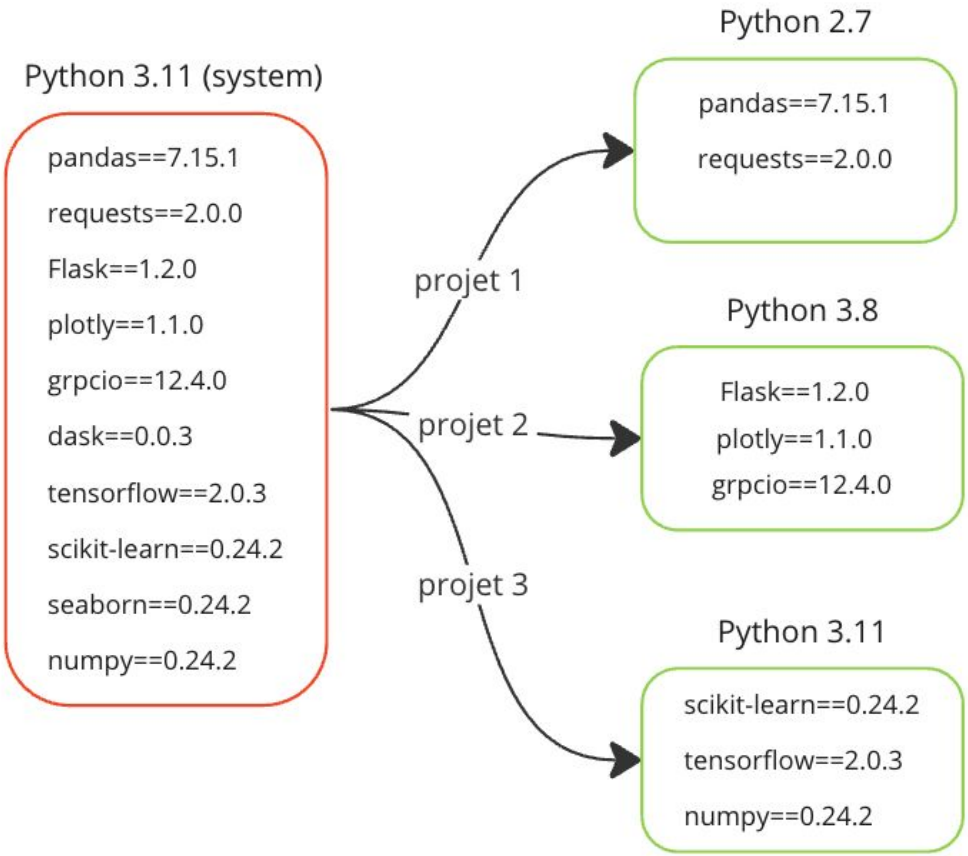
Pourquoi utiliser un virtual env pour ses projets Python ?



Commandes utiles

Avec python

```
# créer l'env
python -m venv my_venv

#activer l'env
source my_venv/bin/activate
```

Avec anaconda

```
# créer l'env
conda create --name my_env python=3.8

#activer l'env
conda activate my_env
```

Avec pipfile

```
# créer l'env
pipenv --python 3.8

#activer l'env
pipenv shell
```

Le nom de l'env entre parenthèses dans le terminal indique quel env est actif

Désactiver un env

```
deactivate
```

Docker

Docker c’est une plateforme de conteneurisation qui permet de créer, de déployer et de gérer des applications de manière plus efficace et portable. Cela, indépendamment de l’environnement sur lequel l’application tourne.

1. **Container** : Docker utilise des containers pour encapsuler des applications et leurs dépendances. Un container est une unité légère, isolée et autonome qui contient tout ce dont une application a besoin pour fonctionner, y compris le code, les bibliothèques et les configurations.

2. **Isolation** : Les containers offrent une isolation efficace, ce qui signifie que chaque container fonctionne indépendamment des autres, même s'ils partagent le même système d'exploitation hôte. Cela permet d'éviter les conflits entre les applications et les dépendances.

