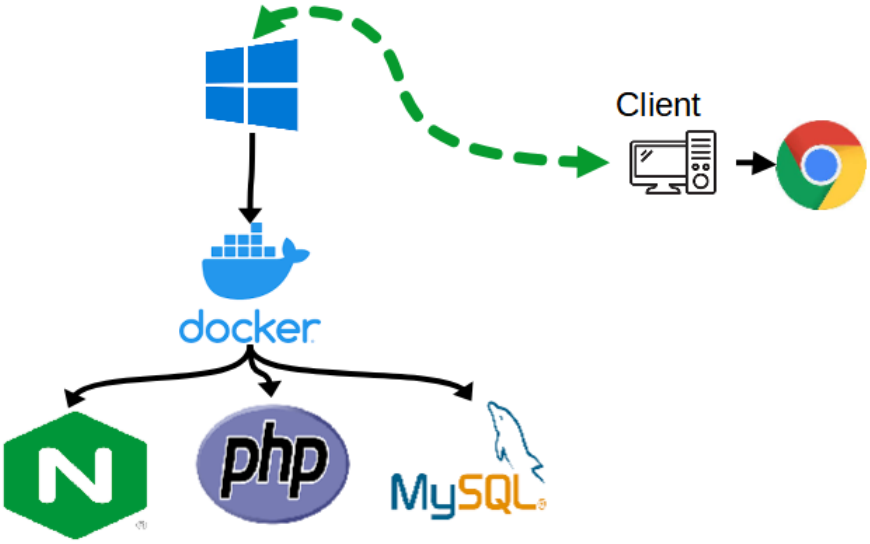
Docker	ACADÉMIE DE RENNES 3 juin 2025
Objectif : Apprendre à utiliser des containers	
Moyens : •	

1 Objectif

Ce document a pour objectif de démystifier le principe des container docker avec la mise en place d'un serveur **Nginx** avec **Docker**. **Vous trouverez en annexe A un ensemble de commandes utiles pour exploiter pleinement le containers.**



2 Présentation de docker

Les conteneurs sont semblables à une forme de virtualisation qui permet d'isoler les applications. Il reste cependant différent de la virtualisation. Toutes les fonctionnalités de la virtualisation du système d'exploitation ne sont pas disponibles. Il y a seulement les ressources nécessaires. Il en résulte un temps de démarrage plus rapide ! Un développeur peut spécifier les ressources dont il a besoin, le système d'exploitation dont il a besoin pour exécuter l'application et il n'a pas à se soucier des dépendances puisque le conteneur s'en occupe.

Machines virtuelles vs conteneurs Une machine virtuelle (VM) est comme une copie d'un ordinateur physique réel utilisant une technologie d'hyperviser proche de l'émulation.

Plusieurs VM peuvent fonctionner sur le même matériel physique. En revanche, les conteneurs se trouvent installés sur un serveur physique et son système d'exploitation hôte, voir dans une VM. Chaque conteneur partage le noyau de l'OS hôte cependant les binaires et les bibliothèques nécessaires à l'exécution de ces applications lui sont spécifiques.

Un conteneur Docker est semblable à un ordinateur dans votre ordinateur. On peut aussi le voir comme un serveur distant. On peut partager l'image d'un conteneur (une copie à un instant t, re-executable). On peut donc diffuser des logiciels, ou des données via un conteneur dans une version fixe. Ainsi à partir des mêmes sources répliquées, les autres utilisateurs obtiendront les mêmes résultats.

Architecture générale Docker utilise une architecture client-serveur, qui implique les trois principaux composants.

- le client Docker,
- le démon Docker
- le registre Docker (dockerhub par défaut).

Daemon

Type de programme informatique, un processus ou un ensemble de processus qui s'exécute en arrière-plan plutôt que sous le contrôle direct d'un utilisateur.

