

- **Qu'est-ce que la science des données ?**

Extraction de connaissances à partir de données (modélisation, prédiction, visualisation)

- **Défis fréquents :**

- Environnements non reproductibles
- Dépendances complexes
- Problèmes de déploiement

- **Solution : Docker et la conteneurisation**

- **Qu'est-ce que Docker ?**

- Docker = Plateforme pour automatiser le déploiement d'applications dans des **conteneurs**
- Les conteneurs sont :
 - Légers
 - Portables
 - Isolés
- Image Docker = Environnement prêt à l'emploi

Pourquoi Docker en science des données ?

- **Reproductibilité** : mêmes résultats sur toute machine
- **Isolation** : chaque projet dans son propre environnement
----- > Réduction du temps de mise en place des environnements / moins d'erreur de compatibilité
- **Collaboration** : partage d'environnements avec l'équipe
- **Automatisation** : intégration dans des pipelines CI/CD
----- > Meilleure collaboration entre data scientists, devs et ops
- **Portabilité** : passage facile de l'environnement local au cloud



Cloud computing?

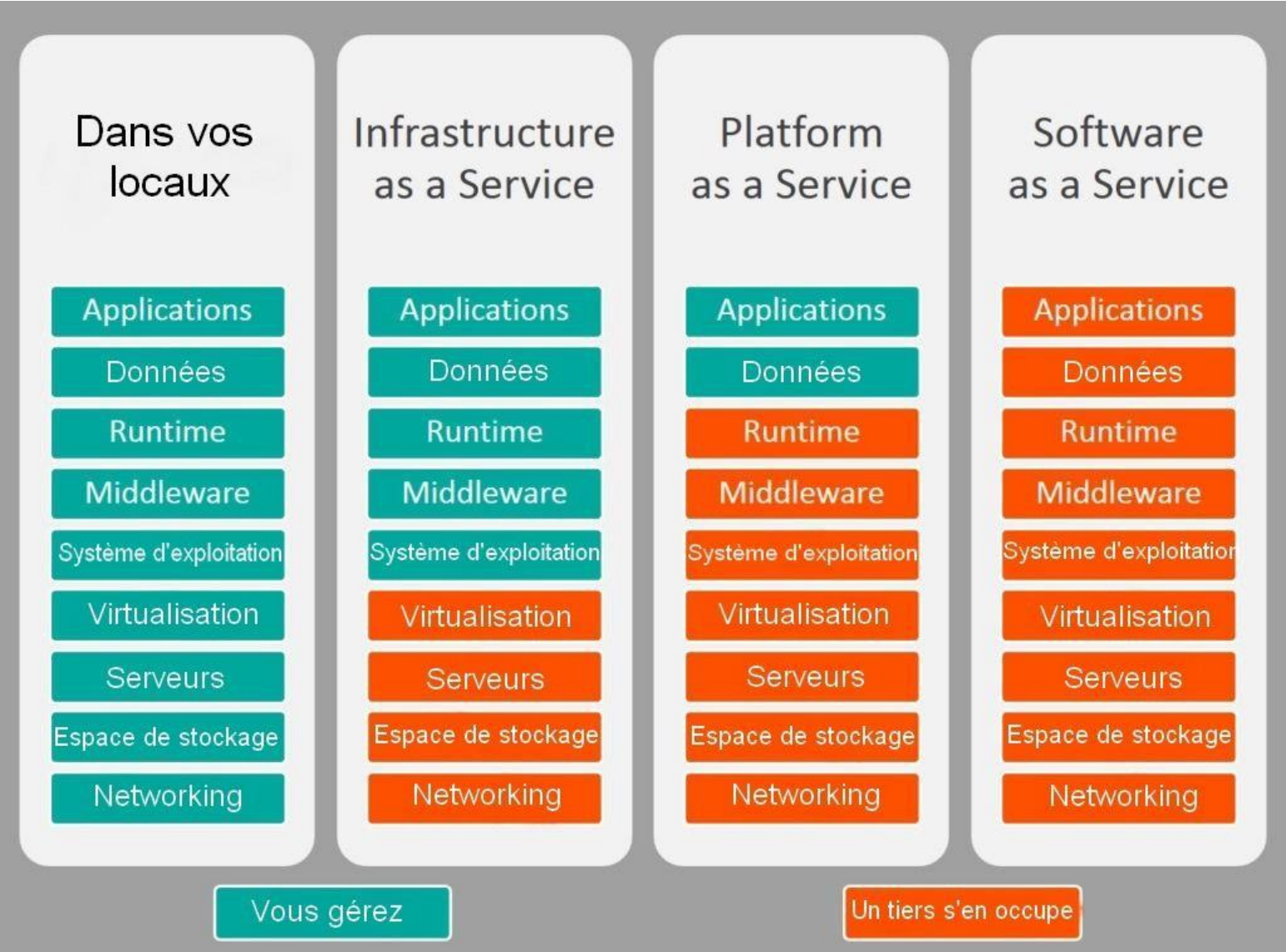
Le **cloud**, ou **informatique en nuage**, désigne l'utilisation de **serveurs distants** (généralement via Internet) pour stocker, gérer, traiter et accéder à des données et des logiciels — au lieu de les faire fonctionner localement sur un ordinateur personnel ou un serveur interne.

Caractéristiques clés du cloud :

- Accès à distance
→ Vos fichiers, applications ou bases de données sont accessibles depuis n'importe quel appareil connecté à Internet.
- Stockage en ligne
→ Plus besoin de clé USB ou de disque dur local : vos données sont dans le cloud (ex : Google Drive, Dropbox).

Ressources à la demande

- Vous pouvez louer de la puissance de calcul, du stockage, ou des services logiciels selon vos besoins.
- Évolutivité
→ Vous pouvez facilement augmenter ou réduire les ressources utilisées, sans acheter du matériel.
- Paiement à l'usage
→ Vous ne payez que pour ce que vous utilisez (comme l'électricité).



Les types du cloud

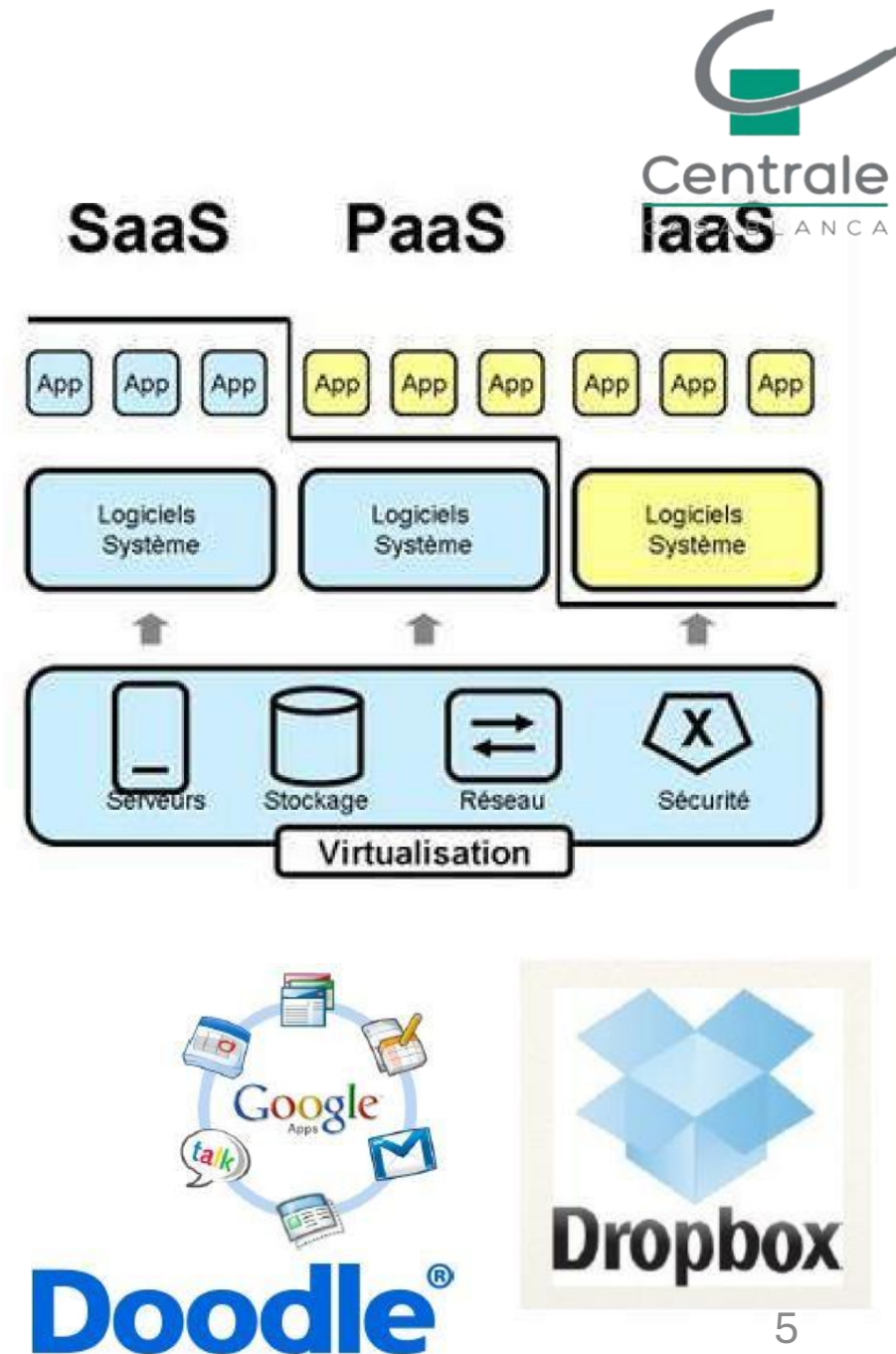
SaaS(Software as a Service)

SaaS est l'acronyme pour Software as a Service. Le mode SaaS est un mode d'utilisation d'une solution logicielle qui se fait en utilisant l'application à distance qui est hébergée par l'éditeur.

