

Ce document a ete cree a partir d echanges slackware qui avait la *pretention* de transmettre mon apprentissage de docker
Les Labs ont ete largement inspires de formations docker officielles et qui avaient l avantage d etre ancienne afin de decouvrir les nouveautes au fur et a mesure*

Dockez bien ...

[INFO]

[NOTE]

Vous souhaitez aller au plus vite et faire le projet docker de 42:

Vous trouverez ci-dessous la procedure d'install pour 42, chercher dans cette page "42 INSTALL"

J ai identifie plusieurs methodes d'installation de Docker :

- Docker For Mac (la plus recente Juin 2016)
- Docker Toolbox (qui demande un hyperviseur tel que VisualBox et une machine linux virtuelle (VM). _Je pense que c est la methode demandee pour les exos_
- Docker avec HyperViseur VisualBox et VM boot2docker (C est la plus ancienne et celle qui est utilise dans la video Docker_OLD). Cette Methode a ete remplacee par Docker ToolboxDocker °
- Docker methode 42: cf 42 INSTALL

Je vais essayer d explorer avec vous ces 3 methodes.

Une ptite note: je ne maitrise absolument pas Slack et les messages vont etre haches (Desole)

Une plus grande note: je ne maitrise pas non plus Docker (et donc ce qui est ecrit est simplement ma comprehension)

[QUESTIONS]

- est ce que je peux installer ces 3 'types' de Docker ?

==> Oui, il faut toutefois avoir un environnement (des variables d environnement Shell) different (modifié)

[Docker For Mac]

Methode la plus recente (date de premiere sortie ?Juin 2016?)

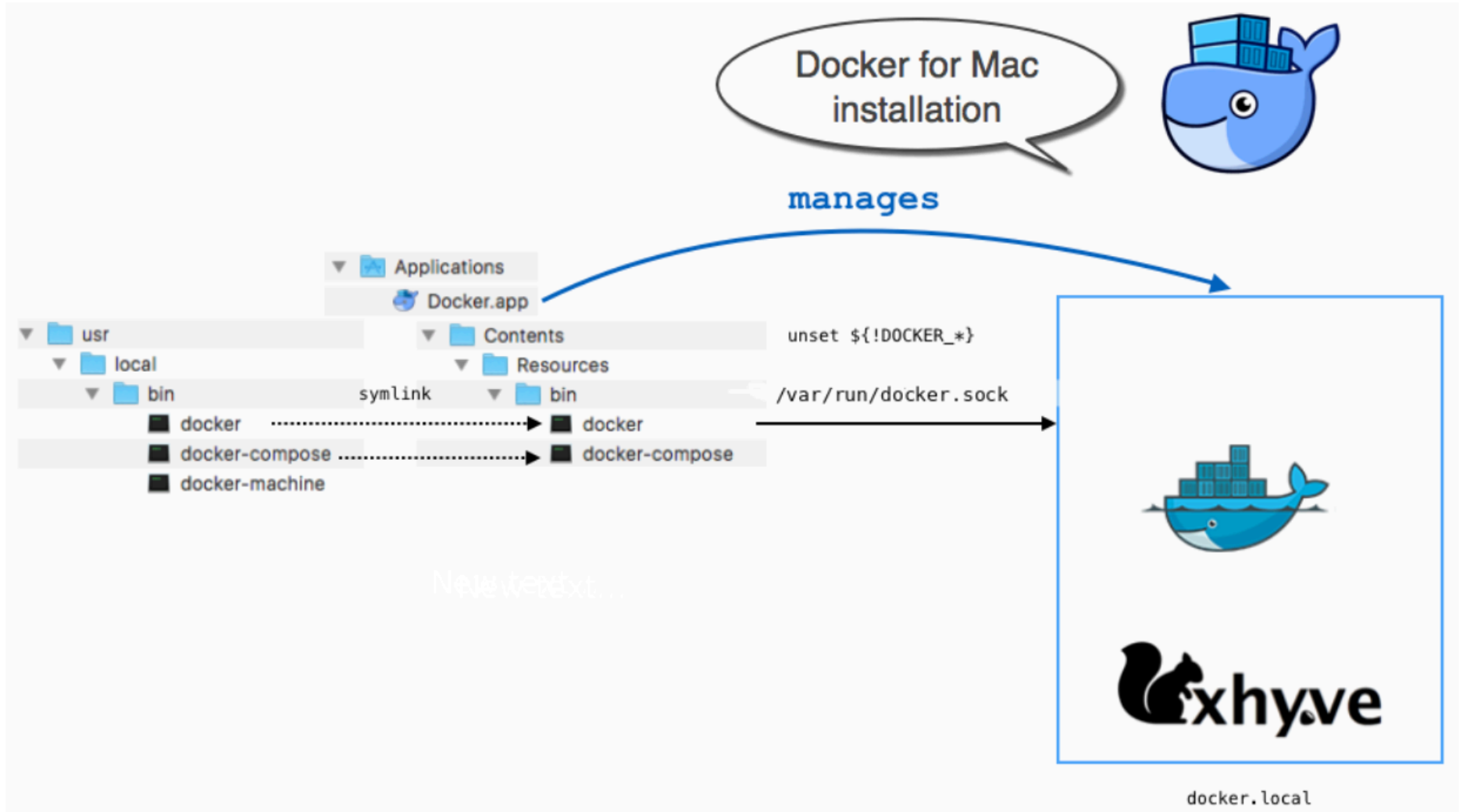
Mac a besoin d un noyau linux:

- * On utilise un hyperviseur HyperKit
- * Et on installe une VM Linux (moby) dedans

Note:

On va dire que HyperKit est comme VirtualBox ou VMware ou bien *_Hyper Kit_* (modifié)

This setup is shown in the following diagram.



[Docker For Mac]

Note:

HyperKit *_is a toolkit for embedding hypervisor capabilities in your application. It includes a complete hypervisor, based on_* ***xhyve/bhyve***

[ressources]

<https://docs.docker.com/docker-for-mac/docker-toolbox/#the-docker-for-mac-environment>

[INFO] Docker For Mac

Une fois Docker For Mac installé, examinons les liens décrits dans l'image ci-dessus

```
e1rlp12% ls -l /usr/local/bin/docker; ls -ls /usr/local/bin/docker-compose
lrwxr-xr-x  1 27072  4249   66 Jul 23 11:28 /usr/local/bin/docker -> /Users/aaiche/Library/Group
Containers/group.com.docker/bin/docker
8 lrwxr-xr-x  1 27072  4249   74 Jul 23 11:28 /usr/local/bin/docker-compose -> /Users/aaiche/Library/Group
Containers/group.com.docker/bin/docker-compose
e1rlp12%
```

[Docker For Mac]

[Question]

***Comment accéder à cet VM installé dans HyperKit?**

```
e1rlp10% screen ~/Library/Containers/com.docker.docker/Data/com.docker.driver.amd64-linux/tty
```

Puis,

Touche Clavier : return

Ainsi on :

```
/ # ls
Database    containers  init        port        run         tmp
Users       dev         lib         private     sbin        usr
Volumes     etc         media       proc        srv         var
bin         home        mnt         root        sys
```

[Question]

***Quel est la version du kernel de cette VM Linux et la version de Docker?**

```
/ # uname -a
Linux moby 4.9.49-moby #1 SMP Wed Sep 27 23:17:17 UTC 2017 x86_64 Linux
/ #
/ # docker --version
Docker version 17.09.0-ce, build afdb6d4
/ #
/ # docker version
Client:
 Version:      17.09.0-ce
 API version:  1.32
 Go version:   go1.8.3
 Git commit:   afdb6d4
 Built:        Tue Sep 26 22:39:28 2017
 OS/Arch:      linux/amd64

Server:
 Version:      17.09.0-ce
 API version:  1.32 (minimum version 1.12)
 Go version:   go1.8.3
 Git commit:   afdb6d4
 Built:        Tue Sep 26 22:45:38 2017
 OS/Arch:      linux/amd64
 Experimental: true
/ #
```

[Note Importante]

Notez bien la version du kernel, tous les containers que vous lancerez utiliseront ce kernel* (modifié)*Notez bien la version du kernel, tous les containers que vous lancerez utiliseront ce kernel

[Docker For Mac]

[Note]

Dans cette machine virtuelle, on peut entrer un login,

A quoi cela sert ce login ?

==> Je ne sais pas

```
/ # login
moby login: root
Welcome to Moby
moby:~# uname -a
Linux moby 4.9.49-moby #1 SMP Wed Sep 27 23:17:17 UTC 2017 x86_64 Linux
moby:~# ls
moby:~#
```

Enfin, pour quitter le 'screen sur la machine virtuelle'

```
Ctrl a et Ctrl \
```

et

```
y
```

[Docker Toolbox]

Plus ancienne Methode que Docker For Mac. Et n est plus recommande, sauf si on peut installe Docker For Mac ou bien si on veut jouer le kernel linux

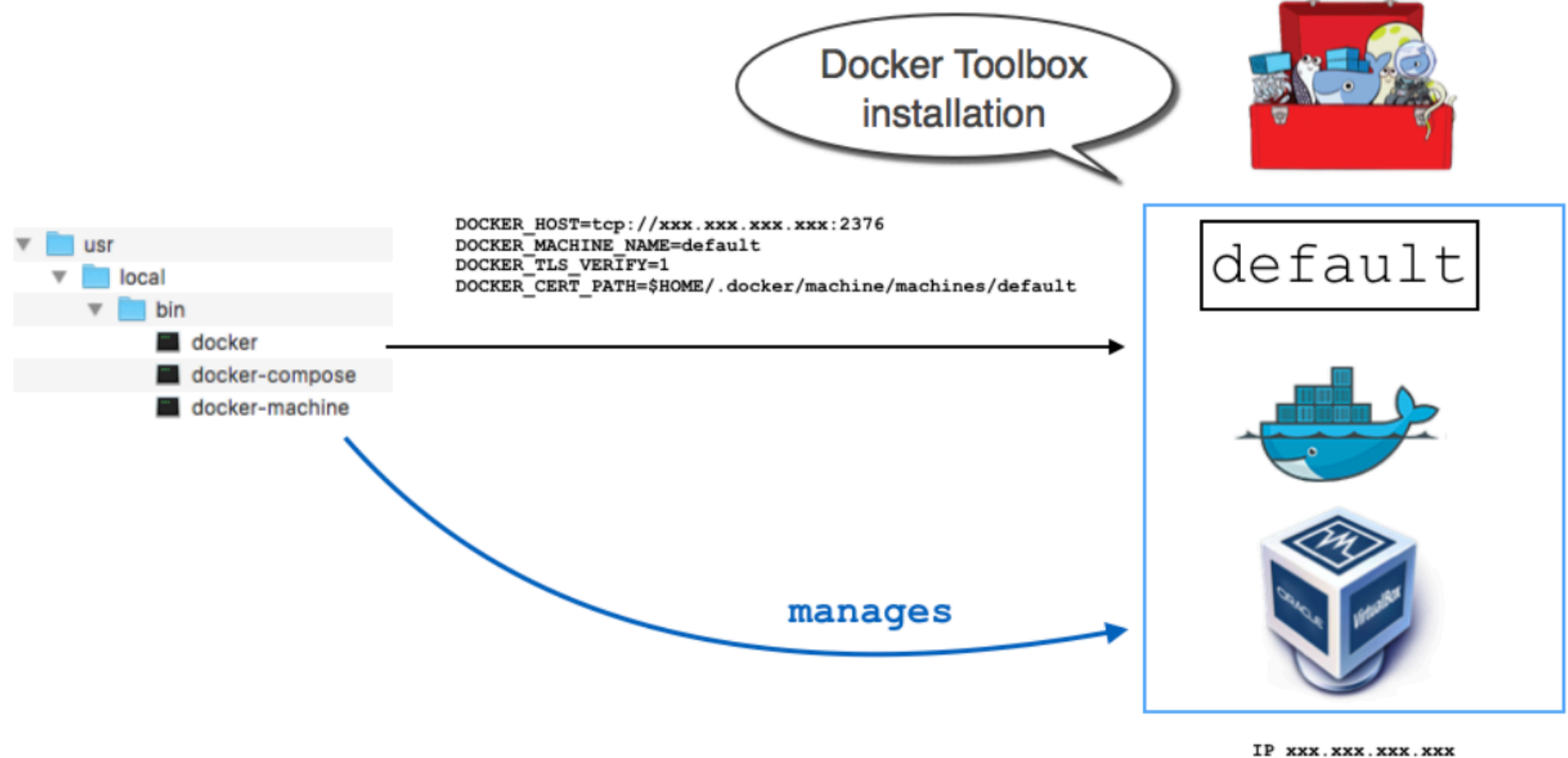
Mac a besoin d un noyau linux:

- On utilise VirtualBox (pour y installer une VM linux)
- Et on installe une VM Linux utilisant la distribution linux boot2docker

[ressources]

<https://docs.docker.com/docker-for-mac/docker-toolbox/#the-docker-toolbox-environment>

This setup is shown in the following diagram.



[Docker Toolbox]

Une fois installé via *MSC*, examinons les variables d'environnement:

```
e1rlp10% set | egrep --binary-files=text DOCKER
DOCKER_CERT_PATH=/Users/aaiche/.docker/machine/machines/default
DOCKER_HOST=tcp://192.168.99.100:2376
DOCKER_MACHINE_NAME=default
DOCKER_TLS_VERIFY=1
```

e1r1p10%

[Docker Toolbox]

[Questions]

* Quelle la version du kernel de la VM?

[illegible]

[Note Importante]

Notez bien la version du kernel, tous les containers que vous lancerez utiliseront ce kernel

[Question]

* Quelle est la version de docker dans cette VM?

```
docker@default:~$ docker --version
Docker version 18.06.0-ce, build 0ffa825
```

```
docker@default:~$ docker version
Client:
 Version:      18.06.0-ce
 API version:  1.38
 Go version:   go1.10.3
 Git commit:   0ffa825
 Built:        Wed Jul 18 19:04:39 2018
 OS/Arch:      linux/amd64
 Experimental: false

Server:
 Engine:
  Version:      18.06.0-ce
  API version:  1.38 (minimum version 1.12)
  Go version:   go1.10.3
  Git commit:   0ffa825
  Built:        Wed Jul 18 19:13:39 2018
  OS/Arch:      linux/amd64
  Experimental: false
docker@default:~$
```

[Docker avec boot2docker]

Une fois installé on peut faire quelques vérifications:

```
elr1p12% set | egrep --binary-files=text DOCKER
DOCKER_CERT_PATH=/Users/aaiche/.boot2docker/certs/boot2docker-vm
DOCKER_HOST=tcp://192.168.59.104:2376
DOCKER_TLS_VERIFY=1
command='export DOCKER_TLS_VERIFY=1'
elr1p12%
```

```
elr1p12% which boot2docker
/Users/aaiche/.brew/bin/boot2docker
elr1p12%
```

```
elr1p12% boot2docker ip
192.168.59.104
elr1p12%``
```


[Note importante]

la video DOCKER_OLD ou l auteur code avec son c.. utilise cette methode

Enfin jettons un coup d oeil sur le site pour nous convaincre que *_docker_machine_* a remplace *_boot2docker_*
[references]

<https://docs.docker.com/machine/migrate-to-machine/#subcommand-comparison>

[INFO]

```
e1rlp12% which dockutil
/usr/bin/dockutil
e1rlp12% dockutil -h
usage:      dockutil -h
usage:      dockutil --add <path to item> | <url> [--label <label>] [ folder_options ] [ position_options ] ....
usage:      dockutil --remove <dock item label>
....
```

dockutil n a rien a avoir Docker, il s agit d un utilitaire pour le Dock de Mac

[TUTORIELS]

[question]

* Est ce que j ai besoin d installer Docker pour l utiliser?

==> Non, regardez le lien ci-dessous (il faut qd meme un browser, internet, et un ordi, ... :=)]

<https://labs.play-with-docker.com/>

Besoin de *_s amuser_* avec Docker, voici un tutoriel (la aussi pas besoin d installer Docker)

<https://training.play-with-docker.com/>

Encore une tonne d infos, videos, slides,

<http://container.training/>

[INSTALL - Docker For Mac]

a) Avant, on verifie

```
elrlp12% which docker
docker not found
elrlp12%
elrlp12% set | egrep --binary-files=text DOCKER
elrlp12%
```

b) Utiliser l'utilitaire "Managed Software Center", puis installer Docker [uniquement l'icone ou il y la petite baleine]

c) puis, verifier

```
elrlp12% which docker
/usr/local/bin/docker
elrlp12%
```

d) Verifier l'etat du service docker

Le message ci-dessous veut dire que le service *n est pas demarre*

```
```elrlp12% docker info
Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?
elrlp12%```
```

ou

```
elrlp12% docker info
Error response from daemon: Bad response from Docker engine
elrlp12%
```

### **\*e) Demarrer le service docker\***

- Utiliser l'utilitaire Launchpad (i.e. la petite fusée sur le Dock)
- ou
- Utiliser le clavier "Pomme" + "espace" + et puis taper "docker"

Une petite baleine apparaît en haut à droite, près de l'heure

### **\*f) Verifier\***

```
e1rlp12% docker info
Containers: 0
 Running: 0
 Paused: 0
 Stopped: 0
Images: 0
Server Version: 17.09.0-ce
....
```

Le service est démarré

### **\*g) Verifier votre espace de stockage personnel\***

Voici un fichier (disk de la machine virtuelle) qui prend plus de 1 GB de votre espace personnel

```
e1rlp12% ls -l -h /Users/$USERNAME/Library/Containers/com.docker.docker/Data/com.docker.driver.amd64-linux/Docker.qcow2
-rw-r--r--@ 1 aaiche 2017_paris 1.3G Jul 23 17:20 /Users/aaiche/Library/Containers/com.docker.docker/Data/com.docker.driver.amd64-
linux/Docker.qcow2
e1rlp12%
```

### **\*h) Déplacer le fichier ci-dessus sur un autre point de montage (e.g. sgoinfre)\***

- Arrêter le service docker
- Déplacer le fichier et créer un lien symbolique

```
mv /Users/<username>/Library/Containers/com.docker.docker/Data/com.docker.driver.amd64-linux/Docker.qcow2 /<path-of-your-choice>/
```

```
ln -s /<path-of-your-choice>/Docker.qcow2 /Users/<username>/Library/Containers/com.docker.docker/Data/com.docker.driver.amd64-
linux/Docker.qcow2
```

Par exemple en utilisant sgoinfre

```
elrlp12% mv /Users/$USERNAME/Library/Containers/com.docker.docker/Data/com.docker.driver.amd64-linux/Docker.qcow2
/sgoinfre/goinfre/Perso/$USERNAME``
``elrlp12% ln -s /sgoinfre/goinfre/Perso/$USERNAME/Docker.qcow2
/Users/$USERNAME/Library/Containers/com.docker.docker/Data/com.docker.driver.amd64-linux/Docker.qcow2
```

```
elrlp12% ls -l /Users/$USERNAME/Library/Containers/com.docker.docker/Data/com.docker.driver.amd64-linux/Docker.qcow2
lrwxr-xr-x 1 aaiche 2017_paris 43 Jul 23 17:37 /Users/aaiche/Library/Containers/com.docker.docker/Data/com.docker.driver.amd64-
linux/Docker.qcow2 -> /sgoinfre/goinfre/Perso/aaiche/Docker.qcow2
elrlp12%
```

**\*i) Demarrer le service Docker\***

**\*j) Verifier votre version\***

```
elrlp12% docker --version
Docker version 17.09.0-ce, build afdb6d4
elrlp12% docker version
Client:
 Version: 17.09.0-ce
 API version: 1.32
 Go version: go1.8.3
 Git commit: afdb6d4
 Built: Tue Sep 26 22:40:09 2017
 OS/Arch: darwin/amd64

Server:
 Version: 17.09.0-ce
 API version: 1.32 (minimum version 1.12)
 Go version: go1.8.3
 Git commit: afdb6d4
 Built: Tue Sep 26 22:45:38 2017
 OS/Arch: linux/amd64
 Experimental: true
elrlp12%
```

**\*k) Dock well ...\***

### **\*[NOTE IMPORTANTE]\***

A partir de la version 1.3 les commandes CLI ont ete re-organisees.

[reference]

<https://blog.docker.com/2017/01/whats-new-in-docker-1-13/>

Ils precedent `_image_` a toutes les commandes relatives aux images ou ben `_container_` celles qui sont relatives aux containers

Par exemple

La nouvelle facon de faire

```
docker container ls
docker image rm
```

L ancienne

```
docker ps
docker rmi
```

### **\*[QUESTION]\***

**\*Comment lire les versions de Docker?\***

=> Je ne sais pas

**\*[Mes Premieres Images & Containers - Docker For Mac]\***

**\*But: Telecharger plusieurs flavors de Linux et voir quel est la version du kernel\***

**\*a) Rappel\***

Docker For Mac utilise une VM Linux dont le noyau est :

```
elrl2p9% screen ~/Library/Containers/com.docker.docker/Data/com.docker.driver.amd64-linux/tty``
``/ # uname -a
Linux moby 4.9.49-moby #1 SMP Wed Sep 27 23:17:17 UTC 2017 x86_64 Linux
/ #
```

### **\*b) PULL (telechargement a partir de Docker Hub de plusieurs versions de Linux)\***

Par exemple, pour telecharger fedora avec le tag 28

```
elrl2p9% docker image pull fedora:28
```

### **\*c) List toutes les images pull'ees\***

```
elrl2p9% docker image ls -a
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu	16.04	e13f3d529b1a	7 days ago	115MB
debian	8.11	79f4bda91989	7 days ago	127MB
alpine	3.6	da579b235e92	2 weeks ago	4.03MB
fedora	28	cc510acfc70	2 months ago	253MB
centos	6.6	3d7ac13b921a	8 months ago	203MB

### **\*d) Examen des noyaux (kernels) de toutes ces images\***

```
elrl2p9% docker run --interactive --tty --rm ubuntu:16.04 uname -a
Linux 3270afb59a03 4.9.49-moby #1 SMP Wed Sep 27 23:17:17 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux
elrl2p9% docker run --interactive --tty --rm debian:8.11 uname -a
Linux e79276a71f4c 4.9.49-moby #1 SMP Wed Sep 27 23:17:17 UTC 2017 x86_64 GNU/Linux
elrl2p9% docker run --interactive --tty --rm alpine:3.6 uname -a
Linux 24da3fa3a691 4.9.49-moby #1 SMP Wed Sep 27 23:17:17 UTC 2017 x86_64 Linux
elrl2p9% docker run --interactive --tty --rm fedora:28 uname -a
Linux 1c667eec043f 4.9.49-moby #1 SMP Wed Sep 27 23:17:17 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux
elrl2p9% docker run --interactive --tty --rm centos:6.6 uname -a
Linux a71cba30e30d 4.9.49-moby #1 SMP Wed Sep 27 23:17:17 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux
```

### **\*e) Conclusion\***

**\*On voit bien que toutes ces versions utilisent le meme kernel que la machine Virtuelle installee par Docker For Mac. En faite il n y a pas de kernel dans les images.\***

## **\*[INSTALL Docker Toolbox]\***

J'ai identifié plusieurs méthodes :

- \*Managed Software Center, puis Docker Toolbox, et enfin Launchpad/Docker Quickstart Terminal
- \*Managed Software Center, puis Docker Toolbox, et enfin Launchpad/Kitematic
- \*Managed Software Center, puis Docker Toolbox, et enfin Commandes CLI

L'utilitaire "Managed Software Center", puis Docker Toolbox (uniquement l'icône avec la boîte à outil rouge):

Cela installe :

- \*Docker Client
- \*Docker Machine
- \*Docker Compose
- \*Docker Quickstart Terminal App (voir le launchpad)
- \*Kitematic (voir le launchpad)
- \*Virtual Box (voir le launchpad)
- \*Utiliser Kitematic Beta: permet d'installer le tout et d'utiliser l'interface graphique afin de chercher des images et les faire démarrer ...

### **\*a) Avant on vérifie:\***

```
elrlp12% which docker; which docker-machine
docker not found
docker-machine not found
elrlp12%
```

### **\*Quelques répertoires...\***

```
elrlp12% ls -l ~/Library/Containers/com.docker.docker
ls: /Users/aaiche/Library/Containers/com.docker.docker: No such file or directory
elrlp12%
elrlp12% ls -l ~/.docker
ls: /Users/aaiche/.docker: No such file or directory
elrlp12%
elrlp12% ls -l ~/VirtualBox\ VMs
elrlp12%
```



**\*b) Utiliser l'utilitaire "Managed Software Center", puis installer Docker Toolbox (uniquement l'icone avec la boite a outil rouge)\***

Cela installe :

\*Docker Client

\*Docker Machine

\*Docker Compose

\*Docker Quickstart Terminal App

\*Kitematic

\*Virtual Box

**\*c) [restart Mac : j ai lu cette etape sur certaines procedures. Rebooter apres avoir installer VirtualBox]\***

**\*d) On verifie\***

\*Docker Client, Docker Machine, Docker Compose

```
e1rlp12% which docker; which docker-machine; which docker-compose
/usr/local/bin/docker
/usr/local/bin/docker-machine
/usr/local/bin/docker-compose
e1rlp12%
e1rlp12% docker-machine ls
NAME ACTIVE DRIVER STATE URL SWARM DOCKER ERRORS
e1rlp12%
e1rlp12% set | egrep --binary-files=text DOCKER
e1rlp12%
e1rlp12% docker info
Cannot connect to the Docker daemon. Is the docker daemon running on this host?
e1rlp12%
```

\*Docker Quickstart Terminal App (voir le launchpad)

\*Kitematic (voir le launchpad)

\*Virtual Box (voir le launchpad)

**\*Quelques repertoires...\***

```
e1rlp12% ls -l ~/Library/Containers/com.docker.docker
ls: /Users/aaiche/Library/Containers/com.docker.docker: No such file or directory
e1rlp12% ls -l ~/.docker
```

```
total 0
drwxr-xr-x 3 aaiche staff 102 Jul 25 10:01 machine/
elrlp12% ls -lh ~/.docker/machine/cache/boot2docker.iso
-rw-r--r-- 1 aaiche staff 35M Jul 25 10:01 /Users/aaiche/.docker/machine/cache/boot2docker.iso
elrlp12% ls -l ~/VirtualBox\ VMs
elrlp12% ls -la ~/VirtualBox\ VMs
total 0
drwx----- 2 aaiche 2017_paris 68 Jul 23 16:28 ./
drwxr-xr-x 58 aaiche 2017_paris 2040 Jul 25 10:07 ../
elrlp12%
```

## \*[INSTALL Docker Toolbox + Docker Quickstart Terminal]\*

### \*a) Pre-requis\*

\*Docker Toolbox est installe

### \*b) Laisse Docker Quickstart Terminal installer la machine virtuelle et finir la configuration\*

Installer la machine virtuelle: "Docker Quickstart Terminal" ==> Apres plusieurs tentatives et reboot + commandes en ligne ...