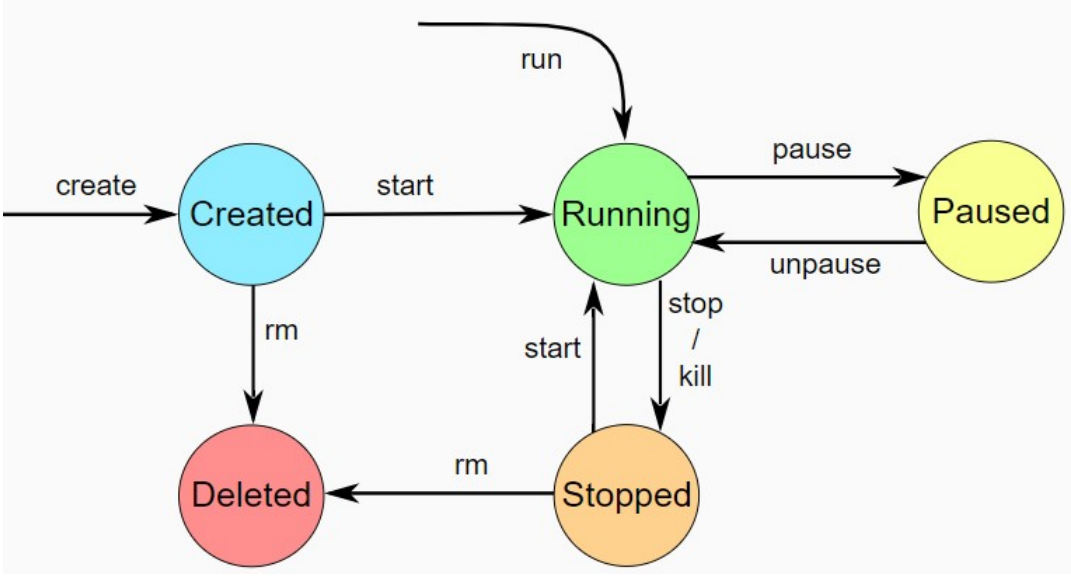
Le client Docker communique avec le démon Docker, qui se charge de construire, d'exécuter et de distribuer les conteneurs Docker.

Container

Ensemble de processus logiciels léger et indépendant, regroupant tous les fichiers nécessaires à l'exécution des processus : code, runtime, outils système, bibliothèque et paramètres.

Le client et le démon Docker peuvent fonctionner sur le même système ou connecter un client à un démon Docker distant.

Les conteneurs peuvent avoir différents états :



Les données stockées à l'extérieur du conteneur peuvent être utilisées même si le conteneur n'existe plus. Cependant, pour cela il est nécessaire de monter les données à l'extérieur du conteneur.

3 Installation de docker

Pour l'installation, il est nécessaire de suivre le tutoriel accessible via ce lien : Tuto docker.

Remarque : Cette étape n'a pas de difficulté.

Docker Hub : est un service fourni par Docker. vous pouvez le comparer à GitHub, mais spécialisé dans le stockage d'image pour Docker.

Pour utiliser Docker (et donc pour la suite), il est recommandé de créer un compte sur le Docker Hub.

4 Premier pas

4.1 Commandes

Cette partie a pour objectif de vous familiariser avec l'interface en **ligne de commande** qui permet de discuter avec le daemon Docker

Depuis un terminal, si vous exécutez la commande docker, vous obtiendrez une liste de commandes exécutables, que voici :

```
$ docker
...
Commands:
    attach Attach to a running container
    build Build an image from a Dockerfile
    commit Create a new image from a container s changes
        ....
    login Register or log in to a Docker registry server
    logout Log out from a Docker registry server
    logs Fetch the logs of a container
    port Lookup the public-facing port that is NAT-ed to PRIVATE_PORT
    pause Pause all processes within a container
    ps List containers
    pull Pull an image or a repository from a Docker registry server
    push Push an image or a repository to a Docker registry server
    restart Restart a running container
    rm Remove one or more containers
    rmi Remove one or more images
    run Run a command in a new container
    ....
```

Grâce à ces différentes commandes, vous pouvez facilement gérer vos containers et vos images. Nous verrons au cours de cette activité, les commandes les plus pratiques.

