COURS DOCKER

5ème année Génie Informatique, Ecole Polytechnique Yaoundé

Présenté par : Ing MASSAGA Aristide

Phd student in software engineering
Master 2 in microservice architecture
Software engineer, devops, microservice architect

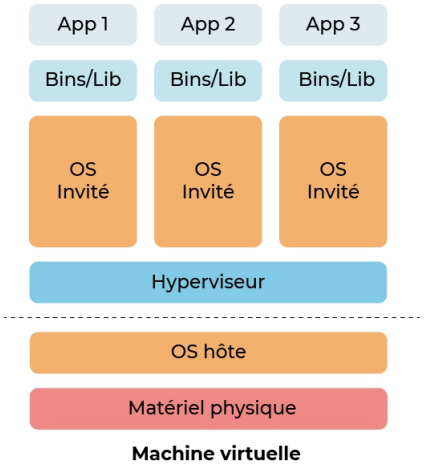
https://m-aristide.github.io

Virtualisation et conteneur

Virtualisation

La virtualisation hardware consiste à faire fonctionner sur une même machine physique, plusieurs systèmes comme s'ils fonctionnaient sur des machines physiques distinctes.
Un serveur est composé de plusieurs "couches" :

- Applications (Ce sont les apps tel que Spotify, gmail, word... qui vont "tourner" sur la machine.)
- OS (Operating System ou système d'exploitation : c'est le premier programme à s'exécuter lors du démarrage de la machine et il assure le lien entre les ressources matériel (hardware) de la machine et les applications qui "tournent" dessus. Par exemple, on parle de MacOs,Windows, iOs...)
- Hardware (machine physique peut être autant un téléphone portable, qu'un ordinateur...)



Lorsqu'on utilise une machine virtuelle (VM), ce qu'on appelle de la virtualisation lourde. En effet, vous recréez un système complet dans le système hôte, pour qu'il ait ses propres ressources.
L'isolation avec le système hôte est donc totale ; cependant, cela apporte plusieurs contraintes :

- une machine virtuelle prend du temps à démarrer ;
- une machine virtuelle réserve les ressources (CPU/RAM) sur le système hôte.

Mais cette solution présente aussi de nombreux avantages :

- une machine virtuelle est totalement isolée du système hôte ;
- les ressources attribuées à une machine virtuelle lui sont totalement réservées ;
- vous pouvez installer différents OS (Linux, Windows, BSD, etc.).

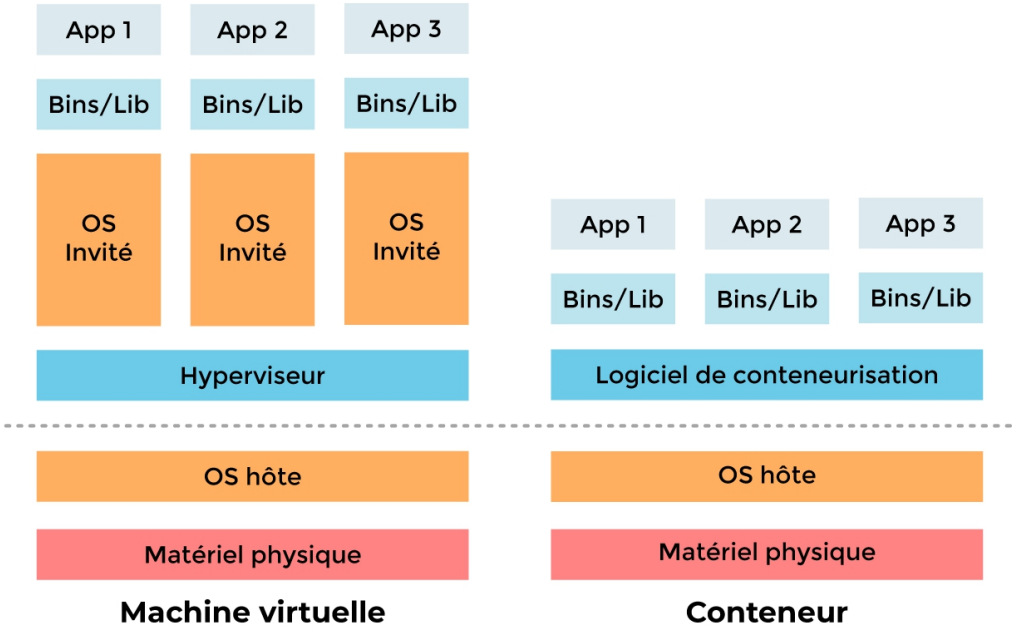
Mais il arrive très souvent que l'application qu'elle fait tourner ne consomme pas l'ensemble des ressources disponibles sur la machine virtuelle.

Conteneur

Les machines virtuelles et les conteneurs sont proches, car ce sont toutes les deux des technologies de virtualisation.

Un conteneur Linux est un processus ou un ensemble de processus isolés du reste du système, tout en étant légers.

Le conteneur permet de faire de la virtualisation légère, c'est-à-dire qu'il ne virtualise pas les ressources, il ne crée qu'une isolation des processus. Le conteneur partage donc les ressources avec le système hôte.