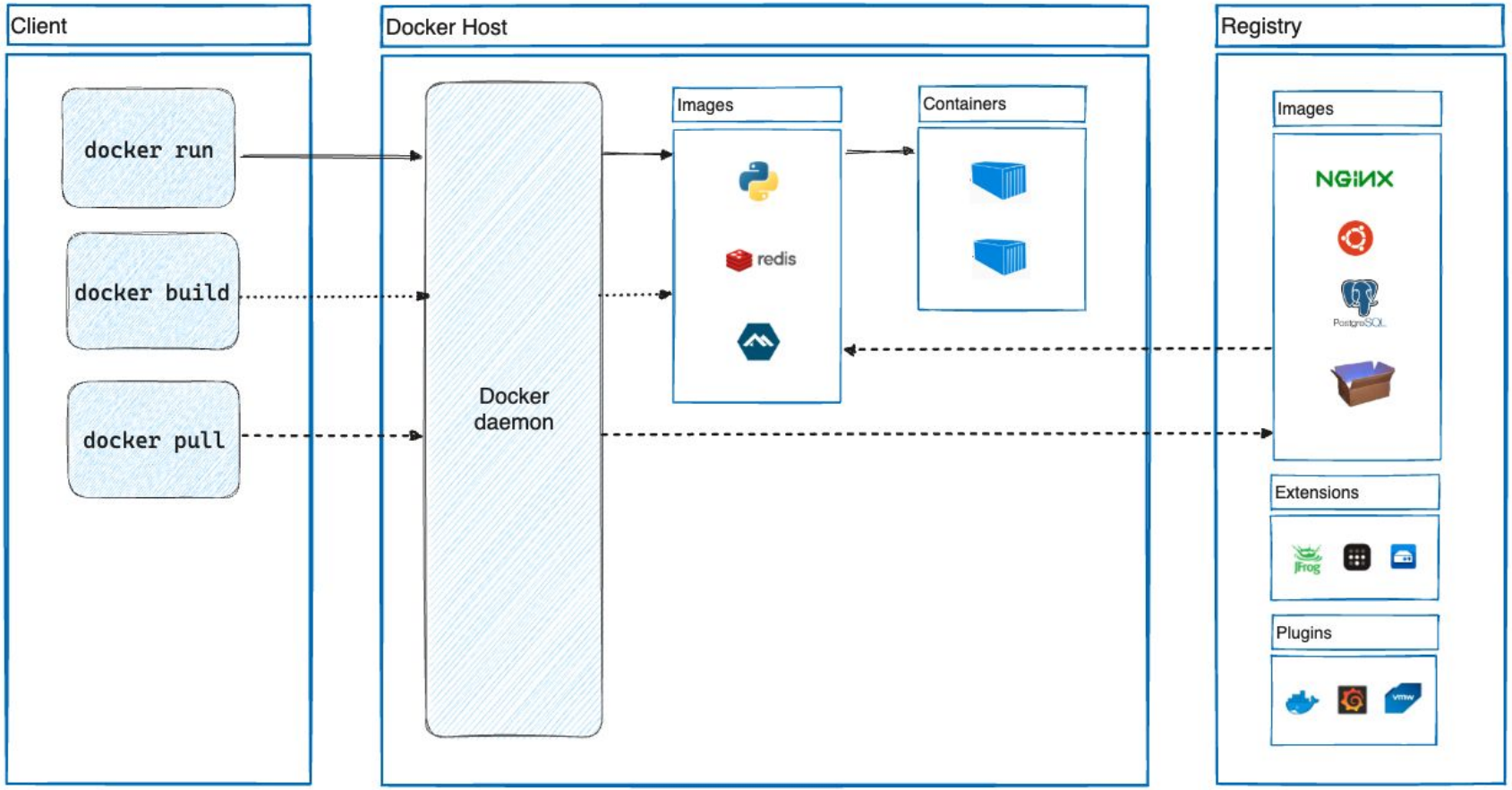
3. **Portabilité** : Les containers Docker sont portables. Il est possible de créer un container sur un système et le déployer sur un autre sans avoir à se soucier des différences d'environnement, tant que Docker est installé sur ces systèmes.

Docker est aujourd’hui utilisé absolument partout dans les métiers de la tech et il est primordial d’en connaître les usages afin de pouvoir naviguer dans des infrastructure data

Docker vs VM

	Conteneur Docker	VM
De quoi s'agit-il ?	Docker est une plateforme logicielle permettant de créer et d'exécuter des conteneurs Docker. Un conteneur Docker est une émulation d'une instance de l'espace utilisateur, la partie du système d'exploitation où les processus utilisateur s'exécutent.	Il s'agit d'une émulation d'une machine physique, y compris du matériel virtualisé, qui exécute un système d'exploitation.
Virtualisation	Le conteneur masque les détails du système d'exploitation au code de l'application.	La machine virtuelle masque les détails du matériel au code de l'application.
Objectif	Masquer les détails du matériel et augmenter l'utilisation du matériel.	Améliorer la gestion de l'environnement de l'application et assurer la cohérence entre plusieurs environnements.
Géré par	Le moteur Docker assure la coordination entre le système d'exploitation et les conteneurs Docker.	L'hyperviseur assure la coordination entre le matériel physique de l'ordinateur et les machines virtuelles.
Architecture	Partage les ressources avec le noyau hôte sous-jacent.	Exécute son propre noyau et son propre système d'exploitation.
Partage de ressources	À la demande.	Une quantité fixe, définie dans les exigences de configuration d'une image de machine virtuelle.