La solution logicielle étant utilisée, le plus souvent, à partir d'un simple navigateur Internet, elle permet à l'entreprise d'être déchargée de toute contrainte d'installation, de mise à jour ou de maintenance technique. Elle permet également d'être utilisée par les collaborateurs en situation de mobilité.

La mise à disposition d'une solution SaaS peut être facturée par abonnement ou proportionnellement à l'usage et peut parfois comporter des frais de personnalisation et de mise à disposition du service.

Dans le domaine du webmarketing, les plateformes de gestion des campagnes email, les outils de web analytique et les serveurs publicitaires sont généralement proposés en mode SaaS.

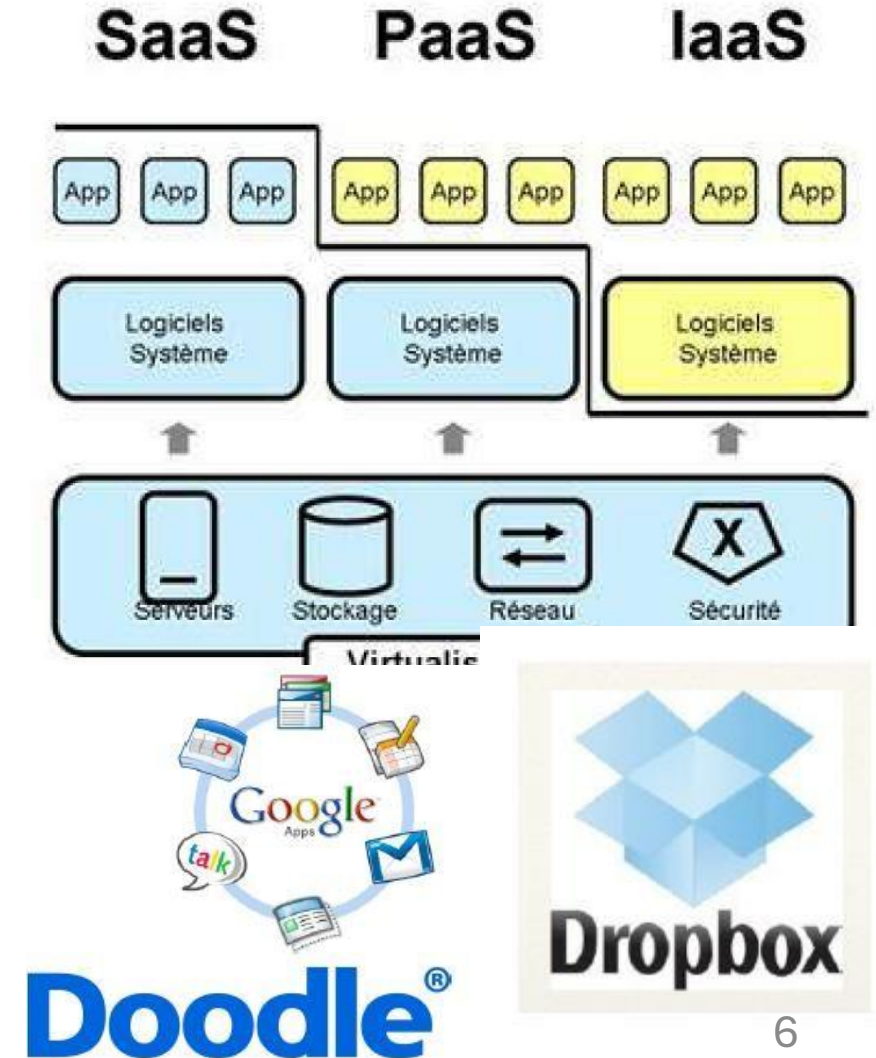


Les types du cloud

SAAS(Software as a Service)

Avantage : plus d'installation, plus de mise à jour (elles sont continues chez le fournisseur), plus de migration de données etc. Paiement à l'usage. Test de nouveaux logiciels avec facilité.

Inconvénient : limitation par définition au logiciel proposé. Pas de contrôle sur le stockage et la sécurisation des données associées au logiciel. Réactivité des applications Web pas toujours idéale.

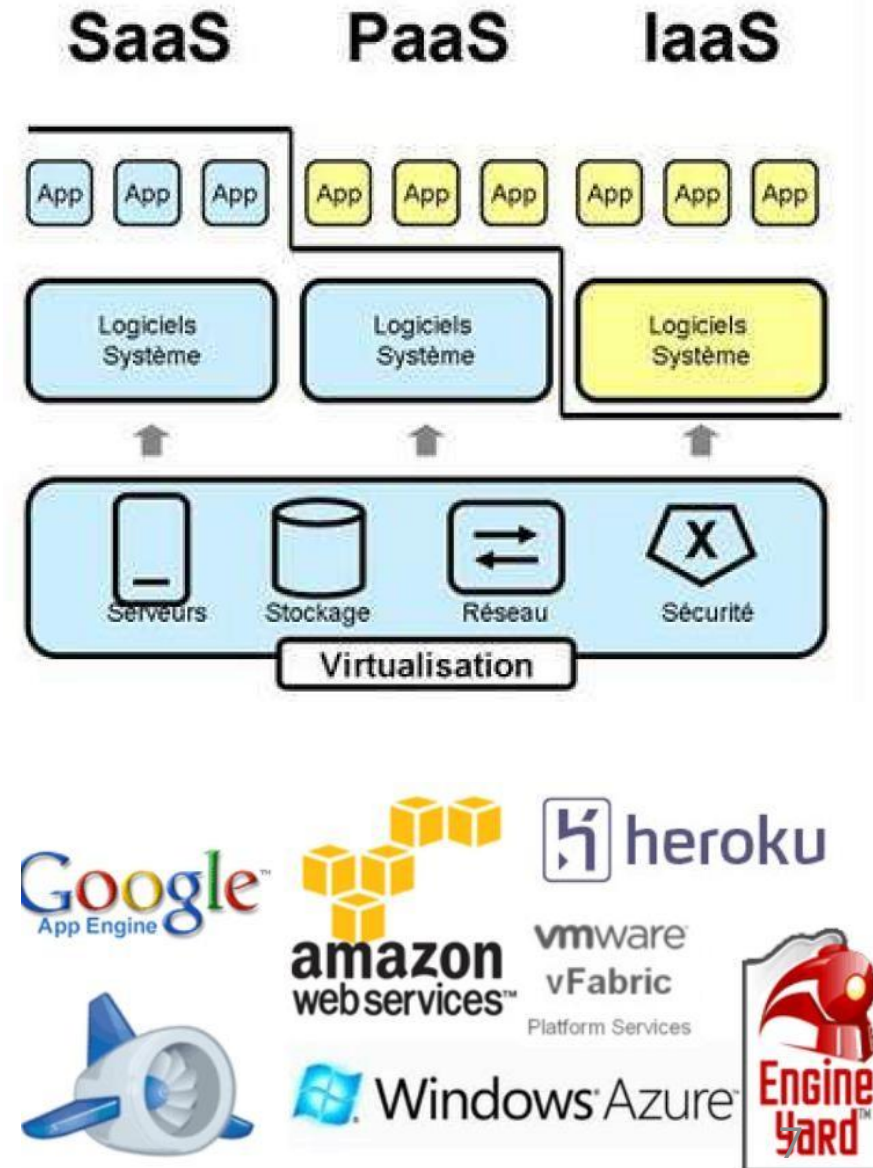


Les types du cloud

PAAS(Platform as a Service)

Il s'agit des plateformes du nuage, regroupant principalement les serveurs mutualisés et leurs systèmes d'exploitation. En plus de pouvoir délivrer des logiciels en mode SaaS, le PaaS dispose d'environnements spécialisés au développement comprenant les langages, les outils et les modules nécessaires.

Ces environnements sont hébergés par un prestataire basé à l'extérieur de l'entreprise ce qui permet de ne disposer d'aucune infrastructure et de personnel de maintenance et donc de pouvoir se consacrer au développement.



Les types du cloud

PAAS(Platform as a Service)

Avantage : le déploiement est automatique, pas de logiciel supplémentaire à acheter ou à installer.

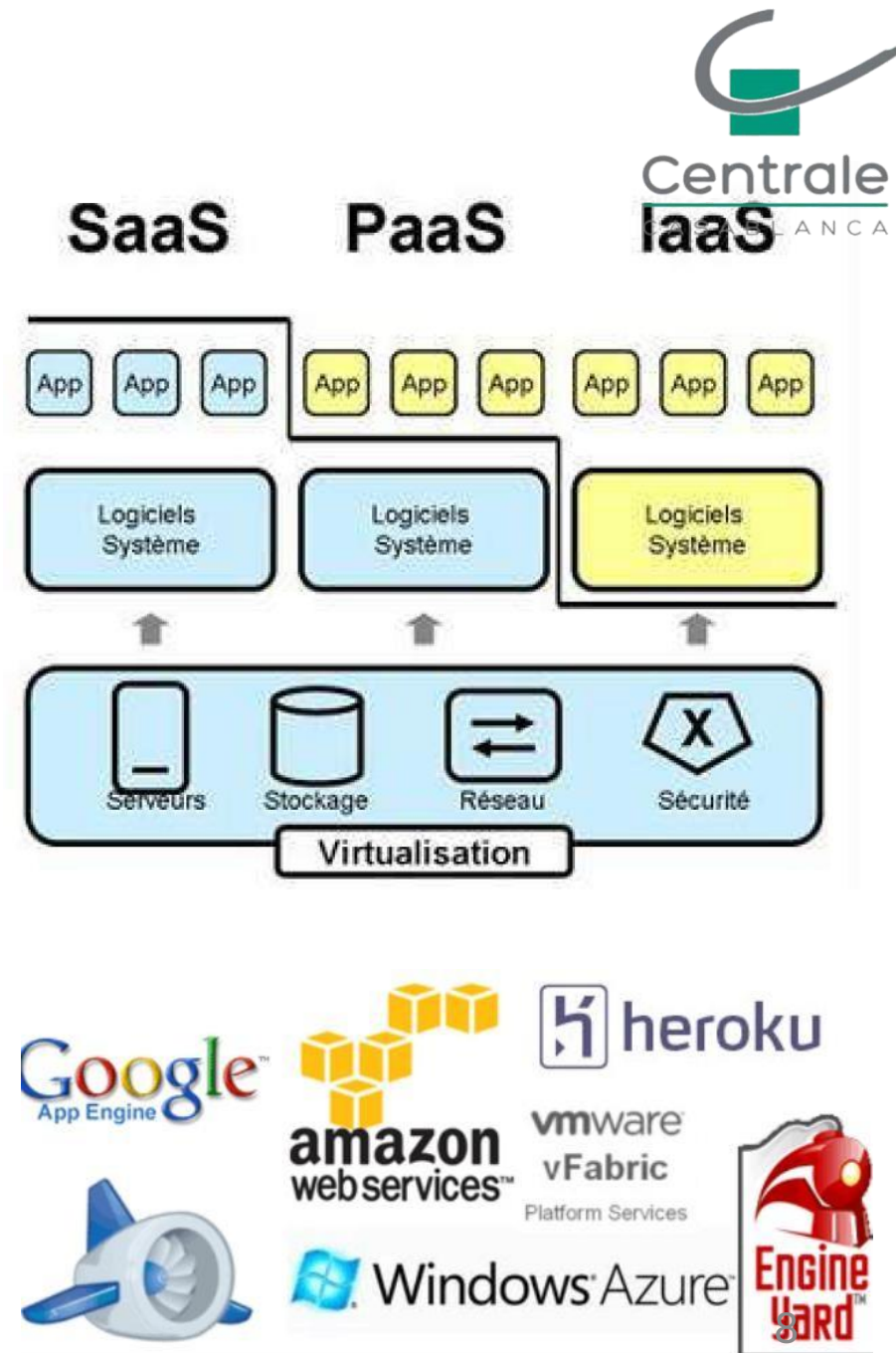
Inconvénient : limitation à une ou deux technologies (ex. : Python ou Java pour Google AppEngine, .NET pour Microsoft Azure, propriétaire pour force.com).

