

Objectifs

Compétences abordées

Ce cours va vous permettre d'acquérir les compétences nécessaires à la compréhension du fonctionnement de Docker et de son utilisation.

Ces compétences vous permettront de :

- mettre en place un environnement de développement pouvant être partagé avec votre équipe ;
- préparer le déploiement d'une application dans un environnement de production basé sur Docker.

Ce dont vous allez avoir besoin

Munissez-vous d'un outil vous permettant d'interagir avec votre machine en ligne de commande (Powershell, Bash, Terminal...).

Vous allez modifier plusieurs fichiers de configuration. Il vous est donc conseillé d'utiliser un éditeur comprenant un outil de coloration syntaxique (votre IDE de prédilection, par exemple).

Bonne croisière !



Un peu de virtualisation

Docker est un système de **virtualisation d'applications**. Avant de pouvoir manipuler Docker il est nécessaire de se questionner sur ce qu'est la virtualisation.

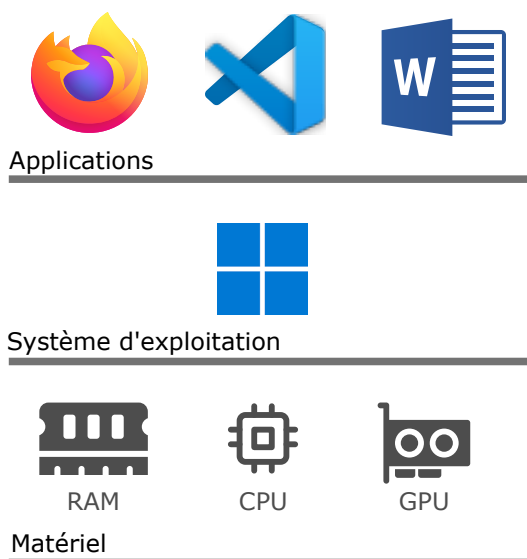
Principe de base

Voici une définition de la virtualisation :

La virtualisation est une technologie que vous pouvez utiliser pour créer des représentations virtuelles de serveurs, de stockage, de réseaux et d'autres machines physiques. Le logiciel virtuel imite les fonctions du matériel physique pour exécuter plusieurs machines virtuelles sur une seule machine physique.

Un article complet provenant du service cloud d'Amazon (AWS) met en avant les [intérêts de la virtualisation](#).

Prenons l'exemple de la machine que vous utilisez au quotidien pour le développement, nous pourrions la représenter comme un ensemble de "couches" :



Descriptif des couches :

- applications : logiciels qui vont démarrer sur la machine
- système d'exploitation : assure le lien entre les ressources matérielles et les applications
- Hardware (machine physique peut être autant un téléphone portable, qu'un ordinateur...)

Une **machine virtuelle** va avoir pour objectif d'encapsuler les couches "applications" et "système d'exploitation". Elle pourra ensuite utiliser les ressources de la machine physique sur laquelle elle est installée via une couche de virtualisation.

Hyperviseur

Cette couche de virtualisation accueille un **hyperviseur**. L'hyperviseur est le logiciel de virtualisation qui sert d'intermédiaire entre les machines virtuelles et le matériel sous-jacent ou le système d'exploitation hôte. L'hyperviseur coordonne l'accès à l'environnement physique. L'hyperviseur fait en sorte de partager les ressources physiques entre plusieurs machines virtuelles.

Par exemple, si la machine virtuelle a besoin de ressources informatiques, comme la puissance de traitement d'un ordinateur, la demande est d'abord adressée à l'hyperviseur. L'hyperviseur transmet ensuite la demande au matériel sous-jacent, qui exécute la tâche.

L'intérêt est donc de pouvoir faire fonctionner plusieurs sous-systèmes sur une même machine physique, ci-dessous une image représentant le principe de fonctionnement :



Plusieurs hyperviseurs existent sur le marché, par exemple :

- Hyper-V sur Windows
- WSL sur Windows (pour virtualiser des machines Linux)
- VirtualBox qui est une solution Oracle
- VMWare

Intérêts de la virtualisation

Question

Selon-vous, quelles pourraient être les avantages et les inconvénients de la virtualisation ?

Votre réflexion pourra être éclairée de recherches.

Questionnaire

Qu'est-ce que Docker ?

Docker est un système de virtualisation qui se veut simple d'utilisation présentant une alternative aux virtualisations basée sur des machines virtuelles (ou "VM") comme détaillées précédemment.

Cette partie vous permettra de mieux appréhender le fonctionnement de Docker.

Vous y étudierez :

- les concepts de conteneurs et d'images
- la façon de mettre en place des images en utilisant des fichiers de configuration `Dockerfile`
- la création de conteneurs composés de plusieurs images en utilisant des fichiers de configuration `docker-compose.yml`

Pour moins de lourdeur, des conteneurs !

Qu'est ce qu'un conteneur ?

Comme présenté précédemment, lors d'une virtualisation utilisant une VM, un hyperviseur vient faire le lien entre la VM et le matériel de la machine physique hôte.

