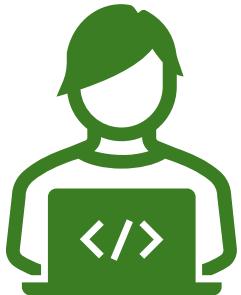


Projet Tutoré & DevOps

Hiba Najjar

S7, GInf4

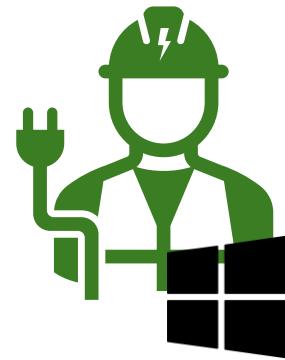
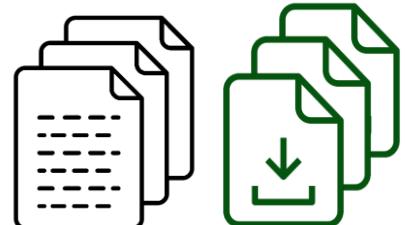
L'ancien processus de développement



Développeur



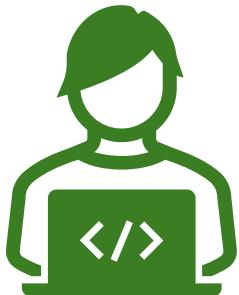
Communique la liste des exigences techniques pour faire fonctionner correctement la nouvelle version du système.



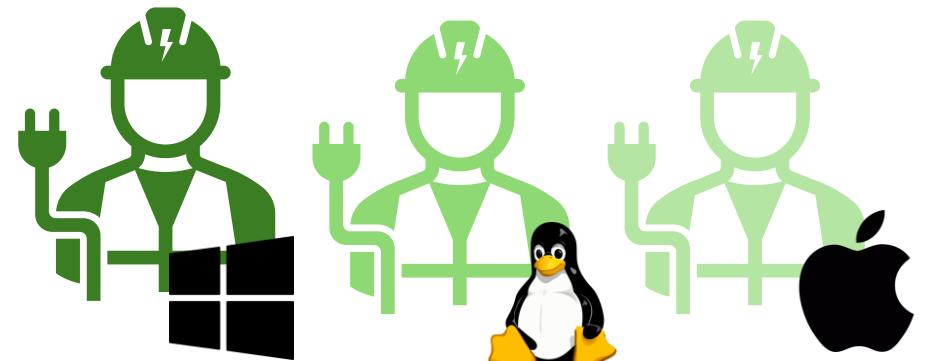
Opérateurs

- Si l'application utilise 10 services, chacun devra les installer sur sa machine
- ✖ Processus d'installation différent pour chaque système d'exploitation
- ✖ De nombreuses étapes, quelque chose peut mal tourner

L'ancien processus de développement

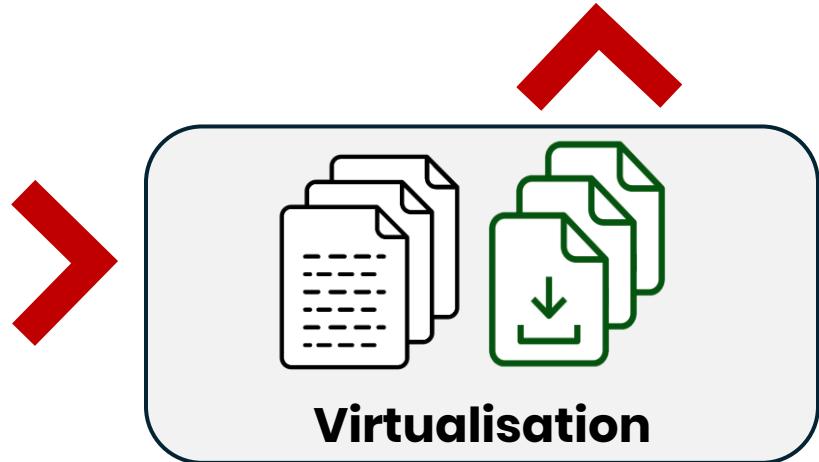
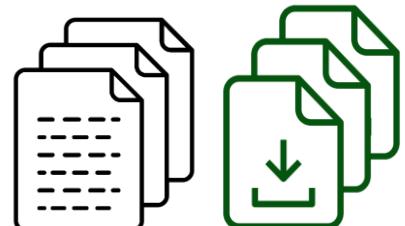


