Lab1:

Lancer un terminal interactif:

```
docker run -it ubuntu bash
```

Exercices dans le conteneur:

1. Mettre à jour les packages
   apt update
2. Installer des outils
   apt install -y curl wget vim
3. Créer des fichiers
   echo "Hello from container" > /tmp/test.txt
   cat /tmp/test.txt
4. Quitter le conteneur
   exit

Comprendre l'éphémérité:

1. Lancer un nouveau conteneur Ubuntu
   docker run -it ubuntu bash
2. À l'intérieur, vérifier:
   ls /tmp/

Le fichier test.txt n'existe pas !
C'est un NOUVEAU conteneur

- Concept clé : Éphémérité
- Chaque docker run crée un NOUVEAU conteneur
- Les modifications sont perdues à la sortie
- Les conteneurs sont immuables par design

Créer sans démarrer

```
docker create --name mon-conteneur nginx
```

Démarrer

```
docker start mon-conteneur
```

Voir tous les conteneurs

- Conteneurs en cours d'exécution
  ```
  docker ps
  ```
- Tous les conteneurs (incluant arrêtés)
  ```
  docker ps -a
  ```

Arrêter (SIGTERM, puis SIGKILL après 10s)

```
docker stop mon-conteneur
```

Redémarrer

```
docker restart mon-conteneur
```

Pause (freeze les processus)

docker pause mon-conteneur

Reprendre

docker unpause mon-conteneur

Arrêt immédiat (SIGKILL)

docker kill mon-conteneur

Supprimer (doit être arrêté)

docker rm mon-conteneur

Forcer la suppression (même en cours)

docker rm -f mon-conteneur

Questions de réflexion

1. Pourquoi les données persistent dans un conteneur arrêté mais pas supprimé?
2. Comment pourrait-on sauvegarder notre application ?
3. Quels sont les avantages de cette éphémérité ?

Réponses

1. Le conteneur arrêté conserve sa couche d'écriture
2. Avec des volumes Docker ou en créant une nouvelle image
3. Environnements propres, reproductibles, pas de "pollution"

Lab2:

Cet exercice consiste à utiliser une base de données PostgreSQL via un conteneur plutôt que de l’installer sur votre machine.

1) Se rendre sur le Docker Hub et rechercher une l'image officielle PostgreSQL.

2) Lancer un conteneur PostgreSQL nommé postgresql-db, qui publie le port 5432, et définit un mot de passe pour l’utilisateur postgres.

3) Créer une table et y insérer des données.

Lab3:Publication d’image

1. Créer une page d’accueil personnalisée

index.html :

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>My First Container</title>
</head>
<body>
    <h1>Mon Conteneur Nginx est Actif !</h1>
    <p>
        Félicitations ! Ce contenu a été servi par votre image Docker personnalisée
        qui utilise le serveur web Nginx.
    </p>
    <p>
        <small>Ce fichier remplace la page d'accueil par défaut (index.html).</small>
    </p>
</body>
</html>
```

2. Lancer un conteneur Nginx

```
docker run -d --name nginx-custom -p 8080:80 nginx
```

3. Copier la page HTML dans le conteneur

```
docker cp index.html nginx-custom:/usr/share/nginx/html/index.html
```

Cela remplace la page d’accueil par défaut à l’intérieur du conteneur.

Vérifier ici:

http://localhost:8080/

4. Créer une image à partir du conteneur modifié

```
docker commit nginx-custom monuserdocker/nginx-custom:v1
```

⚠ Remplace monuserdocker par ton nom Docker Hub.

- docker commit: Crée une nouvelle image à partir des modifications apportées au système de fichiers du conteneur.
- nginx-custom: C'est le **nom** (ou l'**ID)** du conteneur Docker en cours d'exécution (ou récemment arrêté) dont tu veux sauvegarder l'état.
- monuserdocker: Ton nom d'utilisateur Docker Hub (ou le dépôt cible).
- **nginx-custom:v1** → Le nom du dépôt de l'image et le tag de l'image

5. Se connecter à Docker Hub

docker login

6. Pousser l'image vers Docker Hub

docker push monuserdocker/nginx-custom:v1

Pourquoi ne pas utiliser docker commit ?

docker commit :

- ❌ ne garde *aucune trace* de ce que on a modifié
- ❌ impossible à versionner
- ❌ crée souvent des images plus lourdes
- ❌ complique le travail en équipe
- ❌ rend les builds automatiques impossibles (CI/CD)

Autre solution le Dockerfile:

1.Créer un Dockerfile

Dans le même dossier où se trouve index.html

```
FROM nginx:latest

COPY index.html /usr/share/nginx/html/index.html
```

2.Construire l'image

Dans ce même dossier (très important !) :