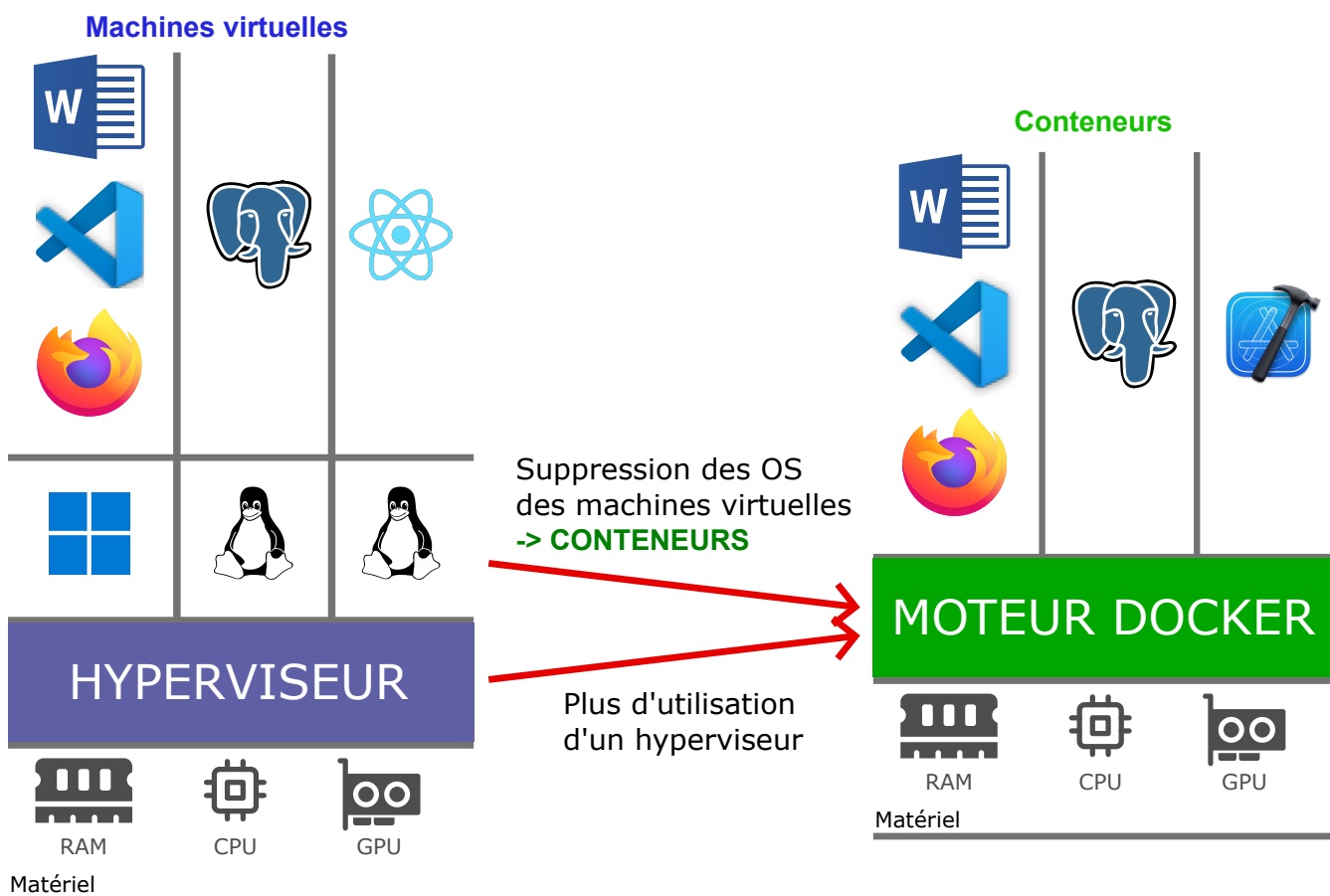
Les hyperviseurs sont des applications lourdes , de plus, les machines virtuelles accueillent des **systèmes d'exploitation complets** qui :

- prennent un espace de stockage important ;
- peuvent être longs à démarrer.

Les équipes de développement Docker ont donc décidé de simplifier l'approche et de baser l'outil sur l'utilisation de conteneurs légers.

Docker ne dépend plus d'un **hyperviseur** et n'a plus de nécessité de reposer sur un système d'exploitation complet.

Il peut être considéré comme une machine virtuelle **minimaliste**. Un conteneur Docker contient uniquement les dossiers et fichiers nécessaires au fonctionnement des applications qu'il contient.



Qu'est-il contenu dans un conteneur ?

Le conteneur contient toutes les installations, dépendances, bibliothèques, processus et codes d'application nécessaires pour faire fonctionner ce qu'il contient.

Un conteneur peut être démarré et arrêté au même titre qu'une application habituelle.

La suite de ce cours vous permettra de comprendre le processus de création de conteneurs avec Docker.

