8e2a

[root@poste]# docker  images
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
centos_version2         latest              c0324d47fc2c        8 seconds ago       303 MB
centos_version1         latest              45c879fd8f3a        9 minutes ago       303 MB
docker.io/centos        latest              e934aafc2206        3 weeks ago         199 MB
```

Modification de variables lors de la création de l'image : -c ou --change

```
[root@poste]# docker  stop  95462e4d6fed
95462e4d6fed

[root@poste]# docker  inspect  -f "{{ .Config.Env }}"  95462e4d6fed
[PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin]

[root@poste]# docker  commit  \
                        --change  "ENV VILLE Aix"  \
                        -c  "ENV PATH /usr/sbin:/usr/bin:/sbin:/bin"  \
                        95462e4d6fed  centos_env
sha256:f910700a7aedd25c816bc7cd421fc6cd95df048e2a0fe1444a05024ac8c0448c

[root@poste]# docker  inspect  -f "{{ .Config.Env }}"  centos_env
[VILLE=Aix PATH=/usr/sbin:/usr/bin:/sbin:/bin]

[root@poste]# docker  images  centos_env
REPOSITORY          TAG                 IMAGE ID            CREATED              SIZE
centos_env          latest              f910700a7aed        About a minute ago   199 MB

[root@poste]# docker  run  -it centos_env
[root@725d142721fb /]# echo  $PATH
/usr/sbin:/usr/bin:/sbin:/bin
[root@725d142721fb /]# echo  $VILLE
Aix
[root@725d142721fb /]# exit
exit
[root@poste]#
```

Création d'une image personnalisée

Le fichier Dockerfile

```
# tree  projet1
projet1
├── Dockerfile
└── src
    ├── database.sql
    └── index.html
```

```
# cat Dockerfile
FROM centos:latest

RUN yum  install -y  wget  gcc  make
RUN wget  https://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz
RUN tar  zxvf  hello-2.10.tar.gz
RUN cd  hello-2.10 && ./configure && make && make  install
```

```
# docker  build  -t bonjour  .
```

Le fichier Dockerfile

Le fichier Dockerfile contient toutes les instructions pour créer une image personnalisée.

La bonne pratique est de créer un répertoire pour le projet de création d'une image. Il contiendra le fichier Dockerfile, ainsi que tous les fichiers nécessaires.

Exemple :

```
# tree  projet1
projet1
├── Dockerfile
└── src
    ├── database.sql
    └── index.html
```

```
# cat Dockerfile
FROM centos:latest

RUN yum  install -y  wget  gcc  make
RUN wget  https://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz
RUN tar  zxvf  hello-2.10.tar.gz
RUN cd  hello-2.10 && ./configure && make && make  install
```

FROM indique l'image source à utiliser.
Bonne pratique: spécifier la version (tag).

RUN pour exécuter une commande.

La sous-commande build permet de créer une image en utilisant le fichier Dockerfile. Ce dernier est spécifié en argument :

docker build -t nom_de_l_image repertoire_du_fichier_Dockerfile

```
# docker  build  -t bonjour  .
Step 1/5 : FROM centos:latest
latest: Pulling from library/centos
aeb7866da422: Pull complete
Digest: sha256:67dad89757a55bfdfabec8abd0e22f8c7c12a18565147264702280 63ed86593b
Status: Downloaded newer image for centos:latest
 ---> 75835a67d134
Step 2/5 : RUN yum  install -y  wget  gcc  make
 ---> Running in e697ef1a5edd
. . .
Removing intermediate container e697ef1a5edd
 ---> 51e1f0bd34fa
Step 3/5 : RUN wget  https://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz
 ---> Running in f5e89bc3b6ab
. . .
Removing intermediate container f5e89bc3b6ab
 ---> c010a8c15c7a
Step 4/5 : RUN tar  zxvf  hello-2.10.tar.gz
 ---> Running in bc5cb7525b1a
. . .
Removing intermediate container bc5cb7525b1a
 ---> 2fdbfc5d6549
Step 5/5 : RUN cd  hello-2.10 && ./configure && make && make  install
 ---> Running in 16b3673c7c85
. . .
Removing intermediate container 16b3673c7c85
 ---> e6078bfa581b
Successfully built e6078bfa581b
Successfully tagged bonjour:latest
#

# docker  images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
bonjour             latest              e6078bfa581b        4 minutes ago       354MB
centos              latest              75835a67d134        4 weeks ago         199MB

# docker images -a
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
bonjour             latest              e6078bfa581b        4 minutes ago       352MB
<none>              <none>              2fdbfc5d6549        5 minutes ago       351MB
<none>              <none>              c010a8c15c7a        5 minutes ago       348MB
<none>              <none>              51e1f0bd34fa        5 minutes ago       347MB
centos              latest              75835a67d134        3 weeks ago         200MB

# docker history bonjour
IMAGE          CREATED         CREATED BY                                      SIZE
e6078bfa581b   3 minutes ago   /bin/sh -c cd  hello-2.10 && ./configure && …  1.49MB
2fdbfc5d6549   4 minutes ago   /bin/sh -c tar  zxvf  hello-2.10.tar.gz         3.09MB
c010a8c15c7a   4 minutes ago   /bin/sh -c wget  https://ftp.gnu.org/gnu/hel…  726kB
51e1f0bd34fa   4 minutes ago   /bin/sh -c yum  install -y  wget  gcc  make    147MB
75835a67d134   3 weeks ago     /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>      3 weeks ago     /bin/sh -c #(nop)  LABEL org.label-schema.sc…  0B
<missing>      3 weeks ago     /bin/sh -c #(nop) ADD file:fbe9badfd2790f074…  200MB
```

Remarque :

On ajoute une instruction RUN à la fin du fichier Dockerfile pour faire le « nettoyage ». Cependant le résultat est que nous rajoutons une couche (layer) supplémentaire.

D'autre part, le « build » se réalise plus rapidement car il utilise le cache.
L'option --no-cache permet de rebuilder l'image sans utiliser le cache. Ce qui est nécessaire si vous avez modifié un fichier qui doit être copié via le Dockerfile, alors que ce dernier n'est pas modifié.

```
# cat  Dockerfile
FROM centos:latest

RUN yum  install -y  wget  gcc  make
RUN wget  https://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz
RUN tar  zxvf  hello-2.10.tar.gz
RUN cd  hello-2.10 && ./configure && make && make  install
RUN yum remove -y   make gcc wget
RUN rm -rf hello-2.10.tar.gz hello-2.10

# docker  build  -t bonjour2  .
Sending build context to Docker daemon  2.048kB
Step 1/7 : FROM centos:latest
 ---> e934aafc2206
Step 2/7 : RUN yum  install -y  wget  gcc  make
 ---> Using cache
 ---> d47e74858532
Step 3/7 : RUN wget  https://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz
 ---> Using cache
 ---> 8d026888fd51
Step 4/7 : RUN tar  zxvf  hello-2.10.tar.gz
 ---> Using cache
 ---> bb62f56c5e8a
Step 5/7 : RUN cd  hello-2.10 && ./configure && make && make  install
 ---> Using cache
 ---> 5a40877c50fd
Step 6/7 : RUN yum remove -y   make gcc wget
 ---> Running in d454c04cb961
. . .
Removing intermediate container d454c04cb961
 ---> 7714545117f6
Step 7/7 : RUN rm -rf hello-2.10.tar.gz hello-2.10
 ---> Running in 1a7398be8768
Removing intermediate container 1a7398be8768
 ---> c4d2177d1c74
Successfully built c4d2177d1c74
Successfully tagged bonjour2:latest
#
```

C'est beaucoup plus rapide car il utilise le cache.

```
# docker  images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
bonjour2            latest              c4d2177d1c74        3 minutes ago       374MB
bonjour             latest              5a40877c50fd        11 minutes ago      354MB
centos              latest              e934aafc2206        4 weeks ago         199MB
```

Pour minimiser la taille de l'image, il faut minimiser le nombre de couches :

```
# cat  Dockerfile
FROM centos:latest

RUN yum  install -y  wget  gcc  make \
    &&  wget  https://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz \
    &&  tar  zxvf  hello-2.10.tar.gz \
    &&  cd  hello-2.10 && ./configure && make && make  install \
    &&  yum remove -y   make gcc wget \
    &&  cd / && rm -rf hello-2.10.tar.gz hello-2.10
```

```
# docker  build  -t bonjour3  .
Sending build context to Docker daemon  2.048kB
Step 1/2 : FROM centos:latest
 ---> e934aafc2206
Step 2/2 : RUN yum  install -y  wget  gcc  make     &&  wget
https://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz     &&  tar  zxvf  hello-2.10.tar.gz
&&  cd  hello-2.10 && ./configure && make && make  install     &&  yum remove -y   make
gcc wget     &&  cd / && rm -rf hello-2.10.tar.gz hello-2.10
 ---> Running in 7c9dbbe4676e
. . .
Removing intermediate container 7c9dbbe4676e
 ---> bf99120c78ff
Successfully built bf99120c78ff
Successfully tagged bonjour3:latest
#

# docker  images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
bonjour3            latest              bf99120c78ff        31 seconds ago      324MB
bonjour2            latest              c4d2177d1c74        12 minutes ago      374MB
bonjour             latest              5a40877c50fd        20 minutes ago      354MB
```

Pour conserver que la dernière image :

```
# docker  rmi  bonjour2
Untagged: bonjour2:latest
Deleted: sha256:c4d2177d1c748e4d802f7315f0422923bbf4ee0e5dd57da4cc7cf1940fe90d27
Deleted: sha256:b15ec16d082cf5aa77ed909d58cdbfa53a40956e3981b395c3475145e568e7b3
Deleted: sha256:7714545117f6ce4b6bc1b02736b89e932020c10b66a9fbe7089a5085bf19d333
Deleted: sha256:8a1c45c00e4c91baf0e0dd3ebce116667a2573b8832aa753b284dd4ef955dc1b

# docker  rmi  bonjour
Untagged: bonjour:latest
Deleted: sha256:5a40877c50fd3c7d1072fd64777347814fcb22b3a55f9c54161ac3cfaa9cd442
Deleted: sha256:7926c1dbc65668d85c7433f91d8cc25efffebef9bd26e001962b90fba615c3c3
Deleted: sha256:bb62f56c5e8a2cf3a064ee6d0e7afdc29cbff3ba0e4985c9b12b90e39a955afe
Deleted: sha256:f6e4ed59f989febbcf5db6e694ebcd2654224df7306237b5b5ec6b2d4573bb3b
Deleted: sha256:8d026888fd51ab6c9c8e1488842f6cfa326f11cc8e2705f33c6c6134e6fbd7c6
Deleted: sha256:0f34d6969d72d17e54f1ed4834e2be2fcb649feed5e97fd26d69074c75bb5b29
Deleted: sha256:d47e74858532cb56ba39549748353b84d3b8541d3917dce6732fbcc36c4557c3
Deleted: sha256:c2fe39a1aeb2d8c01ef5ded8c1f436b52972cc65e227b68f742c4842bdf099d5


# docker  tag  bonjour3  bonjour
# docker  rmi  bonjour3
Untagged: bonjour3:latest
# docker  images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
bonjour             latest              bf99120c78ff        5 minutes ago       324MB
```

Vérification du fonctionnement :

```
# docker  run  -it  --rm  bonjour
[root@90289509b3a6 /]# ls
anaconda-post.log  dev  home  lib64  mnt  proc  run   srv  tmp  var
bin                etc  lib   media  opt  root  sbin  sys  usr
[root@90289509b3a6 /]# hello
Hello, world!
[root@90289509b3a6 /]# exit
exit
#

# docker  run  --rm   bonjour  hello -g "Bonne journee"
Bonne journee
#

# docker  run  -it  --rm  bonjour  ls
anaconda-post.log  dev    home  lib64  mnt  proc    run   srv  tmp     var
bin            etc lib   media  opt  root    sbin  sys  usr
#

# docker  run  bonjour
# docker  ps  -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
a1a793b93465        bonjour             "/bin/bash"         11 seconds ago      Exited
(0) 7 seconds ago                       sharp_mcclintock
```

Création d'une image personnalisée

Le Dockerfile - ENTRYPOINT

ENTRYPOINT ["executable", "param1", "param2"]

```
# cat  Dockerfile
FROM centos:latest

RUN yum  install -y  wget  gcc  make \
    &&  wget  https://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz \
    &&  tar  zxvf  hello-2.10.tar.gz \
    &&  cd  hello-2.10 && ./configure && make && make  install \
    &&  yum remove -y   make gcc wget \
    &&  cd / && rm -rf hello-2.10.tar.gz hello-2.10

ENTRYPOINT ["hello"]
```

Le Dockerfile - ENTRYPOINT

ENTRYPOINT permet de configurer un conteneur en tant qu'exécutable.

ENTRYPOINT ["executable", "param1", "param2"]

```
# cat  Dockerfile
FROM centos:latest

RUN yum  install -y  wget  gcc  make \
    &&  wget  https://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz \
    &&  tar  zxvf  hello-2.10.tar.gz \
    &&  cd  hello-2.10 && ./configure && make && make  install \
    &&  yum remove -y   make gcc wget \
    &&  cd / && rm -rf hello-2.10.tar.gz hello-2.10

ENTRYPOINT ["hello"]
```

```
# docker  build  -t  bonjour4  .

# docker  run  --rm  bonjour4
Hello, world!

# docker  run  --rm  bonjour4  -g "Bonne journee"
Bonne journee
#
```

Autre exemple :

```
# cat  Dockerfile
FROM centos:latest

RUN yum  install -y  wget  gcc  make \
    &&  wget  https://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz \
    &&  tar  zxvf  hello-2.10.tar.gz \
    &&  cd  hello-2.10 && ./configure && make && make  install \
    &&  yum remove -y   make gcc wget \
    &&  cd / && rm -rf hello-2.10.tar.gz hello-2.10

ENTRYPOINT ["hello","-g","Mon argument"]
```

```
# docker  build  -t  bonjour4  .