Développeur



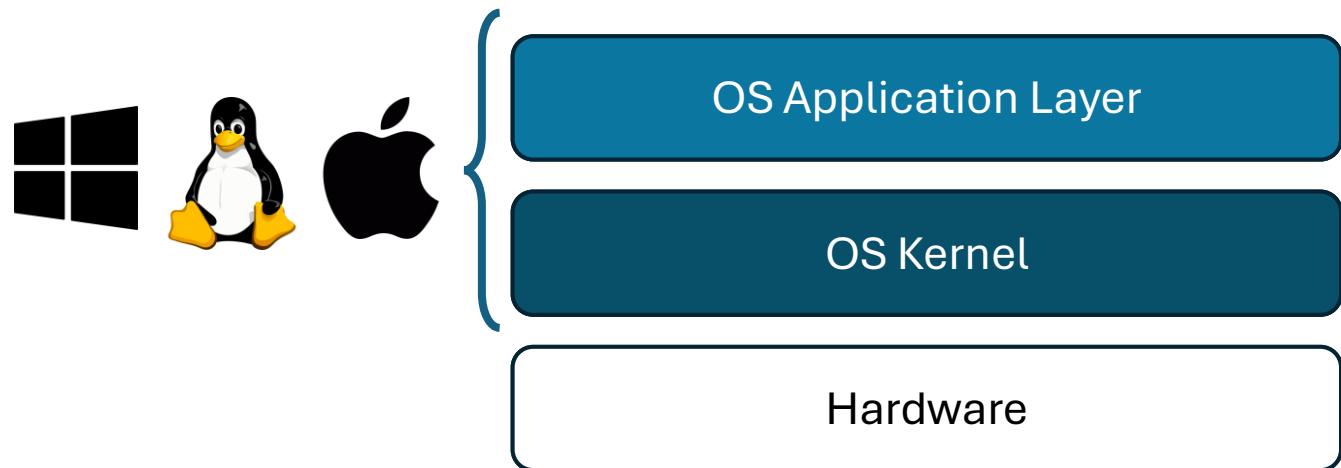
Opérateurs

Communique la liste des exigences techniques pour faire fonctionner correctement la nouvelle version du système.

