

CESI Blanquefort 2015 RARE 08

# Projet Linux : DOCKER / Owncloud 8

Ceci est une révolution !

Julien Bréhier Julien Faure  
20/03/2015

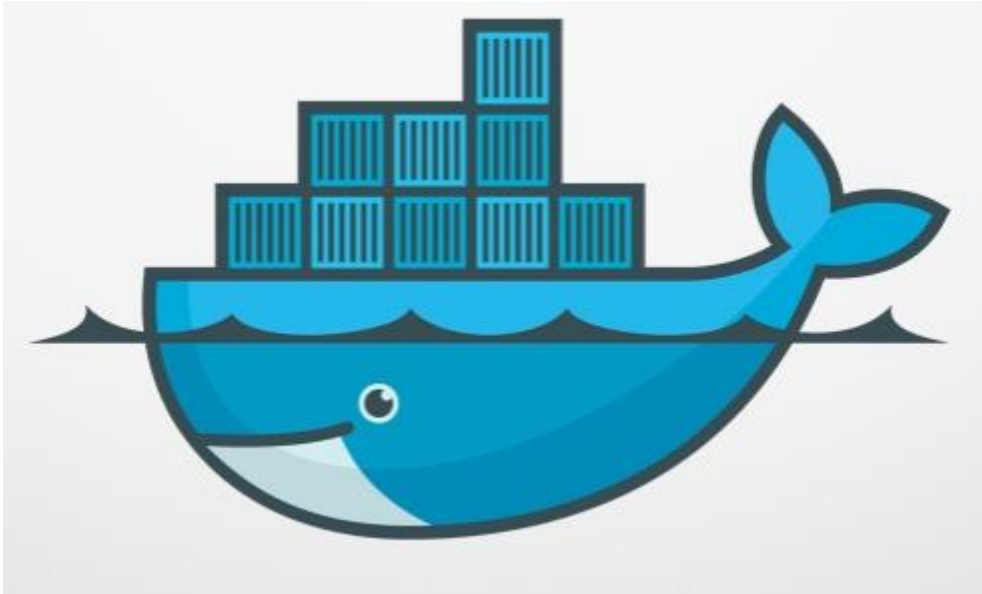
## Table des matières

1	Présentation de Docker .....	3
1.1	Définition .....	3
1.2	Use Cases .....	4
1.3	Caractéristiques techniques générales .....	5
1.4	La virtualisation classique vs Docker .....	5
1.5	Diagramme Docker .....	6
1.6	Les évolutions de Docker .....	7
1.7	Les outils et surcouches .....	7
1.8	Conclusion .....	7
2	Installation de Docker .....	8
2.1	Prérequis .....	8
2.2	Installer Docker sur Ubuntu 14.04 .....	8
3	Manipulation des containers Docker .....	9
3.1	Lancement d'un container .....	9
3.2	Connaitre la version de Docker installée .....	10
3.3	Afficher toutes les commandes docker .....	10
3.4	Afficher l'aide commande spécifique .....	10
3.5	Lancer une application Web dans Docker .....	10
3.6	Voir les containers en fonctionnement .....	11
3.7	Modifier le port d'accès d'un container .....	11
3.8	Vérifier les logs d'un container .....	11
3.9	Afficher les processus en cours dans un container .....	11
3.10	Arrêter un container .....	12
3.11	Supprimer un container .....	12
4	Manipulation des images Docker .....	13
4.1	Lister les images présentes sur l'hôte Docker .....	13
4.2	Télécharger une image .....	13
4.3	Rechercher une image .....	14
5	Créer ses propres images .....	15
5.1	Modifier et sauvegarder une image .....	15
5.2	Modifier le tag d'une image Docker .....	16
5.3	Envoyer son image sur le Docker Hub .....	18
5.4	Supprimer une image de l'hôte Docker .....	18
6	Lier des containers entre eux .....	18
6.1	Etablir le nommage des containers .....	19

6.2	Communiquer grâce aux liens .....	19
7	Gérer les données au sein d'un container.....	20
7.1	Les Data Volumes .....	20
7.2	Ajouter un Data Volume à un container .....	20
7.3	Utiliser un répertoire de l'hôte Docker comme Data Volume .....	21
7.4	Monter un fichier de l'hôte Docker dans le Data Volume .....	21
7.5	Créer et monter un Data Volume Container .....	21
7.6	Suppression d'un Volume Data Container .....	22
7.7	Sauvegarder, restaurer ou migrer des Data Volumes .....	22
8	Installation de Owncloud via Dockerfile.....	23
8.1	Création du dépôt dur DockerHub .....	23
8.2	La création du Dockerfile.....	26
8.3	Création du Dockerfile Owncloud .....	27
9	Test de l'image Owncloud .....	28
9.1	Recherche de l'image owncloud .....	28
9.2	Téléchargement de l'image Owncloud.....	28
9.3	Création d'un container avec l'image owncloud.....	29
9.4	Vérification du fonctionnement du container .....	29
9.5	Arrêt du container .....	30
9.6	Suppression du container.....	30
9.7	Suppression de l'image aries4/owncloud .....	30

# Projet Linux : Docker

Exemple : Owncloud 8



## 1 Présentation de Docker

Source : <http://blog.nicolargo.com/2014/09/support-presentation-docker.html>

### 1.1 Définition

" Docker est une solution permettant d'exécuter un ou plusieurs logiciels dans des environnements séparés (conteneurs) pouvant communiquer entre eux. "

Un conteneur propose:

- un espace isolé permettant d'exécuter des processus La plupart du temps 1 conteneur = 1 application
- un accès root
- une adresse IP pour communiquer avec le reste du monde & autres conteneurs

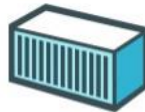
Par sa nature Docker assure que le conteneur utilisé en développement fonctionnera de la même manière en production.

Le but ultime de Docker est de minimiser le temps et les infrastructures entre le développement, les tests, le déploiement et l'utilisation en production.

## 1.2 Use Cases



Build



Ship



Run

Quelques exemples de sociétés utilisant Docker :

- Rackspace
- Red Hat
- IBM
- Google
- Spotify
- Amazon
- Ebay
- Twitter
- Facebook
- ...



### 1.3 Caractéristiques techniques générales

Docker est :

- Logiciel Open-Source (licence Apache 2)
- Développé en langage Go
- Service en ligne (Docker.io)
- Communauté importante et active (DockerCon)
- Solomon Hykes est le fondateur et l'actuel CTO de Docker.io

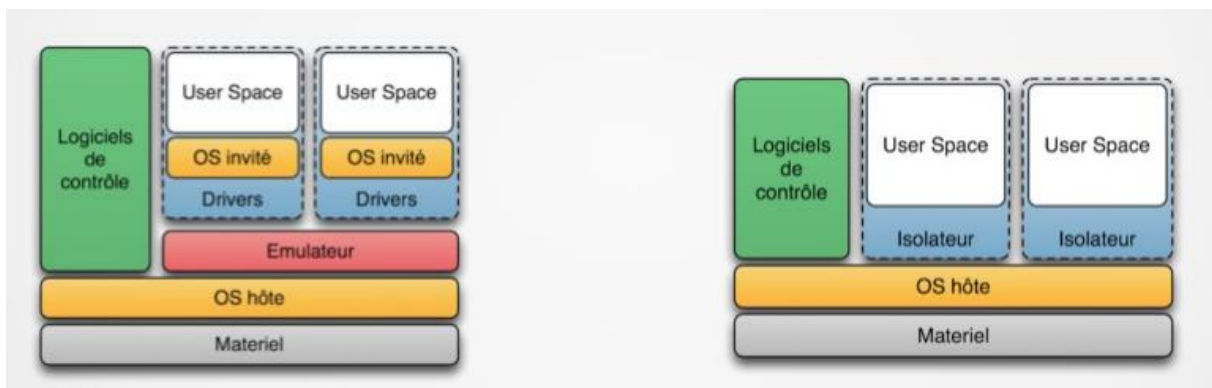
Dans quel but :