4.2 Images

Image

Une image est un container statique. On pourrait comparer une image à une capture d'un container à un moment donné. Lorsqu'on souhaite travailler avec un container, on déclare forcément un container à partir d'une image.

Vous pouvez à tout moment voir votre « bibliothèque » d'images avec la commande "docker images". Vous verrez, au fil des projets, vous aurez de plus en plus d'images. Utilisables comme "moules" à votre guise pour débuter vos projets ou, à minima, vos nouveaux containers.

Pour voir votre bibliothèque d'images :

```
> docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
test_php latest 214adac37e21 24 hours ago 449MB
nginx latest 2b7d6430f78d 3 days ago 142MB
mysql/mysql-server latest eb5713a2c247 4 weeks ago 428MB
phpmyadmin/phpmyadmin latest 4a4023c7e22a 3 months ago 510MB
```

Sur ma machine, il y a 4 images (que vous utiliserez pas la suite).

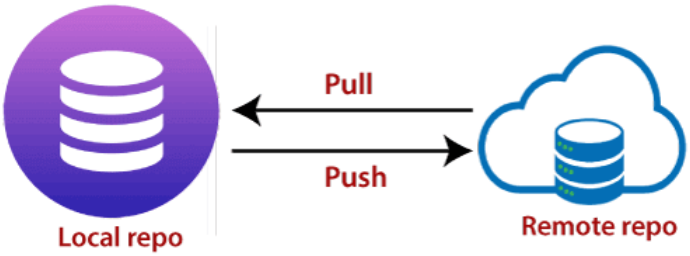
Il existe une plateforme maintenue par Docker sur laquelle tout le monde peut `pusher` et `puller` des images. C'est un peu comme le GitHub des images Docker (vous verrez cela durant votre formation). Il existe des images officielles pour tout et n'importe quoi ! Cette bibliothèque géante partagée, c'est le Docker Hub Registry !

Push

Commande 'générique' permettant de **déposer** des fichiers sur un dépôt.

Pull

Commande 'générique' permettant de **récupérer** des fichiers sur un dépôt.



4.3 Hello World

Pour valider le fonctionnement de Docker (et comme toujours en informatique), vous allez récupérer une image de Hello world et la lancer sur votre machine. Pour cela :

- Allez sur `hub docker`
- Cherchez l'image de `Hello world`
- Télécharger l'image

```
docker pull hello-world
```

- Vérifier avec la commande adapté que votre image a bien été téléchargée.

Commande run . Cette commande permet de créer un nouveau container et télécharger son image si elle n'est pas déjà en local. Pour créer et exécuter un container, il faut utiliser cette commande de la manière suivante

```
docker run hello-world
```

Un grand nombre d'option que l'on verra par la suite permet de configurer le lancement des dockers (liste non-exhaustive) :

- `-d` : lance le container en arrière plan.
- `-t` : alloue terminal
- `-p 8080:80` : redirige un port interne du container vers un port local
- `-rm` : supprime le container à la sortie
- `-v rep_local/rep_container` : lie un répertoire local à un répertoire du container
- ...

Appeler votre très cher professeur si vous avez des difficultés.

Le résultat d'exécution doit être le suivant :

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

Si c'est le cas : Bravo, vous pouvez passer à la suite.

5 Dockerfile

Vous maitrisez maintenant les containers Docker. Il va falloir passer à l'étape suivante : installer un container 'utile'. Vous voudriez avoir à disposition un environnement linux avec la chaine de compilation **gcc** (vue en début d'année). Pour cela, il y a plusieurs solutions.