```
docker build -t nginx-custom .
```

Le . indique à Docker d'utiliser ce dossier comme contexte et donc de trouver le Dockerfile ici.

Le drapeau (-t pour *tag*) permet de donner un nom à l'image que vous créez. Ici, l'image sera nommée nginx-custom

3.Taguer l'image pour Docker Hub

```
docker tag nginx-custom monuserdocker/nginx-custom:latest
```

⚠ Remplace monuserdocker par ton nom Docker Hub.

4.Pousser l'image sur Docker Hub

```
docker push monuserdocker/nginx-custom:latest
```

Pourquoi un Dockerfile est beaucoup mieux ?

1. Reproductibilité

Un Dockerfile décrit exactement comment créer l'image :

```
COPY index.html /usr/share/nginx/html/index.html
```

N'importe qui peut reconstruire la même image avec :

```
docker build .
```

✔ Même résultat
✔ Sans manipulation manuelle
✔ Sans dépendre d'un conteneur existant

2. Historique clair (Git)

Tu peux versionner ton Dockerfile dans Git.
Tu sais quand, par qui, quoi a changé.

3. Automatisable (CI/CD)

GitHub Actions, GitLab CI, Jenkins… savent construire une image à partir d'un Dockerfile.

4. Images plus propres et plus légères

Le Dockerfile part d'une base claire (ex : `nginx:alpine`).
Pas d'artefacts inutiles dans l'image.

5. Maintenabilité

Dans 6 mois, tu peux revoir ton Dockerfile pour comprendre ce qu'il fait.
Avec `docker commit`, impossible de savoir ce qui existe dans l'image sans fouiller.

6. Collaboration facilitée

Toute l'équipe peut partager le même Dockerfile.

Conclusion

👉 **Le Dockerfile est la seule méthode fiable, propre et professionnelle pour créer une image.**
👉 `docker commit` dépanne, mais ne doit pas être utilisé en production.

Lab4:Dockerfile

1.Démarrer un Conteneur Ubuntu Interactif

Nous allons démarrer le conteneur et y attacher un terminal pour pouvoir taper des commandes directement à l'intérieur

```
docker run -it --name ubuntu-nginx -p 8080:80 ubuntu:latest bash
```

2.Installer Nginx dans le Conteneur

À l'intérieur du shell du conteneur Ubuntu, exécutez ces commandes :

Mettre à jour la liste des paquets :

```
apt update
```

Installer Nginx :

```
apt install -y nginx
```

Supprimer toutes les listes APT

```
rm -rf /var/lib/apt/lists/*
```

3.Démarrer nginx

```
service nginx start
```

Vérifier qu'il tourne :

```
service nginx status
```

4. Ouvrir la page d'accueil

Dans votre navigateur, ouvrez :

👉 **http://localhost:8080**

Vous devriez voir la page par défaut

5. Quitter le conteneur

```
exit
```

6. Relancer et entrer dans le conteneur

```
docker start -ai ubuntu-nginx
```

Explications :

start : démarre le conteneur arrêté
-a : attache le terminal pour voir les logs
-i : mode interactif
C'est exactement comme si tu relançais ton docker run mais sans recréer un nouveau conteneur.

⚠ Quand on a fais docker start -ai ubuntu-nginx, le conteneur redémarre, mais le service nginx n'est pas lancé automatiquement. Donc même si le conteneur tourne, rien n'écoute sur le port 80, et donc http://localhost:8080/ ne répond pas.

Solutions:
Il y a plusieurs façons de faire en sorte que nginx se lance automatiquement quand le conteneur démarre. Le problème, c'est que dans votre conteneur Ubuntu "classique", nginx n'est pas configuré pour démarrer tout seul comme sur une VM classique (pas de systemd).

1. Démarrer nginx manuellement

Après avoir relancé le conteneur :

```
service nginx start
```

Puis ouvrez http://localhost:8080 dans votre navigateur.

2. La solution idéale : Dockerfile

Créer un Dockerfile vous permet de construire une image Ubuntu + Nginx prête à l'emploi, avec nginx qui démarre automatiquement :

```
# 1. Partir de l'image Ubuntu officielle
    FROM ubuntu
# 2. Installer Nginx
    RUN apt update
    RUN apt install -y nginx
    RUN rm -rf /var/lib/apt/lists/*
# 3. Exposer le port 80 (celui utilisé par Nginx)
    EXPOSE 80
# 4. Lancer Nginx en mode non-démon afin que le conteneur reste actif
    CMD ["nginx", "-g", "daemon off;"]
```

Ensuite :