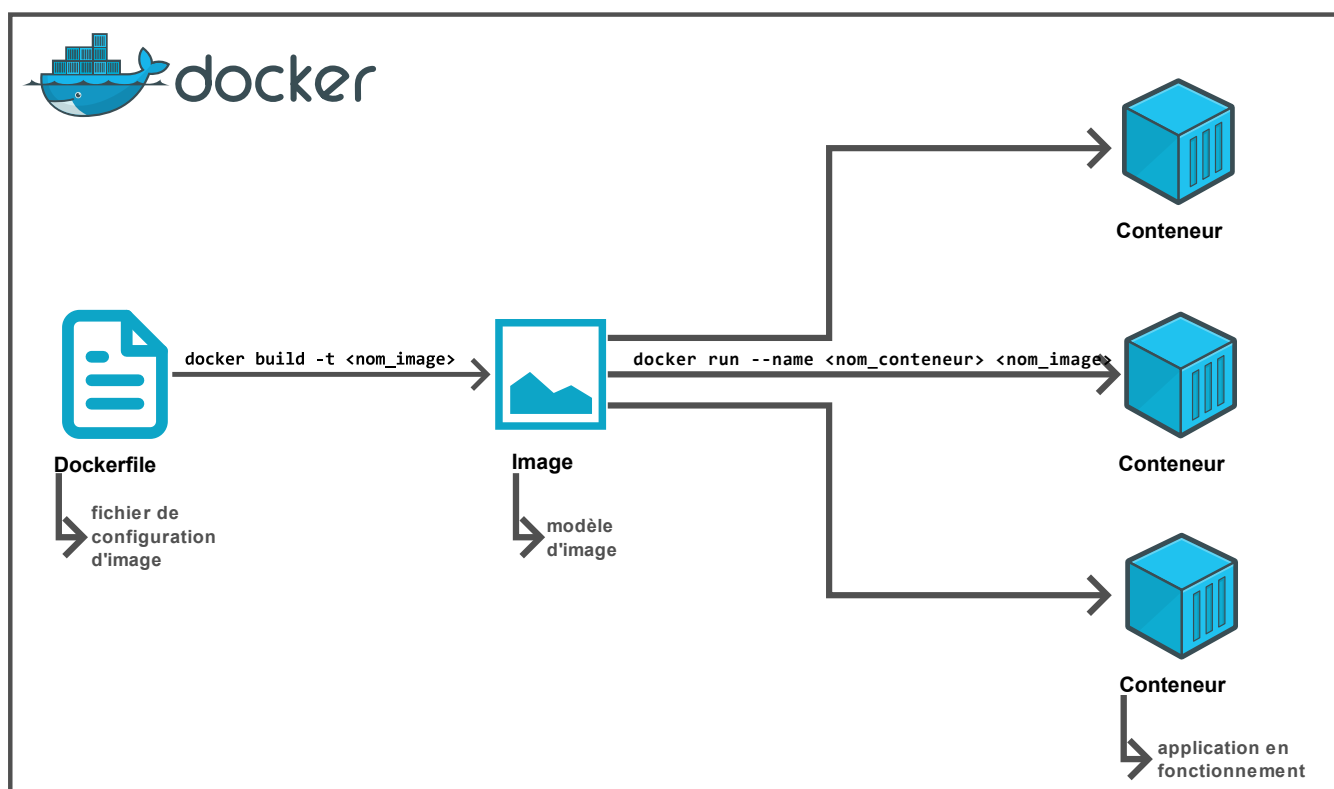
De l'image au conteneur

Construction d'une image

Une image est un **modèle de conteneur**. Autrement dit, le patron de conception d'un conteneur. Un conteneur est considéré comme **une instance d'image**.

Une image peut être configurée en utilisant un **"Dockerfile"** qui est un fichier texte de configuration contenant les instructions nécessaires à son paramétrage. Un **"Dockerfile"** permet donc de créer sa propre image sur-mesure.

Le schéma ci-dessous résume le fonctionnement de création d'un conteneur à partir d'une image :



Banque d'images : Docker Hub

Un ensemble d'images existent et sont utilisables à partir d'une banque d'images nommée [Docker Hub](#).



Il est possible de retrouver un très grand nombre d'applications sur cette banque d'images :

- systèmes de gestion de BDD
- serveurs Web
- conteneur contenant des outils de compilation
- ...

Info

Très souvent il suffit d'utiliser une image pré-existante disponible sur le Docker Hub et de la paramétrer à l'aide d'instructions du fichier Dockerfile.

Avantages de Docker

La distribution et l'utilisation d'un logiciel sous forme d'image Docker vous offrent :

- regroupement de dépendances

Les images Docker contiennent toutes leurs propres dépendances, ce qui signifie que vous n'avez aucune installation à faire vous-même.

- installation multiplateforme

Les conteneurs Docker contiennent sont utilisable sur n'importe quel OS, il suffit que le moteur Docker soit installé sur la machine hôte.

- sécurité

Les fichiers d'un conteneur ne peuvent pas accéder aux fichiers de la machine hôte, de sorte que les utilisateurs peuvent faire confiance aux applications dockerisées.

- facilité d'utilisation

Les conteneurs Docker peuvent toujours être exécutés à l'aide d'une seule commande `docker run`.

- facilités de mise à jour

Les conteneurs Docker peuvent être facilement remplacés par des versions plus récentes, tandis que toutes les données persistantes peuvent être conservées dans un volume de données.

Installation de Docker

Avant de passer à la pratique, vous trouverez dans les pages qui suivent une procédure pour installer Docker sur votre machine.

Procédure d'installation

En suivant la procédure suivante vous pourrez mettre en place un conteneur Docker permettant de déployer (en local ou en ligne en fonction de votre environnement) une base de données PostgreSQL.