\*Launchpad, puis

\*Click icone "Docker Quickstart Terminal": j ai choisi "Terminal"

**\*Le script reste bloqué : apres plusieurs tentatives,\***

```
bash --login '/Applications/Docker/Docker Quickstart Terminal.app/Contents/Resources/Scripts/start.sh'
Last login: Tue Jul 24 14:47:11 on ttys006
elrlp12% bash --login '/Applications/Docker/Docker Quickstart Terminal.app/Contents/Resources/Scripts/start.sh'
Running pre-create checks...
(default) Default Boot2Docker ISO is out-of-date, downloading the latest release...
....
(default) Waiting for an IP...
Waiting for machine to be running, this may take a few minutes...
...
Provisioning with boot2docker...
```

### \*c) On verifie\*

```
elrlp12% docker-machine ls
```

NAME	ACTIVE	DRIVER	STATE	URL	SWARM	DOCKER	ERRORS
default	-	virtualbox	Running	tcp://192.168.99.100:2376		v18.06.0-ce	

```
elrlp12%
```

### **\*Quelques repertoires...\***

```
elrlp12% ls -l ~/Library/Containers/com.docker.docker
ls: /Users/aaiche/Library/Containers/com.docker.docker: No such file or directory
elrlp12%
elrlp12% ls -l ~/.docker
total 0
drwxr-xr-x 5 aaiche staff 170 Jul 25 10:28 machine/
elrlp12% ls -la ~/VirtualBox\ VMs
total 0
drwx----- 2 aaiche 2017_paris 68 Jul 23 16:28 ./
drwxr-xr-x 58 aaiche 2017_paris 2040 Jul 25 10:50 ../
elrlp12% ~/.docker/machine/machines/default
zsh: permission denied: /Users/aaiche/.docker/machine/machines/default
elrlp12% ls ~/.docker/machine/machines/default
boot2docker.iso config.json id_rsa server-key.pem
ca.pem default/ id_rsa.pub server.pem
cert.pem disk.vmdk key.pem
elrlp12%
elrlp12% set | egrep --binary-files=text DOCKER
elrlp12%
```

### **\*d) Demarrer le service\***

#### **\*Le service n est pas demarre:\***

```
elrlp12% docker info
Cannot connect to the Docker daemon. Is the docker daemon running on this host?
elrlp12%
```

### **\*Demarrer le service\***

```
elrlp12% docker-machine ls
NAME ACTIVE DRIVER STATE URL SWARM DOCKER ERRORS
default - virtualbox Running tcp://192.168.99.100:2376 v18.06.0-ce
elrlp12% docker-machine env
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.100:2376"
export DOCKER_CERT_PATH="/Users/aaiche/.docker/machine/machines/default"
export DOCKER_MACHINE_NAME="default"
```

```
Run this command to configure your shell:
eval $(docker-machine env)
elrlp12%
elrlp12% eval $(docker-machine env default)
elrlp12%
elrlp12% set | egrep --binary-files=text DOCKER
DOCKER_CERT_PATH=/Users/aaiche/.docker/machine/machines/default
DOCKER_HOST=tcp://192.168.99.100:2376
DOCKER_MACHINE_NAME=default
DOCKER_TLS_VERIFY=1
command='set | egrep --binary-files=text DOCKER'
elrlp12%
elrlp12% docker-machine ls
NAME ACTIVE DRIVER STATE URL SWARM DOCKER ERRORS
default * virtualbox Running tcp://192.168.99.100:2376 v18.06.0-ce
```

### **\*Verifions que le service est demarre:\***

```
elrlp12% docker info
Containers: 0
 Running: 0
 Paused: 0
 Stopped: 0
Images: 0
Server Version: 18.06.0-ce
...
```

### **\*Testons avec hello-world:\***

```
elrlp12% docker version
Client:
 Version: 1.11.2
 API version: 1.23
...
Server:
 Version: 18.06.0-ce
 API version: 1.38
...
elrlp12%
elrlp12% docker container run hello-world
docker: 'container' is not a docker command.
See 'docker --help'.
elrlp12% docker run hello-world
 Hello from Docker!
```

**\*[Question]\***

Savez vous pourquoi `_docker container run hello-world_` ne fonctionne pas !!

Reponse:

Regardez la version de docker: Elle est < 1.3

## \*[Mes Premières Images & Containers - Docker Toolbox]\*

**\*But: Telecharger plusieurs flavors de Linux et voir quel est la version du kernel\***

**\*a) Rappel\***

Docker Toolbox utilise une VM Linux dont le noyau est :

[illegible]

### **\*b) PULL (telechargement a partir de Docker Hub de plusieurs versions de Linux)\***

Par exemple, pour telecharger fedora avec le tag 28

```
elrlp12% docker image pull fedora:28
docker: 'image' is not a docker command.
See 'docker --help'.
elrlp12% docker pull fedora:28
```

### **\*c) List toutes les images pull'ees\***

```
elrlp12% docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu	16.04	e13f3d529b1a	8 days ago	114.5 MB
ubuntu	<none>	e13f3d529b1a	8 days ago	114.5 MB
debian	8.11	79f4bda91989	8 days ago	126.8 MB
debian	<none>	79f4bda91989	8 days ago	126.8 MB
alpine	3.6	da579b235e92	2 weeks ago	4.028 MB
alpine	<none>	da579b235e92	2 weeks ago	4.028 MB
fedora	28	cc510acfc70	11 weeks ago	252.9 MB
fedora	<none>	cc510acfc70	11 weeks ago	252.9 MB
centos	6.6	3d7ac13b921a	8 months ago	202.6 MB
centos	<none>	3d7ac13b921a	8 months ago	202.6 MB

```
elrlp12%
```

### **\*d) Examen des noyaux (kernels) de toutes ces images\***

```
elrlp12% docker run --interactive --tty --rm
elrlp12% docker run --interactive --tty --rm ubuntu:16.04 uname -a
Linux 5214f924c230 4.9.93-boot2docker #1 SMP Thu Jul 19 18:29:50 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux
elrlp12% docker run --interactive --tty --rm debian:8.11 uname -a
Linux 15b93089d89a 4.9.93-boot2docker #1 SMP Thu Jul 19 18:29:50 UTC 2018 x86_64 GNU/Linux
elrlp12% docker run --interactive --tty --rm alpine:3.6 uname -a
Linux b626d0c3bc58 4.9.93-boot2docker #1 SMP Thu Jul 19 18:29:50 UTC 2018 x86_64 Linux
elrlp12% docker run --interactive --tty --rm fedora:28 uname -a
Linux e0d7a3d7a7e9 4.9.93-boot2docker #1 SMP Thu Jul 19 18:29:50 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux
elrlp12% docker run --interactive --tty --rm centos:6.6 uname -a
Linux 8f9c39953b24 4.9.93-boot2docker #1 SMP Thu Jul 19 18:29:50 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux
```

### **\*e) Conclusion\***

**\*On voit bien que toutes ces versions utilisent le meme kernel que la machine Virtuelle (default) installee a partir de boot2docker.iso\***

**\*[Note : INSTALL Docker Toolbox]\***

**\*Contrairement a Docker For Mac, les images sont stockees dans `~/docker/machine/machines_*`**

**\*C est donc le repertoire `~/docker_` qu'il faut déplacer dans `_sgoinfre_` et le linker symboliquement en cas de besoin d espace disque\***

**\*[Docker Toolbox + Kitematic]\***

GUI permettant de faire ce que l on fait en CLI.

Peut etre interessant de jeter un coup d oeil afin de voir de maniere graphique les differentes options.



aaiche ▾

K7

Containers



















[+ NEW](#)

Search for Docker images from Docker Hub

FILTER BY

**All**[Recommended](#)[My Repos](#)[My Images](#)

## Recommended

 <b>kitematic</b> <b>hello-world-nginx</b> A light-weight nginx container that demonstrates the features of Kitematic ♥ 103 ↓ 10M ... <a href="#">CREATE</a>	 <b>official</b> <b>ghost</b> Ghost is a free and open source blogging platform written in JavaScript ♥ 797 ↓ 51M ... <a href="#">CREATE</a>	 <b>official</b> <b>jenkins</b> Official Jenkins Docker image ♥ 3.7K ↓ 64M ... <a href="#">CREATE</a>
 <b>official</b> <b>redis</b> Redis is an open source key-value store that functions as a data structure server. ♥ 5.5K ↓ 873M ... <a href="#">CREATE</a>	 <b>official</b> <b>rethinkdb</b> RethinkDB is an open-source, document database that makes it easy to build and scale realtime... ♥ 469 ↓ 24M ... <a href="#">CREATE</a>	 <b>kitematic</b> <b>minecraft</b> The Minecraft multiplayer server allows two or more players to play Minecraft together ♥ 106 ↓ 137K ... <a href="#">CREATE</a>
 <b>official</b> <b>solr</b> Solr is the popular, blazing-fast, open source enterprise search platform built on Apache... ♥ 555 ↓ 27M ... <a href="#">CREATE</a>	 <b>official</b> <b>elasticsearch</b> Elasticsearch is a powerful open source search and analytics engine that makes data easy to... ♥ 2.9K ↓ 171M ... <a href="#">CREATE</a>	 <b>official</b> <b>postgres</b> The PostgreSQL object-relational database system provides reliability and data integrity. ♥ 5.2K ↓ 413M ... <a href="#">CREATE</a>
 <b>official</b> <b>ubuntu-upstart</b> Upstart is an event-based replacement for the /sbin/init daemon which starts processes... ♥ 87 ↓ 1M ... <a href="#">CREATE</a>	 <b>official</b> <b>memcached</b> Free & open source, high-performance, distributed memory object caching system. ♥ 1.1K ↓ 415M ... <a href="#">CREATE</a>	 <b>official</b> <b>rabbitmq</b> RabbitMQ is an open source multi-protocol messaging broker. ♥ 2.0K ↓ 161M ... <a href="#">CREATE</a>
 <b>official</b> <b>celery</b> Celery is an open source asynchronous task queue/job queue based on distributed... ♥ 217 ↓ 2M ... <a href="#">CREATE</a>	 <b>official</b> <b>mysql</b> MySQL is a widely used, open-source relational database management system (RDBMS). ♥ 6.6K ↓ 412M ... <a href="#">CREATE</a>	 <b>official</b> <b>mongo</b> MongoDB document databases provide high availability and easy scalability. ♥ 4.7K ↓ 624M ... <a href="#">CREATE</a>
 <b>official</b> <b>mariadb</b>	 <b>official</b> <b>percona</b>	 <b>official</b> <b>crate</b>



DOCKER CLI























### **\*[Docker Toolbox + Kitematic]\***

Par exemple voici les images telechargees precedemment



My Images

	local <b>ubuntu</b> No description.	 16.04		CREATE
	local <b>debian</b> No description.	 8.11		CREATE
	local <b>alpine</b> No description.	 3.6		CREATE
	local <b>fedora</b> No description.	 28		CREATE
	local <b>centos</b> No description.	 6.6		CREATE
	kitematic <b>hello-world-nginx</b> No description.	 latest		CREATE

**\*[INSTALL Docker Toolbox + Docker Kitematic Beta]\***

**\*a) Pre-requis\***

\*Docker Toolbox est installe

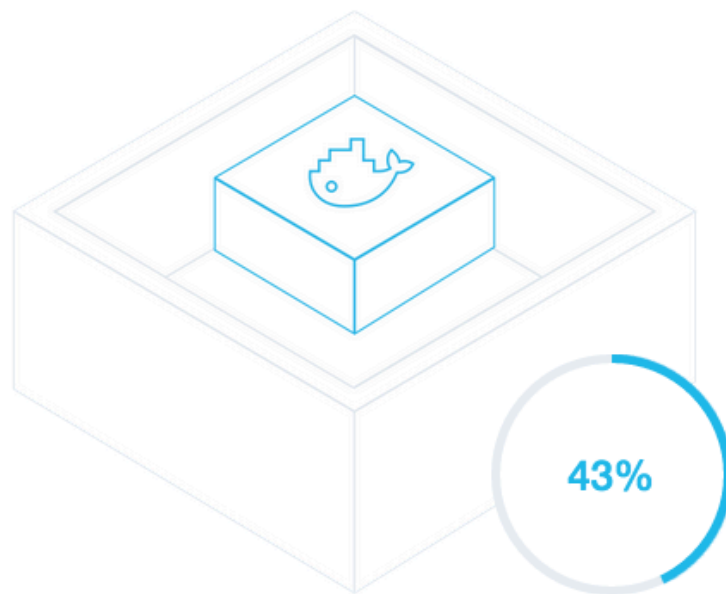
```
e1r1p12% docker-machine ls
NAME ACTIVE DRIVER STATE URL SWARM DOCKER ERRORS
e1r1p12%
```

**\*b) Laisse Docker Kitematic Beta installer la machine virtuelle et finir la configuration\***

\*Launchpad, puis Docker Kitematic

**\*[INSTALL Docker Toolbox + Docker Kitematic Beta]\***

Note : rate !! pour le 42 %



## Starting Docker VM

To run Docker containers on your computer, Kitematic is starting a Linux virtual machine. This may take a minute...

### **\*[INSTALL Docker Toolbox + Docker Kitematic Beta]\***



















Une fois l'installation terminée

#### **\*{Note}\***

Remarquez "Docker CLI" en bas à gauche: permet d'ouvrir un terminal et les variables d'environnement nécessaires



Recommended

 <p>kitematic <b>hello-world-nginx</b> A light-weight nginx container that demonstrates the features of Kitematic</p> <p>♡ 103 ↓ 10M ... <a href="#">CREATE</a></p>	 <p>official <b>ghost</b> Ghost is a free and open source blogging platform written in JavaScript</p> <p>♡ 797 ↓ 51M ... <a href="#">CREATE</a></p>	 <p>official <b>jenkins</b> Official Jenkins Docker image</p> <p>♡ 3.7K ↓ 64M ... <a href="#">CREATE</a></p>
 <p>official <b>redis</b> Redis is an open source key-value store that functions as a data structure server.</p> <p>♡ 5.5K ↓ 873M ... <a href="#">CREATE</a></p>	 <p>official <b>rethinkdb</b> RethinkDB is an open-source, document database that makes it easy to build and scale realtime...</p> <p>♡ 469 ↓ 24M ... <a href="#">CREATE</a></p>	 <p>kitematic <b>minecraft</b> The Minecraft multiplayer server allows two or more players to play Minecraft together</p> <p>♡ 106 ↓ 137K ... <a href="#">CREATE</a></p>
 <p>official <b>solr</b> Solr is the popular, blazing-fast, open source enterprise search platform built on Apache...</p> <p>♡ 555 ↓ 27M ... <a href="#">CREATE</a></p>	 <p>official <b>elasticsearch</b> Elasticsearch is a powerful open source search and analytics engine that makes data easy to...</p> <p>♡ 2.9K ↓ 171M ... <a href="#">CREATE</a></p>	 <p>official <b>postgres</b> The PostgreSQL object-relational database system provides reliability and data integrity.</p> <p>♡ 5.2K ↓ 413M ... <a href="#">CREATE</a></p>
 <p>official <b>ubuntu-upstart</b> Upstart is an event-based replacement for the /sbin/init daemon which starts processes...</p> <p>♡ 87 ↓ 1M ... <a href="#">CREATE</a></p>	 <p>official <b>memcached</b> Free &amp; open source, high-performance, distributed memory object caching system.</p> <p>♡ 1.1K ↓ 415M ... <a href="#">CREATE</a></p>	 <p>official <b>rabbitmq</b> RabbitMQ is an open source multi-protocol messaging broker.</p> <p>♡ 2.0K ↓ 161M ... <a href="#">CREATE</a></p>
 <p>official <b>celery</b> Celery is an open source asynchronous task queue/job queue based on distributed...</p> <p>♡ 217 ↓ 2M ... <a href="#">CREATE</a></p>	 <p>official <b>mysql</b> MySQL is a widely used, open-source relational database management system (RDBMS).</p> <p>♡ 6.6K ↓ 412M ... <a href="#">CREATE</a></p>	 <p>official <b>mongo</b> MongoDB document databases provide high availability and easy scalability.</p> <p>♡ 4.7K ↓ 625M ... <a href="#">CREATE</a></p>
 <p>official <b>mariadb</b></p>	 <p>official <b>percona</b></p>	 <p>official <b>crate</b></p>

## **\*[INSTALL Docker Toolbox + Docker Kitematic Beta]\***

### **\*a) On verifie\***

```
elrlp12% set | egrep --binary-files=text DOCKER
DOCKER_CERT_PATH=/Users/aaiche/.docker/machine/machines/default
DOCKER_HOST=tcp://192.168.99.100:2376
DOCKER_TLS_VERIFY=1
command='set | egrep --binary-files=text DOCKER'
elrlp12% docker-machine ls
NAME ACTIVE DRIVER STATE URL SWARM DOCKER ERRORS
default * virtualbox Running tcp://192.168.99.100:2376 v18.06.0-ce
elrlp12%
elrlp12% docker run hello-world
...
Hello from Docker!
...
elrlp12%
elrlp12% docker --version
Docker version 1.11.2, build b9f10c9
elrlp12% docker version
Client:
 Version: 1.11.2
 API version: 1.23
 Go version: go1.5.4
 Git commit: b9f10c9
 Built: Wed Jun 1 21:20:08 2016
 OS/Arch: darwin/amd64

Server:
 Version: 18.06.0-ce
 API version: 1.38
 Go version: go1.10.3
 Git commit: 0ffa825
 Built: Wed Jul 18 19:13:39 2018
 OS/Arch: linux/amd64
elrlp12%
```

## **\*INSTALL Docker Toolbox + Commandes CLI]\***

### **\*a) Pre-requis\***

\*Docker Toolbox est installe

```
elrlp12% docker-machine ls
```

NAME	ACTIVE	DRIVER	STATE	URL	SWARM	DOCKER	ERRORS
elrlp12%							
elrlp12% set   egrep --binary-files=text						DOCKER	
command='set   egrep --binary-files=text						DOCKER'	
elrlp12%							

## **\*b) Creer la machine virtuelle\***

```
elrlp12% docker-machine ls
NAME ACTIVE DRIVER STATE URL SWARM DOCKER ERRORS
elrlp12% docker-machine create --driver virtualbox default
Running pre-create checks...
Creating machine...
(default) Copying /Users/aaiche/.docker/machine/cache/boot2docker.iso to
/Users/aaiche/.docker/machine/machines/default/boot2docker.iso...
(default) Creating VirtualBox VM...
(default) Creating SSH key...
(default) Starting the VM...
(default) Check network to re-create if needed...
(default) 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: docker-machine env default
elrlp12%
```

## **\*c) Mettre en place l'environnement requis par docker-machine\***

```
elrlp12% docker-machine env default
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.101:2376"
export DOCKER_CERT_PATH="/Users/aaiche/.docker/machine/machines/default"
export DOCKER_MACHINE_NAME="default"
Run this command to configure your shell:
eval $(docker-machine env default)

elrlp12% eval $(docker-machine env default)
elrlp12%
```



### **\*[Note]\***

Si vous ouvrez un autre terminal, ces variables d'environnement ne seront pas mises en place.  
Il faut donc ajouter la commande `shell _eval $(docker-machine env default)_` dans votre profile shell  
E.g.

```
elrlp12% echo $SHELL
/bin/zsh
elrlp12% echo 'eval $(docker-machine env default)' >> ~/.zshrc
```

### **\*d) On verifie\***

```
elrlp12% set | egrep --binary-files=text DOCKER
DOCKER_CERT_PATH=/Users/aaiche/.docker/machine/machines/default
DOCKER_HOST=tcp://192.168.99.101:2376
DOCKER_MACHINE_NAME=default
DOCKER_TLS_VERIFY=1
command='set | egrep --binary-files=text DOCKER'
elrlp12%
elrlp12% docker-machine ls
NAME ACTIVE DRIVER STATE URL SWARM DOCKER ERRORS
default * virtualbox Running tcp://192.168.99.101:2376 v18.06.0-ce
elrlp12%
elrlp12% docker run hello-world
...
Hello from Docker!
...
```

### **\*[42 INSTALL] [INSTALL docker, docker-machine, docker-compose, virtualbox COMME suggere dans le sujet docker]\***

Cette methode a l'avantage d'utiliser les dernieres versions de docker

### **\*a) Install VirtualBox via MSC\***

### **\*b) Ques petites verifications\***

```
elrlp12% which docker docker-machine docker-compose
docker not found
docker-machine not found
docker-compose not found
elrlp12%``
```

```
elrlp12% set | egrep --binary-files=text DOCKER
command='set | egrep --binary-files=text DOCKER'
elrlp12%
```

```
elrlp12% ls ~/.brew/Cellar/docker*
zsh: no matches found: /Users/aaiche/.brew/Cellar/docker*
elrlp12%
```

### **\*c) Insall docker et ...\***

```
elrlp12% brew install docker docker-machine docker-compose
...
```

### **\*d) Qques petites verifications...\***

```
elrlp12% which docker docker-machine docker-compose
/Users/aaiche/.brew/bin/docker
/Users/aaiche/.brew/bin/docker-machine
/Users/aaiche/.brew/bin/docker-compose
elrlp12%
```

```
elrlp12% ls ~/.brew/Cellar/docker*
/Users/aaiche/.brew/Cellar/docker:
18.06.0/

/Users/aaiche/.brew/Cellar/docker-compose:
1.22.0/

/Users/aaiche/.brew/Cellar/docker-machine:
0.15.0/
elrlp12%
```

### \*e) Installer la VM default\*

```
elrlp12% docker-machine ls
NAME ACTIVE DRIVER STATE URL SWARM DOCKER ERRORS
elrlp12%
elrlp12% docker-machine create --driver virtualbox default
Creating CA: /Users/aaiche/.docker/machine/certs/ca.pem
...
Docker is up and running!
To see how to connect your Docker Client to the Docker Engine running on this virtual machine, run: docker-machine env default
elrlp12%
elrlp12% docker-machine ls
NAME ACTIVE DRIVER STATE URL SWARM DOCKER ERRORS
default - virtualbox Running tcp://192.168.99.100:2376 v18.06.0-ce
elrlp12%
```

### \*f) Configurer\*

```
elrlp12% docker-machine env default
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.100:2376"
export DOCKER_CERT_PATH="/Users/aaiche/.docker/machine/machines/default"
export DOCKER_MACHINE_NAME="default"
Run this command to configure your shell:
eval $(docker-machine env default)
elrlp12%
elrlp12% eval $(docker-machine env default)
elrlp12% set | grep --binary-files=text DOCKER
DOCKER_CERT_PATH=/Users/aaiche/.docker/machine/machines/default
DOCKER_HOST=tcp://192.168.99.100:2376
DOCKER_MACHINE_NAME=default
DOCKER_TLS_VERIFY=1
command='set | grep --binary-files=text DOCKER'
elrlp12%
elrlp12% docker-machine ls
NAME ACTIVE DRIVER STATE URL SWARM DOCKER ERRORS
default * virtualbox Running tcp://192.168.99.100:2376 v18.06.0-ce
elrlp12%
```

### \*[Note]\*

Si vous ouvrez un autre terminal, ces variables d'environnement ne seront pas mises en place.  
Il faut donc ajouter la commande shell `_eval $(docker-machine env default)_` dans votre profile shell

Par exemple

```
e1rlp12% echo $SHELL
/bin/zsh
e1rlp12% echo 'eval $(docker-machine env default)' >> ~/.zshrc
```

### **\*f) Versions installees...\***

```
e1rlp12% docker version
Client:
 Version: 18.06.0-ce
 API version: 1.38
 ...
Server:
 Engine:
 Version: 18.06.0-ce
 API version: 1.38 (minimum version 1.12)
 ...
```

### **\*g) Un petit test enfin avec l image hello world\***

```
e1rlp12% docker image ls
REPOSITORY TAG IMAGE ID CREATED SIZE
e1rlp12%
e1rlp12% docker container run hello-world
...
Hello from Docker!
...
e1rlp12%
e1rlp12% docker image ls
REPOSITORY TAG IMAGE ID CREATED SIZE
hello-world latest 2cb0d9787c4d 2 weeks ago 1.85kB
e1rlp12%
```

**\*[INSTALL docker, docker-machine, docker-compose, virtualbox COMME suggere dans le sujet cloud-1] . BESOIN D ESPACE\***

### **\*h) Besoin d espace dans votre espace de stockage perso ?\***

Par exemple, on peut déplacer certains répertoires qui consommeront de l'espace disque de votre `_Home_` vers `_sgoinfre_`

### **\*Arreter la VM\***

```
elrlp12% docker-machine stop default
Stopping "default"...
Machine "default" was stopped.
elrlp12%
elrlp12% docker-machine ls
NAME ACTIVE DRIVER STATE URL SWARM DOCKER ERRORS
default - virtualbox Stopped Unknown
elrlp12%
```

### **\*Deplacer ces repertoires\***

```
elrlp12% mv ~/.docker /sgoinfre/goinfre/Perso/$USERNAME
elrlp12%
```

### **\*Creer un lien symbolique\***

```
elrlp12% ln -s /sgoinfre/goinfre/Perso/$USERNAME/.docker ~/.docker
elrlp12%
elrlp12% ls -al ~/ | egrep docker
drwxr-xr-x 4 aaiche 2017_paris 136 Jul 23 16:28 .boot2docker/
lrwxr-xr-x 1 aaiche 2017_paris 38 Jul 25 18:24 .docker -> /sgoinfre/goinfre/Perso/aaiche/.docker
elrlp12%
```

### **\*Demarrer la VM\***

```
elrlp12% docker-machine start default
Starting "default"...
...
Started machines may have new IP addresses. You may need to re-run the `docker-machine env` command.
```

```
elrlp12% set | egrep --binary-files=text DOCKER
DOCKER_CERT_PATH=/Users/aaiche/.docker/machine/machines/default
DOCKER_HOST=tcp://192.168.99.100:2376
```

```
DOCKER_MACHINE_NAME=default
DOCKER_TLS_VERIFY=1
command='set | egrep --binary-files=text DOCKER'
elrlp12%
```

```
elrlp12% docker-machine ls
NAME ACTIVE DRIVER STATE URL SWARM DOCKER ERRORS
default * virtualbox Running tcp://192.168.99.100:2376 - v18.06.0-ce
```

## **\*[LABs] 1) Environnement:\***

### **\*Note:\***

**\*On utilise une version > 1.3: donc on peut utiliser la nouvelle syntaxe de docker\***

### **\*References:\***

**\*Lab tres ancien, mais interessant car c etait le debut...\***

[https://us.pycon.org/2016/site\\_media/media/tutorial\\_handouts/DockerSlides.pdf](https://us.pycon.org/2016/site_media/media/tutorial_handouts/DockerSlides.pdf)

```
elrlp12% docker version
Client:
 Version: 18.06.0-ce
 API version: 1.38
 Go version: go1.10.3
 Git commit: 0ffa825
 Built: unknown-buildtime
 OS/Arch: darwin/amd64
 Experimental: false

Server:
 Engine:
 Version: 18.06.0-ce
 API version: 1.38 (minimum version 1.12)
 Go version: go1.10.3
 Git commit: 0ffa825
 Built: Wed Jul 18 19:13:39 2018
 OS/Arch: linux/amd64
 Experimental: false
```

```
e1rlp12% docker-machine ls
NAME ACTIVE DRIVER STATE URL SWARM DOCKER ERRORS
default * virtualbox Running tcp://192.168.99.100:2376 - v18.06.0-ce
```

## **\*[LABs] 2) Hello World\***

**\*Lister toutes les images disponibles dans votre Docker Engine\***

```
e1rlp12% docker image ls -a
REPOSITORY TAG IMAGE ID CREATED SIZE
e1rlp12%
```

**\*Pull image busybox\***

