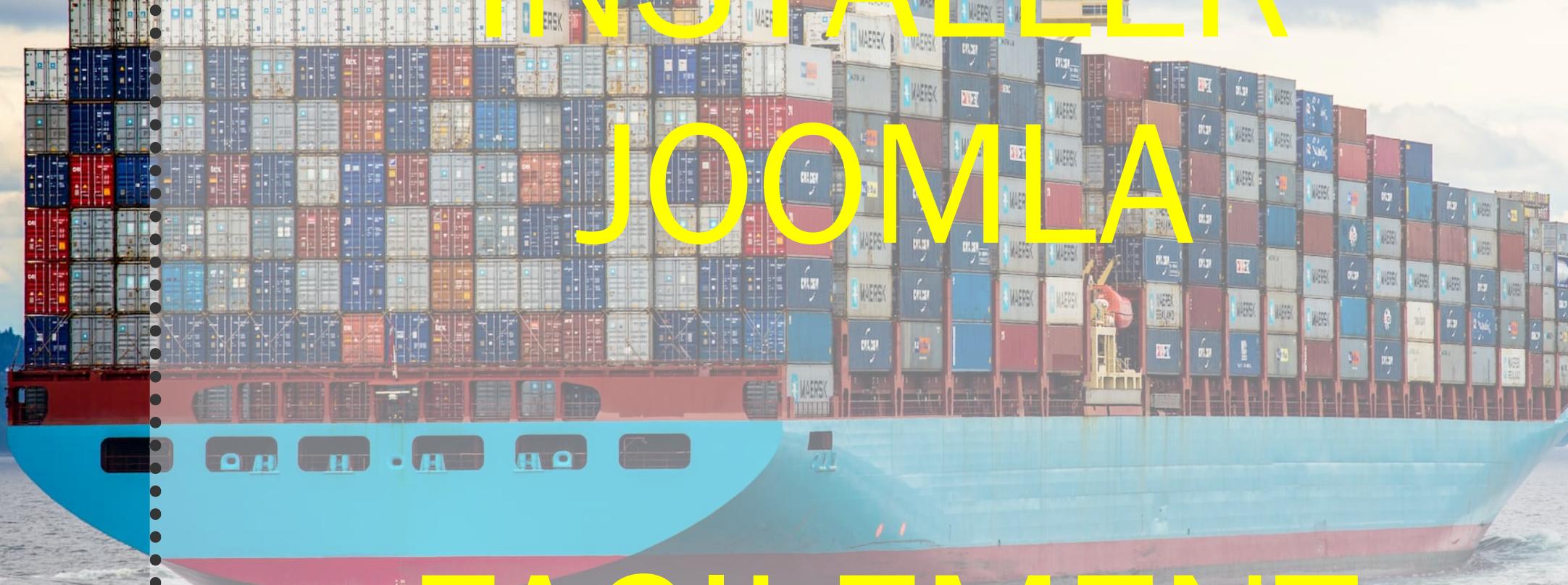


THE
JOOMLA



ETAPE 0 - INSTALLATION DE DOCKER

L'installation de Docker est gratuite pour un usage personnel ou dans le cadre d'une petite structure :

[https://www.docker.com/products/personal/.](https://www.docker.com/products/personal/)

Vous pouvez l'installer sur Linux, Mac ou Windows.

La façon la plus simple est d'utiliser Docker Desktop.

ÉTAPE 1 - AU DÉBUT ÉTAIT DOCKER ...

Prenons le temps de découvrir Docker... Grâce à lui, il n'est plus nécessaire d'installer PHP ou Apache pour faire tourner un site web.

On retrouve sur [Docker Hub](#) un très grand nombre "d'images" qui permettent d'exécuter des logiciels comme PHP, PHP+Apache, MySQL et bien, bien d'autre choses.

Toutes ces images sont totalement gratuites; elles peuvent être publiques ou privées. On peut créer ses propres images et les stocker sur Docker Hub gratuitement.

Nous allons utiliser les images PHP disponibles sur
https://hub.docker.com/_/php

Durant cette première étape, de découverte, nous allons exécuter un simple script PHP afin de montrer comment ... ne pas installer PHP et Apache.

Nous changerons ensuite la version de PHP de 7.4 vers 8.1 en juste quelques ... touches enfoncées au clavier.



Nous aurons besoin d'une image Docker qui inclus PHP et Apache pour faire tourner notre script. Par chance, une telle image existe :-)



Nous utiliserons les instructions `docker run`.

