



Chapitre 2 : Docker Initiation

Docker et Kubernetes

CCI Strasbourg

Chapitre 2 : Docker initiation

Objectifs

En suivant ce chapitre, nous allons aborder Docker avec les modules suivants :

- 2.1 Installation et configuration
- 2.2 Concepts fondamentaux : Images, conteneurs, registres
- 2.3 Dockerfile : Ecriture, build et publication
- 2.4 Docker Compose : Définir et exécuter des applications multi-conteneurs

Chapitre 2 : Docker initiation

Module 2.1

Installation et configuration

Chapitre 2 : Docker initiation

Module 2.1 – Installation et configuration

Sommaire :

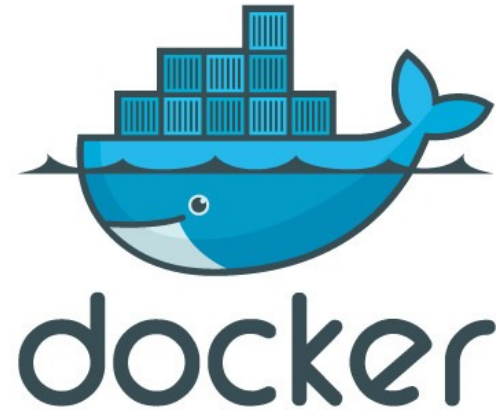
- Présentation de Docker
- Installation et configuration (TP)



Chapitre 2 : Docker initiation

Module 2.1 – Installation et configuration

Présentation de Docker

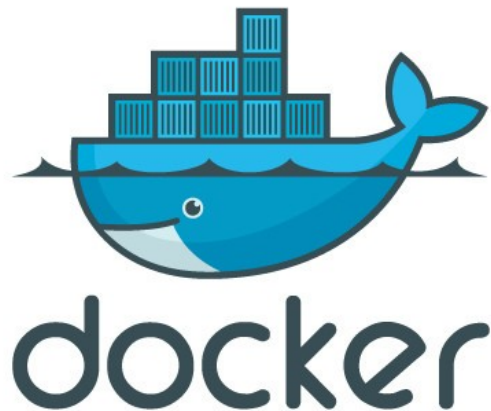


Chapitre 2 : Docker initiation

Module 2.1 – Présentation de Docker

Docker est un logiciel open source qui facilite le déploiement, la gestion et l'exécution des conteneurs.

Docker est basé sur la technologie LXC (Linux Containers).



Chapitre 2 : Docker initiation

Module 2.1 – Présentation de Docker

Pour rappel, en tant que solution qui offre la possibilité de gérer des conteneurs, voici les avantages d'utiliser Docker :

- « Virtualisation » de l'OS en utilisant des images
- Meilleures performances que la virtualisation classique
- Déploiement rapide et facile d'applications

Chapitre 2 : Docker initiation

Module 2.1 – Présentation de Docker

Docker fournit notamment :

- Docker Engine : il s'agit du moteur de Docker qui va effectuer les actions sur les conteneurs. → si on installe Docker sur un serveur, on installera uniquement le moteur.
- Docker Desktop : il s'agit d'une interface utilisateur qui simplifie la gestion des dockers → généralement sur un poste de développeur

Chapitre 2 : Docker initiation

Module 2.1 – Présentation de Docker

Docker Hub est une plateforme cloud de partage de conteneurs (registre) qui permet aux développeurs de stocker, partager et distribuer des applications.

Il existe différentes offres qui permet aux utilisateurs d'avoir plus ou moins de fonctionnalités (image privée, etc...)

Chapitre 2 : Docker initiation

Module 2.1 – Installation et configuration de Docker

TP : Installer DockerDesktop sur votre poste.

- Se rendre sur <https://docs.docker.com/get-docker/> et suivre la section adaptée à votre OS pour installer Docker.
- Une fois DockerDesktop installé, vous pouvez vous connecter à un compte DockerHub (créer un compte si vous n'en avez pas déjà).

Pour les utilisateurs Linux, pour se connecter à DockerHub depuis Docker Desktop, il faut d'abord générer une clé, cf

<https://docs.docker.com/desktop/get-started/#credentials-management-for-linux-users>

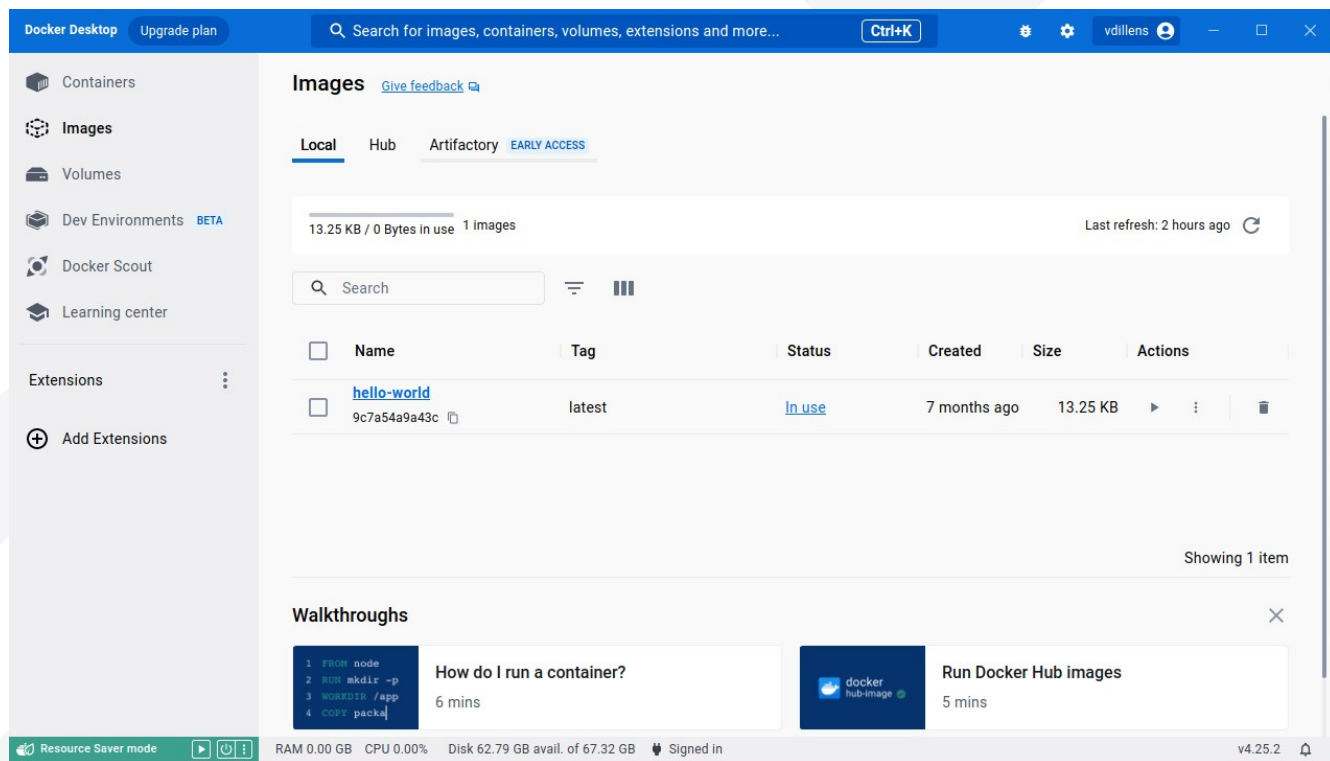
Remarques : vous pouvez installer Docker sur un OS différent de votre système en utilisant une VM.