```
e1rlp12% docker image pull busybox
...
e1rlp12% docker image ls
REPOSITORY TAG IMAGE ID CREATED SIZE
busybox latest 22c2dd5ee85d 9 days ago 1.16MB
```

**\*Create Container et RUN Container et executer une CMD shell\***

```
e1rlp12% docker container run busybox echo hello world
hello world
e1rlp12%
```

## **\*[LABs] 3) Un container plus utile\***

**\*Pull et 'entrer' dans le container\***

```
elrlp12% docker image pull ubuntu
...
elrlp12% docker image ls
REPOSITORY TAG IMAGE ID CREATED SIZE
ubuntu latest 74f8760a2a8b 9 days ago 82.4MB
busybox latest 22c2dd5ee85d 9 days ago 1.16MB
elrlp12% docker container run --interactive --tty ubuntu bash
root@1328812480d2:/# hostname
1328812480d2
root@1328812480d2:/#
root@1328812480d2:/# exit
exit
elrlp12%
```

#### **\*[LABs] 4) Faire qqchse avec notre container\***

##### **\*Run container\***

```
elrlp12% docker container run --interactive --tty ubuntu bash
root@238b2b4ed7cb:/# figlet hello
bash: figlet: command not found
root@238b2b4ed7cb:/#
```

Ok, on a besoin d installer `_figlet_`

##### **\*D'abord combien de packages avons nous dans ce container\***

```
root@238b2b4ed7cb:/# dpkg -l | wc -l
94
root@238b2b4ed7cb:/#
```

##### **\*Installons `_figlet_`, recomptons le nbre de packages, utilisons `_figlet_`\***

```
root@238b2b4ed7cb:/# apt-get update
root@238b2b4ed7cb:/# dpkg -l | wc -l
94
```



**\*Combien de container constuits a partir de l image `_ubuntu_` avons nous dans le disque?: 2\***

```
e1rlp12% docker container ls -a | egrep ubuntu
b2603f34dad0 ubuntu "bash" About a minute ago Exited (127) 10 seconds ago
sleepy_shockley
4973fde0adcd ubuntu "bash" 7 minutes ago Exited (130) About a minute ago
condescending_easley
```

## \*[LABs] 6) Demarrons un container en mode NON INTERACTIF\*

**\*Run directement sans \_pull'er\_ l'image, ce petit container: il affiche date et heure a chaque seconde\***

```
e1rlp12% docker container run jpetazzo/clock
Unable to find image 'jpetazzo/clock:latest' locally
latest: Pulling from jpetazzo/clock
a3ed95caeb02: Pull complete
ldb09adb5ddd: Pull complete
Digest: sha256:446edaa1594798d89ee2a93f660161b265db91b026491e4671c14371eff5eea0
Status: Downloaded newer image for jpetazzo/clock:latest
Thu Jul 26 12:46:03 UTC 2018
Thu Jul 26 12:46:04 UTC 2018
Thu Jul 26 12:46:05 UTC 2018
...
```

### \*Remarques:\*

- Docker a automatiquement \_pull'er\_ l'image jpetazzo/clock\_ avant de creer le container

```
e1rlp12% docker image ls | egrep jpet
jpetazzo/clock latest 12068b93616f 3 years ago 2.43MB
e1rlp12%
```

- Le container vas s'executer \_a jamais\_
- Pour stopper le container :

^C

## \*[LABs] 7) Demarrons un container en BACKGROUND\*

**\*Les containers peuvent etre demarres en background avec le flag `-d` ou bien `--detach`\***

```
e1rlp12% docker container run --detach jpetazzo/clock
0aa4c458a757677f9c77b6c1b23c5605970d5eb26d54a45c61362f7809e80d69
e1rlp12%
```

### **\*Remarques:\***

- On ne voit plus les logs
- Mais pas de soucis, on pourra les recuperer
- Docker nous donne le `_Container ID_`

**\*Comment voir tous les Containers qui sont demarres ?\***

```
e1rlp12% docker container ls
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
0aa4c458a757	jpetazzo/clock	"/bin/sh -c 'while d..."	3 minutes ago	Up 3 minutes		cocky_mayer

ou bien `_a l'ancienne mode_`,

```
e1rlp12% docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
0aa4c458a757	jpetazzo/clock	"/bin/sh -c 'while d..."	3 minutes ago	Up 3 minutes		cocky_mayer

**\*Demarrer le container en background nous donne le `_Container ID_`, mais lorsque je liste les containers on voit un ID qui n'est pas le meme\***

**\*En faite, lorsque on liste les containers via `_docker container ls_`, le ID est tronque\***

**\*Et si on veut afficher le `_Container ID_` en entier:\***

e1rlp12% docker container ls --no-trunc							
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES	
0aa4c458a757677f9c77b6c1b23c5605970d5eb26d54a45c61362f7809e80d69	jpetazzo/clock	"/bin/sh -c 'while date; do sleep 1; done'"	6 minutes ago	Up 6 minutes		cocky_mayer	

**\*Et si on ne veut afficher QUE le *\_Container ID\_* en entier:\***

```
e1rlp12% docker container ls --no-trunc --quiet
0aa4c458a757677f9c77b6c1b23c5605970d5eb26d54a45c61362f7809e80d69
```

**\*Ques autres flags utiles pour lister les containers:\***

**\*Lister tous les containers existants (demarres ou non):\***

```
e1rlp12% docker container ls --all
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
0aa4c458a757 jpetazzo/clock "/bin/sh -c 'while d..." 16 minutes ago Up 16 minutes cocky_mayer
034ea38812f0 jpetazzo/clock "/bin/sh -c 'while d..." 45 minutes ago Exited (0) 45 minutes ago
affectionate_allen
b2603f34dad0 ubuntu "bash" About an hour ago Exited (127) About an hour ago
sleepy_shockley 4973fde0adcd ubuntu "bash" About an hour ago Exited (130) About an hour ago
condescending_easley
ala149a7e765 busybox "echo hello world" 2 hours ago Exited (0) 2 hours ago
naughty_dubinsky
```

**\*Lister le dernier container demarre:\***

```
e1rlp12% docker container ls --latest
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
0aa4c458a757 jpetazzo/clock "/bin/sh -c 'while d..." 18 minutes ago Up 18 minutes cocky_mayer
```

**\*Lister uniquement les *\_Containers ID\_*:\***

```
e1rlp12% docker container ls --quiet
0aa4c458a757
```

**\*Combiner les flags:\***

```
e1rlp12% docker container ls --latest --no-trunc --quiet
0aa4c458a757677f9c77b6c1b23c5605970d5eb26d54a45c61362f7809e80d69
```

## \*[LABs] -8) Afficher les logs d'un container\*

**\*Verifions que le container démarre en background\***

```
e1rlp12% docker container ls
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
0aa4c458a757	jpetazzo/clock	"/bin/sh -c 'while d..."	32 minutes ago	Up 32 minutes		cocky_mayer

**\*Affichons les logs de ce container\***

```
e1rlp12% docker container logs 0aa4c458a757
Thu Jul 26 13:14:20 UTC 2018
Thu Jul 26 13:14:21 UTC 2018
Thu Jul 26 13:14:22 UTC 2018
...
```

**\*Remarques:\***

**\*On n pas ete oblige de preciser le \_Container ID\_ en entier, et on peut utiliser que les 2 premiers digits de ce container, voyons cela:\***

```
e1rlp12% docker container logs 0a
Thu Jul 26 13:14:20 UTC 2018
Thu Jul 26 13:14:21 UTC 2018
Thu Jul 26 13:14:22 UTC 2018
...
```

**\*On peut bien sur specifier le \_Container ID\_ en entier\***

```
e1rlp12% docker container ls --no-trunc --quiet
0aa4c458a757677f9c77b6c1b23c5605970d5eb26d54a45c61362f7809e80d69
e1rlp12% docker container logs 0aa4c458a757677f9c77b6c1b23c5605970d5eb26d54a45c61362f7809e80d69
Thu Jul 26 13:14:20 UTC 2018
Thu Jul 26 13:14:21 UTC 2018
Thu Jul 26 13:14:22 UTC 2018
...
```

### **\*[LABs] 9) Afficher les logs d'un container a la maniere de la commande shell `_tail_`\***

**\*Vous avez remaque que lorsqu'on a affiche les logs du container, on a ete inonde de logs\***

**\*Essayons ceci:\***

```
e1rlp12% docker container logs --tail 3 0a
Thu Jul 26 14:04:14 UTC 2018
Thu Jul 26 14:04:15 UTC 2018
Thu Jul 26 14:04:16 UTC 2018
e1rlp12%
```

- Le parametre `_3_` est le nombre de lignes a afficher
- `--tail` permet d afficher les dernieres lignes d un fichier (ici les logs du Container)

### **\*[LABs] 10) Afficher les logs EN TEMPS REEL d'un container a la maniere de la commande shell `_tail -f_`\***

**\*Essayons ceci:\***

```
e1rlp12% docker container logs --tail 1 --follow 0a
Thu Jul 26 14:22:50 UTC 2018
Thu Jul 26 14:22:51 UTC 2018
Thu Jul 26 14:22:52 UTC 2018
...
```

- Cela n'affiche que la dernière ligne des logs
- Puis continue d'afficher les logs en temps réel
- Utiliser `_^C (CTRL C)_` pour interrompre

## \*[LABs] 11) Arrêter (Mettre Fin) un container\*

**\*Il ya 2 manières de terminer notre container qui est `_detach'e_` (en background):\***

- Kill en utilisant la commande docker `_kill_`
- Stop en utilisant la commande docker `_stop_`

La première manière stoppe le container immédiatement en envoyant le signal `_KILL_`

La seconde est élégante (polie), on envoie le signal `_TERM_`, puis après 10 secondes on envoie le signal `_KILL_`

Remarque: le signal `_KILL_` ne peut pas être intercepté et `_termine de force_` le container

**\*Mettons fin à notre container:\***

**\*Vérifions d'abord qu'il est toujours démarré:\***

```
e1rlp12% docker container ls
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
0aa4c458a757	jpetazzo/clock	"/bin/sh -c 'while d..."	About an hour ago	Up About an hour		cocky_mayer

**\*Et mettons fin à ce container:\***

```
```e1rlp12% docker container kill 0aa4c458a757
0aa4c458a757```
```

Re-vérifions s'il tourne toujours:

```
e1rlp12% docker container ls
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

Remarque:

Le container est toujours dans le disque:

Listons tous les containers *_stoppes_*:

```
e1rlp12% docker container ls --all
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS          PORTS          NAMES
0aa4c458a757   jpetazzo/clock  "/bin/sh -c 'while d..."  2 hours ago   Exited (137)    2 minutes ago   cocky_mayer
```

[LABs] 12) Mettre un container en arriere-plan (*_background_*), le mettre en avant-plan, et le redemarrer

Nous avons jusqu'a present demarrer un container en avant-plan et en arriere-plan. Regardons maintenant comment:

*Mettre un container en arriere-plan (background)

*Mettre un container en avant-plan (foreground)

*Redemarrer un container arrete

[Note]:

Background and foreground

The distinction between foreground and background containers is arbitrary.

From Docker's point of view, all containers are the same.

All containers run the same way, whether there is a client attached to them or not.

It is always possible to detach from a container, and to reattach to a container (modifié)

[LABs] 12a) Mettre un container en arriere-plan (*_background_*)

a) Se *_detach'er_* d un container: le mettre un container en arriere-plan (*background*):

Avant de commencer nettoyons un peu nos containers ...

```
e1rlp12% docker container ls
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS          PORTS          NAMES
```

Demarrons un nouveau container:


```
e1rlp12% docker container run --interactive --tty jpetazzo/clock
Thu Jul 26 16:18:48 UTC 2018
Thu Jul 26 16:18:49 UTC 2018
Thu Jul 26 16:18:50 UTC 2018
...
```

Puis, _detachons_ nous de ce container en utilisant la sequence:

^P^Q

_[_Note]: ^ est le raccourci pour la touche clavier CTRL_

Enfin, Verifions si le container _tourne_ toujours en arriere-plan:

```
e1rlp12% docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
795ee0921765       jpetazzo/clock     "/bin/sh -c 'while d..." 3 minutes ago       Up 3 minutes                tender_franklin
```

[Attention]:

Se _detach'er_ d un container en faisant _^C_ ne fait pas la meme chose, cela arrete le container. Faites le test: a la place de faire _^P^Q_, faites _^C_ et essayer de vous _re'attacher_ sur le container

[LABs] 12b) Mettre un container en avant-plan (_foreground_)

a) S' _attach'er_ a un container: le mettre un container en avant-plan (_foreground_):

Avant de commencer regardons si le container tourne an arriere-plan:

```
e1rlp12% docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
795ee0921765       jpetazzo/clock     "/bin/sh -c 'while d..." 12 minutes ago       Up 12 minutes                tender_franklin
```

Puis, *_attach'ons_* nous de ce container en utilisant:

```
e1rlp12% docker container attach 795ee0921765
Thu Jul 26 16:33:11 UTC 2018
Thu Jul 26 16:33:12 UTC 2018
Thu Jul 26 16:33:13 UTC 2018
...
```

[LABs] 12c) Redemarrer un container arrete

Quand un container est arrete, il est dans l etat "*_Exited_*". Il peut etre re-demarre en utilisant la commande *_start_*

Avant de commencer demarrons et arretons un container

```
e1rl2p13% docker container run jpetazzo/clock
...
Sat Aug 18 13:20:54 UTC 2018
Sat Aug 18 13:20:55 UTC 2018
Sat Aug 18 13:20:56 UTC 2018
Sat Aug 18 13:20:57 UTC 2018
^C%
```

docker container ls --> ne donne rien

et *_docker container ls -a_* --> nous montre qu il y a un container et examinons son etat

```
e1rl2p13% docker container ls
CONTAINER ID      IMAGE               COMMAND             CREATED           STATUS              PORTS              NAMES
e1rl2p13% docker container ls -a
CONTAINER ID      IMAGE               COMMAND             CREATED           STATUS              PORTS              NAMES
d5e15fe7809c      jpetazzo/clock     "/bin/sh -c 'while d..." About a minute ago Exited (0) About a minute ago
sleepy_villani

e1rl2p13% docker container ls --filter "status=exited"

CONTAINER ID      IMAGE               COMMAND             CREATED           STATUS              PORTS              NAMES
d5e15fe7809c      jpetazzo/clock     "/bin/sh -c 'while d..." 2 minutes ago     Exited (0) 2 minutes ago
sleepy_villani
```

Re'startons le Container

```
e1r12p13% docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
e1r12p13% docker container start d5
d5
e1r12p13% docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
d5e15fe7809c        jpetazzo/clock     "/bin/sh -c 'while d..." 3 minutes ago       Up 1 second        ports              sleepy_villani
```

On peut aussi s'y connecter lorsque on le démarre
docker container start --interactive ...

[LABs] 13) INFO: image vs container

- Une image ne peut pas être modifiée
- Une image est un ensemble de layers (couches)
- Un container se crée à partir d'une image

Si une image ne peut pas être modifiée, comment la changer ?

- On ne peut pas changer une image

Alors on fait comment?

- On crée un container à partir de cette image
- On fait les changements dans ce container
- Une fois satisfait, on transforme ce container en une nouvelle couche (layer)
- Une nouvelle image est créée en empilant la nouvelle couche sur l'ancienne image

[LABs] 14) Créer une image from scratch

<https://www.youtube.com/watch?v=W5YNNMRI7cQ>

TBD

Creating the first images

There is a special empty image called scratch.

- It allows to build from scratch.

The docker import command loads a tarball into Docker.

- The imported tarball becomes a standalone image.
- That new image has a single layer.

Note: you will probably never have to do this yourself.

[LABs] 15) Creer d autres images

docker commit

- Saves all the changes made to a container into a new layer.
- Creates a new image (effectively a copy of the container).

docker build

- Performs a repeatable build sequence.
- This is the preferred method!

[LABs] 16) qqes commandes sur les images

Lister les images dans votre host

```
docker image ls
```

Chercher des images

```
docker search zookeeper
```

- Stars: popularite de l image
- Official: images deposees par Docker Inc
- Automated: images creees automatiquement par Docker Hub (la 'recette' de creation est toujours disponible)

Telecharger des images

- Explicitement:

```
docker pull
```

- Implicitement, l image est telechargee si elle n est pas disponible localement:

```
docker container run
```

tags

- Les images peuvent avoir un *_tag_*
- Un *_tag_* est une version ou une variante de l image
- *_tag latest_* donne la derniere version

[LABs] 17) Construire une image -- commit [Manual process = bad.]

Creer un container a partir d une image de base

```
docker container run --interactive --tty ubuntu bash
```

Installer manuellement du software, par exemple *_figlet_*

```
e1r12p13% docker container run --interactive --tty ubuntu bash
....
root@3b8105f1c7eb:/# apt-get update && apt-get install figlet && exit
...
```

```
The following NEW packages will be installed:
  figlet
....
exit
e1r12p13%
```

Lister les containers

```
e1r12p13% docker container ls --all
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
3b8105f1c7eb	ubuntu	"bash"	2 minutes ago	Exited (0)	About a minute ago	goofy_raman

Examiner la difference entre le container et l image de base

```
e1r12p13% docker container diff 3b8105f1c7eb
C /etc
C /etc/alternatives
A /etc/alternatives/figlet
...
```

- A: Added
- D Deleted
- C: Changed

Creer la nouvelle image a partir du container

****_docker commit_**** cree une nouvelle couche (layer) avec les changements effectues a l image de base et cree la nouvelle image en utilisant cette nouvelle couche

```
e1r12p13% docker commit 3b8105f1c7eb
sha256:e2c5435017a02a4b5ee051abce81ff0d6115e3cd2d380408ddd10d8030754a0e  <-- ID de la nouvelle image
e1r12p13%
```

Lister les images

```
e1r12p13% docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
------------	-----	----------	---------	------

Créer un nouveau container à partir de cette nouvelle image

```
e1r12p13% docker container run --interactive --tty e2c5435017a0
root@d261f64103f2:/# figlet coucou

  _/ _/ _/ \_/_/_/ _/ _/ \_/_/_/
 | ( ( ) | | | | ( ( ) | | | |
 \_/_/ \_/_/ \_/_/ \_/_/ \_/_/ \_/_/
```

$$\begin{array}{ccccccc} \diagup & \diagup & \diagdown & \diagdown & \diagup & \diagup & \diagdown & \diagdown \\ | & (_ | & (_) & | & | _ | & | & (_ | & (_) & | & | _ | & | \\ \diagdown & \diagdown & \diagup & \diagup & \diagdown & \diagdown & \diagup & \diagup & \diagdown & \diagdown \end{array}$$

Donner un nouveau tag a cette nouvelle image nous permet d'utiliser plus facilement l'image plutôt que d'utiliser son ID

```
e1r12p13% docker image ls
REPOSITORY          TAG                IMAGE ID           CREATED            SIZE
<none>              <none>            e2c5435017a0      3 minutes ago     126MB
...
e1r12p13% docker image tag e2c5435017a0 myfiglet
e1r12p13% docker image ls
REPOSITORY          TAG                IMAGE ID           CREATED            SIZE
myfiglet            latest            e2c5435017a0      3 minutes ago     126MB
...
```

[LABs] 18) Construire une image -- Dockerfile [Automated process = good.]

* **Dockerfile** * est une recette de creation d une image. Il contient une serie d instructions disant a Docker comment creer l image

docker image build construit l'image a partir du Dockerfile

Dockerfile doit etre contenu dans un repertoire ***VIDE***, car il cree un `_Contexte_`. Ce dernier var faire une sorte de `_tar_` et l'envoyer a Docker

1) _mkdir myimage; cd myimage; vim Dockerfile_

Dockerfile:

```
FROM ubuntu -----> indique 1 image de base
RUN apt-get update -----> chaque RUN sera execute durant le build
RUN apt-get install figlet -----> la commande passee a RUN NE DOIT PAS ETRE INTERACTIVE
```

2) Construire l'image

`_docker image build --tag myfiglet .`

`._` : indicates the location of the `_build context_`. `_build context_` est le repertoire contenant Dockerfile

```
e1rl2p13% docker image build --tag myfiglet .
Sending build context to Docker daemon  2.048kB
Step 1/3 : FROM ubuntu
latest: Pulling from library/ubuntu
....
Status: Downloaded newer image for ubuntu:latest
---> 735f80812f90
Step 2/3 : RUN apt-get update
---> Running in 2351151ab15d
....
Removing intermediate container 2351151ab15d
---> 6ded8cbaae87
suivante
Step 3/3 : RUN apt-get install figlet
---> Running in 6944b0c6da97
....
Removing intermediate container 6944b0c6da97
---> aeelfe21535f
Successfully built aeelfe21535f
Successfully tagged myfiglet:latest
```

<--- C est pour cela qu on cree un repertoire VIDE
<--- Image de Base

<--- Ce sera 1 ID de 1 image ubuntu et sera utilise comme image de base
<--- On cree un Container TEMPORAIRE pour executer le RUN
<--- SUPPRIMER le Container TEMPORAIRE
<--- Image conservee et Sera utilisee comme image de base pour l'instruction

<--- On cree un Container TEMPORAIRE pour executer le RUN
<--- SUPPRIMER le Container TEMPORAIRE
<--- CONTAINER FINAL
<--- CONTAINER FINAL
<--- On lui donne a notre image

3) Listons les images creees

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
myfiglet	latest	aeelfe21535f	2 minutes ago	126MB
ubuntu	latest	735f80812f90	3 weeks ago	83.5MB

[Note]:

Sending build context to Docker daemon 2.048 kB

- The build context is the `.` directory given to docker build.
- It is sent (as an archive) by the Docker client to the Docker daemon.

- This allows to use a remote machine to build using local files.
- Be careful (or patient) if that directory is big and your link is slow.

[Note]:

The history command lists all the layers composing an image.

For each layer, it shows its creation time, size, and creation command.

When an image was built with a Dockerfile, each layer corresponds to a line of the Dockerfile.

```
e1rl2p13% docker history myfiglet
IMAGE          CREATED          CREATED BY          SIZE          COMMENT
aeelfe21535f   11 minutes ago   /bin/sh -c apt-get install figlet   1.02MB
6ded8cbaae87   11 minutes ago   /bin/sh -c apt-get update           41.4MB
735f80812f90   3 weeks ago      /bin/sh -c #(nop)  CMD ["/bin/bash"] 0B
....
```

[LABs] 19) Dockerfile CMD

CMD définit la commande à lancer par défaut quand aucune commande n'est fournie

CMD peut apparaître à n'importe quelle ligne SAUF la première ligne

Essayons d'abord ceci:

```
Dockerfile:
FROM ubuntu
RUN apt-get update
RUN apt-get install figlet
CMD figlet -f small hello

docker image build --tag myfiglet
```

Résultat: A chaque fois que le container est lancé, l'instruction CMD est exécutée par défaut

```
e1rl2p13% docker container run --interactive --tty myfiglet
_ _ _
```

*On peut définir plusieurs *_CMD_, et chaque *_CMD_ efface la précédente. Par conséquent avoir plusieurs *_CMD_ est inutile*

Essayons ceci:

```
FROM ubuntu
RUN apt-get update
RUN apt-get install figlet
CMD figlet -f small hello
CMD figlet -f lean hello
CMD figlet -f script hello
```

```
e1r12p13% docker container run --interactive --tty myfiglet
```



*** 'Ecraser' la commande par defaut definie par CMD***

```
elr12p13% docker container run --interactive --tty myfiglet bash
root@db79594a8c1e:/#
```

Ici, ***_bash_*** remplace la commande definie dans ***_CMD_***

*** Differences entre CMD et RUN***

CMD : commande est exécutée lors du lancement du container (command is executed at run-time)

RUN : commande est executee lors de la creation (= build) de l image (command is executed at build-time)

A horizontal timeline with a dashed line and three vertical tick marks. The first tick mark is labeled 'CMD', the second is labeled 'RUN', and the third is unlabeled.

Run-time	Build-time
Run commands in containers at launch time	Add layers to images to images
Equivalent of <code><command></code> <code>docker run <args> <command></code>	Used to install apps
One CMD per Dockerfile	

* Deux styles de syntaxe pour les commandes CMD*

Shell Form	Exec Form
Commands are expressed the same way as shell commands	Recommended Method
Commands are prepended with by <code>"/bin/sh -c"</code>	JSON array style <code>["command", "arg13, ...]</code>
e.g. : CMD echo "hello world"	Allows commands to be run inside of containers that don't need a shell
Variable expansion: e.g. : CMD echo \$var1	Avoids ?string munging? by the shell
...	No shell features: No variable expansion No special characters (<code>&&</code> , <code> </code> , <code>></code> , ...)

[LABs] 20) Dockerfile ENTRYPOINT

a) ENTRYPOINT definit une commande de base (et ses parametres) pour le container

Dockerfile:

```
FROM ubuntu
RUN apt-get update
RUN apt-get install figlet
ENTRYPOINT ["figlet", "-f", "script", "coucou"]
```

build:

```
elr12p13% docker image build --tag myfiglet .
```

```
*run:*
```

```
elr12p13% docker container run --interactive --tty myfiglet
```

*c1ccc(cc1)-c2ccccc2

b) Est ce qu'on peut écraser la commande définie par ENTRYPOINT par la commande définie lors du lancement du container ==> NON

run:

```
elr12p13% docker container run --interactive --tty myfiglet /bin/bash
```

C1=CC=C(C=C1)C(=O)Nc2ccc(cc2)C(=O)Nc3ccccc3

==> ttes les commandes entrees lors de l execution sont ajoutees a la commande entryptpoint

c) 'ecraser' la commande contenue dans ENTRYPOINT

```
elr12p13% docker container run --interactive --tty --entrypoint /bin/bash myfiglet
root@e2c613e5738a:/#
```

d) Comme pour CMD, ENTRYPOINT peut être mise sur n'importe quelle ligne sauf la première et chaque ENTRYPOINT remplace la précédente

[LABs] 21) Dockerfile ENTRYPOINT et CMD

La commande ENTRYPPOINT sera la commande de base

La commande CMD contient le(s) parametre(s) par defaut a la commande de base

```
Dockerfile:
FROM ubuntu
RUN apt-get update
RUN apt-get install figlet
ENTRYPOINT ["figlet", "-f", "script"]
CMD ["hello", "world"]
```

build:

```
elr12p13% docker image build --tag myfiglet .
```

run:

```
elr12p13% docker container run --interactive --tty myfiglet
```

*** 'Ecraser 'les parametres contenus daans CMD par la commande en ligne***

```
elr12p13% docker container run --interactive --tty myfiglet hola el mundo
```

l'annexe 22 de l'annexe 22

[LABs] 22) Copier des fichiers durant le build de l'image

COPY permet de copier un fichier ou repertoire durant le build de l'image

Build un container qui compile le programme C suivant et se trouvant localement:

```
hello.c:
int main () {
    puts("Hello, world!");
    return 0;
}
```

Dockerfile:

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y build-essential      --> contient le compilateur, option "-y" car la commande passee a RUN NE DOIT PAS ETRE
INTERACTIVE
COPY hello.c /
RUN make hello
CMD /hello
```

Build context:

```
e1r12p13% ls
Dockerfile  hello.c
```

build:

```
e1r12p13% docker image build --tag myhello .
```

run:

```
e1r12p13% docker container run myhello
Hello, world!
e1r12p13%
```

[INFO] 23) ADD vs COPY

ADD est une instruction plus ancienne que ***COPY***.

