Docker

Une version à jour et éditable de ce livre est disponible sur Wikilivres, une bibliothèque de livres pédagogiques, à l'URL : https://fr.wikibooks.org/wiki/Docker

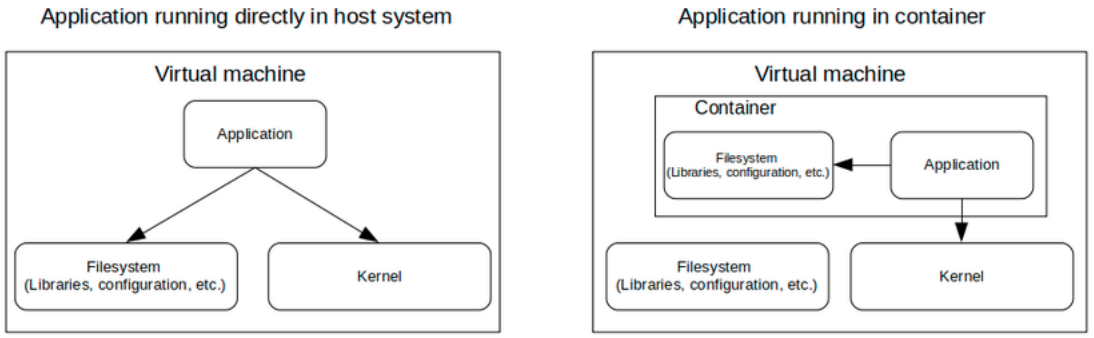
---

Sections

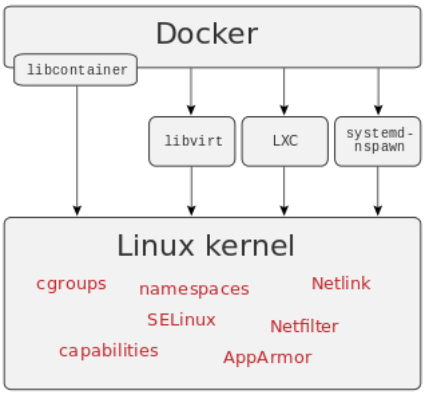
Introduction

Principe

Docker est un logiciel libre conçu pour lancer des applications dans des conteneurs logiciels. Ces conteneurs sont plus légers en ressources que les machines virtuelles car ils partagent leur noyau.



Différence entre un conteneur et une VM.



Docker sur Linux

Images et conteneurs

Les conteneurs sont construis à partir d'images qui partages leur couches en différentiel[1].

Docker met de plus à disposition un hub pour partager des images : https://hub.docker.com/. On y trouve par exemple celles permettant de faire tourner un site en MediaWiki : https://hub.docker.com/_/mediawiki.

Installation

Il existe plusieurs versions de Docker[2] :

- Docker CE (*community engine*) : gratuit. Idéal sur un PC.
- Docker EE (*enterprise engine*) : version payante certifiée, plutôt pour les serveurs.
- Docker Enterprise : payant et dispose d'outils supplémentaires, par exemple pour gérer les images et les conteneurs.

Une fois installé, la commande suivante doit fonctionner : `docker --version`.

Linux

En 2019 on trouve des binaires pour les distributions de Linux suivantes : CentOS, Debian, Fedora et Ubuntu[3].

Par exemple sur Ubuntu :

Terminal en superutilisateur

```
apt-get install apt-transport-https ca-certificates curl
gnupg-agent software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg |
apt-key add -
apt-key fingerprint 0EBFCD88
add-apt-repository    "deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs)
stable"
apt-get update
apt-get install docker-ce docker-ce-cli containerd.io
docker -v
```

```
curl -L "https://github.com/docker/compose/releases/download
/1.25.4/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local
/bin/docker-compose
chmod +x /usr/local/bin/docker-compose
docker-compose --version
```

Après installation, le daemon Docker se lancera à chaque démarrage avec la possibilité de remonter certains conteneurs automatiquement[4].

 A ce stade seul root peut utiliser Docker. Il faut ajouter les utilisateurs qui doivent le lancer au groupe "docker" :

```
sudo usermod -aG docker mon_utilisateur
```

MacOS

A télécharger sur https://docs.docker.com/docker-for-mac/install/.

Windows

Télécharger et installer Docker Desktop depuis le site officiel.