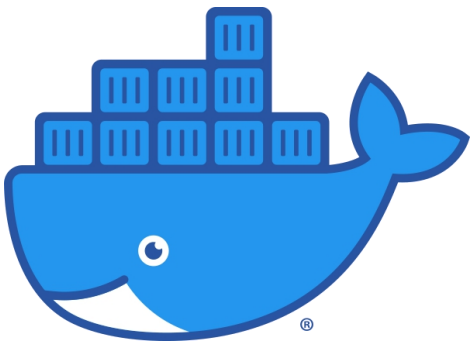


Les conteneurs, au sens d'OpenVZ et LXC, apportent une isolation importante des processus systèmes ; cependant, les ressources CPU, RAM et disque sont totalement partagées avec l'ensemble du système. Les conteneurs partagent entre eux le kernel Linux ; ainsi, il n'est pas possible de faire fonctionner un système Windows ou BSD dans celui-ci.
Ainsi l'utilisation de conteneur comme système de virtualisation apporte :

- Réduction de l'utilisation des ressources RAM et CPU
- Démarrez rapidement vos conteneurs
- Faible isolation entre les conteneurs et l'OS

Présentation de docker plateforme

Docker est un projet open source (Apache 2.0) écrit en GO et hébergé sur GitHub: https://github.com/docker. Docker, c'est donc à la fois le nom d'une technologie, les conteneurs, et celui d'une entreprise américaine créée par Solomon Hykes.



Contrairement aux systèmes de conteneur existant tels LXC ou OpenVZ, Docker apporte une notion importante dans le monde du conteneur. Dans la vision Docker, un conteneur ne doit faire tourner qu'un seul processus.

Docker Inc distribue 3 versions de Docker différentes :
- Docker Community Edition (Linux seulement) ;
- Docker Desktop (Mac ou Windows) ;
- Docker Enterprise (Linux seulement).

Docker est composé de trois éléments :

- le daemon Docker qui s'exécute en arrière-plan et qui s'occupe de gérer les conteneurs (Containerd avec runC)
- une API de type REST qui permet de communiquer avec le daemon
- Le client en CLI (command line interface) : commande docker

Par défaut, le client communique avec le daemon Docker via un socket Unix (/var/run/docker.sock) mais il est possible d'utiliser un socket TCP.

Dans docker nous allons manipuluer deux concepts principaux :
- Image : template de conteneur en read-only contenant un systeme de base et une application.
- Conteneur : Image exécutable d'un environnement complet incluant code, librairies, outils et configuration.

Avantages de Docker :
- Légèreté : Du fait que les conteneurs n'incluent pas la logique de l'OS mais uniquement ce qui est nécessaire au bon fonctionnement de l'applications
- Portabilité : Il est possible de créer, déployer et démarrer des conteneurs sur n'importe quel ordinateur ou serveur distant.
- Performance : Docker permet une montée en charge facile des applications car il va pouvoir les multiplier sur tout un ensemble d'OS, et fournir de la haute disponibilité.

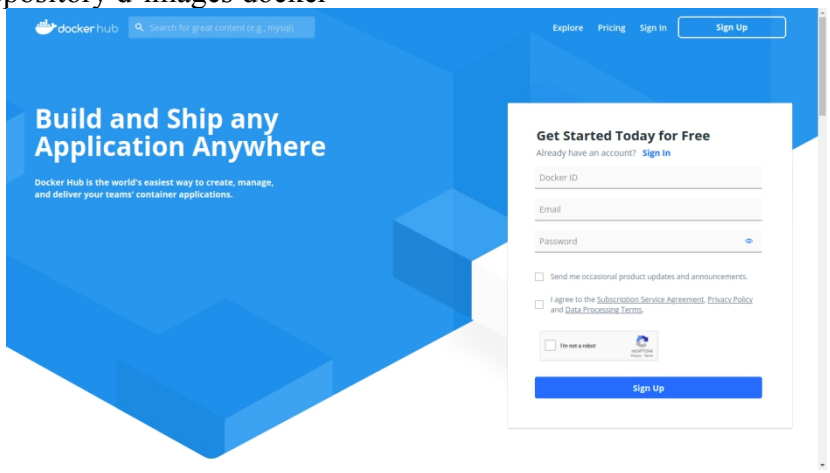
Le problème principal lié à l'utilisation de conteneur est la sécurité :
- Il facile pour un pirate de passer du conteneur vers le système d'exploitation hôte
- Attention aux applications déjà conteneurisées : vous aller peut-être faire entrer un cheval de Troie dans votre serveur.