Pas de contrôle des machines virtuelles sous jacentes.

Convient uniquement aux applications Web.

Les cibles sont les développeurs.

Google App Engine est le principal acteur proposant ce genre d'infrastructures.

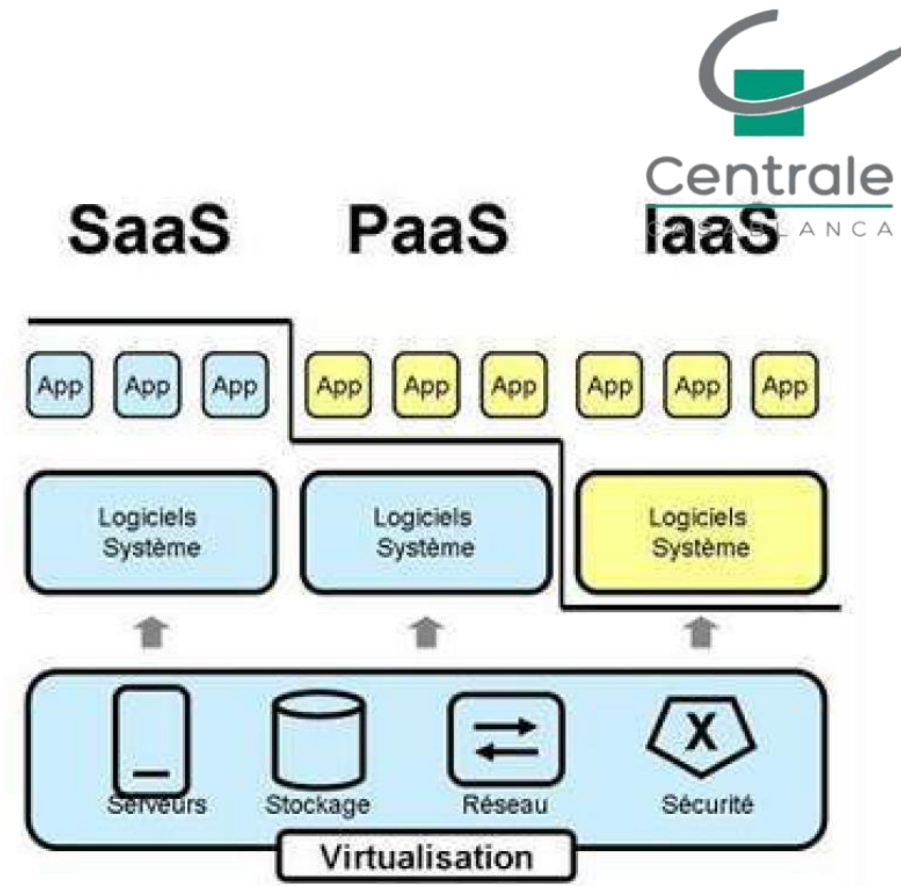


Les types d

IAAS(Infrastructure as a Service)

Il s'agit de la mise à disposition, à la demande, des ressources d'infrastructures dont la plus grande partie est localisée à distance dans des Datacenters.

L'IaaS permet l'accès aux serveurs et à leurs configurations pour les administrateurs de l'entreprise. Le client a la possibilité de louer des clusters, de la mémoire ou du stockage de données. Le coût est directement lié au taux d'occupation.



Les types de cloud

Avantage : grande flexibilité, contrôle total des systèmes (administration à distance par SSH ou Remote Desktop, RDP), qui permet d'installer tout type de logiciel métier.

Inconvénient : besoin d'administrateurs système comme pour les solutions de serveurs classiques sur site.

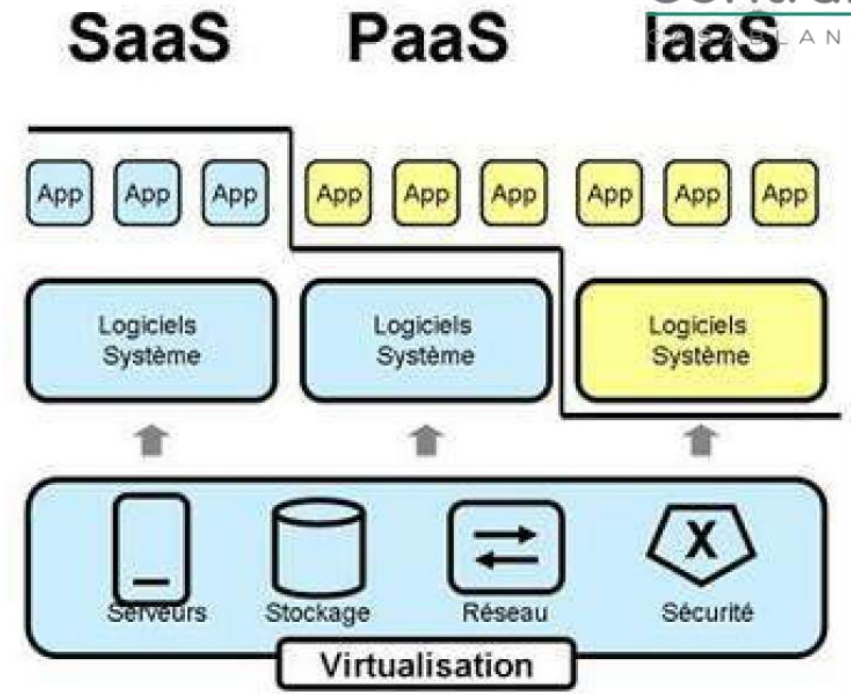
Pas de format std. pour les machines.

Temps et difficulté pour créer une VM

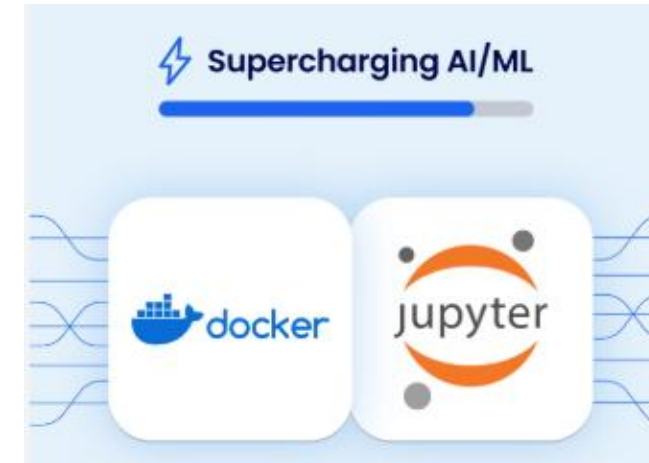
Les cibles sont les responsables d'infrastructures informatiques.

Amazon EC2 est le principal qui propose ce genre d'infrastructures.

OpenStack est un exemple d'infrastructure.



Amazon
EC2



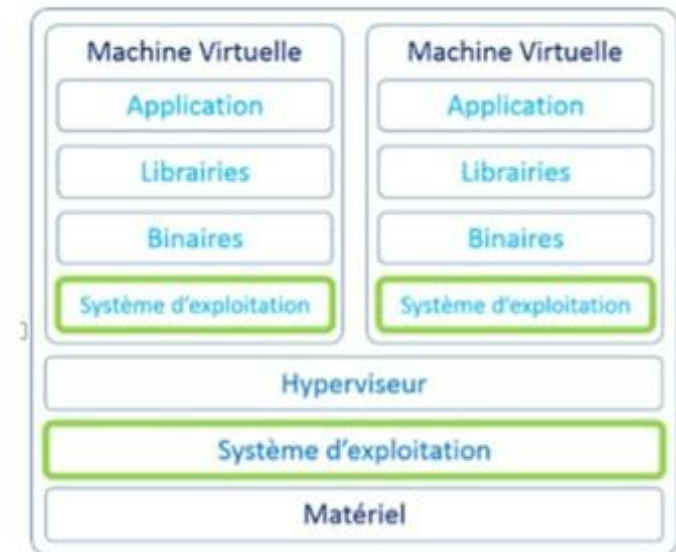
- Docker, c'est quoi ?
 - Docker est une technologie de conteneurisation qui:
 - Facilite la gestion de dépendance au sein d'un projet et ce, à tous les niveaux (développement et déploiement).
 - Est disponible sur Linux, Windows et Mac OS.