# docker  run  --rm  bonjour4
Mon argument

# docker  run  --rm  bonjour4  -g "Bonne journee"
Bonne journee
#
```

Création d'une image personnalisée

Le Dockerfile – les mots clefs

ENTRYPOINT [“cat”]
CMD [“-EVT”, “/ETC/HOSTS”, “/ETC/MOTD”]

LABEL
EXPOSE VOLUME
COPY ADD
USER
ENV ARG
WORKDIR STOPSIGNAL

Le Dockerfile – les mots clefs

Les lignes commençant par dièse sont des commentaires.

CMD définit la commande à exécuter lors du démarrage du conteneur.

CMD cat /etc/passwd /etc/group La commande est prise comme une chaîne de caractères
Commande exécutée : /bin/sh -c "cat /etc/passwd /etc/group"
Ne pas privilégier cette syntaxe.

CMD ["cat","/etc/passwd","/etc/group"]
Commande exécutée : cat /etc/passwd /etc/group

ENTRYPOINT [“commande”]
CMD [“ARG1”, “ARG2”] RÉSULTAT : COMMANDE ARG1 ARG2

ENTRYPOINT [“cat”]
CMD [“-EVT”, “/ETC/HOSTS”, “/ETC/MOTD”] RÉSULTAT : CAT -EVT /ETC/HOSTS /ETC/MOTD

Plus généralement : Le mot clé 'ENTRYPOINT' est l'exécutable du conteneur et le mot clé 'CMD' ses arguments par défaut.
Qu'avec CMD : Si le conteneur est exécuté avec des arguments, le mot clé 'CMD' n'est pas pris en compte.
Qu'avec ENTRYPOINT : Si le conteneur est exécuté avec des arguments, ceux-ci sont les arguments de la commande spécifiée par ENTRYPOINT.

Le tableau ci-dessous indique les relations entre CMD et ENTRYPOINT :

(tableau issu du site docs.docker.com)

	No ENTRYPOINT	**ENTRYPOINT exec_entry p1_entry**	**ENTRYPOINT [“exec_entry”, “p1_entry”]**
No CMD	*error, not allowed*	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry
CMD [“EXEC_CMD”, “P1_CMD”]	exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry exec_cmd p1_cmd
CMD [“P1_CMD”, “P2_CMD”]	p1_cmd p2_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry p1_cmd p2_cmd
CMD EXEC_CMD P1_CMD	/bin/sh -c exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry /bin/sh -c exec_cmd p1_cmd

exec_cmd ou exec_entry : commande
p1_cmd, p2_cmd, p1_entry : argument

LABEL maintener="baranger@spherius.fr"
LABEL sert à définir des méta données.
Le mot clé maintener définit l'auteur de l'image.

EXPOSE pour exposer un port réseau.

VOLUME pour créer un volume pour le stockage.

COPY pour copier un fichier ou répertoire de l'hôte vers l'image.

ADD pour copier un fichier (ou répertoire) de l'hôte ou depuis une URL vers l'image, il sert également à décompresser automatiquement une archive (tar, zip, etc).

USER définit l'utilisateur qui lance la commande du ENTRYPOINT ou du CMD.

ENV pour définir des variables d'environnement pour l'image.
on peut surcharger la valeur lors de l'exécution (run) par l'option "-e".

ARG similaire à ENV, mais juste le temps de la construction de l'image.

WORKDIR définit le répertoire de travail.
Il correspond au répertoire de travail lorsque l'on se connecte au conteneur.
Il sert de répertoire de base pour les chemins relatifs du Dockerfile pour les instructions qui sont après WORKDIR : ADD, COPY, RUN, CMD, ENTRYPOINT.

STOPSIGNAL définit le signal qui sera envoyé au conteneur lorsqu'il sera stoppé par docker stop.

Exemple :

```
# tree .
.
├── Dockerfile
└── src
    ├── site.conf
    └── index.modele
```

```
# cat  Dockerfile
FROM ubuntu

LABEL description="Test de creation d image" \
      maintainer="Baranger Jean-Marc" \
      version="1.0"

ARG reppage=/var/www/html
ARG repconf=/etc/apache2
ENV ville Paris
ENV APACHE_RUN_USER www-data
ENV APACHE_RUN_GROUP www-data
ENV APACHE_LOG_DIR /var/web/log/apache2
ENV APACHE_PID_FILE /var/run/apache2.pid
ENV APACHE_RUN_DIR /var/run/apache2
ENV APACHE_LOCK_DIR /var/lock/apache2

RUN export DEBIAN_FRONTEND=noninteractive && apt-get update && apt-get -y -q upgrade &&
apt-get -y -q install apache2

COPY src/index.modele ${reppage}/index.html
COPY src/site.conf  ${repconf}/apache2.conf

EXPOSE 80 443
VOLUME /var/www/html

WORKDIR /var/www

CMD ["apache2ctl","-D","FOREGROUND"]
```

```
# docker  build  -t  jmb/apache  .

# docker  images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
jmb/apache          latest              0476afa3d9e6        7 seconds ago       215MB

# docker  run  -d  --name site  jmb/apache
de785fe243a19b5f6b60e2f468f80a1a0239faeb9b43e693215bb349cea467a0

# docker  inspect  site  |  grep  -i  ipaddress
                    "IPAddress": "172.17.0.2",
```

Le site web fonctionne avec http://172.17.0.2.

```
# docker inspect site
[
    {
        "Id": "de785fe243a19b5f6b60e2f468f80a1a0239faeb9b43e693215bb349cea467a0",
        "Created": "2018-05-06T13:51:02.384708049Z",
        "Path": "apache2ctl",
        "Args": [
            "-D",
            "FOREGROUND"
        ],

        "Mounts": [
            {
                "Type": "volume",
                "Name":
"6993da272bf94e5cc5fdbe510aec1e40d6b4b428409ef273df2316b485e7873d",
                "Source":
"/var/lib/docker/volumes/6993da272bf94e5cc5fdbe510aec1e40d6b4b428409ef273df2316b485e7873d
/_data",
                "Destination": "/var/www/html",
                "Driver": "local",
                "Mode": "",
                "RW": true,
                "Propagation": ""
            }
        ],
        "Config": {
            "ExposedPorts": {
                "443/tcp": {},
                "80/tcp": {}
            },
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
                "ville=Paris",
                "APACHE_RUN_USER=www-data",
                "APACHE_RUN_GROUP=www-data",
                "APACHE_LOG_DIR=/var/web/log/apache2",
                "APACHE_PID_FILE=/var/run/apache2.pid",
                "APACHE_RUN_DIR=/var/run/apache2",
                "APACHE_LOCK_DIR=/var/lock/apache2"
            ],
            "Cmd": [
                "apache2ctl",
                "-D",
                "FOREGROUND"
            ],
            "ArgsEscaped": true,
            "Image": "jmb/apache",
            "Volumes": {
                "/var/www/html": {}
            },
            "WorkingDir": "/var/www",
            "Labels": {
                "description": "Test de creation d image",
                "maintainer": "Baranger Jean-Marc",
                "version": "1.0"
            }
        },
```

```
# docker  exec  -it  -e ville=Aix  site  /bin/bash
root@de785fe243a1:/var/www# echo $ville
Aix
root@de785fe243a1:/var/www# cat html/index.html
<html>
<body><h1>BONJOUR</h1></body>
</html>
root@de785fe243a1:/var/www# pwd
/var/www
root@de785fe243a1:/var/www# sed 's/BONJOUR/SALUT A TOUS/g' \
                              html/index.html  >  html/autre.html
root@de785fe243a1:/var/www#
```

On peut accéder à la page : http://172.17.0.2/autre.html

```
root@146e4a0a20cb:/var/www# touch NOUVEAU
root@146e4a0a20cb:/var/www# ls
NOUVEAU  html
root@146e4a0a20cb:/var/www# exit
exit

# docker stop site
# ls
/var/lib/docker/volumes/6993da272bf94e5cc5fdbe510aec1e40d6b4b428409ef273d
f2316b485e7873d/_data
autre.html  index.html
```

Création d'une image personnalisée

Le Dockerfile – les bonnes pratiques

FROM LABEL

- Répertoire pour organisation
- Optimiser de la taille de l'image
- Limiter les processus root
- Le fichier .dockerignore

Le Dockerfile – les bonnes pratiques

Pour FROM :
Utiliser l'image avec sa version (tag). Choisir une image de taille réduite.

Pour LABEL :
Utilisez le pour les meta datas, cela peut être pratique pour fournir des informations.

Créer un répertoire pour votre projet :
Il est recommandé de créer un répertoire spécifique pour votre projet de création d'image.
Les fichiers utilisés lors de la création de l'image (build) ne peuvent pas être dans un répertoire au dessus de celui du fichier Dockerfile.

```
# tree projet1
projet1
├── Dockerfile
└── src
    ├── database.sql
    └── index.html
```

Exemple d'échec :

```
# docker  build  -t perso  .
Sending build context to Docker daemon  3.072kB
Step 1/2 : FROM centos:latest
 ---> e934aafc2206
Step 2/2 : COPY ../perso.bash  /
COPY failed: Forbidden path outside the build context: ../perso.bash ()
```

Optimiser la taille de l'image :

- on peut minimiser la taille de l'image en limitant les layers.

```
RUN cmd1                    RUN cmd1 \
RUN cmd2                        && cmd2 \
RUN cmd3                        && cmd3
```

- en supprimant tout ce qui est inutile au bon fonctionnement de l'image.

```
COPY  src/appli.tar.gz  appli.tar.gz
RUN   tar  zxf appli.tar.gz -C .  \
      &&  yum  install  -y  gcc  make
      &&  cd  appli  &&  ./configure  &&  make  &&  make  install \
      &&  cd ..  &&  rm  -rf  appli  appli.tar.gz \
      &&  yum  remove  -y  gcc  make
```

Optimisation pour limiter les processus root :

- Limiter les processus root.

SANS le mot clé user :

```
[phase2]# docker  top  test
UID                 PID                 PPID                C                   STIME
TTY                 TIME                CMD
root                1176                1161                2                   17:48
pts/0               00:00:00            /bin/bash
```

AVEC le mot clé user :

```
[phase2]# docker  top  test
UID                 PID                 PPID                C                   STIME
TTY                 TIME                CMD
user1               838                 822                 0                   17:40
pts/0               00:00:00            /bin/bash
```

```
[phase2]# cat Dockerfile
FROM centos:latest

LABEL description="Test de creation d image" \
      maintainer="Baranger Jean-Marc" \
      version="1.0"

ENV ville=Paris \
    pays=France \
    societe=Spherius

COPY monbonjour.bash /usr/bin/bonjour

RUN     useradd  user1 \
    &&  chmod a+x /usr/bin/bonjour \
    &&  yum  install -y  wget  gcc  make \
    &&  wget  https://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz \
    &&  tar  zxvf  hello-2.10.tar.gz \
    &&  cd  hello-2.10 && ./configure && make && make  install \
    &&  yum remove -y  make gcc wget \
    &&  cd / && rm -rf hello-2.10.tar.gz hello-2.10

USER user1
```

Le fichier .dockerignore :

Le fichier .dockerignore permet de définir les fichiers et répertoires qu'il ne faudra pas intégrer dans l'image.

Exemple :

Sans .dockerignore

```
# tree -a .
.
├── conf
│   └── app1.conf
├── Dockerfile
├── mot_de_passe
└── sources
    └── app1

# cat Dockerfile
FROM centos:latest
COPY  .  /opt/monappli    # Copy du répetoire courant au sein de /op/mnappli
. . .

# docker build -t image1 .
      Au sein de l'image tous les fichiers sont présents.
```

Avec .dockerignore

```
# tree -a .
.
├── conf
│   └── app1.conf
├── Dockerfile
├── .dockerignore
├── mot_de_passe
└── sources
    └── app1

# cat Dockerfile
FROM centos:latest
COPY  .  /opt/monappli
. . .

# cat .dockerignore              ou          # cat .dockerignore
.dockerignore                                **
Dockerfile                                   !sources
mot_de_passe
conf

# docker build -t image2 .
      Au sein de l'image, il n'est conservé que les fichiers adéquates.
```

Création d'une image personnalisée

Sauvegarde et restauration d'une image

docker save -o ../dockerfile_backup/bonjour.save bonjour

docker load -i dockerfile_backup/bonjour.save

Sauvegarde et restauration d'une image

Il est important de sauvegarder ses projets relatifs à la création d'images, en particulier le fichier Dockerfile.

Les mots clés save et load ne concernent que les images.
Note: Les sous commandes export et import concernent les conteneurs, et non les images.

Sauvegarde : save

```
# docker  save  -o ../dockerfile_backup/bonjour.save  bonjour
```

```
# file  ../dockerfile_backup/bonjour.save
../dockerfile_backup/bonjour.save: POSIX tar archive
```

Restauration : load

```
# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
```

```
# docker  load  -i dockerfile_backup/bonjour.save
37f3ed40a71c: Loading layer  126.4MB/126.4MB
Loaded image: bonjour:latest
```

```
# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
bonjour             latest              8328cf5fa2a0        22 minutes ago      324MB
```

Notes

Docker et le réseau

Dans ce chapitre nous allons étudier les réseaux Docker.

Docker et le réseau

- Les ports réseaux
- Les drivers réseaux
- La création d'un réseau
- La connexion d'un conteneur

Docker et le réseau

Les ports réseaux

docker run -itd -p port_hôte:port_docker image

docker inspect container1

docker port container1

docker run -itd -P image

Les ports réseaux

Par défaut, les ports d'un conteneur sont « privés », c'est à dire non accessible de l'extérieur.
Pour rendre « public » un port :

docker run -itd -p port_hôte:port_docker . . . image

Mappage d'un port :

```
[root@poste]# docker run -itd -p 90:80 --name=container1 nginx
6cd3945bc31be275c544008d53668b3c0441e32e72a4ec06d87440b38cdf8d4a

[root@poste]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS              PORTS                NAMES
6cd3945bc31b        nginx               "nginx -g 'daemon ..."   44 seconds ago
Up 43 seconds       0.0.0.0:90->80/tcp   container1

[root@poste]# docker inspect container1
            "PortBindings": {
                "80/tcp": [
                    {
                        "HostIp": "",
                        "HostPort": "90"
                    }
                ]
            },
```

du poste via un browser avec http://172.17.0.2 on obtient la page de garde nginx : port 80
du poste via un browser avec http://localhost:90 on obtient la page de garde nginx : port 90

Autre exemple :

La sous commande port permet de lister les ports exposés d'un conteneur.

```
[root@poste]# docker run -d --name container1  -p 90:80 -p 93:443  nginx
864fe7c69b26007038ea18246138e919477f28ff66fbd14de5fba879d200e9f3
```

```
[root@poste]# docker  port  container1
443/tcp -> 0.0.0.0:93
80/tcp -> 0.0.0.0:90
```

L'option P :

L'option -P est utilisée pour mapper automatiquement n'importe quel port réseau du hôte Docker à un port du conteneur (une valeur aléatoire dans une plage des ports éphémères).

```
[root@poste]#  docker  run  -itd  -P  nginx
7056a3881be3e00dbfab14aafe757eff0c5b9366de4000b97a54bbe94b21e929
```

```
[root@poste]# docker  ps
CONTAINER ID        IMAGE          COMMAND                  CREATED          STATUS
PORTS                   NAMES
7056a3881be3        nginx          "nginx -g 'daemon of…"   6 seconds ago    Up 5 seconds
0.0.0.0:32768->80/tcp   happy_ardinghelli
```

Docker et le réseau

Les drivers réseaux

docker network { create | inspect | ls | rm | connect | disconnect }

```
# docker  network  ls
NETWORK ID          NAME                DRIVER              SCOPE
c7dcffacfd61        bridge              bridge              local
6fadbecb9703        host                host                local
a8805480d7ad        none                null                local

# docker  network  inspect  bridge
```

Les drivers réseaux

Syntaxe de la commande

docker network { create | inspect | ls | rm | connect | disconnect }

Pour lister les réseaux :

```
# docker  network  ls
NETWORK ID          NAME                DRIVER              SCOPE
c7dcffacfd61        bridge              bridge              local
6fadbecb9703        host                host                local
a8805480d7ad        none                null                local
```

```
# ip  a
. . .
5: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:28:91:03:54 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:28ff:fe91:354/64 scope link
       valid_lft forever preferred_lft forever
97: veth428c1dd@if96: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master
docker0 state UP
    link/ether ae:16:6e:83:62:bb brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::ac16:6eff:fe83:62bb/64 scope link
       valid_lft forever preferred_lft forever
```

Par défaut les conteneurs sont connectés à docker0 qui correspond à bridge.

Les drivers :

Bridge : c'est le driver par défaut, pour créer un réseau interne entre conteneurs.

Overlay : ce driver permet de définir un réseau interne entre différents hôtes.
Les réseaux overlay connectent plusieurs démons Docker ensemble (différents hôtes) et permettent aux services d'un cluster Swarm de communiquer entre eux. Vous pouvez également utiliser des réseaux overlay pour faciliter la communication entre un service Swarm et un conteneur autonome, ou entre deux conteneurs autonomes sur différents démons Docker. Cette stratégie supprime le besoin d'effectuer un routage au niveau du système d'exploitation (hôte) entre ces conteneurs.

Host : utilisé que par Swarm, ce driver permet d'avoir la même interface que le hôte.
Pour les conteneurs, l'isolation du réseau entre le conteneur et l'hôte Docker est supprimée. Le réseau de l'hôte est directement utilisé.

Macvlan : pour affecter des adresses macs aux conteneurs et les faire apparaître comme des machines physiques sur le réseau entreprise.
Le démon Docker achemine le trafic vers les conteneurs par leurs adresses MAC.
L'utilisation du pilote macvlan est parfois le meilleur choix lorsqu'il s'agit d'anciennes applications qui s'attendent à être directement connectées au réseau physique, plutôt que de passer par la pile réseau de l'hôte Docker.

None : pour désactiver le réseau. Il peut être utilisé pour exploiter un driver réseau personnalisé.

