

Docker

Introduction

Pourquoi Docker ?

- Travailler consécutivement ou simultanément sur différents projets à la fois
- Bénéficier d'environnements avec les standards PHP/apache/mysql optimisés
- Simplifier et unifier la gestion des différents environnements : poste de travail, dev, pré-production, tests, production, etc.
- Définir de l'environnement sous forme de code : embarquer la définition de l'environnement avec le package de l'application, automatiser, et donc accélérer les déploiements
- Réduire la fracture entre l'expertise du développeur et celle de l'administrateur système, dans la mouvance du DevOps

L'environnement Drupal : Un LAMP optimisé

Étant donné que Drupal 8 ne nécessite que apache/mysql/PHP pour fonctionner, les émulateurs serveur Wampserver, XAMPP pourraient sembler suffire.

En réalité, les besoins du projet et du développeur demandent souvent d'aller plus loin.

À minima :

- les configurations d'origine de PHP et Mysql doivent être élevées pour augmenter les ressources disponibles
- les lanceurs de commandes *drush* et *composer* sont nécessaires pour effectuer des tâches de reconstruction Drupal, installer de nouvelles librairies...

L'environnement Drupal : Optimisation des performances

En réalité, les projets Drupal 8 demandent souvent d'aller plus loin : Pour répondre à des objectifs de performance ou de volume de données important, il peut être requis :

- serveur web *nginx* plutôt qu'*apache*
- *PostgreSQL* plutôt que *Mysql*
- reverse proxy *Varnish*
- moteurs de cache, par ex. Memcache, Redis

Ces outils sont en interaction étroite la couche applicative. Leur configuration relève du travail du développeur. Il doit donc disposer de ces solutions dans son environnement de travail pour en garder la maîtrise.

L'environnement Drupal : Les besoins du projet

Au niveau applicatif, certaines fonctionnalités peuvent nécessiter, par exemple :

- une application de moteur de recherche sous java : *SOLR*
 - message workers (*rabbitmq*)
 - socket de temps réel (*socket.io*)
-
- l'activation et la configuration d'extensions PHP additionnelles :
 - *OpCache*
 - *mcrypt* (cryptage)
 - génération d'image : *ImageMagick*
- ...

L'environnement Drupal : Outils de développements *back-end*

- extensions PHP *Xdebug* (var dumper dans l'IDE à l'aide de points d'arrêts)
- lanceur de commandes PHP *drush*
- lanceur de commandes PHP *composer*
- lanceur de commandes PHP *drupal console* (génération de code)
- task runner PHP *phing*

L'environnement Drupal : Outils de développement *front-end*

Le développement front-end moderne tend à être fait accompagné d'utilitaires (souvent écrits en *node js*) permettant de réaliser automatiquement des tâches telles que :

- préprocess SASS
- génération de polices d'icônes
- génération de sprites
- minification de scripts et feuilles css

...

Les différentes tâches étant orchestrées par les task runner *gulp* ou *grunt*

Docker et la virtualisation

Pour l'équipe de développement d'un projet Drupal, le premier bénéfice est de proposer à l'ensemble des collaborateurs un environnement de développement identique, propre à un projet donné, qui émulera le serveur Web.

Cela a plusieurs intérêts :

- cela dispense le développeur de gérer lui-même la mise en place d'un environnement complexe
- au sein du poste de travail du développeur, **Docker** permet d'isoler les environnements indépendants entre différents projets sans créer de conflits (ex: versions de php)

Docker et la virtualisation

- évite d'observer des comportements différents ou anormaux du même site selon les environnements (*collaborateur 1, collaborateur 2, préproduction, production*)
- offre large d'*images* libres disponibles proposées par la communauté
- dans un contexte où les développements front-end et back-end se complexifient et la dépendance à divers outils et configurations se multiplient, automatiser leur mise en place devient indispensable
- définition de l'environnement sous forme de code : embarquer la définition de l'environnement avec le package de l'application, automatiser, et donc accélérer les déploiements