- Plate-forme de développement souple
- Intégration continue
- Déploiement/mise en production
- Plate-forme PaaS

Comment ça marche :

- Docker est une solution de virtualisation légère basée sur des conteneurs (containers)
- Un conteneur est un environnement d'exécution isolé (vraiment) avec ses propres ressources
- Tous les conteneurs utilisent le même noyau qui est celui de la machine hôte
- Docker se basait uniquement sur le noyau Linux et ses technologies LXC, namespaces et cgroups puis sur libcontainer depuis la version 0.9

### 1.4 La virtualisation classique vs Docker



Virtualisation classique	Containers ou virtualisation légère
+ flexible: émulation complète ou partielle d'une machine sur une autre - coûts de mise en œuvre - instance consommatrice en ressources  VMWare, Xen, VirtualBox, KVM, Hyper-V...	+ coûts de mise en œuvre + rapidité de lancement des environnements - flexibilité (virtualisation de l'environnement d'exécution, pas de la machine)  OpenVZ, Vserver, Jail et... Docker

Les performances :

- Un PC portable peut faire tourner jusqu'à 100 conteneurs
- 1000 conteneurs sur un serveur
- A l'intérieur des conteneurs, les logiciels tournent aussi vite que si ils étaient lancées sur l'OS hôte. Les opérations sur les conteneurs se font dans la seconde

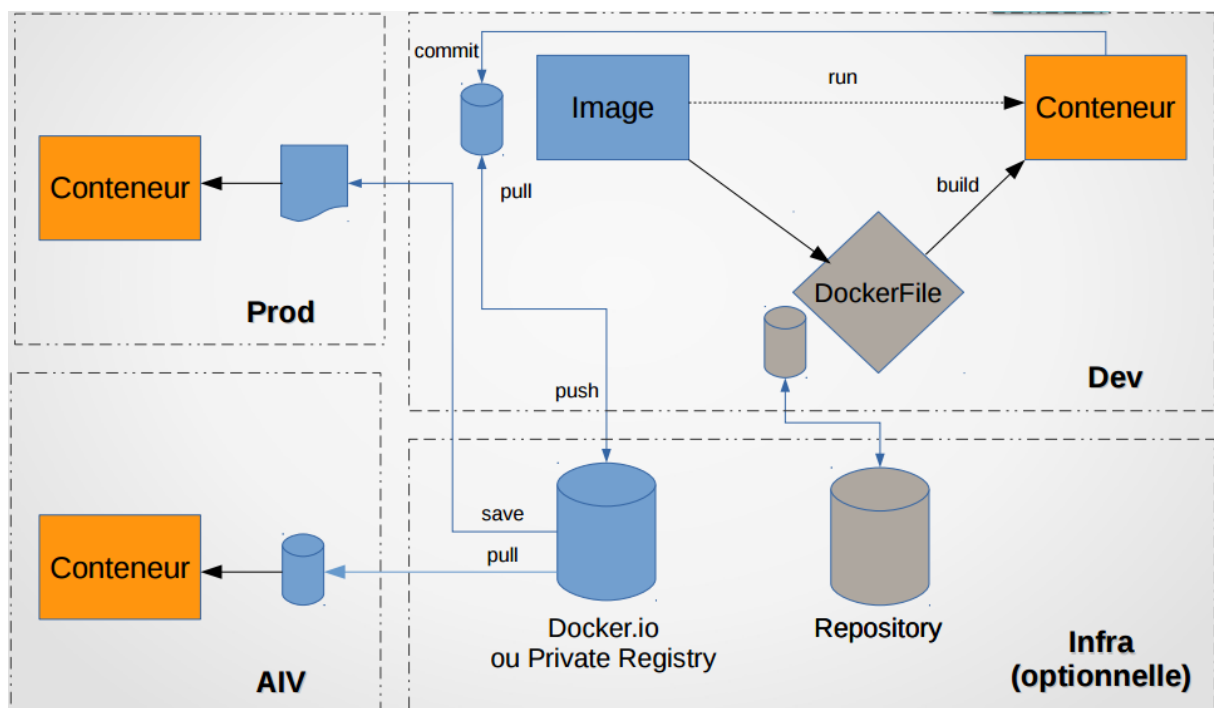
Du neuf avec du vieux ... :

C'est en partie vrai mais Docker apporte nativement:

- un environnement d'administration des conteneurs avec notamment le build basé sur les DockerFiles
- la gestion en version des conteneurs
- une registry pour stocker ses images/conteneurs
- une API REST

Docker est donc plus une surcouche aux solutions comme Jail ou OpenVZ

## 1.5 Diagramme Docker



## 1.6 Les évolutions de Docker

<https://blog.docker.com/2015/02/orchestrating-docker-with-machine-swarm-and-compose/>

Docker va désormais proposer de l'orchestration de containers nativement. Pour ce faire il va utiliser 3 éléments :

- Machine : cet outil permet le déploiement de Docker engine sur différentes plateformes et donc sur plusieurs serveurs simultanément.
- Swarm : cet outil permet de réaliser des clusters de Docker engines et de n'en faire qu'un seul et même hôte virtuel.
- Compose : cet outil permet de réaliser et de distribuer des containers sur plusieurs hôtes simultanément.

## 1.7 Les outils et surcouches

Plusieurs outils permettent d'ores et déjà de réaliser ces opérations mais sont souvent spécialisés dans un domaine d'application particuliers et surtout proposent différentes logiques de mise en place et d'exploitation :

- Mesos
- Kubernetes
- Decking.io
- Shipyard
- ...

On peut aussi retrouver différentes distributions Linux intégrant certains outils tels que :

- <http://www.projectatomic.io/>
- <http://www.ubuntu.com/cloud/tools/snappy>
- <https://coreos.com/>

## 1.8 Conclusion

Enfin pour finir, rien ne vaut la source officielle :

<https://www.docker.com/>



## 2 Installation de Docker

L'installation de Docker ne présente pas de difficultés particulières. Il peut y avoir certaines variantes en fonction de la distribution du système d'exploitation de l'hôte mais globalement, les étapes restent les mêmes.

Nous allons vous présenter comment installer Docker sur une distribution Ubuntu 14.04.

### 2.1 Prérequis

On pourrait installer le paquet Docker directement depuis les dépôts Ubuntu, mais il n'est pas aussi à jour que celui du dépôt officiel. Docker étant un projet encore jeune, il est préférable d'installer la dernière version corrigeant certains bugs. Nous allons donc installer Docker depuis le dépôt officiel.