```
[root@poste]# docker  network  inspect  bridge
    {
        "Name": "bridge",
        "Id": "d846b80431d3cadb9df03471cb44ef0a7e5c45baa25ccb05807ec92d132f1d1e",
        "Created": "2018-05-04T09:33:03.087493099+02:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Containers": {},
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
            "com.docker.network.bridge.enable_icc": "true",
            "com.docker.network.bridge.enable_ip_masquerade": "true",
            "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
            "com.docker.network.bridge.name": "docker0",
            "com.docker.network.driver.mtu": "1500"
        },
        "Labels": {}
    }
```

Docker et le réseau

La création d'un réseau

```
# docker  network  create  reseau1

# docker  network  create \
                     --subnet 10.10.10.0/24    \
                     --gateway 10.10.10.254    \
                     --ip-range 10.10.10.0/28  \
                   reseau2

# docker  network  create  --driver bridge  reseau3
```

La création d'un réseau

```
[root@poste]# docker  network  create  reseau1
96eb7446bccad64c601f132f8628df2db9dfbfe95e2fc24be3511bcd65865313
```

```
[root@poste]# docker  network  create \
                     --subnet 10.10.10.0/24  --gateway 10.10.10.254  \
                     --ip-range 10.10.10.0/28  reseau2
95f0c7dc741460984f2882ba9799b9fdb2ee01eea4c873de8c110b420f346e91
```

Le reseau2 est créé avec des adresses IP de 1 à 14.

```
[root@poste]# docker  network  ls
NETWORK ID          NAME                DRIVER              SCOPE
c7dcffacfd61        bridge              bridge              local
6fadbecb9703        host                host                local
a8805480d7ad        none                null                local
96eb7446bcca        reseau1             bridge              local
95f0c7dc7414        reseau2             bridge              local
```

Une interface a été créé pour chaque réseau.

Les machines du reseau1 peuvent communiquer entre elles et avec l'extérieur. Elles ne peuvent pas communiquer avec les machines du reseau2.

```
[root@poste]# docker  network  create  --driver  bridge  reseau3
```

```
[root@poste]# docker  network  inspect  reseau1
[
    {
        "Name": "reseau1",
        "Id": "96eb7446bccad64c601f132f8628df2db9dfbfe95e2fc24be3511bcd65865313",
        "Created": "2018-05-07T17:08:06.749472363+02:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

```
[root@poste]# docker  network  inspect  reseau2
[
    {
        "Name": "reseau2",
        "Id": "95f0c7dc741460984f2882ba9799b9fdb2ee01eea4c873de8c110b420f346e91",
        "Created": "2018-05-07T17:12:17.693931069+02:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "10.10.10.0/24",
                    "IPRange": "10.10.10.0/28",
                    "Gateway": "10.10.10.254"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

Sur le système hôte :

```
[root@poste]# ip  a
5: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:28:91:03:54 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:28ff:fe91:354/64 scope link
       valid_lft forever preferred_lft forever
97: veth428c1dd@if96: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master
docker0 state UP
    link/ether ae:16:6e:83:62:bb brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::ac16:6eff:fe83:62bb/64 scope link
       valid_lft forever preferred_lft forever
98: br-96eb7446bcca: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN
    link/ether 02:42:e7:ad:7c:5e brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.1/16 brd 172.18.255.255 scope global br-96eb7446bcca
       valid_lft forever preferred_lft forever
99: br-95f0c7dc7414: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN
    link/ether 02:42:3e:4c:d4:be brd ff:ff:ff:ff:ff:ff
    inet 10.10.10.254/24 brd 10.10.10.255 scope global br-95f0c7dc7414
       valid_lft forever preferred_lft forever
```

Docker et le réseau

La connexion d'un conteneur

Les serveurs DNS

```
# docker run … --network reseau1 … un_conteneur
# docker network connect reseau1 un_conteneur
# docker network disconnect reseau1 un_conteneur
```

La connexion d'un conteneur

Un conteneur Docker utilise la même configuration DNS que le serveur hôte (/etc/resolv.conf). Dans le cas où le serveur hôte n'a pas de fichier resolv.conf, le conteneur crée son propre fichier resolv.conf avec les adresses de Google (8.8.8.8 et 8.8.4.4).

docker run … **--network reseau1** … nom_du_conteneur

Connexion d'un conteneur :

```
[root@poste]# docker network connect reseau1 un_conteneur
```

Cette opération peut être faite à chaud (conteneur en cours de fonctionnement).

```
root@740883f2d2a6:/# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
106: eth0@if107: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group
default
    link/ether 02:42:0a:0a:0a:01 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.10.10.1/24 brd 10.10.10.255 scope global eth0
114: eth1@if115: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group
default
    link/ether 02:42:ac:12:00:04 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.18.0.4/16 brd 172.18.255.255 scope global eth1
```

Déconnexion d'un conteneur :

```
[root@poste]# docker network disconnect reseau1 un_conteneur
```

Exemple :

```
[root@mars ~]# docker run --rm -it --name perso alpine
/ #
/ # hostname
8609799b2531
/ #
/ # cat /etc/hosts
127.0.0.1    localhost
::1    localhost ip6-localhost ip6-loopback
fe00::0      ip6-localnet
ff00::0      ip6-mcastprefix
ff02::1      ip6-allnodes
ff02::2      ip6-allrouters
172.17.0.2   8609799b2531
/ #
/ # cat /etc/resolv.conf                    Identique au fichier du hôte.
# Generated by NetworkManager
search home
nameserver 212.27.40.240
nameserver 212.27.40.241
/ #
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
80: eth0@if81: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
/ #
/ # ip route
default via 172.17.0.1 dev eth0
172.17.0.0/16 dev eth0 scope link  src 172.17.0.2
/ #
/ # ping -c2  www.google.fr
PING www.google.fr (74.125.206.94): 56 data bytes
64 bytes from 74.125.206.94: seq=0 ttl=41 time=37.034 ms
64 bytes from 74.125.206.94: seq=1 ttl=41 time=55.402 ms

--- www.google.fr ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 37.034/46.218/55.402 ms
/ #
/ # ping -c2 127.17.0.2    si un conteneur 127.17.0.2 sur le même réseau
PING 127.17.0.2 (127.17.0.2): 56 data bytes
64 bytes from 127.17.0.2: seq=0 ttl=64 time=0.610 ms
64 bytes from 127.17.0.2: seq=1 ttl=64 time=0.276 ms

--- 127.17.0.2 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.276/0.443/0.610 ms
/ #
```

Notes

Docker et le stockage

Dans ce chapitre nous allons étudier la gestion du stockage Docker.

Docker et le stockage

- Le stockage inter-conteneur
- Les volumes : volumes-from
- Les volumes
- L'option mount
- Les volumes : inspect
- Les volumes : suppression
- Le stockage des images
- Les registry privés

Docker et le stockage

Le stockage inter-conteneur

-v rep_du_hôte:rep_du_container

--volume rep_du_hôte:rep_du_container

```
-v  /root/Docker/nginx/site:/usr/share/nginx/html
-v $(pwd):/tmp  -v $(pwd):/home
-v /usr/share/nginx/html
```

Le stockage inter-conteneur

Syntaxe pour l'utilisation des volumes :

-v rep_du_hôte:rep_du_container
ou
--volume rep_du_hôte:rep_du_container

```
# docker  run  -d  --name container1  \
         -v  /root/Docker/nginx/site:/usr/share/nginx/html nginx
```

Le site qui apparaît correspond aux pages localisées sur le poste hôte au sein du répertoire :
/root/Docker/nginx/site

Démonstration :

```
[root@poste]# ls
datas  nginx

[root@poste]# docker  run  -it  --name un_container  \
                           -v $(pwd):/tmp  -v $(pwd):/home  centos

[root@65fa2f995d85 /]# ls  /tmp
datas  nginx
[root@65fa2f995d85 /]# cat  /etc/hosts
127.0.0.1    localhost
::1    localhost ip6-localhost ip6-loopback
. . .
172.17.0.2    65fa2f995d85
```

```
[root@65fa2f995d85 /]# cat  /etc/hosts  >  /tmp/fichier1
[root@65fa2f995d85 /]# ls  /tmp
datas  fichier1  nginx
[root@65fa2f995d85 /]# useradd  -m  -d /home/user2  user2
[root@65fa2f995d85 /]# ls -a /home/user2
.  ..  .bash_logout  .bash_profile  .bashrc
[root@65fa2f995d85 /]# exit

[root@poste]# ls
datas  fichier1  nginx  user2
[root@poste]# ls  -a  user2
.  ..  .bash_logout  .bash_profile  .bashrc
[root@poste]# cat  fichier1
127.0.0.1    localhost
::1    localhost ip6-localhost ip6-loopback
. . .
172.17.0.2    65fa2f995d85
```

```
[root@poste]# docker  start  un_container
un_container
[root@poste]# docker  attach  un_container
[root@65fa2f995d85 /]# ls  /tmp
datas  fichier1  nginx  user2
[root@65fa2f995d85 /]# exit
```

```
[root@poste]# docker  start  -i  un_container
[root@65fa2f995d85 /]# ls  /tmp
datas  fichier1  nginx  user2
[root@65fa2f995d85 /]# cat  /etc/passwd
root:x:0:0:root:/root:/bin/bash
. . .
user2:x:1000:1000::/home/user2:/bin/bash
[root@65fa2f995d85 /]# userdel  -r  user2
[root@65fa2f995d85 /]# ls  /home
datas  fichier1  nginx
[root@65fa2f995d85 /]# rm  /tmp/fichier1
[root@65fa2f995d85 /]# exit

[root@poste]# ls
datas  nginx
```

```
[root@poste]# docker  inspect  un_container
        "Mounts": [
            {
                "Type": "bind",
                "Source": "/root",
                "Destination": "/tmp",
                "Mode": "",
                "RW": true,
                "Propagation": "rprivate"
            },
            {
                "Type": "bind",
                "Source": "/root",
                "Destination": "/home",
                "Mode": "",
                "RW": true,
                "Propagation": "rprivate"
            }
        ],
```

Remarque :

```
[root@poste]# docker  run  -d  --name un_container  \
                           -v /usr/share/nginx/html  nginx
5255e1c92365530456649432870 2edf721460a31d82b01294d4dab5d2a991120
```

```
[root@poste]# docker  inspect  un_container
        "Mounts": [
            {
                "Type": "volume",
                "Name":
"44bcfbee43f5684a34dc7a614093d298f27dbb59a2fd704fb5a302cb72831055",
                "Source":
"/var/lib/docker/volumes/44bcfbee43f5684a34dc7a614093d298f27dbb59a2fd704fb5a302cb72831055
/_data",
                "Destination": "/usr/share/nginx/html",
                "Driver": "local",
                "Mode": "",
                "RW": true,
                "Propagation": ""
            }
        ],
```

```
[root@poste]# ls  -l
/var/lib/docker/volumes/44bcfbee43f5684a34dc7a614093d298f27dbb59a2fd704fb
5a302cb72831055/_data/
total 8
-rw-r--r--. 1 root root 537  9 avril 18:01 50x.html
-rw-r--r--. 1 root root 612  9 avril 18:01 index.html
[root@poste]#
```

Docker et le stockage

Les volumes : volumes-from

```
# docker  run  -d  --name web  \
                      -v /usr/share/nginx  -v /var/log/nginx  nginx

# docker  run  -it  --name os  --volumes-from web  alpine
```

Les volumes : volumes-from

L'option « --volumes-from » permet d'attacher des volumes existants d'un conteneur à un autre conteneur.

```
[root@mars ~]# docker  run  -d  --name web  \
                      -v /usr/share/nginx  -v /var/log/nginx  nginx
38612896e47d81f42d2cec849d01782e2f89cde3cccafbc2179d625c36315916

[root@mars ~]# docker  inspect  web
. . .
        "Mounts": [
            {
                "Type": "volume",
                "Name":
"acbace259e1214f7d53683d2cc4a3c17788fc11384f45befa71645adae6051c0",
                "Source":
"/var/lib/docker/volumes/acbace259e1214f7d53683d2cc4a3c17788fc11384f45befa71645adae6051c0
/_data",
                "Destination": "/usr/share/nginx",
                "Driver": "local",
                "Mode": "",
                "RW": true,
                "Propagation": ""
            },
            {
                "Type": "volume",
                "Name":
"2d3b6391fab022fea11f735ee0f821f8332413359c4ea41e415842159bd2d0ad",
                "Source":
```

```
"/var/lib/docker/volumes/2d3b6391fab022fea11f735ee0f821f8332413359c4ea41e415842159bd2d0ad
/_data",
                "Destination": "/var/log/nginx",
                "Driver": "local",
                "Mode": "",
                "RW": true,
                "Propagation": ""
            }
        ],
. . .
```

```
[root@mars ~]# docker  run  -it  --name os  --volumes-from web  alpine
/ # ls /var/log/nginx
access.log  error.log
/ # ls /usr/share/nginx/html
50x.html    index.html
/ # exit
[root@mars ~]#
```

Docker et le stockage

Les volumes

docker volume { create | inspect | ls | rm | prune }

```
docker volume create datas

-v datas:/mnt centos          -v datas:/mnt:ro

docker volume create --driver local             \
                     --opt type=tmpfs           \
                     --opt device=tmpfs         \
                     --opt o=size=100m,uid=1000 \
                data_tmp
```

Les volumes

Les volumes vont permettre :

- d'assurer la persistance des données d'un conteneur.
- à plusieurs conteneurs d'accéder aux mêmes données.

Syntaxe :

docker volume { create | inspect | ls | rm | prune }

```
[root@poste]# docker volume create datas
datas
```

```
[root@poste]# docker volume ls
local               data_tmp
local               datas
```

Sur le poste hôte, le contenu du volume est accessible à :

/var/lib/docker/volumes/datas/_data/

Démonstration :

```
[root@poste]# docker  run  -it  --name appli1  -v datas:/mnt  centos

[root@be5e78ba9325 /]# vi  /mnt/fichier1
[root@be5e78ba9325 /]# cat  /mnt/fichier1
bonjour
[root@be5e78ba9325 /]#


[root@poste]# docker  run  -it  --name appli2  -v datas:/mnt  ubuntu

root@4c27632eb043:/# ls /mnt
fichier1
root@4c27632eb043:/# cat /mnt/fichier1
bonjour
```

On peut positionner des options au moment du montage :

```
[root@poste]# docker run -it --rm --name appli1  -v datas:/mnt:ro  centos

[root@8d42a6afad7e /]# ls /mnt
base_personnel.sql  base_societes.sql  donnees  fichier1

[root@8d42a6afad7e /]# touch /mnt/fichier2
touch: cannot touch '/mnt/fichier2': Read-only file system
[root@8d42a6afad7e /]#
```

Pour identifier le ou les conteneur(s) utilisant un volume spécifique :

```
# docker  ps  --filter  volume=Nom_du_Volume
```

Création de volumes spécifiques :

```
# docker volume create --driver local \
                       --opt type=xfs \
                       --opt device=/dev/sda2 \
                  volume_xfs

# docker volume create --driver local \
                       --opt type=nfs \
                       --opt o=addr=192.168.1.1,rw \
                       --opt device=:/path/to/dir \
                  volume_nfs
```

```
[root@poste]# docker volume create --driver local \
                                   --opt type=tmpfs \
                                   --opt device=tmpfs \
                                   --opt o=size=100m,uid=1000 \
                              data_tmp
```

Attention : comme on utilise tmpfs. Les données sont supprimées à l'arrêt du conteneur.

```
[root@poste]# docker volume ls
local               data_tmp
```

```
[root@poste]# docker run -it --rm --name appli1 -v data_tmp:/mnt centos

[root@37856175a288 /]# cd /mnt
[root@37856175a288 mnt]# dd if=/dev/zero of=fichier1 bs=1M count=80
80+0 records in
80+0 records out
83886080 bytes (84 MB) copied, 0.150562 s, 557 MB/s

[root@37856175a288 mnt]# df -h
Filesystem                    Size  Used Avail Use% Mounted on
overlay                        27G  8.8G   19G  33% /
tmpfs                          64M     0   64M   0% /dev
tmpfs                         920M     0  920M   0% /sys/fs/cgroup
tmpfs                         100M   80M   20M  80% /mnt
/dev/mapper/centos_mars-root   27G  8.8G   19G  33% /etc/hosts
shm                            64M     0   64M   0% /dev/shm
tmpfs                         920M     0  920M   0% /proc/scsi
tmpfs                         920M     0  920M   0% /sys/firmware
```

Docker et le stockage

L'option mount

```
$ docker  run  -d  --name test \
            --mount  source=datas,target=/usr/share/html \
          nginx:latest
```

L'option mount

L'option --mount est plus récente que l'option -v. Sa syntaxe est le plus lisible que -v.
Lorsque l'on utilise des services, seule l'option --mount est supportée.