Installation Docker Engine + client graphique

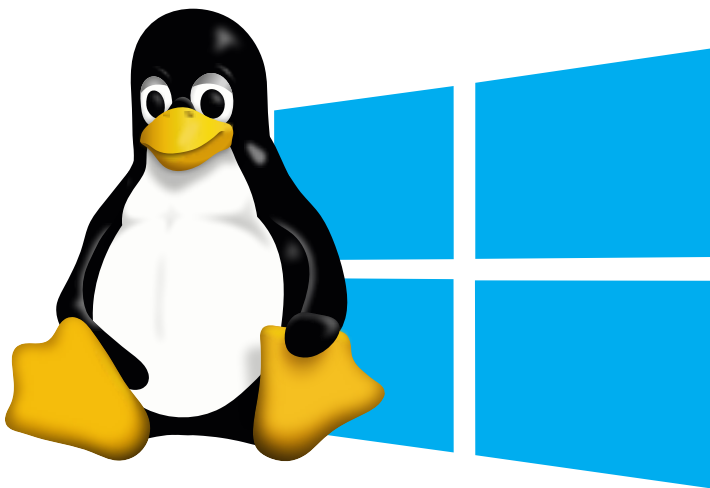
Installez le "Docker Desktop" disponible à l'adresse suivante : <https://www.docker.com/>

Lors de l'installation acceptez l'utilisation du **WSL 2 Backend** et non pas Hyper-V.

Qu'est-ce que WSL-2 ?

WSL-2 est l'acronyme de "Windows Subsystem for Linux", c'est un **noyau Linux** compilé pour Windows. Cet outil permet notamment d'accéder aux **outils et applications Linux** directement depuis l'environnement Windows.

Dans le cas d'une utilisation avec Docker, WSL-2 permet d'exécuter des conteneurs basés sur Linux (un grand nombre de conteneurs le sont).



Lancement de docker

Une fois Docker installé vous pourrez le démarrer via "Docker Desktop". L'application va automatiquement démarrer le moteur de conteneurs Docker (aka. "Docker Engine").

Note

Installez et lancez Docker en cliquant sur "Docker Desktop". Une fenêtre devrait s'ouvrir.

Installation de Docker sous Ubuntu Server

Rien de mieux que la procédure de la documentation officielle Docker : [procédure d'installation de Docker sur Ubuntu](#)



Ubuntu

Les conteneurs en pratique

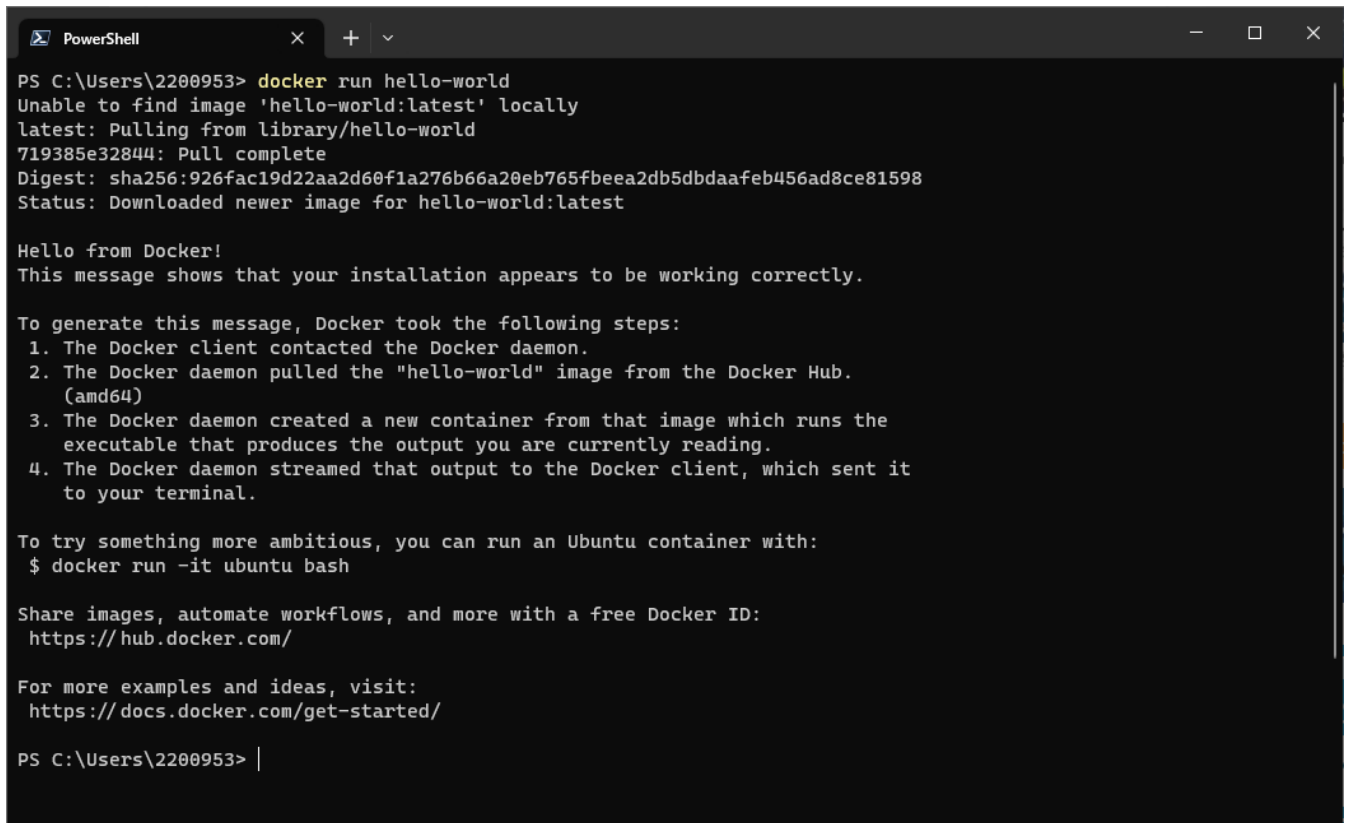
Hello World !