Chapitre 2 : Docker initiation

Module 2.1 – Installation et configuration de Docker

TP : Installer DockerDesktop sur votre poste.

Résultat



```
vincent@ccipc1:~$ docker --version  
Docker version 24.0.7, build afdd53b
```

Chapitre 2 : Docker initiation

Module 2.1 - Questions ?



Chapitre 2 : Docker initiation

Module 2.2

Concepts fondamentaux : Images, conteneurs, registres

Chapitre 2 : Docker initiation

Module 2.2 – Concepts fondamentaux : Images, conteneurs, registres

Sommaire :

- Les images dans Docker
- Les conteneurs avec Docker
- Les registres
- Manipulation de Docker (TP)



Chapitre 2 : Docker initiation

Les images dans Docker

Chapitre 2 : Docker initiation

Module 2.2 – Les images dans Docker

Les images sont des éléments importants dans Docker, elles permettent de livrer / déployer un ensemble cohérent (code, librairies, dépendances, configuration, etc.).

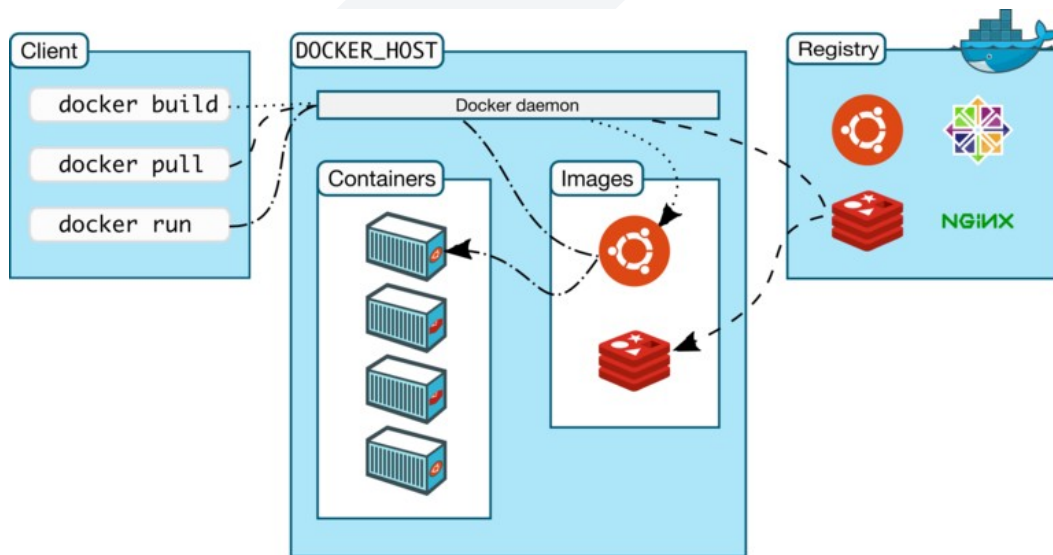
Comment obtenir des images dans Docker ?

- Les images sont stockées dans des registres, Docker Hub est le registre public officiel de Docker. Pour télécharger une image en local sur son poste, il existe une commande « `docker pull` », pour ensuite créer un conteneur depuis une image il faudra utiliser une commande « `docker run` »
- Les images peuvent aussi être construites depuis des conteneurs de notre système hôte via une commande « `docker commit` ». Dans le principe, l'instance actuelle du conteneur sera figé et stocké sous forme d'image réutilisable.
- Une autre méthode pour construire une image va être via un fichier DockerFile (abordé plus loin dans le cours) et la commande « `docker build` ».