```
$ docker  run  -d  --name test \
            --mount  source=datas,target=/usr/share/html \
         nginx:latest
```

L'exemple ci-dessus est équivalent à :

```
$ docker  run  -d  --name test \
            -v  datas:/usr/share/html \
          nginx:latest
```

Exemple de l'intégration d'un driver de stockage :
L'exemple ci-dessous installe le driver vieux/sshfs pour la création d'un volume via un accès SSH.

```
[root@mars ~]# docker plugin install --grant-all-permissions vieux/sshfs
latest: Pulling from vieux/sshfs
52d435ada6a4: Download complete
Digest: sha256:1d3c3e42c12138da5ef7873b97f7f32cf99fb6edde75fa4f0bcf9ed277855811
Status: Downloaded newer image for vieux/sshfs:latest
Installed plugin vieux/sshfs
[root@mars ~]#
```

Utilisation de ce driver :

```
[root@mars ~]#
[root@mars ~]# docker volume create --driver vieux/sshfs \
                            -o sshcmd=user1@192.168.0.15:/home/user1 \
                            -o password=user1 \
                       volumessh
volumessh
[root@mars ~]#
[root@mars ~]# docker volume ls
DRIVER              VOLUME NAME
vieux/sshfs:latest  volumessh
[root@mars ~]#
[root@mars ~]# docker  volume  inspect  volumessh
[
    {
        "CreatedAt": "0001-01-01T00:00:00Z",
        "Driver": "vieux/sshfs:latest",
        "Labels": {},
        "Mountpoint": "/mnt/volumes/d33a6ab416d2b313de50f5d3563560ce",
        "Name": "volumessh",
        "Options": {
            "password": "user1",
            "sshcmd": "user1@192.168.0.15:/home/user1"
        },
        "Scope": "local"
    }
]
[root@mars ~]#
```

On peut créer le volume (monvol) en même temps que la création du conteneur :

```
[root@mars ~]# docker run -dit --rm --name os \
                  --mount type=volume,volume-driver=vieux/sshfs,\
                    src=monvol,target=/test,\
                    volume-opt=sshcmd=user1@192.168.0.15:/home/user1/rep,\
                    volume-opt=password=user1 \
                 alpine
```

Docker et le stockage

Les volumes : inspect

```
# docker  volume  inspect  data_tmp
[
    {
        "CreatedAt": "2018-05-07T12:49:14+02:00",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/data_tmp/_data",
        "Name": "data_tmp",
        "Options": {
            "device": "tmpfs",
            "o": "size=100m,uid=1000",
            "type": "tmpfs"
        },
        "Scope": "local"
    }
]
```

Les volumes : inspect

```
[root@poste]# docker  volume  inspect  datas
[
    {
        "CreatedAt": "2018-05-07T12:36:16+02:00",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/datas/_data",
        "Name": "datas",
        "Options": {},
        "Scope": "local"
    }
]
```

```
[root@poste]# docker  volume  inspect  data_tmp
[
    {
        "CreatedAt": "2018-05-07T12:49:14+02:00",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/data_tmp/_data",
        "Name": "data_tmp",
        "Options": {
            "device": "tmpfs",
            "o": "size=100m,uid=1000",
            "type": "tmpfs"
        },
        "Scope": "local"
    }
]
```

Docker et le stockage

Les volumes : suppression

```
# docker  volume  rm  data_tmp

# docker  volume  prune
```

Les volumes : suppression

On peut supprimer un volume à condition qu'il ne soit associé à aucun conteneur (même arrêté).

```
[root@poste]# docker  volume  rm  data_tmp
data_tmp
```

Le "Grand Nettoyage" : prune

```
[root@poste]# docker  volume  prune
WARNING! This will remove all local volumes not used by at least one container.
Are you sure you want to continue? [y/N] y
Deleted Volumes:
b2311e15a0c5ceac31771bb13784f766ef295311a892a505670c0de156a8c8ff
datas
. . .
812f12cdfe08afd53342d94c960d8d3f780d4446fb29a844c3372d2e9898d5c6
9b456d380b2da0f2123dee1181eaa51cdabe3743d14a24a8aa5a2278a288a3d3

Total reclaimed space: 525.72Mb
```

Docker et le stockage

Le stockage des images

- Docker Hub
- Autres registry sur le Cloud
- Un registry privé
- Une archive tar

Le stockage des images

Il existe différentes possibilités pour stocker ses images :

1/ Docker Hub

C'est le registry par défaut.

On peut créer un compte directement sur Docker Hub.

```
# docker  login                 pour s'authentifier avec un accès par login et mot de passe.
# docker  logout                pour se déconnecter.

# docker  push   image:tag      pour exporter son image vers le registry.
# docker  pull   image:tag      pour importer son image en local.
```

2/ Autres registry sur le Cloud

Il existe également des registry disponible sur le Cloud. Elles sont payantes mais proposent des services intéressants pour les entreprises.

```
# docker login  serveur
```

3/ Un registry privé

On peut avoir son propre registry au sein de son entreprise.
L'intérêt est de centraliser les images sur un serveur.

4/ Une archive tar

Au lieu de proposer une image, on peut mettre à disposition une archive tar de notre image.

docker image save -o appli1.tar jmb/appli1

docker image import appli1.tar

Docker et le stockage

Les registry privés

docker run -d --name registre -p 90:5000 registry:2

tag

push

logs

Les registry privés

La création d'un registre privé s'appuie sur l'image registry disponible sur Docker Hub.

```
# docker run -d --name registre -p 90:5000 registry:2
```

Pousser des images dans le registre : tag & push

```
# docker tag image_source host_du_registry:port_registry/nom_image
# docker push host_du_registry:port_registry/nom_image
```

Pour suivre l'activité du registry : logs

```
# docker logs nom_du_registre
```

Le protocole pour les 'registry privés' est OpenSource. Il existe donc des solutions qui ont été développées qui intègrent des fonctionnalités supplémentaires, telle qu'une interface graphique de gestion (exemple : Portus).

Démonstration :

```
# docker  run  -d  --name registre  -p 5000:5000  registry:2
```

on accède au registry via le host par : http://localhost:5000/v2

```
# docker  tag  salut  localhost:5000/salut
```

```
# docker  push  localhost:5000/salut
The push refers to repository [localhost:5000/salut]
080d1212cbb4: Pushed
latest: digest: sha256:a6d71c80b3c8a908a3dd3e376cfc02d688996886cc4dad4118e31297c6a3b4af
size: 524
```

on accède au détail sur le registry par : http://localhost:5000/v2/salut/tags/list
http://localhost:5000/v2/_catalog

Download d'une image :

```
# docker  pull  localhost:5000/salut
```

Remarques :

1/ Si le conteneur de registry est supprimé, on perd tout car les données ne sont pas persistantes.
Il serait bon d'exploiter des volumes pour en assurer la persistance.

```
$ docker run -d \
           -p 5000:5000 \
           --restart=always \
           --name registry \
           -v /mnt/registry:/var/lib/registry \
      registry:2
```

2/ Il n'y a pas d'accès sécurisé, tout le monde peut y accéder.
Il serait bon de sécuriser l'accès.
Veuillez vous référer à la procédure indiquée sur le site de documentation de Docker.

Notes

Docker Compose

Dans ce chapitre nous allons étudier la gestion des applications utilisant plusieurs conteneurs.

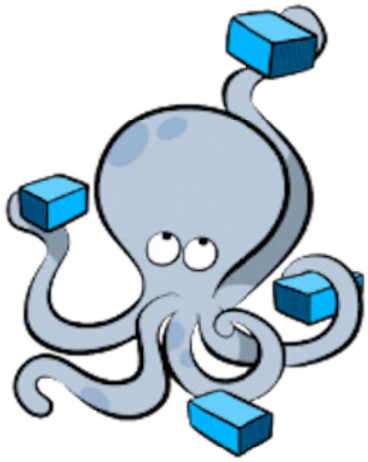
Docker Compose

- Présentation et installation de Docker Compose
- Les fichiers YAML configuration
- La commande docker-compose
- Le déploiement multi conteneurs

Docker Compose

Présentation et installation de Docker Compose

environnement
multi conteneurs
qui
interagissent



Présentation et installation de Docker Compose

Docker Compose permet de créer en une opération un environnement multi conteneurs qui interagissent. Docker Compose est un outil d'orchestration pour Docker qui définit un ensemble de conteneurs et leurs interdépendances sous la forme d'un fichier YAML.

Vous pouvez ensuite utiliser Docker Compose pour afficher une partie ou la totalité de votre pile d'applications, suivre la sortie de l'application, etc.

On peut configurer autant de conteneurs que l'on souhaite, comment ils doivent être construits et connectés, et où les données doivent être stockées. Lorsque le fichier YAML est terminé, on exécute une seule commande pour générer, exécuter et configurer tous les conteneurs.

Il est donc possible de déployer plusieurs micro services (conteneurs) qui interagissent entre eux.

Le fichier yaml permet d'homogénéiser le déploiement d'une application complexe, de définir les dépendances des différents services. Le processus est « industrialisé ».

Installation de Docker Compose

Les procédures pour installer Docker Compose sont disponibles pour Mac, Windows et Linux sur le site de documentation Docker :
https://docs.docker.com/compose/install/#install-compose

Pour Linux :

Veillez à déployer la dernière version de Docker Compose, se référer à la page suivante :
https://github.com/docker/compose/releases

```
# url_download="https://github.com/docker/compose/releases/download"

# curl -L ${url_download}/1.22.0/docker-compose-$(uname -s)-$(uname -m) \
       -o /usr/local/bin/docker-compose

# chmod  +x  /usr/local/bin/docker-compose
```

Ou via pip :

```
# yum  install  python-pip
# pip  install  docker-compose
```

Vérification :

```
# docker-compose --version
docker-compose version 1.22.0, build f46880fe

# docker-compose version
docker-compose version 1.22.0, build f46880fe
docker-py version: 3.4.1
CPython version: 3.6.6
OpenSSL version: OpenSSL 1.10.f  25 May 2017
```

Désinstallation de Docker Compose :

```
# rm  /usr/local/bin/docker-compose
```

ou

```
# pip  uninstall  docker-compose
```

Docker Compose

Les fichiers YAML de configuration

➓ docker-compose.yml ou docker-compose.yaml

```
services:                    définition des services
  nginx:                          définition d'UN service
    image: nginx
    container_name: web
    environment:
      - ville=Paris
      - pays=France
    volumes:
      - vol_datas:/usr/share/nginx/html
    networks:
      - net_web
    ports:
      - "80:80"
```

Les fichiers YAML configuration

Le fichier de configuration doit s'appeler docker-compose.yml ou docker-compose.yaml.

Attention au nom du répertoire où se trouve votre projet, son nom servira de préfixe lors de la création de volumes, réseaux, etc.

```
[root@poste projetbase]# cat  docker-compose.yml

version: '3'                      indique la version de yaml

volumes:                          définition des volumes
  vol_datas:

networks:                         définition des réseaux
  net_web:

services:                         définition des services
  nginx:                                définition d'UN service
    image: nginx                        image à utiliser
    container_name: web                 nom du conteneur
    environment:                        paramétrage des variables du conteneur
      - ville=Paris
      - pays=France
    volumes:                            paramétrage des volumes du conteneur
      - vol_datas:/usr/share/nginx/html
    networks:                           paramétrage des réseaux du conteneur
      - net_web
    ports:                              paramétrage des ports du conteneur
      - "80:80"
```

La commande suivante permet de valider le fichier yaml et de le visualiser :

```
[root@poste projetbase]# docker-compose  config

networks:
  net_web: {}
services:
  nginx:
    container_name: web
    environment:
      pays: France
      ville: Paris
    image: nginx
    networks:
      net_web: null
    ports:
    - 80:80/tcp
    volumes:
    - vol_datas:/usr/share/nginx/html:rw
version: '3.0'
volumes:
  vol_datas: {}
```

Quelques mots clefs :

```
web:
  # construction en indiquant le répertoire du fichier Dockerfile
  build: .

  # build à partir d'une image
  image: centos           ou     image: centos:latest
  image: jmb/monimg       ou     image: mon_registry:5000/monimg

  # définir des variables à partir d'un fichier
  env_file: .env
  env_file: [.env, .development.env]

  # expose des ports réseaux pour linker des services (pas des hosts)
  expose: ["5000"]

  # exécution d'une commande
  command: bundle exec thin -p 3000
  command: [bundle, exec, thin, -p, 3000]

  # surcharge d'entrypoint
  entrypoint: /app/start.sh
  entrypoint: [php, -d, vendor/bin/phpunit]


  # link des services, fait que `db` soit disponible sous l'alias `database`
  links:
    - db:database
    - redis

  # s'assure que `db` fonctionne avant le démarrage du service
  depends_on:
    - db
```

Docker Compose

La commande docker-compose

```
[root@mars projetbase]# docker-compose  up  -d
Creating network "projetbase_net_web" with the default driver
Creating volume "projetbase_vol_datas" with default driver
Creating web ... done
```

```
# docker-compose    ps
                        top
                        images
                        start  |  stop  |  restart
                        logs

                        down

          Options : -p   -f
```

La commande docker-compose

```
# docker-compose --help
Define and run multi-container applications with Docker.

Usage:
  docker-compose [-f <arg>...] [options] [COMMAND] [ARGS...]
  docker-compose -h|--help

Options:
  -f, --file FILE             Specify an alternate compose file
                              (default: docker-compose.yml)
  -p, --project-name NAME     Specify an alternate project name
                              (default: directory name)
  --verbose                   Show more output
  --log-level LEVEL           Set log level (DEBUG, INFO, WARNING, ERROR, CRITICAL)
  --no-ansi                   Do not print ANSI control characters
  -v, --version               Print version and exit
  -H, --host HOST             Daemon socket to connect to

  --tls                       Use TLS; implied by --tlsverify
  --tlscacert CA_PATH         Trust certs signed only by this CA
  --tlscert CLIENT_CERT_PATH  Path to TLS certificate file
  --tlskey TLS_KEY_PATH       Path to TLS key file
  --tlsverify                 Use TLS and verify the remote
  --skip-hostname-check       Don't check the daemon's hostname against the
                              name specified in the client certificate
  --project-directory PATH    Specify an alternate working directory
                              (default: the path of the Compose file)
  --compatibility             If set, Compose will attempt to convert deploy
                              keys in v3 files to their non-Swarm equivalent

Commands:
  build              Build or rebuild services
  bundle             Generate a Docker bundle from the Compose file
  config             Validate and view the Compose file
```

```
  create              Create services
  down                Stop and remove containers, networks, images, and volumes
  events              Receive real time events from containers
  exec                Execute a command in a running container
  help                Get help on a command
  images              List images
  kill                Kill containers
  logs                View output from containers
  pause               Pause services
  port                Print the public port for a port binding
  ps                  List containers
  pull                Pull service images
  push                Push service images
  restart             Restart services
  rm                  Remove stopped containers
  run                 Run a one-off command
  scale               Set number of containers for a service
  start               Start services
  stop                Stop services
  top                 Display the running processes
  unpause             Unpause services
  up                  Create and start containers
  version             Show the Docker-Compose version information
```

On retrouve des fonctionnalités similaires à la commande docker.