### 2.2 Installer Docker sur Ubuntu 14.04

Tout d'abord on vérifie que APT gère le HTTPS :

```
Si /usr/lib/apt/methods/https existe > OK

Sinon

apt-get update
apt-get install apt-transport-https
```

On ajoute le la keychain du dépôt Docker :

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys
36A1D7869245C8950F966E92D8576A8BA88D21E9
```

On ajoute le dépôt Docker et on installe le paquet :

```
$ sudo sh -c "echo deb https://get.docker.com/ubuntu docker main\
> /etc/apt/sources.list.d/docker.list"
$ sudo apt-get update
$ sudo apt-get install lxc-docker
```

On peut aussi utiliser un script via curl qui automatise toute la procédure précédente :

```
$ curl -sSL https://get.docker.com/ubuntu/ | sudo sh
```

On ajoute Docker au démarrage de la machine :

```
[ubuntu@ip-172-31-16-57~]$ sudo update-rc.d docker defaults
```

Pour vérifier que tout fonctionne :

```
$ sudo docker run -i -t ubuntu:14.04 /bin/bash
```

Cela doit télécharger l'image ubuntu :14.04 et lancer bash dans le container :

```
ubuntu@ip-172-31-16-56:~$ sudo docker run -i -t ubuntu:14.04 /bin/bash
Unable to find image 'ubuntu:14.04' locally
511136ea3c5a: Pull complete
511136ea3c5a: Download complete
f3c84ac3a053: Download complete
ala958a24818: Download complete
9fec74352904: Download complete
d0955f21bf24: Download complete
Status: Downloaded newer image for ubuntu:14.04
root@718d4d202682:/# pwd
/
root@718d4d202682:/# hostname
718d4d202682
root@718d4d202682:/#
```

### 3 Manipulation des containers Docker

Docker permet de créer des containers contenant des applications spécifiques. Aussi, il y a de nombreuses commandes permettant d'effectuer des actions sur ces containers.

#### 3.1 Lancement d'un container

```
ubuntu@ip-172-31-33-174:~$ sudo docker run -d -P training/webapp python
app.py
Unable to find image 'training/webapp:latest' locally
Pulling repository training/webapp
31fa814ba25a: Download complete
511136ea3c5a: Download complete
f10ebce2c0e1: Download complete
82cdea7ab5b5: Download complete
5dbd9cb5a02f: Download complete
74fe38d11401: Download complete
64523f641a05: Download complete
0e2afc9aad6e: Download complete
e8fc7643ceb1: Download complete
733b0e3dbcee: Download complete
alfeb043c441: Download complete
e12923494f6a: Download complete
a15f98c46748: Download complete
Status: Downloaded newer image for training/webapp:latest
dea2ac2aee49f7e7ea2328863f953aa78dbfb66d1513c1a08d043af1abce2b33
```

### 3.2 Connaître la version de Docker installée

```
ubuntu@ip-172-31-33-174:~$ sudo docker version
Client version: 1.5.0
Client API version: 1.17
Go version (client): go1.4.1
Git commit (client): a8a31ef
OS/Arch (client): linux/amd64
Server version: 1.5.0
Server API version: 1.17
Go version (server): go1.4.1
Git commit (server): a8a31ef
```

### 3.3 Afficher toutes les commandes docker

```
ubuntu@ip-172-31-33-174:~$ sudo docker

Commands:
  attach      Attach to a running container
  build       Build an image from a Dockerfile
  commit      Create a new image from a container's changes
  cp          Copy files/folders from a container's filesystem to the host
  path
  create      Create a new container
  diff        Inspect changes on a container's filesystem
  events      Get real time events from the server
  exec        Run a command in a running container
  export      Stream the contents of a container as a tar archive
  history     Show the history of an image
  images      List images
  import      Create a new filesystem image from the contents of a tarball
```

### 3.4 Afficher l'aide commande spécifique

```
ubuntu@ip-172-31-33-174:~$ sudo docker pull --help

Usage: docker pull [OPTIONS] NAME[:TAG]

Pull an image or a repository from the registry

  -a, --all-tags=false    Download all tagged images in the repository
  --help=false            Print usage
```

### 3.5 Lancer une application Web dans Docker

Pour lancer un container Docker, il suffit d'exécuter la commande Run comme suit :

```
$ sudo docker run -d -P training/webapp python app.py
```

### 3.6 Voir les containers en fonctionnement

```
ubuntu@ip-172-31-33-174:~$ sudo docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
dea2ac2aee49	training/webapp:latest	"python app.py"	4 minutes ago
Up 4 minutes	0.0.0.0:49153->5000/tcp	reverent_jang	

### 3.7 Modifier le port d'accès d'un container

Pour accéder au container depuis le Web via un port spécifique.

```
ubuntu@ip-172-31-33-174:~$ sudo docker run -d -p 5000:5000 training/webapp python app.py
```

Chaque modification de port crée un nouveau container en laissant le container de base intact

```
ubuntu@ip-172-31-33-174:~$ sudo docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
7566e38a1e87	training/webapp:latest	"python app.py"	2 hours ago
Up 2 hours	0.0.0.0:49155->5000/tcp	sharp_bell	
9e00aaa94895	training/webapp:latest	"python app.py"	2 hours ago
Up 2 hours	0.0.0.0:5000->5000/tcp	sharp_payne	
dea2ac2aee49	training/webapp:latest	"python app.py"	3 hours ago
Up 3 hours	0.0.0.0:49153->5000/tcp	reverent_jang	

### 3.8 Vérifier les logs d'un container

```
ubuntu@ip-172-31-33-174:~$ sudo docker logs -f sharp_payne
```

```
* Running on http://0.0.0.0:5000/
90.16.183.146 - - [15/Mar/2015 18:56:53] "GET / HTTP/1.1" 200 -
90.16.183.146 - - [15/Mar/2015 18:56:53] "GET /favicon.ico HTTP/1.1" 404 -
90.16.183.146 - - [15/Mar/2015 18:56:53] "GET /favicon.ico HTTP/1.1" 404 -
90.16.183.146 - - [15/Mar/2015 19:00:44] "GET / HTTP/1.1" 200 -
90.16.183.146 - - [15/Mar/2015 21:03:46] "GET / HTTP/1.1" 200 -
```

On voit ici toutes les requêtes http faites sur le container depuis sa création.

### 3.9 Afficher les processus en cours dans un container

```
ubuntu@ip-172-31-33-174:~$ sudo docker top sharp_payne
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	5002	2953	0	18:56	?	00:00:01	python app.py

La fin de la commande sert à spécifier le nom du container dont on souhaite afficher les processus. Cela permet d'afficher toutes les informations relatives au processus.

### 3.10 Arrêter un container

```
ubuntu@ip-172-31-33-174:~$ sudo docker stop reverent_jang
reverent_jang
```

Une fois arrêté, le nom du container spécifié s'affiche de nouveau.

Démarrer ou redémarrer un container

Il suffit d'utiliser la commande « docker start » pour démarrer un container arrêté ou « restart » pour redémarrer un container sans l'avoir arrêté

```
ubuntu@ip-172-31-33-174:~$ sudo docker start reverent_jang
reverent_jang
```

```
ubuntu@ip-172-31-33-174:~$ sudo docker restart reverent_jang
reverent_jang
```

### 3.11 Supprimer un container

La commande pour supprimer un container est « docker rm [nom\_container] »

```
ubuntu@ip-172-31-33-174:~$ sudo docker rm reverent_jang
Error response from daemon: Conflict, You cannot remove a running
container. Stop the container before attempting removal or use -f
FATA[0000] Error: failed to remove one or more containers
```

Comme le montre le message ci-dessus, il n'est pas possible de supprimer un container en cours d'exécution. Cela évite la suppression en cas d'utilisation. Il faut d'abord arrêter le container avant de pouvoir le supprimer.

```
ubuntu@ip-172-31-33-174:~$ sudo docker stop reverent_jang
reverent_jang
ubuntu@ip-172-31-33-174:~$ sudo docker rm reverent_jang
reverent_jang
```

**Note importante : la suppression d'un container est définitive.**

## 4 Manipulation des images Docker

Pour créer des containers, Docker se base sur des modèles, les images. Ces images sont stockées sur l'hôte sur lequel est installé Docker. Si l'image souhaitée n'est pas disponible sur l'hôte, il est nécessaire de la télécharger depuis un dépôt. De base, les images sont téléchargées depuis le dépôt « Docker Hub Registry ». Ce dépôt permet de télécharger des images officielles ou faites par d'autres utilisateurs mais également d'y envoyer des images que l'on aurait modifiées. Ces images peuvent être envoyées sur la partie « publique » du dépôt pour les mettre à disposition de toute la communauté, ou sur sa partie « privée » afin d'en limiter l'accès aux personnes souhaitées.