Exécution image docker à partir du docker hub

docker hub

https://hub.docker.com/

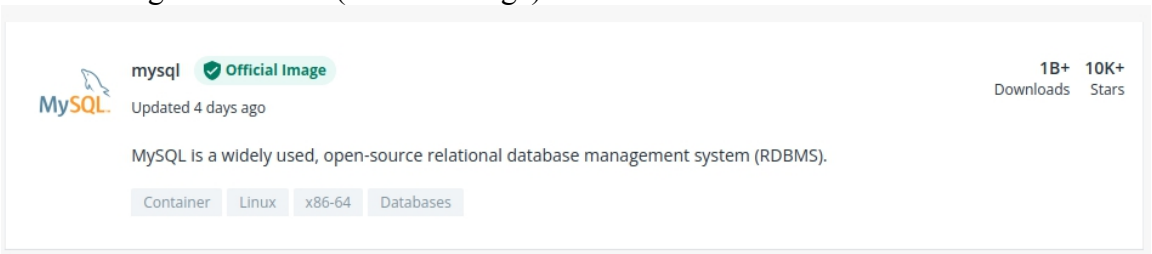
C'est un repository d'images docker



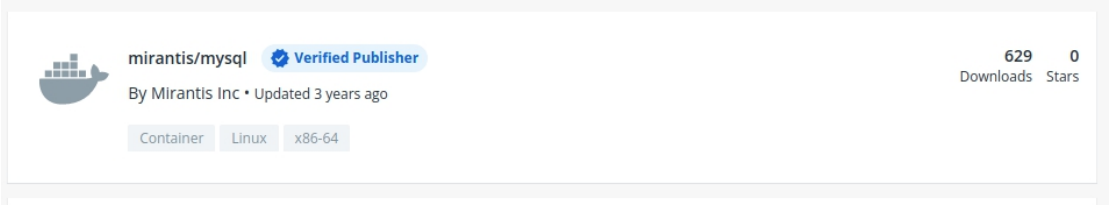
Ce qui nous intérèsse ici c'est de comprendre les informations que nous donne docker hub sur les images docker.

Sur le hub il existe 3 catégories d'images :

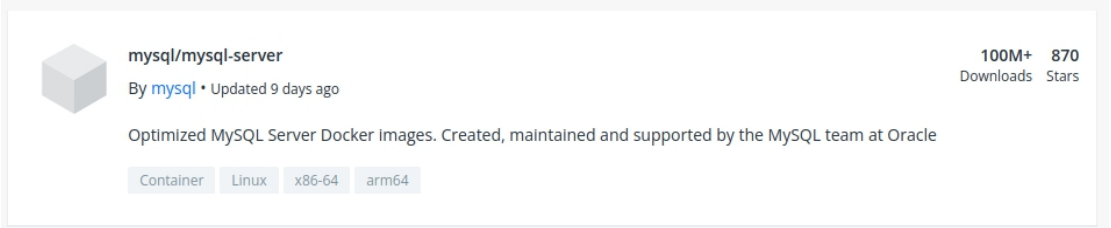
- les images officielles (Official image)



- les images dont l'auteur a été vérifié (Verified Publisher)

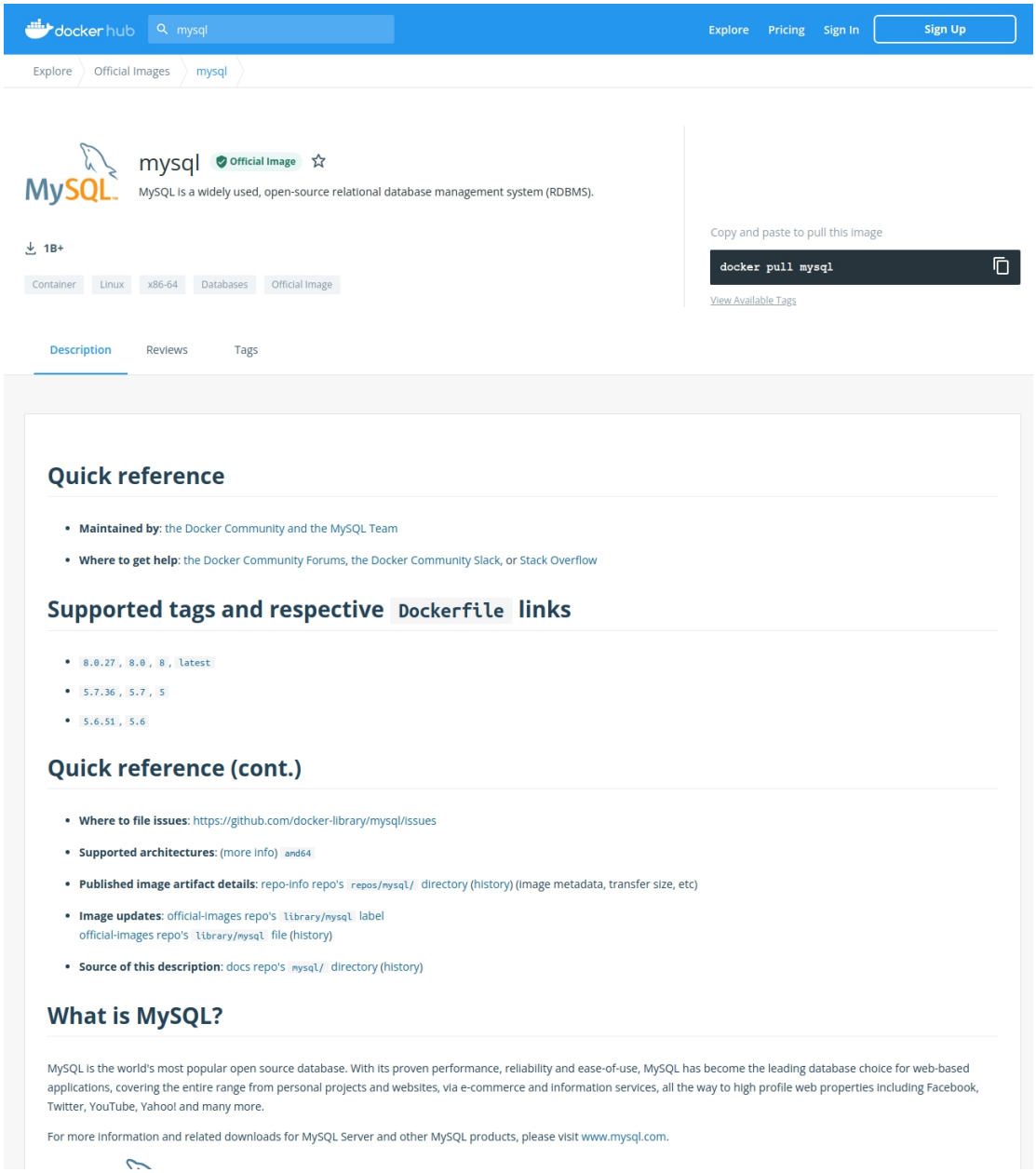


- les images tiers, car tout le monde peut publier une image



Le images officielles et les images vérifiées apportent de la sécurité quant à l'origine de l'image docker.