- Les versions antérieures à la 2.1.7 sur Windows pro en AD (active directory), imposent qu'un administrateur du domaine autorise l'accès à C: (un admin local ne suffit pas).
- En avril 2020 la version 2.3 (et les suivantes, au moins : 2.3.0.1, 2 et 3) freeze très régulièrement (broken pipe). Il était donc préconisé d'utiliser la 2.2.0.5[5]. Mais c'est résolu dans Docker 3.0.0.

Uniquement pour les versions de Windows pro, car Hyper-V est nécessaire. A télécharger sur https://docs.docker.com/docker-for-windows/install/.

Une interface graphique de gestion de conteneur nommée Kitematic[6] peut être intégrée dans un deuxième temps.

⚠ Toute désinstallation de Docker supprimera les images construites sur le poste, et pourra donc occasionner de longs téléchargements après réinstallation et relance.

⚠ Sur Windows pro en AD, il faut ajouter le compte qui utilisera Docker Desktop dans le groupe "docker-users" :

```
net localgroup docker-users AD\mon_compte /add
```

hosts

Sur Windows, Docker Desktop modifie le fichier hosts en ajoutant :

```
# Added by Docker Desktop
192.168.1.20 host.docker.internal
192.168.1.20 gateway.docker.internal
# To allow the same kube context to work on the host and the
container:
127.0.0.1 kubernetes.docker.internal
# End of section
```

WSL

Anciennement baptisé *Tech Preview*, *Docker Desktop WSL 2 Backend* utilise Windows Subsystem for Linux pour optimiser les performances de Docker sur Windows[7], en l'installant dans une VM Linux pouvant accéder à C:.

Références

1. https://docs.docker.com/storage/storagedriver/

2. https://docs.docker.com/install/overview/
3. https://docs.docker.com/install/linux/docker-ce/ubuntu/
4. https://www.ionos.fr/digitalguide/serveur/configuration/tutoriel-docker-installation-et-premiers-pas/
5. https://docs.docker.com/docker-for-windows/release-notes/#docker-desktop-community-2205
6. https://kitematic.com/
7. https://engineering.docker.com/2019/10/new-docker-desktop-wsl2-backend/

Gestion des conteneurs

Registres

Les images des conteneurs peuvent être stockées dans un registre, privé ou public comme https://hub.docker.com/.

Pour en récupérer une :

```
  docker login
  docker image pull $IMAGE_NAME || echo " >> missing image"
```

Exemple :

```
 docker image pull maarch/maarchrm:latest
```

Les images téléchargées sont stockées :

- Sur Linux, dans /var/lib/docker/.
- Sur MacOS, dans /Users/nom_utilisateur/Library/Containers/com.docker.docker/.
- Sur Windows, dans C:\ProgramData\docker\ ou C:\ProgramData\DockerDesktop\.

Pour mettre à jour une image sur un registre à partir d'un Dockerfile :

```
  docker image build
  docker image push $IMAGE_NAME
```

run

"run" s'utiliser pour lancer un conteneur à partir de son image (automatiquement téléchargée si absente en local). Une fois la commande exécutée, le conteneur s'arrête. Exemple qui liste le dossier courant :

```
 docker run maven:latest ls
```

```
 bin
 boot
 dev
 etc
 home
 lib
 lib64
 media
```

```
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
```

Pour rester connecté dedans, on utilise le mode interactif (-it) :

```
docker run -it redis
redis-cli
```

Pour partager un dossier avec la machine hôte, on peut utiliser "-v" pour "volume", ou "--mount" si le conteneur a des services[1]. Exemple qui liste le dossier de la machine hôte partagé avec le conteneur :

```
docker run -it -v "/$(pwd)/":/usr/src/mymaven maven ls usr/src
/mymaven
```

exec

Pour exécuter une commande dans un conteneur. Ex :

```
docker exec redis sh -c 'pwd'
```

Autre exemple :

```
docker exec php8.0-fpm sh -c 'cd /var/www/mon_app; composer
install'
```

Fonctionne aussi avec un fichier :

```
docker exec redis script.sh
```

Dans tous les conteneurs

Pour lancer une commande dans tous les conteneurs, il faut créer le fichier suivant[2] :

```
for container in `docker ps -q`; do
  docker inspect --format='{{.Name}}' $container;
  docker exec -it $container $1;
done
```