```
# docker-compose  up  --help
    -d, --detach          Detached mode: Run containers in the background,
    --quiet-pull          Pull without printing progress information
    --no-deps             Don't start linked services.
    --force-recreate      Recreate containers even if their configuration
                          and image haven't changed.
    --no-start            Don't start the services after creating them.
       . . .
```

Utilisation de la commande avec le fichier yaml précédent :

```
[root@mars projetbase]# docker-compose  up  -d
Creating network "projetbase_net_web" with the default driver
Creating volume "projetbase_vol_datas" with default driver
Creating web ... done
```

L'image nginx est transférée si nécessaire. Le volume, le réseau et le conteneur sont créés. Le conteneur 'web' est démarré.

```
[root@poste projetbase]# docker ps -a
CONTAINER ID    IMAGE      COMMAND                  CREATED             STATUS
PORTS                  NAMES
21e6929ddeee    nginx      "nginx -g 'daemon of…"   2 minutes ago       Up 2 minutes
0.0.0.0:80->80/tcp     web
```

```
[root@poste projetbase]# docker volume ls
DRIVER              VOLUME NAME
local               projetbase_vol_datas
```

```
[root@poste projetbase]# docker network ls
NETWORK ID          NAME                  DRIVER              SCOPE
8981e88724ed        projetbase_net_web    bridge              local
```

Remarque :

Attention pour que la commande fonctionne, il faut être positionné sur le répertoire du projet.
Dans le cas contraire, vous aurez le message suivant :

```
[root@poste  autre_repertoire]# docker-compose  ps
ERROR:
        Can't find a suitable configuration file in this directory or any
        parent. Are you in the right directory?

        Supported filenames: docker-compose.yml, docker-compose.yaml
```

Quelques exemples de la commande docker-compose :

```
[root@poste projetbase]# docker-compose  images
Container   Repository    Tag       Image Id      Size
------------------------------------------------------
web         nginx        latest   ae513a47849c   104 MB
```

```
[root@poste projetbase]# docker-compose  top
[root@poste projetbase]# docker-compose  top  nginx         un service
web
UID    PID    PPID   C   STIME   TTY   TIME                     CMD
------------------------------------------------------------------------------
root   3564   3549   0   11:52   ?   00:00:00  nginx: master process nginx -g daemon off;
101    3612   3564   0   11:52   ?   00:00:00  nginx: worker process
```

```
[root@poste projetbase]# docker-compose  ps
[root@poste projetbase]# docker-compose  ps  nginx          un service
Name         Command          State         Ports
-------------------------------------------------------
web    nginx -g daemon off;   Up      0.0.0.0:80->80/tcp
```

```
[root@poste projetbase]# docker-compose  stop  nginx        un service
Stopping web ... done
```

```
[root@poste projetbase]# docker-compose  start  nginx
Starting nginx ... done
```

```
[root@poste projetbase]# docker-compose  restart  nginx
Restarting web ... done
```

```
[root@poste projetbase]# docker-compose  logs
[root@poste projetbase]# docker-compose  logs  nginx        un service
Attaching to web
web      | 172.18.0.1 - - [08/May/2018:11:52:38 +0000] "GET / HTTP/1.1" 200 612 "-"
"Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0" "-"
web      | 2018/05/08 11:52:38 [error] 5#5: *1 open() "/usr/share/nginx/html/favicon.ico"
failed (2: No such file or directory), client: 172.18.0.1, server: localhost, request:
"GET /favicon.ico HTTP/1.1", host: "172.18.0.2"
web      | 172.18.0.1 - - [08/May/2018:11:52:38 +0000] "GET /favicon.ico HTTP/1.1" 404
170 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0" "-"
```

Arrêt des services du projet :

```
[root@poste projetbase]# docker-compose  stop
```

Le contenu du site est localisé au sein du volume :

```
# ls  /var/lib/docker/volumes/projetbase_vol_datas/_data
50x.html  index.html
```

Suppression des images et conteneurs d'un projet :

```
[root@poste projetbase]# docker-compose  down
Stopping web ... done
Removing web ... done
Removing network projetbase_net_web
```

```
# docker  system  prune  --volumes
WARNING! This will remove:
        - all stopped containers
        - all networks not used by at least one container
        - all volumes not used by at least one container
        - all dangling images
        - all build cache
Are you sure you want to continue? [y/N] y
Deleted Volumes:
projetbase_vol_datas

Total reclaimed space: 1.177kB
```

ou : Suppression des images et des volumes

```
[root@poste projetbase]# docker-compose  down  --volumes
```

```
[root@poste projetbase]# docker-compose  images
Container   Repository   Tag   Image Id   Size
------------------------------------------------
```

```
[root@poste projetbase]# docker-compose  ps
Name   Command   State   Ports
------------------------------------------
```

L'option -p :

Cette option définie le nom du projet. Par défaut le nom du projet est le nom du répertoire dans lequel est exécutée la commande docker-compose.

```
[root@mars projet_site]# docker-compose  -p projet1  up -d
Creating network "projet1_default" with driver "bridge"
Creating projet1_phpfpm_1 ... done
Creating projet1_web_1        ... done
Creating projet1_phpmyadmin_1 ... done
```

L'option -f :

Cette option de spécifier un fichier docker-compose différent des fichiers par défaut :
Les fichiers par défaut sont en docker-compose.yml puis, si présent, docker-compose.override.yml.

```
[root@mars projet_site]# docker-compose  -f  production.yml   up -d
```

L'intérêt pratique de cette option est de pouvoir exploiter plusieurs fichiers yml pour générer le projet. Les fichiers seront pris en compte dans l'ordre apparition au sein de la ligne de commande. Il est possible de rajouter des services ou de modifier le comportement d'un service existant.

Exemple :

docker-compose-base.yml
Ce fichier contient la configuration principale.

docker-compose-prod.yml
Ce fichier contient le delta pour l'environnement de production.
Possibilité de définir des variables spécifiques pour la production (réseaux, volumes, …)
Possibilité de modifier la configuration d'un service défini par le fichier de « base ».
Possibilité d'ajouter des services spécifiques pour la production .

docker-compose -f docker-compose-base.yml -f docker-compose-prod.yml up -d

docker-compose-dev.yml
Ce fichier contient le delta pour l'environnement de développement.
Même réflexion que pour le fichier de production.

docker-compose -f docker-compose-base.yml -f docker-compose-dev.yml up -d

docker-compose-sauvegarde.yml
Ce fichier contient le delta pour une action spécifique de sauvegarde.

docker-compose -f docker-compose-base.yml -f docker-compose-sauvegarde.yml up -d

Docker Compose

Le déploiement multi conteneurs

- Projet wordpress : mysql et wordpress
- Projet nginx-php : nginx et php

Le déploiement multi conteneurs

Un projet wordpress : mysql et wordpress

Le déploiement de l'application wordpress avec sa base de données.
On s'intéressera tout particulièrement aux dépendances entres les deux conteneurs (depends_on, les variables).
On notera également l'utilisation d'un volume et de restart.

Un projet nginx-php : nginx et php

L'intérêt de ce projet est de mettre en pratique le build d'une image via Docker Compose.

Un projet wordpress : mysql et wordpress

Le déploiement de l'application wordpress avec sa base de données.
On notera également l'utilisation d'un volume et de restart.

depends_on : permet de lister les services qui doivent être démarrés au préalable, en respectant l'ordre dans lequel ils sont spécifiés.

Le lien entre les différents services se font via le réseau commun entre eux. Il est préconisé d'éviter d'utiliser le réseau par défaut, mais de créer un réseau propre à ces services.

```
# tree  wordpressdb
wordpressdb
└── docker-compose.yml
```

```
[wordpressdb]# more  docker-compose.yml
version: '3'

services:
   bdd:
     image: mysql:5.7
     volumes:
       - bdd_data:/var/lib/mysql
     restart: always
     environment:
       MYSQL_ROOT_PASSWORD: somewordpress
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: wordpress

   wordpress:
     depends_on:
       - bdd
     image: wordpress:latest
     ports:
       - "8000:80"
     restart: always
     environment:
       WORDPRESS_DB_HOST: bdd:3306
       WORDPRESS_DB_USER: wordpress
       WORDPRESS_DB_PASSWORD: wordpress
volumes:
    bdd_data:
```

```
[wordpressdb]# docker-compose  up  -d
Creating network "wordpressdb_default" with the default driver
Creating volume "wordpressdb_bdd_data" with default driver
Creating wordpressdb_bdd_1 ... done
Creating wordpressdb_wordpress_1 ... done
```

http://localhost:8000 et ça fonctionne !

```
[wordpressdb]# docker-compose  down
Stopping wordpressdb_wordpress_1 ... done
Stopping wordpressdb_bdd_1       ... done
Removing wordpressdb_wordpress_1 ... done
Removing wordpressdb_bdd_1       ... done
Removing network wordpressdb_default
```

Un projet nginx-php : nginx et php

L'intérêt de ce projet est de mettre en pratique le build d'une image via Docker Compose.

Links : permet de lier un conteneur avec d'autres services.
Attention ce paramètre est devenu obsolète, il est équivalent à depends_on.

```
# tree  buildnginx
buildnginx/
├── docker-compose.yml
├── nginx
│   ├── conf
│   │   └── default
│   └── Dockerfile
└── site
    └── index.php
```

La page du site :

```
[root@mars buildnginx]# cat site/index.php
<?php
phpinfo();
```

Le projet nginx, avec son fichier de configuration :

```
[root@mars buildnginx]# cat nginx/conf/default
server {
  listen 80;
  root /usr/share/nginx/html;
  index index.php index.html index.html;
  server_name 127.0.0.1;
  location / {
    try_files $uri /index.php$is_args$args;
  }
  location ~ \.php$ {
    fastcgi_split_path_info ^(.+\.php)(/.+)$;
    fastcgi_pass phpfpm:9000;
    fastcgi_index index.php;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    include fastcgi_params;
  }
}
[root@mars buildnginx]#
```

Le projet nginx, avec son fichier Dockerfile :

```
[root@mars buildnginx]# cat nginx/Dockerfile
FROM tutum/nginx

COPY conf/default /etc/nginx/sites-available/default
COPY conf/default /etc/nginx/sites-enabled/default

EXPOSE 80 443

[root@mars buildnginx]#
```

Le fichier docker-compose.yml :

```
[root@mars buildnginx]# cat docker-compose.yml
version: '3'

services:
  nginx:
    build: ./nginx
    ports:
      - "80:80"
      - "443:443"
    links:
      - phpfpm
    volumes:
      - ./site:/usr/share/nginx/html

  phpfpm:
    image: php:fpm
    ports:
      - "9000:9000"
    volumes:
      - ./site:/usr/share/nginx/html

[root@mars buildnginx]#
```

```
[root@mars buildnginx]# docker-compose up -d
Creating network "buildnginx_default" with the default driver
Building nginx
Step 1/4 : FROM tutum/nginx
 ---> a2e9b71ed366
Step 2/4 : COPY conf/default /etc/nginx/sites-available/default
 ---> 8c99cb9cc3ac
Step 3/4 : COPY conf/default /etc/nginx/sites-enabled/default
 ---> 14bee41b6d8d
Step 4/4 : EXPOSE 80 443
 ---> Running in 0ba807bf2795
Removing intermediate container 0ba807bf2795
 ---> 9c64c3e1c732
Successfully built 9c64c3e1c732
Successfully tagged buildnginx_nginx:latest
WARNING: Image for service nginx was built because it did not already exist. To rebuild
this image you must use `docker-compose build` or `docker-compose up --build`.
Creating buildnginx_phpfpm_1 ... done
Creating buildnginx_nginx_1 ... done
[root@mars buildnginx]#
```

```
[root@mars buildnginx]# docker-compose ps
Name                 Command                          State   Ports
-------------------------------------------------------------------------------
buildnginx_nginx_1   /usr/sbin/nginx                  Up      0.0.0.0:443->443/tcp,
                                                              0.0.0.0:80->80/tcp
buildnginx_phpfpm_1  docker-php-entrypoint php-fpm    Up      0.0.0.0:9000->9000/tcp
[root@mars buildnginx]#
```

Le site fonctionne avec :
http://127.0.0.1 pour la page de garde nginx.
http://127.0.0.1/index.php pour la page d'informations php.

Si on modifie le fichier docker-compose.yml, il suffit de refaire : docker-compose up -d
Si on modifie le fichier Dockerfile, on peut faire 'docker-compose build', mais attention au cache.

Notes

Docker Machine

Dans ce chapitre nous allons étudier l'utilisation de Docker Machine.

Docker Machine

- Présentation et installation
- Création de machines virtuelles
- Utilisation

Docker Machine

Présentation et installation

déploiement et management
de postes « dockerisés »
en local ou distant

des hôtes gérés
via docker-machine

Présentation et installation

Docker Machine est un outil qui vous permet d'installer Docker Engine sur des hôtes virtuels et de gérer les hôtes avec des commandes 'docker-machine'. Vous pouvez utiliser Docker Machine pour créer des hôtes Docker sur votre Mac ou Windows local, sur votre réseau d'entreprise ou sur des fournisseurs de cloud tels qu'Azure ou AWS.

À l'aide des commandes 'docker-machine', vous pouvez démarrer, inspecter, arrêter et redémarrer un « hôte géré », mettre à niveau le client et le démon Docker et configurer un client Docker pour qu'il échange avec votre hôte.

Installation :

Docker Machine s'installe sur macOS, Windows et Linux.

Sous Windows :

Avec boot2docker, docker-machine est déjà préinstallé.
Sinon, il suffit de télécharger l'exécutable de Docker Machine.

On peut également télécharger docker-toolbox. Son installation déploie tout le nécessaire pour Docker, en autres : virtualbox avec une VM Boot2Docker, Docker Client pour Windows, Docker Machine pour Windows, Docker Compose pour Windows, Kitematic pour Windows et git pour Windows. Kitematic est une interface graphique pour installer des applications via Docker.

Sous Linux :

```
# base=https://github.com/docker/machine/releases/download/v0.14.0 \
  && curl -L $base/docker-machine-$(uname -s)-$(uname -m) >/tmp/docker-machine \
  && install /tmp/docker-machine /usr/local/bin/docker-machine
```

Vérification :

```
[root@mars machine]#  docker-machine  version
docker-machine version 0.14.0, build 89b8332
```

La commande docker-machine :

```
[root@mars machine]# docker-machine --help
Usage: docker-machine [OPTIONS] COMMAND [arg...]

Create and manage machines running Docker.

Commands:
  active            Print which machine is active
  config            Print the connection config for machine
  create            Create a machine
  env               Display the commands to set up the environment for the Docker client
  inspect           Inspect information about a machine
  ip                Get the IP address of a machine
  kill              Kill a machine
  ls                List machines
  provision         Re-provision existing machines
  regenerate-certs  Regenerate TLS Certificates for a machine
  restart           Restart a machine
  rm                Remove a machine
  ssh               Log into or run a command on a machine with SSH.
  scp               Copy files between machines
  mount             Mount or unmount a directory from a machine with SSHFS.
  start             Start a machine
  status            Get the status of a machine
  stop              Stop a machine
  upgrade           Upgrade a machine to the latest version of Docker
  url               Get the URL of a machine
  version           Show the Docker Machine version or a machine docker version
  help              Shows a list of commands or help for one command
```

Docker Machine

Création de machines virtuelles

Les drivers

docker-machine create --driver virtualbox vm1

docker-machine rm vm1

Création de machines virtuelles

Le driver :

Le pilote (driver) sert de connecteur à des services tiers tels que Azure, Amazon, etc.
Le driver 'virtualbox' est utilisé pour une machine virtuelle Virtualbox, de même le driver 'hyperv' pour une machine virtuelle sous HyperV.

La commande 'docker-machine create' :

```
docker-machine create --driver virtualbox vm1
```

Cette commande télécharge une distribution Linux légère (boot2docker) avec le démon Docker installé. Elle crée et démarre une VM VirtualBox avec Docker en cours d'exécution.