En plus de copier des fichiers ou répertoires, ***ADD*** permet :

- copier et "dezipper" automatiquement un fichier "zippe" pourvu que le format soit reconnu (tar, gzip, bzip2, ...)
 ADD /foo.tar.gz /tmp//tmp/
- copier une URL
 ADD http://foo.com/bar.go /tmp/main.go

[NOTE:]* *COPY est la méthode ***recommandée***

https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/#add-or-copy

[LABs] 24) Copier un fichier ou répertoire du host vers le container ou vice-versa.

a) Copier fichier Container --- vers ---> host:

*** Créons un Container et Utilisons son _bash_:**

```
e1r12p13% docker container run --name myCtnr --interactive --tty ubuntu /bin/bash
```

*** Créons un Fichier dans _myCtnr_:**

```
root@d602541eebd3:/# ls
bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys tmp usr var
root@d602541eebd3:/# echo "Ce fichier est cree dans le container et s appelle new_file.txt" > new_file.txt
root@d602541eebd3:/# ls
bin boot dev etc home lib lib64 media mnt new_file.txt opt proc root run sbin srv sys tmp usr var
root@d602541eebd3:/# exit
```

*** Copions le fichier du Container dans le host:***

```
e1rl2p13% ls
e1rl2p13%
e1rl2p13% docker container cp myCtner:/new_file.txt .
e1rl2p13% ls
new_file.txt
e1rl2p13% cat new_file.txt
Ce fichier est cree dans le container et s appelle new_file.txt
```

*** Et verifions que nous avons bien reçu le fichier:***

```
e1rl2p13% ls
new_file.txt
e1rl2p13% cat new_file.txt
Ce fichier est cree dans le container et s appelle new_file.txt
```

b) Copier fichier host --- vers ---> Container

*** Pas de magie: la commande est:***

```
docker container cp ./new_host_file.txt myCtner:/new_host_file.txt
```

[Note]: Le container n a pas besoin d etre demarre

[LABs] 25) Copier un fichier repertoire d un container vers un autre container

TBD

[LABs] 26) Nommage d'un container

***a) Un container possede: ***

- un identifiant
- un nom par défaut (si l'on ne spécifie pas de nom lorsque on instancie un container). Ce nom est la concatenation de :
 - Une humeur (furious, goofy, suspicious, boring...)
 - Un nom d'un inventeur célèbre (tesla, darwin, wozniak...)

b) Specifier lors de l'instanciation du container:

```
e1r12p13% docker container run --name ticktock jpetazzo/clock
^C%
e1r12p13% docker container ls -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
NAMES					
bbb858f0f835	jpetazzo/clock	"/bin/sh -c 'while d..."	27 seconds ago	Exited (0) 23 seconds ago	
ticktock					

```
^
|
|
```

[Note]:

Si l'on nomme un container avec un nom d'un container existant, Docker refusera de créer le container.
Ceci assure l'unicité d'une ressource

```
e1r12p13% docker container run --name ticktock jpetazzo/clock
docker: Error response from daemon: Conflict. The container name "/ticktock" is already in use by container
"bbb858f0f83550877f25d838494fb00a986d81df3ee3c098d2ea16589cabc2bb". You have to remove (or rename) that container to be able to reuse
that name.
See 'docker run --help'.
```

c) Renommer un container (à partir de la version Docker 1.5 [fév. 2015])

```

elr12p13% docker container rename ticktock tic-tac
elr12p13% docker container ls -a

```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
bbb858f0f835	jpetazzo/clock	"/bin/sh -c 'while d..."	2 minutes ago	Exited (0) 2 minutes ago		tic-tac
						^

[INFO] 27) JSON: JavaScript Object Notation

Easy for humans to read and write
Easy for machines to parse and generate

*Data Types: *

- Number
- String: use of double quotes
- Boolean: True or False
- Array: ordered list of 0 or more values, use of []
- Object: unordered collection of key/value pairs
- Null: empty value

*JSON Syntax Rules: *

- Uses key/value pairs - {"Name": "Brad"} is an object - key = "Name" and value = "Brad"
- Note also that both key and values are using double quotes
- Must use the specified data types
- File type is `_.json_`
- MIME type associated with JSON is "Application/json"

*JSON Example: **JSON Example: *

```

{
    "name": "Brad Traversy",
    "age": 35,
    "address": {
        "street": "5 main st",

```

```

<---- object
<---- key/value pairs. Value has a Data Type = String
<---- Value has a Data Type = Number (No double quotes)
<---- Value has a Data Type = Object (Embedded Object)

```

```

        "city": "Boston"
      },
      "children": ["Brianna", "Nicholas"]    <---- Array of Strings
    }
  }
}

```

[INFO] 28) JSON et utilitaire jq

*[reference]: *

https://www.youtube.com/watch?v=_ZTibHotSew

***a) install jq: ***

```
brew install jq
```

***b) Online Debugging tool: <https://jqplay.org/> ***

***c) Comment fonctionne jq: ***

```

+-----+           +-----+           +-----+
| Input   | -----> | filter  | -----> | Output   |
+-----+           +-----+           +-----+

```

***d) extraire la totalite de l input: ***

```

input:
  objet
  {
    "x": 1,
    "y": 2
  }

```

```

e1rl2p13% echo '{"x": 1, "y": 2}' | jq '.'
^
|

```

La totalite de l'input ____/

```
{  
  "x": 1,  
  "y": 2  
}
```

***e) extraire une "propiete" d un objet: ***

```
input:  
  objet  
  {  
    "x": 1,  
    "y": 2  
  }
```

```
elr12p13% echo '{"x": 1, "y": 2}' | jq '.x'  
1  
propiete 'x' de l'input ____/
```

***f) extraire une "propiete" d un objet imbrique dans un autre objet: ***

```
input:  
  objet  
  {  
    "x": {  
      "y": "z"  
    }  
  }
```

```
elr12p13% echo '{"x": {"y": "z"}}' | jq '.x.y'  
"z"  
Le 2eme '.' est le resultat est  
l'objet imbrique ____/
```

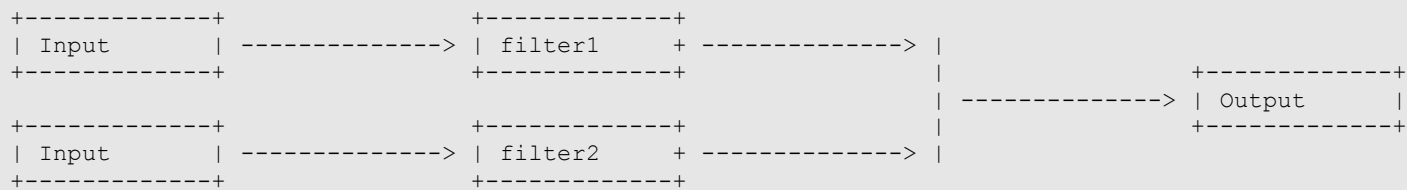
***g) Plusieurs filtres a la fois (l un apres l autre): ***



```
input :
  object
  {
    "x": {
      "y": "z"
    }
  }
```

```
elr12p13% echo '{"x": {"y": "z"}}' | jq '.x'
{
  "y": "z"
}
elr12p13% echo '{"x": {"y": "z"}}' | jq '.x | .y'
"z"
```

***h) Plusieurs filtres a la fois (en parallele): ***



```
input :
  object
  {
    "x": {
      "y": "z"
    }
  }
```

```
}
```

```
e1r12p13% echo '{"x": 1}' | jq '. , .'
```

```
{
  "x": 1
}
{
  "x": 1
}
```

[LABs] 29) Inspecter un container: jq

****_docker container inspect_**** donne des informations detaillees sur le container. Ces informations sont rendues en JSON

***Exple: Quelle est l image utilisee par le Container ? ***

```
e1r12p13% docker container ls -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
bbb858f0f835	jpetazzo/clock	"/bin/sh -c 'while d..."	About an hour ago	Exited (0) About an hour ago		tic-tac

```
e1r12p13% docker container inspect tic-tac | jq '[0].Config.Image'
```

```
"jpetazzo/clock"
```

***Exple: Quelle est le code de retour (_Exit Status_) lorsque le Container a ete arrete ? ***

```
e1r12p13% docker container inspect tic-tac | jq '[0].State.ExitCode'
```

```
0
```

[LABs] 30) Inspecter un container: GO template

[references]:

<https://golang.org/pkg/text/template/>
<https://container-solutions.com/docker-inspect-template-magic/>
<https://container42.com/2016/03/27/docker-quicktip-7-psformat/>

```
e1r12p13% docker container ls -a
CONTAINER ID        IMAGE               COMMAND                  CREATED              STATUS              PORTS              NAMES
bbb858f0f835       jpetazzo/clock     "/bin/sh -c 'while d..." About an hour ago    Exited (0) About an hour ago
e1r12p13% docker container inspect --format 'The name of this container is : {{.Name}}' tic-tac
The name of this container is : /tic-tac
e1r12p13%
```

- ***_.*** : represente toute les infos en entree (=input)
- ***_Name_*** : propriete Name

***Ques commandes a essayer: ***

```
$ docker ps --format '{{.Names}}\t{{.Image}}'
$ docker ps --format 'table {{.Names}}\t{{.Image}}'
$ watch -n 2 'docker ps --format "table {{.ID}}\t {{.Image}}\t {{.Status}}"'
$ docker images --format "{{.ID}}: {{.Repository}}"
$ docker images --format "table {{.ID}}\t{{.Repository}}\t{{.Tag}}"
$ docker inspect --format='{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' $INSTANCE_ID
$ docker inspect --format='{{range .NetworkSettings.Networks}}{{.MacAddress}}{{end}}' $INSTANCE_ID
$ docker inspect --format='{{range $p, $conf := .NetworkSettings.Ports}} {{ $p }} -> {{(index $conf 0).HostPort}} {{end}}' $INSTANCE_ID
$ docker inspect --format='{{(index (index .NetworkSettings.Ports "8787/tcp") 0).HostPort}}' $INSTANCE_ID
$ docker inspect --format='{{range .Config.Env}} -e "{{.}}" {{end}} {{range .Mounts}} -v {{.Source}}:{{.Destination}}{{if .Mode}}:{{.Mode}}{{end}}{{end}} -ti {{.Config.Image}}' $CONTAINER_ID_OR_NAME
```

[LABs] 31a) Network Basics: port mappings

[Info: port vs adresse ip]:

Prenons exemple sur une administration telle que la Mairie

- Une Mairie a une adresse **== *adresse ip***
- Une Mairie propose plusieurs services, etat civil,... **== *port***
- Il arrive que la Mairie soit ouverte mais qu'un service est ferme, on peut donc lister tous les services **== (ports) a l'ecoute**

```
netstat -anpo | egrep -i listen
```

*Lancer un container `_nginx_` web server a partir de l image `_nginx_`: *

```
elr12p13% docker container run --detach --publish-all nginx``
```

***Examiner les ports exposes ***

***a) 'Entrer' dans le container: ***

```
elr12p13% docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
d06d6bd92e54        nginx              "nginx -g 'daemon of..." 6 seconds ago       Up 5 seconds        0.0.0.0:32768->80/tcp  wonderful_ramanujan
elr12p13% docker container exec --interactive --tty wonderful_ramanujan bash
root@d06d6bd92e54:/#
```

[Note]:

- La commande `*_exec_*` est utilisé pour debugger dans un container
- A partir de la version Docker 1.3
 - *_exec_* permet d executer une commande a l interieur d un container démarré
 - *_exec_* n est pas censée etre utilisé en Production

***b) Si 'netstat' n est pas installe, alors: ***

```
root@d06d6bd92e54:/# apt-get update; apt-get install net-tools
```

***c) Examiner les ports a l ecoute: ***

```
root@d06d6bd92e54:/# netstat -anpo | egrep -i listen
tcp        0      0 0.0.0.0:80          0.0.0.0:*        LISTEN      1/nginx: master pro  off (0.00/0/0)
           ^
           |__ il y a un service utilisant TCP qui est a l ecoute sur le port 80
root@d06d6bd92e54:/# exit
```


[LABs] 31b) Network Basics: port mappings

*d) Il faut associer un port du host vers le port à l'écoute dans le Container: *

=> C est ce qu'on a fait en lançant le Container, cf. ci-dessus. Voici la commande que l'on lance :

```
e1rl2p13% docker container run --detach --publish-all nginx
```

Option qui nous permet d'associer le port
du Container à un port du host

[Note]: Cette option ne nous permet pas d'attribuer un port explicite sur le host

*On peut voir le résultat de cette association (mapping des ports) en : *

```
e1rl2p13% docker container ls
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
d06d6bd92e54	nginx	"nginx -g 'daemon of...'"	4 minutes ago	Up 4 minutes	0.0.0.0:32768->80/tcp

port du host _____
port du Container _____

*Comment lire ce mapping: *

+-----+-----+		+-----+-----+
Host ip port	----->	Container ip port
+-----+-----+		+-----+-----+
La requête entre par		La requête est
la	----->	traitée dans le
		container
+-----+-----+		+-----+-----+

*Comment lire autrement ce mapping - Remarquez l'ordre des ports: c est différent !!! *

```
e1rl2p13% docker container port wonderful_ramanujan
```

```
80/tcp -> 0.0.0.0:32768

elr12p13% docker container port wonderful_ramanujan 80
0.0.0.0:32768
```

***Acceder au serveur _nginx_ de Ce container: ***

```
elr12p13% curl $(docker-machine ip default):32768
...
<title>Welcome to nginx!</title>
...
```

ou bien

Utiliser votre browser

[LABs] 32) Network Basics: port mappings choisir soi-meme le port

***Verifier que le port de votre choix est libre dans votre host: ***

- Par exple, si on veut utiliser le port 4242 de notre host:

```
elr12p13% netstat -an | egrep 4242
elr12p13%
==> Aucun service n utilise le port 4242 dans notre host
```

Lancer le container avec le port host de votre choix, par exple: si on choisit le port 4242

```
elr12p13% docker container run --detach --publish 4242:80 nginx

elr12p13% docker container ls
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
598d480a3880	nginx	"nginx -g 'daemon of..."	9 seconds ago	Up 8 seconds	0.0.0.0:4242->80/tcp	stupefied_noyce

***Verifions: ***

```
e1rl2p13% curl $(docker-machine ip default):4242
...
<title>Welcome to nginx!</title>
...
```

ou bien

Utiliser votre browser

***[LABs] 33) Network Basics: chercher l'adresse ip d'un container en l'inspectant ***

```
$ docker inspect --format '{{ .NetworkSettings.IPAddress }}' <yourContainerID>
```

***Par exple: ***

```
e1rl2p13% docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
598d480a3880        nginx              "nginx -g 'daemon of..." 8 minutes ago       Up 8 minutes       0.0.0.0:4242->80/tcp
stupefied_noyce

e1rl2p13% docker inspect --format '{{ .NetworkSettings.IPAddress }}' stupefied_noyce
172.17.0.3
```

***Et si on la ping'gue: elle est ??? ***

```
ping cette adresse
```

***[Note]: ***

- ***C'est une adresse privée, car elle commence par: ***

```
172.16.xx.xx
```

- ***D autres adresses ip privees: ***

```
10.0.xx.xx --> 10.255.xx.xx  
172.16.xx.xx --> 172.31.xx.xx  
192.168.xx.xx --> 192.168.xx.xx
```

[LABs] 34) Network Basics: driver bridge (=default)

Il y a 3 trois networks par default:

- *_bridge_*
- *_host_*
- *_none_*

- ***Examinons les *_networks_* disponible: ***

```
e1r12p13% docker network ls  
NETWORK ID          NAME                DRIVER              SCOPE  
4795011ead9d        bridge             bridge              local  
d54c6c355849        host               host                local  
42020a52f400        none              null                local
```

- ***[Note]:***

=> le network de type **_bridge_** porte le meme nom **_bridge_**

- ***Quel est le subnet du reseau bridge: inspectons cette ressource: ***

```
e1r12p13% docker network inspect bridge --format '{{range .IPAM.Config}} {{.Subnet}} {{end}}'  
172.17.0.0/16
```

***Quels sont les containers attaches au reseau bridge: On fait un peu de nettoyage avant: ***

```
e1rl2p13% docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
e1rl2p13%
e1rl2p13% docker network inspect -f '{{ range $key, $value := .Containers }} {{ printf "%s:\t%s\n" $value.Name $value.IPv4Address }}{{ end }}' bridge
e1rl2p13%
==> Aucun Container sur le reseau par default bridge. Normal: car il n y a aucun Container
```

***Creons qqes containers (que l on nomme ici _web1_, _web2_, et _web3_) sans preciser le reseau - ils seront par default attaches au reseau _bridge_: ***

```
e1rl2p13% docker container run --detach --name=web1 --publish 1111:5000 streamweaver/dfdemo
e1rl2p13% docker container run --detach --name=web2 --publish 2222:5000 streamweaver/dfdemo
e1rl2p13% docker container run --detach --name=web3 --publish 3333:5000 streamweaver/dfdemo

e1rl2p13% docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
2822e2117184       streamweaver/dfdemo "python run.py"     12 seconds ago     Up 11 seconds      0.0.0.0:3333->5000/tcp web3
fb0e1e80daf0       streamweaver/dfdemo "python run.py"     32 seconds ago     Up 32 seconds      0.0.0.0:2222->5000/tcp web2
e36208f7c516       streamweaver/dfdemo "python run.py"     About a minute ago Up About a minute   0.0.0.0:1111->5000/tcp web1
```

[LABs] 34b) Network Basics: driver bridge (=default)

***Maintenant Rexaminons les containers attaches au reseau _bridge_: ***

```
e1rl2p13% docker network inspect -f '{{ range $key, $value := .Containers }} {{ printf "%s:\t%s\n" $value.Name $value.IPv4Address }}{{ end }}' bridge
web1:    172.17.0.2/16
web2:    172.17.0.3/16
web3:    172.17.0.4/16

==> Les Containers SONT tous DANS LE MEME sous reseau: 172.17.0.0
```

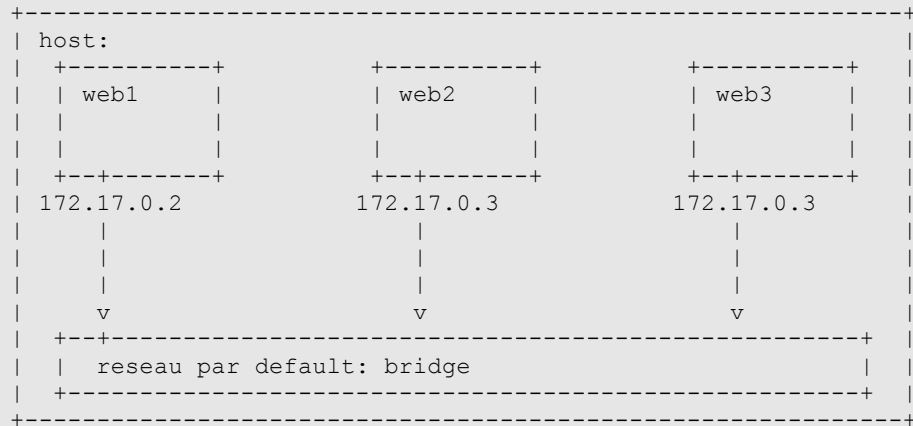
***Entrons dans le container _web1_ et ping'gons un autre Container: ***

```
elr12p13% docker container exec -it web1 /bin/sh
/opt/flaskapp # ping -c2 172.17.0.3; ping -c2 172.17.0.4
PING 172.17.0.3 (172.17.0.3): 56 data bytes
64 bytes from 172.17.0.3: seq=0 ttl=64 time=0.089 ms
64 bytes from 172.17.0.3: seq=1 ttl=64 time=0.087 ms
...
PING 172.17.0.4 (172.17.0.4): 56 data bytes
64 bytes from 172.17.0.4: seq=0 ttl=64 time=0.083 ms
64 bytes from 172.17.0.4: seq=1 ttl=64 time=0.094 ms
...
/opt/flaskapp # exit
```

***On peut aussi faire: ***

```
docker container exec -it web1 ping -c3 172.17.0.3
```

***Voici en realite ce que l'on vient de creer en utilisant le network par default `_bridge_`: ***



[LABs] 35) Network Basics: creer un network driver bridge (=default)

***Creeons un nouveau reseau et appelons le `_myBridgeNetwork_` :**

```

elrl2p13% docker network create myBridgeNetwork
57f1ec7057dcfc1275a4c33f5c83ce30d4e23026870a181764ee7fba587bc86e
elrl2p13% docker network ls
  NETWORK ID          NAME           DRIVER          SCOPE
  4795011ead9d        bridge        bridge         local
  d54c6c355849        host         host          local
>> 57f1ec7057dc        myBridgeNetwork bridge         local      <<<<<<
  42020a52f400        none         null          local

```

***Quel est le subnet du reseau bridge: ***

```

elrl2p13% docker network inspect myBridgeNetwork --format '{{range .IPAM.Config}} {{.Subnet}} {{end}}'
172.18.0.0/16

==> Resultat : c est un nouveau sous-reseau

```

***Quels sont les containers attaches au reseau _myBridgeNetwork_? ***

```

elrl2p13% docker network inspect -f '{{ range $key, $value := .Containers }}{{ printf "\t%s:\t%s\n" $value.Name $value.IPv4Address }}{{ end }}' myBridgeNetwork
elrl2p13%

==> Result: aucun

```

***Creons qqes containers attaches au reseau _myBridgeNetwork_: ***

```

docker container run --detach --name=web4 --publish 4444:5000 --network myBridgeNetwork streamweaver/dfdemo
docker container run --detach --name=web5 --publish 5555:5000 --network myBridgeNetwork streamweaver/dfdemo
docker container run --detach --name=web6 --publish 6666:5000 --network myBridgeNetwork streamweaver/dfdemo

```

***Maintenant quels sont a nouveau les containers attaches au reseau _myBridgeNetwork_: ***

```

elrl2p13% docker network inspect -f '{{ range $key, $value := .Containers }}{{ printf "%s:\t%s\n" $value.Name $value.IPv4Address }}{{ end }}' myBridgeNetwork
web4:      172.18.0.2/16

```

```
web5:    172.18.0.3/16
web6:    172.18.0.4/16
```

==> Remarque : C est un nouveau subnet : 172.18.0.0

***Voici en realite ce que l'on vient de creer en utilisant le network cree `_myBridgeNetwork_` : ***

```
+-----+
| host:                                     |
| +-----+       +-----+       +-----+ |
| | web4 |       | web5 |       | web6 | |
| |     |       |     |       |     | |
| +-----+       +-----+       +-----+ |
| 172.18.0.2       172.18.0.3       172.18.0.4 |
| |         |       |         |       |         |
| |         |       |         |       |         |
| |         v       |         v       |         v |
| +-----+-----+-----+ |
| | reseau cree: myBridgeNetwork | |
| +-----+-----+-----+ |
+-----+
```

***[LABs] 36) Network Basics: network driver null appele 'none' ***

Reseau utilise pour ISOLER des Containers

***Quel est le subnet du reseau `_none_` : ***

```
e1rl2p13% docker network inspect none --format '{{range .IPAM.Config}} {{.Subnet}} {{end}}'
e1rl2p13%
```

==> Resulat : rien

***Quels sont les containers attaches au reseau `_none_` ? ***


```
e1rl2p13% docker network inspect -f '{{ range $key, $value := .Containers }}{{ printf "\t%s:\t%s\n" $value.Name $value.IPv4Address }}{{ end }}' none
e1rl2p13%

==> Resultat : Aucun
```

***Creons qqes containers attaches au reseau _none_ : ***

```
e1rl2p13% docker container run --detach --name=web7 --publish 7777:5000 --network none streamweaver/dfdemo
0d64b95d01b7793f18bce1a052e34b1134c1c83e9e3f81f5d8bfe5adf74eccb3
e1rl2p13% docker container run --detach --name=web8 --publish 8888:5000 --network none streamweaver/dfdemo
c8a31bf7c1f3038d59f5ffcb071fcbda96621f0611ff3a6e42557d59104ff4d8
e1rl2p13%
```

***Est ce que ces containers ont une interface reseau? ***

```
e1rl2p13% docker container exec web7 ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
...
    inet 127.0.0.1/8 scope host lo
...

==> Resultat: NON, ils n ont que la loopback. Cf. Note
```

[Note]:

*_La ***loopback***, c est comme si vous mettez un bouchon a la place du cable reseau de votre PC. Tous les paquets IP_*
envoyees a la loopback traversent les differentes couches reseaux de votre PC (differeents drivers), et quand ils arrivent
au bouchon, ils remontent les memes couches. Car votre PC a cette adresse IP.
Ce qui est important a noter, c est qu on utilise cette adresse IP loopback pour descendre ttes les couches reseaux sans
jamais sortir a l exterieur et ainsi faire du test en local

***Maintenant quels sont a nouveau les containers attaches au reseau _none_ : ***

```
e1rl2p13% docker network inspect -f '{{ range $key, $value := .Containers }}{{ printf "%s:\t%s\n" $value.Name $value.IPv4Address }}{{ end }}' none
web7:
web8:
```

==> Resultat: les containers web7 et web8 sont dans le reseau none mais n ont aucune adresse ip

***Voici en realite ce que l'on vient de creer en utilisant le network cree _none_: ***

```
+-----+
| host:                                     |
| +-----+           +-----+           |
| | web7   |           | web8   |           |
| |         |           |         |           |
| +-----+           +-----+           |
|   x           x           |
|   ^           ^           |
|   |           |           |
|   |           |           |
| x: bouchon                                     |
| Seule 1 interface loopback existe             |
| Pas de possibilite pour ces Containers de communiquer avec |
| l exterieur                                     |
+-----+
```

[LABs] 37) Network Basics: Deconnecter un container d un reseau

***Quels sont les containers attaches au reseau bridge: ***

```
e1rl2p13% docker network inspect -f '{{ range $key, $value := .Containers }}{{ printf "%s:\t%s\n" $value.Name $value.IPv4Address }}{{ end }}' bridge
web3:      172.17.0.4/16
web1:      172.17.0.2/16
web2:      172.17.0.3/16
```

***Quelle est l adresse ip de _web3_: ***

```
e1rl2p13% docker exec web3 hostname -i
172.17.0.4
```

On peut aussi lister les interfaces ethernet de `_web3_`:

```
e1rl2p13% docker exec web3 ip addr show
1: lo: ...
...
    inet 127.0.0.1/8 scope host lo
...
19: eth0@if20: ...
...
    inet 172.17.0.4/16 brd 172.17.255.255 scope global eth0
...
```

***Deconnecter `_web3_` du reseau bridge? ***

```
e1rl2p13% docker container ls | egrep web3
2822e2117184      streamweaver/dfdemo  "python run.py"      2 hours ago         Up 2 hours           0.0.0.0:3333->5000/tcp
web3

==> Remarque: web3 est UP

e1rl2p13% docker network disconnect bridge web3
```

***Maintenant quels sont les containers attaches au reseau bridge: ***

```
e1rl2p13% docker network inspect -f '{{ range $key, $value := .Containers }}{{ printf "%s:\t%s\n" $value.Name $value.IPv4Address }}{{ end }}' bridge
web1:      172.17.0.2/16
web2:      172.17.0.3/16
```