Ensuite le lancer avec la commande en paramètre :

```
./dockers.sh date

/php7.3-fpm
Thu Mar 19 11:15:34 Europs 2020
/php7.4-fpm
Thu Mar 19 12:15:34 Europs 2020
```

Rentrer dans un conteneur

Pour utiliser le conteneur, on peut rentrer dedans en utilisant le mode interactif :

```
docker exec -it redis bash
```

Cela permet d'enchainer les commandes (comme si on était sur un serveur hôte en SSH).

build

Pour lancer un conteneur à construire, il faut le faire depuis le dossier de son Dockerfile[3] :

```
docker build .
```

ps

Pour obtenir la liste des conteneurs lancés :

```
docker ps
```

Pour tous les conteneurs :

```
docker ps -a
```

Pour en supprimer un :

```
docker rm nom_du_coteneur
```

image

Pour toutes les images :

```
docker image ls
```

Pour en supprimer une :

```
docker rmi nom_image
```

save et load

Permet de sauvegarde une image. Pour toutes les sauvegarder :

```
 docker save $(docker images -q) -o images_docker.tar
```

On peut ensuite les recharger (par exemple sur une autre machine) avec :

```
docker load --input images_docker.tar
```


Un save sur Docker Desktop Windows 2.3 ne permettra pas un load sur la 2.2.

stats

Pour connaitre la consommation en ressources de chaque conteneur :

```
docker stats --no-stream
```

Ex :

```
CONTAINER ID   NAME    CPU %     MEM USAGE / LIMIT     MEM %
NET I/O           BLOCK I/O         PIDS
f10f50a685a6   app_1   0.00%     77.97MiB / 15.38GiB   0.50%
322kB / 1.19MB    11.1MB / 9.2MB    6
6ef747c18fe3   app_2   1.00%     48.89MiB / 15.38GiB   0.31%
6.67kB / 0B       92.7MB / 5.91MB   6
```

prune

Pour supprimer les conteneurs arrêtés[4] :

```
docker container prune
```

Les images :

```
docker image prune
```

Les réseaux :

```
docker network prune
```

Les conteneurs, images et réseaux inutilisés :

```
docker system prune
```

Enfin pour les volumes (généralement volumineux) :

```
docker volume prune
```

> **Remarque :** mieux faut traiter les conteneurs avant les volumes car prune ne supprime pas les volumes utilisés par des conteneurs.

Il existe des filtres[5] :

- -a all : supprime tout.
- -f force.
- --filter : filtre par clé valeur. Si la clé est "until", alors il filtre par date (ex : until=720h).
- --volumes : supprime les volumes des objets en paramètre.

Références

1. https://docs.docker.com/storage/volumes/
2. https://gist.github.com/timhodson/ea11c76424e5b3f36c017d9d0ca7ad10
3. https://docs.docker.com/engine/reference/builder/
4. https://docs.docker.com/engine/reference/commandline/container_prune/
5. https://docs.docker.com/engine/reference/commandline/system_prune/

Dockerfile

Une fois le processus Docker lancé, on peut commencer à lui demander de construire des conteneurs, de zéro ou à partir d'images d'un registre.

Rôle

Il existe deux sortes de conteneur :

- Celui issu d'une image téléchargée et utilisée telle qu'elle.
- Celui construit avec des instructions personnalisées.

Par convention, les instructions pour construire un conteneur sont écrites dans un fichier appelé "Dockerfile". Il est recommandé de les placer dans un dépôt Git pour les versionner.

Commandes Dockerfile

Les commentaires sont précédés du croisillon (#).

ADD

Permet de copier des fichiers ou dossiers de la machine hôte vers le conteneur[1].

⚠ Les Dockerfile n'ont jamais accès aux volumes de l'hôte, sauf dans l'ENTRYPOINT.

ARG

Crée une variable qui sera accessible dans la première image (FROM). Ex :

```
ARG NODE_VERSION=10
...
RUN curl -sL https://deb.nodesource.com/setup_${NODE_VERSION}.x |
bash
```

⚠ Éviter d'utiliser ce système pour bénéficier du *build cache* : ainsi Docker ne retéléchargera pas toutes les dépendances du Dockerfile lors du changement d'une variable située en début de Dockerfile, et n'affectant qu'une seule d'entre elles.