Chapitre 2 : Docker initiation

Module 2.2 – Les images dans Docker

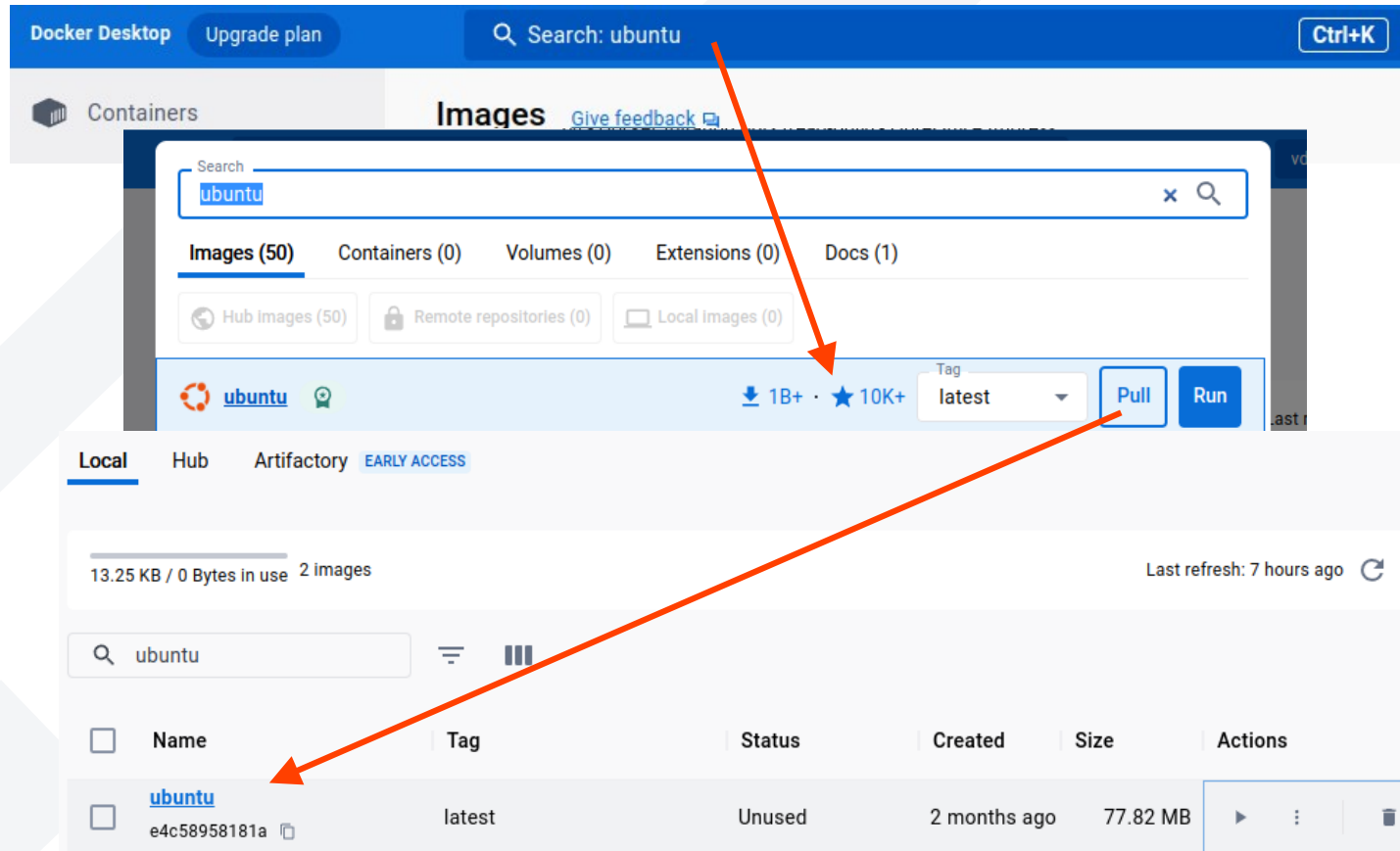
Le schéma ci-dessous illustre une architecture Docker et la notion d'images :



Chapitre 2 : Docker initiation

Module 2.2 – Les images dans Docker

Récupérer une image : Exemple d'un « pull » d'une image avec Docker Desktop



Chapitre 2 : Docker initiation

Module 2.2 – Les images dans Docker

Récupérer une image : Exemple d'un « pull » d'une image avec Docker

```
vincent@ccipc1:~$ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
aece8493d397: Pull complete
Digest: sha256:2b7412e6465c3c7fc5bb21d3e6f1917c167358449fecac8176c6e496e5c1f05f
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview ubuntu
```

Chapitre 2 : Docker initiation

Module 2.2 – Les images dans Docker

Lister les images : Exemple avec Docker Desktop

Directement dans le dashboard, depuis le menu « Images »

Docker Desktop Upgrade plan Search for images, containers, volumes, extensions and more...

Containers
Images
Volumes
Dev Environments **BETA**
Docker Scout
Learning center

Extensions
Add Extensions

Images [Give feedback](#)

Local Hub Artifactory **EARLY ACCESS**

13.25 KB / 0 Bytes in use 2 Images

Search

<input type="checkbox"/>	Name	Tag	Status
<input type="checkbox"/>	ubuntu e4c58958181a	latest	Unused
<input type="checkbox"/>	hello-world 9c7a54a9a43c	latest	In use

Chapitre 2 : Docker initiation

Module 2.2 – Les images dans Docker

Lister les images : Exemple avec Docker

L'utilisation de la commande « docker images » permet de lister les images présentes en local

```
vincent@ccipc1:~$ docker images
REPOSITORY    TAG       IMAGE ID      CREATED        SIZE
ubuntu        latest    e4c58958181a  7 weeks ago   77.8MB
hello-world    latest    9c7a54a9a43c  6 months ago  13.3kB
vincent@ccipc1:~$
```

Chapitre 2 : Docker initiation

Les conteneurs avec Docker

Chapitre 2 : Docker initiation

Module 2.2 – Les conteneurs dans Docker

Les conteneurs vont être des instances des images, ces conteneurs vont vivre tant que la commande est en cours.

Cela signifie que si la commande se termine, le conteneur va s'arrêter automatiquement.

Si la commande du conteneur est un service / daemon qui tourne continuellement, alors le conteneur va continuer de s'exécuter sauf si on lui donne l'ordre de s'arrêter.