- Vous installer cygwin sur windows pour émuler un système Linux ⇒ Lourd et peu stable

- Vous installer une machine virtuelle linux et y installer gcc ⇒ Lourd et mais stable
- Vous utiliser toutes les fonctionnalités des containers docker ⇒ **léger et stable**

5.1 1er test

Voici la procédure que vous devez réaliser sur la machine hôte (dans un powershell sous Windows) :

1. Installer l'image de Ubuntu (vous pouvez aller voir sa description sur docker hub). Pour cela dans un powershell Windows taper la commande suivante :

```
docker pull ubuntu
```

2. Créer et lancer l'image d'ubuntu. Profiter en pour nommer le container (dans l'exemple le container est nommé ubuntuPV)

```
docker run -d -t --name ubuntuPV ubuntu
```

3. Vérifier que votre container est bien lancé :

```
docker ps
```

Vous devez voir votre container.

4. Lancer un terminal bash de votre container ubuntu :

```
docker exec -it ubuntuPV bash
```

5. Dans le terminal Ubuntu, vérifier l'accès à gcc :

```
gcc -v
```

6. Y-a-t'il un problème ?

5.2 Création de votre propre image

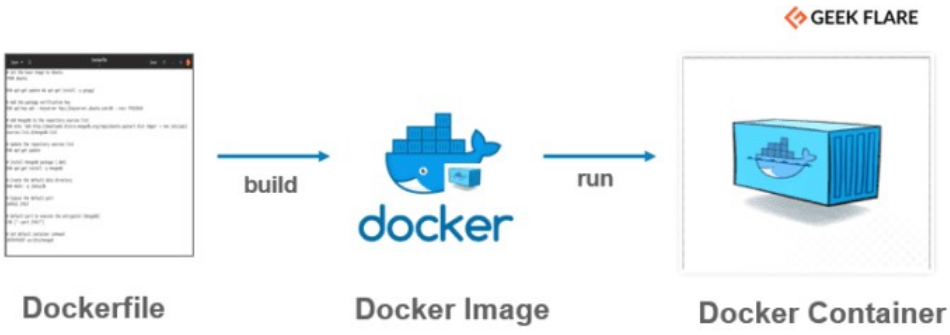
Comme vous avez pu le constater, l'image officiel d'Ubuntu n'inclut pas gcc. Vous allez maintenant apprendre à créer une image avec les éléments que vous souhaitez :

- GCC
- vim (comme cela vous aurez accès à un éditeur de texte sur cette distribution).

Pour cela, vous allez utiliser un dockefile.

Dockerfile

Un Dockerfile est un simple fichier texte placé sur la machine hôte regroupant l'ensemble des commandes pour construire l'image. Une fois l'image personnelle créée, vous pourrez créer une instance exécutable de cette image.



1. Ainsi, la première étape est de créer dans un répertoire de votre choix sur la machine hôte un fichier nommé "Dockerfile" (sans extension). Il contiendra toutes les instructions pour créer votre image.

Le fichier Dockerfile doit contenir les instructions qui suivent :

2. La première instruction obligatoire doit indiquer l'image de base

```
FROM ubuntu:latest
```

3. Ensuite, il faut écrire l'instruction `RUN` pour exécuter une commande dans notre container. Vous voulez :
   - Mettre à jour,
   - Installer GCC
   - Installer vim

```
RUN apt-get update -yq \
&& apt-get install gcc -yq \
&& apt-get install vim -yq \
&& apt-get clean -y
```

4. Vous voulez aussi que votre répertoire de travail par défaut (au lancement de l'image) soit : `/usr/src/`

```
WORKDIR /usr/src
```

Il existe beaucoup d'autres options pour configurer votre image (que nous verrons pas dans cette activité).

5. Vous allez maintenant créer votre image. Placer dans le répertoire du fichier et exécuter la commande

```
docker build -t imageperso .
```

Cette commande construit l'image de ubuntu+Gcc+Vim et la nomme `imageperso`.