COPY

Copie des fichiers ou dossiers de la machine hôte vers le conteneur[2]. Ainsi, l'image les embarquera (contrairement à s'ils étaient montés dans un volume).

Ex :

```
COPY entrypoint.sh /usr/local/bin/entrypoint.sh
```

ENTRYPOINT

Nom d'un fichier exécutable (par exemple en shell) qui s'exécutera après chaque lancement du conteneur[3].

Ex :

```
ENTRYPOINT /usr/local/bin/entrypoint.sh
```

Le fichier exécuté par l'entrypoint est le principal moyen d'accéder à la machine hôte, puisque le Dockerfile ne peut pas (en dehors de COPY).

ENV

Crée une variable d'environnement accessible à toutes les images du Dockerfile[4].

EXPOSE

Ouvre le port logiciel mentionné du conteneur[5].

FROM

Part d'une image Docker pour en créer une autre qui la complète[6].

Cela permet par exemple le *multistage build* : se servir d'une première image comme *builder* volumineux (avec des sources, des dépendances nécessaires à des installations), duquel on copie seulement les fichiers à utiliser dans une autre image plus légère, qui sera montée en conteneur[7].

LABEL

Ajoute une description à une image[8].

RUN

Lance une commande. Ex :

```
RUN ls
```

STOPSIGNAL

Signal d'arrêt à envoyer au conteneur, au format nombre ou nom. Par exemple SIGKILL pour le tuer.

USER

Se connecte à l'utilisateur et mot de passe en paramètre pour lancer les instructions `RUN`, `CMD` ou `ENTRYPOINT`[9].

VOLUME

Point de montage accessible par l'hôte et les autres conteneurs[10].

WORKDIR

Définit le répertoire de travail, c'est-à-dire celui à partir duquel les commandes partent (ex : `pwd`), y compris quand on se connecte au conteneur après son montage.

.dockerignore

Un fichier .dockerignore peut être placé dans un dépôt pour que son conteneur ignore le contenu qui y figure lors des commandes `ADD` et `COPY`. Exemple pour Symfony :

```
node_modules/*
var/*
vendor/*
```

Bonnes pratiques

Voir https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

Exemple d'image construite

Voici un exemple de serveur Apache avec PHP7.1[11] :

```
FROM debian:sid

ENV TZ=Europs/Paris
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ >
/etc/timezone

ENV MEDIAWIKI_VERSION wmf/1.30.0-wmf.4

# XXX: Consider switching to nginx.
RUN set -eux; \
```

```
    apt-get update; \
    apt-get install -y --no-install-recommends \
        ca-certificates \
        apache2 \
        libapache2-mod-php7.1 \
        php7.1-mysql \
        php7.1-cli \
        php7.1-gd \
        php7.1-curl \
        php7.1-mbstring \
        php7.1-xml \
        imagemagick \
        netcat \
        git \
    ; \
    rm -rf /var/lib/apt/lists/*; \
    rm -rf /var/cache/apt/archives/*; \
    a2enmod rewrite; \
    a2enmod proxy; \
    a2enmod proxy_http; \
    # Remove the default Debian index page.
    rm /var/www/html/index.html


# MediaWiki setup
RUN set -eux; \
    mkdir -p /usr/src; \
    git clone \
        --depth 1 \
        -b $MEDIAWIKI_VERSION \
        https://gerrit.wikimedia.org/r/p/mediawiki/core.git \
        /usr/src/mediawiki \
    ; \
    cd /usr/src/mediawiki; \
    git submodule update --init skins; \
    git submodule update --init vendor; \
    cd extensions; \
    # Extensions
    # TODO: make submodules shallow clones?
    git submodule update --init --recursive VisualEditor; \
    git submodule update --init --recursive Math; \
    git submodule update --init --recursive EventBus; \
    git submodule update --init --recursive Scribunto; \
    git submodule update --init --recursive ParserFunctions; \
    git submodule update --init --recursive SyntaxHighlight_GeSHi;
\
    git submodule update --init --recursive Cite; \
    git submodule update --init --recursive Echo; \
    git submodule update --init --recursive Flow; \
    git submodule update --init --recursive PageImages; \
    git submodule update --init --recursive TextExtracts; \
```

```
    git submodule update --init --recursive MobileFrontend; \
    git submodule update --init --recursive TemplateData; \
    git submodule update --init --recursive ParserFunctions; \
    git submodule update --init --recursive Citoid


COPY php.ini /usr/local/etc/php/conf.d/mediawiki.ini

COPY apache/mediawiki.conf /etc/apache2/
RUN echo "Include /etc/apache2/mediawiki.conf" >> /etc/apache2
/apache2.conf

COPY docker-entrypoint.sh /entrypoint.sh

EXPOSE 80 443
ENTRYPOINT ["/entrypoint.sh"]
CMD ["apachectl", "-e", "info", "-D", "FOREGROUND"]
```