Docker et la virtualisation

- À noter que Docker offre d'autres atouts qui dépassent la gestion des environnements de développements et peut être utilisée en production, avec des options supérieures à ce qu'offre les solutions de virtualisation plus classiques, mais que nous ne traiterons pas ici.
- Dans la mouvance du *dev ops*, Docker et la containerisation peut-être une des briques clés dans la gestion d'une infrastructure complexe :
 - « infrastructure as code », infrastructure défini par du code plutôt que par une série d'opérations manuelles, donc naturellement plus simple à automatiser.
 - architectures micro services
 - infrastructures résilientes
 - infrastructures extensibles (*scalable*)
 - déploiement rapide et automatisé

Docker et la virtualisation

- Cet environnement, défini sous forme de code dans des fichiers Docker, doit fonctionner de la même façon sur chacun des postes de travail, que celui-ci soit sur Linux, Windows ou Mac.
- Il visera à être aussi proche que possible de l'environnement de production, toutefois adapté pour intégrer les besoins spécifique d'un environnement de développement (par ex, outils de *debug*, génération de code).

Qui fait quoi ?

Qui doit gérer Docker dans le cadre d'un projet ? :

On peut aborder Docker en utilisateur ou en configurateur.

Au sein d'une équipe, il est difficile d'envisager que tout développeur devienne un expert de la configuration Docker, dont les compétences requises sont plus proches de celles d'administrateur système que développeur.

Docker en utilisateur

- Les développeurs doivent avoir une idée du fonctionnement de Docker et d'un certain nombre de tâches d'administration générales : démarrer/stopper un container, entrer dans un container, synchroniser un volume...
- Le développeur, à défaut de créer et maintenir ses propres images, doit aussi pouvoir récupérer et utiliser les images proposées communauté.

Docker en configurateur

Il est souhaitable qu'une seule personne seulement soit responsable de créer et mettre à disposition les environnements *dockerisés* au sein de l'équipe.

- Il écrit ses propres images : les *Dockerfile*, fichiers de configurations associés, *endpoints*
- Il écrit des *docker-compose*
- Vérifier la qualité des images la communauté






Dockerhub

- Docker Hub (hub.docker.com), est un registre où sont référencées la plupart des images publiques disponibles.
- Les images sont souvent hébergées sur GitHub.
- À la demande d'une certaine image, via *docker pull*, c'est ce registre que docker ira d'abord interroger.

Dockerhub

Repositories (34072)

All

	php official	3.4K STARS	10M+ PULLS	> DETAILS
	php-zendserver official	143 STARS	1M+ PULLS	> DETAILS
	phpmyadmin/phpmyadmin public automated build	526 STARS	10M+ PULLS	> DETAILS
	phpunit/phpunit public automated build	58 STARS	500K+ PULLS	> DETAILS
	webdevops/php-apache-dev public automated build	53 STARS	10M+ PULLS	> DETAILS

Dockerhub

[Explore](#)[Help](#)[Sign up](#)[Sign in](#)

PUBLIC | AUTOMATED BUILD

phpmyadmin/phpmyadmin ☆

Last pushed: 5 days ago

[Repo Info](#)[Tags](#)[Dockerfile](#)[Build Details](#)

Short Description

A web interface for MySQL and MariaDB.

Full Description

Official phpMyAdmin Docker image

Run phpMyAdmin with Alpine, supervisor, nginx and PHP FPM.

build failing
docker pulls 20M
docker stars 526
55MB 38 layers
version latest

All following examples will bring you phpMyAdmin on `http://localhost:8080` where you can enjoy your happy MySQL administration.