### 4.1 Lister les images présentes sur l'hôte Docker

```
ubuntu@ip-172-31-33-174:~$ sudo docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED
ubuntu	trusty-20150228.11	2103b00b3fdf	5 days ago
188.3 MB			
ubuntu	14.04	2103b00b3fdf	5 days ago
188.3 MB			
ubuntu	14.04.2	2103b00b3fdf	5 days ago
188.3 MB			
ubuntu	latest	2103b00b3fdf	5 days ago
188.3 MB			
ubuntu	trusty	2103b00b3fdf	5 days ago
188.3 MB			
training/webapp	latest	31fa814ba25a	9 months ago
278.8 MB			

On peut voir le nom de l'image, son dépôt d'origine, sa taille, son ID et sa date de création. Si plusieurs images ont le même nom comme ici, cela permet d'identifier l'image souhaitée, par exemple en fonction de la version, afin de choisir celle qui convient.

La commande suivante permet de lancer un container avec image spécifique :

```
ubuntu@ip-172-31-33-174:~$ sudo docker run ubuntu:14.04.2
```

Par défaut, si aucune image n'est spécifiée, c'est la version « latest » qui est lancée.

### 4.2 Télécharger une image

La commande « pull » permet de télécharger une image depuis le dépôt.

Il est possible de télécharger une image sans la lancer, permettant de l'exécuter ultérieurement beaucoup plus rapidement.

```
ubuntu@ip-172-31-33-174:~$ sudo docker pull test/test
```

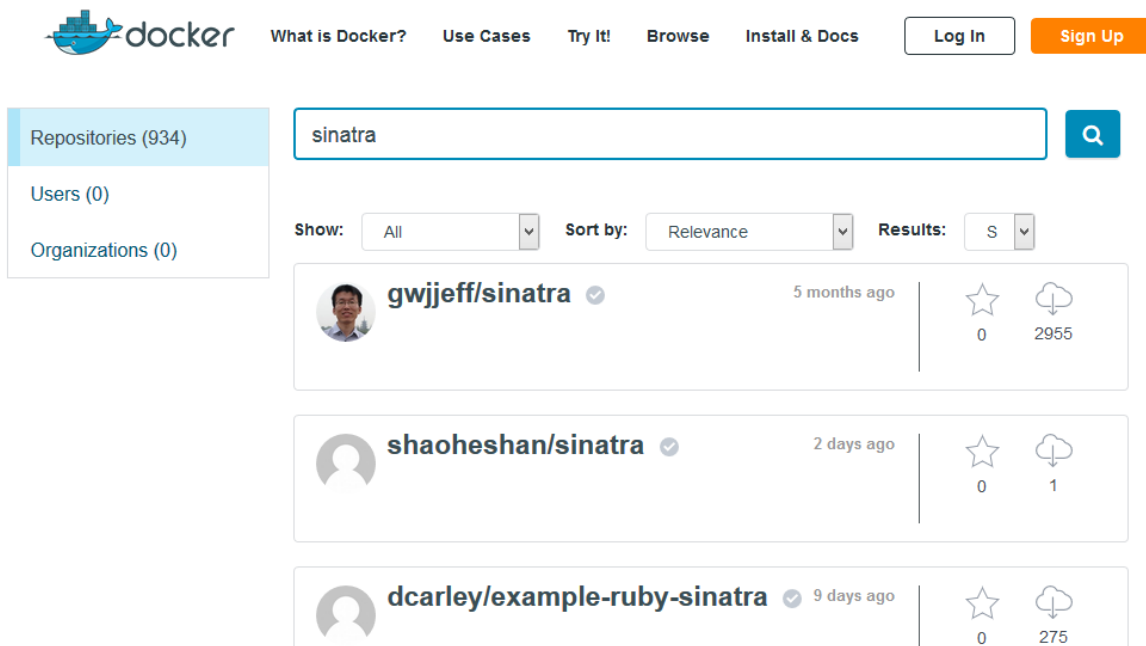
### 4.3 Rechercher une image

Si l'on ne connaît pas le nom de l'image associée à l'application souhaitée, il est possible de faire une recherche au sein des images du repository. Par exemple, si l'on souhaite une image où Sinatra (création d'applications Web en Ruby) est installé, il convient de lancer la commande suivante :

```
ubuntu@ip-172-31-33-174:~$ sudo docker search sinatra
NAME                                DESCRIPTION
STARS      OFFICIAL    AUTOMATED
training/sinatra
5
shaoheshan/sinatra
0 [OK]
dcarley/example-ruby-sinatra
0 [OK]
larmar/sinatra-puppet
0 [OK]
zoomix/sinatra-galleria
0 [OK]
smashwilson/minimal-sinatra
0 [OK]
andyshinn/sinatra-echo
0 [OK]
synctree/sinatra-echo
0 [OK]
yoheimuta/docker-sinatra
0 [OK]
```

Cela permet d'afficher toutes les images relatives à Sinatra ainsi que la note qui leur réputation utilisateurs.

Les différentes images peuvent également être recherchées dans l'outil de recherche intégré au site Docker :



The screenshot shows the Docker Hub search results for the term 'sinatra'. The top navigation bar includes the Docker logo, 'What is Docker?', 'Use Cases', 'Try It!', 'Browse', 'Install & Docs', 'Log In', and 'Sign Up'. On the left, a sidebar lists 'Repositories (934)', 'Users (0)', and 'Organizations (0)'. The search bar contains 'sinatra'. Below the search bar, filters are set to 'Show: All', 'Sort by: Relevance', and 'Results: S'. Three repositories are listed:

Repository	Updated	Stars	Downloads
gwjjeff/sinatra	5 months ago	0	2955
shaoheshan/sinatra	2 days ago	0	1
dcarley/example-ruby-sinatra	9 days ago	0	275

## 5 Créer ses propres images

Il est possible de modifier une image existante puis de l'enregistrer pour l'adapter à ses besoins. Il est également possible de créer complètement sa propre image en utilisant les « Dockerfile ».

### 5.1 Modifier et sauvegarder une image

Nous nous basons ici sur l'image « training/sinatra ».

On télécharge d'abord l'image :

```
ubuntu@ip-172-31-33-174:~$ sudo docker pull training/sinatra
Pulling repository training/sinatra
f0f4ab557f95: Download complete
...
bfab314f3b76: Download complete
Status: Downloaded newer image for training/sinatra:latest
```

On lance l'image que l'on souhaite modifier :

```
ubuntu@ip-172-31-33-174:~$ sudo docker run -t -i training/sinatra /bin/bash
root@91cdf611f5f:/#
```

A l'intérieur du container on ajoute le json GEM que l'on souhaite :

