Docker

Docker

Installation: https://docs.docker.com/desktop/

windows: https://docs.docker.com/desktop/setup/install/windows-install/
linux: https://docs.docker.com/desktop/setup/install/linux/
mac: https://docs.docker.com/desktop/setup/install/mac-install/

Préparation
Faire un git clone du repos: https://github.com/idrissa-mgs/intro-devops/tree/main

Pour les exo/dev: se mettre sur votre branche (git checkout -b <ma-branche>

Docker

Imaginez ceci :

Vous avez développé une superbe application sur votre machine locale. Tout fonctionne parfaitement : les services s'exécutent, la base de données répond, et vous êtes satisfait du résultat.

- Un(e) collègue souhaite travailler sur le projet, il est sur un poste potentiellement différent du vôtre
- Vous souhaitez déployé cet application sur le cloud ou sur vos serveurs distants

Comment faire tout ceci sans soucis ?

Définition

Docker est une plateforme open-source qui permet de packager et déployer des applications facilement et rapidement.

Docker enveloppe et lance les application dans des environnement appelées conteneurs, qui rassemblent tous les éléments nécessaires à leur fonctionnement : bibliothèques, outils système, code et environnement d’exécution.

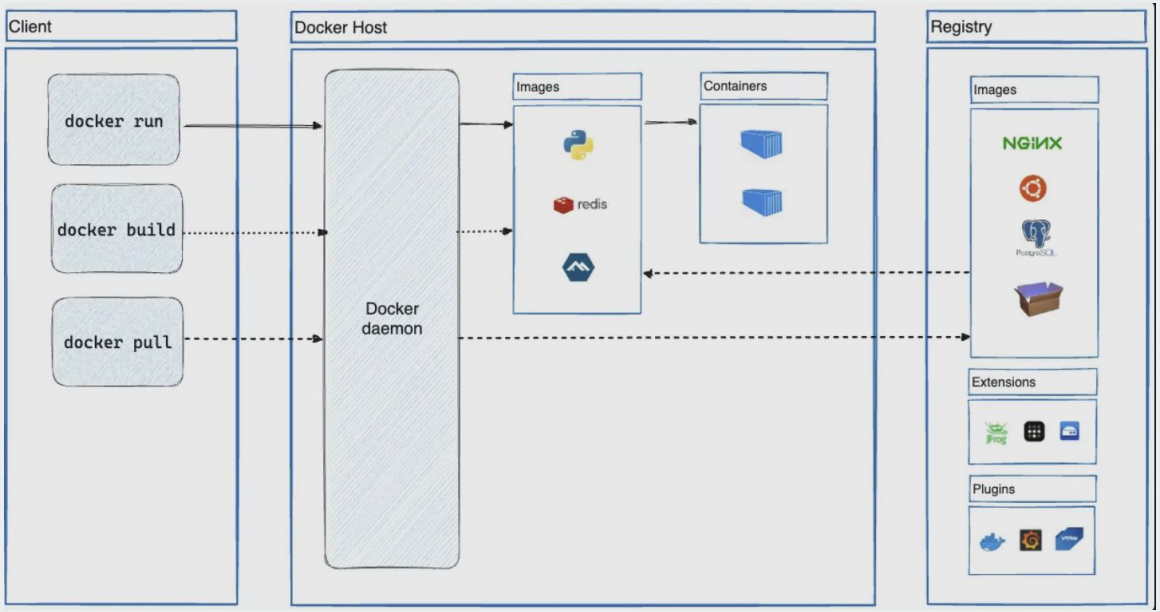
Docker

Avantages

- **Portabilité:** Via ses containers docker permet de packager les applications avec toutes leurs dépendances assurant ainsi un environnement reproductible qu'on peut déployer presque partout.
- **Isolation:** Chaque application/service peut facilement être isolée avec toutes ses dépendances
- **Efficacité & Flexibilité**: Docker offre plus de flexibilité qu'une machine virtuelle avec moins de temps de requis pour le démarrage et une gestion plus facile des mises à jour.

Docker

Architecture



Docker

Terminologie

- **Image:** Une image Docker est un modèle immuable qui contient tout ce dont une application a besoin pour fonctionner : le code, les dépendances, et les configurations.
- **Conteneur:** Un conteneur est une instance d'une image en cours d'exécution. C'est l'environnement isolé où tourne votre application.
- **Dockerfile**: Un Dockerfile est un fichier texte qui contient des instructions pour construire une image.
- **Registry:** Un dépôt où sont stockées les images Docker. (https://hub.docker.com/)
- **docker-compose**: Composant de docker qui permet de gérer et d'orchestrer plusieurs conteneurs Docker en même temps.

Docker

Docker en action

- Créer une simple application web basé sur fast api.
- Créer un Dockerfile pour packager l’application
- Construire l’image docker de l’app (docker run)
- Créer et lancer le conteneur

Docker

Docker en action

- Fastapi: application en local

```python
from fastapi import FastAPI

app = FastAPI()


@app.get("/")
def read_root():
    return "Hello, World! We are learning Docker"
```

Pour lancer l'application:

uvicorn api:app --reload --host 0.0.0.0 --port 8000

Docker

Docker en action

Et si on avait une application plus complexe avec plusieurs intervenants/contributeurs et qu'il fallait déployer l'application dans un autre environnement (cloud,etc).