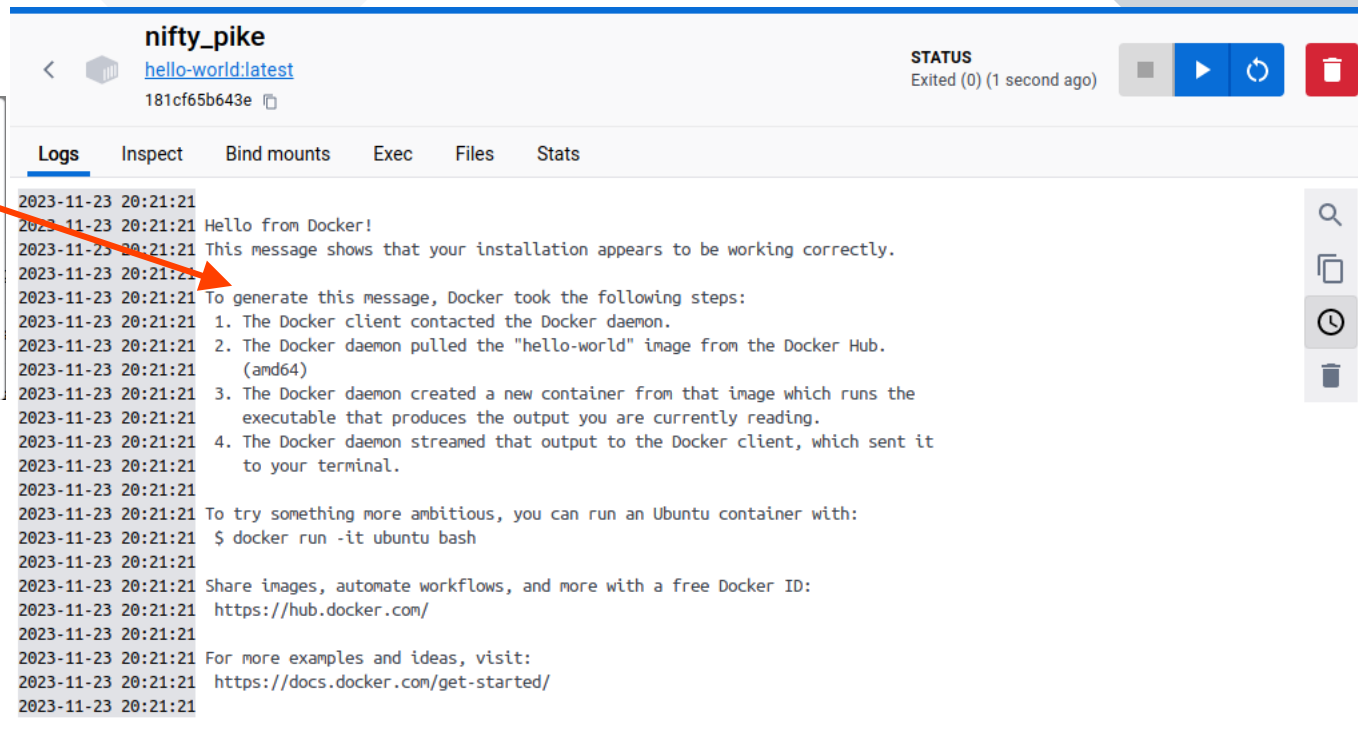
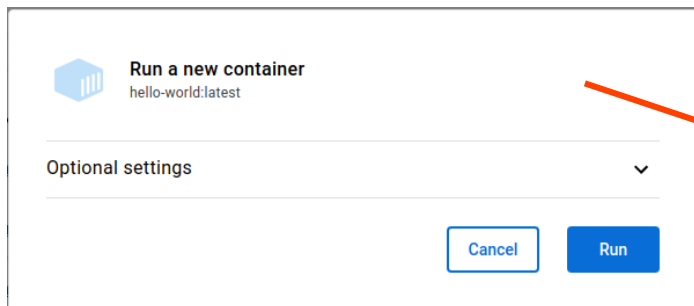
Comment créer des conteneurs dans Docker ?

Les docker peuvent se lancer de plusieurs manières, la première que nous allons voir est la commande « docker run ».

Chapitre 2 : Docker initiation

Module 2.2 – Les conteneurs dans Docker

Créer un conteneur simple « hello-world » avec Docker Desktop



Chapitre 2 : Docker initiation

Module 2.2 – Les conteneurs dans Docker

Créer un conteneur simple « hello-world » avec Docker

```
vincent@ccipc1:~$ docker run hello-world
```

```
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/
```

Chapitre 2 : Docker initiation

Module 2.2 – Les conteneurs dans Docker

Créer un conteneur persistant avec Docker Desktop, exemple avec nginx

 **Run a new container**
nginx:latest

Optional settings ^

Container name

A random name is generated if you do not provide one.

Ports
Enter "0" to assign randomly generated host ports.

Host port
 :80/tcp

Volumes

Host path

Container path

Environment variables

Variable

Value

nginx_example

[nginx:latest](#)
8f8624185af1
[65111:80](#)

STATUS
Running (2 minutes ago)

Logs Inspect Bind mounts Exec Files Stats

```
2023-11-23 20:16:47 /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
2023-11-23 20:16:47 /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
2023-11-23 20:16:47 /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
2023-11-23 20:16:47 10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
2023-11-23 20:16:47 10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
2023-11-23 20:16:47 /docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
2023-11-23 20:16:47 /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
2023-11-23 20:16:47 /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
2023-11-23 20:16:47 /docker-entrypoint.sh: Configuration complete; ready for start up
2023-11-23 20:16:48 2023/11/23 19:16:48 [notice] 1#1: using the "epoll" event method
2023-11-23 20:16:48 2023/11/23 19:16:48 [notice] 1#1: nginx/1.25.3
2023-11-23 20:16:48 2023/11/23 19:16:48 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2023-11-23 20:16:48 2023/11/23 19:16:48 [notice] 1#1: OS: Linux 6.4.16-linuxkit
2023-11-23 20:16:48 2023/11/23 19:16:48 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2023-11-23 20:16:48 2023/11/23 19:16:48 [notice] 1#1: start worker processes
2023-11-23 20:16:48 2023/11/23 19:16:48 [notice] 1#1: start worker process 29
2023-11-23 20:16:48 2023/11/23 19:16:48 [notice] 1#1: start worker process 30
2023-11-23 20:18:46 172.17.0.1 - - [23/Nov/2023:19:18:46 +0000] "GET / HTTP/1.1" 200 37 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.0.0 Safari/537.36" "-"
2023-11-23 20:18:47 2023/11/23 19:18:47 [error] 29#29: *1 open() "/usr/share/nginx/html/favicon.ico" failed (2: No such file or directory), client: 172.17.0.1, server: localhost, request: "GET /favicon.ico HTTP/1.1", host: "localhost:65111", referer: "http://localhost:65111/"
2023-11-23 20:18:47 172.17.0.1 - - [23/Nov/2023:19:18:47 +0000] "GET /favicon.ico HTTP/1.1" 404 555 "http://localhost:65111/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.0.0 Safari/537.36" "-"
```