Question

Une fois Docker démarré, ouvrez un Powershell et essayez la commande suivante :

```
docker run hello-world
```

Si tout se passe bien vous devriez obtenir un résultat similaire à la capture suivante :



```
PS C:\Users\2200953> docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
719385e32844: Pull complete
Digest: sha256:926fac19d22aa2d60f1a276b66a20eb765fbee2db5dbdaafeb456ad8ce81598
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/

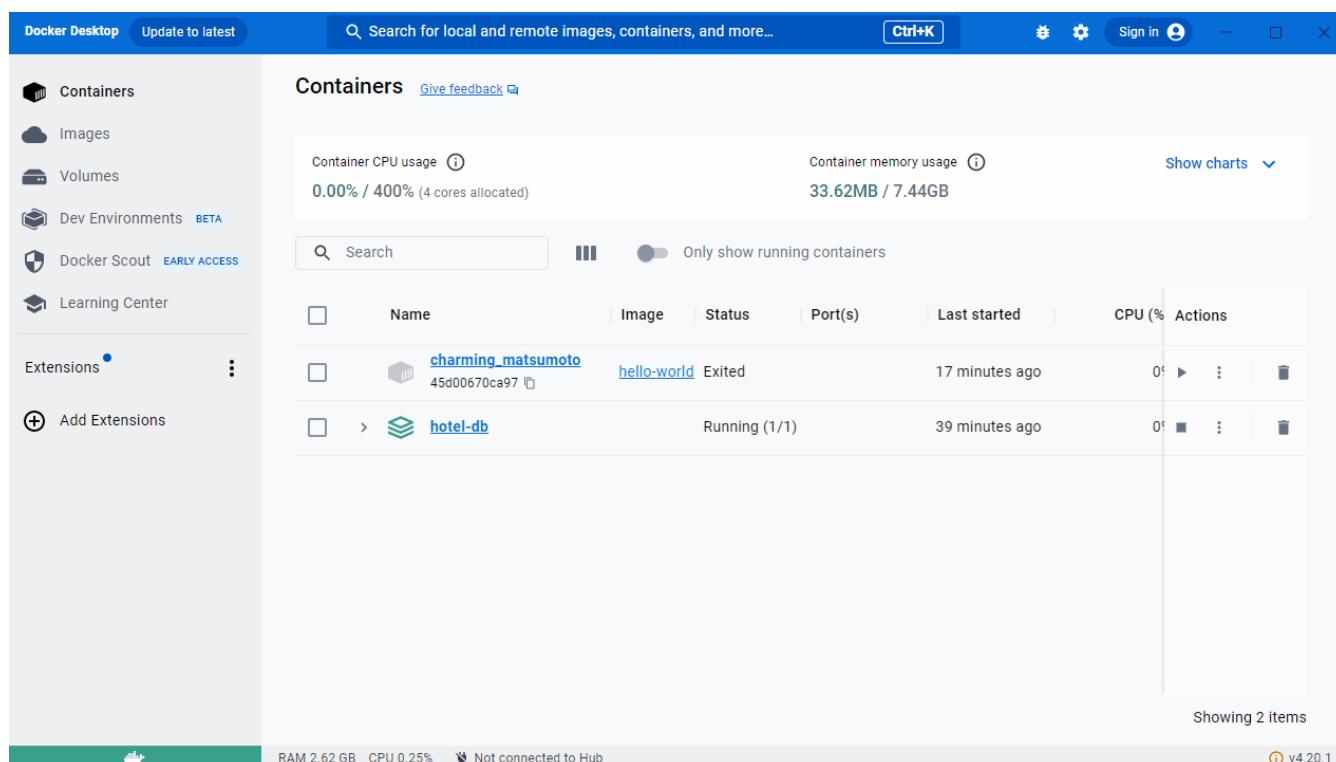
PS C:\Users\2200953> |
```

Attardons nous quelques instants sur le détails des étapes effectuées par Docker :