Comment on va utiliser docker pour nous permettre de packager notre application

Docker

Docker en action

Etape 1: créer le Dockerfile qui va représenter l'image de notre application

Structure d'un Dockerfile:

```
# Étape 1 : Choisir une image de base
FROM <base-image>:<tag>

# Étape 2 : Ajouter des métadonnées (optionnel)
LABEL maintainer="your-email@example.com"
LABEL version="1.0"
LABEL description="Description of the application."

# Étape 3 : Définir le répertoire de travail
WORKDIR /app

# Étape 4 : Copier les fichiers nécessaires dans l'image
COPY . /app

# Étape 5 : Installer les dépendances
RUN <install-commands>

# Étape 6 : Définir des variables d'environnement (optionnel)
ENV ENV_VARIABLE=value

# Étape 7 : Exposer un ou plusieurs ports
EXPOSE <port>

# Étape 8 : Nettoyage (optionnel, pour réduire la taille de l'image)
RUN rm -rf /var/lib/apt/lists/*

# Étape 9 : Définir la commande d'exécution
CMD ["<command>", "arg1", "arg2"]
```

Docker

Docker en action

Etape 1: créer le Dockerfile qui va représenter l'image de notre application

```dockerfile
# Image de base
FROM python:3.10-slim

# Étape 2 : Définir le répertoire de travail dans le conteneur
WORKDIR /app

# Étape 3 : Copier les fichiers nécessaires
COPY requirements.txt requirements.txt
COPY main.py main.py

# Étape 4 : Installer les dépendances
RUN pip install --no-cache-dir -r requirements.txt

# Étape 5 : Exposer le port pour FastAPI
EXPOSE 8000

# Étape 6 : Lancer le serveur Uvicorn (commande dépendante de l'application)
CMD ["uvicorn", "api:app", "--host", "0.0.0.0", "--port", "8000"]
```

Docker

Docker en action

Etape 2: Construire l'image à l'aide de la ligne de command docker

```
docker build -t <nom_de_l'image>:<tag> <chemin vers le dockerfile>
```

Vérifier si l'image a été construite en listant les images existantes: ***docker image ls***

Docker

Docker en action

Etape 3: Lancer le conteneur à l'aide de la ligne de command docker

```
docker run <image_name>:<tag> \
--port <host-port>:<target-port> \
--name <container_name>  \
--volume <host_path>:<container_path> \
```

Options:

- port: permet de mapper un port sur ta machine sur un port exposé de ton application
- name: nom qu'on veut donner au conteneur
- volume): si on veut synchroniser des fichiers locaux avec d'autres dans le conteneur (copie synchrone)
- d: si on souhaite lancer le conteneur en arrière plan
- rm: pour supprimer automatiquement le conteneur après son arrêt

docker run fastapi:1.0 --port 8000:8000 --name fastapi-app

Docker

Docker en action

Etape 4: Aller sur le port ouvert en question:

ici http://0.0.0.0:8000/ ou http://localhost:8000/

Docker

Docker en action

Exercice:
Reprendre l'exercice précédent en introduisant une variable d'environnement qu'on va afficher à la page d'accueil

"Hello World! We are learning Docker" => "Hello <ENV_VAR>! We are learning Docker"

Docker

La notion de Volume

Que se passe t-il si jamais on fait des modifications au niveau de notre application et qu'on souhaite redéployer cela. On sera obligé d'arrêter le conteneur et le relancer.
Et si on utiliser les volumes pour faire des copies synchrones.

Pour relancer le conteneur on va devoir ceci.

docker stop fastapi-app // pour arreter le conteneur
docker rm fastapi-app // Pour supprimer le conteneur (pas nécessaire)
docker build -t fastapi:1.1 . // creer à nouveau l'image
docker run fastapi:1.0 --port 8000:8000 --name fastapi-app

Docker

La notion de Volume

Les **volumes** sont des ressources que Docker utilise pour sauvegarder durablement les données d'un conteneur en faisant appel à un stockage externe au conteneur.

Les volumes permettent:

- **Conserver les données** même si le conteneur est supprimé ou recréé.
- **Partager des données** entre plusieurs conteneurs.
- **Gérer facilement les fichiers** en dehors du système de fichiers du conteneur.