Chaque image docker a une page où, les informations sur l'image, son utilisation, etc sont données



docker pull

❖ docker pull hello-world

cette commande ira télécharger un l'image docker depuis le docker hub

❖ docker pull masscorp-dev.win:5000/hello-world

cette commande ira télécharger l'image docker sur le repository dont l'adresse est masscorp-dev.win:5000

docker image

La liste des commandes disponibles applicables aux images

```
aristide@aristide: ~ 143x38
aristide@aristide:~$ docker image

Usage:  docker image COMMAND

Manage images

Commands:
  build       Build an image from a Dockerfile
  history     Show the history of an image
  import      Import the contents from a tarball to create a filesystem image
  inspect     Display detailed information on one or more images
  load        Load an image from a tar archive or STDIN
  ls          List images
  prune       Remove unused images
  pull        Pull an image or a repository from a registry
  push        Push an image or a repository to a registry
  rm          Remove one or more images
  save        Save one or more images to a tar archive (streamed to STDOUT by default)
  tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE

Run 'docker image COMMAND --help' for more information on a command.
```

❖ docker image ls

```
aristide@ubuntu-s-1vcpu-1gb-fra1-01:~$ docker image ls
REPOSITORY                                TAG           IMAGE ID       CREATED         SIZE
masscorp-dev.win:5000/vmc_experience      latest        12cf2c8bfaaa   3 days ago      1.6GB
masscorp-dev.win:5000/vmc_experience      <none>        0fafc3cb4dcb   2 weeks ago     1.59GB
masscorp-dev.win:5000/app-sica            latest        5a5980c1c36e   2 months ago    215MB
masscorp-dev.win:5000/vmc_experience      <none>        1a9ef64bd719   2 months ago    829MB
masscorp-dev.win:5000/licence-front       latest        b83ecc769de7   2 months ago    135MB
vmcwordpress                              latest        a165f6f4778f   3 months ago    551MB
wordpress                                 latest        baf5889057ff   3 months ago    551MB
php                                       7.4-apache    93e55f680811   3 months ago    414MB
ubuntu                                    latest        1318b700e415   3 months ago    72.8MB
masscorp-dev.win:5000/front-sica          latest        8e29faed0e53   3 months ago    143MB
mysql                                     5.7           8cf625070931   4 months ago    448MB
mysql                                     latest        c60d96bd2b77   4 months ago    514MB
```

- ➢ REPOSITORY : indique le nom de l'image en incluant l'origine de l'image
- ➢ TAG : c'est la version de l'image
  - latest : c'est la version par défaut. Lorsqu'une image est construite sans y mettre de version
  - <none> : signie qu'on a construit une nouvelle image avec la même version qu'une image qui existait déjà
  - 5.7 : c'est une version spécifique
- ➢ IMAGE ID : l'identifiant unique de chaque image
- ➢ CREATED : la date de création de l'image
- ➢ SIZE : le poids de l'image

docker ps

```
aristide@ubuntu-s-1vcpu-1gb-fra1-01:~$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED       STATUS       PORTS
                                NAMES
b6784d76cfd6   masscorp-dev.win:5000/vmc_experience   "docker-entrypoint.s…"   3 days ago    Up 3 days    443/tcp
                                experience.primusevent.com
eeae156b2e65   diouxx/apache-proxy                  "/opt/apache-proxy-s…"   4 weeks ago   Up 2 weeks   0.0.0.0:80->80/tcp, :::80->80/tcp, 0
.0.0.0:443->443/tcp, :::443->443/tcp   apache-proxy
e1d87ab12b0f   vmcwordpress                         "docker-entrypoint.s…"   7 weeks ago   Up 2 weeks   80/tcp
                                store.primusevent.com
635a7dc298e4   vmcwordpress                         "docker-entrypoint.s…"   3 months ago  Up 2 weeks   80/tcp
                                vmr.primusevent.com
915a0c176493   mysql:5.7                            "docker-entrypoint.s…"   3 months ago  Up 2 weeks   33060/tcp, 0.0.0.0:3308->3306/tcp, :
:3308->3306/tcp                db_mysql
daf986ef3d22   wordpress                            "docker-entrypoint.s…"   3 months ago  Up 2 weeks   80/tcp
                                vmc.masscorp-dev.win
a07fab642447   masscorp-dev.win:5000/front-sica      "nginx -g 'daemon of…"   3 months ago  Up 2 weeks   80/tcp
                                app-front-sica
cb09a3bb826a   registry:2                           "/entrypoint.sh /etc…"   6 months ago  Up 2 weeks   0.0.0.0:5000->5000/tcp, :::5000->500
0/tcp                          registry_registry_1
```