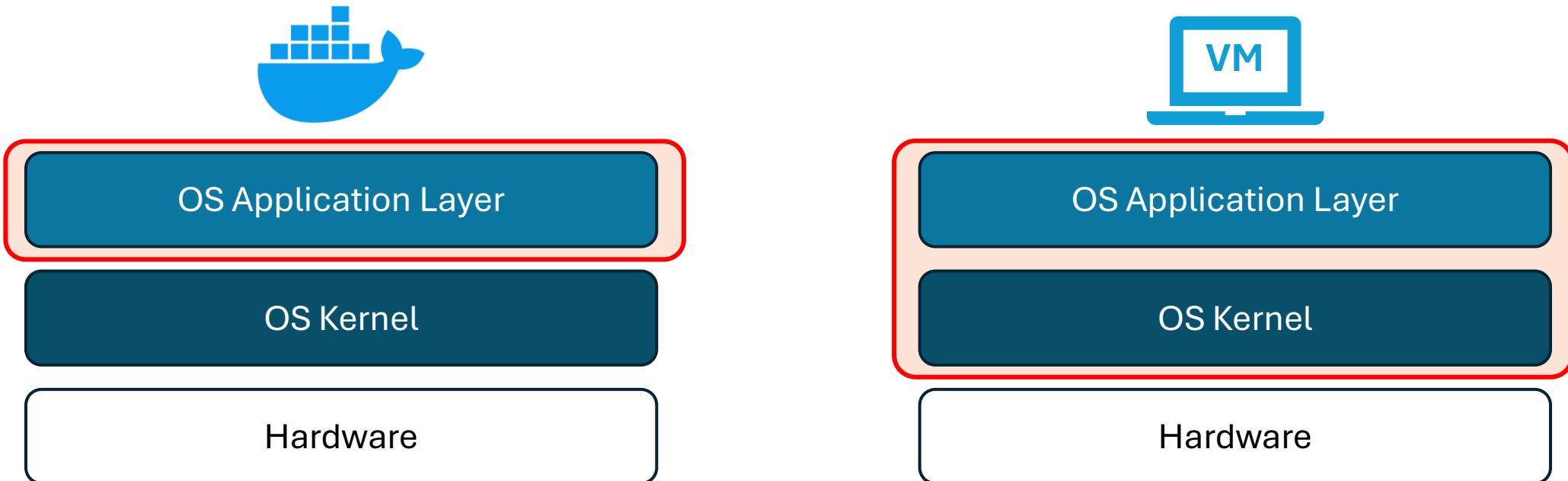


Virtualisation

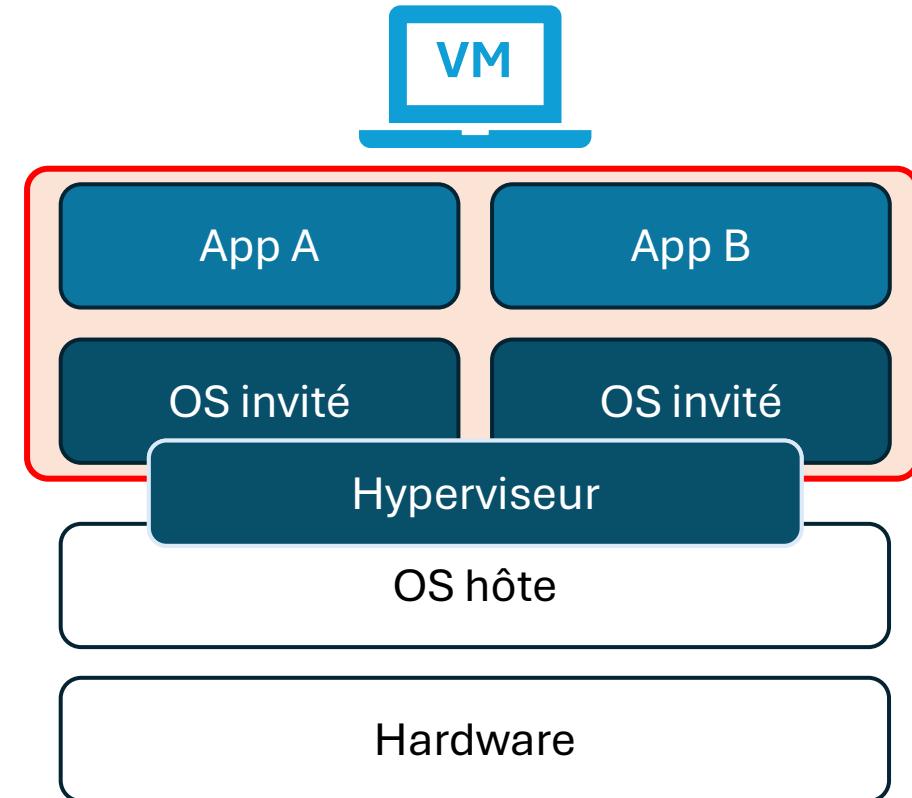
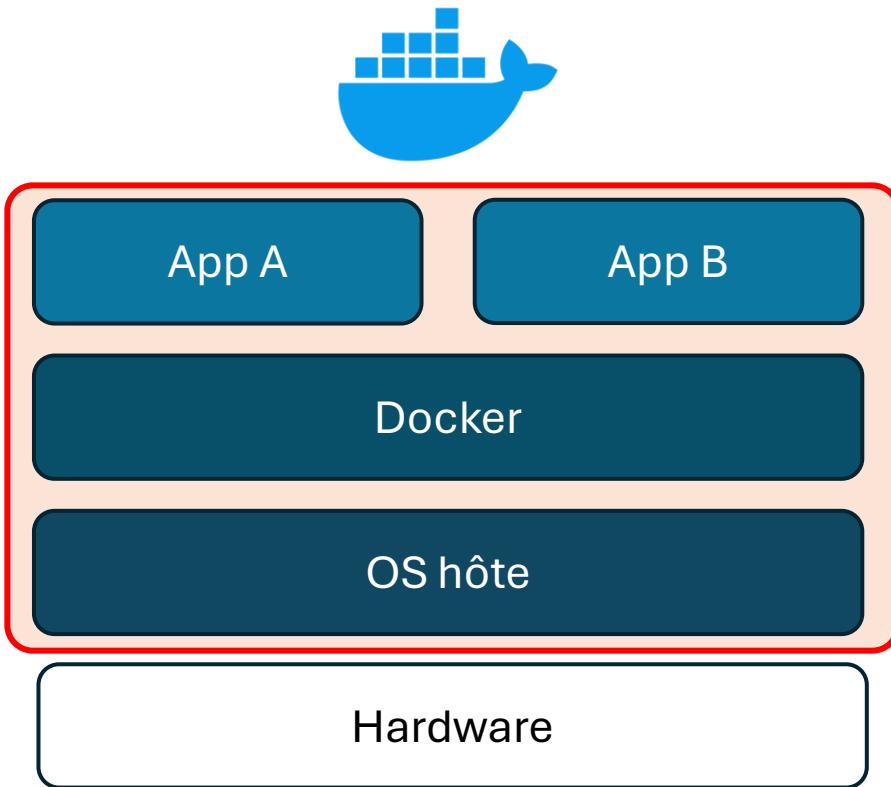
Virtualisation dans l'OS