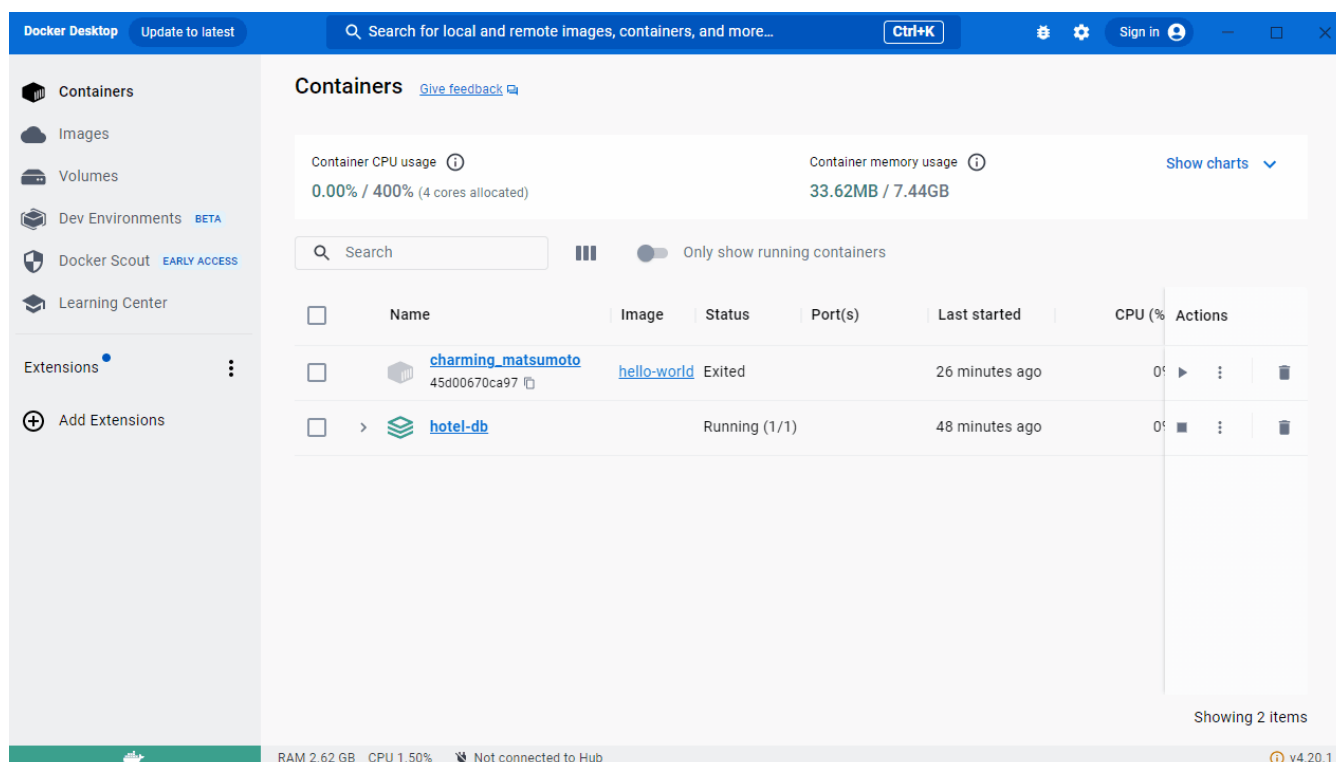
1. Le client Docker (utilisé en console Powershell) est entré en contact avec le "Docker daemon" (aussi appelé "**moteur Docker**")
2. N'ayant pas l'image "hello-world" disponible, le moteur Docker a téléchargé l'image depuis la [banque d'images "Docker Hub"](#).
3. Le moteur Docker a créé un nouveau conteneur à partir de l'image. Une fois démarré le conteneur a produit le texte.
4. Le moteur Docker a transmis le flux de données (le texte produit) au client Docker (la console Powershell).

Analyse du conteneur "hello-world"

Il est possible d'analyser le contenu d'un container grâce à l'onglet "Containers" de Docker Desktop. Le gif suivant présente la façon d'accéder aux "logs" d'un conteneur (vous remarquerez qu'il s'agit de ce qui est redirigé vers la console Powershell).



Les dossiers et fichiers contenus dans un conteneur peuvent également être visibles via le client Docker :



Dans le cas du conteneur basé sur une image "hello-world" le fichier contient une partie de code non compréhensible mais également le fameux texte affiché en console.

Info

Il est possible d'afficher les conteneurs en cours d'exécution grâce à la ligne de commande : `docker container ls`

Manipulation d'images

Il vous est possible de manipuler les images disponibles sur une installation de Docker de deux façons :

- en ligne de commande ;
- via l'onglet "Images" du client graphique "Docker Desktop".

Vous trouverez ici les commandes les plus utilisées.

Liste des images

Info

Commande : `docker images`

Suppression d'une image

Info

Commande : `docker image rm <nom_image>`

Question

Essayez de supprimer l'image intitulée "hello-world". Que se passe t-il ?

Pour arriver à vos fins, il vous faudra donc utiliser une autre commande :

`docker container rm <id_conteneur>`

Question

Supprimez le conteneur puis supprimez l'image "hello-world".

Configurer une image : Dockerfile

Jusqu'à présent nous avons vu comment simplement utiliser une image disponible sur le Docker Hub (cas du "hello-world").

Nous avons vu que l'image "hello-world" contenait le code nécessaire pour afficher un message de bienvenue puis s'arrêtait. Comportement plutôt limité.

Rappelons l'objectif : embarquer dans une image l'ensemble des fichiers nécessaires au fonctionnement d'une application (base de données, site internet, application desktop, tout est possible).

Il va donc nous falloir un mécanisme permettant de configurer une image : le Dockerfile.

L'image "docker/whalesay"

Dans la suite de cette partie nous utiliserons l'image [docker/whalesay](#).

Cette image embarque une version de [Ubuntu](#) sur laquelle est installé le programme [cowsay](#) qui fait parler une vache ASCII (pas de question, s'il vous plaît).

Dans le cas de l'image [docker/whalesay](#) la vache a été remplacée par une baleine porte-conteneurs (toujours pas de question).

Question

Tapez la commande `docker run docker/whalesay` . Que se passe-t-il ?