Docker

La notion de Volume: Utilisation

docker run fastapi:1.0 --port 8000:8000 --name fastapi-app --volume $(pwd):/app (ou “$(pwd)”:/app sur windows)

On fait une copie synchrone de notre répertoire de travail dans le conteneur. Tous les fichiers de notre répertoir de travail seront également visible dans le dossier /app. Ainsi les changements dans l’application seront directement pris en compte

Docker

Docker commands cheat sheet:
link: https://docs.docker.com/get-started/docker_cheatsheet.pdf

Images

docker build -t <image-name>:<tag> #construire l'image à partir d'un docker file
docker image ls #lister les images
docker rmi *<image-name>:<tag>* #supprimer une image

Conteneurs

docker run --name <cont-name> <image-name>:<image-tag> -d (background)
docker exec -it <cont-name> bash #lancer une commande bash à l'intérieur du conteneur)
docker container ls (ou docker ps) #lister les conteneurs

Docker compose

docker-compose up -d (background) #lancer les conteneurs définis dans le docker-compose.yaml, créer si besoin les images
docker-compose down # arrêter et supprime les conteneurs lancer par d-c up
docker-compose start/stop #démarre/arrete les conteneurs, ne crée pas d'image/supprime pasles conteneurs
docker-compose ps # lance les conteneurs en cours d'éxécution

Docker

Docker-compose:

Docker Compose est un outil qui permet de lancer et gérer plusieurs conteneurs Docker en même temps. L'outil idéal si on souhaite gérer plusieurs composants d'une même application (notion de microservices).

Pour l'utiliser, on crée un docker-compose.yaml

Docker

Docker-compose:

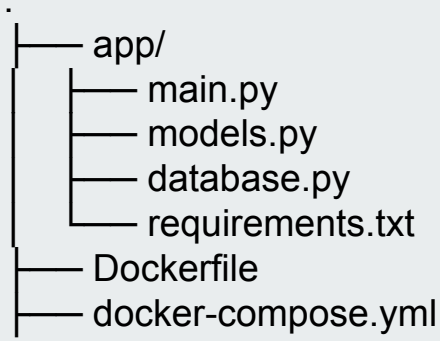
Structure générale d'un fichier docker-compose.yaml

```yaml
version: "3.9"  # Version de Docker Compose

services:       # Définir les services (conteneurs)
  service_1_name:  # Nom du service (ex : app, db)
    build:               # Ou, construire à partir d'un Dockerfile
      context: .         # Chemin vers le Dockerfile
    ports:               # Mappage des ports
      - "host_port:container_1_port"
    environment:         # Variables d'environnement
      - ENV_VAR_NAME=value
    volumes:          # Volumes pour stocker ou partager des données comme celui vu avec le dockerfile
      - type: bind
        source: host_path         # Chemin du fichier/dossier sur la machine hote (locale)
        target: conteneur_path     # Chemin du fichier/dossier dans le conteneur
    depends_on:          # Dépendances entre services
      - service_2_name

  service_2_name: # Nom du service (ex : app, db)
    image: image_2_name:tag  # Image Docker à utiliser
    ports: # Mappage des ports
      - "host_port:container_2_port"
    volumes: # Volumes pour stocker ou partager des données
      - volume_name:container_path
    environment: # Variables d'environnement
      - ENV_VAR_NAME=value
volumes:               # (Optionnel) Définir des volumes nommés
  volume_name:
```

Docker

Docker-compose:

Cas pratique: On va associer une base de données postgre à notre api



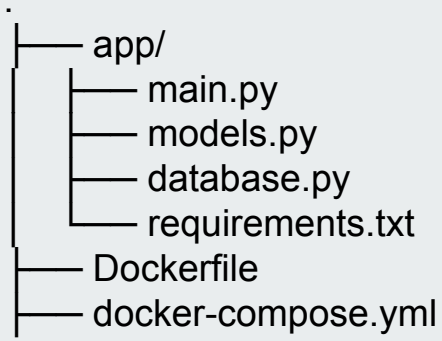
Etapes:

1. créer un dossier app qui va contenir l'api et ses dépendances
2. créer le dockerfile pour l'api
3. créer le docker-compose.yaml qui nous permettra de construire les conteneurs pour l'api et la db
4. Lancer le tout avec la commande (*docker-compose up*)

Docker

Docker-compose:

Cas pratique: On va associer une base de données postgre à notre api



Exercice:
L'étape 1 étant faite continuer pour faire étapes suivantes (*se positionner dans le dossier dockercompose*)

L'api est associé à une base de données dans laquelle on a une table d'utilisateurs.

Pour tester si tout fonctionne bien, on peut, une fois les conteneurs lancés aller sur l'url http://localhost/8000
Pour voir la liste des users sur http://localhost/8000/users
Pour ajouter un nouvel utilisateur:
curl -X POST "http://localhost:8000/users/" -H "Content-Type: application/json" -d '{"name": "UserName", "age": UserAge}'

Docker

DockerHub: https://hub.docker.com/

Le registry par défaut de docker on peut y stocker nos images.
C'est ici que docker vient chercher les images si ces dernières n'existe pas en local

Login into Docker

```
docker login -u <username>
```

Publish an image to Docker Hub

```
docker push <username>/<image_name>
```

Search Hub for an image

```
docker search <image_name>
```

Pull an image from a Docker Hub

```
docker pull <image name>
```

Docker

Docker Desktop: https://docs.docker.com/desktop/

Offre un interface graphique pour voir et gérer vos ressources (images, conteneurs, etc)