Vous avez maintenant construit une nouvelle image que vous pouvez utiliser, partager,....

6. Vérifier avec les commandes appropriés que vous pouvez créer un container permettant d'avoir un environnement Linux avec gcc et vim

5.3 Fichiers partagés

Vous avec un container fait 'maison' avec un environnement de développement gcc. Une question reste en suspend : commet faire pour que mon container est accès aux fichiers de mon ordinateur ? Rien de plus simple, il suffit de créer un parage de fichier entre l'hote et docker.

1. Dans le répertoire du dockerfile, créer un sous-répertoire nommé src (pour source) contenant un fichier C helloworld.c. Ce fichier contient le code source pour afficher helloworld.
2. Créer et lancer le container (instance de l'image `imageperso`) grâce à la commande ci-dessous.

```
docker run -d -t -v C:\Users\Pierre\...\src:/usr/src/
        --name containerPV imageperso
```

Le partage de fichier se fait grâce à l'option `-v [repertoire hote]: [repertoire docker]`. Le nom du container est `containerPV` (vous pouvez bien sur le changer). L'image d'origine du container est `imageperso`.

Vous pouvez maintenant tester le bon fonctionnement de gcc (compilateur C)

3. Ouvrer un shell de votre container (invité de commande)

```
docker exec -it containerPV bash
```

4. Vérifier que votre fichier partagé est présent (`hello_wrold.c`)

```
ls
```

5. Compiler le code source C avec la commande

```
gcc hello_world.c -o progExecutable
```

6. Vous venez alors de créer un programme nommé `progExecutable`. Pour l'exécuter il suffit de taper

```
./progExecutable
```

Normalement, un message 'hello world' apparaît. C'est que tout fonctionne.

7. Vous pouvez quitter le container en tapant

```
exit
```

Il reste toujours en exécution même si vous n'avez plus de visuel sur le terminale.

8. Pour retourner dans le container, il est possible de refaire la commande suivante.

```
docker exec -it containerPV bash
```

6 Orchestration des containers

Vous souhaiteriez avoir un serveur LEMP constitué de :

- **Nginx**
- **PHP**
- **mysql**
- **Phpmyadmin**

Vous pouvez créer et lancer des containers 'à la main' comme vu précédemment. Vous verrez si vous essayer que ce n'est pas le plus pratique. Il existe une solution appropriée : Docker compose.

Docker compose

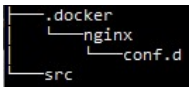
Docker Compose va vous permettre d'orchestrer vos conteneurs, et ainsi de simplifier vos déploiements sur de multiples environnements. Docker Compose est un outil écrit en Python qui permet de décrire, dans un fichier YAML, plusieurs conteneurs comme un ensemble de services.

En résumé, il suffit de configurer correctement un fichier nommé : `docker-compose.yml` pour pouvoir lancer un ensemble de container.

6.1 1er pas : Nginx

Le 1re test est de regarder uniquement si vous arrivez à orchestrer avec docker compose un unique container composer uniquement de **Nginx**.

1. Préparer vous un répertoire de travail contenant un fichier nommé `docker-compose.yml` . Vous pouvez ouvrir ce fichier avec Notepad++ par exemple.
2. Créer les différents répertoires :



3. Dans votre fichier d'orchestration copier les lignes ci-dessous :

```
#version de docker compose
version: '3.8'
# Services que le veut lancer
services:

  # Service nomme nginx
  nginx:
    #L'image que je veux recuperer
    image: nginx:1.18
    #nom du container
    container_name: nginxPVseul
    #redirection de port
    ports:
      - 80:80
    #repertoires partages
    volumes:
      - ./src:/var/www/html
      - ./.docker/nginx/conf.d/default.d:/etc/nginx/conf.d/default.conf
    #configuration du repertoire de travail
    working_dir: /var/www/html
```

Les commentaires (commençant par un '#') permette de comprendre le rôle des différentes lignes.