Chapitre 2 : Docker initiation

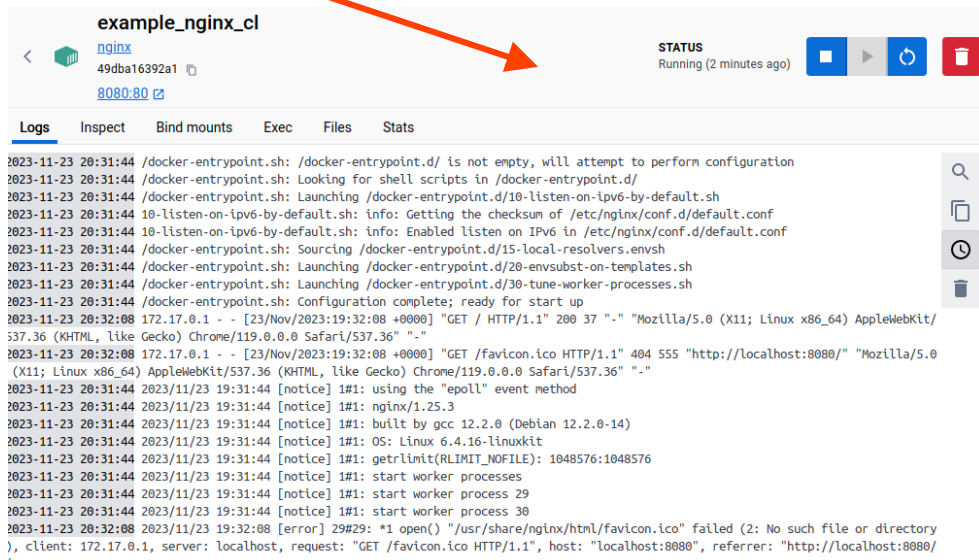
Module 2.2 – Les conteneurs dans Docker

Créer un conteneur persistant avec Docker, exemple avec nginx

La commande docker run retourne l'identifiant du conteneur qui a été créé.

La commande ci-dessous est l'équivalent de l'exemple précédent avec Docker Desktop.

```
vincent@ccipc1:~$ docker run --name example_nginx_cl -v /home/vincent/nginx_volume:/usr/share/nginx/html:ro -d -p 8080:80 nginx  
49dba16392a1e6d6bd47de3244a6efbadff448d669200fd905aabd2247702c6d
```

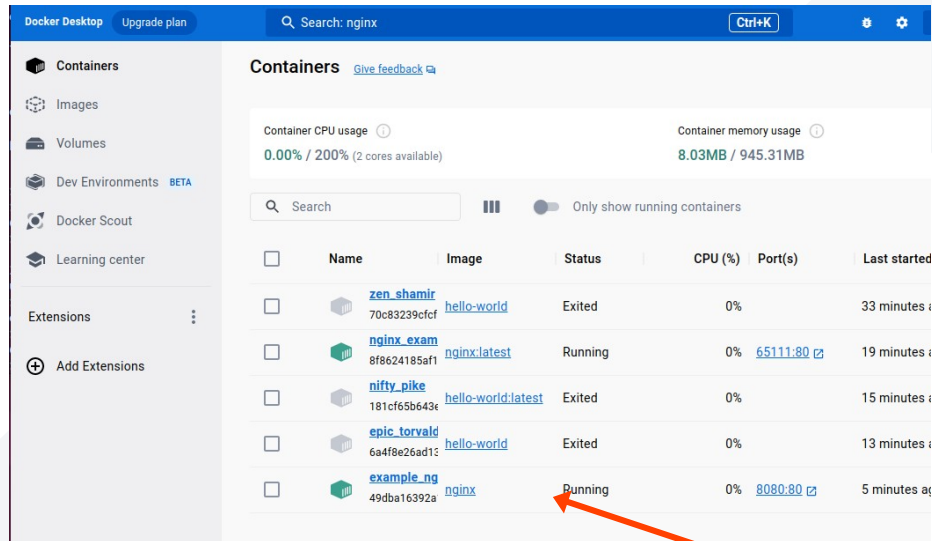


The screenshot shows the Docker Desktop interface for a container named **example_nginx_cl**. The container is running, as indicated by the status "Running (2 minutes ago)". The container ID is **49dba16392a1e6d6bd47de3244a6efbadff448d669200fd905aabd2247702c6d**. The container is mapped to port **8080:80**. The logs show the container's startup sequence, including the nginx configuration and the start of the worker processes. The logs also show an error message: `*1 open() "/usr/share/nginx/html/favicon.ico" failed (2: No such file or directory`.

Chapitre 2 : Docker initiation

Module 2.2 – Les conteneurs dans Docker

Lister les conteneurs



Dans Docker Desktop, la liste des conteneurs est obtenu depuis le menu « Containers ».

La commande « docker ps » permet d'afficher la liste des conteneurs

```
vincent@ccipc1:~$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
49dba16392a1   nginx         "/docker-entrypoint...." 5 minutes ago  Up 5 minutes  0.0.0.0:8080->80/tcp               example_nginx_cl
8f8624185af1   nginx:latest  "/docker-entrypoint...." 20 minutes ago Up 20 minutes  0.0.0.0:65111->80/tcp             nginx_example

vincent@ccipc1:~$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
49dba16392a1   nginx         "/docker-entrypoint...." 5 minutes ago  Up 5 minutes  0.0.0.0:8080->80/tcp               example_nginx_cl
6a4f8e26ad13   hello-world   "/hello"                13 minutes ago Exited (0) 13 minutes ago          epic_torvalds
181cf65b643e   hello-world:latest "/hello"                15 minutes ago Exited (0) 15 minutes ago          nifty_pike
8f8624185af1   nginx:latest  "/docker-entrypoint...." 20 minutes ago Up 20 minutes  0.0.0.0:65111->80/tcp             nginx_example
70c83239cfcf   hello-world   "/hello"                24 hours ago  Exited (0) 33 minutes ago          zen_shamir
```

Chapitre 2 : Docker initiation

Les registres

Chapitre 2 : Docker initiation

Module 2.2 – Les registres

Les registres (registries) sont des services qui stockent et distribuent des images Docker. Nous pouvons voir les registres comme des armoires qui stockent les images qui vont servir de base aux conteneurs.