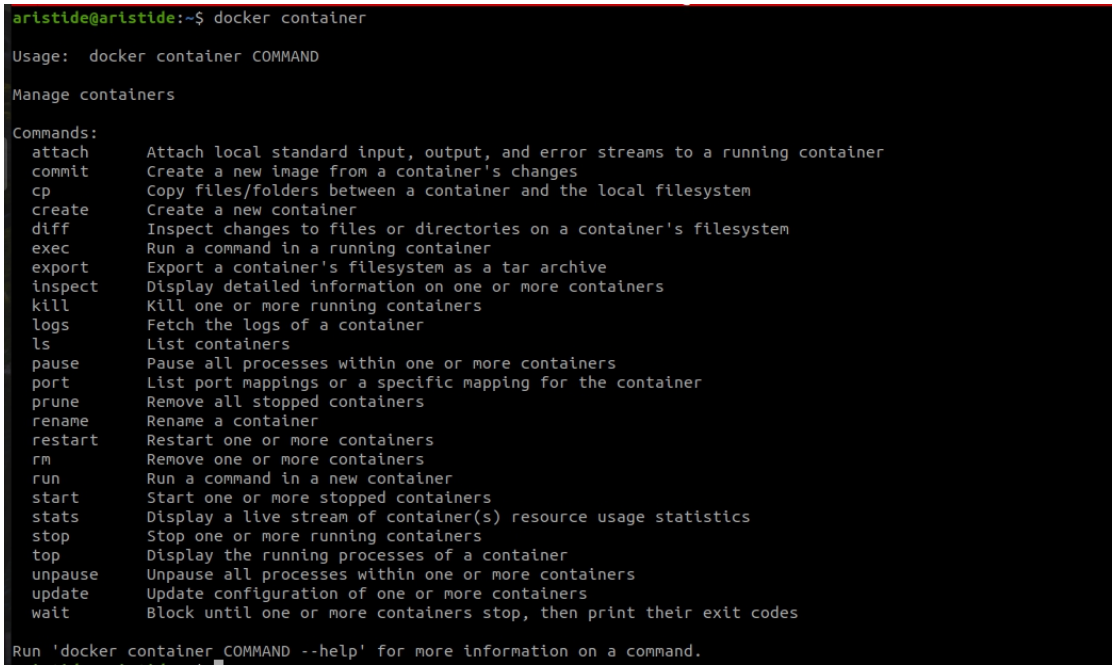
- ➢ CONTAINER ID : c'est l'identifiant unique du container
- ➢ IMAGE : c'est l'image d'origine
- ➢ NAMES : c'est le nom du container et c'est particulièrement dans le réseau

- ➢ COMMAND : c'est le processus que le docker exécute
- ➢ CREATED : c'est la date de lancement du container
- ➢ STATUS : il indique l'état d'exécution du container
  - up : le container est démarré
  - exited : le container est arrêté mais présent
  - restarting : le container est en train de redémarré
  - etc
- ➢ PORTS : ce sont les ports déclarrés lors de la construction de l'image et les ports que auxquels ces ports sont mapés sur la machine physique.

Exemple :

```
docker pull httpd
docker run -p 8080:80 httpd
docker run -p 8080:80 -d httpd
docker ps
htop
```

docker container



❖ docker inspect name_container

Permet d'obtenir les informations sur le container en cours d'exécution. Les informations sont énromes :

- Les informations sur la configuration réseau du container : NetworkSettings
- Les variables d'environnement passées au container
- Les informations sur les volumes

Exemple: `docker inspect name_container`

❖ docker start name_container

Permet de démarrer un container existant. Le container est démarré avec toute sa configuration de lorsqu'il a été créé.

Exemple :

```
docker stop name_container
docker start name_container
```

❖ docker exec

Permet d'exécuter une commande à l'intérieur du container

docker exec name_container /bin/ls -la /

```
aristide@ubuntu-s-1vcpu-1gb-fra1-01:~$ docker exec dreamy_banach /bin/ls -la /
total 76
drwxr-xr-x   1 root root 4096 Nov 21 18:36 .
drwxr-xr-x   1 root root 4096 Nov 21 18:36 ..
-rwxr-xr-x   1 root root    0 Nov 21 18:36 .dockerenv
drwxr-xr-x   1 root root 4096 Nov 19 01:45 bin
drwxr-xr-x   2 root root 4096 Oct  3 09:15 boot
drwxr-xr-x   5 root root  340 Nov 21 18:36 dev
drwxr-xr-x   1 root root 4096 Nov 21 18:36 etc
drwxr-xr-x   2 root root 4096 Oct  3 09:15 home
drwxr-xr-x   1 root root 4096 Nov 19 01:45 lib
drwxr-xr-x   2 root root 4096 Nov 15 00:00 lib64
drwxr-xr-x   2 root root 4096 Nov 15 00:00 media
drwxr-xr-x   2 root root 4096 Nov 15 00:00 mnt
```

Des options intéressantes avec la commande exec ce sont les options -i et -t
docker exec -it name_container /bin/bah
Cette commande permet d'ouvrir un terminal dans le docker

```
aristide@ubuntu-s-1vcpu-1gb-fra1-01:~$ docker exec -it dreamy_banach /bin/bash
root@bb6065f174b8:/usr/local/apache2# ls
bin  build  cgi-bin  conf  error  htdocs  icons  include  logs  modules
root@bb6065f174b8:/usr/local/apache2#
```

Devoir :
- démarrer un container docker apache (httpd)
- modifier la page d'accueil par défaut d'apache pour qu'elle affiche son nom.
- le docker doit être exposé sur le port 6907
Donc en tapant dans le navigateur http://localhost:6907, je vois voir s'afficher votre nom.