```
$ docker-machine   create   --driver  virtualbox   poste1
Running pre-create checks...
Creating machine...
(poste1) Copying C:\Users\Baranger\.docker\machine\cache\boot2docker.iso to
C:\Users\Baranger\.docker\machine\machines\poste1\boot2docker.iso...
(poste1) Creating VirtualBox VM...
(poste1) Creating SSH key...
(poste1) Starting the VM...
(poste1) Check network to re-create if needed...
(poste1) Waiting for an IP...
Waiting for machine to be running, this may take a few minutes...
Detecting operating system of created instance...
Waiting for SSH to be available...
Detecting the provisioner...
Provisioning with boot2docker...
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...
Checking connection to Docker...
Docker is up and running!
To see how to connect your Docker Client to the Docker Engine running on this virtual
machine, run:
C:\Program Files\Docker Toolbox\docker-machine.exe env poste1
```

```
$ docker-machine  ls
NAME      ACTIVE   DRIVER       STATE     URL                         SWARM DOCKER        ERRORS
default   *        virtualbox   Running   tcp://192.168.99.100:2376         v18.05.0-ce
poste1    -        virtualbox   Running   tcp://192.168.99.101:2376         v18.05.0-ce
```

Pour afficher l'IP d'une machine :

```
$ docker-machine ip poste1
 192.168.99.101
```

Pour afficher l'url d'une machine :

```
$ docker-machine url poste1
tcp://192.168.99.101:2376
```

Pour supprimer d'une machine :

```
$ docker-machine rm poste1
```

Pour supprimer toutes les machines :

```
docker-machine rm -f $(docker-machine ls -q)
```

Docker Machine

Utilisation

docker-machine {start|stop|restart} machine_name

eval $(docker-machine env poste_distant)
eval $(docker-machine env -u)

docker-machine ssh poste1
docker-machine ssh poste1 hostname

docker-machine {ip | url | status | env | inspect | upgrade} poste1

Utilisation

L'arrêt et démarrage de machines :

docker-machine {start|stop|restart} machine_name

```
$ docker-machine stop poste1
Stopping "poste1"...
Machine "poste1" was stopped.

$ docker-machine ls
NAME      ACTIVE   DRIVER       STATE     URL                         SWARM DOCKER        ERRORS
default   -        virtualbox   Running   tcp://192.168.99.100:2376       v18.05.0-ce
poste1    -        virtualbox   Stopped                                   Unknown


$ docker-machine start poste1
Starting "poste1"...
(poste1) Check network to re-create if needed...
(poste1) Windows might ask for the permission to configure a dhcp server. Sometimes, such
confirmation window is minimized in the taskbar.
(poste1) Waiting for an IP...
Machine "poste1" was started.
Waiting for SSH to be available...
Detecting the provisioner...
Started machines may have new IP addresses. You may need to re-run the `docker-machine
env` command.

$ docker-machine ls
NAME      ACTIVE   DRIVER       STATE     URL                         SWARM DOCKER        ERRORS
default   -        virtualbox   Running   tcp://192.168.99.100:2376       v18.05.0-ce
poste1    *        virtualbox   Running   tcp://192.168.99.101:2376       v18.05.0-ce
```

L'environnement de travail :

Les commandes Docker s'exécutent sur le poste local. Afin de pouvoir les exécuter sur une machine distante, il est nécessaire de positionner l'environnement de travail sur ce poste distant :

```
$ eval "$(docker-machine env poste_distant)"
```

Pour annuler l'environnement distant :

```
$ eval $(docker-machine env -u)
```

Exemple :

```
$ eval "$(docker-machine env poste1)"

$ docker-machine ls
NAME     ACTIVE   DRIVER       STATE     URL                         SWARM DOCKER        ERRORS
.  .  .
poste1   *        virtualbox   Running   tcp://192.168.99.101:2376       v18.05.0-ce
.  .  .

$ docker-machine active
poste1

$ docker-compose up -d
```

Permet de déployer et d'exécuter sur le poste distant le « projet » qui est localisé en local.

Pour exécuter une commande sur une machine :

```
$ docker-machine ssh poste1 hostname
poste1
```

Pour se connecter à une machine :

```
$ docker-machine ssh poste1
                        ##         .
                  ## ## ##        ==
               ## ## ## ## ##    ===
           /"""""""""""""""""\___/ ===
      ~~~ {~~ ~~~~ ~~~ ~~~~ ~~~ ~ /  ===- ~~~
           \_______ o           __/
             \    \         __/
              \____\_______/
 _                 _   ____     _            _
| |__   ___   ___ | |_|___ \ __| | ___   ___| | _____ _ __
| '_ \ / _ \ / _ \| __| __) / _` |/ _ \ / __| |/ / _ \ '__|
| |_) | (_) | (_) | |_ / __/ (_| | (_) | (__|   <  __/ |
|_.__/ \___/ \___/ \__|_____\__,_|\___/ \___|_|\_\___|_|
Boot2Docker version 18.05.0-ce, build HEAD : b5d6989 - Thu May 10 16:35:28 UTC 2018
Docker version 18.05.0-ce, build f150324
docker@poste1:~$ hostname
poste1
docker@poste1:~$ pwd
/home/docker
docker@poste1:~$ id
uid=1000(docker) gid=50(staff) groups=50(staff),100(docker)
docker@poste1:~$ exit
```

Pour copier un fichier du poste local vers la machine :

```
$ docker-machine  scp  ~/localfile  machine:~/
```

Pour copier un fichier de la machine vers le poste local :

```
$ docker-machine  scp  machine:~/machinefile  ~/
```

Pour identifier la machine active :

```
$ docker-machine active
poste1
```

Pour afficher l'IP d'une machine :

```
$ docker-machine ip poste1
 192.168.99.101
```

Pour afficher l'url d'une machine :

```
$ docker-machine url poste1
tcp://192.168.99.101:2376
```

Pour afficher l'état d'une machine :

```
$ docker-machine status poste1
Running
```

Pour afficher l'environnement d'une machine :

```
$ docker-machine env poste1
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.101:2376"
export DOCKER_CERT_PATH="C:\Users\Baranger\.docker\machine\machines\poste1"
export DOCKER_MACHINE_NAME="poste1"
export COMPOSE_CONVERT_WINDOWS_PATHS="true"
# Run this command to configure your shell:
# eval $("C:\Program Files\Docker Toolbox\docker-machine.exe" env poste1)
```

Pour upgrader une machine :

```
$ docker-machine upgrade master
Waiting for SSH to be available...
Detecting the provisioner...
Upgrading docker...
Stopping machine to do the upgrade...
Upgrading machine "master"...
Default Boot2Docker ISO is out-of-date, downloading the latest release...
Latest release for github.com/boot2docker/boot2docker is v18.06.1-ce
Downloading C:\Users\Baranger\.docker\machine\cache\boot2docker.iso from
https://github.com/boot2docker/boot2docker/releases/downl
oad/v18.06.1-ce/boot2docker.iso...
0%....10%....20%....30%....40%....50%....60%....70%....80%....90%....100%
Copying C:\Users\Baranger\.docker\machine\cache\boot2docker.iso to
C:\Users\Baranger\.docker\machine\machines\master\boot2docker.i
so...
Starting machine back up...
(master) Check network to re-create if needed...
(master) Windows might ask for the permission to configure a dhcp server. Sometimes, such
confirmation window is minimized in the taskbar.
(master) Waiting for an IP...
Restarting docker...
```

Pour le détail d'une machine :

```
$ docker-machine inspect poste1
{
    "ConfigVersion": 3,
    "Driver": {
        "IPAddress": "192.168.99.101",
        "MachineName": "poste1",
        "SSHUser": "docker",
        "SSHPort": 11768,
        "SSHKeyPath": "C:\\Users\\Baranger\\.docker\\machine\\machines\\poste1\\id_rsa",
        "StorePath": "C:\\Users\\Baranger\\.docker\\machine",
        "SwarmMaster": false,
        "SwarmHost": "tcp://0.0.0.0:3376",
        "SwarmDiscovery": "",
        "VBoxManager": {},
        "HostInterfaces": {},
        "CPU": 1,
        "Memory": 1024,
        "DiskSize": 20000,
        "NatNicType": "82540EM",
        "Boot2DockerURL": "",
        "Boot2DockerImportVM": "",
        "HostDNSResolver": false,
        "HostOnlyCIDR": "192.168.99.1/24",
        "HostOnlyNicType": "82540EM",
        "HostOnlyPromiscMode": "deny",
        "UIType": "headless",
        "HostOnlyNoDHCP": false,
        "NoShare": false,
        "DNSProxy": true,
        "NoVTXCheck": false,
        "ShareFolder": ""
    },
    "DriverName": "virtualbox",
    "HostOptions": {
        "Driver": "",
        "Memory": 0,
        "Disk": 0,
        "EngineOptions": {
            "ArbitraryFlags": [],
            "Dns": null,
            "GraphDir": "",
            "Env": [],
            "Ipv6": false,
            "InsecureRegistry": [],
            "Labels": [],
            "LogLevel": "",
            "StorageDriver": "",
            "SelinuxEnabled": false,
            "TlsVerify": true,
            "RegistryMirror": [],
            "InstallURL": "https://get.docker.com"
        },
        "SwarmOptions": {
            "IsSwarm": false,
            "Address": "",
            "Discovery": "",
            "Agent": false,
            "Master": false,
            "Host": "tcp://0.0.0.0:3376",
            "Image": "swarm:latest",
            "Strategy": "spread",
            "Heartbeat": 0,
            "Overcommit": 0,
            "ArbitraryFlags": [],
            "ArbitraryJoinFlags": [],
            "Env": null,
            "IsExperimental": false
```

```
        },
        "AuthOptions": {
            "CertDir": "C:\\Users\\Baranger\\.docker\\machine\\certs",
            "CaCertPath": "C:\\Users\\Baranger\\.docker\\machine\\certs\\ca.pem",
            "CaPrivateKeyPath": "C:\\Users\\Baranger\\.docker\\machine\\certs\\ca-
key.pem",
            "CaCertRemotePath": "",
            "ServerCertPath":
"C:\\Users\\Baranger\\.docker\\machine\\machines\\poste1\\server.pem",

            "ServerKeyPath":
"C:\\Users\\Baranger\\.docker\\machine\\machines\\poste1\\server-key.pem",
            "ClientKeyPath": "C:\\Users\\Baranger\\.docker\\machine\\certs\\key.pem",
            "ServerCertRemotePath": "",
            "ServerKeyRemotePath": "",
            "ClientCertPath": "C:\\Users\\Baranger\\.docker\\machine\\certs\\cert.pem",
            "ServerCertSANs": [],
            "StorePath": "C:\\Users\\Baranger\\.docker\\machine\\machines\\poste1"
        }
    },
    "Name": "poste1"
}
```

Notes

Docker Swarm

Dans ce chapitre nous allons étudier l'utilisation de Docker Swarm.

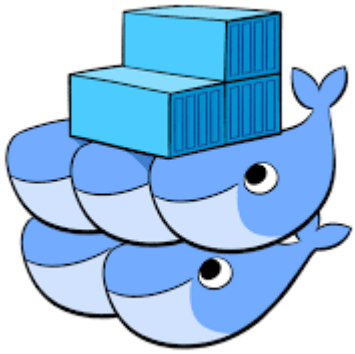
Docker Swarm

- Présentation
- Activer un cluster
- Déployer un service
- Scalabilité et load balancing
- Réseaux et volumes
- Un fichier docker-compose

Docker Swarm

Présentation

Orchestration
de conteneurs
sur plusieurs serveurs



Présentation

Docker Swarm est la composante d'orchestration de Docker.
Docker Swarm permet d'orchestrer des conteneurs, des services, non plus sur un serveur mais répartis sur plusieurs serveurs.

Docker Swarm est intégré à Docker Engine.

Docker Swarm est basé sur une architecture maître / esclave. Chaque cluster contient au moins un nœud (hôte) pour la gestion du cluster (maître) et un certain nombre de nœuds de type « Worker » (esclave). Le maître est responsable de la gestion du cluster et de la répartition des charges. Les esclaves exécutent les unités de travail. Les applications basées sur conteneurs sont distribuées sur les nœuds Docker en tant que service.
Le terme service désigne des tâches individuelles qui sont traitées chacune dans son propre conteneur sur un nœud du cluster.

Conception décentralisée : pour construire un cluster Swarm, on peut utiliser n'importe quelle machine. C'est Docker Engine qui spécifie les maîtres et les esclaves au moment de l'exécution de la commande 'docker init' ou 'docker join'.

Mise à l'échelle (Scaling) : pour chaque service, vous pouvez déclarer le nombre de tâches que vous souhaitez exécuter. Lorsque vous effectuez une augmentation ou une réduction, le manager Swarm s'adapte automatiquement en ajoutant ou en supprimant des tâches pour conserver l'état souhaité.

Réseau multi-hôte : un cluster Swarm utilise un réseau avec le driver overlay. Ce qui permet à des conteneurs sur des nœuds différents de communiquer entre elles. Le manager swarm attribue automatiquement des adresses aux conteneurs du réseau overlay lorsqu'il initialise ou met à jour l'application.

Découverte de services : le manager du cluster swarm assigne à chaque service du cluster un nom DNS unique et équilibre la charge des conteneurs en cours d'exécution. Vous pouvez interroger chaque conteneur exécuté dans le cluster via un serveur DNS intégré au cluster.

Équilibrage de charge : vous pouvez exposer les ports des services à un équilibreur de charge externe. En interne, le cluster vous permet de spécifier le mode de répartition des conteneurs de service entre les nœuds.

Sécurisé par défaut : tous les échanges au sein du cluster sont sécurisés. Vous avez la possibilité d'utiliser des certificats auto-signés ou vos propres certificats.

Mises à jour progressives (Rolling updates) : au moment du déploiement, vous pouvez appliquer des mises à jour de service aux nœuds de manière incrémentale. En cas de problème, vous pouvez restaurer une tâche dans une version précédente du service.

Service répliqué et service global :

Docker Swarm propose deux modes dans lesquels les services sont définis : le service répliqué et le service global.

Service répliqué : c'est une tâche qui s'exécute dans un nombre de réplicas défini par l'utilisateur. Chaque réplica est une instance d'un conteneur. Le redimensionnement est très aisé pour répondre à une augmentation de charge. Un site web peut en une ligne de commande être redimensionné en exécutant 10 ,20 ou 50 réplicas.

Service global : quand un service s'exécute en mode global, tous les nœuds du cluster démarrent une tâche. Si un nœud est ajouté au cluster, la tâche est automatiquement démarrée dessus si les contraintes de ressources et de placement sont respectées. Les services globaux sont utilisés pour la surveillance d'applications ou pour des services anti-virus. Pour créer un service global, utiliser l'option '--mode global'.

Limitation des ressources :

Pour limiter les ressources disponibles du service, les options --reserve-memory et --reserve-cpu peuvent-être utilisées. Si aucun nœud ne peut satisfaire les conditions, le service est dans l'état suspendu.

Si un service réclame plus de mémoire qu'un nœud peut lui fournir, une erreur de type OOME (Out Of Memory Exception) sera alors détecté. Le OOM killer du noyau tuera alors le container ou le démon docker. Pour éviter ce type d'erreur, vérifier que votre application dispose d'assez de ressource mémoire pour s'exécuter.

Les contraintes :

Des contraintes d'emplacement peuvent-être attribuées pour contrôler sur quels nœuds un service doit s'exécuter. Les contraintes peuvent être une région ou le type du système d'exploitation. Utiliser l'option --constraint lors de la création du service pour les utiliser. Les contraintes sont cumulatives (ET logique). Si plusieurs contraintes sont utilisées le service ne se déploiera que sur les nœuds où elles sont toutes satisfaites.

Les préférences :

Alors que les contraintes d'emplacement limitent les nœuds sur lesquels un service peut être exécuté, les préférences de placement tentent de placer les services sur les nœuds appropriés de manière algorithmique (actuellement, elles ne sont réparties que de manière uniforme). Par exemple, si vous attribuez une étiquette de rack à chaque nœud, vous pouvez définir une préférence d'emplacement pour répartir le service de manière uniforme sur les nœuds portant l'étiquette de rack, par valeur. Ainsi, si vous perdez un rack, le service sera toujours exécuté sur des nœuds situés sur d'autres racks.

Les préférences de placement ne sont pas strictement appliquées. Si aucun nœud n'a l'étiquette que vous spécifiez dans votre préférence, le service sera déployé comme si la préférence n'avait pas été définie. L'option --placement-pref permet de spécifier ces préférences. Pour avoir de multiples préférences de placement, utiliser plusieurs fois l'option. L'ordre est important, les préférences de placement sont appliquées dans l'ordre lors de la prise de décision pour la planification.

Lors de la mise à jour en ligne de commande avec docker service update , l'option --placement-pref-add ajoute le nouvel emplacement à la suite de ceux existant. A contrario, l'option --placement-pref-rm supprime le placement existant.

Les ports de swarm :

2377 : pour le management du cluster.
7946 : pour la communication entre les nœuds.
4789 : pour le trafic réseau overlay.

Docker Swarm

Activer un cluster

Activation d'un poste de management – le Leader