Les commandes que nous utiliserons lors cette étape seront les mêmes que vous soyez sous Linux, Mac ou Windows.

En exécutant l'instruction ci-dessous nous allons télécharger PHP
7.4.29 ainsi qu'Apache et démarrer l'image:

```
docker run --detach --name step_1_1a -p 80:80 php:7.4.29-apache
```

Tip: Lors des exécutions ultérieures, l'image PHP étant déjà présente, elle



Volumes

LOCAL

REMOTE REPOSITORIES

Dev Environments PREVIEW Search In Use only

NAME ↑

TAG

IMAGE ID

CREATED

SIZE

php

IN USE

7.4.29-apache

daa6a70e590e

17 days ago

452.28 MB

Nous pouvons constater que l'image PHP est maintenant présente dans Docker Desktop.

On obtient aussi l'information en ligne de commande : `docker`

`image list`

 Volumes

 Dev Environments PREVIEW

Nous voyons aussi qu'une application (un `container` en terme Docker) est également en cours d'exécution.

En ligne de commande : `docker container list`

apache

- `--detach` : par défaut, `docker run` exécute le container et

You don't have permission to access this resource.

Apache/2.4.53 (Debian) Server at 127.0.0.1 Port 80

Tentons d'accéder au site local : <http://127.0.0.1:80>.

Il fonctionne mais n'affiche rien puisque nous n'avons encore rien mis en place.

Le site est fonctionnel, Apache est prêt mais il n'y a pas de fichier index.php; ajoutons-en un.

Avec l'instruction ci-dessous, on peut lancer une console Linux et afficher le contenu de l'image "comme si" c'était un dossier sur notre disque dur :

```
docker exec -it step_1_1a /bin/bash
```

Une fois dans la console, créons rapidement un fichier `index.php` et quittons la console; nous n'en aurons plus besoin.

```
echo "<?php" > index.php
```

| | |
|---|--|
| System | Linux b320ef3da6cc 5.4.72-microsoft-standard-WSL2 #1 SMP Wed Oct 28 23:40:43 UTC 2020 x86_64 |
| Build Date | Apr 20 2022 12:13:36 |
| Configure Command | <pre>--configure --build=x86_64-linux-gnu --with-config-file-path=/usr/local/etc/php --with-config-file-scan-dir=/usr/local/etc/php/conf.d --enable-option-checking=fatal --with-mhash --with-pic --enable-ftp --enable-mbstring --enable-mysqlind --with-password-argon2 --with-sodium=shared --with-pdo-sqlite=/usr --with-sqlite3=/usr --with-curl --with-iconv --with-openssl --with-readline --with-zlib --disable-phpdbg --with-pear</pre> |
| Server API | Apache 2.0 Handler |
| Virtual Directory Support | disabled |
| Configuration File (php.ini) Path | /usr/local/etc/php |
| Loaded Configuration File | (none) |
| Scan this dir for additional .ini files | /usr/local/etc/php/conf.d |

Bingo ! Notre première instance Docker exécutant un script PHP !

Reprennons l'instruction que nous avions utilisé :

```
docker run --detach --name step_1_1a -p 80:80 php:7.4.29-apache
```

On voit donc qu'on cible la version 7.4.29 de PHP. En se rendant sur la page https://hub.docker.com/_/php?tab=tags et en cherchant des images de type -apache, on retrouve p.ex. les versions `php:8.1.1-apache` ou encore `php:8.1.5-apache`.

Changeons **7.4.29** par **8.1.5** et pour l'exemple, utilisons un autre port (nous utiliserons le port **801** cette fois).

```
docker run --detach --name step_1_1b -p 801:80 php:8.1.5-apache
```

```
docker exec -it step_1_1b /bin/bash
```

```
echo "<?php" > index.php
echo "phpinfo();" >> index.php
exit
```

| | |
|-----------------------------------|--|
| System | Linux 1cc61753e463 5.4.72-microsoft-standard-WSL2 #1 SMP Wed Oct 28 23:40:43 UTC 2020 x86_64 |
| Build Date | Apr 20 2022 11:23:11 |
| Build System | Linux 553c541b4bda 5.10.0-13-cloud-amd64 #1 SMP Debian 5.10.106-1 (2022-03-17) x86_64 GNU/Linux |
| Configure Command | './configure' '--build=x86_64-linux-gnu' '--with-config-file-path=/usr/local/etc/php' '--with-config-file-scan-dir=/etc/php' '--enable-fpm' '--enable-mbstring' '--enable-mysqlind' '--with-password-argon2' '--with-sodium=shared' '--with-pdo-sqlite=/usr' '--with-sqlite3=/usr' '--with-pdo-psql' '--with-readline' '--with-zlib' '--disable-phpdbg' '--with-pear' 'with-libdir=lib/x86_64-linux-gnu' '--disable-cgi' '--with-apxs2' 'build_alias=x86_64-linux-gnu' |
| Server API | Apache 2.0 Handler |
| Virtual Directory Support | disabled |
| Configuration File (php.ini) Path | /usr/local/etc/php |
| Loaded Configuration File | (none) |
| Additional .ini files read | |