```
root@91cdf611f5f:/# gem install json
Fetching: json-1.8.2.gem (100%)
Building native extensions. This could take a while...
```



```
Successfully installed json-1.8.2
1 gem installed
Installing ri documentation for json-1.8.2...
Installing RDoc documentation for json-1.8.2...
```

Une fois les modifications effectuées, on quitte le container :

```
root@91cdfe611f5f:/# exit
```

Il faut ensuite sauvegarder l'image modifiée avec la commande « Commit » :

```
ubuntu@ip-172-31-33-174:~$ sudo docker commit -m "Ajout gem json" -a
"Julien Bréhier" \
> 91cdfe611f5f utilisateur/sinatra_v2
75693b61e10c3c9e2be2670c0ae77a3d7c98435297ed1e4039b03bce5bc48778
```

L'attribut « -m » permet de commenter la modification effectuée et le « -a » permet d'inscrire le nom de l'auteur de la modification. On spécifie ensuite l'ID de l'image de base puis le nom à donner à la nouvelle image.

On peut constater que l'image apparait bien dans la liste des images de l'hôte Docker :

```
ubuntu@ip-172-31-33-174:~$ sudo docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED
utilisateur/sinatra_v2	latest	75693b61e10c	4 minutes ago
ubuntu	14.04.2	2103b00b3fdf	5 days ago
ubuntu	latest	2103b00b3fdf	5 days ago
ubuntu	trusty	2103b00b3fdf	5 days ago
ubuntu	trusty-20150228.11	2103b00b3fdf	5 days ago
ubuntu	14.04	2103b00b3fdf	5 days ago
training/sinatra	latest	f0f4ab557f95	9 months ago
training/webapp	latest	31fa814ba25a	9 months ago

## 5.2 Modifier le tag d'une image Docker

Pour configurer le tag d'une image, il faut utiliser la commande « Docker tag ».

```
ubuntu@ip-172-31-33-174:~$ sudo docker tag 75693b61e10c
utilisateur/sinatra:V3
```

Il suffit d'indiquer l'ID de l'image à tagger, le dépôt et le nom de l'image, puis d'ajouter le nouveau tag (ici, V3)

```
ubuntu@ip-172-31-33-174:~$ sudo docker images utilisateur/sinatra
```

REPOSITORY	TAG	IMAGE ID	CREATED
VIRTUAL SIZE			
utilisateur/sinatra	V3	75693b61e10c	22 hours ago
451.9 MB			

### 5.3 Envoyer son image sur le Docker Hub

Comme nous l'avons vu, il est possible de partager une image que l'on a créée sur le Docker Hub. Il faut pour cela utiliser la commande « `docker push` » comme suit.

```
ubuntu@ip-172-31-33-174:~$ sudo docker push utilisateur/sinatra
The push refers to a repository [utilisateur/sinatra] (len: 1)
Sending image list

Please login prior to push:
Username:
```

Une fois la commande entrée, il n'y a qu'à renseigner son nom d'utilisateur Docker (un compte doit au préalable avoir été créé) pour envoyer l'image dans le dépôt spécifié, ici « `utilisateur/sinatra` ».

### 5.4 Supprimer une image de l'hôte Docker

Pour supprimer une image, comme celle qui nous a servi de modèle pour créer notre propre image, utiliser la « `rmi` » suivi du nom de l'image (ou son ID) :

```
ubuntu@ip-172-31-33-174:~$ sudo docker rmi training/sinatra
Untagged: training/sinatra:latest
```

Une fois la commande lancée, le nom de l'image supprimée est affichée.

Si un container est encore lié à l'image, il faut d'abord le supprimer, sinon le message suivant s'affiche :

```
ubuntu@ip-172-31-33-174:~$ sudo docker rmi training/sinatra
Error response from daemon: Conflict, cannot delete f0f4ab557f95 because
the container 91cdfe611f5f is using it, use -f to force
FATA[0000] Error: failed to remove one or more images
```

## 6 Lier des containers entre eux

Docker, en plus de la translation de port que nous avons décrit précédemment, dispose d'un système de lien qui permet de lier plusieurs containers ensemble et d'envoyer des informations de connexion de l'un à l'autre.

Quand des containers sont liés, des informations du container source peuvent être envoyées au container de destination. Le container de destination peut ainsi voir certaines données du container source.

## 6.1 Etablir le nommage des containers

Pour établir des liens, Docker s'appuie sur le nom des containers. De base, Docker attribue automatiquement un nom à un container lors de sa création mais il est possible de modifier le nom d'un container afin que celui-ci corresponde à l'application qu'il contient ou à une politique de nommage.

Pour nommer un container d'une manière spécifique, il suffit de spécifier l'attribut « --name » au moment de sa création, comme ci-dessous :

```
ubuntu@ip-172-31-33-174:~$ sudo docker run -d -P --name web training/webapp
python app.py
```

Nous avons ici appelé notre container « web » pour qu'il corresponde à sa fonction.

```
ubuntu@ip-172-31-33-174:~$ sudo docker ps -l
```

CONTAINER ID	IMAGE	COMMAND	CREATED
2746186ca3f1	training/webapp:latest	"python app.py"	About a minute ago
Up	About a minute	0.0.0.0:49155->5000/tcp	web

On peut voir que le container a bien été nommé « Web ». Comme le nom des containers est unique, il sera nécessaire de supprimer le si l'on souhaite nommer un autre container de la même manière.

## 6.2 Communiquer grâce aux liens

Grâce aux liens, deux containers peuvent échanger des informations de manière sécurisée puisqu'il s'agit d'un tunnel créé entre un container source et un container de destination.

Pour créer un lien, il suffit d'ajouter l'attribut « --link » lors de la création d'un container. Dans l'exemple ci-dessous, nous allons créer un container contenant une base de données. Il sera donc nommé « db » :

```
ubuntu@ip-172-31-33-174:~$ sudo docker run -d --name db training/postgres
```

On va ensuite créer un nouveau container que l'on appellera « Web » et le lier au container « db » :

```
ubuntu@ip-172-31-33-174:~$ sudo docker run -d -P --name web --link db:db
training/webapp python app.py
```

L'attribut « --link » prends la forme suivante : **--link <nom or id>:alias**

Le nom correspond au nom du container et l'alias correspond au nom du lien.

Pour vérifier le lien entre deux containers, il suffit de taper la commande suivante :

```
sudo docker inspect -f "{{ .HostConfig.Links }}" web  
[/db:/web/db]
```

La commande nous indique que le lien /db lie le container web au container db.

L'avantage de la création des liens entre les containers est qu'il n'y pas besoin de spécifier les ports du container puisqu'ils sont basés uniquement sur les noms des containers.

Bien entendu il est possible de lier plusieurs containers de destination à un container source. Typiquement, à quand fois qu'un lien est créé, cela ajoute une entrée dans le fichier « /etc/hosts » du container. Ainsi, même si le container source est redémarré, la configuration réseau du lien étant sauvegardée dans ce fichier, la communication entre les containers fonctionne de nouveau correctement.

## 7 Gérer les données au sein d'un container

Docker permet de stocker et gérer des données directement dans un container que l'on a créé. Ce sont les « Data Volumes ». Cependant, il peut être intéressant de créer un container dédié au stockage des données persistantes, ce sont les « Data Volumes Containers ».