4. Dans le fichier de configuration nginx nommé `default.d` placé dans le répertoire créé précédemment `.docker/nginx/conf.d/`, coller les lignes suivantes :

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;
    root /var/www/html;
    index index.html;


}
```

5. Pour lancer le ou les container à partir d'un fichier `docker-compose.yml`, il suffit de taper la commande dans le répertoire :

```
docker compose up -d
```

6. Pour vérifier que votre container est bien lancé, vous pouvez taper la commande :

```
docker compose ps
```

Vous devez alors voir apparaître votre container.

```
NAME            COMMAND                  SERVICE         STATUS          PORTS
nginxPVseul     "/docker-entrypoint.…"   nginx           running         0.0.0.0:80->80/tcp
```

7. Pour ouvrir un terminal de votre container nginx, vous pouvez tapper la commande

```
docker exec -it nginxPVseul bash
```

8. Pour en sortir, il suffit de taper la commande

```
exit
```

6.2 Lemp global

Pour exécuter le serveur Lemp, il faut compléter (et modifier) le fichier `docker-compose.yml`. L'objectif est de rajouter des services en plus de `nginx`. Par soucis de simplicité, les différents fichiers vous sont donnés (dans le répertoire `ex3-lemp_complet`) :

```
.
├── docker-compose.yml
├── .docker
│   ├── mysql
│   │   └── my.cnf
│   ├── nginx
│   │   └── conf.d
│   │       └── default.d
│   └── php
│       └── Dockerfile
├── mysqldata
│   └── hotel_reservation.sql
└── src
    ├── formulaire_hotel_bdd_II.php
    ├── gestion_formulaire_hotel_bdd_II.php
    ├── index.php
    ├── suppression_reservation_II.php
    └── visualiser_reservation_hotel_bdd_II.php
```

1. Après avoir **Etudier et commenter** les différents fichiers, activer et tester ce serveur sur docker
2. Vous devez produire
   - Un document présentant les quelques commandes permettant de créer et lancer un ensemble de container de type LEMP. le fichier `docker-compose.yml` ainsi que l'ensemble des répertoires et fichiers de configuration devra aussi être disponible.

A Commendes docker utiles

Obtenir les commandes pour docker ou pour docker compose

```
docker
```

ou

```
docker compose
```

Visualiser les containers en cours d'exécution

```
docker ps
```

Visualiser l'ensemble des containers en exécution et arrêtés

```
docker ps
```

Supprimer un container

```
docker rm nomDuContainer
```

Créer une image `nomImage` à partir d'un fichier Dockerfile (se placer dans le répertoire contenant le fichier`dockerfile`)

```
docker build -t nomimage .
```

Supprimer une image

```
docker rmi nomImage
```

Créer et lancer un conteneur Docker à partir d'une image

```
docker run [options] <image_name>
```

Lancer un bash d'un conteneur

```
docker exec -it mon_conteneur /bin/bash
```

Visualiser les logs d'un container

```
docker logs nomContainer
```

Créer et lancer un ensemble de containers à partir de fichier `docker-compose.yml` (se placer dans le répertoire contenant le fichier`docker-compose.yml`)

```
docker compose up -d
```

Stopper et supprimer un ensemble de containers (se placer dans le répertoire contenant le fichier `docker-compose.yml`)

```
docker compose down
```

Démarrer un ensemble de containers (se placer dans le répertoire contenant le fichier`docker-compose.yml`)

```
docker compose start
```

Stopper un ensemble de containers (se placer dans le répertoire contenant le fichier `docker-compose.yml`)

```
docker compose stop
```

Partager un volume entre l'hôte et le containers (dans le fichier `docker-compose.yml`)

```
volumes:
     - ./repHote:repContainer
```

Rediriger un port dans un container (dans le fichier `docker-compose.yml`)

```
ports :
       - 8080:80
```

avec

- 8080 : port hôte
- 80 : port du containers