Le mécanisme de Docker se centre autour des conteneurs et de leur orchestration, et c'est en cela que la conteneurisation se différencie de la virtualisation.

Cas d'étude: <https://www.kaggle.com/datasets/yasserh/housing-prices-dataset>

Conteneurisation vs Virtualisation?

- Virtualisation :
 - Dans la virtualisation, un hyperviseur (ou un logiciel de virtualisation) est utilisé pour créer des machines virtuelles (VM) sur un serveur physique.
 - Chaque machine virtuelle fonctionne comme un système informatique autonome, avec son propre système d'exploitation, ses applications et ses ressources allouées.
 - La virtualisation offre une isolation forte entre les différentes machines virtuelles, car elles sont complètement séparées les unes des autres.
 - Cependant, la virtualisation est souvent plus lourde en termes de ressources système, car chaque VM nécessite un système d'exploitation complet.



Virtualisation

Hyperviseur de type 1 – "bare metal" (sur métal nu)

Un hyperviseur de type 1 s'installe directement sur le matériel physique (pas de système d'exploitation hôte entre les deux).

Exemples :

- VMware ESXi
- Microsoft Hyper-V (en mode serveur)
- Xen
- KVM (natif Linux)

Caractéristiques :

- Très performant
- Moins de surcouche logicielle
- Utilisé dans les centres de données, serveurs cloud, environnements professionnels

Avantages :

- Meilleure efficacité (accès direct au matériel)
- Plus de stabilité et de sécurité
- Idéal pour la production à grande échelle

Inconvénients :

- Installation et gestion plus complexes
- Moins adapté aux débutants

Hyperviseur de type 2 – "hosted" (hébergé)

Un hyperviseur de type 2 s'installe au-dessus d'un système d'exploitation hôte (comme une application classique).

Exemples :

- Oracle VirtualBox
- VMware Workstation / Fusion
- Parallels Desktop
- Hyper-V (en mode desktop sur Windows 10/11)

Caractéristiques :

- Facile à installer sur un PC personnel
- Convivial pour le développement, les tests ou la formation

Avantages :

- Simple à mettre en œuvre
- Idéal pour les tests, démos, environnements de dev

Inconvénients :

- Moins performant (car passe par l'OS hôte)
- Dépend des ressources de l'OS principal

Conteneurisation vs Virtualisation?



Conteneurisation

Conteneurisation :

- Dans la conteneurisation, des conteneurs sont utilisés pour exécuter des applications et leurs dépendances de manière isolée sur un système hôte.
- Les conteneurs partagent le même noyau du système d'exploitation de l'hôte, mais ils sont isolés au niveau des processus, des systèmes de fichiers et des ressources.
- Les conteneurs sont généralement plus légers que les machines virtuelles car ils n'incluent pas de système d'exploitation complet, mais partagent plutôt les ressources du système d'exploitation hôte.
- Les conteneurs sont souvent utilisés pour créer des environnements de développement cohérents, pour automatiser le déploiement d'applications.



Avantages Docker?

- On peut exécuter plusieurs applications en utilisant différents OS sur une même machine hôte, sans avoir à créer les systèmes en entier ;
- Les conteneurs peuvent être migrés n'importe où en générant leurs images une seule fois ;
- Docker est multiplateforme, on peut l'exécuter sur Windows, MacOS et Linux, bien sûr ;
- Les langages les plus populaires et les plus utilisés sont pris en charge par lui ;
- Il est léger, rapide, performant, facile à utiliser et moins gourmand en ressources ;
- Les applications conteneurisées sont faciles à déployer, à exécuter, à maintenir et à partager ;
- La gestion des ressources utilisées par les applications est moins fastidieuse ;
- La virtualisation coûte cher en matière de ressources, car elles doivent être prédéfinies à l'avance, ce qui pose quelques problèmes au niveau de l'extension des applications. Et c'est là qu'interviennent la plateforme de conteneurs Docker.



Les trois concepts clés de Docker

Il existe trois concepts clés dans Docker :

les conteneurs, les images et les fichiers Docker (Dockerfile).

Ces trois concepts forment la chaîne de déploiement de conteneurs et possèdent deux sens de lecture. Dans notre cas, nous allons partir du résultat final, les conteneurs, pour remonter jusqu'à leur origine, les Dockerfile, en passant par les images qui se trouvent entre les deux.





- Les trois concepts clés de Docker

Premier concept clé de Docker : les conteneurs

- **Un conteneur est un espace dans lequel une application tourne avec son propre environnement. Les applications qu'un conteneur peut faire tourner sont de tous types : site web, API, db, etc.**
- Chaque conteneur est une instance d'une **image**. Il possède son propre environnement d'exécution et donc ses propres répertoires. Nous verrons comment déployer un conteneur.
- La force des conteneurs Docker réside dans le grand panel de **configurations** qui nous est proposé. Nous pouvons ainsi **changer le port** d'un conteneur à l'intérieur du système Docker mais également les faire facilement **communiquer** entre eux.



Les trois concepts clés de Docker

Deuxième concept clé de Docker : les images

- **Les images représentent le contexte que plusieurs conteneurs peuvent exécuter.** Elles sont aux conteneurs ce que les classes sont aux objets en Programmation Orientée Objet : un moule.
- Par contexte, il faut entendre la disposition des dossiers, la disponibilité de certaines librairies, le bindage de certains ports en interne et en externe du conteneur mais également un ensemble de commandes à exécuter au lancement d'un conteneur.
- Tout comme les conteneurs, nous regarderons les différentes manières d'obtenir des images.



Les trois concepts clés de Docker

Troisième concept clé de Docker : le Dockerfile

Un Dockerfile est un fichier qui liste les instructions à exécuter pour build une image. Il est lu de haut en bas au cours du processus de build.



Troisième concept clé de Docker : le Dockerfile

- 1.**FROM**: Spécifie l'image de base à utiliser. C'est la première instruction dans un Dockerfile.
- 2.**LABEL**: Permet d'ajouter des métadonnées à l'image comme le mainteneur, la version, la description, etc.
- 3.**RUN**: Exécute des commandes dans le conteneur pendant la construction de l'image. Par exemple, installer des paquets, cloner des dépôts Git, etc.
- 4.**COPY / ADD**: Copie des fichiers depuis le système de fichiers de l'hôte vers l'image. ADD a une fonctionnalité supplémentaire de décompression automatique pour les fichiers tar.
- 5.**WORKDIR**: Définit le répertoire de travail pour toutes les instructions RUN, CMD, ENTRYPOINT, COPY et ADD suivantes.
- 6.**ENV**: Définit des variables d'environnement pour le conteneur.
- 7.**EXPOSE**: Expose des ports du conteneur.
- 8.**CMD / ENTRYPOINT**: Définit la commande par défaut à exécuter lorsque le conteneur est démarré. CMD fournit des arguments par défaut qui peuvent être remplacés au moment du démarrage du conteneur, tandis que ENTRYPOINT définit une commande qui ne peut pas être remplacée, mais qui peut être complétée par des arguments supplémentaires lors du démarrage du conteneur.
- 9.**VOLUME**: Déclare un point de montage pour un volume.



1. Vérifier les prérequis

Avant d'installer Docker :

- Un système 64 bits est requis.
- Virtualisation activée dans le BIOS (surtout sur Windows).
- Droits administrateur.

2. Télécharger Docker Desktop

Allez sur le site officiel, Choisissez votre système d'exploitation :Windows, macOS (Intel ou Apple Silicon) ou Linux (Ubuntu, Debian, etc.)

3. Installer Docker

🟦 Sur Windows :Double-cliquez sur l'installateur .exe. Suivez les étapes (autoriser WSL 2 si proposé).Redémarrez si demandé.

🍏 Sur macOS :Ouvrez le fichier .dmg téléchargé. Glissez Docker dans le dossier Applications. Lancez Docker Desktop.

🐧 Sur Linux (ex. Ubuntu) :

```
bash
```

```
sudo apt update  
sudo apt install docker.io  
sudo systemctl start docker  
sudo systemctl enable docker
```

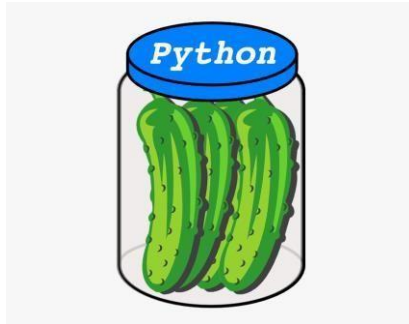
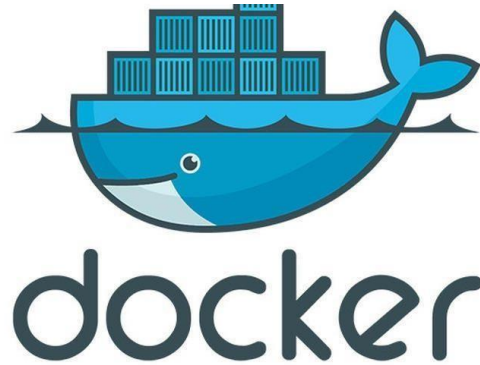


Docker Desktop, c'est quoi ?

- **Docker Desktop** est une application officielle fournie par Docker qui permet d'utiliser **Docker facilement** sur un poste **Windows** ou **macOS**.
- Elle fournit une interface graphique et tous les outils nécessaires pour créer, exécuter et gérer des conteneurs **en local**.

Docker Hub, c'est quoi ?

Docker Hub est un **registre en ligne** (ou *registry*) où vous pouvez **trouver, stocker, partager et gérer des images Docker**.
C'est comme une **bibliothèque centrale d'images de conteneurs**.



Exemple d'application

1. CUDA (Compute Unified Device Architecture)

CUDA est une plateforme et une API développée par NVIDIA qui permet d'exploiter les GPU pour effectuer des calculs parallèles. Elle donne accès directement aux cœurs du GPU et permet aux développeurs de créer des applications performantes en utilisant des langages de programmation comme C, C++, Python, ou encore Fortran.