Pour faire parler la baleine, vous pouvez ajouter une commande `cowsay` de la façon suivante :

```
docker run docker/whalesay cowsay "Booya!"
```

Il n'est pas très pratique de devoir tout le temps souffler à la baleine ce qu'il faut dire. Dans la suite de cette partie nous allons créer un fichier de configuration "Dockerfile" pour automatiser la commande "cowsay".

Création d'un Dockerfile

Question

Créez dans un nouveau dossier nommé "**whalesay**" un fichier nommé "**Dockerfile**" (attention, la casse a son importance. De plus, il n'y a pas d'extension).

Ajoutez au fichier les instructions suivantes :

```
FROM docker/whalesay
CMD cowsay "Hey ! Je sais quoi dire !"
```

Détails sur les instructions utilisées :

- **FROM** : indique l'image de base à utiliser pour la nouvelle image que nous souhaitons créer
- **CMD** : processus appelé dans l'image

Il est maintenant possible de créer une nouvelle image basée sur "docker/whalesay" en utilisant la commande suivante :

```
docker build -t thatsmyswhale .
```

L'option `-t` permet de définir le nom de l'image qui sera créée.

Si tout se passe bien, une nouvelle image nommée "thatmyswhale" devrait être créée.

Question

Quelle serait la commande à utiliser pour démarrer un conteneur à partir de cette image ?

La baleine commence à être de plus en plus intéressante, faisons maintenant en sorte de lui faire dire des messages aléatoires. Nous utiliserons l'outil "**fortune**" qui affiche des citations.

Changez le contenu du fichier "Dockerfile" de la façon suivante :

```
FROM docker/whalesay
RUN apt-get -y update
RUN apt-get install -y fortunes
CMD /usr/games/fortune | cowsay
```

Question

Contruisez à nouveau l'image et démarrez un conteneur.

Vous venez d'apprendre à construire une image sur-mesure à partir d'un fichier "Dockerfile".


Vous verrez par la suite comment faire en sorte de créer des images pour embarquer des serveurs web ou encore des applications React (mais ceci n'est pas tant différent que de faire parler des baleines).

Composition d'images : Docker compose

Commandes utiles

Il est possible d'interagir avec le démon Docker en utilisant un ensemble de commandes.

Voici un récapitulatif des commandes les plus utilisées :

 Cheatsheet for Docker CLI			
Run a new Container	Manage Containers	Manage Images	Info & Stats
<p>Start a new Container from an Image</p> <pre>docker run IMAGE docker run nginx</pre> <p>...and assign it a name</p> <pre>docker run --name CONTAINER IMAGE docker run --name web nginx</pre> <p>...and map a port</p> <pre>docker run -p HOSTPORT:CONTAINERPORT IMAGE docker run -p 8080:80 nginx</pre> <p>...and map all ports</p> <pre>docker run -P IMAGE docker run -P nginx</pre> <p>...and start container in background</p> <pre>docker run -d IMAGE docker run -d nginx</pre> <p>...and assign it a hostname</p> <pre>docker run --hostname HOSTNAME IMAGE docker run --hostname srv nginx</pre> <p>...and add a dns entry</p> <pre>docker run --add-host HOSTNAME:IP IMAGE</pre> <p>...and map a local directory into the container</p> <pre>docker run -v HOSTDIR:TARGETDIR IMAGE docker run -v ~/.usr/share/nginx/html nginx</pre> <p>...but change the entrypoint</p> <pre>docker run -it --entrypoint EXECUTABLE IMAGE docker run -it --entrypoint bash nginx</pre>	<p>Show a list of running containers</p> <pre>docker ps</pre> <p>Show a list of all containers</p> <pre>docker ps -a</pre> <p>Delete a container</p> <pre>docker rm CONTAINER docker rm web</pre> <p>Delete a running container</p> <pre>docker rm -f CONTAINER docker rm -f web</pre> <p>Delete stopped containers</p> <pre>docker container prune</pre> <p>Stop a running container</p> <pre>docker stop CONTAINER docker stop web</pre> <p>Start a stopped container</p> <pre>docker start CONTAINER docker start web</pre> <p>Copy a file from a container to the host</p> <pre>docker cp CONTAINER:SOURCE TARGET docker cp web:/index.html index.html</pre> <p>Copy a file from the host to a container</p> <pre>docker cp TARGET CONTAINER:SOURCE docker cp index.html web:/index.html</pre> <p>Start a shell inside a running container</p> <pre>docker exec -it CONTAINER EXECUTABLE docker exec -it web bash</pre> <p>Rename a container</p> <pre>docker rename OLD_NAME NEW_NAME docker rename 096 web</pre> <p>Create an image out of container</p> <pre>docker commit CONTAINER docker commit web</pre>	<p>Download an image</p> <pre>docker pull IMAGE[:TAG] docker pull nginx</pre> <p>Upload an image to a repository</p> <pre>docker push IMAGE docker push myimage:1.0</pre> <p>Delete an image</p> <pre>docker rmi IMAGE</pre> <p>Show a list of all Images</p> <pre>docker images</pre> <p>Delete dangling images</p> <pre>docker image prune</pre> <p>Delete all unused images</p> <pre>docker image prune -a</pre> <p>Build an image from a Dockerfile</p> <pre>docker build DIRECTORY docker build .</pre> <p>Tag an image</p> <pre>docker tag IMAGE NEWIMAGE docker tag ubuntu ubuntu:18.04</pre> <p>Build and tag an image from a Dockerfile</p> <pre>docker build -t IMAGE DIRECTORY docker build -t myimage .</pre> <p>Save an image to .tar file</p> <pre>docker save IMAGE > FILE docker save nginx > nginx.tar</pre> <p>Load an image from a .tar file</p> <pre>docker load -i TARFILE docker load -i nginx.tar</pre>	<p>Show the logs of a container</p> <pre>docker logs CONTAINER docker logs web</pre> <p>Show stats of running containers</p> <pre>docker stats</pre> <p>Show processes of container</p> <pre>docker top CONTAINER docker top web</pre> <p>Show installed docker version</p> <pre>docker version</pre> <p>Get detailed info about an object</p> <pre>docker inspect NAME docker inspect nginx</pre> <p>Show all modified files in container</p> <pre>docker diff CONTAINER docker diff web</pre> <p>Show mapped ports of a container</p> <pre>docker port CONTAINER docker port web</pre>