Références

1. https://docs.docker.com/engine/reference/builder/#add
2. https://docs.docker.com/engine/reference/builder/#copy
3. https://docs.docker.com/engine/reference/builder/#entrypoint
4. https://docs.docker.com/engine/reference/builder/#env
5. https://docs.docker.com/engine/reference/builder/#expose
6. https://docs.docker.com/engine/reference/builder/#from
7. https://docs.docker.com/develop/develop-images/multistage-build/
8. https://docs.docker.com/engine/reference/builder/#label
9. https://docs.docker.com/engine/reference/builder/#user
10. https://docs.docker.com/engine/reference/builder/#volume
11. https://github.com/wikimedia/mediawiki-docker/blob/master/dev/Dockerfile

Docker-compose

La commande `docker-compose` est un utilitaire généralement fourni avec Docker, permettant d'orchestrer plusieurs images et conteneurs avec la même commande[1]. Pour ce faire, les paramétrages de l'ensemble des conteneurs doit être définit dans le fichier *docker-compose.yml* à la racine du projet.

Installation

Linux

Lancer[2] :

```
curl -L "https://github.com/docker/compose/releases/download
/1.24.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local
/bin/docker-compose
chmod +x /usr/local/bin/docker-compose
```

Commandes

version

Version de docker-compose[3]. Exemple en 2019 :

```
version: '3.7'
```

networks

Définition du réseau des VM Docker.

services

Liste des conteneurs à construire.

image

Nom de l'image à télécharger sur https://hub.docker.com/. Elle peut être suivie d'un tag pour en indiquer la version.

Exemples :

```
    image: 'mariadb'
```

```
    image: 'mariadb:latest'
```

```
    image: 'mariadb:10.4'
```

extends

"image" permet donc de lancer un groupe d'applications, qui sont par ailleurs lançables individuellement. Mais pour partager des configurations on peut aussi utiliser "extends"[4] :

```
  extends:
    file: webapp/docker-compose.yml
    service: webapp
```

build

Alternativement à l'image, on peut indiquer le chemin d'un dockerfile pour construire son propre conteneur.

Si le conteneur ne partage aucun fichier avec d'autres, indiquer simplement le nom du dossier contenant le dockerfile :

```
        build: './php7.3-fpm'
```

Sinon, préciser le contexte où le conteneur devra récupérer les fichiers partagés nécessaires à son build :

```
        build:
            context: .
            dockerfile: './php7.3-fpm/Dockerfile'
```

volumes

Mapping des répertoires partagés entre la machine hôte et le conteneur :

```
        volumes:
            - '$HOME/www:/var/www'
```

La variable $HOME vaut "~" par défaut (dossier de l'utilisateur courant), mais peut être remplacée dans le fichier .env.

ports

Mapping du partage des ports. Ex :

```
        ports:
            - 3306:3306
```

environment

Injecte des variables d'environnement dans le conteneur. Très utile pour que les conteneurs soient à l'heure de la machine hôte :

```
        environment:
            TZ: "Europe/Paris"
```

env_file

Définit le nom d'un fichier contenant des variables d'environnement récupérables dans *docker-compose.yml*, avec la syntaxe "${ma_variable}". Exemple :

```
    env_file: .env
    environment:
        HOST_UID: "${UID}"
```

⚠ Sur Windows le changement de l'UID entraine une modification des droits de tous les fichiers.

depends_on

Permet de spécifier qu'un conteneur doit en attendre un autre pour être lancé.

restart

Indique si le conteneur doit se lancer au démarrage du daemon Docker (donc de la machine hôte). Vaut "no" par défaut. Ex :

```
        restart: always
```

Autre valeur possible : unless-stopped

container_name

Permet de forcer un nom de conteneur.

hostname

Permet de forcer un nom de machine dans le conteneur. Utile si on a une application qui doit pointer dessus dans son .env (car "localhost" fonctionne quand le serveur était installé directement sur la machine hôte mais pas dans un conteneur).

network

Permet de forcer une adresse IP pour le conteneur.