Le registre par défaut est Docker Hub (registre officiel public de Docker).

Il existe ensuite la notion de registre public et privé. Dans le cadre d'une entreprise, vous pouvez utiliser des **registres privés** pour stocker vos propres images Docker. Cela peut être utile si vous avez des **images sensibles ou spécifiques à votre organisation**.

Quelques exemples de registres privés populaires : Amazon Elastic Container Registry (ECR), Google Container Registry (GCR) et Azure Container Registry (ACR).

Chapitre 2 : Docker initiation

Module 2.2 – Les registres

Pour interagir avec un registre, il existe 2 commandes : pull et push.

Pour partager une image avec d'autres personnes, il est nécessaire de la publier sur le registre, il faudra alors effectuer un **docker push utilisateur/nom_image:tag**

Pour récupérer une image il faudra effectuer **un docker pull utilisateur/nom_image:tag**

Chapitre 2 : Docker initiation

Manipulation de Docker (TP)

Chapitre 2 : Docker initiation

Module 2.2 – Manipulation de Docker (TP)

Suivre le TP fournit par le formateur.

Chapitre 2 : Docker initiation

Module 2.3 – Dockerfile : Écriture, build et publication

Sommaire :

- Présentation de Dockerfile
- Ecriture d'un Dockerfile
- Construire une image via un Dockerfile
- Publier un Dockerfile
- Utiliser Dockerfile (TP)



Chapitre 2 : Docker initiation

Présentation de Dockerfile

Chapitre 2 : Docker initiation

Module 2.3 – Présentation de Dockerfile

Un Dockerfile est un simple fichier texte qui comporte une série d'instructions permettant de créer une image Docker. Pour se faciliter la vie, on nomme généralement ce fichier « Dockerfile » car il s'agit du nom par défaut utilisé dans les commandes.

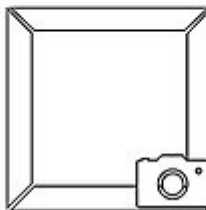
Via une commande « docker build », Docker va lire et interpréter les instructions pour construire une image, qui va ensuite pouvoir être utilisée pour le lancement de conteneurs.

1. Dockerfile



Build

2. Docker Image



Run

3. Docker Container(s)



Chapitre 2 : Docker initiation

Écriture d'un Dockerfile

Chapitre 2 : Docker initiation

Module 2.3 – Écriture d'un Dockerfile

Voici un premier exemple de fichier Dockerfile et le résultat de la commande docker build (l'option -t permet de préciser le nom de l'image à créer) :

```
FROM debian

RUN apt-get update -y
RUN apt-get upgrade -y
RUN apt-get install nano curl -y
```

```
FROM debian

RUN apt-get update -y \
&& apt-get upgrade -y \
&& apt-get install nano curl -y
```

docker build -t premier_exemple .

```
[+] Building 34.2s (8/8) FINISHED
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 131B
=> [internal] load metadata for docker.io/library/debian:latest
=> [1/4] FROM docker.io/library/debian
=> [2/4] RUN apt-get update -y
=> [3/4] RUN apt-get upgrade -y
=> [4/4] RUN apt-get install nano curl -y
=> exporting to image
=> => exporting layers
=> => writing image sha256:753f4e2ec32ad13407e4b45fec325ed3ffdb3363fd63369564651c209a554d82
=> => naming to docker.io/library/premier_exemple
```

docker:desktop-linux

0.1s
0.1s
0.2s
0.1s
0.0s
0.0s
6.5s
2.5s
24.1s
0.4s
0.4s
0.0s

docker build -t deuxieme_exemple .

```
[+] Building 29.0s (6/6) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 130B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/debian:latest
=> CACHED [1/2] FROM docker.io/library/debian
=> [2/2] RUN apt-get update -y && apt-get upgrade -y && apt-get install nano curl -y
=> exporting to image
=> => exporting layers
=> => writing image sha256:f26ff988847130ec798131cffa3154a633c742fa82b88f1e164325e0d68e78b7
=> => naming to docker.io/library/deuxieme_exemple
```

docker:desktop-11

0.0s
0.0s
0.0s
0.0s
0.0s
0.0s
28.2s
0.5s
0.5s
0.0s

Ces 2 exemples font la même chose même si le traitement est un peu différent...

Chapitre 2 : Docker initiation

Module 2.3 – Écriture d'un Dockerfile

Structure de base d'un Dockerfile avec les commandes les plus courantes :

- **FROM** : il s'agit de la première ligne d'un Dockerfile, il spécifie le point de départ de votre image en prenant comme base une autre image, par exemple : FROM debian:9 ou FROM debian:latest
- **RUN** : lance des commandes lors du build de l'image
- **ADD** : permet d'ajouter des fichiers dans le conteneur
- **WORKDIR** : définit le répertoire de travail pour les commandes lors de la construction d'une image (équivalent de la commande « cd »)
- **EXPOSE** : indique les ports qui sont exposés par le conteneur, EXPOSE 80 signifie que l'application à l'intérieur écoute le port 80.
- **VOLUME** : indique un répertoire qui sera partagé avec le système hôte
- **CMD** : définit la commande par défaut qui sera exécutée lorsque le conteneur démarre. Cette commande pourra être surchargé lors de l'exécution du conteneur.