2. cuDNN (CUDA Deep Neural Network Library)

cuDNN est une bibliothèque GPU optimisée, conçue spécifiquement pour les réseaux de neurones profonds (Deep Neural Networks). Elle est utilisée par des frameworks d'apprentissage automatique comme TensorFlow, PyTorch, Keras, et MXNet pour accélérer les calculs liés aux réseaux de neurones.

Dans ce jeu de données chaque ligne représente les mesures de l'électrocardiogramme d'un patient. La dernière colonne permet de labelliser les patients :

- "0" lorsqu'ils sont sains
- "1" lorsqu'ils comportent une anomalie.

Les autres colonnes représentent les différents points de mesures de l'électrocardiogramme pour chaque patient.

Consignes

- Chargez le dataset "[ecg.csv](#)". Observez la dernière colonne du jeu de données pour vérifier que l'échantillon est bien équilibré (c'est à dire que la proportion des électrocardiogrammes sains et comportant une anomalie sont sensiblement identiques).
- Vous pouvez maintenant concevoir votre premier modèle de réseau de neurones et l'entraîner.

https://drive.google.com/file/d/1KRW7_lHLtaoPUjZZaaPpiEzWv_RyOmG7/view?usp=sharing

Concevoir le modèle

Entraîner le modèle

Répondre à la question

Quel est le niveau de précision (accuracy) de notre modèle sur notre jeu de données de test ?

Tourner la solution sur un conteneur docker

```
# Base image with Ubuntu and CUDA support  
FROM nvidia/cuda:11.8.0-cudnn8-runtime-ubuntu22.04
```

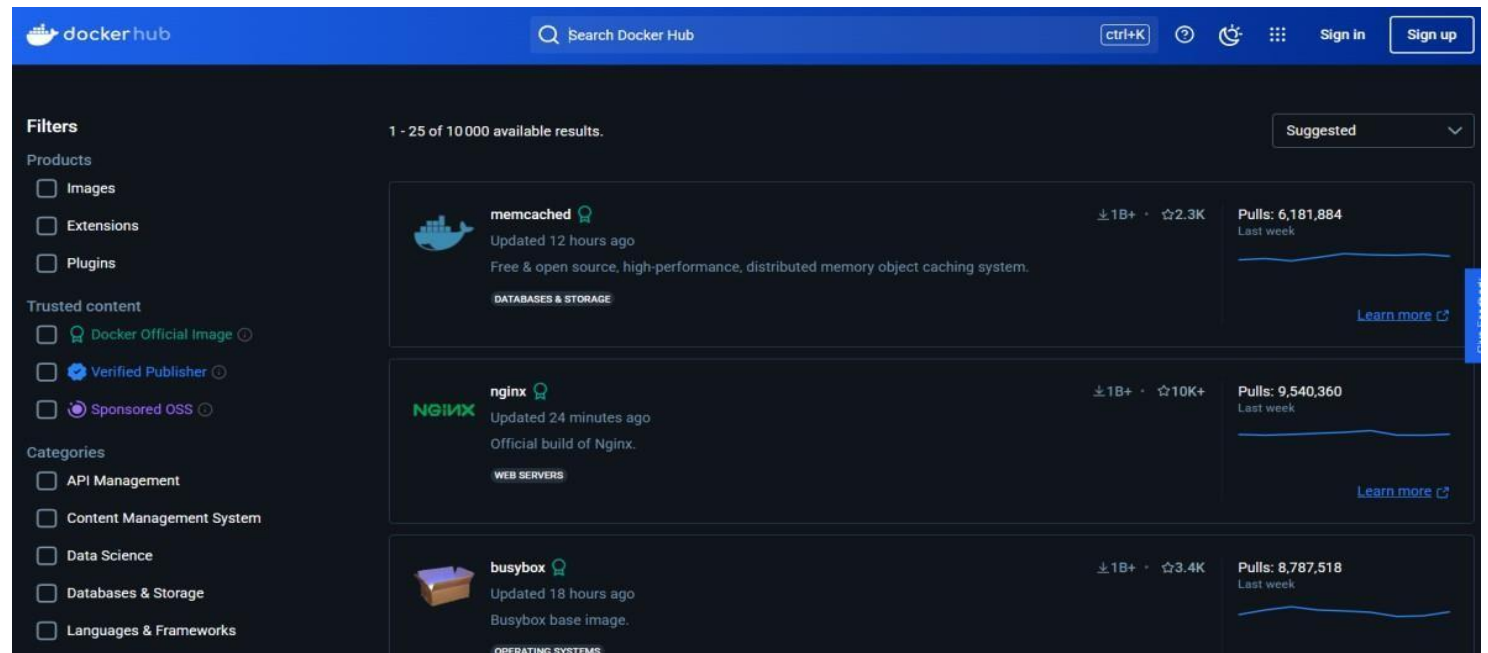
```
# Install basic dependencies  
RUN apt-get update && apt-get install -y --no-install-recommends \  
python3-pip \  
python3-dev \  
build-essential \  
wget \  
curl \  
git \  
&& rm -rf /var/lib/apt/lists/*
```

1. Base Image :

- image officielle de NVIDIA contenant CUDA 11.8 et cu8, compatible avec TensorFlow.

2. Dépendances :

- Les outils de développement sont nécessaires pour installer certaines bibliothèques Python.
- curl, wget, et git sont souvent utiles dans les environnements de développement.



Tourner la solution sur un conteneur docker

```
# Create symbolic link for python3  
RUN ln -s /usr/bin/python3 /usr/bin/python
```

```
# Upgrade pip  
RUN pip install --upgrade pip
```

```
# Install TensorFlow (GPU version), Keras, Scikit-learn, and Pandas  
RUN pip install tensorflow keras scikit-learn pandas
```

```
# Verify installation  
RUN python -c "import tensorflow as tf; print('TensorFlow:', tf.__version__)" CC \  
python -c "import keras; print('Keras:', keras.__version__)" CC \  
python -c "import sklearn; print('Scikit-learn:', sklearn.__version__)" CC \  
python -c "import pandas as pd; print('Pandas:', pd.__version__)"
```

```
# Set working directory  
WORKDIR /workspace
```

```
# Default command  
CMD ["python"]
```

3. Symbolic Link :

- Cela permet d'utiliser la commande python (par défaut Python 2) pour pointer vers Python 3.

4. Python Libraries :

- TensorFlow, Keras, Scikit-learn et Pandas sont installés via pip.

5. Vérification :

- Des commandes Python simples vérifient que les bibliothèques sont correctement installées.

5. Working Directory :

- Le dossier de travail est défini comme /workspace, où les utilisateurs peuvent déposer leurs scripts.

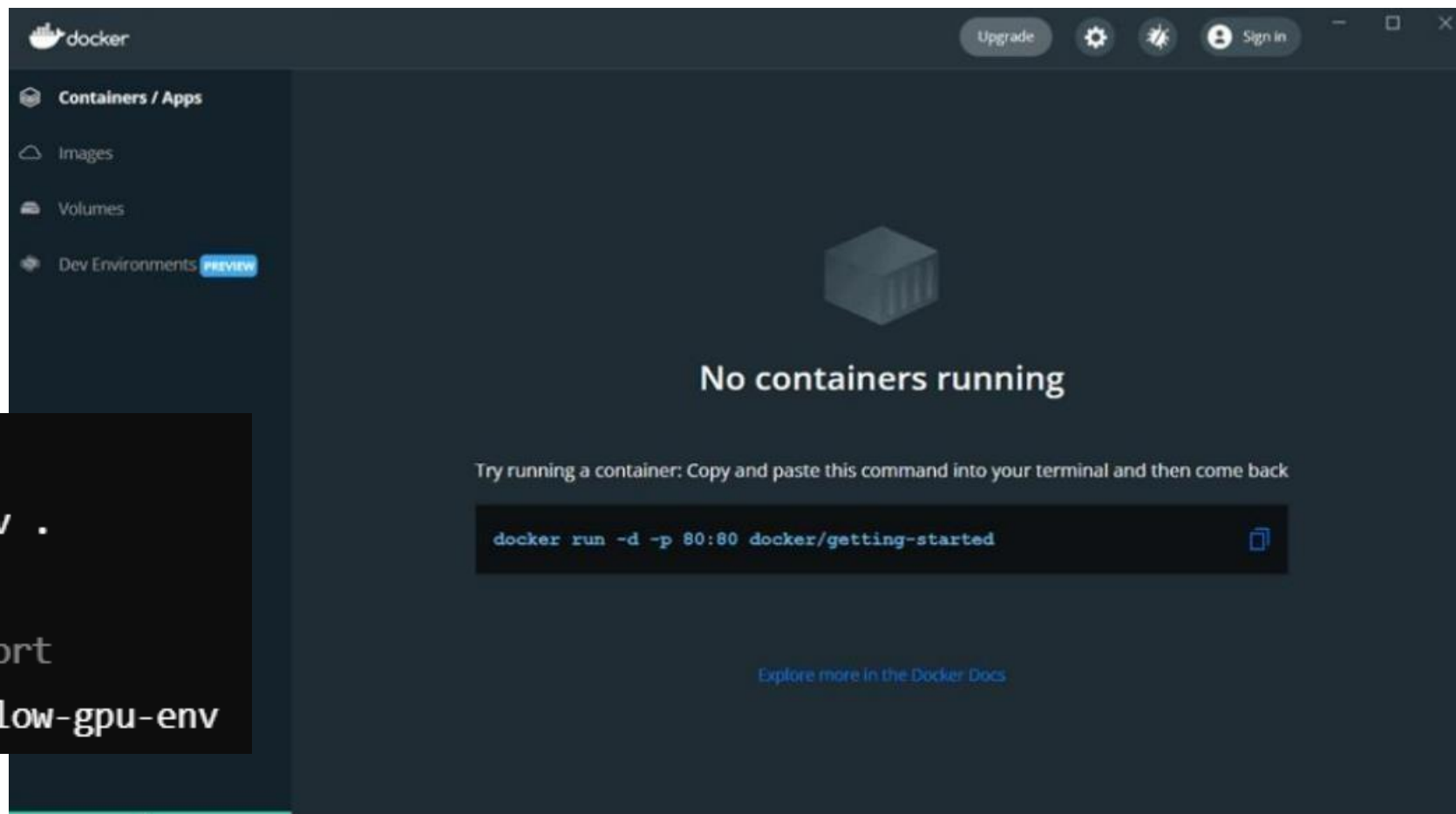
Tourner la solution sur un conteneur docker