⚠ Dans docker-compose.yml, il faut toujours remplir le paramètre "default" de "networks" pour ne pas qu'il prenne une plage utilisée.

extra_hosts

Remplit le /etc/hosts du conteneur. Ex :

```
extra_hosts:
    - "mon_serveur_local.localhost:172.20.0.2"
```

command et entrypoint

Pour exécuter un script à chaque lancement du conteneur.

Exemples

Minimal

Exemple de docker-compose.yml contenant un seul conteneur CentOS, qui a le droit d'accéder au dossier *~/www* :

```
version: '3.2'
services:
  centos:
    image: 'centos/systemd'
    volumes:
        - '$HOME/www:/var/www'
```

Complet

```
version: '3.2'
```

```
networks:
    default:
        driver: bridge
        ipam:
            driver: default
            config:
                - subnet: 172.170.0.0/16

services:
    mariadb:
        hostname: 'mariadb'
        image: 'mariadb:10.4'
        ports:
            - 3306:3306
        environment:
            MYSQL_ROOT_PASSWORD: 'wikibooks'
        # Partage pour les commandes SQL "into outfile" et "load
data infile"
        volumes:
            - '$HOME/www:/var/www'
        restart: always
        networks:
            default:
                ipv4_address: 172.170.0.3

    adminer:
        hostname: 'adminer'
        image: 'adminer'
        ports:
            - 8080:8080
        restart: always
        networks:
            default:
                ipv4_address: 172.170.0.4
```

Remarque : une alternative aux IP fixes (permettant des noms de domaines personnalisés) et de passer par des ports (comme pour Adminer ci-dessus, qui est accessible sur http://localhost:8080).

Gestion

Pour relancer le conteneur sur Linux :

```
docker-compose stop; docker-compose build; docker-compose up -d
```

Pour relancer le conteneur sur Windows :

```
docker-compose stop; docker-compose build; docker-compose up -d
```

Pour rentrer dedans :

```
docker-compose exec centos bash
```

Ou le lancer et rentrer dedans en même temps :

```
docker-compose run centos bash
```

Ou exécuter une seule commande shell dedans avant de revenir à la machine hôte :

```
docker-compose exec centos sh -c 'ls -alh'
```

Logs

Pour voir les logs de tous les conteneurs en live :

```
  docker-compose logs -f
```

Pour voir les logs d'un seul conteneur :

```
  docker-compose logs nom_du_conteneur
```

Supprimer les logs

L'emplacement des logs d'un conteneur est visible avec :

```
  docker inspect --format='{{.LogPath}}' nom_du_conteneur
```

Sur Linux

Pour supprimer les logs de tous les conteneurs sur Linux :

```
  find /var/lib/docker/containers/ -type f -name "*.log" -delete
```

Puis redémarrer les conteneurs pour qu'ils recréent des logs.

Sur Windows

Sur Windows, comme les logs sont dans le fichier C:\ProgramData\DockerDesktop\vm-data\DockerDesktop.vhdx, il faut d'abord se connecter à la VM Docker pour exécuter cette commande[5]. Exemple en DOS :

```
docker run --privileged -it -v /var/run/docker.sock:/var/run
/docker.sock jongallant/ubuntu-docker-client
docker run --net=host --ipc=host --uts=host --pid=host -it
--security-opt=seccomp=unconfined --privileged --rm -v /:/host
alpine /bin/sh
chroot /host
```

Références

1. https://xataz.developpez.com/tutoriels/utilisation-docker/#LXII-B
2. https://docs.docker.com/compose/install/
3. https://docs.docker.com/compose/compose-file/
4. https://docs.docker.com/compose/extends/
5. https://blog.jongallant.com/2017/11/ssh-into-docker-vm-windows/

Kubernetes

Kubernetes permet une orchestration de plusieurs conteneurs en production. En effet, il offre des options de quotas et relances automatiques.

La configuration peut être réalisée par l'interface graphique ou par la commande kubectl[1].

pod

Un *pod* désigne un groupe de conteneur géré par Kubernetes[2].

replica set

Un *replica set* désigne un ensemble de pods, dont le nombre maximum peut y être défini.

secret

Mot de passe chiffré.