Persistance des données d'un container

Docker offre plusieurs mécanismes de persistance des données d'un container en cours d'exécution : volume et bind mounts

docker volume

Ce sont des espaces de stockage créés, gérés et controlés par docker qui sont utilisable par les container. Les volumes sont le mécanisme privilégié pour la persistance des données générées et utilisées par les conteneurs Docker. Les volumes présentent plusieurs avantages par rapport aux montages liés :

- Les volumes sont plus faciles à sauvegarder ou à migrer que les montages bind.
- Vous pouvez gérer les volumes à l'aide des commandes Docker CLI ou de l'API Docker.
- Les volumes fonctionnent sur les conteneurs Linux et Windows.
- Les volumes peuvent être partagés de manière plus sûre entre plusieurs conteneurs.
- Les pilotes de volume vous permettent de stocker des volumes sur des hôtes ou des fournisseurs de clouds distants, de chiffrer le contenu des volumes ou d'ajouter d'autres fonctionnalités.
- Les nouveaux volumes peuvent avoir leur contenu pré-rempli par un conteneur.
- Les volumes sur Docker Desktop sont beaucoup plus performants que les montages liés à partir d'hôtes Mac et Windows.

```
aristide@aristide:~$ docker volume

Usage:  docker volume COMMAND

Manage volumes

Commands:
  create      Create a volume
  inspect     Display detailed information on one or more volumes
  ls          List volumes
  prune       Remove all unused local volumes
  rm          Remove one or more volumes
```

Exemple de volume avec mysql :

docker volume create mysql

docker volume ls

docker volume inspect mysql

```
aristide@ubuntu-s-1vcpu-1gb-fra1-01:~$ docker volume inspect mysql
[
    {
        "CreatedAt": "2021-11-21T19:46:58Z",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/mysql/_data",
        "Name": "mysql",
        "Options": {},
        "Scope": "local"
    }
]
```

docker pull mysql

docker run --name myserver -e MYSQL_ROOT_PASSWORD=my-secret-pw -d -v mysql:/var/lib/mysql mysql

docker exec -it myserver /bin/bash

mysql -pmy-secret-pw

CREATE DATABASE mabase;

show databases;

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mabase             |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.01 sec)
```

exit

exit

docker rm -f myserver

docker run --name newmyserver -e MYSQL_ROOT_PASSWORD=my-secret-pw -d -v mysql:/var/lib/mysql mysql

docker exec -it newmyserver /bin/bash

show databases;

docker bind mounts

Il consiste à faire correspondre un dossier/fichier à l'intérieur du container avec un dossier/fichier dans le système de fichier de la machine hôte. Ils dépendent de la structure des répertoires et du système d'exploitation de la machine hôte. Le fichier ou le répertoire est référencé par son chemin absolu sur la machine hôte.

Il n'est pas nécessaire que le fichier ou le répertoire existe déjà sur l'hôte Docker. Il est créé à la demande s'il n'existe pas encore. Les montages liés sont très performants,

mais ils dépendent du fait que le système de fichiers de la machine hôte dispose d'une structure de répertoire spécifique.

Exemple de volume avec mysql :
```
mkdir -p ~/volumes/mysql
docker run --name myserver -e MYSQL_ROOT_PASSWORD=my-secret-pw -d -v ~/volumes/mysql:/var/lib/mysql mysql
```
ou
```
docker run --name myserver -e MYSQL_ROOT_PASSWORD=my-secret-pw -d --mount type=bind,source=~/volumes/mysql,target=/var/lib/mysql mysql mysql
```

Création d'un réseau et exécution de 2 containers dans le réseau

L'une des raisons pour lesquelles les conteneurs et services Docker sont si puissants est que vous pouvez les connecter entre eux, ou les connecter à des systèmes non Docker.
Cette rubrique n'entre pas dans les détails spécifiques aux systèmes d'exploitation concernant le fonctionnement des réseaux Docker. Vous ne trouverez donc pas d'informations sur la façon dont Docker manipule les règles iptables sur Linux ou sur la façon dont il manipule les règles de routage sur les serveurs Windows, et vous ne trouverez pas d'informations détaillées sur la façon dont Docker forme et encapsule les paquets ou gère le cryptage.

Docker offre plusieurs manières de mettre des containers en réseau :
- **User-defined bridge networks** : Les réseaux de pont définis par l'utilisateur sont les meilleurs lorsque vous avez besoin que plusieurs conteneurs communiquent sur le même hôte Docker.
- **Host networks** : Les réseaux d'hôtes sont plus adaptés lorsque la pile réseau ne doit pas être isolée de l'hôte Docker, mais que vous souhaitez que d'autres aspects du conteneur soient isolés.
- **Overlay networks** : Les réseaux superposés sont idéaux lorsque vous souhaitez que les conteneurs s'exécutant sur différents hôtes Docker communiquent, ou lorsque plusieurs applications travaillent ensemble à l'aide de services swarm.
- **Macvlan networks** : Les réseaux Macvlan sont les plus adaptés lorsque vous migrez à partir d'une configuration VM ou lorsque vous souhaitez que vos conteneurs ressemblent à des hôtes physiques sur votre réseau, chacun ayant une adresse MAC unique.
- **Third-party network plugins** : Des plugins réseau tiers vous permettent d'intégrer Docker à des piles réseau spécialisées.

User-defined bridge networks
Ce sont des réseaux bridge créés par l'utilisateur. En effet, il existe un réseau bridge par défaut dans docker dans lequel tout nouveau container est ajouté lorsqu'un réseau n'a pas été spécifié.
Les User-defined bridge networks ont les caractéristiques suivantes :
- Fournissent une résolution DNS automatique entre les conteneurs.
- Assurent une meilleure isolation.

- Les conteneurs peuvent être attachés et détachés des réseaux définis par l'utilisateur à la volée.
- Chaque réseau défini par l'utilisateur crée un pont configurable.
- Les conteneurs liés sur le réseau pont par défaut partagent des variables d'environnement.

Exemple :

Pour cet exemple nous allons mettre déployer phpmyadmin et mysql dans un même réseau et désormais c'est par phpmyadmin que nous allons manipuler notre base de données.

docker network