Sans aucune prise de tête et zéro conflit !!! Nous avons installé une nouvelle version de PHP.

À la fin de ce chapitre, nous venons d'apprendre:

- à utiliser Docker,
- à définir la version de PHP que nous souhaitons utiliser,
- à nommer nos containers,
- à définir des ports différents selon nos containers.

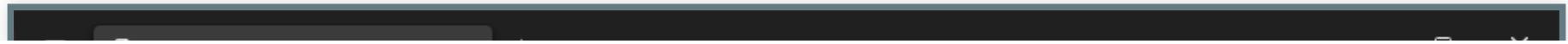
Passons maintenant à la vitesse supérieure et synchronisons les

*Soyez certain d'être dans le sous-dossier
step_1_php/2_volume pour exécuter les exemples
fournis.*

Afin de synchroniser un fichier de notre disque dur avec le container,
utilisons un volume:

```
docker run --detach --name step_1_2 -p 81:80 -v $(pwd) :/var/www/html php:8.1.5-apache
```

Si vous êtes sous Windows (MS DOS), remplacez `$ (pwd)` par `%CD%`



Explication des nouveaux arguments utilisés dans notre commande

```
docker run --detach --name step_1_2 -p 81:80 -v  
$(pwd) :/var/www/html php:8.1.5-apache
```

- `--name step_1_2` : par clarté, nous utilisons un autre nom,

Retournons dans notre navigateur et appuyons sur la touche F5. Le changement est immédiat.



À la fin de ce chapitre, nous avons appris, en plus:

- à synchroniser un dossier (et ses sous-dossiers) de notre ordinateur avec le container.

Dans la troisième étape de ce chapitre, nous allons garantir que les fichiers qui seraient créés depuis Docker le soient avec les bonnes permissions (`user:group` et `chmod`).

ETAPPE 1.3 GESTION DES PERMISSIONS LINUX

Ce chapitre ne s'applique pas aux utilisateurs qui travaillent sous Windows (mais bien si vous travaillez sous WSL).

votre machine host).

Contrairement à Windows, le système des permissions Linux est plus strict avec une notion de groupe-utilisateur-reste_du_monde (le fameux `chmod`).

Imaginons que le script PHP qui s'exécute dans Docker serait le suivant, càd créer un fichier dans le dossier courant :

```
$filename=__DIR__."/maintenant.txt";
```

Exécutons un nouveau container avec l'instruction ci-dessous :

```
docker run --detach --name step_1_3_1 -p 82:80 -v $(pwd):/var/www/html php:7.4.29-
```

Accédons maintenant à l'URL <http://127.0.0.1:82/>. Nous verrons que nous avons une erreur.

Bonjour Bruxelles !, je suis un script s'exécutant depuis Docker

Bonjour, nous sommes le 09-05-22 21:01:04

Warning: file_put_contents(/var/www/html/maintenant.txt): failed to open stream: Permission denied in
/var/www/html/index.php on line 26

Erreur, impossible de créer le fichier /var/www/html/maintenant.txt

Docker (utilisateur `root : root`) n'a donc pas réussi à créer un fichier sur notre disque dur; il faut donc lui indiquer qu'il faut utiliser notre utilisateur local.

Modifions notre instruction et analysons les changements :

```
docker run --detach --name step_1_3_2 -p 83:80 -u $UID:$GID -v $(pwd):/var/www/html
```

- `--name step_1_3`: comme d'habitude, utilisons un nom

Bonjour Bruxelles !, je suis un script s'exécutant depuis Docker

Bonjour, nous sommes le 07-05-22 22:05:50

Le fichier </var/www/html/maintenant.txt> a été créé

Cette fois-ci, avec l'URL <http://127.0.0.1:82/>, notre fichier est créé sans souci.

On peut voir le fichier maintenant.txt dans notre dossier.