CronJob

Lance un conteneur dédié à une tâche planifiée (cron), à chaque fois qu'elle tourne[3].

Ingress

Accès à l'application (DNS).

PVC

Persistent volume : stocke des données indépendamment du redémarrage des pods.

Références

1. https://kubernetes.io/fr/docs/reference/kubectl/overview/
2. https://kubernetes.io/fr/docs/concepts/workloads/pods/pod/
3. https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/

Problèmes connus

Accéder aux logs

Par exemple si un conteneur ne se lance pas ou se relance toutes les secondes, un motif plus précis qu'en console peut se trouver dans les logs.

Pour le démon :

```
tail /var/log/docker.log
```

Pour les conteneurs

```
docker-compose logs
```

Ces deux commandes acceptent l'argument "-f" pour les afficher en temps réel.

Pour un seul conteneur

```
docker-compose logs nom_du_conteneur
```

ou :

```
docker logs nom_du_conteneur
```

Récupérer l'IP d'un conteneur

```
docker inspect -f '{{range .NetworkSettings.Networks}}
{{.IPAddress}}{{end}}' nom_du_conteneur
# ou
docker inspect
--format='{{.NetworkSettings.Networks.apps.IPAddress}}'
nom_du_conteneur # où "apps" est le nom du réseau
```

Réinitialiser les conteneurs à zéro


Cette opération peut prendre du temps car Docker retélécharge tout les paquets ensuite.

Linux :

```
docker rm -f $(docker ps -a -q); docker rmi -f $(docker images
-q); docker network rm $(docker network ls -q)
```

Windows :

```
docker rm -f $(docker ps -a -q); docker rmi -f $(docker images
-q); docker network rm $(docker network ls -q)
```

La partie *network* peut être exécutée indépendamment, par exemple en cas de *ERROR: Pool overlaps with other one on this address space*.

Messages d'erreur

Sous Windows

/usr/bin/env: 'php\r': No such file or directory

Utiliser "winpty". Ex :

```
docker exec -it php7.3-fpm bash
```

Sinon[1] :

```
docker exec -it <container> bash
cd bin
tr -d '\015' <console >console.new
mv console console.old
mv console.new console
```

Certains conteneurs ne peuvent pas être lancés (timeout)

Vérifier que le partage Windows a bien été fait : clic droit, Settings, Resources, File Sharing, C: (puis relancer Docker Desktop).

Le partage Windows ne fonctionne pas

Si ça n'a jamais fonctionné : ajouter son compte dans le groupe "docker-users".

C'est peut-être lié à la plage d'IP de Docker, remettre celle par défaut.

Si ça marchait sur Windows 10 pro dans un Active Directory et que ça ne fonctionne plus en dehors de l'AD ou en VPN, c'est un bug avec Docker Desktop 2.1.0.5 qui semble résolu dans la 2.1.6.1. En effet, seul un admin de l'AD peut autoriser le partage des volumes, et le port 445 doit être ouvert.

Pour tester si ça marche :

```
docker run -v c:/Users:/data alpine ls data
```

500: {"Message":"Unhandled exception: Drive has not been shared"}'

Dans Docker Desktop, partager le volume concerné.

502 Bad Gateway dans Nginx et *Bus error* dans les commandes PHP

Redémarrer Docker Desktop.

Sinon c'est un processus PHP qui gonfle à outrance à cause du code.

ERROR: failed to create new listening socket: socket(): Address family not supported by protocol (97)

Relancer Docker Desktop.

Error: mounting wslCLIDest: stat /mnt/host/c/Program Files/Docker/Docker /resources/wsl/docker-wsl-cli.iso: no such file or directory

Décocher *Use the WSL 2 based engine* dans les options et relancer Docker Desktop[2].

fatal: not a git repository (or any parent up to mount point /var) Stopping at filesystem boundary (GIT_DISCOVERY_ACROSS_FILESYSTEM not set)

Redémarrer Docker Desktop.