Docker Pull Command



```
docker pull phpmyadmin/phpmyadmin
```

Owner



phpmyadmin

Source Repository

[phpmyadmin/docker](#)

Docker en pratique

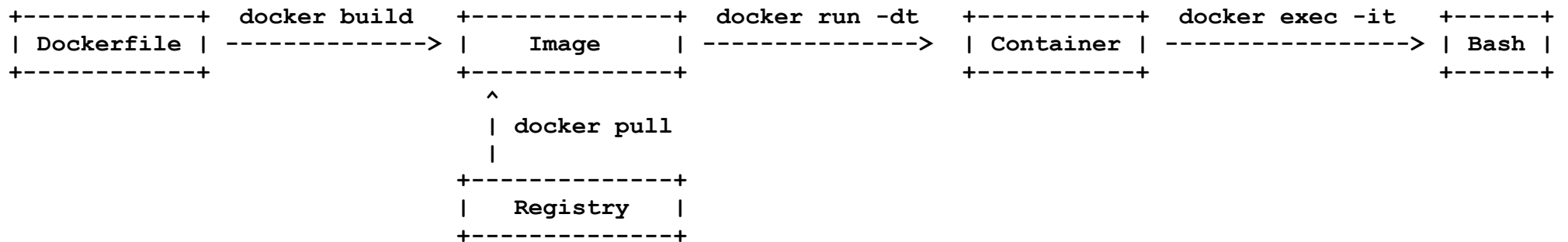
Docker sous Windows

- Télécharger l'installateur Docker Toolbox (https://docs.docker.com/toolbox/toolbox_install_windows)

Il installera :

- Docker CLI client, qui permettra de gérer images et containers
- Docker Machine pour lancer Docker Engine depuis les terminaux Windows (*Powershell* notamment)
- Docker Compose

Docker, images et containers



Commandes docker

- **docker pull**

Récupération d'une image depuis le registre vers la machine locale.

```
docker pull bitnami/apache
```

- **docker run**

Lancer un conteneur (à partir d'une image)

```
docker run bitnami/apache
```

option --name (attribution d'un nom au conteneur):

```
docker run --name apache bitnami/apache
```

option --volumes (liaison d'un volume):

```
docker run -v /path/to/app:/app bitnami/apache
```

option --ports (assignation d'un port):

```
docker run -p 8080:80 bitnami/apache
```

option --detach (lancement en tâche de fond):

```
docker run -d bitnami/apache
```

option --environment (définition d'une variable d'environnement):

```
docker run -e APACHE_HTTP_PORT_NUMBER=8080 bitnami/apache
```

option --links (ajout d'un lien à un autre container):

```
docker run -l db:mysql bitnami/apache
```

ex :

```
docker run --name apache -v /path/to/app:/app -p 8080:80 -d bitnami/apache
```

- **docker attach**

S'attacher à un conteneur

```
docker attach apache
```

Commandes docker

- **docker start/stop/restart**

Démarrer un conteneur

```
docker start apache
```

Stopper un container en fonctionnement

```
docker stop apache
```

Redémarrer un container

```
docker restart apache
```

- **docker exec**

Lancer une commande dans un conteneur

```
docker exec apache pwd
```

option -tty et --interactive (alloue un tty et gard STDIN ouvert)

```
docker exec -t apache bash
```

- **docker rm**

Effacer un conteneur

```
docker rm apache
```

- **docker ps**

Lister les containers en fonctionnement

```
docker ps
```

option --all (tous les containers existants, même stoppés) :

```
docker ps -a
```

- **docker build**

Créer une image à partir d'un Dockerfile

```
docker build -f Dockerfile
```

Commandes docker

- **docker images**

Lister toutes les images

```
docker images
```

- **docker rmi**

Effacer une image

```
docker rmi bitnami/apache
```

- **docker top**

Afficher les process en cours dans le conteneur

```
docker top apache
```

- **docker cp apache:/path/to/file**

Copie fichier et dossiers entre le conteneur et le système de fichiers local

```
docker cp apache:/path/to/folder /local_path/to/folder
```

```
docker cp apache:/path/to/file /local_path/to/file
```