Si on tape l'instruction ls -l dans notre console Linux, on constate qu'en effet le fichier maintenant.txt, qui a été créé par Docker, utilise bien notre utilisateur local; exactement ce qu'on souhaitait.

```
drwxr-xr-x 2 christophe christophe 4096 May  7 22:06 images
-rw-r--r-- 1 christophe christophe 1112 May  7 21:53 index.php
-rw-r--r-- 1 christophe christophe    33 May  7 22:05 maintenant.txt
-rw-r--r-- 1 christophe christophe 1843 May  7 22:05 readme.md
```

À la fin de ce chapitre, nous avons appris, en plus:

- à spécifier les permissions sur Docker doit utiliser lors de l'accès au système de fichiers.

Une dernière astuce pour la route : l'utilisation d'images Alpine. Suite à la prochaine étape.

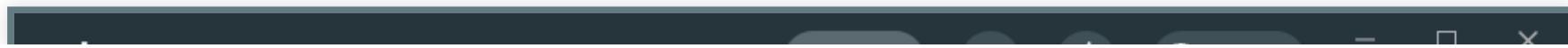
TYPE ALPINE

Sous Linux, une image Alpine est synonyme d'ultra-légère. Le maximum a été retiré et juste l'essentiel est conservé.

Nous avons jusqu'à présent utilisé une image officielle maintenue par la communauté Docker https://hub.docker.com/_/php. Ces images sont sécurisées car validées par la communauté.

Il n'existe pas toujours une image alpine officielle (ce n'est pas le cas pour une image PHP incluant Apache).

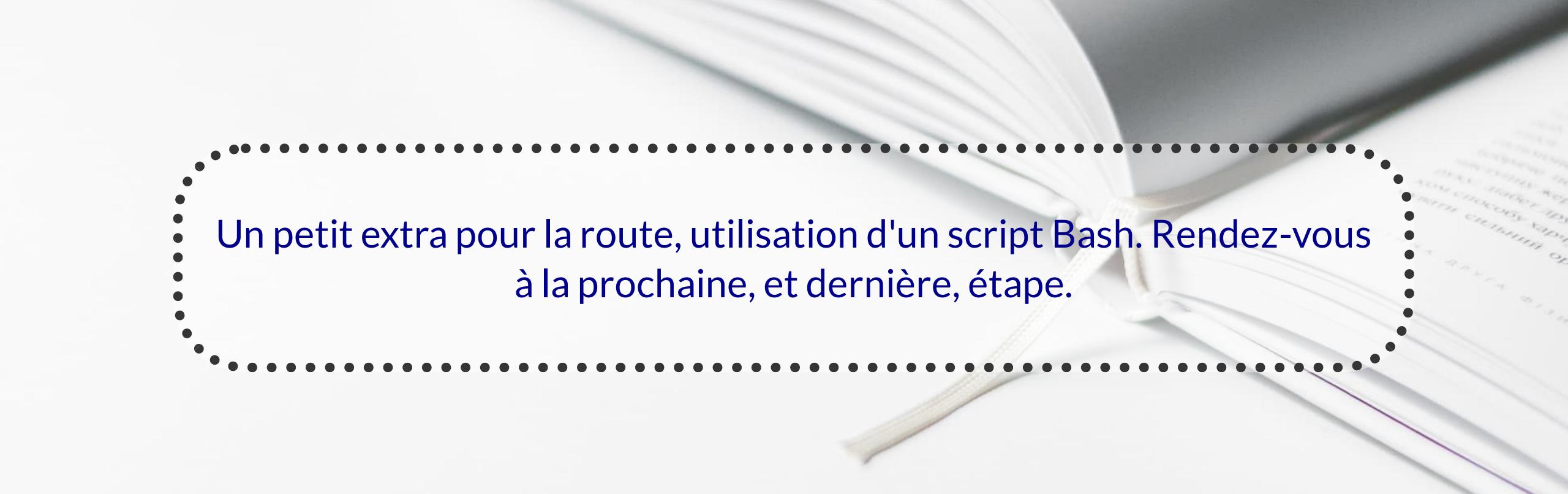
Si on cherche sur Docker Hub, il est toutefois possible d'en trouver comme [eriksoderblom/alpine-apache-php](#) qui propose PHP 8 + Apache en mode hyper léger.



L'intérêt de ce type d'images est donc de réduire l'occupation mémoire de votre ordinateur et de vous permettre de lancer plusieurs images sans trop de ralentissement.