Machine Virtuelles & Conteneurs



Machine Virtuelles & Conteneurs Docker



Machine Virtuelles & Conteneurs Docker



- Partagent le noyau du système hôte
 - Isolation plus légère
 - Légers: ~centaines de Mo
 - Faibles charges de fonctionnement
 - Démarrage rapide/instantané
 - Sécurité bonne mais plus vulnérable
- Embarquent un OS complet
 - Isolation plus forte (~2 machines)
 - Espace plus grand (plusieurs Go)
 - Consomment plus de RAM/CPU
 - Démarrage ralenti par l'OS
 - Sécurité très élevée

Installation Docker ...

Go to www.docker.com/

The screenshot shows the official Docker website. At the top, there is a navigation bar with links for AI, Products, Developers, Pricing, Support, Blog, and Company. To the right of the navigation bar are a search icon, a "Sign In" button, and a prominent blue "Get Started" button. Below the navigation bar, there is a banner featuring a video player interface with play and pause buttons. A callout box contains the text "NEW Webinar series – Secure your software supply chain". The main headline reads "The same containers you love. Built for what's next." Below the headline, a subtext states "Build, secure, share, and run agents and apps on the container platform trusted by 20M+ developers." At the bottom, there are two blue buttons: "Choose plan" and "Download Docker Desktop". The "Download Docker Desktop" button is circled in red.

AI ▾ Products ▾ Developers ▾ Pricing Support Blog Company ▾

Q Sign In Get Started

NEW Webinar series – Secure your software supply chain

The same containers you love. Built for what's next.

Build, secure, share, and run agents and apps on the container platform trusted by 20M+ developers.

Choose plan Download Docker Desktop

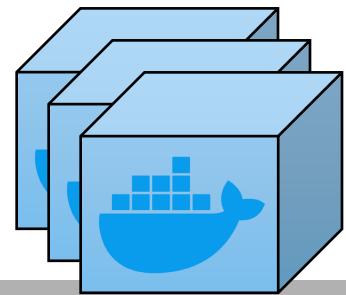
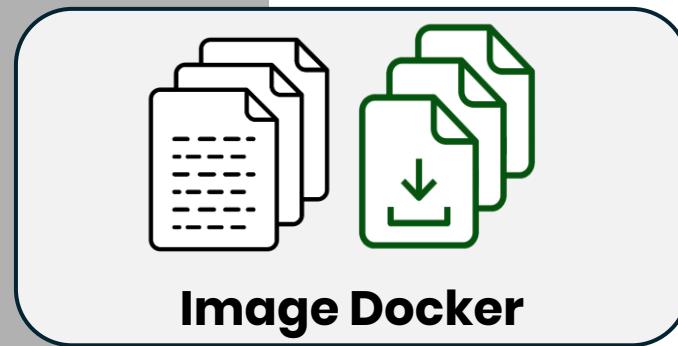
Images & Conteneurs



Image Docker = un package qui inclut tout ce qui est nécessaire à l'exécution d'une application

- Le code
- L'exécution
- Les variables d'environnement
- Les bibliothèques
- Les fichiers de configuration