```
docker  swarm  init  --advertise-addr 192.168.99.101
```

Intégration d'un poste au noeud du cluster swarm – les Workers:

```
docker  swarm  join  --token SWMTKN...ozm  192.168.99.101:2377
```

Promouvoir un poste en manager

Supprimer un cluster swarm

Activer un cluster

Création des machines :

Préalable pour les manipulations et exemples qui suivent au sein de ce module :

```
$ docker-machine  create  --driver virtualbox  master
$ docker-machine  create  --driver virtualbox  slave
$ docker-machine  create  --driver virtualbox  node1
$ docker-machine  create  --driver virtualbox  node2

$ docker-machine ls
NAME      ACTIVE DRIVER      STATE    URL                         SWARM DOCKER       ERRORS
default   *      virtualbox  Running  tcp://192.168.99.100:2376       v18.05.0-ce
master    -      virtualbox  Running  tcp://192.168.99.101:2376       v18.05.0-ce
node1     -      virtualbox  Running  tcp://192.168.99.103:2376       v18.05.0-ce
node2     -      virtualbox  Running  tcp://192.168.99.104:2376       v18.05.0-ce
slave     -      virtualbox  Running  tcp://192.168.99.102:2376       v18.05.0-ce
```

Création d'un cluster Swarm :

Activation d'un poste de management – le Leader :

```
$ docker-machine ssh master

docker@master:~$ docker  swarm  init  --advertise-addr 192.168.99.101
Swarm initialized: current node (cchvo0h2w4ve8lm8e9ojjm71i) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-
26xfub8oviboh5nwcfrq1plhr6a2zdeyqjv23v13o3qgsp8fqn-f4fokyffk8
095kced59ybuozm 192.168.99.101:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the
instructions.
```

```
docker@master:~$ docker  info
. . .
Swarm: active
 NodeID: cchvo0h2w4ve8lm8e9ojjm71i
 Is Manager: true
 ClusterID: ve58q4u86v2o6psoj5x2nvsw8
 Managers: 1
 Nodes: 1
 Orchestration:
  Task History Retention Limit: 5
 Raft:
  Snapshot Interval: 10000
  Number of Old Snapshots to Retain: 0
  Heartbeat Tick: 1
  Election Tick: 10
 Dispatcher:
  Heartbeat Period: 5 seconds
 CA Configuration:
  Expiry Duration: 3 months
  Force Rotate: 0
 Autolock Managers: false
 Root Rotation In Progress: false
 Node Address: 192.168.99.101
 Manager Addresses:
  192.168.99.101:2377
. . .
```

```
docker@master:~$ docker node ls
ID                            HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS ENGINE VERSION
cchvo0h2w4ve8lm8e9ojjm71i *  master    Ready   Active        Leader         18.05.0-ce
```

```
docker@master:~$ docker  swarm  join-token  worker
To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-
26xfub8oviboh5nwcfrq1plhr6a2zdeyqjv23v13o3qgsp8fqn-f4fokyffk8095kced59ybuozm
192.168.99.101:2377

docker@master:~$ exit
```

Intégration d'un poste au noeud du cluster swarm – les Workers:

```
$ docker-machine  ssh  node1

docker@node1:~$  docker  swarm  join  --token SWMTKN-1-
26xfub8oviboh5nwcfrq1plhr6a2zdeyqjv23v13o3qgsp8fqn-
f4fokyffk8095kced59ybuozm  192.168.99.101:2377
This node joined a swarm as a worker.

docker@node1:~$ docker  info
Swarm: active
 NodeID: 3xvdlirccokq8iqt7sne97h0j
 Is Manager: false
 Node Address: 192.168.99.103
 Manager Addresses:
  192.168.99.101:2377
docker@node1:~$ exit
```

```
$ docker-machine  ssh  node2

docker@node2:~$  docker  swarm  join  --token SWMTKN-1-
26xfub8oviboh5nwcfrq1plhr6a2zdeyqjv23v13o3qgsp8fqn-
f4fokyffk8095kced59ybuozm  192.168.99.101:2377
This node joined a swarm as a worker.
docker@node2:~$ exit
```

```
$ docker-machine  ssh  master
docker@master:~$ docker node ls
ID                           HOSTNAME STATUS  AVAILABILITY  MANAGER STATUS ENGINE VERSION
cchvo0h2w4ve8lm8e9ojjm71i *  master   Ready   Active        Leader         18.05.0-ce
3xvdlirccokq8iqt7sne97h0j     node1   Ready   Active                       18.05.0-ce
482tiakc8tw4yr8tj8zhbnt3p     node2   Ready   Active                       18.05.0-ce
docker@master:~$ exit
```

Promouvoir un poste en manager :

Poste slave d'un cluster Swarm :

- devenir un poste du noeud du cluster.
- Promouvoir le poste en manager.

Le Leader est le gestionnaire principal du cluster qui prend toutes les décisions de gestion et d'orchestration du cluster.
Il est préconisé d'avoir un nombre impair de managers. Si le 'Leader' ne fonctionne plus, un manager 'Reachable' deviendra 'Leader'.

```
$ docker-machine ssh slave

docker@slave:~$  docker  swarm  join  --token  SWMTKN-1-
26xfub8oviboh5nwcfrq1plhr6a2zdeyqjv23v13o3qgsp8fqn-
f4fokyffk8095kced59ybuozm 192.168.99.101:2377
This node joined a swarm as a worker.
docker@slave:~$ exit
```

```
$ docker-machine ssh master

docker@master:~$ docker  node  promote  slave
Node slave promoted to a manager in the swarm.
```

```
docker@master:~$ docker  node  ls
ID                            HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS  ENGINE VERSION
cchvo0h2w4ve8lm8e9ojjm71i *   master    Ready   Active        Leader          18.05.0-ce
3xvdlirccokq8iqt7sne97h0j     node1     Ready   Active                        18.05.0-ce
482tiakc8tw4yr8tj8zhbnt3p     node2     Ready   Active                        18.05.0-ce
4vflg2wiy7hnghv7dsmxzonlo     slave     Ready   Active        Reachable       18.05.0-ce
```

```
docker@master:~$ docker  info
Swarm: active
 NodeID: cchvo0h2w4ve8lm8e9ojjm71i
 Is Manager: true
 ClusterID: ve58q4u86v2o6psoj5x2nvsw8
 Managers: 2
 Nodes: 4
 Orchestration:
  Task History Retention Limit: 5
 Raft:
  Snapshot Interval: 10000
  Number of Old Snapshots to Retain: 0
  Heartbeat Tick: 1
  Election Tick: 10
 Dispatcher:
  Heartbeat Period: 5 seconds
 CA Configuration:
  Expiry Duration: 3 months
  Force Rotate: 0
 Autolock Managers: false
 Root Rotation In Progress: false
 Node Address: 192.168.99.101
 Manager Addresses:
  192.168.99.101:2377
  192.168.99.102:2377
```

Le mot clé demote pour déclasser un maître en esclave.

Une variante pour la création d'un cluster Swarm :

```
$ eval "$(docker-machine env master)"

$ docker swarm init \
        --listen-addr $(docker-machine ip master) \
        --advertise-addr $(docker-machine ip master)


$ token=$(docker swarm join-token -q worker)

$ eval "$(docker-machine env worker1)"
$ docker swarm join --token $token $(docker-machine ip master):2377

$ eval "$(docker-machine env worker2)"
$ docker swarm join --token $token $(docker-machine ip master):2377
```

Supprimer un cluster swarm :

Sur les nœuds du cluster (workers) : docker swarm leave

Sur le Leader : docker node rm <node_name>

docker swarm leave --force

Docker Swarm

Déployer un service

docker service create ….

docker service ps
logs
inspect

Déployer un service

```
$ docker-machine ssh master

docker@master:~$ docker service create \
                 --name service1 alpine ping docker.com
ponbamjrlhdx2n6paeoy22pxx
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service converged
```

```
docker@master:~$ docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE
PORTS
ponbamjrlhdx        service1            replicated          1/1
alpine:latest
```

```
docker@master:~$ docker service ps service1
ID            NAME        IMAGE          NODE   DESIRED STATE  CURRENT STATE
ERROR               PORTS
163lya04w606  service1.1  alpine:latest  node1  Running        Running about a minute ago
```

```
docker@master:~$ docker service logs service1
service1.1.163lya04w606@node1    | PING docker.com (34.232.188.57): 56 data bytes
```

```
docker@master:~$ docker  service  inspect  --pretty service1

ID:             ponbamjrlhdx2n6paeoy22pxx
Name:           service1
Service Mode:   Replicated
 Replicas:      1
Placement:
UpdateConfig:
 Parallelism:   1
 On failure:    pause
 Monitoring Period: 5s
 Max failure ratio: 0
 Update order:      stop-first
RollbackConfig:
 Parallelism:   1
 On failure:    pause
 Monitoring Period: 5s
 Max failure ratio: 0
 Rollback order:    stop-first
ContainerSpec:
 Image:
alpine:latest@sha256:e1871801d30885a610511c867de0d6baca7ed4e6a2573d506bbec7fd3b03873f
 Args:          ping docker.com
Resources:
Endpoint Mode:  vip
```

```
docker@master:~$ docker  service  inspect  service1
[
    {
        "ID": "ponbamjrlhdx2n6paeoy22pxx",
        "Version": {
            "Index": 3746
        },
        "CreatedAt": "2018-06-21T13:38:06.307904051Z",
        "UpdatedAt": "2018-06-21T13:38:06.307904051Z",
        "Spec": {
            "Name": "service1",
            "Labels": {},
            "TaskTemplate": {
                "ContainerSpec": {
                    "Image":
"alpine:latest@sha256:e1871801d30885a610511c867de0d6baca7ed4e6a2573d506bbec7fd3b03873f",
                    "Args": [
                        "ping",
                        "docker.com"
                    ],
                    "StopGracePeriod": 10000000000,
                    "DNSConfig": {},
                    "Isolation": "default"
                },
                "Resources": {
                    "Limits": {},
                    "Reservations": {}
                },
                "RestartPolicy": {
                    "Condition": "any",
                    "Delay": 5000000000,
                    "MaxAttempts": 0
                },
                "Placement": {
                    "Platforms": [
                        {
                            "Architecture": "amd64",
                            "OS": "linux"
                        },
                        {
                            "OS": "linux"
```

```
                        },
                        {
                            "Architecture": "arm64",
                            "OS": "linux"
                        },
                        {
                            "Architecture": "386",
                            "OS": "linux"
                        },
                        {
                            "Architecture": "ppc64le",
                            "OS": "linux"
                        },
                        {
                            "Architecture": "s390x",
                            "OS": "linux"
                        }
                    ]
                },
                "ForceUpdate": 0,
                "Runtime": "container"
            },
            "Mode": {
                "Replicated": {
                    "Replicas": 1
                }
            },
            "UpdateConfig": {
                "Parallelism": 1,
                "FailureAction": "pause",
                "Monitor": 5000000000,
                "MaxFailureRatio": 0,
                "Order": "stop-first"
            },
            "RollbackConfig": {
                "Parallelism": 1,
                "FailureAction": "pause",
                "Monitor": 5000000000,
                "MaxFailureRatio": 0,
                "Order": "stop-first"
            },
            "EndpointSpec": {
                "Mode": "vip"
            }
        },
        "Endpoint": {
            "Spec": {}
        }
    }
]
docker@master:~$
```

Docker Swarm

Scalabilité et load balancing

scale --replicas

```
docker  service  scale  service1=6

$ docker  service  ps  service1
ID            NAME        IMAGE          NODE   DESIRED STATE  CURRENT STATE ERROR PORTS
l63lya04w606 service1.1  alpine:latest  node1  Running        Running 8 minutes ago

4nf2uaixpqia service1.2  alpine:latest  slave  Running        Running 39 seconds ago

jlj15j6gmguy service1.3  alpine:latest  node2  Running        Running 39 seconds ago

moa4plqe2qtu service1.4  alpine:latest  node2  Running        Running 39 seconds ago

dbzmrysi4eb1 service1.5  alpine:latest  node1  Running        Running 41 seconds ago

okqh401dlyqq service1.6  alpine:latest  master Running        Running 38 seconds ago
```

Scalabilité et load balancing

Mise à l'échelle (Scaling): pour chaque service, vous pouvez déclarer le nombre de tâches que vous souhaitez exécuter. Lorsque vous effectuez une augmentation ou une réduction, le manager Swarm s'adapte automatiquement en ajoutant ou en supprimant des tâches pour conserver l'état souhaité.

Rapprochement de l'état souhaité (réconciliation): le nœud Swarm Manager surveille en permanence l'état du cluster et réconcilie toutes différences entre l'état réel et l'état souhaité. Par exemple, si vous configurez un service pour exécuter 10 réplicas d'un conteneur et qu'une machine de travail hébergeant deux de ces répliques tombe en panne, le manager crée deux nouveaux réplicas pour remplacer ceux qui sont tombés en panne.

```
docker@master:~$ docker  service  scale  service1=6
service1 scaled to 6
overall progress: 6 out of 6 tasks
1/6: running   [==================================================>]
2/6: running   [==================================================>]
3/6: running   [==================================================>]
4/6: running   [==================================================>]
5/6: running   [==================================================>]
6/6: running   [==================================================>]
verify: Service converged
docker@master:~$
```

```
docker@master:~$ docker  service  ls
ID                  NAME                MODE                REPLICAS            IMAGE
PORTS
ponbamjrlhdx        service1            replicated          6/6
alpine:latest
docker@master:~$
```

```
docker@master:~$ docker  service  ps  service1
ID            NAME        IMAGE          NODE   DESIRED STATE  CURRENT STATE ERROR  PORTS
l63lya04w606  service1.1  alpine:latest  node1  Running        Running 8 minutes ago

4nf2uaixpqia  service1.2  alpine:latest  slave  Running        Running 39 seconds ago

jlj15j6gmguy  service1.3  alpine:latest  node2  Running        Running 39 seconds ago

moa4plqe2qtu  service1.4  alpine:latest  node2  Running        Running 39 seconds ago

dbzmrysi4eb1  service1.5  alpine:latest  node1  Running        Running 41 seconds ago

okqh401dlyqq  service1.6  alpine:latest  master Running        Running 38 seconds ago

docker@master:~$
```

```
docker@master:~$ docker  service  logs  service1
service1.6.okqh401dlyqq@master    | PING docker.com (34.232.188.57): 56 data bytes
service1.2.4nf2uaixpqia@slave     | PING docker.com (34.232.188.57): 56 data bytes
service1.5.dbzmrysi4eb1@node1     | PING docker.com (52.3.45.201): 56 data bytes
service1.1.l63lya04w606@node1     | PING docker.com (34.232.188.57): 56 data bytes
service1.4.moa4plqe2qtu@node2     | PING docker.com (54.209.102.157): 56 data bytes
service1.3.jlj15j6gmguy@node2     | PING docker.com (52.3.45.201): 56 data bytes
docker@master:~$
```

```
docker@master:~$ docker  service  inspect  --pretty service1

ID:             ponbamjrlhdx2n6paeoy22pxx
Name:           service1
Service Mode:   Replicated
 Replicas:      6
Placement:
UpdateConfig:
 Parallelism:   1
 On failure:    pause
 Monitoring Period: 5s
 Max failure ratio: 0
 Update order:      stop-first
RollbackConfig:
 Parallelism:   1
 On failure:    pause
 Monitoring Period: 5s
 Max failure ratio: 0
 Rollback order:    stop-first
ContainerSpec:
 Image:
alpine:latest@sha256:e1871801d30885a610511c867de0d6baca7ed4e6a2573d506bbec7fd3b03873f
 Args:          ping docker.com
Resources:
Endpoint Mode:  vip
docker@master:~$
```

Remarques :

Pour modifier le nombre d'instance d'un service en cours de fonctionnement, on peut également utiliser la syntaxe suivante :