Pré-requis :

- Assurez-vous que le pilote NVIDIA est installé sur votre machine hôte.
- Installez [NVIDIA Container Toolkit](#) pour permettre aux conteneurs d'accéder au GPU.

```
# Build the Docker image
docker build -t tensorflow-gpu-env .

# Run the container with GPU support
docker run --gpus all -it tensorflow-gpu-env
```



Tourner la solution sur un conteneur docker

```
# Set working directory  
WORKDIR /workspace
```

```
COPY ..
```

```
# Default command  
CMD ["python"]
```

L'instruction COPY nous permet de copier un fichier (ou des fichiers) du système hôte dans l'image. Cela signifie que les fichiers deviennent une partie de chaque conteneur créé à partir de cette image.

un conteneur est éphémère (c'est-à-dire qu'il ne garde pas de données une fois arrêté)

```
docker run --rm --gpus all -it \ --mount type=bind, source=/home/user/data,  
target=/workspace/data
```

Tourner la solution sur un conteneur docker

```
# Définir des variables d'environnement ENV  
DEBIAN_FRONTEND=noninteractive  
ENV TZ=Etc/UTC  
ENV PYTHONUNBUFFERED=1  
ENV PATH="/opt/myapp/bin:$PATH"
```

```
docker run --rm -e PATH=test tensorflow-gpu-env
```

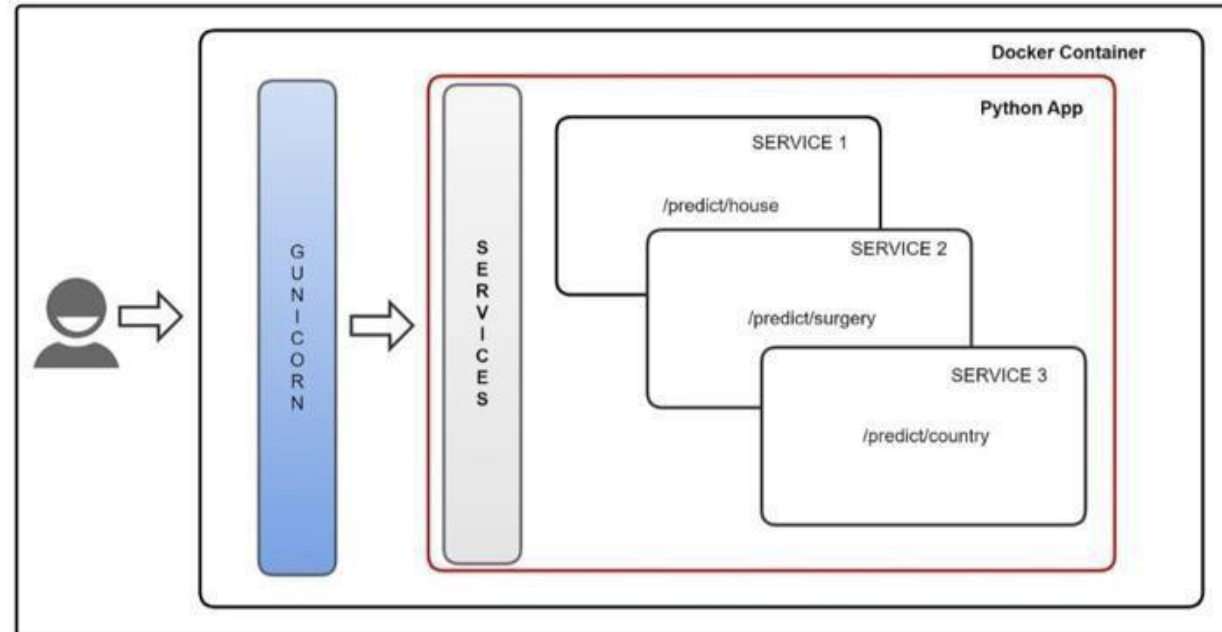
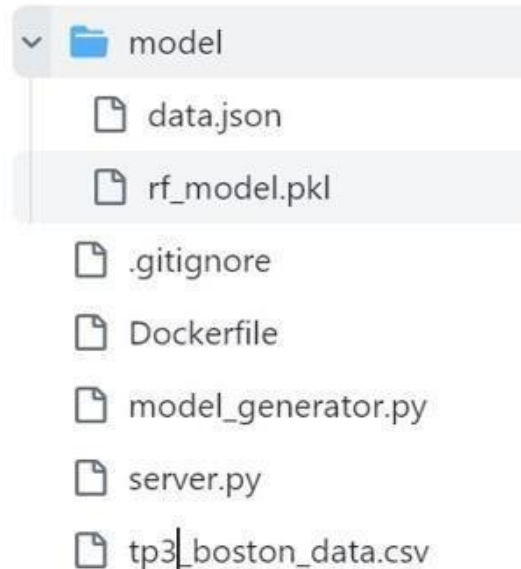
DEBIAN_FRONTEND=noninteractive : Évite les invites interactives pendant l'installation des paquets (utile pour les conteneurs).
TZ=Etc/UTC : Définit le fuseau horaire pour éviter les erreurs liées au temps.
PYTHONUNBUFFERED=1 : Désactive le buffering de la sortie Python pour que les journaux apparaissent immédiatement dans la console.
PATH="/opt/myapp/bin:\$PATH" : Ajoute un chemin personnalisé au PATH du conteneur.

Déployer la solution sur un conteneur docker

Modèles machine learning avec Flask et Docker

Comment?

- Package Python Flask utilisé pour exposer notre modèle d'apprentissage automatique en tant qu'API REST. Flask n'est pas un serveur complet prêt pour la production .
- Nous allons donc utiliser gunicorn qui est un serveur HTTP python pour exposer notre API REST.



Modèles machine learning avec Flask et Docker

FLASK et REST API?

1. Flask :

1. Flask est un framework web Python qui facilite la création d'applications web.
2. Il est minimaliste et offre une grande flexibilité, ce qui permet aux développeurs de construire des applications de toutes tailles, de petites applications simples à des applications web plus complexes.
3. Flask fournit des outils pour la gestion des routes, la gestion des requêtes HTTP, le rendu de templates HTML, la gestion des cookies, etc.

2. API RESTful :

1. Une API RESTful est une interface de programmation d'application qui suit les principes de l'architecture REST.
2. Elle utilise les principes fondamentaux de l'HTTP tels que les méthodes GET, POST, PUT, DELETE pour effectuer des opérations CRUD (Create, Read, Update, Delete) sur les ressources.
3. Les ressources sont identifiées par des URI (Uniform Resource Identifiers), et la manipulation de ces ressources se fait à travers ces URI à l'aide des méthodes HTTP.
4. Les API RESTful sont souvent utilisées pour construire des services web qui peuvent être consommés par d'autres applications ou services, qu'il s'agisse d'applications web, d'applications mobiles ou d'autres services.

Modèles machine learning avec Flask et Docker

1. Construire le modèle d'apprentissage automatique

```
# Data Manipulation libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPRegressor
import joblib

df = pd.read_csv('tp3_boston_data.csv') # Load the dataset

df_x = df[['crim', 'zn', 'indus', 'chas', 'nox', 'rm', 'age', 'dis', 'rad', 'tax', 'ptratio', 'lstat']]
df_y = df[['medv']]

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(df_x)

df_x_scaled = scaler.transform(df_x)
df_x_scaled = pd.DataFrame(df_x_scaled, columns=df_x.columns)
X_train, X_test, Y_train, Y_test = train_test_split(df_x_scaled, df_y, test_size = 0.33, random_state = 5)

mlp = MLPRegressor(hidden_layer_sizes=(60), max_iter=1000)
mlp.fit(X_train, Y_train)
Y_predict = mlp.predict(X_test)

#Saving the machine learning model to a file
joblib.dump(mlp, "model/rf_model.pkl")
```



joblib est une bibliothèque Python utilisée pour la sérialisation d'objets Python (tels que des modèles d'apprentissage automatique) dans des fichiers et leur chargement ultérieur en mémoire. Elle est souvent utilisée en conjonction avec des bibliothèques telles que scikit-learn pour la sauvegarde et le chargement de modèles d'apprentissage automatique.

Modèles machine learning avec Flask et Docker

2. Création de l'API HTTP REST

```
from flask import Flask, jsonify, request
import pandas as pd
import joblib
server.py

app = Flask(__name__)

@app.route("/predict", methods=['POST'])
def do_prediction():
    json = request.get_json()
    model = joblib.load('model/rf_model.pkl')
    df = pd.DataFrame(json, index=[0])

    from sklearn.preprocessing import StandardScaler
    scaler = StandardScaler()
    scaler.fit(df)

    df_x_scaled = scaler.transform(df)

    df_x_scaled = pd.DataFrame(df_x_scaled, columns=df.columns)
    y_predict = model.predict(df_x_scaled)

    result = {"Predicted House Price" : y_predict[0]}
    return jsonify(result)

if __name__ == "__main__":
    app.run(host='0.0.0.0')
```

- utiliser notre modèle d'apprentissage automatique exporté pour créer une API REST simple.
- utiliser Python Flask pour exposer l'API via une interface Web.
- Dans cette logique python, il acceptera les paramètres nécessaires pour prédire le prix cible de la maison à l'aide du modèle entraîné stocké dans le fichier « rf_model.pkl ».
- Définir une ressource HTTP appelée « predict » pour accepter les paramètres permettant de prédire le prix de l'immobilier au format json. Cela peut inclure le nombre de pièces dans une maison, la distance de la ville principale, etc.