→
Image instanciée



Conteneur Docker
= une image en cours d'exécution

Exemple

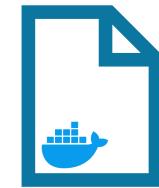
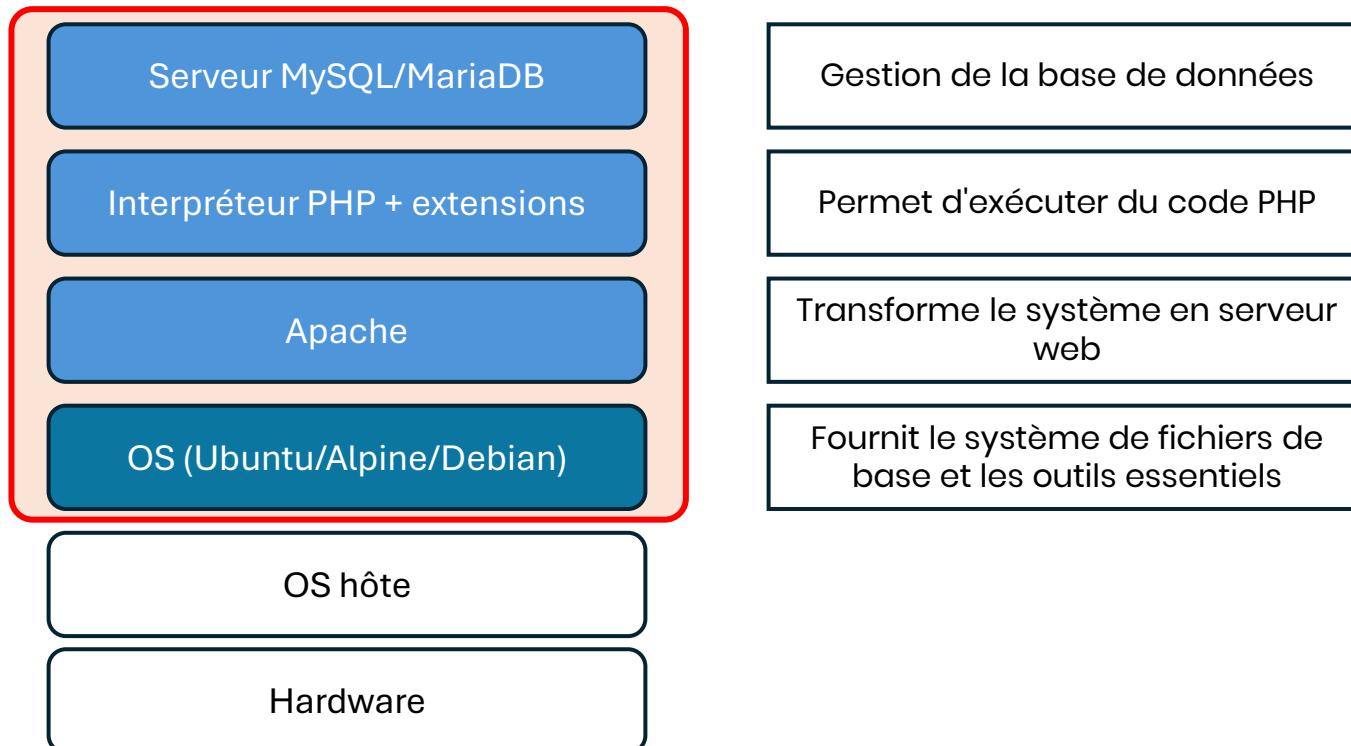


Image docker LAMP
(Linux Apache MySQL PHP)



Registres Docker

Docker Hub Registry

- registre public officiel de Docker,
- plateforme centrale de stockage et gestion d'images Docker.
- contient plusieurs types d'images:
 - Officielles, certifiées par les éditeurs (mysql, redis, nginx, ..)
 - Communautaires, créées et partagées par les utilisateurs,
 - Privées, stockées sur le hub sans être accessible au public

Registres Docker

Docker Hub Registry

- Docker CLI

```
# Rechercher des images
docker search redis
docker search --filter "is-official=true" redis

# Télécharger une image
docker pull nginx:latest

# Pousser une image personnalisée
docker push moncompte/mon-app
docker push moncompte/mon-app:v1.0
```

Registres Docker

- Application

- 1) Search Docker Hub for "hello-world" images from the CLI
- 2) Pull the official hello-world image
- 3) Run a container from that image
- 4) Check that the container ran and exited successfully
- 5) Clean up

```
docker search hello-world
docker search --filter "is-official=true"
docker pull hello-world
docker run --name my-hello hello-world
docker ps -a
docker logs my-hello
docker rm my-hello
docker rmi hello-world
```

Registres Docker

- Application

- 1) Search Docker Hub for “ubuntu” images from the CLI
- 2) Pull the official “ubuntu” image
- 3) Run a container from that image in interactive mode (**-ti**)
- 4) Verify if git is installed
- 5) If not, install git and verify its version
- 6) Clean up