```
aristide@ubuntu-s-1vcpu-1gb-fra1-01:~$ docker network

Usage:  docker network COMMAND

Manage networks

Commands:
  connect     Connect a container to a network
  create      Create a network
  disconnect  Disconnect a container from a network
  inspect     Display detailed information on one or more networks
  ls          List networks
  prune       Remove all unused networks
  rm          Remove one or more networks
```

docker network create my-net

docker network ls

```
aristide@ubuntu-s-1vcpu-1gb-fra1-01:~$ docker network ls
NETWORK ID     NAME                DRIVER    SCOPE
5cf3e9ee0fe9   bridge              bridge    local
78775f5380e6   host                host      local
87c4e508f2b5   my-net              bridge    local
```

docker run --name myserver -e MYSQL_ROOT_PASSWORD=my-secret-pw -v mysql:/var/lib/mysql --network my-net -d mysql

docker inspect myserver

```
    "IPv6Gateway": "",
    "MacAddress": "",
    "Networks": {
        "my-net": {
            "IPAMConfig": null,
            "Links": null,
            "Aliases": [
                "bc16cc811c12"
            ],
            "NetworkID": "87c4e508f2b5eec70044af4fa3e8dcd961d0876ef392645f486e172bc2dcd315",
            "EndpointID": "332d09ef83b1a4038001bdc0433894c84b2d8a74d0a8e84039e76078577e94c3",
            "Gateway": "172.21.0.1",
            "IPAddress": "172.21.0.2",
            "IPPrefixLen": 16,
```

docker pull phpmyadmin

docker run --name myadmin -d --link myserver:db -p 8080:80 --network my-net -d phpmyadmin

Devoir :

Déployer un container wordpress utilisant la base de données mysql pour son fonctionnement

Construction image docker

Les images docker utilisées jusqu'à présent provenaient du docker hub. Dans cette secton nous allons apprendre à construire une image.

Dockerfile

Nous allons créer un fichier nommé "Dockerfile". Dans ce fichier Dockerfile, nous allons saisir les instructions décrivant le contenu de l'image Docker que nous voulons créer.

Le Dockerfile peut être comparé à un programme, par conséquent, il existe un langage avec une syntaxe et des mots clés pour l'écriture d'un dockerfile. Il est par la suite interprété par le docker engine afin de produit l'image.

Chaque instruction que nous allons donner dans notre Dockerfile va créer une nouvelle layer correspondant à chaque étape de la construction de l'image.

```
aristide@ubuntu-s-1vcpu-1gb-fra1-01:~$ docker pull httpd
Using default tag: latest
latest: Pulling from library/httpd
eff15d958d66: Already exists
ba1caf8ba86c: Pull complete
ab86dc02235d: Pull complete
0d58b11d2867: Pull complete
e88da7cb925c: Pull complete
```

Sur cette capture, chaque ligne après représente une couche/layer c'est-à-dire une instruction lors de la construction de l'image.
L'intérêt de ce système de couche, est que les couches sont réutilisables. C'est le cas de la couche ci-dessous, qui était déjà présente sur la machine avant que ce pull soit effectué.

```
eff15d958d66: Already exists
```

On construit une image docker devant exécuter une application java.

```
#image de base
FROM ubuntu

# indique des metadatas sur l'image
LABEL autor="Aristide MASSAGA"
LABEL version="1.0"
LABEL description="Cours docker 5gi 2021"


# mettre à jour la source des paquets
RUN apt update

# installer la jdk 8
RUN apt install -y openjdk-8-jdk

# test de la version
RUN  java -version

# crée une variable d'environnement valable dans ce fichier
ARG JAR_FILE=*.jar

# cette instruction est équivalente à faire un cd le ligne de commande
# elle change le répertoire d'exécution des commandes qui viennent après elle
WORKDIR /

