

Containers

Containers Notebook Demo

Emmanuel BRUNO

Université de Toulon, LIS UMR CNRS 7020

emmanuel.bruno@univ-tln.fr

2026-10-14

Table des matières

1	Introduction	1
1.1	Contexte	1
2	Préparation de l'environnement	2
2.1	Configuration de l'accès à Docker ..	2
3	Premier container	2
3.1	Exécution d'un container simple	2
4	Images Docker	3
4.1	Gestion des images	3
5	Containers actifs	3
5.1	Surveillance des containers	3
6	Container interactif	4
6.1	Exécution interactive	4
7	Nettoyage	6
7.1	Suppression des containers	6
8	Conclusion	6

1 Introduction

Ce document fournit des notes complètes pour l'introduction aux containers Docker. Il couvre les concepts clés, deux approches d'exécution depuis un notebook (Docker-in-Docker vs engine de l'hôte), et les commandes essentielles.

1.1 Contexte

Le devcontainer utilisé pour ces démonstrations contient les clients `docker` et `podman`. Pour les démonstrations reproduites ici nous utilisons le client `docker`.

Deux approches courantes pour exécuter Docker depuis un environnement isolé :

- **Docker-in-Docker:** un service Docker s'exécute dans un container séparé. Le client Docker du notebook se connecte au démon via TCP (port 2375). Avantage : isolation complète du démon de l'hôte. Inconvénient : complexité réseau, performances légèrement réduites.
- **Utiliser l'engine de l'hôte (dooD, Docker-out-of-Docker):** le client Docker dans le notebook utilise le socket Unix de l'hôte (`/var/run/docker.sock`). Avantage : accès direct aux images/containers de l'hôte et meilleures performances. Inconvénient : moindre isolation (risques de sécurité si le notebook est compromis).

Dans ces notes, les exemples sont réalisés avec l'engine de l'hôte via le socket Unix partagé. Comme le démon peut être partagé entre plusieurs exécutions simultanées, chaque ressource créée par la leçon est repérée par le label `ebpro-render=true`, ce qui permet un nettoyage ciblé sans jamais toucher aux ressources des autres.

2 Préparation de l'environnement

2.1 Configuration de l'accès à Docker

Par défaut, le client Docker utilise le socket Unix de l'engine de l'hôte (`/var/run/docker.sock`) : il n'est donc pas nécessaire d'exporter `DOCKER_HOST`. En CI, le socket de l'hôte est monté dans le conteneur de rendu (dooD).

Le nettoyage ci-dessous est **ciblé par label** : il ne supprime que les containers marqués `ebpro-render=true`, laissés par une exécution précédente de cette leçon. Sur un démon partagé, il ne faut **jamais** exécuter `docker system prune` complet, qui détruirait les ressources des exécutions concurrentes.

Tous les containers lancés par la suite dans cette leçon seront créés avec l'option `--label ebpro-render=true` afin de pouvoir être nettoyés précisément.

3 Premier container

3.1 Exécution d'un container simple

Vérifions que la communication fonctionne avec l'image `hello-world` :

```
docker run --label ebpro-render=true
hello-world
```

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
```

```
4f55086f7dd0: Pulling fs layer
```

```
4f55086f7dd0:
Downloading 2.415kB/2.415kB
```

```
4f55086f7dd0: Verifying Checksum
```

```
4f55086f7dd0: Download complete
```

```
4f55086f7dd0:
Extracting 2.415kB/2.415kB
```

```
4f55086f7dd0:
Extracting 2.415kB/2.415kB
```

```
4f55086f7dd0: Pull complete
Digest: sha256:11a3eb1597e864c51b3f56733cb2e8a90bf0c284
Status: Downloaded newer image for
hello-world:latest
```

```
Hello from Docker!
This message shows that your
installation appears to be working
correctly.
```

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.

2. The Docker daemon pulled the "hello-world" image from the Docker Hub.