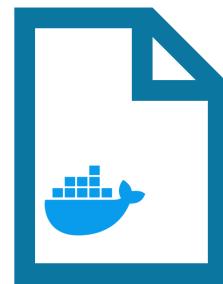
```
docker search ubuntu
docker pull ubuntu
docker run -ti --name ubuntu_cont1 ubuntu
root@xxxxxx: /# apt-get update -y && apt-get install -y git
root@xxxxxx: /# git --version
root@xxxxxx: /# exit
docker rm ubuntu_cont1
docker rmi ubuntu
```

Création d'une image Docker

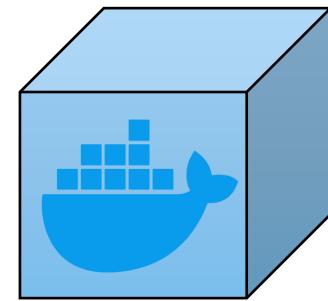


Dockerfile

contient les instructions qui créeront toutes les couches nécessaires à l'image



Docker Image



Docker Container



Build

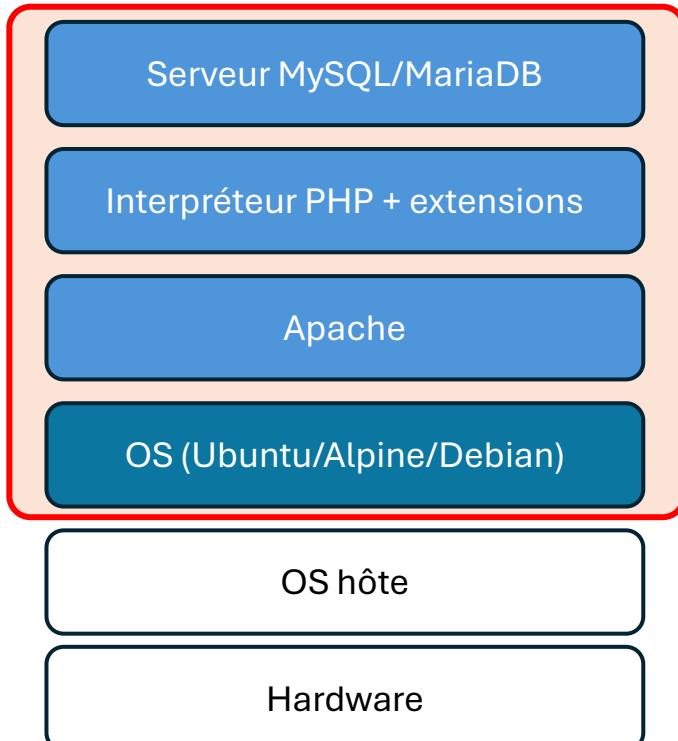


Run

Example



Image docker LAMP
(Linux Apache MySQL PHP)



Gestion de la base de données

```
RUN apt-get install mysql-server
```

Permet d'exécuter du code PHP

```
RUN apt-get install php php-curl
```

Transforme le système en serveur web

```
RUN apt-get install apache2
```

Fournit le système de fichiers de base et les outils essentiels

```
FROM ubuntu:20.04
```

Création d'une image Docker

Commandes de base:

FROM : Indique l'image de base à partir de laquelle l'image sera construite (point de départ obligatoire de presque tous les Dockerfile).

LABEL : Ajoute des métadonnées à l'image sous forme de paires clé–valeur (auteur, version, URL, etc.), utiles pour la documentation et l'outillage.

ARG : Définit des variables de build utilisables uniquement pendant la construction de l'image (inaccessibles au conteneur en exécution).

ENV : Définit des variables d'environnement visibles à la fois pendant le build et à l'intérieur du conteneur au runtime.

RUN : Exécute une commande dans une couche intermédiaire lors du build (installation de paquets, configuration, etc.) et enregistre le résultat dans l'image.

COPY : Copie des fichiers ou dossiers depuis le contexte de build (ton projet local) vers le système de fichiers de l'image/conteneur.

ADD : Comme COPY, mais peut aussi récupérer une ressource distante (URL) et extraire automatiquement certaines archives locales (tar) ; recommandé seulement quand ces fonctionnalités supplémentaires sont nécessaires.

Création d'une image Docker

Commandes de base (suite):

ENTRYPOINT : Définit le « programme principal » du conteneur, la commande de base qui sera toujours lancée quand le conteneur démarre.