Modèles machine learning avec Flask et Docker

3. Fichier Docker du service

Dockerfile

```
#Utiliser l'image de base avec python 3.7
FROM python:3.7

#Définir notre répertoire de travail comme app
WORKDIR /app

#Installation des packages python pandas, scikit-learn et
gunicorn
RUN pip install pandas scikit-learn flask gunicorn

# Copiez le répertoire models et les fichiers server.py
ADD ./models ./models
ADD server.py server.py

#Exposition du port 5000 de le conteneur
EXPOSE 5000

#Démarrage de l'application python
CMD ["gunicorn", "--bind", "0.0.0.0:5000", "server:app"]
```

- La partie la plus importante de la création de l'image Docker consiste à définir le contenu du Dockerfile qui inclut toutes les dépendances requises ainsi qu'à copier le contenu de votre application dans le conteneur.

Modèles machine learning avec Flask et Docker

4. Créer et exécuter le conteneur Docker

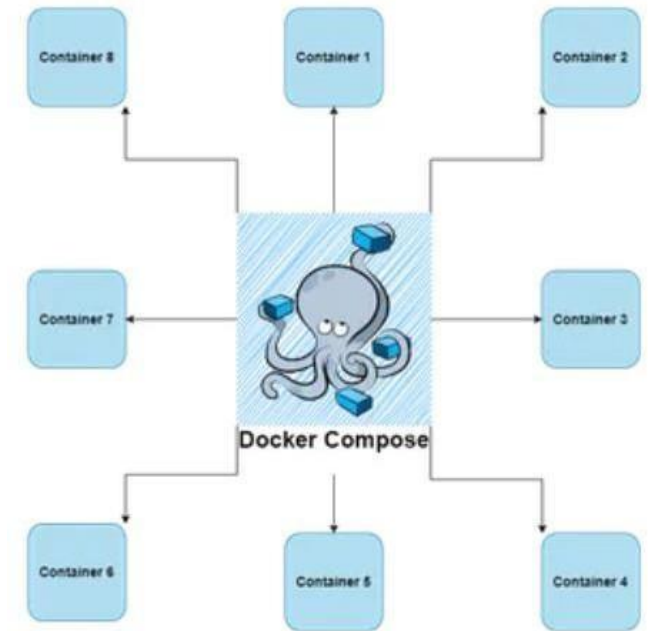
```
#Construisez l'image docker avec le nom ml-model  
docker build -t ml-model .
```

```
#Lancez un conteneur en arrière plan  
docker run -d -p 5000:5000 ml-modèle
```

```
#Appeler le service de modèle d'apprentissage automatique à l'aide de curl ou de n'importe quel client REST  
curl --location --request POST ' http://localhost:5000/predict' --header 'Content-Type : application/json' --  
data-raw '{  
  "crim": 0.85204,"zn":0, "indus": 8.14,"chas":0,"nox": 0.538,"rm":5.965,"age": 89.2,"dis":4.0123,"rad":  
  4,"tax":307,"ptratio ": 21,"lstat":13.83  
}'
```

Docker compose

- Docker Compose est un outil qui permet de définir et d'exécuter des applications multi-conteneurs, en établissant même une couche de communication entre ces services.
- Il permet de déclarer les services, les réseaux, les volumes et les configurations d'une application dans un seul fichier appelé docker-compose.yml.
- Il suit une syntaxe déclarative pour définir les services et leurs configurations, qui peuvent être configurés avec des détails tels que:
 - l'image Docker à utiliser / les fichiers Docker personnalisés pour créer l'image,
 - les PORTS à exposer,
 - les volumes à monter et les variables d'environnement.

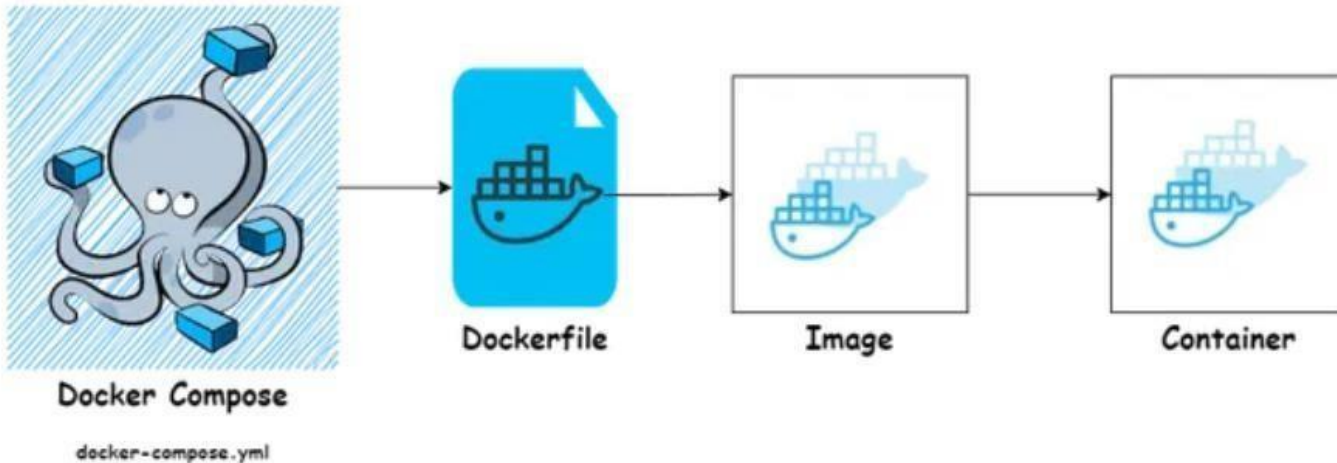


Docker compose

- Docker Compose : mise en route

Voici à quoi ressemble un fichier docker-compose.yml.
Il suit la syntaxe Yet_Another_Markup_Language(YAML),
qui contient des paires clé-valeur.

```
docker-compose up
```

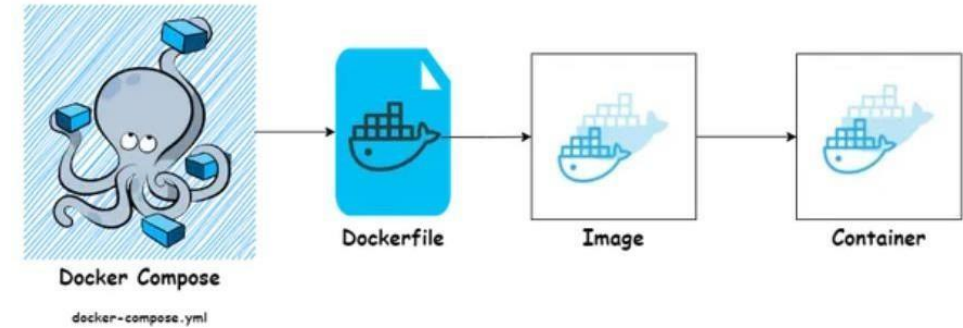


```
version: "3.9"
services:
  web-app:
    build: .
    container_name: fast-api-app
    ports:
      - 8000:8000
```

1. recherchera le fichier docker-compose.yml et créera les images qui sont écrites dans la section services du fichier.
2. En même temps, elle commence même à exécuter l'image, créant ainsi un conteneur à partir de celle-ci.
3. si nous allons sur localhost:8000, nous pouvons voir notre application en cours d'exécution.

Docker compose

- Docker Compose : mise en route
- une seule ligne de code, permet à la fois de créer l'image et d'exécuter le conteneur.
- Nous remarquons même que nous n'avons pas écrit le numéro de PORT dans la commande et que le site Web fonctionne toujours. C'est parce que nous avons écrit le PORT du conteneur dans le fichier docker-compose.yml.
- ne crée l'image que si l'image n'existe pas, si l'image existe déjà, elle exécute simplement l'image.



```
$ docker-compose ps
```

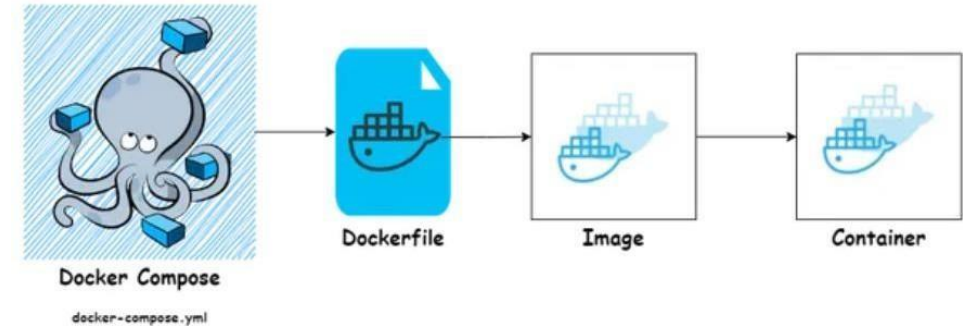
NAME	IMAGE	COMMAND
fast-api-app	compose_basics-web-app	"uvicorn app:app --h..."

SERVICE	CREATED	STATUS	PORTS
web-app	10 minutes ago	Up 10 minutes	0.0.0.0:8000->8000/tcp

Docker compose

- Docker Compose : mise en route

- **Nom** : c'est le nom du conteneur que nous avons écrit dans le fichier docker-compose.yml.
- **Image** : l'image générée à partir du fichier docker-compose.yml. compose_basics est le dossier où résident app.py, Dockerfile et docker-compose.yml.
- **Commande** : la commande s'exécute lorsque le conteneur est créé. Elle a été écrite dans l'ENTRYPOINT du Dockerfile.
- **Service** : il s'agit du nom du service créé à partir du fichier docker-compose.yml. Nous n'avons qu'un seul service dans le fichier docker-compose.yml et le nom que nous avons donné est web-app.
- **Statut** : indique la durée d'exécution du conteneur. Si le conteneur est quitté, le statut sera alors Quitté.
- **PORTS** : Spécifié les PORTS que nous avons exposés au réseau hôte.



```
$ docker-compose ps
```

NAME	IMAGE	COMMAND
fast-api-app	compose_basics-web-app	"uvicorn app:app --h..."