***Maintenant, examinons l'adresse ip de `_web3_` ***

La commande `_hostname -i_` ne fonctionne pas ici (En faite elle utilise le fichier `_etc/hosts_...`). On peut lister les interfaces ethernet de `_web3_` en utilisant :

```
e1rl2p13% docker exec web3 ip addr show
1: lo: ...
...
```

```
    inet 127.0.0.1/8 scope host lo
    ...

==> Resultat: il n y a plus d interface reseau excepte la loopback
```

***On peut egalement retirer les Containers `_web's_...` de `_none_` et de `_myBridgeNetwork_` ***

[LABs] 38) Network Basics: Connecter un container a un reseau

***Quels sont les containers attaches au reseau `_myBridgeNetwork_`: ***

```
e1rl2p13% docker network inspect -f '{{ range $key, $value := .Containers }}{{ printf "%s:\t%s\n" $value.Name $value.IPv4Address }}{{ end }}' myBridgeNetwork
web4:    172.18.0.2/16
web5:    172.18.0.3/16
web6:    172.18.0.4/16
```

***Quelle est l adresse ip de `_web3_` qui n appartient a aucun reseau: ***

On peut aussi lister les interfaces ethernet de `_web3_` en utilisant:

```
e1rl2p13% docker exec web3 ip addr show
1: lo: ....
    ...
    inet 127.0.0.1/8 scope host lo
    ...
```

***Deconnecter web3 du reseau `_myBridgeNetwork_`? ***

```
e1rl2p13% docker network connect myBridgeNetwork web3
```

***Maintenant quels sont les containers attaches au reseau `_myBridgeNetwork_`: ***

```
e1rl2p13% docker network inspect -f '{{ range $key, $value := .Containers }}{{ printf "%s:\t%s\n" $value.Name $value.IPv4Address }}{{ end }}' myBridgeNetwork
web4:      172.18.0.2/16
web5:      172.18.0.3/16
web6:      172.18.0.4/16
web3:      172.18.0.5/16
```

***Quelle est l'adresse ip de `_web3_`? ***

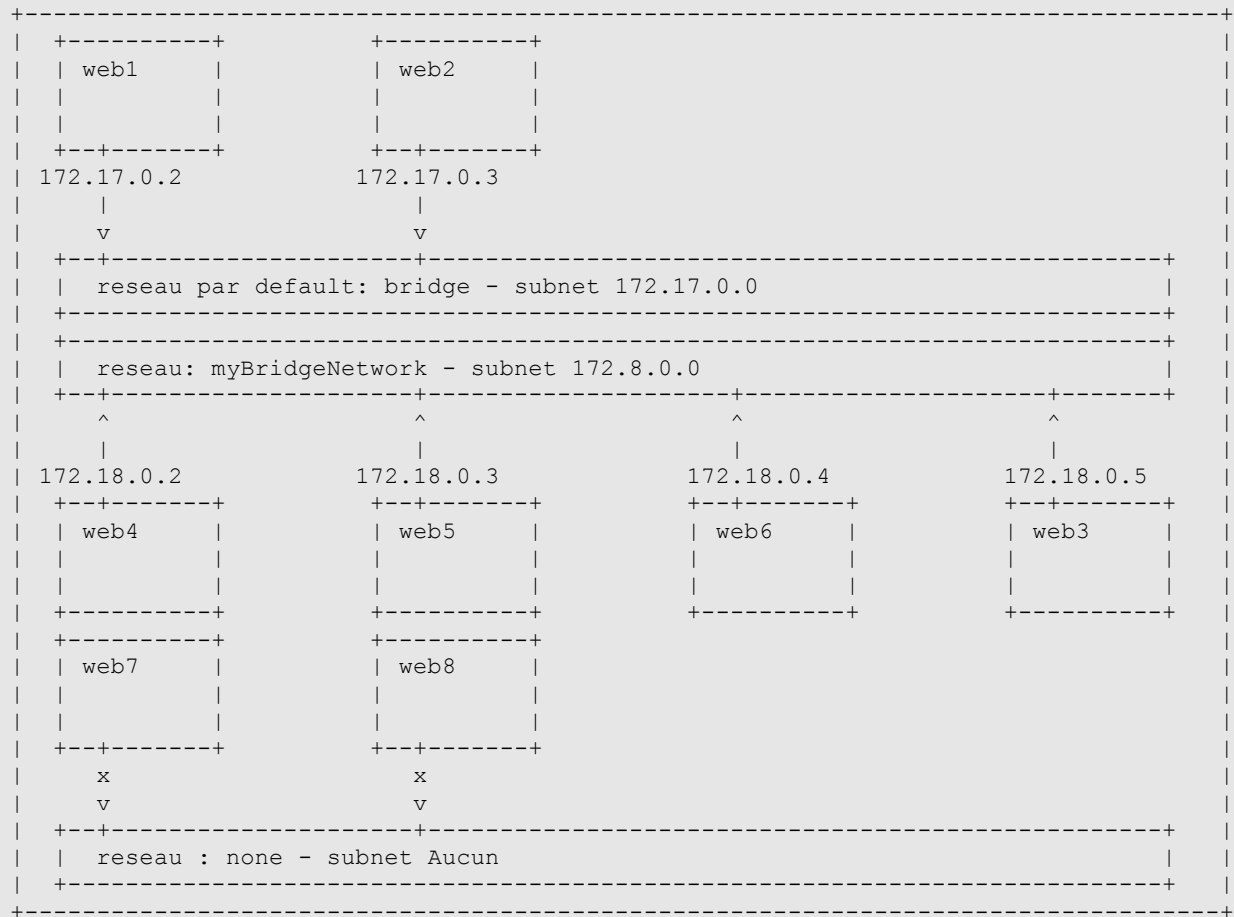
```
e1rl2p13% docker exec web3 ip addr show
1: lo: ...
    ...
    inet 127.0.0.1/8 scope host lo
    ...
28: eth1@if29: ...
    ...
    inet 172.18.0.5/16 brd 172.18.255.255 scope global eth1
    ...
```

[LABs] 39a) Network Basics: Connecter un container a 2 reseaux

***a) Voci d'abord mes differents reseaux et les Containers crees et attches a ces differents reseaux: ***

```
e1rl2p13% docker network inspect -f '{{ range $key, $value := .Containers }}{{ printf "%s:\t%s\n" $value.Name $value.IPv4Address }}{{ end }}' myBridgeNetwork
web4:      172.18.0.2/16
web5:      172.18.0.3/16
web6:      172.18.0.4/16
web3:      172.18.0.5/16
e1rl2p13% docker network inspect -f '{{ range $key, $value := .Containers }}{{ printf "%s:\t%s\n" $value.Name $value.IPv4Address }}{{ end }}' bridge
web1:      172.17.0.2/16
web2:      172.17.0.3/16
e1rl2p13% docker network inspect -f '{{ range $key, $value := .Containers }}{{ printf "%s:\t%s\n" $value.Name $value.IPv4Address }}{{ end }}' none
web7:
web8:
```

***b) Un petit schéma représentant les différents réseaux dans notre host: ***



[LABs] 39b) Network Basics: Connecter un container a 2 reseaux

*Connectons le Container `_web8_` aux 2 reseaux `_bridge_` et `_myBridgeNetwork_` *

*c) Quelle est l'adresse ip de `_web8_` qui n'appartient a aucun reseau (i.e. `_none_`) ? *

```
e1rl2p13% docker exec web8 ip addr show
1: lo: ...
...
inet 127.0.0.1/8 scope host lo
...
```

***d) Stop the container. Je pense qu'on peut le faire à chaud aussi: ***

```
e1rl2p13% docker container ls | egrep web8
c8a31bf7c1f3      streamweaver/dfdemo  "python run.py"      About an hour ago    Up 1 second
web8
e1rl2p13% docker container stop web8
web8
e1rl2p13% docker container ls | egrep web8
e1rl2p13%
```

***e) Deconnecter _web8_ du réseau _none_ : ***

```
e1rl2p13% docker network disconnect none web8
e1rl2p13%
```

***f) Connecter _web8_ au réseau _bridge_ _myBridgeNetwork_? ***

```
e1rl2p13% docker network connect myBridgeNetwork web8
e1rl2p13% docker network connect bridge web8
e1rl2p13% docker container start web8
```

***g) Examinons à nouveau nos différents réseaux: ***

```
e1rl2p13% docker network inspect -f '{{ range $key, $value := .Containers }}{{ printf "%s:\t%s\n" $value.Name $value.IPv4Address }}{{ end }}' myBridgeNetwork
web4:      172.18.0.2/16
web5:      172.18.0.3/16
web6:      172.18.0.4/16
web3:      172.18.0.5/16
web8:      172.18.0.6/16
```

```

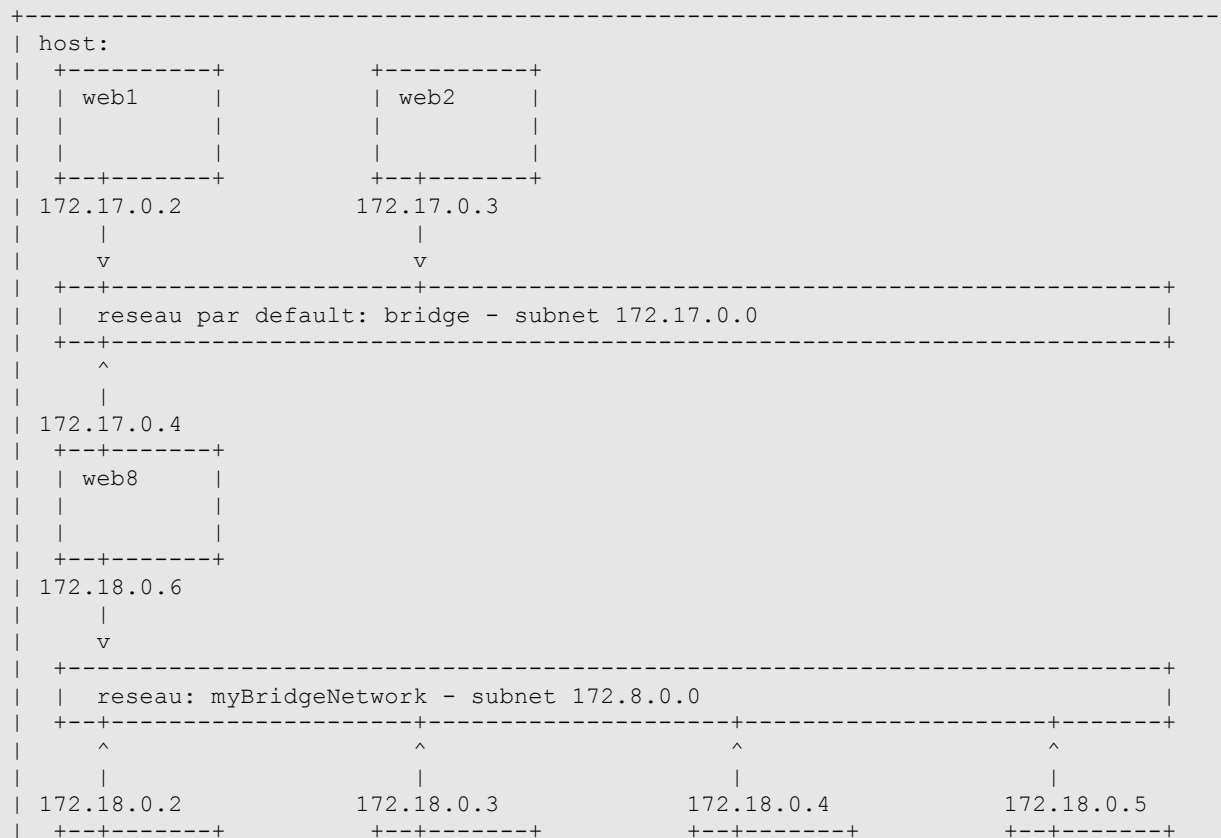
elr12p13% docker network inspect -f '{{ range $key, $value := .Containers }}{{ printf "%s:\t%s\n" $value.Name $value.IPv4Address }}{{ end }}' bridge
web1:      172.17.0.2/16
web2:      172.17.0.3/16
web8:      172.17.0.4/16

elr12p13% docker network inspect -f '{{ range $key, $value := .Containers }}{{ printf "%s:\t%s\n" $value.Name $value.IPv4Address }}{{ end }}' none
web7:

```

[LABs] 39c) Network Basics: Connecter un container a 2 reseaux

*h) Un petit schéma representant les differents reseaux dans notre host: *



web4	web5	web6	web3
web7			
reseau : none - subnet Aucun			

[LABs] 39d) Network Basics: Connecter un container a 2 reseaux

*i) On peut regarder de plus les interfaces reseaux de `_web8_` *

```
e1rl2p13% docker container exec web8 ip addr show
1: lo: ...
    inet 127.0.0.1/8 scope host lo
30: eth0@if31: ...
    inet 172.18.0.6/16 brd 172.18.255.255 scope global eth0
32: eth1@if33: ...
    inet 172.17.0.4/16 brd 172.17.255.255 scope global eth1
```

*j) On peut utiliser les interfaces de `_web8_` (i.e. `_eth0_` et `_eth1_`) et envoyer des `_ping's_` sur les autres containers: *

```
e1rl2p13% docker container exec web8 ping -c2 -I eth0 172.18.0.2
e1rl2p13% docker container exec web8 ping -c2 -I eth0 172.17.0.2
e1rl2p13% docker container exec web8 ping -c2 -I eth1 172.18.0.2
e1rl2p13% docker container exec web8 ping -c2 -I eth1 172.17.0.2
```

[Note]:

- Certains `_ping's_` vont fonctionner et d'autres pas. Aidez vous du schema pour comprendre !

[LABs] 39e) Network Basics: Faisons un peu de ménage avant d'explorer d autres types de networks

***Voici qqes commandes que j utilise pour supprimer les Containers demarres ou pas, les Images, les Networks ... : ***

***Par exple: pour les Containers: ***

```
e1rl2p13% docker container stop web1 web2 web3 web4 web5 web6 web7 web8
e1rl2p13% docker container ls -a
e1rl2p13% docker container prune --force
e1rl2p13% docker container ls -a
```

***Par exple: pour les Reseaux: ***

```
e1rl2p13% docker network prune --force
```

***Par exple: pour les Images: ***

```
e1rl2p13% docker image ls
e1rl2p13% docker image prune --force
e1rl2p13% docker image rm <id_image_1> <id_image_2> ...
```

***Par exple: pour le system en entier: ***

```
e1rl2p13% docker system prune --force
```

***Par exple: pour les VMs executant le *_docker engine_*: ***

```
docker-machine ls
docker-machine stop <nom de la machine>
docker-machine rm <nom de la machine>
```

***Astuce: pour se rappeler de la commande: ci-dessous, une petite photo pour comprendre la signification de prune: ***
<https://www.ma-belle-maison.fr/jardin/pourquoi-elaguer-regulierement-vos-arbres.html>

[LABs] 40) Network Basics: driver host

TBD

D'après la page ci-dessous, le driver `*_host_*` ne serait supporté que dans linux
<https://docs.docker.com/network/host/>

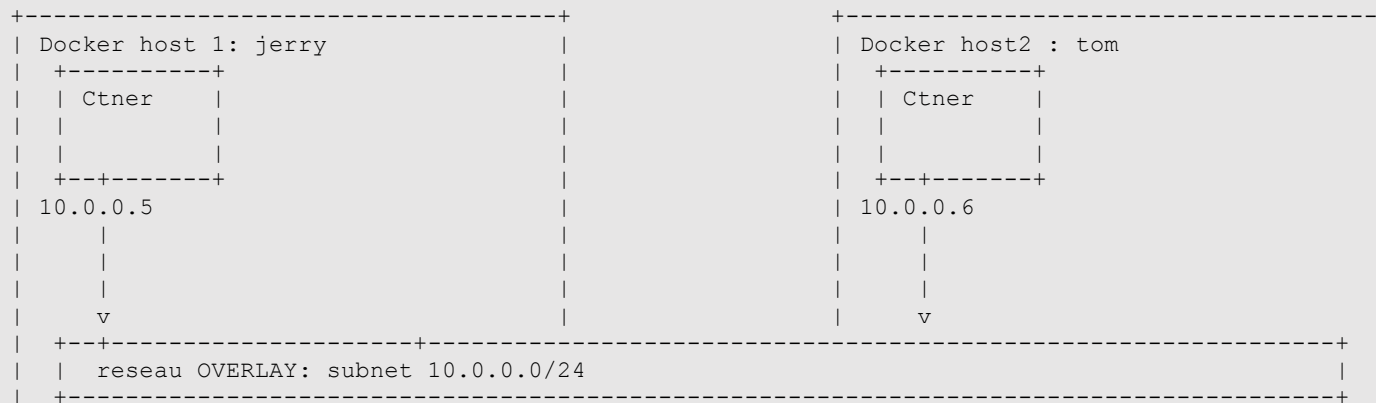
The host networking driver only works on Linux hosts, and is not supported on Docker for Mac, Docker for Windows, or Docker EE for Windows Server.

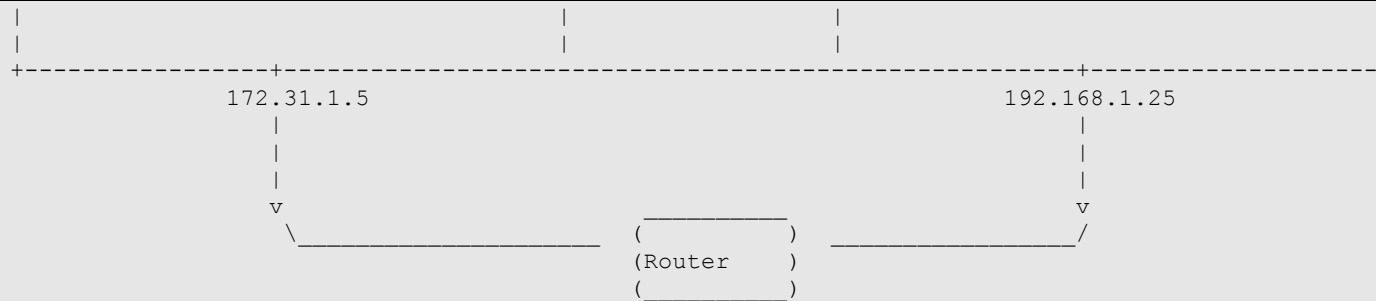
***[INFO] 41) Network Basics: driver overlay: ***

J'ai l'impression qu'il faut utiliser Swarm avec ce type de réseau

Voici ma comprehension de ce type de reseau: il s agit de creer un reseau reliant les Containers 'loges' dans differents hosts

Voici un exple vu sur youtube





Les reseaux 172.31.xx.xx et 192.168.x.xx ne peuvent communiquer que si il y a un routeur. Ceci n a rien avoir avec Docker et c est le principe de Inter-Net (= Inter-Reseaux = connecter des reseaux differents)

*[LABs] 42a) Network Basics: driver overlay: *

[Note]: Ne pas se preoccuper pour l instant de `_Swarm_`

*Essayons d implementer l exemple precedent, toutefois les adresses ip des machines (celles donnees lorsqu on fait `_docker-machine ip <id-vm>` ne seront pas modifiees pour etre des subnets differents) *

*Info: afin d eviter de reconfigurer les variables d environnement qui permettent au `_docker client_` de communiquer avec `_docker-engine_` *
Nous allons utiliser 2 onglets shell appeles `_tom-tab_` et `_jerry-tab_` et configurer les variables

*a) Creons les machines contenant les `_docker engines_` : *

```

tom-tab%elr12p13% docker-machine create -d virtualbox tom
tom-tab%elr12p13% eval $(docker-machine env tom)
tom-tab%elr12p13% docker-machine ls
  NAME      ACTIVE  DRIVER      STATE     URL                                     SWARM   DOCKER      ERRORS
  tom       *       virtualbox   Running   tcp://192.168.99.105:2376              -       v18.06.0-ce

jerry-tab%elr12p13% docker-machine create -d virtualbox jerry
jerry-tab%elr12p13% eval $(docker-machine env jerry)
jerry-tab%elr12p13% docker-machine ls
  NAME      ACTIVE  DRIVER      STATE     URL                                     SWARM   DOCKER      ERRORS
  jerry     *       virtualbox   Running   tcp://192.168.99.106:2376              -       v18.06.0-ce
  tom       -       virtualbox   Running   tcp://192.168.99.105:2376              -       v18.06.0-ce
  
```

***b) Creons un cluster *_swarm_* : ***

```
tom-tab%elr12pl3% docker swarm init --advertise-addr $(docker-machine ip tom)
Swarm initialized: current node (kmlci589mswr8x6ivf6r634ub) is now a manager.
To add a worker to this swarm, run the following command:
    docker swarm join --token SWMTKN-1-00w9h2522eqz7ltc5vv1lr5zzp0fdzawk289pcvfa3gu0tg1a0-2jsil9kvhrtj4pleo2462rxsm
192.168.99.105:2377
To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

jerry-tab%elr12pl3% docker swarm join --token SWMTKN-1-00w9h2522eqz7ltc5vv1lr5zzp0fdzawk289pcvfa3gu0tg1a0-2jsil9kvhrtj4pleo2462rxsm
192.168.99.105:2377
This node joined a swarm as a worker.
```

***c) Creons un network overlay: ***

```
tom-tab%elr12pl3% docker network create --driver overlay my_overlay_network
wds0sqjnicnu4brolzocs8geu

tom-tab%elr12pl3% docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
...
wds0sqjnicnu        my_overlay_network  overlay             swarm
...
```

***[LABs] 42b) Network Basics: driver overlay: ***

***d) Creons un *_service_* avec 2 Containers *_alpine_* en utilisant le reseau OVERLAY *_my_network_overlay_* : ***

Chaque Container va s executer sur chaque *_docker-engine_* (i.e. *_tom_* et *_jerry_*)

```
tom-tab%elr12pl3% docker service create --name my_service --network my_overlay_network --replicas 2 alpine sleep 1d
```

***e) Examinons sur chaque *_docker-engine_* les Containers qui y *_tournent_* ansi que leurs adresses ip : ***

```
tom-tab%elr12pl3% docker container ls
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
63f541d0e24e	alpine:latest	"sleep 1d"	About a minute ago	Up About a minute		
my_service.2.ayfkbt1vm3vlzxoewrqjkehdt						
tom-tab%elr12p13% docker inspect --format='{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}'					63f541d0e24e	
10.0.0.6						
jerry-tab%elr12p13% docker container ls						
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
a2e35543181e	alpine:latest	"sleep 1d"	8 minutes ago	Up 8 minutes		
my_service.1.hfq9e3ys96ob80ygcs5ctlii						
jerry-tab%elr12p13% docker inspect --format='{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}'					a2e35543181e	
10.0.0.5						

***f) Enfin qqes tests: tels que ping pour voir si ils peuvent communiquer : ***

```

tom-tab%elr12p13% docker container exec -it 63f541d0e24e hostname -i
10.0.0.6
tom-tab%elr12p13% docker container exec -it 63f541d0e24e ping -c2 10.0.0.5
64 bytes from 10.0.0.5: seq=0 ttl=64 time=0.556 ms
64 bytes from 10.0.0.5: seq=1 ttl=64 time=0.486 ms

jerry-tab%elr12p13% docker container exec -it a2e35543181e hostname -i
10.0.0.5
jerry-tab%elr12p13% docker container exec -it a2e35543181e ping -c2 10.0.0.6
64 bytes from 10.0.0.6: seq=0 ttl=64 time=0.553 ms
64 bytes from 10.0.0.6: seq=1 ttl=64 time=0.478 ms

```

***[LABs] 43) Network Basics: driver overlay - avec 2 hosts physiques differents ***

TBD

J ai essaye, mais cela n a pas fonctionne ici a 42, je supposerai que ce sont des problemes de firewall et n ai pas la main dessus

***Pour ceux qui veulent essayer rapidement, il est tjrs possible d utiliser les labs en ligne de docker, e.g: ***

<https://training.play-with-docker.com/ops-s1-swarm-intro/>.

[LABs] 44) Network Basics: Explorer DNS

Il y a visiblement un DNS qui est implemente en interne sur les nouvelles versions

TBD

*[LABs] 45a) OLD WAY - Network Basics: Connecter 2 Containers via --link *

***Pourquoi OLD WAY: ***

cf. <https://docs.docker.com/network/links/>

Warning: The --link flag is a legacy feature of Docker. It may eventually be removed. Unless you absolutely need to continue using it, we recommend that you use user-defined networks to facilitate communication between two containers instead of using --link. One feature that user-defined networks do not support that you can do with --link is sharing environmental variables between containers. However, you can use other mechanisms such as volumes to share environment variables between containers in a more controlled way.

***Le lien permet aux Containers de se découvrir et de se transmettre des données. ***

Quand on cree un lien, on cree un tuyau entre le Container Source et le Container Destinataire

***Syntaxe: ***

```
docker container run \  
    --name <name_of_the_container_to_run>  
    --link <name_of_the_container_to_link>:<ALIAS_name_of_the_container_to_link>
```

***a) Creons un container wordpress _relie_ a un container _mysql_ : ***

***info: *** _wordpress_ a besoin de _mysql_ pour fonctionner

```
+-----+  
| host:                                     |  
| +-----+                               +-----+ |  
| | my_mysql_container |                 | my_wordpress_container | | |
| |                   | +-----+         |                   | |  
| |                   | | Link           |                   | |  
| |                   | +-----+         |                   | |  
| |                   |                   |                   | |
```

```

| | | | |
| +---+-----+ | | | |
| IP: xx.xx.xx.xx Port: 3306 | IP: xx.xx.xx.xx Port: 4242 |
| | | | |
| | | | |
| v |
| 3306 | 4242 |
| | |
+-----+
IP de la machine docker_engine: docker ip <default> xx:xx:xx:xx
|
|
4242
|
v

```

*[LABs] 45b) OLD WAY - Network Basics: Connector 2 Containers via --link *

***b) Creons le container _my_mysql_container_ . On n expose pas le port de ce container, mais on peut le faire: ***

Remarquez que nous passons une variable d environnement a ce container

```

elrl2p13% docker container run --name my_mysql_container --env MYSQL_ROOT_PASSWORD=my_password --detach mysql
elrl2p13% docker container exec my_mysql_container hostname -i
172.17.0.2
elrl2p13% docker container exec my_mysql_container env | egrep MYSQL
MYSQL_ROOT_PASSWORD=my_password
MYSQL_MAJOR=8.0
MYSQL_VERSION=8.0.12-1debian9

```

***c) On peut creer une database _my_db_ par exemple: ***

```

elrl2p13% docker container exec -it my_mysql_container bash
root@10695fcbee4c:/# echo "create database my_db" | mysql -u root -p
Enter password:my_password
root@10695fcbee4c:/# echo "show databases" | mysql -u root -p
Enter password:my_password
...
my_db
...
root@10695fcbee4c: ^P^Q      <--- pour ne pas arreter le container. ^ = touch clavier CTRL

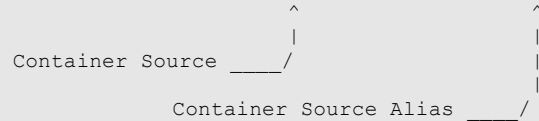
```



```
e1rl2p13%
```

***d) Creons un container `_my_wordpress_container_` et `_linkons_` le au container `_my_mysql_container_` :** *

```
e1rl2p13% docker container run --name my_wordpress_container --link my_mysql_container:mysql_alias -p 4242:80 -d wordpress
```