Cela ne fonctionne que si et seulement si vos besoins sont hyper simples. Vous souhaitez p.ex. installer quelque chose avec `apt-get install`? Impossible car le gestionnaire de paquets `apt` n'est pas

installé. Vous avez vraiment le strict minimum et vous aurez à tout



Un petit extra pour la route, utilisation d'un script Bash. Rendez-vous
à la prochaine, et dernière, étape.

*Ce chapitre, optionnel, propose un script Bash qui nécessite Linux pour s'exécuter. Si vous êtes sous Windows (MS DOS), vous ne pourrez pas l'utiliser.
Mais bien si vous êtes sous WSL.*

Au tout début du chapitre, nous avons vu la commande suivante :

```
docker run --detach --name step_1_1a -p 80:80 php:7.4.29-apache
```

Imaginons que nous souhaitons relancer la même commande, une

Docker nous dit que nous avons déjà un container nommé `step_1_1` précédemment créé. Si nous changeons le nom en, p.ex. `step_1_1_bis`, nous aurons un autre souci :

```
docker: Error response from daemon: driver failed programming external connectivit
```

En effet, nous disons que `step_1_1_bis` doit utiliser le port `80` mais ce dernier est déjà utilisé (par le container `step_1_1a`), il nous

faudrait alors aussi changer le numéro du port (comme nous l'avons

Le script `docker-up.sh` proposé dans ce chapitre permet de vérifier si le container est déjà en cours d'exécution. Si c'est le cas, il sera d'abord arrêté proprement puis supprimé. Et seulement ensuite récréé.

Le script permet donc de simplifier l'exécution des commandes `docker run` comme vues lors des précédents chapitres de cette première étape de découverte de Docker.

À la fin de cette dernière étape du premier chapitre de découverte, nous avons appris, en plus, à utiliser un script qui va arrêter, supprimer et relancer un container. Facile et propre.

Le script permet aussi de spécifier certaines variables comme p.ex. la version de PHP à utiliser.



ÉTAPE 2 - ... JOOMLA VINT
ENSUITE

ÉTAPE 2.1 - INSTALLONS JOOMLA

Lors de la première étape, nous n'avions pas besoin d'une base de données, juste de PHP et d'Apache. Et comme il existe une image Docker qui reprend et PHP et Apache, c'était facile. Un coup de `docker run` et tout roule.

Lorsqu'on a besoin de plusieurs services, une commande `docker`

Lorsqu'on a besoin de plusieurs services, il faut un fichier `docker-compose.yml` à la racine du projet. On y définit la liste des services dont on a besoin.

Vous trouverez un exemple du fichier `docker-compose.yml` sur la page de description de l'image Joomla :

<https://hub.docker.com/> /joomla recherchez `docker-compose` sur

```
version: '3.1'

services:
  joomladb:
    image: mysql:5.6
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: example

  joomla:
    image: joomla
    restart: always
    links:
      - joomladb:mysql
```

Créons ce fichier dans un dossier de votre disque dur.

Ouvrez une console et rendez-vous dans ce dossier.

Ceci fait, lancez la commande `docker compose up --detach`.

Docker va commencer à télécharger `joomla` et `joomladb`, les deux services mentionnés dans le fichier `docker-compose.yml`.



Volumes

LOCAL

REMOTE REPOSITORIES



Dev Environments

PREVIEW

 Search In Use only

| NAME ↑ | TAG | IMAGE ID | CREATED | SIZE |
|--------|---------------------|----------|--------------|---------------------------|
| joomla | IN USE | latest | 9d238aff0b46 | 18 days ago 533.95 MB |
| mysql | IN USE | 5.6 | dd3b2a5dcb48 | 5 months ago 302.53 MB |

Si on regarde la liste des images disponibles dans Docker Desktop,
nous voyons que nous avons maintenant, `joomla` et `mysql`.

On obtient aussi l'information en ligne de commande : `docker`

`image list`

 Volumes

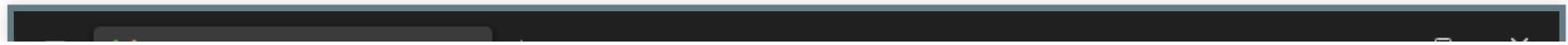
 Dev Environments **PREVIEW**

 step_2_install_joomla-joomladb-1 mysql:5.6
RUNNING

 step_2_install_joomla-joomla-1 joomla
RUNNING PORT: 80

On retrouve aussi notre application qui, cette fois, contient deux containers en cours d'exécution.