- **docker logs**

Afficher les logs du conteneur

```
docker log apache
```


TTY et volumes

- Docker peut permettre l'installation et l'utilisation d'une application sans qu'aucun fichier n'apparaisse avec le poste de travail.
- Les options -t (tty) associées aux commandes *exec* ou *run* permettront d'accéder à une console (ou terminal) à l'intérieur du système de fichier du container, y naviguer et les éditer via la ligne de commande (via l'interpréteur bash ou shell le plus souvent).

```
docker exec -t apache bash
```

- Il sera souvent utile, dans le cadre de travail du développeur, de pouvoir voir les dossier et fichier les plus amenés à être modifiés sur son espace de travail et de les éditer avec son IDE, en définissant un mapping de volumes.

```
docker run -v /path/to/app:/app bitnami/apache
```

Les fichiers et dossiers du container seront alors synchronisés avec des fichiers et dossiers locaux. Cela permettra en outre d'assurer la persistance (ajout/modification de fichiers php, donnée de la base) du travail réalisé sur le projet si le conteneur doit être détruit (volontairement ou *involontairement*) et *recréé*.

Images et containers

- L'image est définie par un fichier *Dockerfile*.
- Un *Dockerfile*, bien qu'elle soit écrit dans un micro-langage spécifique, ressemble à une succession de commande script shell/bash sur un système d'exploitation vierge, toutes précédés de l'instruction *RUN*
- On peut créer ses propres images, mais on utilise souvent d'une image existante par héritage ou duplication.
- Le *Dockerfile* de l'image qu'on souhaite containeriser commence par la définition d'une autre image dont elle hérite. Cette dernière définit l'installation du système d'exploitation : par ex. Ubuntu ou Debian Stretch. C'est par cette instruction que commence le du fichier.
- Sont définies successivement toutes les instructions qu'il faut installer au niveau du serveur : apache, librairies, etc.

Images et containers

- Un *Dockerfile* est donc assez lisible pour toute personne un minimum familiarisé à l'administration de système en ligne de commande.
- Pour image sous système unix seront *apt-get*, *chmod*, *clown*, *cd*, *adduser*
- D'autres instructions existent : *COPY*, *VOLUME*, *ENV*, *ENTRYPOINT*, *CMD*. Pour connaître leur utilisation : <https://docs.docker.com/engine/reference/builder/#usage>

Principe d'isolation

- Pour être fidèle à l'esprit de *Docker*, il faut concevoir l'environnement de son application, s'il celui-ci est un minimum complexe, comme un ensemble d'images en interaction.
- Chaque image gère un composant bien précis, tout du moins un ensemble de composants centrés une logique applicative précise. En pratique, il est difficile de déterminer une logique absolue qui permette de choisir ce qui doit être englobé ou pas. Il faut prendre en compte différents principes :
- On doit chercher l'**isolation**: les images/containers doivent être en relation, mais indépendants, ce qui permettra de les reconstruire éventuellement sans en réengager la re-construction de l'ensemble.
- Si les images sont bien isolées, elle deviennent plus générales et sont susceptibles d'être réutilisées dans d'autre projets, voire devenir des images publiques.

Surcharge des images

- Une logique de surcharge permet de gérer ce qui est spécifique au projet (php.ini, my.cnf). Une image bien conçue doit permettre à son utilisateur de surcharger une série de paramètres, ceux qui sont les plus susceptibles de varier.
- Ceux-ci ne sont pas codés «en dur» dans le Dockerfile mais via des variables d'environnement externalisées dans des fichiers de configuration complémentaires.

Containers

Un container est en quelque sorte

- une instance d'une image
- une machine virtuelle créé à partir d'une image

On peut créer plusieurs container (2 par projet) à partir d'une image apache/php.

Le container est ce sur quoi on travaille, sur ce l'environnement en fonctionnement.

Il va comporter tout ce qui st spécifique au projet,

On le définit via un *docker run* ou via un fichier *docker-compose*

les fichiers du projet (drupal), définir où ceux-cid sont sur poste de travail, quel port utiliser pour requête.