Chapitre 2 : Docker initiation

Module 2.3 – Écriture d'un Dockerfile

Exemple plus complet :

```
# Utiliser l'image Debian comme base
FROM debian:bullseye-slim

# Mettre à jour les paquets et installer Apache et PHP
RUN apt-get update && \
    apt-get install -y apache2 php

# Définir le répertoire de travail
WORKDIR /var/www/html

# Copier les fichiers du site web statique depuis le répertoire local vers l'image
COPY index.html /var/www/html/
COPY style.css /var/www/html/

# Exposer le port 80 pour permettre l'accès au site web
EXPOSE 80

# Commande par défaut pour démarrer Apache une fois le conteneur lancé
CMD ["apache2ctl", "-D", "FOREGROUND"]
```


Chapitre 2 : Docker initiation

Construire une image via un Dockerfile

Chapitre 2 : Docker initiation

Module 2.3 – Construire une image via un Dockerfile

Comme vu précédemment la construction d'une image est réalisé via la commande docker build.

```
docker build -t nom_de_votre_image:tag .
```

L'option -t permet de définir le nom de l'image.

Il est également possible de préciser une étiquette (un tag) pour réaliser des versions de l'image.

Chapitre 2 : Docker initiation

Publier un Dockerfile

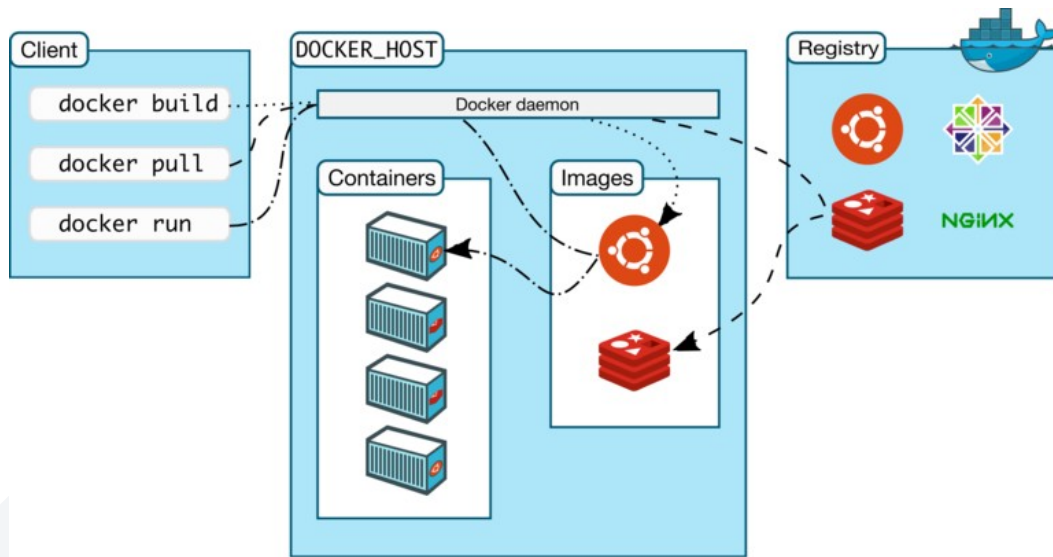
Chapitre 2 : Docker initiation

Module 2.3 – Publier un Dockerfile

Publier une image est l'action qui rend disponible l'image à travers un registre. Pour rappel, il existe des registres publics et privés.

Par défaut, les images sont publiées sur le registre officiel public de Docker (Docker Hub). Sur ce registre, il est possible d'opter pour des options payantes pour pouvoir publier des images privées (sur l'offre personnelle et gratuite, vous ne pouvez avoir uniquement qu'une image privée).

L'autre option pour stocker des images de manière privée est d'héberger sur son infrastructure un registre privée.



Chapitre 2 : Docker initiation

Module 2.3 – Publier un Dockerfile

Exemple de publication sur Docker Hub

- La première étape est d'avoir un compte Docker Hub et d'y être connecté via une commande « docker » login : **docker login -u -p**
- Ensuite, il est possible de publier l'image via la commande docker push :
docker push <username>/votre_image
- Une fois l'image publiée, vous la trouverez sur l'interface web de Docker Hub et vous pourrez la télécharger via une commande docker pull : **docker pull <username>/votre_image**

Chapitre 2 : Docker initiation

Module 2.3 – Publier un Dockerfile

La notion de « tag »

- Comme nous l'avons vu précédemment dans le cours, il est possible de mettre des étiquettes pour différencier des versions de l'image, par défaut, on se trouve sur le tag « **latest** ».

La commande docker tag permet d'apposer un tag et aussi de renommer une image, par exemple :

docker tag httpd:test fedora/httpd:version1.0.test



- Si nous voulons publier une version précise et différente de l'image, il faut réaliser au préalable une commande docker tag, puis la publier :

docker tag <username>:votreimage:latest <username>:votreimage:1.0

docker push <username>:votreimage:1.0

- Vous trouverez ensuite sur Docker Hub les 2 versions :

This repository contains 2 tag(s).

Tag	OS	Type	Pulled	Pushed
 1.0		Image	41 minutes ago	2 minutes ago
 latest		Image	41 minutes ago	an hour ago

Chapitre 2 : Docker initiation

Module 2.3 – Dockerfile (TP)

Suivre le TP fournit par le formateur.

Chapitre 2 : Docker initiation

Module 2.3 - Questions ?



Chapitre 2 : Docker initiation

Module 2.4

Docker Compose : Définir et exécuter des applications multi-conteneurs

Chapitre 2 : Docker initiation

Module 2.4 – Docker Compose

Sommaire :

- Présentation de Docker Compose
- Exemple de Docker Compose
- Exercice avec Docker Compose (TP)



Chapitre 2 : Docker initiation

Présentation de Docker Compose

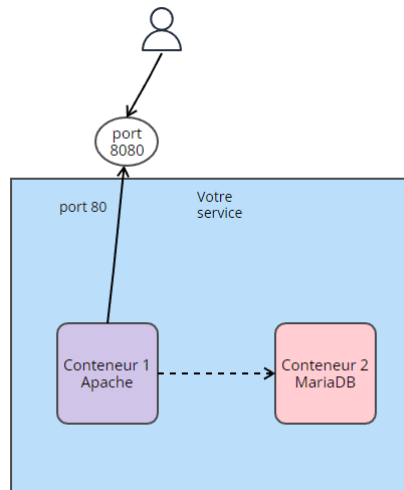
Chapitre 2 : Docker initiation

Module 2.4 – Présentation de Docker Compose

Docker Compose est un outil qui facilite la gestion des applications multi-conteneurs avec Docker.