```
(amd64)
```

3. The Docker daemon created a new container from that image which runs the

```
executable that produces the
output you are currently reading.
```

4. The Docker daemon streamed that output to the Docker client, which sent it

```
to your terminal.
```

To try something more ambitious, you

```
can run an Ubuntu container with:
$ docker run -it ubuntu bash
```

Share images, automate workflows, and more with a free Docker ID:
<https://hub.docker.com/>

For more examples and ideas, visit:
<https://docs.docker.com/get-started/>

Cette commande effectue plusieurs opérations :

1. Recherche l'image `hello-world` localement
2. Si absente, la télécharge depuis Docker Hub
3. Crée un nouveau container à partir de cette image
4. Exécute le container qui affiche un message de bienvenue
5. Le container s'arrête automatiquement après avoir terminé

L'image `hello-world` est particulièrement légère (environ 1.84 kB) et sert principalement à vérifier que Docker est correctement installé et configuré.

4 Images Docker

4.1 Gestion des images

Pour lister toutes les images disponibles localement :

```
docker images
```

```
i Info → U In Use
IMAGE                                     ID
DISK USAGE  CONTENT SIZE  EXTRA
ghcr.io/ebpro/solen:quarto-full-
build-20261010T172146Z
                                     b4ed11d0542c
```

```
6.88GB      0B      U
hello-
world:latest      e2ac70e7319a
10.1kB      0B      U
```

Cette commande affiche :

- **REPOSITORY**: le nom de l'image (ex: `ubuntu`, `nginx`)
- **TAG**: la version ou variante de l'image (ex: `latest`, `22.04`)
- **IMAGE ID**: identifiant unique de l'image
- **CREATED**: date de création de l'image
- **SIZE**: taille de l'image sur le disque

Les images Docker sont construites en couches (layers), ce qui permet de réutiliser les couches communes entre différentes images, optimisant ainsi l'espace disque et le temps de téléchargement.

5 Containers actifs

5.1 Surveillance des containers

Pour voir les containers en cours d'exécution :

```
docker ps
```

```
CONTAINER    ID                IMAGE
COMMAND      CREATED
STATUS       PORTS            NAMES
415b7921a604      ghcr.io/ebpro/
solen:quarto-
full-build-20261010T172146Z      "bash
-lc 'source /h..." 22 seconds ago Up
21 seconds      quarto-
render-38598955399
```

Pour voir tous les containers (en cours d'exécution et arrêtés) :

```
docker ps -a
```

CONTAINER	ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
288e0bb3fb8e		hello-world	"/hello"	2 seconds ago	Exited (0)	1 second ago	frosty_goldstine
9e46a5e3b4d5		hello-world	"/hello"	4 seconds ago	Exited (0)	2 seconds ago	pedantic_meninsky
415b7921a604		ghcr.io/ebpro/solen:quarto-full-build-20261010T172146Z	"bash -lc 'source /h..."	23 seconds ago	Up	22 seconds	quarto-render-38598955399

Les informations affichées incluent :

- **CONTAINER ID**: identifiant court du container
- **IMAGE**: image source du container
- **COMMAND**: commande exécutée au démarrage
- **CREATED**: date de création du container
- **STATUS**: état actuel (Up pour actif, Exited pour arrêté)
- **PORTS**: mappages de ports entre l'hôte et le container
- **NAMES**: nom attribué au container (généralisé automatiquement ou défini par l'utilisateur)

6 Container interactif

6.1 Exécution interactive

Lançons un container Ubuntu avec un shell interactif :

```
docker run --label ebpro-render=true
ubuntu uname -a
```

Unable to find image 'ubuntu:latest' locally

latest: Pulling from library/ubuntu

4e07a0f12b2c: Pulling fs layer

06ad70e463aa: Pulling fs layer

06ad70e463aa:
Downloading 388B/388B

06ad70e463aa: Verifying Checksum

06ad70e463aa: Download complete

4e07a0f12b2c:
Downloading 425.4kB/41.84MB

4e07a0f12b2c:
Downloading 6.815MB/41.84MB

4e07a0f12b2c:
Downloading 15.33MB/41.84MB

4e07a0f12b2c:
Downloading 23.85MB/41.84MB

4e07a0f12b2c:
Downloading 31.52MB/41.84MB

4e07a0f12b2c:
Downloading 38.34MB/41.84MB

4e07a0f12b2c: Verifying Checksum

4e07a0f12b2c: Download complete

4e07a0f12b2c:
Extracting 426kB/41.84MB

4e07a0f12b2c:
Extracting 3.834MB/41.84MB