Docker Compose

Docker compose

- Plutôt que de lancer un ensemble de commandes run, lancer les différents containers, leur assigner des ports, les lier entre eux, associer des volumes, etc.,
- on préférera faire faire le travail par docker-compose qui s'appuiera sur un fichier docker-compose.yml qui définira l'ensemble des règles de construction de cet ensemble multi-container.
- Ces docker-compose.yml et fichiers associés sont aussi proposées par la communauté, souvent sur github.com
- Éventuellement, un fichier `.env` permettra de définir des variables d'environnements hors du *docker-compose.yml*
- Ces variables d'environnement permettent de gérer ce qui est spécifique à l'espace de travail du développeur :
 - - un mapping des dossiers du poste de travail à synchroniser avec les volumes du container
 - - id et password mysql
 - - etc...

Docker compose

web:

```
image: drupal:8
environment:
  DRUPAL_PROFILE: standard
  DRUPAL_SITE_NAME: Drupal
  DRUPAL_USER: admin
  DRUPAL_PASS: admin
  DRUPAL_DBURL: mysql://drupal:drupal@database:3306/drupal
ports:
  - "8000:80"
volumes:
  - ../drupal/html:/var/www/html
links:
  - database:database
```

database:

```
image: mariadb:10
environment:
  MYSQL_USER: drupal
  MYSQL_PASSWORD: drupal
  MYSQL_DATABASE: drupal
  MYSQL_ROOT_PASSWORD: ''
  MYSQL_ALLOW_EMPTY_PASSWORD: 'yes'
ports:
  - "3306:3306"
```

Commandes docker-compose

`docker-compose up`

Lancer les conteneurs

`docker-compose up -d`

Lancer les conteneurs en tâche de fond

`docker-compose up apache`

Lancer un conteneur spécifique (et dépendances)

`docker-compose kill`

Arrêt des conteneurs

`docker-compose up -d --build`

Lancer les conteneurs et reconstruire les images liées à partir du Dockerfile

`docker-compose rm`

Supprimer les conteneurs

Création des containers

- créer un container à partir de l'image **crccheck/hello-world**

<https://hub.docker.com/r/crccheck/hello-world/>
et afficher « hello drupal world !!! »

- créer et associer 2 container :
web/apache et le générateur de pdf
<https://github.com/tecnospeed/pastor>
et tester la génération de pdf

Docker et Drupal

Drupal et docker

- Un projet basique sous Drupal 8 n'aie besoin que de apache/PHP/mysql pour fonctionner.
- De ce point de vue, les émulateurs serveur Wampserver, XAMPP pourraient sembler suffire.

En réalité, les besoins du projet et du développeur demandent d'aller plus loin.

À minima :

- les configurations d'origine de PHP et Mysql doivent être élevées pour augmenter les ressources disponibles
- les lanceurs de commandes *drush* et *composer* sont nécessaires pour effectuer des tâches de reconstruction Drupal, installer de nouvelle librairies...

Drupal et docker

Concernant Drupal, la communauté Docker semble avoir en majorité retenu les options suivantes :

- **une image avec *apache*, *php*, *drush* et *composer***

Le Dockerfile associé se chargera principalement, en étendant une image d'apache php/apache officielle :

- d'ajouter des package linux nécessaires ou souvent utiles à Drupal (*apt-transport-https*, *libmcrypt-dev*, *libxml2-dev*, *libxslt1-dev*, *linux-libc-dev*, *mysql-client*, etc.)
- d'ajouter les extensions php nécessaires ou souvent utiles à Drupal (*gd*, *mcrypt*, *opcache*, *pdo_mysql*, etc.)
- de personnaliser le fichier de configuration php.ini au niveau de certaines variables (ex : *memory_limit*, etc.)
- d'installer *composer* et *drush* globalement
- de personnaliser le fichier de configuration apache2.conf (ex : AllowOverride All au niveau du dossier des dossier public drupal)

Les fichiers Drupal n'étant pas présenté sur l'image, ils seront gérés sur un volume du container

Drupal et docker

- **une image mysql, sur la base d'une image mysql officielle**

Les Dockerfile se chargera principalement, à partir d'une image d'apache mysql officielle, de personnaliser le mysql.cnf au niveau de certaines variables (ex : *max_allowed_packet*, etc.)