Application multi-conteneurs ?

- C'est une application qui va nécessiter l'intervention de plusieurs conteneurs pour fonctionner. Il serait possible de tout packager dans un seul conteneur mais cette manière de faire pose des problèmes d'évolutivité.
- Une application multi-conteneurs offre une meilleure gestion des dépendances, une évolutivité simplifiée, une facilité de déploiement, et une gestion plus efficace des ressources



Chapitre 2 : Docker initiation

Module 2.4 – Présentation de Docker Compose

L'outil Docker Compose

- Docker compose prend comme forme un fichier YAML qui se nomme par défaut docker-compose.yml. Ce fichier va définir toute la configuration de votre application (réseaux, volumes, dépendances, etc.)

La structure de base d'un fichier docker-compose.yml

- **version** : définit la « norme » du fichier docker-compose, en effet comme docker évolue, la norme du fichier peut évoluer pour prendre en charge des nouveaux-mots clé ou en déprécier certains
- **services** : sous cette section, nous allons définir les composants de l'application global qui vont prendre la forme d'un conteneur à l'exécution. Chaque service va utiliser une image de base et définir ses paramètres (ports, variables d'environnement, etc.)
- **networks** : dans cette section, nous pouvons modifier les paramètres réseaux
- **volumes** : les volumes vont permettre de persister les données → ceci va permettre de conserver les données lorsque les conteneurs vont redémarrer.

Chapitre 2 : Docker initiation

Exemple de Docker Compose

Chapitre 2 : Docker initiation

Module 2.4 – Exemple de Docker Compose

Exemple de fichier Docker Compose

```
version: '3'

services:
  # Service MariaDB
  mariadb:
    image: mariadb:latest
    environment:
      MYSQL_ROOT_PASSWORD: example_root_password
      MYSQL_DATABASE: example_db
      MYSQL_USER: example_user
      MYSQL_PASSWORD: example_password
    ports:
      - "3306:3306"
    volumes:
      - mariadb_data:/var/lib/mysql

  # Service Apache (avec PHP)
  apache:
    image: php:apache
    ports:
      - "8080:80"
    volumes:
      - ./html:/var/www/html
    depends_on:
      - mariadb
```

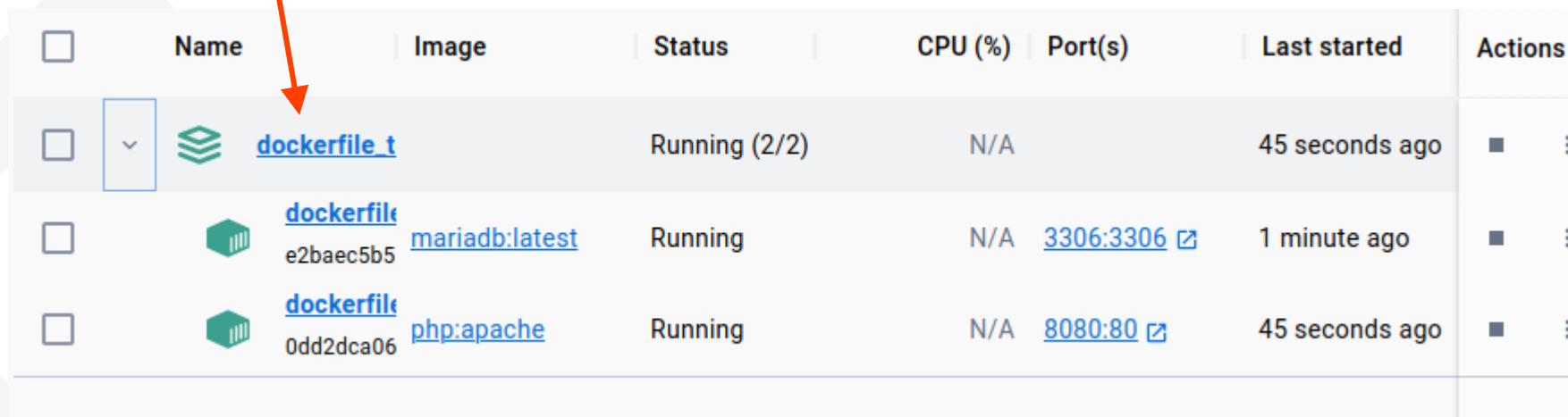
Chapitre 2 : Docker initiation












Module 2.4 – Exemple de Docker Compose

Exécution de fichier Docker Compose

Pour exécuter le fichier docker-compose.yml, il faudra exécuter la commande :

docker-compose up -d



<input type="checkbox"/>	Name	Image	Status	CPU (%)	Port(s)	Last started	Actions
<input type="checkbox"/>	 dockerfile_t		Running (2/2)	N/A		45 seconds ago	 
<input type="checkbox"/>	 dockerfile e2baec5b5	mariadb:latest	Running	N/A	3306:3306 	1 minute ago	 
<input type="checkbox"/>	 dockerfile 0dd2dca06	php:apache	Running	N/A	8080:80 	45 seconds ago	 

Chapitre 2 : Docker initiation

Module 2.4 – Exemple de Docker Compose

Gérer les conteneurs de Docker Compose

Une fois lancé, les conteneurs peuvent être stoppés avec la commande suivante :

docker-compose stop

Pour supprimer les conteneurs, il faudra utiliser la commande :

docker-compose rm

Une commande permet de stopper **ET** retirer les conteneurs :

docker-compose down

Par défaut docker-compose down ne retire pas les volumes, si vous souhaitez retirer les volumes, il faut ajouter l'option -v :

docker-compose down -v

Chapitre 2 : Docker initiation

Exercice avec Docker Compose (TP)

Chapitre 2 : Docker initiation

Module 2.4 – Exercice avec Docker compose (TP)

Suivre le TP fournit par le formateur.

Chapitre 2 : Docker initiation

Module 2.4 - Questions ?