```
e1rl2p13% docker container ls
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
4db43eb97c8d	wordpress	"docker-entrypoint.s..."	38 seconds ago	Up 36 seconds	0.0.0.0:4242->80/tcp	
my_wordpress_container						
10695fcbec4c	mysql	"docker-entrypoint.s..."	35 minutes ago	Up 35 minutes	3306/tcp, 33060/tcp	
my_mysql_container						

***[LABs] 45c) OLD WAY - Network Basics: Connecter 2 Containers via `--link` ***

***e) En creant le lien, toutes les variables d environnement de `_my_mysql_container_` sont passees automatiquement au container `_my_wordpress_container_` :** *

```
e1rl2p13% docker container exec my_wordpress_container env | egrep -i my_sql_alias
MY_SQL_ALIAS_PORT=tcp://172.17.0.2:3306
MY_SQL_ALIAS_PORT_3306_TCP=tcp://172.17.0.2:3306
MY_SQL_ALIAS_PORT_3306_TCP_ADDR=172.17.0.2
MY_SQL_ALIAS_PORT_3306_TCP_PORT=3306
MY_SQL_ALIAS_PORT_3306_TCP_PROTO=tcp
....
MY_SQL_ALIAS_NAME=/my_wordpress_container/my_sql_alias
MY_SQL_ALIAS_ENV_MYSQL_ROOT_PASSWORD=my_password
....
```

***f) En creant le lien, on remarque egalement que l'assoication entre adresse ip et hostname de `_my_mysql_container_` sont transmises au container `_my_wordpress_container_` :** *

```
e1rl2p13% docker container exec my_wordpress_container cat /etc/hosts
127.0.0.1 localhost
```


***i) Finalement testons notre wordpress : ***

```
e1rl2p13% curl -L 192.168.99.101:4242
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
...
```

ou bien:

browser <http://192.168.99.101:4242>

Pour compléter votre installation, vous aurez besoin d'entrer ces informations:

- Nom de la base de données: my_db <--- nous l'avons créé dans `_my_mysql_container_`
- identifiant: root <--- username de `_my_mysql_container_`
- Mot de passe: my_password <--- celui de la database
- Adresse de la base de données: 172.17.0.2 <--- adresse ip de `_my_mysql_container_`

[LABs] 45e) OLD WAY - Network Basics: Connecter 2 Containers via --link ***j) Une fois complète les 1ers formulaires, on a notre wordpress avec sa database prêt à l'emploi: ***


```
cd8eca578189      jpetazzo/namer:master   "rackup --host 0.0.0..."   5 seconds ago      Up 4 seconds      0.0.0.0:32769->9292/tcp
determined_zhukovsky

elr12p13% docker volume ls
DRIVER          VOLUME NAME
elr12p13%

elr12p13% docker container inspect cd8eca578189 | jq '.[0].Mounts'
[]

elr12p13% docker container inspect cd8eca578189 | jq '.[0].Volumes'
null
elr12p13%

elr12p13% docker volume ls
DRIVER          VOLUME NAME
elr12p13%

elr12p13% docker container port cd8eca578189
9292/tcp -> 0.0.0.0:32769
elr12p13%
```

***[LABs] 46b) Volumes: [bind mount] Partager du code entre le host et le container ***

***b) Accéder à l'application via le container lancé, pour voir ce qu'elle fait: ***

Elle affiche des noms de compagnies avec une couleur bleue

```
elr12p13% docker-machine ip
192.168.99.101
elr12p13%
```

***via curl: ***

```
elr12p13% docker container port cd8eca578189
9292/tcp -> 0.0.0.0:32769

elr12p13% docker-machine ip
192.168.99.101

elr12p13% curl 192.168.99.101:32769
<html>
```

```

<style>
  h1, h2 {
    font-family: Georgia, Times New Roman, Times, serif;
    color: royalblue;
    margin: 0;
  }
</style>
<title>Company name generator</title>
<body>
  <h1>Batz-Rippin</h1>
  <h2>unleash integrated infomediaries</h2>
</body>
</html>

```

***[LABs] 46c) Volumes: [bind mount] Partager du code entre le host et le container ***

On va telecharger le code de cette application, puis
partager un repertoire local avec un nouveau container
***et enfin changer la couleur du texte *'a_chaud'* ***

***c) Telechargeons la meme application qui tourne sur le container teste precedemment: ***

```

elr12p13% git clone https://github.com/jpetazzo/namer
elr12p13% cd namer
elr12p13% ls
  Dockerfile          README.md             config.ru
  Gemfile              company_name_generator.rb  docker-compose.yml
elr12p13%

```

***d) Examinons le *_Dockerfile_*: ***

Lors de la creation du container le *./src* est copie dans le container

```

FROM ruby
MAINTAINER Education Team at Docker <education@docker.com>
COPY . /src
WORKDIR /src
RUN bundler install
CMD ["rackup", "--host", "0.0.0.0"]

```

EXPOSE 9292	<---- le port expose par cette appli. que je vais devoir mapper sur un port du host local
-------------	---

EXPOSE 9292	<---- le port expose par cette appli. que je vais devoir mapper sur un port du host local
-------------	---

***e) Maintenant on veut modifier le code de cette application, on peut soit : ***

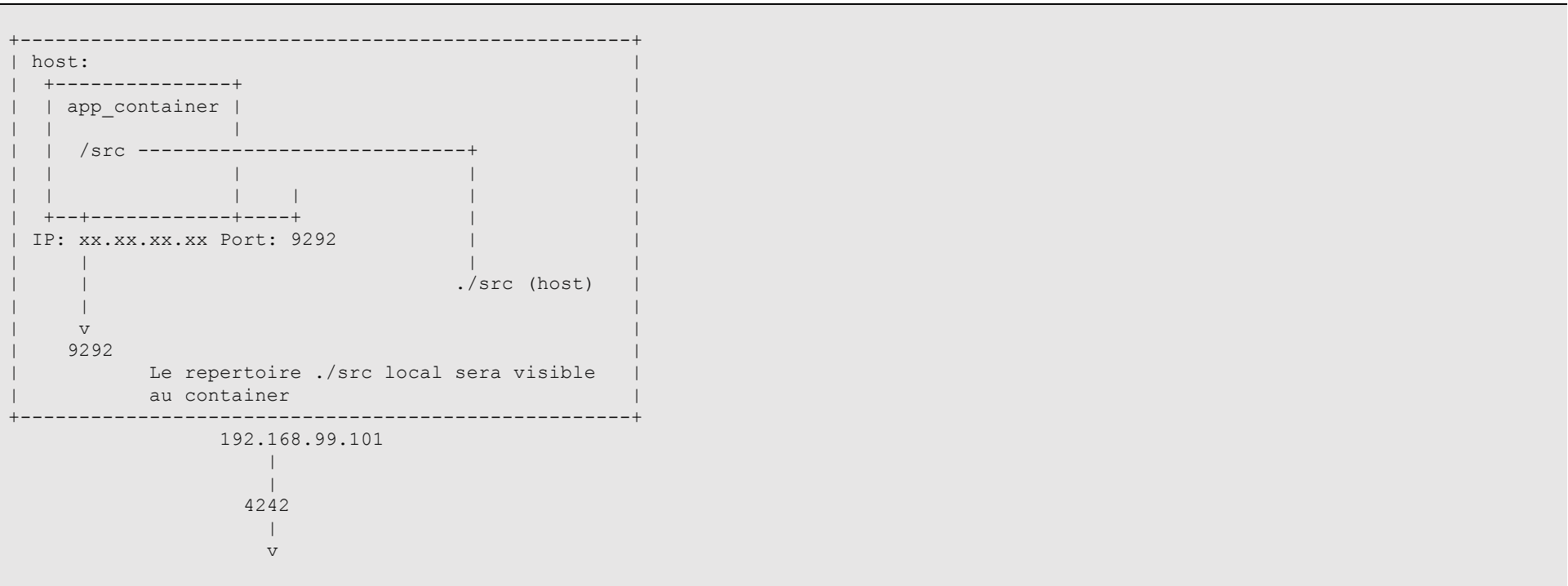
- Modifier le source en local (sur le host) et reconstruire le container ou bien

- ***Partager*** un repertoire local (host) avec celui du container. Cette methode nous evite de reconstruire le container a chaque fois que l'on fait

- **'Partager'** un repertoire local (host) avec celui du container. Cette methode nous evite de reconstruire le container a chaque fois que l'on fait une modification au code source

*[LABs] 46d) Volumes: [bind mount] Partager du code entre le host et le container *

***f) Examinons la 2eme solution: Demarrons un nouveau container en 'partgeant' un volume local (e.g. `./src`) avec le container afin d obtenir: ***



***Creons ce container: ***

```

elr12p13% docker container run --detach --publish 4242:9292 --volume $(pwd):/src:rw jpetazzo/namer:master
elr12p13% docker container ls

```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
>>> 0c77e5994aab	jpetazzo/namer:master	"rackup --host 0.0.0..."	6 seconds ago	Up 5 seconds	0.0.0.0:4242->9292/tcp	
cocky_booth <<< cd8eca578189	jpetazzo/namer:master	"rackup --host 0.0.0..."	About an hour ago	Up About an hour	0.0.0.0:32769->9292/tcp	

```

determined_zhukovsky
elr12p13%

```

***_--volume_ prend en arguments: ***

```

+-----+-----+-----+
| host-path | container-path | rw --> read/write (par default) |
| *ABSOLUTE PATH* | | ro --> read only |
+-----+-----+-----+
*Absolute path: car le path est relatif, cela fait autre chose !!! *

```

***[LABs] 46e) Volumes: [bind mount] Partager du code entre le host et le container ***

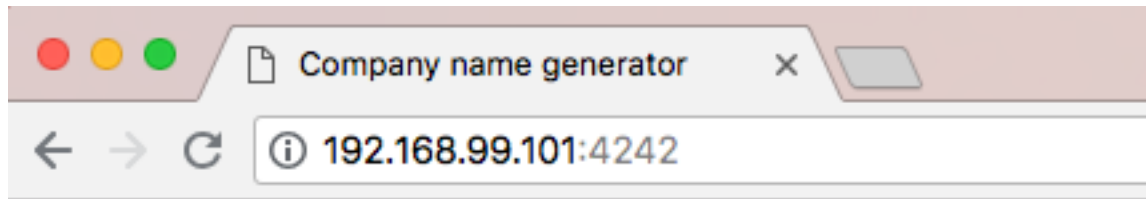
***g) Acceder a l application de ce container nouvellement cree: ***

```

elr12p13% curl 192.168.99.101:4242
<html>
...
    color: royalblue;
....

```

***ou bien: via browser: ***



Marks-Hayes

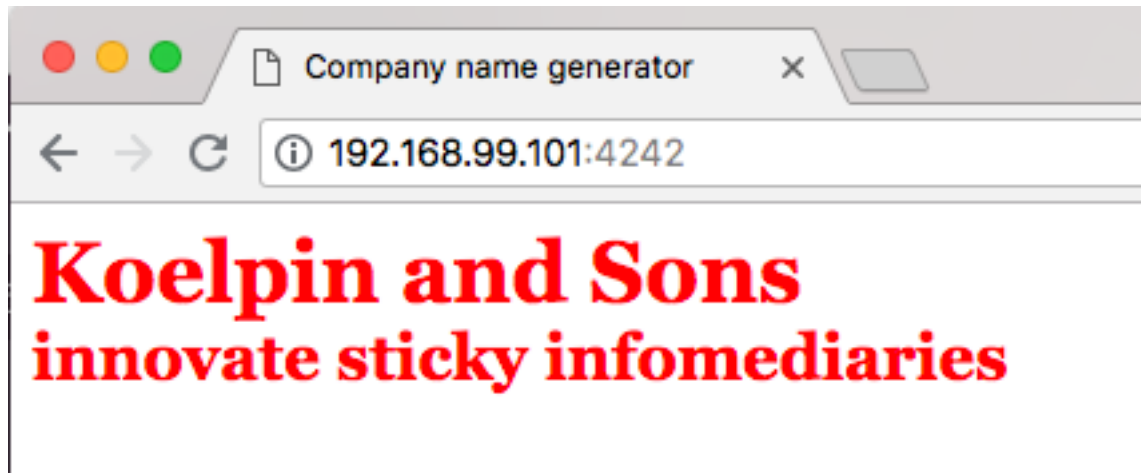
e-enable impactful schemas

***[LABs] 46f) Volumes: [bind mount] Partager du code entre le host et le container ***

h) Changer la couleur `'color'`: red du fichier `_company_name_generator.rb` (`'a chaud'`: le container tourne tjrs)

```
e1r12p13% egrep -i color company_name_generator.rb
          color: royalblue;
e1r12p13% vim company_name_generator.rb
e1r12p13% egrep -i color company_name_generator.rb
          color: red;
e1r12p13%
```

***i) Rafraichir la page du browser: ***



***[LABs] 46g) Volumes: [bind mount] Partager du code entre le host et le container ***

***j) Inspect le dernier container cree: ***

```
elr12p13% docker container ls
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                    NAMES
>>> 0c77e5994aab  jpetazzo/namer:master  "rackup --host 0.0.0..." 6 seconds ago  Up 5 seconds  0.0.0.0:4242->9292/tcp    cocky_booth <<<
cd8eca578189   jpetazzo/namer:master  "rackup --host 0.0.0..." About an hour ago  Up About an hour  0.0.0.0:32769->9292/tcp
determined_zhukovsky
elr12p13%
```

```
elr12p13% docker container inspect 0c77e5994aab | jq '.[0].Mounts'
[
  {
    "Type": "bind",
    "Source": "/Users/aaiche/wip/docker/tutorials/namer",
    "Destination": "/src",
    "Mode": "rw",
    "RW": true,
    "Propagation": "rprivate"
  }
] ....

elr12p13% docker container inspect 0c77e5994aab | jq '.[0].Mounts' | jq '.[0].Type'
"bind"
```

```
e1r12p13% docker container inspect 0c77e5994aab | jq '.[0].Volumes'
null
```

```
e1r12p13% docker volume ls
DRIVER          VOLUME NAME
e1r12p13%
```

***k) [INFO] Autre methode pour faire un *_bind mount_* On utilise l option *_--mount_* qui a l'avantage d etre plus lisible: ***

```
docker container run --detach --publish 4242:9292 --mount type=bind,source=$(pwd),destination=/src:rw jpetazzo/namer:master
```

[LABs] 47) Volumes: creation de volumes

Les Volumes sont des repertoires speciaux dans un container.

Le container est detaché du Volume. En réalité, les donnees du Volume sont stockees dans le host. Par conséquent elles sont Persistantes, meme s le container est supprime.

Le Volume peut etre partage entre les containers ou bien entre le container et le host

Le Volume peut etre attache a un container

Les volumes peuvent etre declarés :

- via l instruction ***_VOLUME_*** dans *_Dockerfile_*
- en CLI: utilisation du flag ***_--volume_*** ou bien ***_--mount_*** (instruction plus recente) avec *_docker container run_*

[LABs] 48) Volumes: creer un volume et le monter dans un container

***a) Creer un Volume Docker et le monter dans un container: ***

```
elr12p13% docker container run -it --volume myContainerVolumeName:/myFirstContainerVolumeMountpoint ubuntu bash
```

Volume a creer

Notez qu'on utilise un path relatif

Ou sera monte le volume dans le container

```
root@056782aece4b:/# exit
```

e1r12p13%

***b) un peu d inspection: ***

```
e1r12p13% docker container ls -a
```

CONTAINER ID

IMAGE

COMMAND

```
"bash"
```

CREATED

5 minutes ago

STATUS

Exited (127) 51 seconds ago

PORTS

NAMES

modest_brown

```
elr12p13% docker container inspect 056782aece4b | jq '[0].Mounts'
```

[

{

```
"Type": "volume",
```

```
"Name": "myContainerVolumeName",
```

```
"Source": "/mnt/sda1/var/lib/docker/volumes/myContainerVolumeName/_data",
```

```
"Destination": "/myFirstContainerVolumeMountpoint",
```

```
"Driver": "local",
```

```
"Mode": "z",
```

```
"RW": true,
```

```
"Propagation": ""
```

```
<--- Type = volume
```

```
<--- Nom du volume
```

<--- ???

```
<--- Point de montage dans le container
```

```
e1r12p13% docker container inspect 056782aece4b | jq '[0].Volumes'
```

```

null

```

```
e1r12p13% docker volume ls
```

DRIVER

VOLUME NAME

local

```
myContainerVolumeName
```

***[INFO]: Autre methode - utilisation de l option `--mount` qui est plus lisible: ***

```
docker container run -it --mount source=myContainerVolumeName,destination=/myContainerVolumeMountpoint ubuntu bash
```

***[LABs] 49) Volumes: Est ce que les donnees ajoutees dans un volume sont persistantes ***

***a) Creer un Volume Docker et le monter dans un container et Creer un fichier dans ce volume: ***

```
e1rl2p13% docker container run -it --rm --volume myContainerVolumeName:/myFirstContainerVolumeMountpoint ubuntu bash
                                     ^
                                     |
                                     \ permet de supprimer le container une fois qu il est 'exit.é

root@0c05e6103278:/# echo "Ceci est le contenu du fichier nouvellement cree a partir du container et ECRIT dans son volume attache"
> /myFirstContainerVolumeMountpoint/new_file.txt

root@0c05e6103278:/# cat /myFirstContainerVolumeMountpoint/new_file.txt
    Ceci est le contenu du fichier nouvellement cree a partir du container et ECRIT dans son volume attache

root@0c05e6103278:/# exit;

e1rl2p13%
```

***b) Verifions que le container est detruit, (grace a l option `--rm`): ***

```
e1rl2p13% docker container ls -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
e1rl2p13%						

***c) Est que le volume `_myContainerVolumeName_` existe tjrs? ***

```
e1rl2p13% docker volume ls
```

DRIVER	VOLUME NAME
local	myContainerVolumeName

e1rl2p13%

<--- le volume (cree avec le container cree et supprimé) existe tjrs

***f) Conclusion: ***

- Lors de la creation du volume, ces donnees restent persitantes et sont meme accessibles depuis la vm
- Le volume peut etre attache a un nouveau container
- On dit que les volumes sont DECOUPLÉS des containers

[LABs] 50) Volumes: *_volume-from_* - Partager des volumes entre containers

En realite les volumes sont les memes repertoires dans le docker engine

***a) Creer des Volumes Docker dans le container master et Creons 2 fichiers dans chaque volume : ***

```
tab1-elr12p13% docker container run -it --name master --volume ContainerVolumeName1:/files --volume ContainerVolumeName2:/logs ubuntu bash
...
root@82108f909aa6:/# echo "Contenu du fichier new_file_1.txt" > /files/new_file_1.txt

root@82108f909aa6:/# echo "Contenu du 2eme fichier new_file_2.txt" > /logs/new_file_2.txt

root@82108f909aa6:/# cat /files/new_file_1.txt
Contenu du fichier new_file_1.txt

root@82108f909aa6:/# cat /logs/new_file_2.txt
Contenu du 2eme fichier new_file_2.txt

root@82108f909aa6:/# exit

tab1-elr12p13% docker volume ls
DRIVER          VOLUME NAME
local           ContainerVolumeName1
local           ContainerVolumeName2
tab1-elr12p13%
```

docker volume ls

***c) Sur un autre terminal, creer un Second container slave et Partager des volumes: ***

```
tab2-elr12p13% docker container run -it --name slave1 --volumes-from master ubuntu bash

root@ad8ac9f1450f:/# cat /files/new_file_1.txt
Contenu du fichier new_file_1.txt

root@ad8ac9f1450f:/# cat /logs/new_file_2.txt
Contenu du 2eme fichier new_file_2.txt

root@ad8ac9f1450f:/# exit
tab2-elr12p13%
```

***[LABs] 51) Volumes: Creer un Volume sans Container ***

On peut creer un volume sans forcement creer un container

***a) Creer un Volume Docker: ***

```
e1r12p13% docker volume create --name=files
files
e1r12p13% docker volume create --name=logs
logs

e1r12p13% docker volume ls
DRIVER          VOLUME NAME
local           files
local           logs
e1r12p13%
```

==> Ces volumes ne sont attachees a aucun Container.

***b) Exple *fictif* d utilisation: ***

```
docker container run --detach --volume files:/var/www -v logs:/var/log webserver
docker container run --detach --volume files:/home/ftp ftpserver
docker container run --detach --volume logs:/var/log lunmberjack
```


***[LABs] 52) Volumes: Comment voir si un Volume est monté (utilisé) dans un Container ***

TBD

Je n'ai pas l'impression que ces informations soient contenues dans le volume, toutefois on peut tjrs faire ceci:

- Créons 2 volumes et attachons les à 2 Containers:

```
e1r12p13% docker container run --detach --name master --volume ContainerVolumeName1:/files --volume ContainerVolumeName2:/logs ubuntu sleep 1d
e1r12p13% docker container run --detach --name slave1 --volumes-from master ubuntu sleep 1d
```

- Un petit coup d'inspection:

```
e1r12p13% docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
91a2d4de1324        ubuntu             "sleep 1d"         2 seconds ago      Up 1 second                slave1
dc500b2a3cd2        ubuntu             "sleep 1d"         37 seconds ago     Up 36 seconds                master

e1r12p13% docker volume ls
DRIVER              VOLUME NAME
local               ContainerVolumeName1
local               ContainerVolumeName2

e1r12p13% docker container ls --filter volume=ContainerVolumeName1 --format "table {{.ID}}\t{{.Mounts}}"
CONTAINER ID        MOUNTS
91a2d4de1324        ContainerVolum...,ContainerVolum...
dc500b2a3cd2        ContainerVolum...,ContainerVolum...

e1r12p13% docker container ls --filter volume=ContainerVolumeName2 --format "table {{.ID}}\t{{.Mounts}}"
CONTAINER ID        MOUNTS
91a2d4de1324        ContainerVolum...,ContainerVolum...
dc500b2a3cd2        ContainerVolum...,ContainerVolum...
```

***[LABs] 53) Partager un UNIQUE fichier entre le Container et le host ***

TBD

*[LABs] 54) Creer un Volume avec *_Dockerfile_* *

***Voici un dockerfile qui utilise l instruction *_VOLUME_* : ***

```
Dockerfile:
  FROM alpine
  VOLUME ["/myMointPoint"]          <--- On precise UNIQUEMENT le point de montage
  CMD ["/bin/ls"]
```

***Construisons l image a partir de ce Dockerfile: ***

```
e1rl2p13% docker volume ls
  DRIVER          VOLUME NAME

e1rl2p13% docker image ls
  REPOSITORY    TAG          IMAGE ID          CREATED          SIZE
e1rl2p13% docker image build --tag my_image .
...
Step 1/3 : FROM alpine
...
Step 2/3 : VOLUME ["/myMointPoint"]
...

e1rl2p13% docker image ls
  REPOSITORY    TAG          IMAGE ID          CREATED          SIZE
  my_image      latest       e612b5090490     4 seconds ago    4.41MB
  alpine        latest       11cd0b38bc3c     6 weeks ago      4.41MB

e1rl2p13% docker container ls -a
  CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS          NAMES

e1rl2p13% docker volume ls
  DRIVER          VOLUME NAME
e1rl2p13%
```

***Creons un container a partir de l image cree *_my_image_* et Examinons: ***

```
e1rl2p13% docker container run my_image
e1rl2p13% docker container run my_image
    bin
    ...
    myMountPoint
    ...

e1rl2p13% docker volume ls
    DRIVER          VOLUME NAME
    local           13cdfca3fe3d85743cd4c36f5a067f67a38bda7cd55c44938c73fb3c2730ea3a
e1rl2p13%

e1rl2p13% docker container inspect ae6d56ca1321 | jq '.[0].Mounts'
[
  {
    "Type": "volume",
    "Name": "13cdfca3fe3d85743cd4c36f5a067f67a38bda7cd55c44938c73fb3c2730ea3a",
    "Source": "/mnt/sda1/var/lib/docker/volumes/13cdfca3fe3d85743cd4c36f5a067f67a38bda7cd55c44938c73fb3c2730ea3a/_data",
    "Destination": "/myMountPoint",
    ...
  }
]
```

[Note]:

Je n'ai pas l'impression que l'on peut donner un ***nom au volume*** lorsque il est créé via *_Dockerfile_*

*[LABs] 55) Créer une image via *_commit_* + modifier le Volume avant le *_commit_* *

*Créer un container à partir d'une image de base: *

```
e1rl2p13% docker container run -it --volume myContainerVolumeName:/myMountPoint ubuntu bash
...
```

*Ajouter du contenu au volume: *

```
root@6b1fcae5032c:/# echo "This is my new file content" > /myMountPoint/myNewFile.txt

root@6b1fcae5032c:/# cat /myMountPoint/myNewFile.txt
    This is my new file content
root@6b1fcae5032c:/# exit
```

***Examiner la difference entre le container et l image de base: ***

```
e1rl2p13% docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
6b1fcae5032c        ubuntu             "bash"             2 minutes ago      Exited (0) 10 seconds ago                  youthful_wozniak

e1rl2p13% docker container diff 6b1fcae5032c
A /myMountPoint
C /root
A /root/.bash_history
```

***Creer la nouvelle image a partir du container: ***

```
e1rl2p13% docker image ls
REPOSITORY          TAG                IMAGE ID            CREATED             SIZE
ubuntu              latest             735f80812f90       3 weeks ago        83.5MB

e1rl2p13% docker container commit 6b1fcae5032c
sha256:79763a7f7e18069179546cb2e7a61a281085a0eea671b8b141a516d7df714f2a    <-- id de la nlle image

e1rl2p13% docker image ls
REPOSITORY          TAG                IMAGE ID            CREATED             SIZE
<none>              <none>            79763a7f7e18       7 seconds ago      83.5MB    <-- Nouvelle image
ubuntu              latest             735f80812f90       3 weeks ago        83.5MB
```

***Creer un nouveau container a partir de cette nouvelle image et examiner le volume/point de montage : ***

```
e1rl2p13% docker container run 79763a7f7e18 ls
bin
...
myMountPoint    <-- point de montage est tjrs la
...

e1rl2p13% docker container run 79763a7f7e18 ls -a /myMountPoint
.  ..          <-- Mais pas de fichier !!!!
```

CONCLUSION:

Seul le repertoire (*_myMountPoint_*) a ete conserve, et le contenu du Volume cree n a pas ete conserve

***[LABs] 56) Modifier le Volume via l instruction `_RUN_` ou bien `_ADD_` de `_Dockerfile_` ***

TBD: je n ai pas reussi a midifier le volume dans le `_Dockerfile_` en tentant d utiliser les instructions suivantes:

```
Dockerfile:
FROM alpine
VOLUME ["/myMointPoint"]
ADD myHostFile.txt /myMointPoint
CMD ["/bin/sh"]
```

[Note]:

When you docker commit, the content of volumes is not brought into the resulting image.

- _If a RUN instruction in a Dockerfile changes the content of a volume, those changes are not recorded neither._*

[LABs] 57) OLD WAY: creer un Data Container < Engine 1.9

Un `_Data Container_` est un container cree pour le seul objectf de referencer un ou plusieurs volumes

On n a besoin que de l executer qu une seule fois. Le volume sera alors cree

```
e1rl2p13% docker container run -d --volume /app --name my_datastore busybox echo 'My Datastore'
...
```

Cette derniere commande va creer un volume data a l interieur du container `_my_datastore_`