L'architecture Docker



Les images

Une image Docker est un paquet léger et autonome qui inclut tout le nécessaire pour exécuter une application, y compris le code, les bibliothèques, les dépendances et les paramètres d'environnement.

Cette image est un empilement de couches, toutes basées sur la couche du dessous. Exemple:

Projet python **>** *Image python:3.11* **>** *Image Debian:12*

```
→  app docker image ls
REPOSITORY           TAG       IMAGE ID       CREATED             SIZE
app                  latest    a4d92be7121f   About an hour ago   1.03GB
mongo                latest    1e918da5b1ff   2 hours ago         757MB
tp-api               latest    dd15c7136246   4 days ago          1.08GB
cours-docker-image   latest    93284cab5ca7   6 days ago          131MB
tp-front             latest    9950374aea80   6 days ago          133MB
postgres             latest    fbd1be2cbb1f   2 months ago        417MB
```

- docker image ls (⇔ docker images) : liste les images présentes sur la machine hôte
- docker image rm **image_id** (⇔ docker rmi) : supprime une image sur la machine hôte
- docker pull **image** : récupère une image du repository (docker hub repo publique d'images)
- docker build **dockerfile :** construit une image docker en se basant sur un Dockerfile

Le Dockerfile

app > Dockerfile > ...

```
1   FROM python:3.11
2
3   WORKDIR /app
4
5   COPY . /app
6
7   RUN pip install -r requirements.txt
8
9   EXPOSE 5000
10
11  ENV MONGO_PORT="27017"
12  ENV MONGO_USER="root"
13  ENV MONGO_PSWD="changeme"
14
15  CMD ["python", "app.py"]
```

Image sur laquelle on se base (**plateforme : tag**)

Dossier de travail (création si non existant)

Copie de fichier dans l'image (ici **code** + **packages**)

Exécution d'une commande (installation des packages depuis le fichier copié)

Exposition d'un port (nécessaire pour la communication du script avec l'extérieur)

Variable d'environnement

Lancement du programme principal du container (ici un serveur Flask). **ENTRYPOINT** pour une entrée par défaut.

On crée un Dockerfile par container.
Un Dockerfile est un fichier .yaml qui contient toutes les instructions permettant de construire une image Docker. Il spécifie les étapes nécessaires pour créer cette image, y compris la base à partir de laquelle construire, les fichiers à inclure, les commandes à exécuter, et plus encore.

Les containers

Un conteneur Docker est une instance en cours d'exécution d'une image. Les conteneurs sont isolés les uns des autres et du système hôte, mais ils partagent le même noyau du système d'exploitation. Ils peuvent être créés, démarrés, arrêtés et supprimés de manière simple.

```
→  cours docker container ls
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS                                      NAMES
7d660006dbba   app       "python app.py"          18 hours ago   Up 18 hours   0.0.0.0:5000->5000/tcp                     vibrant_noyce
dc137a922f43   mongo     "docker-entrypoint.s…"   18 hours ago   Up 18 hours   0.0.0.0:27017->27017/tcp, 28017/tcp        mongodb
```

- docker run **image :** instancie et démarre un container basé sur l'image
- docker container ls (⇔ docker ps) : liste les containers en cours d'exécution
- docker container start / stop **container** : éxecute ou stop un container existant

Le docker-compose

docker-compose.yml

```yaml
version: '3'

services:
  api:
    build:
      context: ./app
    container_name: app
    ports:
      - "5000:5000"
    depends_on:
      - mongodb
    environment:
      MONGO_URI: "mongodb://mongodb:27017/"
      MONGO_USER: root
      MONGO_PSWD: changeme

  mongodb:
    image: mongo:latest
    container_name: mongodb
    restart: always
    ports:
      - "27017:27017"
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: changeme
    volumes:
      - mongodb_data:/data/db

volumes:
  mongodb_data:
```

Construction de l'image en se basant sur le Dockerfile situé dans le dossier app

Mapping des ports (**machine hôte : container**)

Variables d'environnement

Sélection d'une image déjà construite mongo:latest

Mapping d'un volume à dossier du container

Construction d'un volume pour la persistance des données

- docker-compose up / down : lance ou stop les containers