Questionnaire

Cas d'utilisation réel

Vous retrouverez dans cette partie un ensemble de fichiers de configuration permettant de "Dockeriser" des services communs :

- [base de données PostgreSQL](#)
- [application React](#)
- [microservice Spring Boot](#)

Dockeriser un SGBDR PostgreSQL

Descriptif du Dockerfile

Pour un système de gestion de base de données PostgreSQL, vous pouvez construire un Dockerfile en vous basant sur l'image officielle "postgres" [disponible sur le Hub Docker](#).

Voici le "Dockerfile" pouvant être utilisé (à adapter à votre case d'utilisation)

```
# Image Docker sur laquelle est basée la nouvelle image que nous allons créer
FROM postgres:15
# Utilisateur "administrateur" du système de gestion de base de données, par
# convention nous laisserons "postgres"
ENV POSTGRE_USER postgres
# Mot de passe (complexe, s'il vous plait) de l'utilisateur administrateur
ENV POSTGRES_PASSWORD <mot-de-passe-super-complexe>
# Création d'une base de données avec un nom prédéfini : "society"
ENV POSTGRES_DB <nom-base-de-données>
# Copie du fichier de création de BDD dans l'image
# ce script sera démarré automatiquement au lancement du conteneur
COPY <script-de-cr ation-bdd> /docker-entrypoint-initdb.d/
```

Ci-dessous, une pr cision de l'utilit  de chaque commande :

- **FROM** : cr ation d'une image **bas e sur une image existante**
- **ENV** : variable d'environnement qui pourront  tre utilis es par le conteneur
- **COPY** : copie un fichier de la machine physique dans l'image Docker

L'instruction **COPY** va utiliser un chemin relatif pour retrouver le fichier   copier dans "/docker-entrypoint-initdb.d/", ceci   partir du dossier dans lequel est contenu le "Dockerfile".

Il vous faut donc mettre le "Dockerfile" et le fichier de script de cr ation de BDD dans le m me dossier.

Cr ation de l'image

1. ouvrir un terminal
2. se positionner dans le dossier contenant le Dockerfile.
3. utiliser la commande cr ation d'image :

```
docker build -t <nom-image> .
```

3. vérifier la bonne construction de l'image :

```
docker image ls
```

Gestion du conteneur

Instanciation

Pour instancier un conteneur vous pouvez utiliser la commande suivante :

```
docker run --name <nom-conteneur> -p 127.0.0.1:5432:5432/tcp <nom-image>
```

- `run` : instruction de démarrage d'un conteneur à partir d'une image
- `--name <nom-conteneur>` : permet de paramétrer le nom qui sera donné au conteneur
- `-p` : paramètre de redirection de port
 - `127.0.0.1` : adresse de la machine locale
 - `5432:5432` : redirection de port. Le port 5432 de la machine locale sera redirigé vers le 5432 du conteneur
 - `/tcp` : protocole de communication réseau impacté par la redirection

Il est possible de lancer le conteneur en mode "détaché" pour reprendre la main sur le terminal en ajoutant le paramètre `-d` à la commande.

Pour vérifier que l'instanciation du conteneur s'est bien déroulée, vous pouvez utiliser la commande :

```
docker container ls -a
```

L'option `-a` permet d'afficher tous les conteneurs, même ceux qui **ne sont pas démarrés**.

Arrêt du conteneur

Un conteneur est identifié par un identifiant qui peut être retrouvé grâce à la commande de listing présentée précédemment :

```
PS C:\Users\2200953\Workspace\CDA\08-bdd\00-code\databases\society\postgresql> docker container ls
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS              PORTS                               NAMES
a121a8980e1f   society-db     "docker-entrypoint.s..." 4 minutes ago  Up 8 seconds       127.0.0.1:5432->5432/tcp           society-db
1f212fc6024a   nginx:alpine   "/docker-entrypoint..." 4 days ago    Restarting (1) 3 seconds ago      hostel-station-nginx-1
```

↑
Identifiant à utiliser
pour gérer les conteneurs

Une fois l'identifiant du conteneur récupéré vous pourrez le stopper avec la commande :

```
docker container stop <id-conteneur>
```

Démarrage d'un conteneur

Si le conteneur a déjà été instancié une première fois il est possible de le redémarrer :

```
docker container start <id-conteneur>
```