```
e1rl2p13% docker volume ls
DRIVER          VOLUME NAME
local           db41ab4094cee60e1450a2e3e5bec85f18b05fafcba42a80eeafaed8acd34077

e1rl2p13% docker volume inspect db41ab4094cee60e1450a2e3e5bec85f18b05fafcba42a80eeafaed8acd34077
[
  {
    ...
    "Mountpoint": "/mnt/sda1/var/lib/docker/volumes/db41ab4094cee60e1450a2e3e5bec85f18b05fafcba42a80eeafaed8acd34077/_data",
```

```
    "Name": "db41ab4094cee60e1450a2e3e5bec85f18b05fafcba42a80eeafaed8acd34077",  
    ...  
  }  
]
```

Les *_Data Containers_* sont utilisés par d'autres containers grâce à *_--volumes-from_*

[Note]:

Maintenant (à partir de *_Docker Engine 1.9_*), on peut créer des volumes en :

```
docker volume create ...
```

***[LABs] 58) Envoyer un signal à un container ***

[references]

<https://github.com/buildkite/docker-signal-test>

[IMPORTANT]: Le processus recevant le signal doit avoir un *_PID_* égal à 1

***Voici un *_Dockerfile_* nous permettant de construire une image: ***

```
Dockerfile:  
FROM ubuntu  
COPY loop.sh /  
RUN chmod +x /loop.sh  
CMD ["/loop.sh"] <--- mode exec
```

***Et voici un script shell qui tournera tout le temps à moins qu'il ne reçoive un signal: ***

[NOTE]: pour voir les signaux disponibles *_man signal_*

```
loop.sh  
#!/usr/bin/env bash
```

```

trap "echo 'Received SIGHUP'; exit 1" SIGHUP
trap "echo 'Received SIGINT'; exit 2" SIGINT
trap "echo 'Received SIGTERM'; exit 15" SIGTERM
trap "echo 'Received SIGSTOP'; exit 17" SIGSTOP
trap "echo 'Received SIGKILL'; exit 42" SIGKILL
le gerer                                     <--- celui la ne sera jamais attrape par le script, c est le noyau qui va

while true; do
    echo "Waiting for a signal..."
    sleep 5
done

```

***Construisons notre image: ***

```
e1rlp12% docker image build --tag=my_image .
```

***Creeons le container c1 a partir de cette image: ***

```

tab1-e1rlp12% docker container run -t --name=c1 --hostname=c1 my_image
c1: Waiting for a signal...
c1: Waiting for a signal...

```

***Sur un autre terminal: verifions le PID du script du Container c1 par exple: ***

docker container exec c1 ps -aef ==> check for pid 1 c est lui qui va recevoir le signal

```

tab2-e1rlp12% docker container exec c1 ps -aef

```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	1	0	0	11:09	pts/0	00:00:00	bash /loop.sh
root	19	1	0	11:10	pts/0	00:00:00	sleep 5
root	20	0	1	11:10	?	00:00:00	ps -aef

<--- Il a bien le PID=1

***Sur un autre terminal, envoyons un signal SIGHUP a c1: ***

```
e1rlp12% docker kill -s=SIGHUP c1
```

*Quel est le code d exit du container: *

```
elr1p12% docker container inspect -f '{{.State.ExitCode}}' c1
1
==> le script retourne bien le code 1 tel que demande dans le script
```

*Creons 3 autres containers **_c2_**, **_c3_**, **_c4_** et envoyons respectivement le signal **_SIGINT_**, **_SIGTERM_** et **_SIGKILL_**, puis examinons le code retour: *

```
tab1-elr1p12% docker container run -t --name=c2 --hostname=c2 my_image
c2: Waiting for a signal...
tab2-elr1p12% docker kill -s=SIGINT c2
elr1p12% docker container inspect -f '{{.State.ExitCode}}' c2
2

tab1-elr1p12% docker container run -t --name=c3 --hostname=c3 my_image
c3: Waiting for a signal...
tab2-elr1p12% docker kill -s=SIGTERM c3
elr1p12% docker container inspect -f '{{.State.ExitCode}}' c3
15

tab1-elr1p12% docker container run -t --name=c4 --hostname=c4 my_image
c4: Waiting for a signal...
tab2-elr1p12% docker kill -s=SIGKILL c4
elr1p12% docker container inspect -f '{{.State.ExitCode}}' c4
137
<---- Le script ne renvoie pas 42 comme demande, car ce signal n est pas traite par le script, mais par le noyau
```

*Examinons tous les codes de sortie de ces containers: *

```
elr1p12% docker container ls -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
316f0a5cd7d7	my_image	"/loop.sh"	2 minutes ago	Exited (137) About a minute ago		c4
68db792798d7	my_image	"/loop.sh"	9 minutes ago	Exited (15) 8 minutes ago		c3
adf8270782ab	my_image	"/loop.sh"	10 minutes ago	Exited (2) 9 minutes ago		c2
24f131d104e4	my_image	"/loop.sh"	10 minutes ago	Exited (1) 10 minutes ago		c1

***[Note]: faites le calul $137 - 128 = 9$, c est bien le code de **_KILL_** ***

[LABs] 59a) Redémarrer un container

[references]

<https://dzone.com/articles/ensuring-containers-are-always-running-with-docker>

***Voici un *_Dockerfile_* nous permettant de construire une image: ***

```
Dockerfile:
FROM ubuntu
COPY crash.sh /
CMD ["/bin/bash", "/crash.sh"]
```

***Et voici un script shell qui s'arrêtera au bout de 30 secondes: ***

```
crash.sh:
#!/bin/bash
sleep 30
exit 1
```

***Construisons notre image: ***

```
e1rlp12% docker image build --tag=my_image .
```

***Créons le container *_c1_* à partir de cette image: ***

```
e1rlp12% docker run -d --name=c1 --hostname=c1 my_image
```

```
e1rlp12% docker container ls -a
CONTAINER ID    IMAGE    COMMAND                  CREATED          STATUS          PORTS          NAMES
4ac6587d5dd9    my_image  "/bin/bash /crash.sh"    13 seconds ago  Up 13 seconds             c1
```

Et au delà de 30 secondes,

```
e1rlp12% docker container ls -a
CONTAINER ID    IMAGE    COMMAND                  CREATED          STATUS          PORTS          NAMES
4ac6587d5dd9    my_image  "/bin/bash /crash.sh"    About a minute ago  Exited (1) 47 seconds ago             c1
```

```
le container s arrete avec le code de retour = 1
```

***Creeons le container `_c2_` avec une policy `--restart always_` : ***

```
elrlp12% docker run -d --name=c2 --hostname=c2 --restart always my_image
```

```
elrlp12% docker container ls -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
4d427cba400f	my_image	"/bin/bash /crash.sh"	20 seconds ago	Up 19 seconds		c2

Cette derniere commande nous montre qu il est demarre depuis 19 secondes, et au dela de 30 secondes,

```
elrlp12% docker container ls -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
4d427cba400f	my_image	"/bin/bash /crash.sh"	About a minute ago	Up 2 seconds		c2

```
le container s arrete et REDEMARRE
```

[LABs] 59b) Redémarrer un container

***Creeons le container `_c3_` avec une policy `--restart on-failure:3_` : ***

```
elrlp12% docker run -d --name=c3 --hostname=c3 --restart on-failure:3 my_image
```

```
elrlp12% docker container ls -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
ba52b8057bfc	my_image	"/bin/bash /crash.sh"	17 seconds ago	Up 16 seconds		c3

Cette derniere commande nous montre qu il est demarre depuis 16 secondes, et au dela de 30 secondes,

```
elrlp12% docker container ls -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
ba52b8057bfc	my_image	"/bin/bash /crash.sh"	43 seconds ago	Up 12 seconds		c3

```
le container s arrete et REDEMARRE.
```

Mais au dela de quelques minutes,

```
elrlp12% docker container ls -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
ba52b8057bfc	my_image	"/bin/bash /crash.sh"	2 minutes ago	Exited (1) 40 seconds ago		c3

```
le container est ARRETE
```

```
==> on-failure: 3 : on a demande de le redemarrer 3 fois
```

***Modifions notre script, reconstruisons notre image, et refaisons le test avec `_c3_` avec un container que nous appellerons `_c4_` : ***

```
crash.sh:
#!/bin/bash
sleep 30
exit 0          <--- modification
```

```
docker image build --tag=my_second_image .
```

***Creeons le container `_c4_` avec une policy `--restart on-failure:3` : ***

```
elrlp12% docker run -d --name=c4 --hostname=c4 --restart on-failure:3 my_second_image
```

```
elrlp12% docker container ls -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS   NAMES
f522ba3dedde   my_second_image  "/bin/bash /crash.sh"   23 seconds ago Up 21 seconds          c4
```

Cette derniere commande nous montre qu il est demarre depuis 16 secondes, et au dela de 30 secondes,

```
elrlp12% docker container ls -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS   NAMES
f522ba3dedde   my_second_image  "/bin/bash /crash.sh"   47 seconds ago Exited (0) 16 seconds ago          c4
```

le container s arrete et ne redemarre pas.

==> on-failure: est gere si le code retour est different de 0

[LABs] 60) Variable d environnement a passer au container

CLI:

```
elrlp12% docker container run --env VAR1=toto --env VAR2=titi alpine /usr/bin/env | egrep VAR
...
VAR1=toto
VAR2=titi
```

***CLI: en utilisant un fichier: ***

```
my_environment_file.txt:
    VAR3=tutu
    VAR4=tata
```

```
e1r1p12% docker container run --env-file my_environment_file.txt alpine /usr/bin/env | egrep VAR
    VAR3=tutu
    VAR4=tata
```

***Dockerfile:**

```
ENV WEBAPP_PORT 8080
```

docker-compose.yml

```
TBD
environment
```

[INFO] 61a) YAML: Yet Another Markup Language ou YAML Ain't Markup Language

- Serialiser les données dans un fichier
- Similaire à Json
- file extension: ***.yml**, ou ***.yaml**

[references]:

Validator yaml : <http://www.yamllint.com/>

Yaml/Json Converter: <http://convertjson.com/yaml-to-json.htm>

***Data Types:**

- Numbers: integer, octal, hexadecimal, float, exponential number, infinity
- String: use of double quotes, single quotes, or none

- Boolean: true, false, yes, no
- Array: ordered list of 0 or more values, use of -
- Object: unordered collection of key/value pairs
- Null: empty value
- Dates

Syntax Rules:

- Faire attention au nombre d espaces
- Pas de tabulation
- Faire attention a l indentation

[INFO] 61b) YAML: Yet Another Markup Language ou YAML Ain't Markup Language

***Comparaison entre Yaml et Json: qqes exemples: ***

Type	Yaml	Json
Object Key Value Pairs	Fruit: Apple Vegetable: Carrot Liquid: Water Meat: Chicken	{ "Fruit": "Apple", "Vegetable": "Carrot", "Liquid": "Water", "Meat": "Chicken" }
Object with 2 key/Value pairs Each value is an array	Fruits: - Apple - Orange - Banana Vegetables: - Carrot - Cauliflower - Tomato	{ "Fruits": ["Apple", "Orange", "Banana"], "Vegetables": ["Carrot", "Cauliflower", "Tomato"] }
Nested objects	Banana: Calories: 105 Fat: 0.4 g Carbs: 27 g Grapes:	{ "Banana": { "Calories": 105, "Fat": "0.4 g", "Carbs": "27 g" }, "Grapes": { "Calories": 105, "Fat": "0.4 g", "Carbs": "27 g" } }

Calories: 62	"Grapes": {
Fat: 0.3 g	"Calories": 62,
Carbs: 16 g	"Fat": "0.3 g",
	"Carbs": "16 g"
	}
	}

[INFO] 62) *_docker-compose_*: Utiliser un fichier Yaml pour entrer tous les parametres du Container et Lancer plusieurs Containers

Dockerfile permet de creer un seul Container, il y a un outil qui permet d en creer plusieurs

****_docker-compose_*: ***

- Anciennement appelé **fig**.
- C est un outil externe
- Est optionnel (***n est pas necessaire pour utiliser Docker***)

****_docker-compose_* permet: ***

- Utiliser un fichier de parametres pour creer un Container car le faire avec *_docker run_* peut etre fastidieux
- Creer une pile (stack) de Containers
- Definir les options, les variables d environnement, les volumes, les reseaux, ou bien anciennement les *_links_*

Comment l'utiliser:

- Décrire l ensemble des containers (stack) dans le fichier ****_docker-compose.yml_* ***
- Verifier la validite du fichier yaml *_docker-compose config_*
- Executer *_docker-compose up_*
- Compose va automatiquement ***Telecharger*** les Images, ***Constuire*** les Containers, et les ***Demarrer***
- Les Containers peuvent etre executes en background ou foreground

Chaque section dans *_docker-compose.yml_* doit contenir:

- *_build_* : indique un path contenant *_Dockerfile_* qui permet de construire l image
- *_image_* : indique le nom d une image

Autres parametres, ils sont optionnels:

- `_command_` : similaire a `_CMD_` de dockerfile
- `_ports_` : similaire a `--publish_` (pour mapper les ports entre le host et le Container)
- `_volumes_` : similaire a `--volume_`
- `_links_` : similaire a `--link_` (Cette methode qui connecte des Containers entre eux est/ou va etre ***obsolete***)

...

Quelques commands:

- Verifier que `_docker-compose_` est installe:

```
docker-compose --version
```

- Verifier le fichier yml

```
docker-compose config
```

- Construire les Containers

```
docker-compose build
```

- Lancer les Containers, par exemple en background

```
docker-compose up -d
```

- Status des Containers: a la place d utiliser *docker ps*, preferer utiliser ***docker-compose ps*** (car cela ne montre que le stack lance)

```
docker-compose ps
```

- Arreter le stack

```
docker-compose kill
```

- Supprimer le stack

```
docker-compose rm
```

[Note]:

Les commandes `_docker-compose_` doivent etre executees dans le repertoire contenant le fichier `_docker-compose.yml_`

[INFO] 63) `_docker-compose_`: Version 1

Pas d'instruction `*_version_*` dans `*_docker-compose.yml_*`

Exemple:

```
docker-compose.yml:

  www:                                <-- debut de section: www
    build: www                        <-- ce repertoire contient Dockerfile. Le nom de l image constuite est:
                                     <repertoire><nom de section>_<??>

    ports:
      - 4242:5000                    <-- www expose le port 4242
    links:
      - redis                        <-- connection avec le Container redis
    user: nobody
    command: python counter.py
    volumes:
      - ./www:/src
  redis:                              <-- debut de section: Image redis
    image: redis                     <-- on utilise l image redis, et pas de modification
```

Particularites version1:

- Chaque section ne doit contenir que `_build_` ou bien `_image_`

[INFO] 64) *_docker-compose_*: Version 2

Le format de *_docker-compose.yml_* ***est different:***

```
version: '2'
+-----+
|           Services           |
+-----+
| Networks (optionnel)        |
+-----+
| Volumes (optionnel)         |
+-----+
```

***Particularites version2: ***

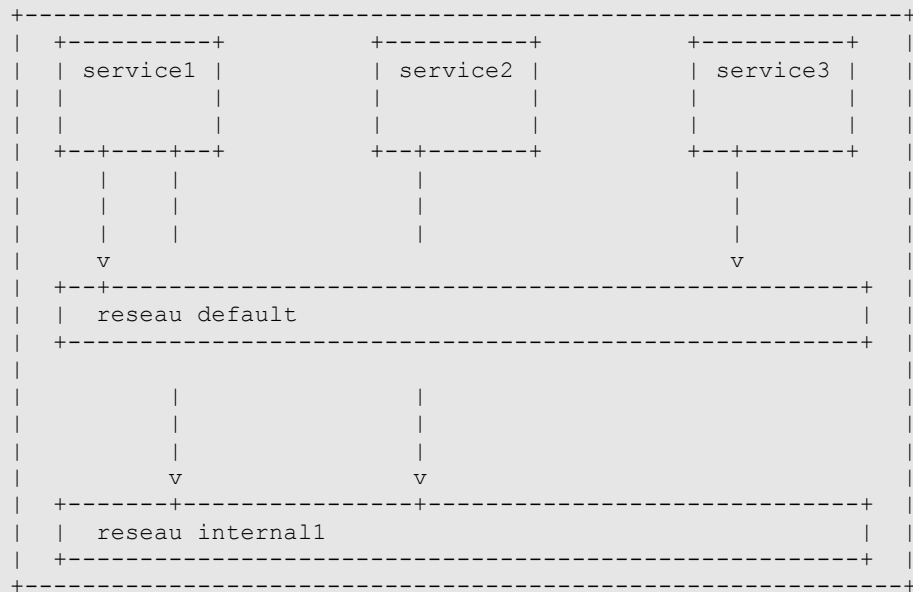
- 3 differentes parties
- La description d un service (une image) peut maintenant contenir a la fois les instructions:
 build et _image_.
 Si c est le cas, *_image_* specifiera le nom de l image
- ***Creation d un network par default qui reliera les differents Containers***

Exemple:

```
version: '2'
services:
  web:
    build:
      context: .           <-- ou se trouve dockerfile
      dockerfile: dockerfile <-- quel fichier dockerfile vais-je utiliser
      image: myDemoWeb     <-- nouveau nom
  mongo:
    image: mongo
```

[LABs] 65a) *_docker-compose_*: Version 2 - Exemple - Networks

Creation de 3 services (Containers) et 2 reseaux:



Voici notre `_docker-compose.yml` :

```
version: '2'

services:
  service1:
    image: ubuntu
    command: sleep 3600
    hostname: service1
    networks:
      - internal1
      - default
    <--- service1: 1er Container
    <--- il s agit d une image ubuntu
    <--- CMD: le container va executer cette commande lors de son instantiation
    <--- hostname du Container
    <--- Ce container aura 2 interfaces reseaux, chacune reliee au reseau ci-dessous
    <--- On a dit (ci-dessus) que la version 2, cree automatiquement un reseau. C est celui-la

  service2:
    image: ubuntu
    hostname: service2
    command: sleep 3600
    networks:
      - internal1
```

```
service3:
  image: ubuntu
  hostname: service3
  command: sleep 3600
  networks:
    - default
networks:
  internall: <--- Creatin d un reseau supplementaire appele internall
    driver: bridge
```

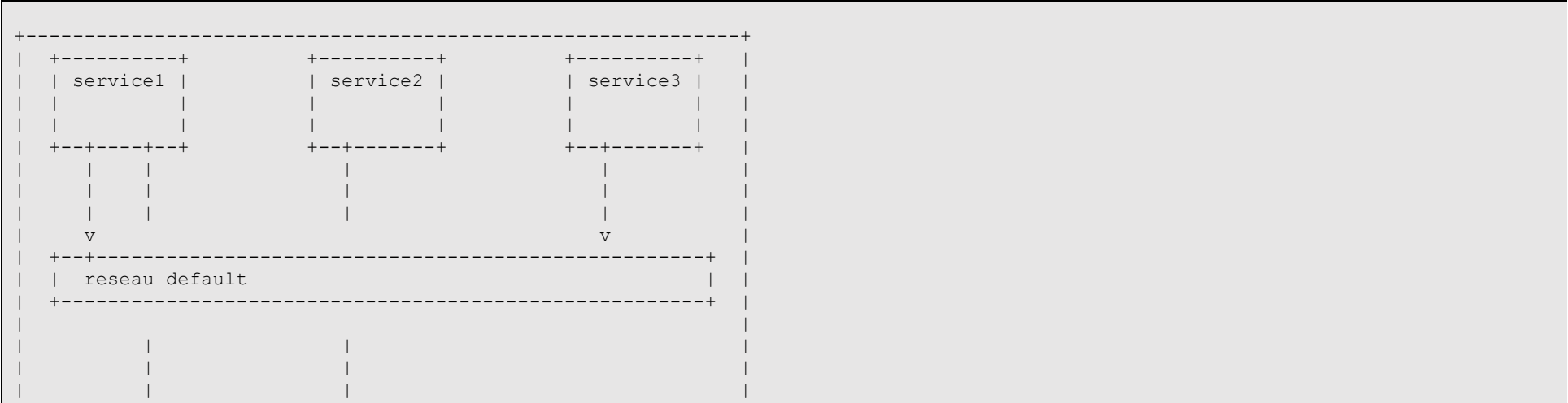
docker-compose va creer des reseaux. Avant de creer ces services, examinons les differents reseaux:

```
e1rlp12% docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
30018d556267	bridge	bridge	local
d1c4e6fa4f40	host	host	local
5c02f6126e40	none	null	local

*[LABs] 65a) *_docker-compose_*: Version 2 - Exemple - Networks*

Creation de 3 services (Containers) et 2 reseaux:



```

|           v           v           |
| +-----+-----+-----+-----+ |
| |   reseau internal1   |         | |
| +-----+-----+-----+-----+ |
+-----+-----+-----+-----+

```

***Voici notre `_docker-compose.yml` :**

```

version: '2'

services:

  service1:
    image: ubuntu
    command: sleep 3600
    hostname: service1
    networks:
      - internal1
      - default
    <--- service1: 1er Container
    <--- il s agit d une image ubuntu
    <--- CMD: le container va executer cette commande lors de son instantiation
    <--- hostname du Container
    <--- Ce container aura 2 interfaces resaux, chacune reliee au reseau ci-dessous
    <--- On a dit (ci-dessus) que la version 2, cree automatiquement un reseau. C est celui-la

  service2:
    image: ubuntu
    hostname: service2
    command: sleep 3600
    networks:
      - internal1

  service3:
    image: ubuntu
    hostname: service3
    command: sleep 3600
    networks:
      - default

networks:
  internal1:
    driver: bridge
    <--- Creatin d un reseau supplementaire appele internal1

```

***`_docker-compose` va creer des reseaux. Avant de creer ces services, examinons les differents reseaux:**

```

elrlp12% docker network ls

```

NETWORK ID	NAME	DRIVER	SCOPE
30018d556267	bridge	bridge	local
d1c4e6fa4f40	host	host	local
5c02f6126e40	none	null	local

[LABs] 65b) *_docker-compose_*: Version 2 - Exemple - Networks

***Build et Run: ***

***a) Verifions la validite de *_docker-compose.yml_* : ***

```
e1rlp12% docker-compose config
networks:
  internall:
    driver: bridge
services:
  service1:
    command: sleep 3600
    hostname: service1
...
version: '2.0'
```

***b) Y a t il des containers en cours d execution ? : ***

```
e1rlp12% docker-compose ps
      Name      Command      State      Ports
-----
e1rlp12%
```

***c) Construisons nos Containers : ***

```
e1rlp12% docker-compose build
service3 uses an image, skipping
service2 uses an image, skipping
service1 uses an image, skipping
e1rlp12%
```

```
e1rlp12% docker-compose images
      Container      Repository      Tag      Image Id      Size
```

```
-----
elrlp12%

==> il n y a aucune image de construite. C est normal, il n y a pas d instruction build dans docker-compose.yml
```

***d) Lancons nos Containers en background: ***

```
elrlp12% docker-compose ps
  Name          Command          State          Ports
  -----
elrlp12%
```

```
elrlp12% docker-compose up -d
  Creating network "compose2_default" with the default driver          <-- creation des reseaux
  Creating network "compose2_internal1" with driver "bridge"
  ....
  Pulling service3 (ubuntu:)...
  ...
  Creating compose2_service2_1 ... done          <-- creation des services
  Creating compose2_service1_1 ... done
  Creating compose2_service3_1 ... done
elrlp12%
```

***e) Examinons maintenant quels sont les images creees et les containers en cours d execution: ***

```
elrlp12% pwd
/Users/aaiche/wip/docker/tutorials/compose2
```

```
elrlp12% docker-compose images
  Container          Repository          Tag          Image Id          Size
  -----
  compose2_service1_1  ubuntu             latest       735f80812f90      79.6 MB
  compose2_service2_1  ubuntu             latest       735f80812f90      79.6 MB
  compose2_service3_1  ubuntu             latest       735f80812f90      79.6 MB
  ^
  |
  \_____ c est le nom du repertoire contenant docker-compose.yml. Desole je n ai pas bien nomme mon repertoire de test, cela
  peut etre confusionnant!!!
```

```
e1rlp12% docker-compose ps
```

Name	Command	State	Ports
compose2_service1_1	sleep 3600	Up	
compose2_service2_1	sleep 3600	Up	
compose2_service3_1	sleep 3600	Up	

***f) Essayer d executer les 2 dernieres commandes dans un autre repertoire: ***

Que remarquez vous ?