En ligne de commande : `docker container list`



Reprennons le fichier `docker-compose.yml` mais partie par partie:

```
services:  
    joomladb:  
        image: mysql:5.6  
        restart: always  
        environment:  
            MYSQL_ROOT_PASSWORD: example
```

On va définir une couche base de données qui va faire tourner

```
links:  
  - joomladb:mysql  
ports:  
  - 80:80  
environment:  
  JOOMLA_DB_HOST: joomladb  
  JOOMLA_DB_PASSWORD: example
```

On définit ici la couche application (Joomla) sans préciser de version. Ce sera donc la dernière version `stable` de ce moment-là. On dit à Joomla que le serveur de base de données associé sera `joomladb`

qu'on a défini plus haut, on indique que le port sera le port `80` et on

Ce n'est pas une bonne idée de ne pas indiquer le numéro de version.

Lorsque Docker voit `image: joomla`, il comprend `image: joomla:stable`. Il va donc chercher à télécharger la dernière version stable. Vous ne savez donc pas, à l'avance, si vous télécharger Joomla `4.1.2` ou `4.1.3` ou ...

Il est toujours préférable de spécifier la version comme p.ex. `image: joomla:4.1.2`. Vous éviterez des surprises.

Voir https://hub.docker.com/_/joomla?tab=tags pour la liste des

Les valeurs se retrouvent dans le fichier `docker-compose.yml`.

| Variable | Valeur |
|----------------------|---|
| Nom de l'hôte | <code>joomladb</code> qui est le nom qu'on a donné au service base de données |
| Nom de l'utilisateur | <code>root</code> (utilisateur par défaut pour MySQL) |
| Mot de passe | <code>exemple</code> |

Configuration de la base de données

Sélectionnez le type de base de données à utiliser *

MySQLi



Entrez le nom d'hôte, généralement "localhost" ou un nom fourni par votre hébergeur. *

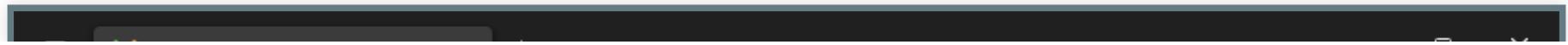
joomladb

Nom d'utilisateur créé par vous même ou fourni par l'hébergeur. *

root

Saisissez le mot de passe d'accès à la base de données





Nous venons d'installer Joomla et MySQL. Si on exécute `ls -l` (ou `dir` sous DOS), on constate quelque chose qui pourrait être surprenant : nous n'avons aucun des fichiers de Joomla sur notre machine; rien du tout.

Comme nous l'avons vu lors de l'étape 1 d'introduction; tout se passe au sein du container. Comme si c'était une boîte noire. Les fichiers ont été installés dans le container, pas sur notre ordinateur.

Si on supprime le container Docker, nous perdons l'intégralité du site;

À la fin de cette étape,

- à créer un fichier `docker-compose.yml` afin de créer et associer plusieurs containers pour une application précise,
- à installer la version que nous souhaitons de Joomla,
- à installer un second service qui, ici, est MySQL,
- et à associer Joomla et MySQL

Toutefois notre site est éphémère. Apprenons à conserver nos

containers puis qu'on déplie le container en cours, on peut voir que le nom du service Apache est `step_2_install_joomla-joomla-1` (c'est-à-dire le nom du dossier en cours suivi du nom du service suivi du chiffre `1`).

On retrouve aussi le nom avec la ligne de commande `docker container list`.

Du coup `docker exec -it step_2_install_joomla-1`

Si on fait un `cat configuration.php`, on peut donc voir le fichier de configuration du site. Puisqu'il n'y a pas d'éditeur de texte dans l'image Joomla, utilisons `sed` pour remplacer la valeur offline de `false` vers `true`:

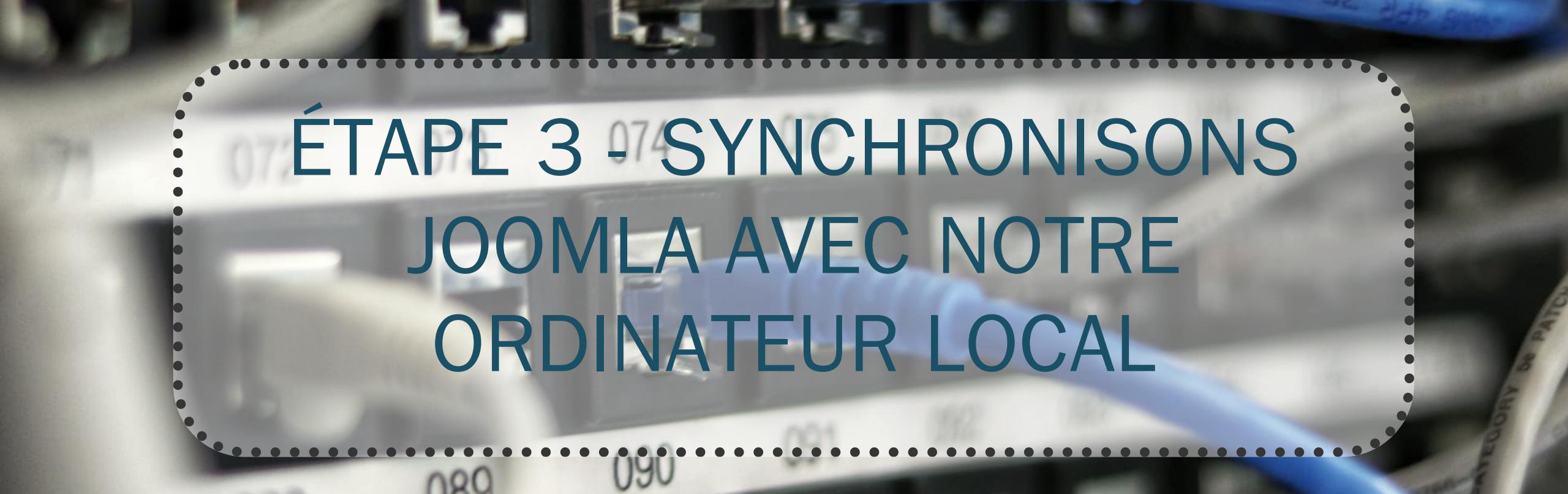
```
sed -i 's/public $offline = false/public $offline = true/g' configuration.php
```

Si on rafraîchit le navigateur, on voit bien qu'on a mis le site hors ligne. Ce qu'on voit dans la session interactive correspond bien à ce qu'on a sur la page du navigateur; nous sommes bien occupés à modifier le

À la fin de ce chapitre d'installation de Joomla, nous avons appris, en plus :

- à adapter un fichier de Joomla en démarrant le container de façon interactive.

Lors du troisième chapitre, nous verrons comment synchroniser les fichiers entre le container et notre disque dur.



ÉTAPE 3 - SYNCHRONISONS JOOMLA AVEC NOTRE ORDINATEUR LOCAL

Nous venons d'installer Joomla, on a pu installer l'une ou l'autre version de Joomla mais on constate que nous n'avons rien en local : si on supprime le container, on perds tout.

Bien sûr, on peut installer une extension de sauvegarde comme Akeeba et s'amuser à prendre un backup; comme on le fait traditionnellement.

On peut aussi veiller à synchroniser notre container (la partie Joomla ainsi que la partie base de données) avec notre disque dur. Cette synchronisation fait appel à ce qu'on nomme dans le monde Docker :

On va donc quitter le site et que ce soit appris à l'écouter, on va également perdre nos données ce qui n'est pas ce que l'on souhaite.

On va vouloir "mapper" le site Joomla à notre disque dur. Quand nous avions lancé une session interactive (grâce à `docker exec -it step_2_install_joomla-joomla-1 /bin/bash`), nous avions constaté que le dossier du site était `/var/www/html`. Ce dossier est le `WORKDIR` (répertoire de travail) de l'image Joomla. On peut retrouver cette information dans la documentation de l'image, après avoir lancé une session interactive ou encore lorsqu'on fait un

service Joomla, d'ajouter la gestion des volumes. Nous allons faire correspondre le dossier `site_joomla` de notre ordinateur avec le site Joomla.

D'abord, pour éviter tout problème de droits d'accès, veuillez créer le dossier `site_joomla` vous même.

```
mkdir -p site_joomla
```

Adaptions le fichier `docker-compose.yml` et ajoutons:

```
volumes:
```

La lecture est peut-être plus aisée de droite à gauche : on va faire correspondre le dossier `/var/www/html` qui se trouve dans le container Docker avec le dossier `site_joomla` se trouvant dans notre répertoire actuel; sur notre machine (=sur notre host).

Si on relance la commande `docker compose up --detach`; on pourra constater qu'on aura bien, lors de la création de l'image, les fichiers de Joomla qui seront synchronisés avec notre disque dur.

Tout comme nous l'avons vu précédemment, les fichiers / dossiers créés depuis Docker ne le sont pas avec notre utilisateur actif mais celui défini dans l'image. Pour PHP, nous l'avons vu, c'était l'utilisateur `root`.