### 7.1 Les Data Volumes

Comme dit précédemment, les data volumes sont des répertoires spécifiquement créés au sein d'un container pour stocker de la donnée. Cela peut présenter plusieurs intérêts :

- Les volumes sont initialisés à la création du container ce qui signifie que si une image de base contient déjà des données, celles-ci sont automatiquement copiées dans le nouveau container
- Les data volumes peuvent être partagées et réutilisées entre les containers
- Les changements effectués sur un data volumes sont immédiatement pris en compte
- Les changements sur un data volume ne sont pas pris en compte lors de l'update d'une image, ce qui permet de modifier l'application sans changer les données de base
- A la suppression du container, les data volumes sont conservés

### 7.2 Ajouter un Data Volume à un container

Pour créer un data volume dans un container, il faut ajouter l'attribut « -v » lors du lancement d'un « docker run » ou « docker create » :

```
ubuntu@ip-172-31-33-174:~$ sudo docker run -d -P --name web2 -v /webapp  
training/webapp python app.py
```

### 7.3 Utiliser un répertoire de l'hôte Docker comme Data Volume

Docker permet également d'utiliser un répertoire contenu sur l'hôte Docker pour le monter dans un container.

```
ubuntu@ip-172-31-33-174:~$ sudo docker run -d -P --name web2 -v /src/webapp:/opt/webapp training/webapp python app.py
```

Ici, le répertoire « /src/webapp » se trouvant sur l'hôte Docker a été monté dans le container dans le répertoire « /opt/webapp ».

Il est important de noter que si l'image du container contenait déjà un répertoire nommé « /opt/webapp », son contenu est remplacé par le contenu du répertoire se trouvant sur l'hôte.

Par défaut, les data volumes sont paramétrés en lecture-écriture mais il est possible de créer un volume en lecture seule en ajoutant le tag « :ro » comme suit :

```
ubuntu@ip-172-31-33-174:~$ sudo docker run -d -P --name web3 -v /src/webapp:/opt/webapp:ro training/webapp python app.py
```

### 7.4 Monter un fichier de l'hôte Docker dans le Data Volume

Docker peut également permettre, en plus d'un répertoire entier, de ne monter qu'un seul fichier spécifique dans le container :

```
ubuntu@ip-172-31-33-174:~$ $ sudo docker run -d -P --name web3 -v ~/.bash_history:/opt/webapp bash
```

Dans cet exemple, une console shell s'ouvre dans le container et à la fermeture du container, l'hôte Docker conservera l'historique des commandes tapées au sein de ce container.

### 7.5 Créer et monter un Data Volume Container

Lors de l'utilisation de données persistantes qu'il faudrait partager entre plusieurs containers, il est intéressant de créer un Data Volume Container afin de ne perdre aucun lien lors de l'arrêt de l'un des containers.

Il faut d'abord créer un container avec un volume à partager. Lorsque le container ne fait pas tourner d'application, seule la couche PostgreSQL de l'image, commune à tous les containers, est utilisée, tout en conservant l'espace disque.

```
ubuntu@ip-172-31-33-174:~$ sudo docker create -v /dbdata --name dbdata training/postgres
```

On peut ensuite utiliser l'attribut « -volume-from » pour monter le volume « /dbdata » nouvellement créé dans un autre container.

```
ubuntu@ip-172-31-33-174:~$ sudo docker run -d --volumes-from dbdata --name db1 training/postgres
```

Toutefois, si l'image de base contenait déjà un répertoire « /dbdata » avec des données, le fait de monter le volume depuis le container nouvellement créé ne rend disponible que les données de ce container. Celles de l'image Postgre ne sont pas accessibles.

## 7.6 Suppression d'un Volume Data Container

Si l'on supprime un container qui dispose d'un volume monté, et c'est valable également pour le container initial ou les containers qui en découlent, le volume ne sera pas supprimé.

Pour supprimer un volume définitivement du disque, il faut supprimer chacun des containers faisant référence au volume avec la commande suivante :

```
ubuntu@ip-172-31-33-174:~$ sudo docker rm -v db1
```

## 7.7 Sauvegarder, restaurer ou migrer des Data Volumes

Pour sauvegarder un volume, il suffit de créer un container dans lequel on monte ce volume avec les attributs suivants :

```
ubuntu@ip-172-31-33-174:~$ sudo docker run --volumes-from dbdata -v $(pwd):/backup ubuntu tar cvf /backup/backup.tar /dbdata
tar: Removing leading `/' from member names
/dbdata/
```

Nous avons ici créé un container dans lequel on a monté le volume du container « dbdata ». Nous avons ensuite monté un répertoire « /backup » sur l'hôte Docker, puis nous avons placé le contenu du volume « dbdata » dans l'archive « backup.tar » à l'intérieur du répertoire « /backup » créé sur l'hôte.

Pour restaurer les données de ce même container, il faut d'abord créer un nouveau container :

```
ubuntu@ip-172-31-33-174:~$ sudo docker run -v /dbdata --name dbdata2 ubuntu /bin/bash
```

Il suffit ensuite de décompresser le fichier Backup dans le Data Volume du nouveau container :

```
ubuntu@ip-172-31-33-174:~$ sudo docker run --volumes-from dbdata2 -v $(pwd):/backup busybox tar xvf /backup/backup.tar
```

## 8 Installation de Owncloud via Dockerfile

Dans le cadre de la mise en place d'une application en production, il est préférable d'utiliser le système de Dockerfile. Ce système permet d'automatiser la création de l'image directement depuis le dépôt DockerHub.

Pour ce faire il faut créer un GitHub ou un Bitbucket (ici Github). Le Dockerfile devra se trouver à la racine du git. En liant le compte github, le dépôt docker ira chercher le Dockerfile et réalisera une image à partir des instructions données dans le Dockerfile. Nous ne nous attarderons pas sur la création d'un git dans ce document, vous pouvez vous référer à <https://help.github.com/>.

### 8.1 Création du dépôt sur DockerHub

<https://hub.docker.com/>

On doit tout d'abord créer un compte DockerHub. Il existe plusieurs plans dont un gratuit nous choisiront ici le gratuit permettant uniquement la création de dépôts publics, mais dans le cadre de la production d'une application interne et/ou confidentielle, on choisirait un plan payant permettant la création de dépôts privés.

#### Plans and Pricing

The Docker Hub Registry is free to use for public repositories. Plans with private repositories are available in different sizes. All plans allow collaboration with unlimited people.

Free	Micro	Small	Medium	Large
<b>\$0/mo</b>	<b>\$7/mo</b>	<b>\$12/mo</b>	<b>\$22/mo</b>	<b>\$50/mo</b>
<b>Unlimited</b> public repositories	<b>Unlimited</b> public repositories	<b>Unlimited</b> public repositories	<b>Unlimited</b> public repositories	<b>Unlimited</b> public repositories
<b>1</b> private repositories	<b>5</b> private repositories	<b>10</b> private repositories	<b>20</b> private repositories	<b>50</b> private repositories
Your current plan	Upgrade	Upgrade	Upgrade	Upgrade



Une fois connecté sur notre profil, on crée un dépôt public :

The screenshot shows the Owncloud user interface for a user named 'aries4'. At the top, there is a navigation bar with links for 'Browse Repos', 'Documentation', 'Community', and 'Help'. A search bar is also present. On the left, a sidebar menu includes 'Summary', 'Repositories', 'Starred', 'Manage', and 'Settings'. The main content area is titled 'Your Recently Updated Repositories' and features a card for a repository named 'owncloud' created 4 days ago. This card shows a commit message 'thanks to jchaney / owncloud', 1 fork, and 0 stars. Below this, there are sections for 'Contributed Repositories' (showing 'jbjf/test') and 'Starred Repositories' (with a link to 'Browse repositories in the Registry'). An 'Activity Feed' at the bottom shows a recent action: '+ aries4 created the repository jbjf/jbjf' 5 days ago. A red rectangle highlights the '+ Add Repository' button in the top right corner.

