TP_docker_basic_1

Docker - Initiation : premiers conteneurs

Objectif général

Découvrir Docker en manipulant des images existantes, en conteneurisant un script Python très simple et en comprenant le cycle de vie d'un conteneur (build, run, inspect, logs, nettoyage).

Prérequis

- Docker Desktop, Docker Engine ou Colima/Rancher Desktop opérationnel
- Accès à un terminal (bash/zsh) avec `curl`, `python3`
- Bases de CLI Linux (navigation, exécution de commandes)

Compétences visées

- Vérifier l'installation et comprendre les informations principales du daemon Docker
- Télécharger et exécuter des images officielles
- Construire une image à partir d'un Dockerfile minimal
- Gérer les conteneurs (logs, stop, rm) et nettoyer l'environnement

---

Fil rouge du TP

Tu vas transformer un script Python qui génère un message de bienvenue en image Docker indépendante. À chaque étape tu apprendras une commande clé pour passer de « je lance un script sur ma machine » à « je lance un conteneur portable ».

Arborescence recommandée : `~/workspace/docker-basic-1`.

Étape 0 — Vérifier son moteur Docker

1. Ouvre un terminal et affiche la version :

```
docker version
```

➜ Repère la différence entre la partie *Client* et *Server*. Note les versions pour ton compte-rendu.

2. Inspecte la configuration :

```
docker info
```

➜ Identifie le driver de stockage, le runtime par défaut, le nombre d'images et de conteneurs déjà présents.

Étape 1 — Exécuter des images existantes

1. Teste la configuration avec `hello-world` :

```
docker run hello-world
```

➜ Observe le téléchargement (pull) et la sortie du programme.

2. Lance un shell éphémère en Alpine :

```
docker run -it --rm alpine:3.20 sh
```

➜ Dans le conteneur, liste `/`, crée un fichier, puis quitte (`Ctrl+D`). Confirme que le fichier n'existe pas en relançant le conteneur.

3. Liste les images et les conteneurs :

```
docker image ls
docker ps -a
```

➜ Nettoie les conteneurs créés par `hello-world` si nécessaire (`docker rm <id>`).

Étape 2 — Préparer le projet Python

1. Crée le dossier de travail et les fichiers :

```
mkdir -p ~/workspace/docker-basic-1 && cd ~/workspace/docker-basic-1
git init .
cat > welcome.py <<'PY'
import datetime
import platform


def banner() -> str:
    return (
        "Bonjour Docker !\n"
        f"Date et heure : {datetime.datetime.utcnow().isoformat()}Z\n"
        f"Système : {platform.system()} {platform.release()}"
    )


if __name__ == "__main__":
    print(banner())
PY
```

2. Vérifie l'exécution locale :

```bash
python3 welcome.py
```

➜ Garde une trace de la sortie pour comparer avec la version conteneurisée.

Étape 3 — Écrire un Dockerfile minimal

1. Crée `Dockerfile` :

```
# syntax=docker/dockerfile:1
FROM python:3.12-slim

WORKDIR /app
COPY welcome.py .

CMD ["python", "welcome.py"]
```

2. Construis l'image :

```
docker build -t welcome:1.0.0 .
```

3. Exécute et compare :

```
docker run --rm welcome:1.0.0
```

➜ Note la différence de date/heure et de plateforme. Explique pourquoi dans le rapport (indice : horodatage et noyau du host).

Étape 4 — Personnaliser l'exécution

Objectif : comprendre `CMD` vs arguments.

1. Passe un message personnalisé en variable d'environnement :

```
docker run --rm -e WELCOME_NAME="Alice" welcome:1.0.0
```

➜ Adapte `welcome.py` pour lire `WELCOME_NAME` (`os.getenv("WELCOME_NAME", "Docker")`) et reconstruis l'image.

2. Surcharge la commande au run :

```
docker run --rm welcome:1.0.0 python -c "print('Commande override')"
```

➜ Explique la différence entre `CMD` dans le Dockerfile et les arguments passés au `docker run`.

Étape 5 — Inspecter, logger, nettoyer

1. Lance le conteneur en arrière-plan :

```
docker run -d --name welcome_app welcome:1.0.0 sleep 60
```

➜ Cette commande démarre le conteneur, exécute `sleep 60` puis s'arrête.

2. Observe les métadonnées :

```
docker inspect welcome_app | jq '.[0].Config.Env'
docker logs welcome_app
docker top welcome_app
```

3. Stoppe et supprime :

```
docker stop welcome_app
docker rm welcome_app
```

4. Nettoie les images inutilisées :

```
docker image prune
```

➜ Explique la différence entre supprimer un conteneur (`docker rm`) et supprimer une image (`docker image rm`).

Livrables attendus

- Repository git contenant `welcome.py` et `Dockerfile`
- Fichier `README.md` précisant les commandes clés (`build`, `run`, `logs`, nettoyage)
- Quelques lignes expliquant :
  - ce qui change entre l'exécution locale et conteneurisée
  - les commandes Docker nouvelles pour toi
  - les difficultés rencontrées

Aller plus loin

- Ajouter une dépendance Python (`pip install requests`) et adapter l'image (impact sur la taille ?).
- Créer un tag `welcome:latest` et pousser l'image sur Docker Hub.
- Réduire la taille de l'image en utilisant `python:3.12-alpine` (attention aux bibliothèques manquantes).
- Utiliser `ENTRYPOINT` au lieu de `CMD` et tester différentes combinaisons.