```
docker build -t ubuntu-nginx .
docker run -d -p 8080:80 ubuntu-nginx
```

- Chaque fois que vous lancez le conteneur, nginx démarre automatiquement.
- Vous n'avez plus besoin de service nginx start.
- Testez:
  Ouvrez : **http://localhost:8080**

Explication:

Étape manuelle	Équivalent Dockerfile
docker run……ubunut	FROM ubuntu
apt update	RUN apt update
apt install -y nginx	RUN apt install -y nginx
rm -rf /var/lib/apt/lists/*	RUN rm -rf /var/lib/apt/lists/*
service nginx start	CMD ["nginx", "-g", "daemon off;"]

Dockerfile optimisé & sécurisé:

```
# 1. Utiliser une image récente
        FROM ubuntu:24.04
# 2. Installation sécurisée et optimisée de Nginx
        RUN apt-get update \
           && apt-get install -y \
           && rm -rf /var/lib/apt/lists/*
# 3. Utiliser un utilisateur non-root pour limiter les dégâts potentiels
RUN useradd -r -d /usr/share/nginx/html -s /usr/sbin/nologin nginxuser \
&& chown -R -d /usr/share/nginx/html nginxuser  /var/www /var/log/nginx
/var/lib/nginx /etc/nginx
# 4. Exposer le port HTTP
EXPOSE 80
# 5. Passe en utilisateur non root
USER nginxuser
# 6. Lance nginx en mode non-démon
CMD ["nginx", "-g", "daemon off;"]
```

Améliorations:
✔ Image Ubuntu épinglée à une version :
→ On évite un changement de comportement futur (ubuntu:latest n'est pas stable).
✔ Utilisateur non-root :
→ Limite fortement l'impact en cas de compromission.
✔ Fusion des RUN
Réduit le nombre de couches → image plus légère.

Explication:

1- useradd -r -d /usr/share/nginx/html -s /usr/sbin/nologin nginxuser

But : créer un utilisateur système nommé nginxuser.

Détails des options :

- useradd : utilitaire pour créer un compte utilisateur.
- -r : crée un **utilisateur système** (UID généralement < 1000).
  → système : pas un compte humain, pas d'entrée dans /etc/skel, pas de mot de passe par défaut, etc.
- -d /var/www (ou --home) : définit le répertoire « home » de l'utilisateur à /var/www.
  → pas forcément créé automatiquement par useradd selon la configuration ; ici on suppose que /var/www existe ou qu'on le crée ailleurs.
- -s /usr/sbin/nologin (ou --shell) : définit le shell de connexion à nologin (ou /bin/false selon la distro).
  → empêche toute connexion interactive (pas de login SSH/shell pour cet utilisateur).
- nginxuser : nom de l'utilisateur créé.

2- chown -R nginxuser:nginxuser /var/www /var/log/nginx /var/lib/nginx /etc/nginx

But : donner la propriété des dossiers listés à nginxuser (utilisateur) et nginxuser (groupe).

Correction Lab2

1) Trouver l'image officielle PostgreSQL sur Docker Hub

Rendez-vous ici :
https://hub.docker.com/_/postgres

L'image officielle se nomme **postgres**.

2) Lancer un conteneur PostgreSQL

```
docker run \
   --name postgresql-db \
   -e POSTGRES_PASSWORD=postgres \
   -d postgres
```

Explications :

--name postgresql-db → nom du conteneur

-e POSTGRES_PASSWORD=postgres → mot de passe de l'utilisateur postgres

-d → exécution en arrière-plan

postgres → nom de l'image officielle

Vérifier que le conteneur fonctionne :

```
docker ps
```

3) Création d'une table et insertion des données:

- **Ouvrir un terminal dans le conteneur :**

  ```
  docker exec -it postgresql-db bash
  ```

- **Lancer le client PostgreSQL :**

  ```
  psql -U postgres
  ```

- **Exécuter vos requêtes :**

  ```
  CREATE TABLE test (id SERIAL PRIMARY KEY, name TEXT);
  INSERT INTO test (name) VALUES ('Alice');SELECT * FROM test;
  ```

- **Sortir du client :**

  ```
  \q
  ```

- **Sortir du conteneur :**

  ```
  exit
  ```

Méthode 2 : exécuter une commande SQL sans entrer dans le conteneur

- Vous pouvez envoyer une requête SQL directement :

```
docker exec -it postgresql-db psql -U postgres -c "SELECT version();"
```

- Créer une table :

```
docker exec -it postgresql-db psql -U postgres -c "CREATE TABLE students(id SERIAL, name TEXT);"
```

- Afficher les tables :

docker exec -it postgresql-db psql -U postgres -c "\dt"

Méthode 3 : exécuter un fichier .sql

Si vous avez un fichier local script.sql, vous pouvez l'exécuter ainsi :

```
docker exec -i postgresql-db psql -U postgres < script.sql
```

Exemple de fichier script.sql :

```
CREATE TABLE demo (id SERIAL PRIMARY KEY, value TEXT);
INSERT INTO demo (value) VALUES ('Bonjour Docker !');
SELECT * FROM demo;
```