```
docker@master:~$ docker  service  update  --replicas 10 service1
```

Suite à cette commande, il y aura 10 instances de service1 qui s'exécuteront.

Docker Swarm

Réseaux et volumes

RESEAUX overlay

```
# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
. . .
mlfha62z3qn0        ingress             overlay             swarm
. . .
```

```
docker@master:~$ docker  network  create  \
                         --attachable  -d overlay  cluster_network
```

VOLUMES Partageable NFS, ...

Réseaux et volumes

Les réseaux :

Suite à la création d'un cluster swarm, il y a un réseau par défaut de type overlay :

```
# docker  network  ls
NETWORK ID          NAME                DRIVER              SCOPE
. . .
mlfha62z3qn0        ingress             overlay             swarm
. . .
```

```
# docker  network  inspect  ingress
[
    {
        "Name": "ingress",
        "Id": "mlfha62z3qn0y1e3a28pwkotb",
        "Created": "2018-10-03T14:57:03.045304102+02:00",
        "Scope": "swarm",
        "Driver": "overlay",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "10.255.0.0/16",
                    "Gateway": "10.255.0.1"
                }
            ]
```

```
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": true,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "ingress-sbox": {
                "Name": "ingress-endpoint",
                "EndpointID":
"b172c8039ae76c59c2ba5490a0fe4cb50b27285509ee3793d01c9f141ffc7b51",
                "MacAddress": "02:42:0a:ff:00:02",
                "IPv4Address": "10.255.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {
            "com.docker.network.driver.overlay.vxlanid_list": "4096"
        },
        "Labels": {},
        "Peers": [
            {
                "Name": "cc3fc17e3ad7",
                "IP": "192.168.1.14"
            }
        ]
    }
]
```

Création d'un réseau dédié au cluster Swarm :

```
$ docker-machine ssh master

docker@master:~$ docker  network  create  \
                         --attachable  -d overlay  cluster_network
```

Pour créer un réseau overlay pouvant être utilisé par des services Swarm ou des conteneurs autonomes pour communiquer avec d'autres conteneurs autonomes s'exécutant sur d'autres démons Docker, ajouter l'option --attachable.

Les volumes :

Lorsque vous démarrez un service et définissez un volume, chaque conteneur de service utilise son propre volume local. Aucun des conteneurs ne peut partager ces données si vous utilisez le pilote de volume local, mais certains pilotes de volume prennent en charge le stockage partagé. Docker pour AWS et Docker pour Azure prennent en charge le stockage persistant à l'aide du plug-in Cloudstor.

Exemple avec NFS :

```
# Création d'un volume réutilisable
  $ docker volume create --driver local \
      --opt type=nfs \
      --opt o=nfsvers=4,addr=192.168.1.1,rw \
      --opt device=:/path/to/dir \
      foo

# A partir d'une commande docker
  $ docker run -it --rm \
    --mount type=volume,dst=/container/path,\
      volume-driver=local,volume-opt=type=nfs,\
      \"volume-opt=o=nfsvers=4,addr=192.168.1.1\",\
      volume-opt=device=:/host/path \
    foo

# A la création d'un service
  $ docker service create \
    --mount type=volume,dst=/container/path,\
      volume-driver=local,volume-opt=type=nfs,\
      \"volume-opt=o=nfsvers=4,addr=192.168.1.1\",\
      volume-opt=device=:/host/path \
    foo

# Au sein d'un fichier docker-compose
  ...
  volumes:
    nfs-data:
      driver: local
      driver_opts:
        type: nfs
        o: nfsvers=4,addr=192.168.1.1,rw
        device: ":/path/to/dir"
```

Docker Swarm

Un fichier docker-compose

```
services:
  web:
    image: utilisateur/depot:tag
    deploy:
      replicas: 5
      restart_policy:
        condition: on-failure
      resources:
        limits:
          cpus: "0.1"
          memory: 50M
  visualizer:
    deploy:
      placement:
        constraints: [node.role == manager]
```

```
$ docker-machine  scp  docker-compose.yml  master:~

$ docker-machine  ssh  master  \
          "docker  stack  deploy  -c docker-compose.yml  monservice"
```

Un fichier docker-compose

Voici un exemple de fichier docker-compose.yml :

```
version: "3"
services:
  web:
    image: utilisateur/depot:tag
    deploy:
      replicas: 5
      restart_policy:
        condition: on-failure
      resources:
        limits:
          cpus: "0.1"
          memory: 50M
    ports:
      - "80:80"
    networks:
      - webnet
  visualizer:
    image: dockersamples/visualizer:stable
    ports:
      - "8080:8080"
    deploy:
      placement:
        constraints: [node.role == manager]
    networks:
      - webnet
networks:
  webnet:
```

Exécution d'un fichier docker-compose.yml :

```
$ docker-machine  scp  docker-compose.yml  master:~

$ docker-machine  ssh  master  \
          "docker  stack  deploy  -c docker-compose.yml  monservice"
```

Notes

Annexes

Dockerfile - les bonnes pratiques

Limiter les processus root

Étape 0 : de base, sans USER user1

Étape 1 : avec USER user1

Étape 2 : sans USER1, et avec dans le script su user1 -c 'commandes'

Étape 3 : sans USER1, et utilisation de su-exec

Étape finale : sans USER user1, avec exec et su-exec

Dockerfile – limiter les processus root

Cette section est en complément des bonnes pratiques pour le fichier Dockerfile.
Le but est de limiter le nombre de processus et de supprimer les processus root.

Résultats d'exécution :

```
[phase2]# docker  build  -t jmb/bonjour2  .
[phase2]# docker  run  -it  --name test2  jmb/bonjour2
[user1@9134d6b5314d /]$ bonjour
Bonjour a toi,
Tu habites a Paris en France
Tu travailles a Spherius
[user1@9134d6b5314d /]$ exit
exit
# docker run -it --name test3 -e ville=Aix -e societe=Docker jmb/bonjour2
[user1@b969d45802f0 /]$ bonjour
Bonjour a toi,
Tu habites a Aix en France
Tu travailles a Docker
```

Avec 1 seul processus non root :

```
[phase3]# cat monbonjour.bash
#!/bin/bash
exec su-exec user1 monscript.bash

[phase3]# cat monscript.bash
#!/bin/bash
hello -g "Bonjour a toi,"
hello -g "Tu habites a ${ville} en ${pays}"
hello -g "Tu travailles a ${societe}"
read

[phase3]# cat Dockerfile
FROM centos:latest
LABEL description="Test de creation d image" \
      maintainer="Baranger Jean-Marc" \
      version="1.0"
ENV ville=Paris \
    pays=France \
    societe=Spherius

COPY monbonjour.bash /usr/bin/bonjour
COPY monscript.bash /usr/bin/monscript.bash

RUN     useradd user1 \
    &&  chmod a+x /usr/bin/bonjour \
    &&  chmod a+x /usr/bin/monscript.bash \
    &&  yum install -y wget git gcc make \
    &&  wget https://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz \
        &&  tar zxvf hello-2.10.tar.gz \
        &&  cd hello-2.10 && ./configure && make && make install \
    &&  git clone https://github.com/ncopa/su-exec.git \
        &&  cd su-exec && make \
        &&  cp su-exec /usr/bin \
    &&  yum remove -y make gcc wget git \
    &&  cd / && rm -rf su-exec hello-2.10.tar.gz hello-2.10
ENTRYPOINT ["bonjour"]
```

```
# docker build -t jmb/bonjour3 --no-cache .
# docker run -it --name test3 jmb/bonjour3
```

```
#  docker top test3
UID                 PID                 PPID                C                   STIME
TTY                 TIME                CMD
user1               22578               22562               1                   20:28
pts/0               00:00:00            /bin/bash /usr/bin/monscript.bash
```

Détail des possibilités :

Étape 0 : de base, sans USER user1

On constate que le processus appartient à root.
Pour certains conteneurs, cela pourrait être gênant.

```
[etape0]# docker  top  test3
UID                 PID                 PPID                C                   STIME
TTY                 TIME                CMD
root                7342                7327                1                   18:06
pts/0               00:00:00            /bin/bash /usr/bin/bonjour


[etape0]# cat   monbonjour.bash
#!/bin/bash
hello -g "Bonjour a toi,"
hello -g "Tu habites a ${ville} en ${pays}"
hello -g "Tu travailles a ${societe}"
read
```

```
[etape0]# cat  Dockerfile
FROM centos:latest
LABEL description="Test de creation d image" \
      maintainer="Baranger Jean-Marc" \
      version="1.0"
ENV ville=Paris \
    pays=France \
    societe=Spherius
COPY monbonjour.bash /usr/bin/bonjour
RUN chmod a+x /usr/bin/bonjour \
    &&  yum  install -y  wget  gcc  make \
    &&  wget  https://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz \
    &&  tar  zxvf  hello-2.10.tar.gz \
    &&  cd  hello-2.10 && ./configure && make && make  install \
    &&  yum remove -y  make gcc wget \
    &&  cd / && rm -rf hello-2.10.tar.gz hello-2.10
ENTRYPOINT ["bonjour"]
```

Étape 1 : avec USER user1

Il est ajouté avant ENTRYPOINT :
USER user1

Il est également ajouté au sein du Dockerfile la création du compte utilisateur user1.

On constate que le processus appartient à user1.
L'ensemble s'exécute avec les privilèges de user1.
Pour certains conteneurs, cela pourrait être gênant.

```
[etape1]# docker  top  test3
UID                 PID                 PPID                C                   STIME
TTY                 TIME                CMD
user1               7580                7565                1                   18:10
pts/0               00:00:00            /bin/bash /usr/bin/bonjour
```

Le script monbonjour.bash est inchangé par rapport à l'étape 0.

```
[etape1]# cat  Dockerfile
FROM centos:latest
LABEL description="Test de creation d image" \
      maintainer="Baranger Jean-Marc" \
      version="1.0"
ENV ville=Paris \
    pays=France \
    societe=Spherius
COPY monbonjour.bash /usr/bin/bonjour
RUN     useradd user1 \
    &&  chmod a+x /usr/bin/bonjour \
    &&  yum  install -y  wget  gcc  make \
    &&  wget  https://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz \
    &&  tar  zxvf  hello-2.10.tar.gz \
    &&  cd  hello-2.10 && ./configure && make && make  install \
    &&  yum remove -y  make gcc wget \
    &&  cd / && rm -rf hello-2.10.tar.gz hello-2.10
USER  user1
ENTRYPOINT ["bonjour"]
```

Étape 2 : sans USER1, et avec dans le script su user1 -c 'commandes'

La ligne avec USER est supprimée du Dockerfile (retour à l'étape 0).
Des modifications sont faites au script pour exécuter que les commandes qui nous intéressent avec les privilèges de user1 (via la commande su).
Pour notre exemple, cela concerne l'ensemble du script.

On constate plusieurs processus appartenant à root et à user1.

```
# docker  top  test3
UID                 PID                 PPID                C                   STIME
TTY                 TIME                CMD
root                18980               18964               0                   18:37
pts/0               00:00:00            /bin/bash /usr/bin/bonjour
root                19003               18980               0                   18:37
pts/0               00:00:00            su user1 -c hello -g "Bonjour a toi," ; \ hello
-g "Tu habites a ${ville} en ${pays}" ; \ hello -g "Tu travailles a ${societe}" ; \ read
user1               19008               19003               0                   18:37
?                   00:00:00            bash -c hello -g "Bonjour a toi," ; \ hello -g
"Tu habites a ${ville} en ${pays}" ; \ hello -g "Tu travailles a ${societe}" ; \ read
```

```
[etape2]# cat  monbonjour.bash
#!/bin/bash
su user1 -c 'hello -g "Bonjour a toi," ; \
             hello -g "Tu habites a ${ville} en ${pays}" ; \
             hello -g "Tu travailles a ${societe}" ; \
             read'
```

Étape 3 : sans USER1, et utilisation de su-exec

On exploite 'su-exec'.
Cette commande su-exec revient à exécuter un su dans le shell courant.

On obtient plus qu'un processus .. mais root !

```
[etape3]# docker  top  test3
UID                 PID                 PPID                C                   STIME
TTY                 TIME                CMD
root                4088                4072                1                   19:36
pts/0               00:00:00            /bin/bash /usr/bin/bonjour
```

```
[etape3]# cat  monbonjour.bash
#!/bin/bash
su-exec user1 hello -g "Bonjour a toi,"  \
              && hello -g "Tu habites a ${ville} en ${pays}"  \
              && hello -g "Tu travailles a ${societe}"  \
              && read
```

Le Dockerfile suivant est un exemple utilisant git :

```
[etape3]# cat  Dockerfile
FROM centos:latest
LABEL description="Test de creation d image" \
      maintainer="Baranger Jean-Marc" \
      version="1.0"
ENV ville=Paris \
    pays=France \
    societe=Spherius
COPY monbonjour.bash /usr/bin/bonjour

RUN     useradd  user1 \
    &&  chmod  a+x  /usr/bin/bonjour \
    &&  yum  install  -y  wget  git  gcc  make \
    &&  wget  https://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz \
        &&  tar  zxvf  hello-2.10.tar.gz \
        &&  cd  hello-2.10  &&  ./configure  &&  make  &&  make  install \
    &&  git  clone  https://github.com/ncopa/su-exec.git \
        &&  cd  su-exec  &&  make \
        &&  cp  su-exec  /usr/bin \
    &&  yum  remove  -y  make  gcc  wget  git \
    &&  cd  /  &&  rm  -rf  su-exec  hello-2.10.tar.gz  hello-2.10

ENTRYPOINT ["bonjour"]
```

Étape finale : sans USER user1, avec exec et su-exec

L'instruction exec permet d'exécuter une commande dans le shell courant.

```
# docker  top  test3
UID                 PID                 PPID                C                   STIME
TTY                 TIME                CMD
user1               22578               22562               1                   20:28
pts/0               00:00:00            /bin/bash /usr/bin/monscript.bash
```

```
# cat  monbonjour.bash
#!/bin/bash
exec su-exec user1 monscript.bash
```

```
# cat  monscript.bash
#!/bin/bash
hello -g "Bonjour a toi,"
hello -g "Tu habites a ${ville} en ${pays}"
hello -g "Tu travailles a ${societe}"
read
```

```
# cat  Dockerfile
FROM centos:latest
LABEL description="Test de creation d image" \
      maintainer="Baranger Jean-Marc" \
      version="1.0"
ENV ville=Paris \
    pays=France \
    societe=Spherius
COPY monbonjour.bash /usr/bin/bonjour
COPY monscript.bash /usr/bin/monscript.bash

RUN     useradd  user1 \
    &&  chmod  a+x  /usr/bin/bonjour \
    &&  chmod  a+x  /usr/bin/monscript.bash \
    &&  yum  install  -y  wget  git  gcc  make \
    &&  wget  https://ftp.gnu.org/gnu/hello/hello-2.10.tar.gz \
        &&  tar  zxvf  hello-2.10.tar.gz \
        &&  cd  hello-2.10  &&  ./configure  &&  make  &&  make  install \
    &&  git  clone  https://github.com/ncopa/su-exec.git \
        &&  cd  su-exec  &&  make \
        &&  cp  su-exec  /usr/bin \
    &&  yum  remove  -y  make  gcc  wget  git \
    &&  cd  /  &&  rm  -rf  su-exec  hello-2.10.tar.gz  hello-2.10

ENTRYPOINT ["bonjour"]
```

Fin de session de Formation

Je vous recommande de relire ce support de cours d'ici les deux semaines à venir, et de refaire des exercices.

Il ne vous reste plus qu'à mettre en œuvre ces nouvelles connaissances au sein de votre entreprise.

Merci, et à bientôt.

Jean-Marc Baranger

Theo Schomaker