SERVICE	CREATED	STATUS	PORTS
web-app	10 minutes ago	Up 10 minutes	0.0.0.0:8000->8000/tcp

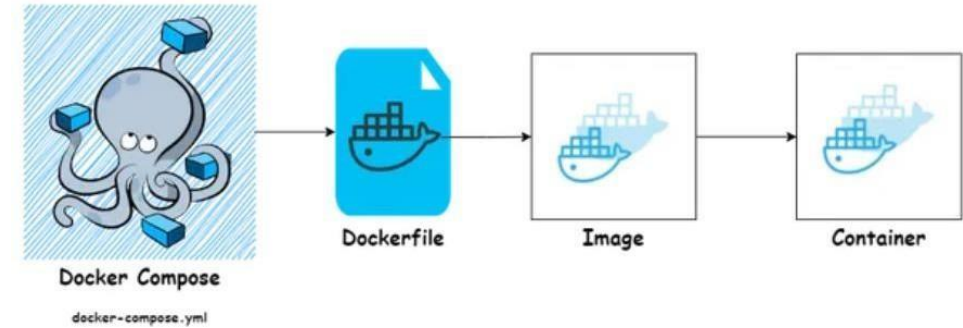
Docker compose

Docker Compose : Services : de quoi s'agit-il ?

Services : chaque docker-compose.yml commence par la liste des services/conteneurs que nous souhaitons créer et utiliser dans notre application.

- Tous les conteneurs avec lesquels vous travaillerez en rapport avec une application spécifique seront configurés sous les services.
- Chaque service a sa propre configuration qui comprend:
 - le chemin d'accès au fichier de build,
 - les PORTS,
 - les variables d'environnement, etc.

Pour l'application fast-api, nous n'avons besoin que d'un seul conteneur, c'est pourquoi nous avons défini un seul service nommé web-app sous les services.



```
version: "3.9"
services:
  web-app:
    build: .
    container_name: fast-api-app
    ports:
      - 8000:8000
```

Docker compose

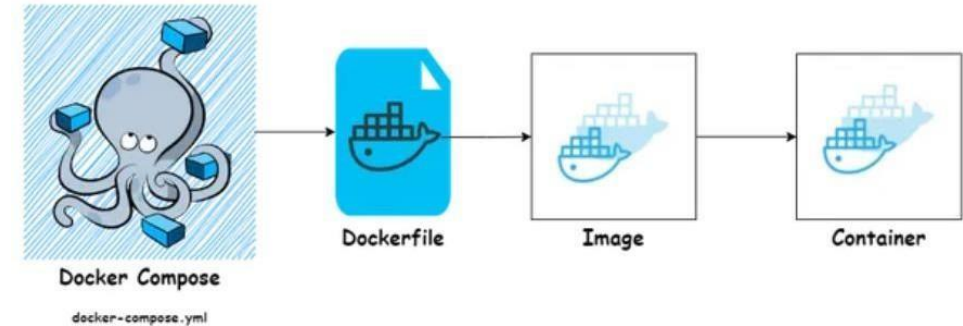
Docker Compose : Services : de quoi s'agit-il ?

Build : le mot-clé build indique l'emplacement du Dockerfile.

- Il fournit le chemin d'accès au Dockerfile.
- Si le Dockerfile et le fichier docker-compose.yml existent dans le même répertoire, alors dot(.) peut être utilisé pour représenter son chemin.

Image : Si nous ne voulons pas créer une image mais plutôt extraire une image du référentiel DockerHub, nous utilisons la commande image.

- Un exemple pourrait être lorsque nous souhaitons intégrer notre application à une base de données. Au lieu de télécharger la base de données dans l'image existante, nous allons extraire une image de base de données, puis lui permettre de se connecter à l'application principale.



```
version: "3.9"
services:
  web-app:
    build: .
    container_name: fast-api-app
    ports:
      - 8000:8000
```

Docker compose

- Docker Compose : Services : de quoi s'agit-il ?
- **container-name** : permet de fournir le nom de notre conteneur lors de sa création.
- **ports** : ce mot-clé sert à fournir les PORTS que notre conteneur exposera afin que nous puissions les visualiser depuis la machine hôte. En ce qui concerne le service d'application Web, nous avons

- **depends_on** : C'est le mot clé écrit dans un service particulier. Ici, nous donnerons le nom de l'autre service dont dépendra le service actuel.

```
version: "3.9"
services:
  web-app:
    build: .
    container_name: fast-api-app
    ports:
      - 8000:8000

  database:
    image: redis:latest
    ports:
      - 6379:6379
```

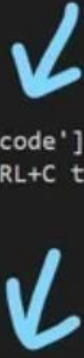
depends_on:

Docker compose

Docker Compose : Partage de fichiers entre le conteneur et l'hôte/conteneurs

- Les volumes Docker sont très utiles pour persister les données dans des conteneurs, en particulier pour des projets impliquant de grandes quantités de données comme dans le Deep Learning.
 - pour persister les fichiers générés par les applications
 - pour faciliter le partage de données entre plusieurs services dans un même environnement Docker Compose.

```
[+] Running 2/1
- Network compose_basics_default Created
- Container fast-api-app Created
Attaching to fast-api-app
fast-api-app | INFO: Will watch for changes in these directories: ['/code']
fast-api-app | INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
fast-api-app | INFO: Started reloader process [1] using StatReload
fast-api-app | INFO: Started server process [8]
fast-api-app | INFO: Waiting for application startup.
fast-api-app | INFO: Application startup complete.
fast-api-app | INFO: 172.25.0.1:55492 - "GET / HTTP/1.1" 200 OK
fast-api-app | WARNING: StatReload detected changes in 'app.py'. Reloading...
```



```
version: "3.9"
services:
  web-app:
    build: .
    container_name: fast-api-app
    ports:
      - 8000:8000
    volumes:
      - ./app.py:/code/app.py
```

```
docker-compose up --force-recreate
```

Docker compose

Docker Compose : Mise en réseau : communication entre services

```
networks:  
  my-network:
```

- Les réseaux dans Docker Compose permettent de définir comment les services (conteneurs) communiquent entre eux dans un environnement Docker.
- Par défaut, Docker Compose crée un réseau par défaut pour tous les services définis, mais vous pouvez également personnaliser vos réseaux pour gérer plus précisément la communication et l'isolation entre les conteneurs.
- Types de réseaux Docker Compose
 - bridge : Le réseau par défaut pour les conteneurs Docker non spécifiés. Il est utilisé lorsque vous ne spécifiez pas de réseau dans votre fichier Compose. Ce type de réseau est destiné à être utilisé pour des applications qui doivent communiquer entre elles au sein d'une même machine.
 - host : Utilise le réseau de l'hôte. Cela permet au conteneur d'utiliser le réseau de la machine hôte sans isolation. Cela peut être utile pour des applications nécessitant des performances réseau maximales, mais il est moins sécurisé car le conteneur a accès à l'ensemble des interfaces réseau de l'hôte.
 - overlay : Utilisé pour les réseaux de conteneurs Docker qui se trouvent sur plusieurs hôtes. Cela permet à des services Docker Compose de se connecter entre des hôtes différents. Ce type de réseau est souvent utilisé avec Docker Swarm ou Kubernetes.
 - none : Ce type de réseau est utilisé lorsque vous souhaitez désactiver complètement le réseau pour un conteneur. Cela peut être utile dans des cas très spécifiques.

Docker compose

Docker Compose : exemple

version: '3.8'

services:

tensorflow:

image: tensorflow/tensorflow:2.11.0-gpu # Version de TensorFlow avec support GPU

runtime: nvidia # Utiliser le runtime NVIDIA pour les GPU

environment:

- NVIDIA_VISIBLE_DEVICES=all # Utiliser tous les GPU disponibles

volumes:

- ./data:/workspace/data # Montez un volume local pour les données (par exemple, des datasets)

- ./notebooks:/workspace/notebooks # Montez un dossier pour les notebooks

ports:

- "8888:8888" # Expose le port Jupyter Notebook

command: "bash -c 'pip install --upgrade pip && jupyter notebook --ip 0.0.0.0 --no-browser --allow-root'"

Prérequis:

Docker et Docker Compose installés.

Un GPU NVIDIA (optionnel, mais nécessaire si vous souhaitez accélérer les calculs avec CUDA).

Le paquet nvidia-docker installé si vous utilisez des GPU.

Docker compose

Docker Compose : exemple MLP pour classification des patients sur ecg dataset

- Fichier Docker Compose (docker-compose.yml)
 - Conteneurise l'entraînement du modèle MLP.
 - Expose le modèle pour une application Flask.
 - Utilise deux services :
 - Un service pour l'entraînement du modèle MLP.
 - Un service pour l'application Flask.

Docker compose

Docker Compose : exemple MLP pour classification des patients sur ecg dataset

train/Dockerfile

FROM python:3.9-slim

WORKDIR /train

COPY requirements.txt /train/

RUN pip install --no-cache-dir -r requirements.txt

COPY . /train/

CMD ["python", "train.py"]

app/Dockerfile

FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt /app/

RUN pip install --no-cache-dir -r requirements.txt

COPY . /app/

CMD ["python", "app.py"]

Docker compose

Docker Compose : exemple MLP pour classification des patients sur ecg dataset

version: '3.8'

services:

train:

build:

context: ./train

volumes:

- ./train/model.pkl:/train/model.pkl # Monte le modèle généré dans le répertoire 'train'

command: python train.py # Lance l'entraînement du modèle

networks:

- ml_network

Docker compose

Docker Compose : exemple MLP pour classification des patients sur ecg dataset

app:

build:

context: ./app

volumes:

- ./train/model.pkl:/app/model.pkl # Monte le modèle Pickle dans le conteneur Flask

ports:

- "5000:5000" # Expose l'application Flask sur le port 5000

networks:

- ml_network

depends_on:

- train # Attendre que le service 'train' soit terminé

networks:

ml_network:

driver: bridge

Docker compose

Docker Compose : exemple MLP pour classification des patients sur ecg dataset

Démarrer l'environnement avec Docker Compose

```
docker-compose up --build
```

Accéder à l'application Flask : Une fois que les services sont démarrés, nous pouvons accéder à l'API Flask en envoyant une requête HTTP POST sur le point de terminaison /predict. Exemple d'appel cURL pour tester l'API Flask.

```
curl -X POST -H "Content-Type: application/json" -d '{"features": [5.1, 3.5, 1.4, 0.2]}'  
http://localhost:5000/predict
```