Pour Joomla, c'est `www-data` et on le voit lorsqu'on fait un `ls -al`. Il nous faut, ici aussi, changer cela pour utiliser notre utilisateur local.

Tout d'abord supprimons le précédent dossier `site_joomla` puis

notion de l'utilisateur mais pour cela il nous faudra deux valeurs, le `user id` et le `group id`.

Sous Linux, on peut retrouver l'ID de son utilisateur et de son groupe comme ceci: `echo "Votre UID est UID et votre GID est GID"`. Nous avons nos valeurs. Adaptons le fichier:

```
user: "1000:1000"
```

Relançons `docker compose up --detach` et, maintenant, les

ÉTAPE 3.2 - AJOUT D'UNE IMAGE

Si l'on ajoute une image depuis le gestionnaire des médias, on constatera que l'image s'ajoute bien dans le dossier

`./site_web/media`, comme attendu.

ÉTAPE 3.3 - INSTALLATION D'AKEEEBA BACKUP ET SAUVEGARDE

L'installation d'une extension ainsi que son utilisation n'est en rien différente. On peut installer Akeeba depuis le web puis lancer une sauvegarde. Si on utilise le profil par défaut, le `jpa` est sauvé dans le dossier

```
./site_joomla/administrator/com_akeebabackup/backup;
```

À la fin de ce chapitre, nous avons appris :

- synchroniser les fichiers de Joomla avec notre disque dur.

Si nous supprimons le container Joomla, nous n'allons plus perdre les fichiers de notre site web. Mais nous perdrions bien la base de données puisqu'elle n'est pas encore synchronisée localement. C'est ce que nous allons apprendre dans le prochain et dernier chapitre.



ÉTAPE 4 - SYNCHRONISONS MYSQL AVEC NOTRE ORDINATEUR LOCAL

Nous venons de voir comment conserver les fichiers de Joomla sur son disque dur. Ainsi, si on supprime le container Docker, on ne perds pas nos fichiers.

Oui mais ? On perds la base de données puisque nous n'avons pas utilisé de volume pour, la même raison, garder trace de notre base de données si on supprime le container.

Lire "Where to Store Data" sur https://hub.docker.com/_/mysql

Comme nous l'avions fait pour Joomla lorsqu'on a créé un dossier `site_joomla`, il nous faudra créer un dossier `db` manuellement afin qu'on n'ait pas de souci de droits d'accès.

Ceci fait, il faut adapter le fichier `docker-compose.yml` et, pour le service MySQL cette fois, d'ajouter la gestion des *volumes*. Ajoutons directement le bon utilisateur:

D'abord, pour éviter tout problème de droits d'accès, veuillez créer le dossier `db` vous même.

```
mkdir -p db
```

Adaptons le fichier `docker-compose.yml` et ajoutons:

```
user: "1000:1000"
volumes:
- ./db:/var/lib/mysql
```

Pourquoi le dossier `/var/lib/mysql`? Car c'est celui qui est référencé par MySQL comme le lieu où sont sauvegardé les données.

Lançons la commande `docker compose up --detach`.

Maintenant, si nous allons dans le dossier `./db`, nous pouvons en effet voir un dossier qui correspond à notre base de données.

Ajoutons les données d'exemples et p.ex. un nouvel utilisateur puis, depuis Docker Desktop, supprimons le container comme nous l'avons fait pour les autres exercices.

Cette fois, tuons le site web : allons dans Docker Desktop et supprimer le container qui contient notre site web. Nous n'allons pas seulement l'arrêter mais bien le supprimer.

Si nous n'avions pas fait de synchronisation (utilisation de volumes en terme Docker), nous aurions tout perdu. Mais maintenant, qu'en est-il ?

Relançons la commande `docker compose up --detach` et

Nous récupérons notre site web, base de données comprises ! Notre site est de nouveau fonctionnel, les extensions que nous avions installés sont toujours présentes, nos articles, nos utilisateurs, ... tout est à nouveau là.

Ceci parce que nous avons, cette fois, créer un volume externe (et donc persistant) et pour le site et pour la base de données.

À la fin de ce dernier chapitre, nous avons appris :

- à manipuler Docker et créer des sites web PHP / Apache,
- à installer Joomla et de créer un site web "dockerisé" (=qui tourne sous forme de container dans Docker),
- à synchroniser les fichiers et la base de données de notre site.



MERCI DE VOTRE ATTENTION !