On peut ajouter éventuellement des container/images additionnelles comme outils de dev (phpmyadmin, mailhog, xgui) ou des applications complémentaires comme elasticsearch, memcache, solr

Questions

- Regarder <https://github.com/mydropteam/docker-devbox> comment s'appelle l'image qu'utilise le container *web* et où sont les sources ?
- comment s'appelle l'image qu'utilise le container *database* et où sont les sources ?
- à quoi servent les fichiers présents dans le dossier config. Pourquoi ne sont-ils pas gérés dans les sources de l'image elle-même ?
- les *env.default* permettent de jouer avec les variables `PORT_WEB`, `SERVERNAME`, `LOCALPATH` et `DOCUMENTROOT` passées au `docker-compose.yml`. Où ces variables sont-elles exploitées ?
- comment adapter `docker-compose.yml` de manière à pouvoir gérer directement, depuis son espace de travail, le fichier *apache2.conf* ?
- Étudier <https://hub.docker.com/r/dropteam/drupal-php/~/dockerfile/>
- à quoi servent les commandes :
 - *gem* ?
 - *npm* ?
 - *docker-php-ext-configure* ?

Questions

- Regarder <https://github.com/mydropteam/docker-devbox> comment s'appelle l'image qu'utilise le container *web* et où sont les sources ?
- comment s'appelle l'image qu'utilise le container *database* et où sont les sources ?
- à quoi servent les fichiers présents dans le dossier config. Pourquoi ne sont-ils pas gérés dans les sources de l'image elle-même ?
- les *env.default* permettent de jouer avec les variables `PORT_WEB`, `SERVERNAME`, `LOCALPATH` et `DOCUMENTROOT` passées au `docker-compose.yml`. Où ces variables sont-elles exploitées ?
- comment adapter `docker-compose.yml` de manière à pouvoir gérer directement, depuis son espace de travail, le fichier *apache2.conf* ?
- Étudier <https://hub.docker.com/r/dropteam/drupal-php/~/dockerfile/>
- à quoi servent les commandes :
 - *gem* ?
 - *npm* ?
 - *docker-php-ext-configure* ?

Créer des images

Réutilisation des images

- Ces propriétés plus ou moins générales, cette image devrait pouvoir être réutilisées sur un projet similaire.
- Sur la base jeu d'héritage entre images, l'intéressant est de pouvoir des images réutilisables et d'autres plus spécifiques étendant ces premières.

Dockerfile

```
FROM bitnami/minideb-extras:jessie-r61
LABEL maintainer "Bitnami <containers@bitnami.com>"

# Install required system packages and dependencies
RUN install_packages libc6 libexpat1 libffi6 libgmp10 libgnutls-deb0-28 libhogweed2
libldap-2.4-2 libnettle4 libp11-kit0 libpcre3 libsasl2-2 libssl1.0.0 libtasn1-6
zlib1g
RUN bitnami-pkg unpack apache-2.4.33-3 --checksum
55bc664737e2ff9d534468088424ee4b04e770e67800e317c9db5b1e58cf2ffe
RUN ln -sf /opt/bitnami/apache/htdocs /app

COPY rootfs /
ENV APACHE_HTTPS_PORT_NUMBER="443" \
    APACHE_HTTP_PORT_NUMBER="80" \
    BITNAMI_APP_NAME="apache" \
    BITNAMI_IMAGE_VERSION="2.4.33-r33" \
    PATH="/opt/bitnami/apache/bin:$PATH"

EXPOSE 80 443

WORKDIR /app
ENTRYPOINT ["/app-entrypoint.sh"]
CMD ["nami", "start", "--foreground", "apache"]
```

Dockerfile : entrypoint.sh

```
#!/bin/bash
```

```
source /etc/apache2/envvars  
#tail -F /var/log/apache2/* &  
exec apache2 -D FOREGROUND
```