On choisit « Automated build ». Il s'agit maintenant de lier son compte Github ou Bitbucket.

Select the source you want to use for your Automated Build



**GitHub**

Select



**Bitbucket**

Select

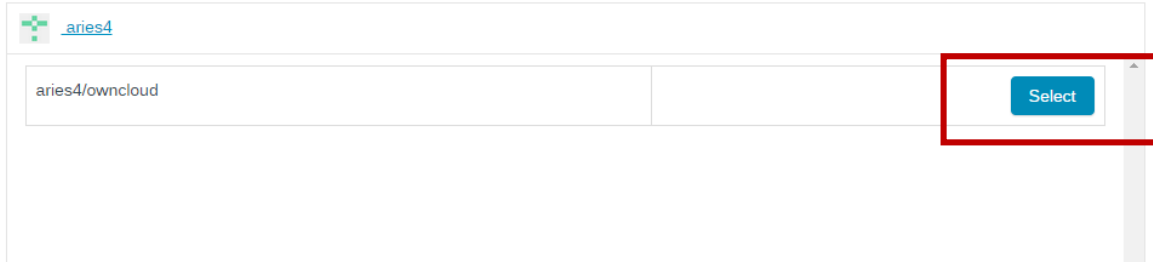
You are connected as aries4

On choisit le dépôt git :

GitHub: Add Automated Build

For more information on Automated Builds, please read the [Automated Build documentation](#).

Select a Repository to build




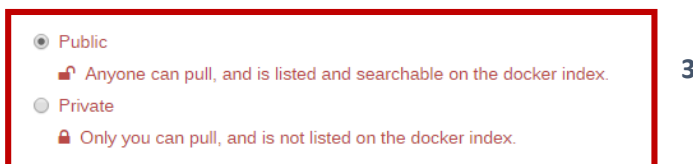
On configure son dépôt Docker :

Namespace (optional) and Repository Name



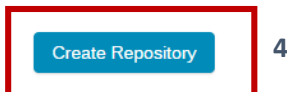
New unique Repo name; 3 - 30 characters. Only lowercase letters, digits and \_ - . characters are allowed

Tags

Active:

☒ When active we will build when new pushes occur



1. On choisit le nom de son dépôt qui sera aussi le nom de son image
2. On choisit la branche git, l'emplacement du Dockerfile et le tag assigné à son image
3. On choisit le niveau de visibilité de son dépôt Docker
4. On crée le dépôt

Dès la création/modification du Dockerfile, le dépôt Docker créera une image automatiquement. Cette image pourra être « pull » depuis le client Docker.

AUTOMATED BUILD REPOSITORY  
 aries4 / owncloud

thanks to jchaney / owncloud

Updated 4 days, 16 hours ago  
 Pull this repository `docker pull aries4/owncloud`

Repository has been deleted

Start a Build

Information	Dockerfile	Build Details	Tags
<p>owncloud 8</p> <p>Thanks to jchaney / owncloud</p> <p>Features</p> <p>ubuntu 14:04 mysql http only for test purpose only, if you want to make it live change autoconfig and implement https.</p> <p>autoconfig.php &gt; "adminlogin" =&gt; "root", "directory" =&gt; "/var/www/owncloud/data", "dbtype" =&gt; "mysql", "dbname" =&gt; "owncloud", "dbuser" =&gt; "root", "dbpass" =&gt; "", "dbhost" =&gt; "localhost", "dbtableprefix" =&gt; ""</p>			

Properties

2015-03-15 15:47:32

aries4

Settings

- Description
- Automated Build
- Webhooks
- Collaborators
- Build Triggers
- Repository Links
- Mark as unlisted

## 8.2 La création du Dockerfile

Le Dockerfile est une suite d'instructions créant une image donnée. Ces instructions sont celles que l'on donnerait lors de la création d'une image classique : apt-get, xxx.conf, etc...

Les différentes instructions sont :

- FROM : l'image de base utilisée ex : ubuntu14:04
- MAINTAINER : le nom de l'auteur
- RUN : la commande à lancer ex : apt-get update && apt-get install ping
- CMD : la commande qui sera exécutée par défaut lors du lancement du container ex : CMD ["executable","param1","param2"]
- EXPOSE : indique le ou les ports d'écoute du container
- ENV : précise certaines variables d'environnement pour le container
- ADD : ajoute un fichier/dossier externe (dans le git) et le copie à un endroit spécifique
- COPY : copie un fichier/dossier interne (dans le container) et le copie à un endroit spécifique
- ENTRYPOINT : process lancé par défaut du container, souvent suivi d'une instruction CMD ex : /bin/bash. CMD et ENTRYPOINT sont interchangeables, si on laisse le ENTRYPOINT par défaut, /bin/sh -c
- VOLUME : le volume qui sera monté par défaut depuis l'hôte
- USER : l'utilisateur qui sera utilisé pour exécuter les instructions RUN, CMD et ENTRYPOINT
- WORKDIR : le dossier par défaut pour les instructions RUN, CMD et ENTRYPOINT
- ONBUILD : définit des instructions à lancer après la construction de l'image

### 8.3 Création du Dockerfile Owncloud

```

FROM          ubuntu:14.04
MAINTAINER    aries4

ADD           bootstrap.sh /usr/bin/

RUN           sh -c "echo 'deb
http://download.opensuse.org/repositories/isv:/ownCloud:/community/xUbuntu_
14.04/ '/' >> /etc/apt/sources.list.d/owncloud.list"

RUN           apt-get update && \
              apt-get install -y --force-yes owncloud wget supervisor
RUN           cd /root
RUN           wget
              http://download.opensuse.org/repositories/isv:ownCloud:community/xUbuntu_14
              .04/Release.key
RUN           apt-key add - < Release.key

RUN           chown -R www-data:www-data /var/www/owncloud && \
              chmod +x /usr/bin/bootstrap.sh

ADD           cron.conf /etc/oc-cron.conf
RUN           crontab /etc/oc-cron.conf

EXPOSE        80
EXPOSE        443

RUN           mkdir -p /var/log/supervisor
ADD           supervisord.conf /etc/supervisor/conf.d/supervisord.conf
ADD           autoconfig.php /var/www/owncloud/config/autoconfig.php
CMD           ["/bin/bash", "-e", "/usr/bin/bootstrap.sh"]

```

Ce Dockerfile est basé sur Ubuntu14.04, on ajoute certains fichiers, on installe les paquets wget, supervisor, owncloud, on change certaines permissions pour assurer le bon fonctionnement de owncloud, on expose le port 80 et 443 et on définit la commande à lancer par défaut.

Une fois le Dockerfile placé dans le git ainsi que tous les fichiers nécessaires, le dépôt Docker va automatiquement créer une image qui pourra être « pull » depuis le client Docker.