# copie les fichiers à l'intérieur du document
COPY ${JAR_FILE} app.jar

# déclare que l'application dans ce container tourne sur le port 8080
EXPOSE 8080

# produira la création d'un volume docker lors de la constru
#VOLUME /bin

# c'est l'instruction de l'instruction qui est exécutée au démarrage du docker
ENTRYPOINT ["java","-jar","/app.jar"]
```

Exemple :
FROM openjdk:8-jdk-alpine
ARG JAR_FILE=todolist-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE} app.jar

EXPOSE 8080
ENTRYPOINT ["java","-jar","/app.jar"]

Pour créer une image :
docker build -t nom_image:tag chemin_vers_fichier_dockerfile

Changement de nom de l'image
docker tag ancien_nom_image:tag nouveau_nom_image:tag

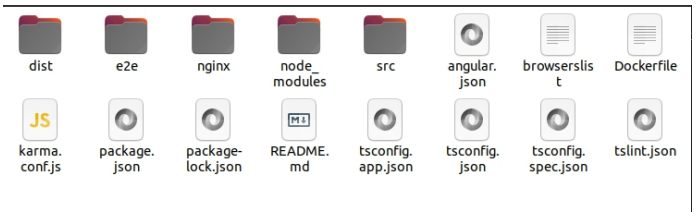
.dockerignore

Lors de la construction de l'image docker, le docker engine charge dans tout le contenu du dossier dans lequel se trouve trouve le dockerfile avant de commencer l'exécution des instructions.
Dans ce processus des fichiers ou dossiers non voulus sont chargés, ce qui fait que cette étape peut se révéler long.
Le fichier .dockerignore, permet de dire au docker engine de d'ignorer certains fichiers.

Voici un exemple de Dockerfile. Le code de cette application se trouve les dossiers nginx et dist uniquement.

```
Dockerfile > ...
1  FROM nginx:stable
2  WORKDIR /app
3  ADD ./nginx/default.conf /etc/nginx/conf.d/default.conf
4  ADD ./dist/ /app/
5
```

Mais ce dockerfile se trouve dans un dossier contenant plusieurs autres fichiers assez lourd.



La conséquence est que le chargement de contexte pour la création de cette image prend beaucoup de temps.
On crée donc un fichier .dockerignore pour empêcher le chargement des dossiers inutiles

```
.dockerignore
1  ./node_modules
2  ./src
3  ./e2e
4  ./.git
```

Publier une image docker

Pour publier une image sur le docker hub, il faut avoir créé un compte sur le docker hub.
Le nom de l'image doit avoir la structure suivant :
username/nom_image:tag

docker push username/nom_image:tag

Exemple :
docker login

```
docker push username/nom_image:tag
```

docker-compose

La communauté Docker a mis au point une solution populaire appelée Fig, qui vous permet d'utiliser un seul fichier YAML pour orchestrer tous vos conteneurs et configurations Docker. Cette solution est devenue si populaire que l'équipe de Docker a décidé de créer Docker Compose à partir de la source de Fig, qui est maintenant dépréciée. Docker Compose permet aux utilisateurs d'orchestrer plus facilement les processus des conteneurs Docker, notamment le démarrage, l'arrêt et la configuration de la liaison et des volumes intra-conteneur.

Installation

Cette installation est valable sur ubuntu

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

```
sudo chmod +x /usr/local/bin/docker-compose
```

```
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

Utilisation

docker-compose se base sur un fichier yaml, pour la configuration du déploiement des services.

Un fichier docker-compose comprend entre autre :

- version : c'est le nom de version du fichier yaml
- services : c'est ici qu'on déclarre les applications qui vont être gérées par le compose
- networks : permet de définir un ou plusieurs réseaux applicables aux services. Un réseau peut être de type driver ou external.
  - driver: c'est un réseau créé automatiquement par docker-compose
  - external: c'est un réseau quelconque existant dans l'environnement docker et qu'on décide d'utiliser dans notre configuration.
- volumes : permet de définir un ou plusieurs volumes applicables aux services. Un volume peut être internal ou external.
  - internal : c'est un volume créé automatiquement par docker-compose
  - external: c'est un volume quelconque existant dans l'environnement docker et qu'on décide d'utiliser dans notre configuration.

Exemple :

Pour exemple nous allons écrire un docker-compose yaml pour déployer une application wordpress, mysql et phpmyadmin

commandes

```
Commands:
  build              Build or rebuild services
  config             Validate and view the Compose file
  create             Create services
  down               Stop and remove resources
  events             Receive real time events from containers
  exec               Execute a command in a running container
  help               Get help on a command
  images             List images
  kill               Kill containers
  logs               View output from containers
  pause              Pause services
  port               Print the public port for a port binding
  ps                 List containers
  pull               Pull service images
  push               Push service images
  restart            Restart services
  rm                 Remove stopped containers
  run                Run a one-off command
  scale              Set number of containers for a service
  start              Start services
  stop               Stop services
  top                Display the running processes
  unpause            Unpause services
  up                 Create and start containers
  version            Show version information and quit
```

➢ Démarrer un docker-compose

Si le fichier de configuration s'appelle docker-compose.yml et si on tape la commande de le dossier où il se trouve alors :

docker-compose up

Si on veut spécifier un fichier autre

docker-compose up -f chemin/vers/fichier.yml

➢ Arrêter un docker-compose

docker-compose stop

ou

docker-compose stop -f chemin/vers/fichier.yml

Références

« Docker Documentation | Docker Documentation ». Consulté le 20 novembre 2021. https://docs.docker.com/.

« Qu'est-ce que sont Docker et les Machines Virtuelles ? » Consulté le 20 novembre 2021. https://www.padok.fr/blog/machines-virtuelles-docker.