```
4e07a0f12b2c:  
Extracting 7.242MB/41.84MB
```

```
4e07a0f12b2c:  
Extracting 9.372MB/41.84MB
```

```
4e07a0f12b2c:  
Extracting 11.5MB/41.84MB
```

```
4e07a0f12b2c:  
Extracting 15.34MB/41.84MB
```

```
4e07a0f12b2c:  
Extracting 17.89MB/41.84MB
```

```
4e07a0f12b2c:  
Extracting 18.74MB/41.84MB
```

```
4e07a0f12b2c:  
Extracting 20.45MB/41.84MB
```

```
4e07a0f12b2c:  
Extracting 22.58MB/41.84MB
```

```
4e07a0f12b2c:  
Extracting 25.56MB/41.84MB
```

```
4e07a0f12b2c:  
Extracting 28.97MB/41.84MB
```

```
4e07a0f12b2c:  
Extracting 32.37MB/41.84MB
```

```
4e07a0f12b2c:  
Extracting 35.36MB/41.84MB
```

```
4e07a0f12b2c:  
Extracting 36.21MB/41.84MB
```

```
4e07a0f12b2c:  
Extracting 37.91MB/41.84MB
```

```
4e07a0f12b2c:  
Extracting 38.76MB/41.84MB
```

```
4e07a0f12b2c:  
Extracting 39.19MB/41.84MB
```

```
4e07a0f12b2c:  
Extracting 39.62MB/41.84MB
```

```
4e07a0f12b2c:  
Extracting 40.47MB/41.84MB
```

```
4e07a0f12b2c:  
Extracting 41.32MB/41.84MB
```

```
4e07a0f12b2c:  
Extracting 41.75MB/41.84MB
```

```
4e07a0f12b2c:  
Extracting 41.84MB/41.84MB
```

```
4e07a0f12b2c: Pull complete
```

```
06ad70e463aa: Extracting 388B/388B
```

```
06ad70e463aa: Extracting 388B/388B
```

```
06ad70e463aa: Pull complete
```

```
Digest: sha256:f144425ff09be612d6d9ad965196e9cdc23dae1f
```

```
Status: Downloaded newer image for  
ubuntu:latest
```

```
Linux e34d60656843 6.8.0-142-generic  
#142-Ubuntu SMP PREEMPT_DYNAMIC Wed  
Sep 2 14:24:27 UTC 2026 x86_64 GNU/  
Linux
```

Pour une session interactive (non compatible avec l'exécution notebook) :

```
docker run -it --label ebpro-  
render=true ubuntu bash
```

Options utilisées :

- `-i` (interactive) : garde l'entrée standard (STDIN) ouverte
- `-t` (tty) : alloue un pseudo-terminal
- `ubuntu` : nom de l'image à utiliser
- `bash` : commande à exécuter dans le container

Cette combinaison permet d'obtenir un shell interactif dans le container, similaire à une connexion SSH. Le container s'exécute tant que le shell est actif et s'arrête dès que vous quittez (commande `exit` ou Ctrl+D).

7 Nettoyage

7.1 Suppression des containers

Pour supprimer un container arrêté spécifique (remplacer `<container_id>` par l'ID réel) :

```
docker rm <container_id>
```

Pour supprimer les containers de cette leçon en une seule commande, on cible le label `ebpro-render=true` :

```
docker ps -aq --filter label=ebpro-render=true | xargs -r docker rm -f
```

```
11bc830e2552
e34d60656843
288e0bb3fb8e
9e46a5e3b4d5
```

Il est recommandé de nettoyer régulièrement les containers arrêtés pour éviter l'accumulation d'espace disque inutile. La commande générique `docker container prune` supprime *tous* les containers arrêtés du démon : sur un démon partagé (CI, devcontainer mutualisé), elle peut détruire les ressources d'exécutions concurrentes. Préférez toujours un nettoyage ciblé par label.

Pour supprimer une image (remplacer `<image_id>` par l'ID réel) :

```
docker rmi <image_id>
```

Notez qu'une image ne peut être supprimée que si aucun container (même arrêté) ne l'utilise.

8 Conclusion

Cette introduction couvre les commandes Docker essentielles pour démarrer avec la containerisation. Les concepts abordés incluent :

- La différence entre images et containers
- L'exécution de containers simples et interactifs
- La gestion et la surveillance des ressources Docker
- Les bonnes pratiques de nettoyage

Pour aller plus loin, explorez les Dockerfiles pour créer vos propres images, Docker Compose pour orchestrer plusieurs containers, et les réseaux Docker pour la communication inter-containers.