CMD : Fournit la commande par défaut ou les arguments par défaut de l'ENTRYPOINT ; peut être remplacé au démarrage via la ligne de commande.

WORKDIR : Définit le répertoire de travail courant pour les instructions suivantes (RUN, CMD, ENTRYPOINT...) et pour le processus principal du conteneur au démarrage.

EXPOSE : Documente le ou les ports sur lesquels l'application à l'intérieur du conteneur écoute ; n'ouvre pas le port sur l'hôte à lui seul.

VOLUME : Déclare un point de montage destiné à stocker des données persistantes ou partagées (vers un volume Docker ou un dossier de l'hôte).

USER : Indique l'utilisateur (et éventuellement le groupe) qui exécutera les instructions suivantes et le processus principal du conteneur, au lieu de l'utilisateur root par défaut.

Création d'une image Docker

Example 1: application Node.js

```
# Utiliser l'image officielle Node.js comme base
FROM node:25-alpine3.21

# Copier package.json qui contient les dépendances
COPY package.json /app/

# Définir le répertoire de travail à l'intérieur du conteneur
WORKDIR /app

# Install dependencies
RUN npm install

# Copier le code
COPY src /app/

# Define the command to run your app
CMD ["node", "server.js"]
```

Création d'une image Docker

Example 2: application SQL

```
# COUCHE OS
FROM debian:stable-slim

# MÉTADONNÉES DE L'IMAGE
LABEL version="0.1" maintainer="Hiba Najjar
<hibanajjar998@gmail.com>

# VARIABLES TEMPORAIRES
ARG APT_FLAGS="-q -y"
ARG DOCUMENTROOT="/var/www/html"

# COUCHE APACHE
RUN apt-get update -y && \
    apt-get install ${APT_FLAGS} apache2

# COUCHE MYSQL
RUN apt-get install ${APT_FLAGS} mariadb-
server
COPY db/articles.sql /
```

```
# COUCHE PHP
RUN apt-get install ${APT_FLAGS} \
    php-mysql \
    php && \
    rm -f ${DOCUMENTROOT}/index.html && \
COPY app ${DOCUMENTROOT}

# OUVERTURE DU PORT HTTP
EXPOSE 80

# RÉPERTOIRE DE TRAVAIL
WORKDIR ${DOCUMENTROOT}

# DÉMARRAGE DES SERVICES LORS DE
# L'EXÉCUTION DE L'IMAGE
ENTRYPOINT service mariadb start &&
    mariadb < /articles.sql && apache2ctl -D
    FOREGROUND
```

Partager une image Docker

1. **Se connecte à Docker,**
2. **renommer son image,**
3. **la pousser vers le repository distant.**

Le repository public se créera automatiquement sur votre compte Docker Hub

```
docker login
docker tag my_hello_app:0.1 hibanaj/myhello_app:first
docker push hibanaj/myhello_app:first
```

Création d'une image Docker

Application

- Recopier et modifier l'application sur GitHub:
[hibanajjar998/cours-devops-25-26/
 >> lesson2_Docker/
 >> exemple1_app_nodejs/](https://github.com/hibanajjar998/cours-devops-25-26/tree/main/lesson2_Docker)
- Construire (build) l'image Docker
- Créer (run) un conteneur et vérifiez que l'application s'y exécute comme prévu
- Partagez votre image sur DockerHub

Fonctionnement des volumes

- les systèmes de fichiers des conteneurs Docker sont éphémères;
- **Comment sauvegarder ces changements?**
 1. Transformer le conteneur en une nouvelle image
 - Déconseillée

```
docker commit <CONTAINER NAME or ID> <NEW IMAGENAME>
```

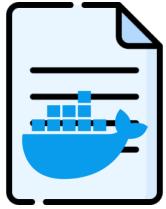
- 2. Utiliser les **volumes**

Fonctionnement des volumes

Un volume ~ disque dur externe que vous branchez sur votre conteneur

```
docker volume create <VOLUMENAME>
docker volume ls
docker volume inspect <VOLUMENAME>
docker volume rm <VOLUMENAME>
```

Fonctionnement des volumes



```
FROM alpine:latest
RUN mkdir /data
WORKDIR /data
```

Fonctionnement des volumes



```
# créer l'image Docker
docker build -t testimage
# Créer un volume
docker volume create data-volume
# Lancer le conteneur en mode interactif:
docker run -ti --name testimage_cont \
            -v data-volume:/data testimage
# Une fois dans le conteneur :
echo "ceci est un test" > test.txt
```

Si on démarre un conteneur avec un volume qui n'existe pas, Docker le créera.