[LABs] 65c) docker-compose: Version 2 - Exemple - Networks

***g) Examinons quels sont les reseaux crees: ***

```
e1rlp12% docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
30018d556267	bridge	bridge	local
>>> 0074d86ed8f3	compose2_default	bridge	local <<<
>>> f71e49a53246	compose2_internal1	bridge	local <<<
d1c4e6fa4f40	host	host	local
5c02f6126e40	none	null	local

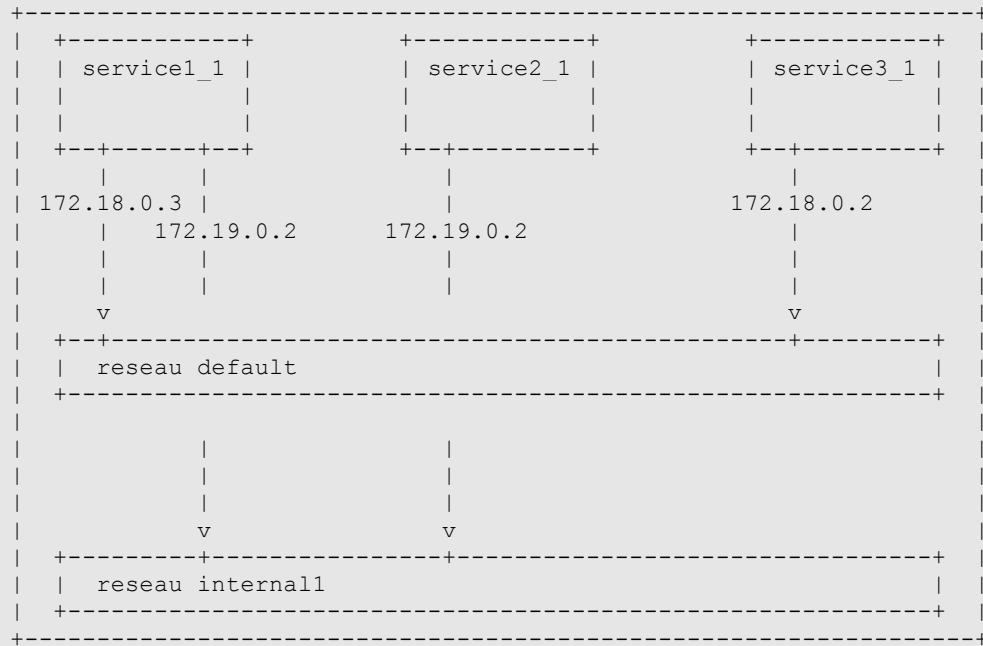
```
e1rlp12%
```

***h) Un petit coup d inspection: ***

```
e1rlp12% docker network inspect -f '{{ range $key, $value := .Containers }} {{ printf "%s:\t%s\n" $value.Name $value.IPv4Address }}{{ end }}' compose2_default
compose2_service3_1: 172.18.0.2/16
compose2_service1_1: 172.18.0.3/16

e1rlp12% docker network inspect -f '{{ range $key, $value := .Containers }} {{ printf "%s:\t%s\n" $value.Name $value.IPv4Address }}{{ end }}' compose2_internal1
compose2_service1_1: 172.19.0.3/16
compose2_service2_1: 172.19.0.2/16
```

***i) Complétons notre schema ci-dessus: ***



[LABs] 65d) _docker-compose_: Version 2 - Exemple - Networks

***j) Un petit coup de test avec _ping_ ***

[Note]: Nous utilisons les _hostnames_ des services. Pour s'en convaincre il faudrait voir les entrées DNS créées

***vérifier que _service1_1_ peut ping'uer _service2_1_ et _service3_1_: ***

```
elrlp12% docker container exec -it compose2_service1_1 apt-get update
elrlp12% docker container exec -it compose2_service1_1 apt-get install -y iputils-ping
```

```
elrlp12% docker container exec compose2_service1_1 ping -c2 service2
 64 bytes from compose2_service2_1.compose2_internal1 (172.19.0.2): icmp_seq=1 ttl=64 time=0.066 ms
 64 bytes from compose2_service2_1.compose2_internal1 (172.19.0.2): icmp_seq=2 ttl=64 time=0.064 ms
```



```
elrlp12% docker container exec compose2_service1_1 ping -c2 service3
 64 bytes from compose2_service3_1.compose2_default (172.18.0.2): icmp_seq=1 ttl=64 time=0.063 ms
 64 bytes from compose2_service3_1.compose2_default (172.18.0.2): icmp_seq=2 ttl=64 time=0.066 ms
```

***verifier que _service2_1 NE peut ping'uer QUE _service1_1 ***

```
elrlp12% docker container exec -it compose2_service2_1 apt-get update
elrlp12% docker container exec -it compose2_service2_1 apt-get install -y iputils-ping
```

```
elrlp12% docker container exec compose2_service2_1 ping -c2 service1
 64 bytes from compose2_service1_1.compose2_internal1 (172.19.0.3): icmp_seq=1 ttl=64 time=0.044 ms
 64 bytes from compose2_service1_1.compose2_internal1 (172.19.0.3): icmp_seq=2 ttl=64 time=0.068 ms
```

```
elrlp12% docker container exec compose2_service2_1 ping -c2 service3
ping: service3: Name or service not known
elrlp12% docker container exec compose2_service2_1 ping -c2 172.18.0.2
...
 2 packets transmitted, 0 received, 100% packet loss, time 1000ms
```

[LABs] 66a) _docker-compose_: Version 2 - Exemple - Volumes

***Partager un volume entre 2 Containers: voci ce que l on veut obtenir: ***

```
+-----+
| +-----+ | +-----+ | | | |
| | service1 | | service2 | |
| |         | |         | |
| |         | |         | |
| +---+-----+ | +---+-----+ |
| |         | |         | |
| |         | |         | |
| |         | |         | |
```


[LABs] 66b) *_docker-compose_*: Version 2 - Exemple - Volumes

Build et Run

***a) Verifions la validite de *_docker-compose.yml_* : ***

```
e1rlp12% docker-compose config
services:
  service1:
    command: sleep 3600
...
volumes:
  data:
    driver: local
```

***b) Construisons nos Containers: ***

```
e1rlp12% docker-compose images
Container    Repository    Tag    Image Id    Size
-----
```

```
e1rlp12% docker-compose build
service2 uses an image, skipping
service1 uses an image, skipping
```

```
e1rlp12% docker-compose images
Container    Repository    Tag    Image Id    Size
-----
e1rlp12%
```

***c) Lancons nos Containers en background: ***

```
e1rlp12% docker-compose up -d
Creating network "compose3_default" with the default driver    <--- creation du reseau par default
```

```

    Creating volume "compose3_data" with local driver
'volume' de docker-compose.yml                                <--- creation du volume correspondant a 1 instruction

    Pulling service2 (ubuntu:)...
    ....
    Creating compose3_service2_1 ... done                       <--- creation des 2 services decrits dans docker-compose.yml
    Creating compose3_service1_1 ... done
e1rlp12%

```

```

e1rlp12% docker-compose ps
      Name                Command             State      Ports
-----
compose3_service1_1      sleep 3600          Up
compose3_service2_1      sleep 3600          Up

```

***d) Examinons les Volumes crees: ***

```

e1rlp12% docker volume ls
DRIVER          VOLUME NAME
local          compose3_data    <--- volume cree

```

e) Ou est monte ce volume dans nos 2 Constainers

```

e1rlp12% docker container exec compose3_service1_1 df -h | egrep MountPoint
/dev/sda1          18G  135M   17G   1% /myService11111MountPoint

e1rlp12% docker container exec compose3_service2_1 df -h | egrep MountPoint
/dev/sda1          18G  135M   17G   1% /myService22222MountPoint

```

***f) Creons un nouveau fichier a partir du Container _service1_ : ***

```

e1rlp12% docker container exec compose3_service1_1 sh -c "echo 'Coucou je suis service1 et cree un nouveau file dans vol.' >
/myService11111MountPoint/service1_newfile.txt"

e1rlp12% docker container exec compose3_service1_1 ls /myService11111MountPoint/
service1_newfile.txt

e1rlp12% docker container exec compose3_service1_1 cat /myService11111MountPoint/service1_newfile.txt
Coucou je suis service1 et cree un nouveau file dans vol.

```

***g) Regardons le contenu de ce fichier a partir du Container `_service2_` :**

```
elrlp12% docker container exec compose3_service2_1 sh -c "ls /myService2222MountPoint/"
service1_newfile.txt

elrlp12% docker container exec compose3_service2_1 sh -c "cat /myService2222MountPoint/service1_newfile.txt"
Coucou je suis service1 et cree un nouveau file dans vol.
```

[INFO] 67) `_docker-compose.YML_` : version3

[Note]: Ce n est pas tres clair pour moi, parce que on parle de `_docker-compose v3_` et j ai l impression que certaines fonctionnalites de la version 3 sont ignorees avec l utilitaire `_docker-compose_`

A partir de la version Docker 1.13, de nouvelles fonctionnalites ont ete introduites, notamment:

- on peut utiliser un seul fichier YML afin de deployer sur un docker swarm (au lieu de le faire en CLI commande apres commande)
- on peut deployer Plusieurs Sevices, sur plusieurs docker-engine (cluster swarm)

• ***Voici la commande qui resume ces fonctionnalites: ***

```
docker stack deploy --compose-file my_docker_stack_file.yml
```

***Un petit schema qui nous permettra de se rappeler de ces differences: ***

+-----+		+-----+
docker client:		
- built-in docker-compose.yml v3		
parser	----->	Docker Swarm
- reads 'deploy' section to deploy a		
service on swarm cluster		

+-----+

+-----+

```
+-----+
| docker-compose:          |
| |                         |
| - has also built-in docker-compose.yml |
|   v3 parser              |
| - IGNORES 'deploy' section |
| - DOES NOT deploy on swarm cluster |
| - works against a single Docker host |
|   only                   |
+-----+
```

----->

```
+-----+
|                               |
|                               |
|   Single Docker              |
|                               |
|                               |
+-----+
```

***[INFO] 68) *_docker-compose_* : comparer les 3 versions de *_docker-compose_* : ***

[reference] :

<https://sreeninet.wordpress.com/2017/03/28/comparing-docker-compose-versions/>

N.B.: je ne ai pas lu cet article

***[INFO] 69) *Dockerfile* - avance ***

- Les instructions Dockerfile sont executees dans l ordre
- Chaque instruction cree une nouvelle layer dans l image
- Les instructions sont bufferisees. Si pas de changement, l instruction est omise et le cache est utilisee
- L instruction *_FROM_* doit etre la premiere instruction
- Commentaires commencent avec *_#_*
- On ne peut avoir qu une seule instruction *_CMD_* et une seule instruction *_ENTRYPOINT_*. *_En realite on peut en avoir plusieurs, seule la derniere sera effective_*

****_FROM_* :***

- Specifie l'Image Source
- On ***peut avoir plusieurs instructions*** `_FROM_` et a chaque `_FROM_` on commence a construire une nouvelle Image

***_MAINTAINER_* :**

- Designe celui qui a écrit le `_Dockerfile_`
- Optionnel

***_RUN_* :**

- Utilisé la plupart du temps pour installer des packages
- Les changements sont enregistrés dans le filesystem
 - 2 manieres :
 - + shell: la commande sera enveloppe dans `_/_bin/sh -c_`
 - + exec: la commande N est PAS enveloppe dans un shell. Et n a pas besoin de `_/_bin/sh_`
- On peut executer plusieurs commandes avec `_RUN_` en un *`_seul coup_`*, ce qui evitera de creer plusieurs layers, e.g.

```
RUN apt-get update && apt-get install -y wget && apt-get clean
```

ou bien

```
RUN apt-get update \
    && apt-get install -y wget \
    && apt-get clean
```

***_EXPOSE_* :**

- Equivalent `_--expose_` en ligne de commande
- Tous les ports sont prives par default. Un port prive n est pas accessible depuis l'exterieur du Container
- Un port public est accessible depuis le host et les autres Containers
- `_Dockerfile_` ne controle pas si le port est public ou pas, c est lors de `_docker run -P ..._` ou bien `_docker run -p <port> ..._` que le port devient public
- Grosso mode, `_EXPOSE_` n autorise pas la communication via le port defini dans le Container. Le container dit simplement mon service est disponible sur ce port. Si on veut l'utiliser depuis l'exterieur, il faut le mapper, i.e. `_publish_`

***_VOLUME_* :**

- Ajout d'un volume

***_WORKDIR_* :**

- Repertoire de travail du Container. Peut affecter `_CMD_` ou bien `_ENTRYPOINT_`
- On peut avoir plusieurs instructions `_WORKDIR_`

***_ENV_* :**

- Specifie une variable d'environnement

***_USER_* :**

- Specifier l'utilisateur à utiliser lors de l'exécution du Container
- On peut plusieurs instructions `_USER_`

***_ONBUILD_* :**

- Appelée trigger
- On construit une image de base Mere avec Dockerfile contenant des instructions `_ONBUILD_`
- Ces instructions ne sont pas exécutées dans cette image de base Mere
- Elles le seront lorsque des images Filles seront créées à partir de cette image de base

***[INFO] 71) Docker Swarm ***

***Definition: ***

- Docker Node – a machine running the Docker daemon. It is an instance of Docker Engine participating in the Swarm.
- Swarm Host – a machine running the Swarm daemon
- Swarm – a series of Docker nodes
- Docker Engines participating in a cluster are running in Swarm mode

***What is Docker Swarm? ***

- Docker Swarm is a native clustering system for Docker. It allows you to define your cluster (swarm) and create/control docker images and containers throughout the cluster through the Docker Swarm daemon.

***Why Use Docker Swarm? ***

- Docker swarm is a relatively simple tool for optimizing your container workloads across your cluster while using the standard docker commands you're already familiar with.

***How We Enable a Swarm Mode For a Docker Engine? ***

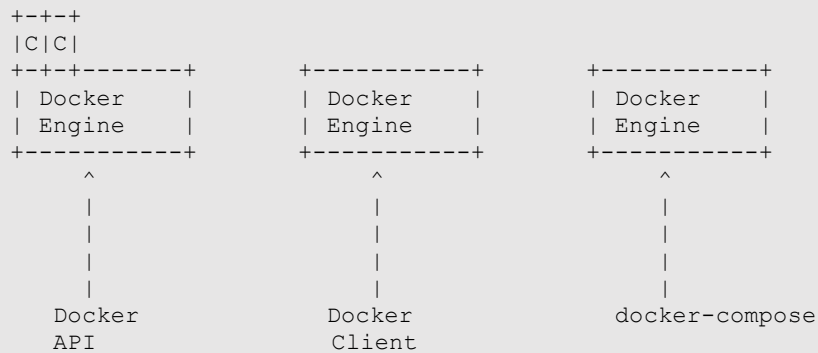
- By Either initializing a Swarm or joining an an existing Swarm

***Where do we run nodes? ***

- You can run one or more nodes on a Single Physical Computer or MULTIPLE Physical Machines

***[INFO] 72) Docker Swarm ***

Jusqu'a present, on a communique avec le Docker Engine EN DIRECT de differentes manieres:



[Note]: On n a pas fait de Lab avec l API, mais c est possible

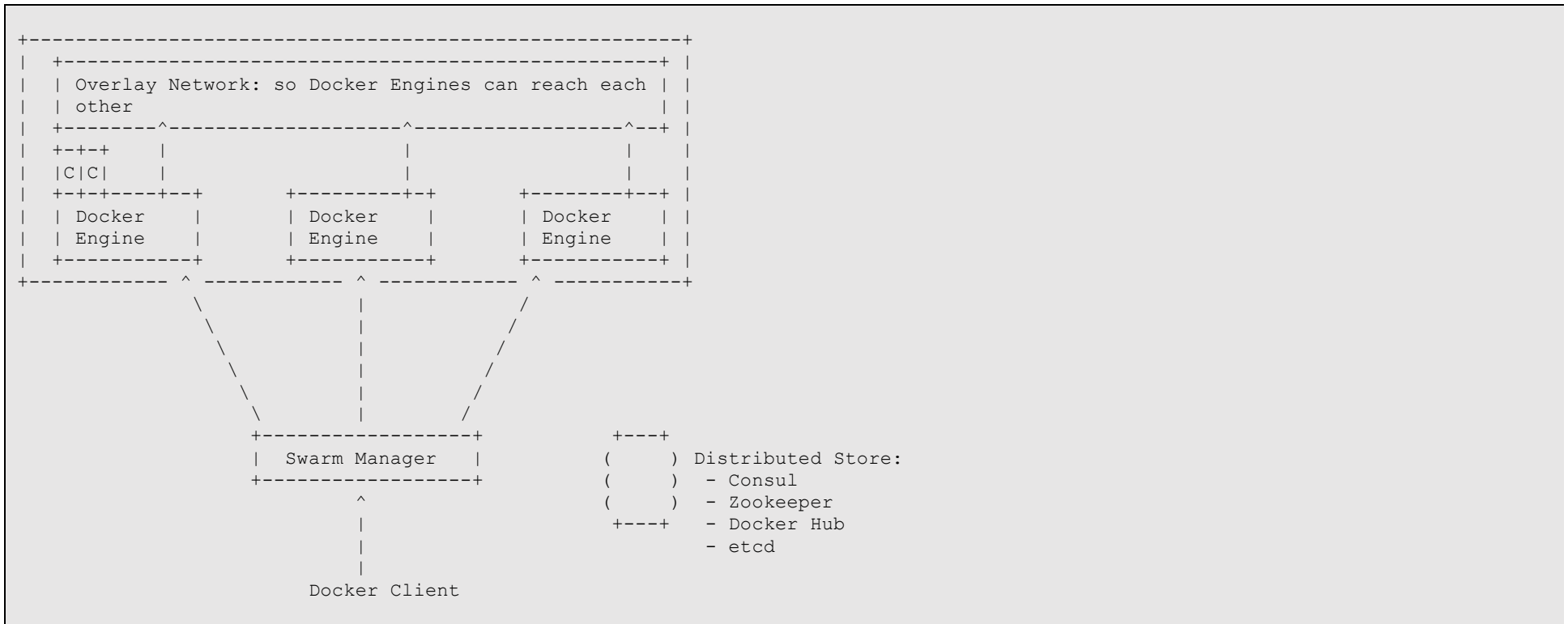
Le but de Swarm est d unifier la communication entre le client et les differents Docker Engines

J ai identifie 2 manieres de 'faire du Swarm':

- a) OLD WAY Docker Engine < 1.12 Maintenant appele Swarm Standalone
- b) NEW WAY: Docker Engine >= 1.12 Appele Swarm MODE

***[INFO] 73) Docker Swarm - OLD WAY: Docker Engine < 1.12 Appele Maintenant 'Swarm Standalone' ***

[Note]: Si vous cherchez de la doc sur ce mode, ne cherchez pas avec les mots cles 'Swarm Standalone', car il a ete nomme comme cela que lorsque 'Swarm Mode' a ete introduit. Cherchez plutot par date, je crois que c est 2015



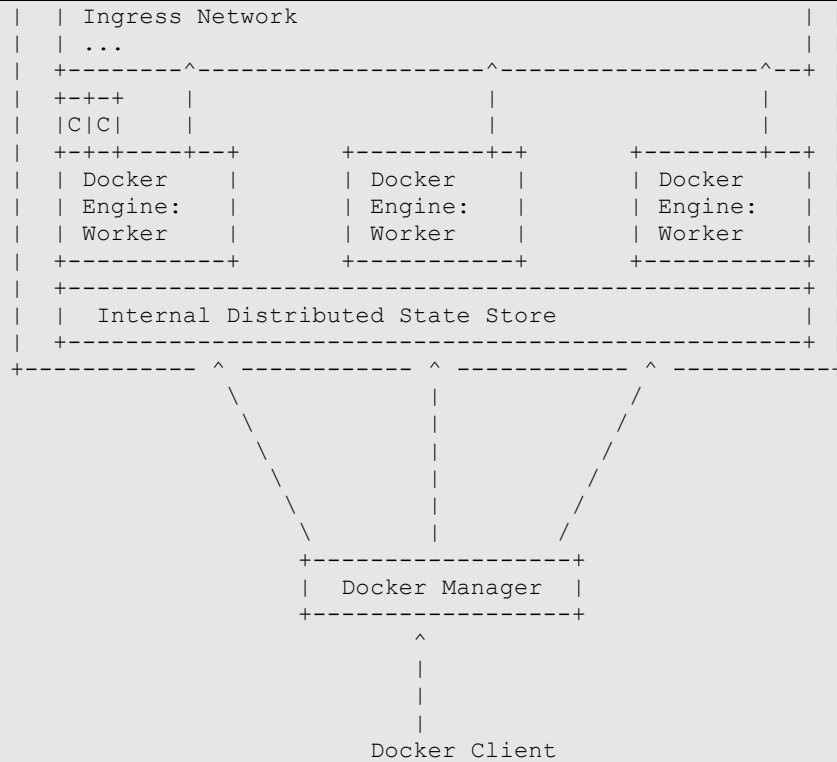
Distributed Store: Tous les Docker Engines s'enregistrent dedans. Sarm Manager s'en sert pour se connecter aux differents Docker Engines

[INFO] 74) Docker Swarm - NEW WAY: Docker Engine >= 1.12 Appel Swarm MODE

2 types de nodes:

- Worker
- Manager





[INFO] 75) Stack - Service - Tasks

Stack:	Services:	Tasks
Group of Services deployed together	Group of Tasks	a Container is Task
	One Container per Task	
		Task 1
	Service 1	Task 2
		...


```

|           | Docker Engine Mgr |
|           | +-----+-----+
|           | | Ctnr Visualizer |
|           | +-----+-----+
|           | ^
+-----+-----+
IP: xx.xx.xx.xx      IP: xx.xx.xx.xx
IP: xx.xx.xx.xx      |
IP: xx.xx.xx.xx      |
|                     |
|                     |

```

[LAB] 75b) Docker Swarm - Deployer stacks

***a) On a besoin de creer les 6 machines docker engine: ***

- 1 Swarm Manager
- 5 Swarm workers

***b) Script qui va nous permettre de creer ses 6 machines et de former un cluster swarm: ***

[reference]: <http://mmorejon.github.io/blog/docker-swarm-con-docker-machine-scripts/>

[note]: j ai modifie le script

```

swarm-create.sh:

#!/bin/bash

# Creating 6 nodes
echo "### Creating node manager ..."
docker-machine create -d virtualbox swarm-manager

echo "### Creating nodes workers ..."
for c in {1..5} ; do
    docker-machine create -d virtualbox node$c
done

# Get IP from leader node
leader_ip=$(docker-machine ip swarm-manager)

# Init Docker Swarm mode
echo "### Initializing Swarm mode ..."
eval $(docker-machine env swarm-manager)
docker swarm init --advertise-addr $leader_ip

```

```
# Swarm tokens
manager_token=$(docker swarm join-token manager -q)
worker_token=$(docker swarm join-token worker -q)

# Join worker nodes
echo "### Joining worker nodes ..."
for c in {1..5} ; do
    eval $(docker-machine env node$c)
    docker swarm join --token $worker_token $leader_ip:2377
done

# Clean Docker client environment
echo "### Cleaning Docker client environment ..."
eval $(docker-machine env -u)
```

***c) on liste tous les nodes : ***

e3rl2p13% docker node ps

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE	ERROR
----	------	-------	------	---------------	---------------	-------

PORTS

e3rl2p13% docker node ls

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS	ENGINE VERSION
qip18euymg3yefmysq94lrnv4	node1	Ready	Active		18.06.1-ce
hzcyd82wpjta0y0578h3fzm0z	node2	Ready	Active		18.06.1-ce
r7toxyvld8lay118qst5kunux	node3	Ready	Active		18.06.1-ce
kbit4ie7g3cb26ck20h5ll10v	node4	Ready	Active		18.06.1-ce
fl2alsyqwt86jqh7p1qrb562	node5	Ready	Active		18.06.1-ce
xcy5upm9hk7ly43akegs6u0pf *	swarm-manager	Ready	Active	Leader	18.06.1-ce

[LAB] 75c) Docker Swarm - Deployer stacks

***d) On lance le Container `_Visualizer_` sur le `_swarm-manager_` : ***

```
visualizer.yml:
  version: '3.4'

  services:
```

```
visualizer:
  image: dockersamples/visualizer
  deploy:
    placement:
      constraints: [node.role == manager]
  ports:
    - 4242:8080
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock:ro
```


```
e3r12p13% docker stack deploy --compose-file=stacks/visualizer/visualizer.yml visualizer
```

***e) On se connecte a ce container ***

```
e3r12p13% docker-machine ip swarm-manager
192.168.99.100
```

Visualizer x

← → ↻ ① Not Secure | 192.168.99.100:4242



filter containers

<div>● node1 worker 0.972G RAM x86_64/linux</div> <div></div>	<div>● node2 worker 0.972G RAM x86_64/linux</div> <div></div>	<div>● node3 worker 0.972G RAM x86_64/linux</div> <div></div>	<div>● node4 worker 0.972G RAM x86_64/linux</div> <div></div>	<div>● node5 worker 0.972G RAM x86_64/linux</div> <div></div>	<div>● swarm-man... manager 0.972G RAM x86_64/linux</div> <div><div>● visualizer_visualizer image : visualizer:latest@sha256:f1e6af29c43641 tag : latest@sha256:f1e6af29c43641 updated : 25/8 16:5 0064f3e4246cd15be1df26fd157153 state : running</div></div>
---	---	---	---	---	---

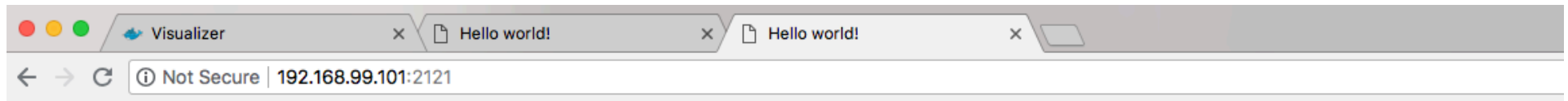
*** e) On lance les Containers _hello-world_ sur les nodes: ***

```
dockercloud-hello-world.yml:
  version: '3.4'
  services:
    web:
      image: dockercloud/hello-world
      deploy:
        #mode: replicated
        replicas: 7
        placement:
          constraints:
            - node.role == worker
      ports:
        - 2121:80
```

```
e3r12p13% docker stack deploy --compose-file=stacks/dockercloud-hello-world/dockercloud-hello-world.yml helloworld
```

***f) On se connecte a ces containers : ***

```
http://192.168.99.10[0-5]:2121
```



Hello world!

My hostname is 6cbb0ce4e8d3

[LAB] 75e) Docker Swarm - Deployer stacks

***g) On peut visualiser la repartition de nos Containers sur les 5 nodes: ***

The screenshot shows the Docker Visualizer web interface in a browser. The address bar indicates the URL is 192.168.99.100:4242. The interface displays a Swarm cluster with 6 nodes, each labeled as a 'worker' or 'manager' with 0.972G RAM and x86_64/linux architecture. The nodes are arranged in a grid. Below each node, containers are listed with their names, images, tags, update times, IDs, and states. The containers are 'helloworld_web' (5 instances) and 'visualizer_visualizer' (1 instance).

Node	Role	RAM	Architecture	Container Name	Image	Tag	Updated	ID	State
node1	worker	0.972G	x86_64/linux	helloworld_web	hello-world:latest@sha256:c6739be467722	latest@sha256:c6739be467722	25/8 16:17	817f7b29bf4f0bc34f33529a3b9e26d	running
node2	worker	0.972G	x86_64/linux	helloworld_web	hello-world:latest@sha256:c6739be467722	latest@sha256:c6739be467722	25/8 16:17	7a93c136e69ac9575939d71426d7f7	running
node3	worker	0.972G	x86_64/linux	helloworld_web	hello-world:latest@sha256:c6739be467722	latest@sha256:c6739be467722	25/8 16:17	6cbb0ce4e8d34e88e1cbf2b2100542	running
node4	worker	0.972G	x86_64/linux	helloworld_web	hello-world:latest@sha256:c6739be467722	latest@sha256:c6739be467722	25/8 16:17	5f0ae9d3cb9c6c6325d664acc1a651c	running
node5	worker	0.972G	x86_64/linux	helloworld_web	hello-world:latest@sha256:c6739be467722	latest@sha256:c6739be467722	25/8 16:17	5044a8acd9af8d506c1eaa3906ee80	running
swarm-man...	manager	0.972G	x86_64/linux	visualizer_visualizer	visualizer:latest@sha256:f1e6af29c43641	latest@sha256:f1e6af29c43641	25/8 16:5	0064f3e4246cd15be1df26fd157153	running

[LAB] 76) Ques commandes a examiner

```
docker container prune --force
docker system df
docker system prune
docker build -t test --squash .
docker service logs myservice
docker image search
```

[LABs] 70) Dockerfile - _ONBUILD_

Image Mere: _Dockerfile_

```
FROM busybox
ONBUILD RUN echo "You won't see me until later"
```

Build image Mere:

```
e1rlp12% docker build -t image_mere .
Sending build context to Docker daemon 15.87kB
Step 1/2 : FROM busybox
...
---> e1ddd7948alc
Step 2/2 : ONBUILD RUN echo "You won't see me until later"
...
Successfully built d00310c7f3c4
Successfully tagged image_mere:latest
```

Lister images:

```
e1rlp12% docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE	
>>> image_mere	latest	d00310c7f3c4	14 seconds ago	1.16MB	<<<
busybox	latest	e1ddd7948alc	13 days ago	1.16MB	

Image Fille: _Dockerfile_

```
FROM image_mere
```

Build Image Fille:

```
e1rlp12% docker build -t image_fille .
Sending build context to Docker daemon 3.072kB
Step 1/1 : FROM image_mere
# Executing 1 build trigger
---> Running in 8e51531ab332
>>> You won't see me until later <<<<
...
Successfully built 252e267bbae8
Successfully tagged image_fille:latest
```

Lister images:

```
e1rlp12% docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE	
>>> image_fille	latest	252e267bbae8	About a minute ago	1.16MB	<<<
image_mere	latest	d00310c7f3c4	3 minutes ago	1.16MB	
busybox	latest	e1ddd7948a1c	13 days ago	1.16MB	

***[Misc.] ***

```
e1rl2p13% docker run -d -v "/var/run/docker.sock:/var/run/docker.sock" -p 9000:9000 portainer/portainer
```