## 9 Test de l'image Owncloud

### 9.1 Recherche de l'image owncloud

```
[centos@ip-172-31-32-124 ~]$ sudo docker search aries4/owncloud
```

NAME	DESCRIPTION	STARS	OFFICIAL
AUTOMATED			
aries4/owncloud	thanks to jchaney / owncloud	0	

[OK]

### 9.2 Téléchargement de l'image Owncloud

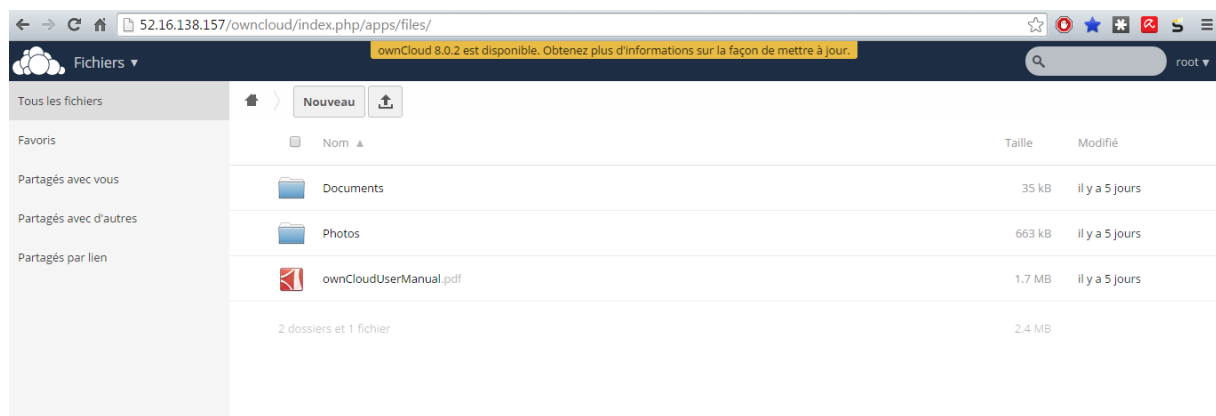
```
[centos@ip-172-31-32-124 ~]$ sudo docker pull aries4/owncloud
Pulling repository aries4/owncloud
042a34510770: Download complete
511136ea3c5a: Download complete
f0dde87450ec: Download complete
76b658ecb564: Download complete
4faa69f72743: Download complete
2103b00b3fdf: Download complete
86ff76a3e439: Download complete
7a431b273e40: Download complete
472ad17159ad: Download complete
323140be58b0: Download complete
214db90a03f4: Download complete
0f24131e820e: Download complete
3335e32c8506: Download complete
bb75884b207a: Download complete
67f40b90c9b8: Download complete
42048c49bc8e: Download complete
03675d7a920f: Download complete
24b1a2af8b66: Download complete
d3f6399d656c: Download complete
2c9660016c5c: Download complete
c22af3982fad: Download complete
Status: Downloaded newer image for aries4/owncloud:latest
```

### 9.3 Création d'un container avec l'image owncloud

```
[centos@ip-172-31-32-124 ~]$ sudo docker run -p 80:80 -v
/home/ubuntu/files:/var/www/owncloud/data -d aries4/owncloud
/usr/lib/python2.7/dist-packages/supervisor/options.py:295: UserWarning:
Supervisord is running as root and it is searching for its configuration
file in default locations (including its current working directory); you
probably want to specify a "-c" argument specifying an absolute path to a
configuration file for improved security.
  'Supervisord is running as root and it is searching '
2015-03-18 21:35:31,111 CRIT Supervisor running as root (no user in config
file)
2015-03-18 21:35:31,111 WARN Included extra file
"/etc/supervisor/conf.d/supervisord.conf" during parsing
2015-03-18 21:35:31,136 INFO RPC interface 'supervisor' initialized
2015-03-18 21:35:31,136 CRIT Server 'unix_http_server' running without any
HTTP authentication checking
2015-03-18 21:35:31,136 INFO supervisord started with pid 6
2015-03-18 21:35:32,138 INFO spawned: 'httpd' with pid 9
2015-03-18 21:35:32,139 INFO spawned: 'mysql' with pid 10
2015-03-18 21:35:33,626 INFO success: httpd entered RUNNING state, process
has stayed up for > than 1 seconds (startsecs)
2015-03-18 21:35:33,626 INFO success: mysql entered RUNNING state, process
has stayed up for > than 1 seconds (startsecs)
```

### 9.4 Vérification du fonctionnement du container

```
[centos@ip-172-31-32-124 ~]$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS            PORTS              NAMES
4b27ae422534       aries4/owncloud:latest  "/bin/bash -e /usr/b  5
minutes ago       Up 5 minutes       443/tcp, 80/tcp    sharp_wozniak
```



## 9.5 Arrêt du container

```
[centos@ip-172-31-32-124 ~]$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS            PORTS              NAMES
4b27ae422534       aries4/owncloud:latest  "/bin/bash -e /usr/b    5
minutes ago        Up 5 minutes       443/tcp, 80/tcp        sharp_wozniak
[centos@ip-172-31-32-124 ~]$ sudo docker stop sharp_wozniak
sharp_wozniak
```

## 9.6 Suppression du container

```
[centos@ip-172-31-32-124 ~]$ sudo docker rm sharp_wozniak
sharp_wozniak
```

## 9.7 Suppression de l'image aries4/owncloud

```
[centos@ip-172-31-32-124 ~]$ sudo docker rmi aries4/owncloud
Untagged: aries4/owncloud:latest
Deleted: 042a34510770330debe8f634c909b127aea18b7652a25c59c21d8a559d9ddbf8
Deleted: c22af3982fad08f173e792406fef95679091a67aed994804a485b38af0a184f2
Deleted: 2c9660016c5c8de0bf4914e9fd328e26593c1a748b5957699ed8db7dccb27ba9
Deleted: d3f6399d656c38666c656e9178808314af43b2888d2813bc6205064c1d60e7ba
Deleted: 24b1a2af8b6695c8c40cdcd5ad973a1aea614d82078b3ad7b90815bb40f2a67d
Deleted: 03675d7a920fab1f9629f60ed8f14d580b623a29e9d6346c283e5028817b184d
Deleted: 42048c49bc8e8f0fde66a9075d09edfc976c0b7e203f6f235ec0ccd563bd6464
Deleted: 67f40b90c9b890c956e43c4d4c6b2af57aec13ca337f076f6f439f770a5f34fb
Deleted: bb75884b207ab1750e5dd401295b5fe4ab7a9a901b95a598259a8a698fc70b56
Deleted: 3335e32c85063f58688fa89d6f99081629c80600af1ca7f37298f257ee5bc2d0
Deleted: 0f24131e820e6759e0b8645dbcaa4e74fbd6d59dc8670c7803d21a3a07262a2b
Deleted: 214db90a03f42793d588614da26fbeb6246c09cf211ce796fb355021a3a70308
Deleted: 323140be58b05e779533a467fefe31d6c9c8a7f377e2f94308242a017a229342
Deleted: 472ad17159ad6918355d2f97b868f7ad93ba70d674f02146c034645c10edb023
Deleted: 7a431b273e40ebba199b529e9f2855329bca826753a63eca686568f5903bd7b
Deleted: 86ff76a3e4396bf3dfebd1aa13504e9791bab043526681d61122b879bf0565ea
Deleted: 2103b00b3fdf1d26a86aded36ae73c1c425def0f779a6e69073b3b77377df348
Deleted: 4faa69f72743ce3a18508e840ff84598952fc05bd1de5fd54c6bc0f8ca835884
Deleted: 76b658ecb5644a4aca23b35de695803ad2e223da087d4f8015016021bd970169
Deleted: f0dde87450ec8236a64aebd3e8b499fe2772fca5e837ecbfa97bd8ae380c605e
```