Error response from daemon: Mount denied: The source path "mon_dossier;C" doesn't exist and is not known to Docker

Sous Git Bash dans Windows, il faut préfixer le chemin local par "/". Ex : `docker run -it --rm -v /${PWD}:/wkDir $IMAGE_TAG yarn dev`[3]

Invalid mode /var/www

Le chemin d'accès dans docker-compose.yml n'est pas compris. Cela se produit pas exemple quand on met des antislashs à la place des slashs.

Cannot connect to the Docker daemon. Is the docker daemon running on this host?

Relancer docker en administrateur.

Cannot start service xxx: Address already in use

Deux conteneurs utilisent le même port. Dans Docker-compose, si l'un des deux avait été retiré, il était peut-être configuré en *restart: always* et il faut le remettre dans docker-compose.yml pour le stopper.

Cannot start service xxx :driver failed programming external connectivity on endpoint

Impossible de lancer un conteneur sur Windows :

- Soit Docker n'a pas accès au volume, et il faut cocher la case "Shared drives" dans Docker Desktop, ou lancer la commande suivante en acceptant le partage :

```
docker run --rm -v c:/Users:/data alpine ls /data
```

- Soit Docker n'a pas accès aux ports de ses conteneurs, et il faut fermer les processus qui les utilisent. Il peut même s'agir d'une deuxième instance de Docker.

Couldn't connect to Docker daemon at http+docker://localhost - is it running?

```
/etc/init.d/docker start
```

Si le démon ne se lance pas, upgrader l'OS et redémarrer. Sinon, réinstaller Docker.

Could not resolve host: xxx (pas de DNS)

Revoir la plage d'IP définie dans le paragraphe "networks" de docker-compose.yml.

Device or resource busy, Cette action ne peut pas être réalisée car le fichier est ouvert dans com.docker.backend.exe

C'est un bug connu (sur Linux et Windows) quand composer installe certains paquets[4]. On ne peut supprimer le fichier qu'en fermant tout Docker (sous Windows en tout cas, il ne suffit pas de le redémarrer).

Cela se produit (en cas de réécriture d'historique ?), repartir d'une branche propre avant de relancer "composer install". Sinon, le lancer dans une VM et récupérer le dossier vendor.

Error response from daemon: Get https://xxx: no basic auth credentials

Sur certains dépôts privés, pour faire un `docker pull` il faut préalablement se loguer. Ex :

```
docker login -u mon_utilisateur -p mon_mdp mon_url
```

Invalid interpolation format for "environment" option in service "documents": "^https?://.*?$"

Échapper le $ interprété dans docker-compose.yml. Par exemple, remplacer :

```
CORS_ALLOW_ORIGIN: "^https?://.*?$"
```

par :

```
CORS_ALLOW_ORIGIN: "^https?://.*?$$"
```

standard_init_linux.go:211: exec user process caused "no such file or directory"

Cela peut se produire quand des conteneurs testés sur Linux sont utilisés sur Windows.

Il faut changer les retours chariots du fichier appelé par "ENTRYPOINT" dans le Dockerfile, de CRLF (Windows) vers LF (Unix). Ex : `dos2unix php7.4-fpm/bin/custom-docker-php-entrypoint`

Puis reconstruire et relancer le conteneur.

Autres solutions

On peut aussi lancer "dos2unix" automatiquement depuis le dockerfile sur une copie de l'exéutable[5].

S'il s'agit d'un dépôt Git, on peut aussi le sauvegarder autrement depuis Linux[6] :

- lancer `git config core.autocrlf false`
- créer un fichier .gitattributes[7] contenant `text eol=lf`

container_linux.go:349: starting container process caused "exec: \"custom-docker-php-entrypoint\": executable file not found in $PATH": unknown

Cela peut se produire quand des conteneurs testés sur Windows sont utilisés sur Linux.

Il faut changer leur donner les droits d'exécution (`chmod +x`).

Puis reconstruire et relancer le conteneur.

Références

1. https://www.thetopsites.net/article/50789087.shtml
2. https://github.com/docker/for-win/issues/6822
3. https://stackoverflow.com/questions/50608301/docker-mounted-volume-adds-c-to-end-of-windows-path-when-translating-from-linux
4. https://github.com/moby/moby/issues/22260
5. https://willi.am/blog/2016/08/11/docker-for-windows-dealing-with-windows-line-endings/
6. https://stackoverflow.com/questions/53165471/building-docker-images-on-windows-entrypoint-script-no-such-file-or-directory
7. https://git-scm.com/docs/gitattributes