Fonctionnement des volumes



```
docker volume inspect data-volume
```

```
[  
  { ...  
    "Mountpoint": "/var/lib/docker/volumes/data-volume/_data",  
    "Name": "data-volume", ....  
  }  
]
```

- Sur **Linux**, le chemin indiqué par **docker volume inspect** est le vrai chemin sur le système de fichiers.
- Sur **Windows**, les volumes sont stockés à l'intérieur de la machine virtuelle WSL 2 que Docker Desktop utilise. > Explorer idéalement depuis Docker Desktop.

Fonctionnement des volumes



```
# Monter un volume ou un chemin local
docker run -ti --name cont1 -v data-volume:/data testimage
docker run -ti --name cont1 -v $PWD/data:/data testimage

# Monter plusieurs volumes
docker run -ti --name cont1 \
    -v data-volume:/data1 \
    -v $PWD/data:/data2 testimage

# Supprimer un volume
docker volume rm data-volume

# Supprimer les volumes inutilisés
docker volume prune
```

Docker-Compose pour la gestion des conteneurs



Image docker LAMP
(Linux Apache MySQL PHP)

Serveur MySQL/MariaDB

Interpréteur PHP + extensions

Apache

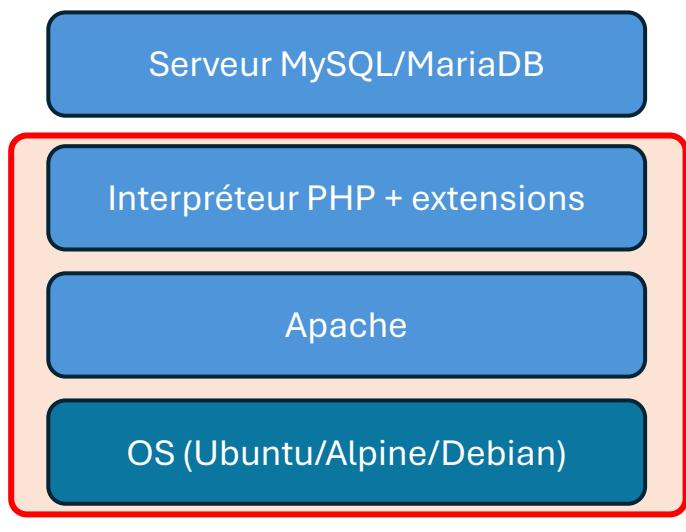
OS (Ubuntu/Alpine/Debian)

```
docker run -d -e MYSQL_ROOT_PASSWORD='test' \
-e MYSQL_DATABASE='test' \
-e MYSQL_USER='test' \
-e MYSQL_PASSWORD='test' \
--volume db-volume:/var/lib/mysql \
--volume $PWD/articles.sql:/docker-
entrypoint-initdb.d/articles.sql \
--name mysql_c mysql:5.7
```

Docker-Compose pour la gestion des conteneurs



Image docker LAMP
(Linux Apache MySQL PHP)



```
docker run -d \
--volume $PWD/app:/var/www/html \
-p 8080:80 \
--link mysql_c \
--name myapp_c myapp
```

Docker Compose pour la gestion des conteneurs

- Docker `compose` permet d'exécuter des applications docker à conteneurs multiples
- Un fichier YAML configure cette composition: `docker-compose.yml`

Application:

1. Préparer les fichiers de l'application
2. Créer un fichier `Dockerfile` pour préparer les couches OS/Apache/PhP
3. Créer l'image de l'application: `docker build -t myapp`
4. Créer le fichier `docker-compose.yml` pour préparer la composition de conteneurs (deux conteneurs)
5. Lancer la composition, qui contiendra deux images: `docker compose up -d`
6. Lancer l'application (<http://localhost:8080>)

Docker-Compose pour la gestion des conteneurs

À votre tour:

1. Aller sur <https://github.com/docker/awesome-compose/>
2. Ouvrir un exemple qui compose 3 conteneurs
3. Observer comment on construit une image depuis le fichier **docker-compose**
4. Revenir à l'exemple précédent et construire l'image **myapp** depuis le fichier de composition.



- Pour ceux qui réussiront cet exercice, mettez votre code sur GitHub dans un nouveau repository privé, m'ajouter comme collaboratrice (**hibanajjar998**)
- Mettre le lien du repository ici: -----