Docker — Les ports

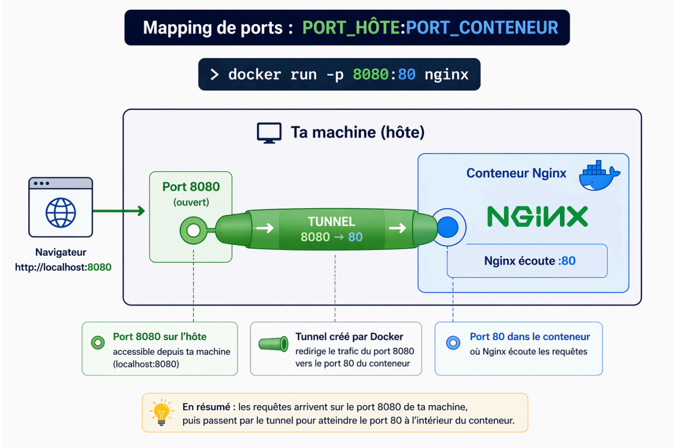
Objectifs

À la fin de cette page, tu seras capable de :

- Expliquer ce qu'est un port réseau
- Comprendre pourquoi un conteneur possède ses propres ports isolés
- Lire et écrire correctement la syntaxe `hôte:conteneur`
- Identifier les ports standards des services courants
- Résoudre un conflit de ports
- Distinguer `EXPOSE` dans un Dockerfile de `-p` dans `docker run`

5 notions-clés

1. **Port** — Numéro qui identifie un service précis sur une machine (de 0 à 65535)
2. **Isolation** — Chaque conteneur possède ses propres ports, indépendants du système hôte
3. **Mapping** — La syntaxe `PORT_HÔTE:PORT_CONTENEUR` crée un "tunnel" entre ta machine et le conteneur
4. **Conflit** — Deux services ne peuvent pas utiliser le même port hôte en même temps
5. **`EXPOSE`** — Simple documentation dans le Dockerfile ; c'est `-p` qui ouvre réellement le port



C'est quoi un port ?

Une machine connectée au réseau peut faire tourner **plusieurs services en même temps** : un serveur web, une base de données, un serveur SSH, un serveur mail...

Pour différencier ces services, on utilise les **ports** : des numéros de 0 à 65535 associés à chaque connexion réseau.

Analogie : pense à une adresse postale d'immeuble.

© 2025 Mr. Lambrechts

- L'adresse IP = le numéro de l'immeuble
- Le port = le numéro de l'appartement

Quand tu tapes `http://localhost:8080` dans ton navigateur :

- `localhost` = ta propre machine (adresse IP `127.0.0.1`)
- `8080` = le port, c'est-à-dire le "bureau" précis auquel tu frappes

---

Les ports standards à connaître

Certains ports sont réservés par convention internationale pour des services précis :

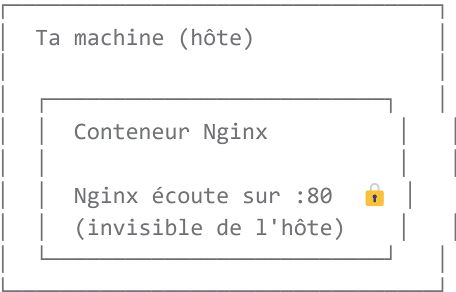
Port	Service	Protocole
80	HTTP (web non sécurisé)	TCP
443	HTTPS (web sécurisé)	TCP
22	SSH (accès distant)	TCP
21	FTP (transfert de fichiers)	TCP
25	SMTP (envoi d'emails)	TCP
3306	MySQL / MariaDB	TCP
5432	PostgreSQL	TCP
6379	Redis (cache)	TCP
27017	MongoDB	TCP
3000	Node.js / apps web modernes (convention)	TCP
8080	HTTP alternatif (souvent utilisé en dev)	TCP
9000	PHP-FPM / divers	TCP

Les ports **0 à 1023** sont dits "privilégiés" : sur Linux, seul `root` peut les utiliser. C'est pourquoi en développement on utilise souvent des ports comme `8080` ou `8443` au lieu de `80` et `443`.

Le problème d'isolation des conteneurs

Chaque conteneur Docker est **isolé** du système hôte. Il possède sa propre interface réseau virtuelle, ses propres ports.

Quand Nginx démarre dans un conteneur, il écoute sur le port **80 du conteneur**. Mais ce port 80 n'est **pas visible** depuis ta machine par défaut.




```
→ http://localhost:80  ❌  (rien n'écoute côté hôte)
```

Pour rendre ce port accessible, il faut créer un **mapping** entre un port de ta machine et le port du conteneur.

© 2025 Mr. Lambrechts

Le mapping de ports — `PORT_HÔTE:PORT_CONTENEUR`

L'option `-p` de `docker run` crée ce tunnel :

```
1  docker run -p 8080:80 nginx
```

```
┌──────────────────────────────────────────────┐
│  Ta machine (hôte)                            │
│                                               │
│  Port 8080  ──────────────────────────────┐   │
│  (ouvert)                                 │   │
│                  ┌─────────────────────┐   │   │
│                  │  Conteneur Nginx     │  │   │  │
│                  │                      │◄─┘   │
│                  │  Nginx écoute :80    │      │
│                  └─────────────────────┘      │
└──────────────────────────────────────────────┘
```

```
→ http://localhost:8080  ✅  (ta machine → conteneur)
```

Lecture de la syntaxe : toujours lire de gauche à droite

- **À gauche** : le port sur **ta machine** (ce que tu tapes dans le navigateur)
- **À droite** : le port **dans le conteneur** (là où le service écoute réellement)

```
1  -p 8080:80
2      ↑    ↑
3      |    └── Port dans le conteneur (Nginx écoute ici)
4      └─────── Port sur ta machine (tu accèdes par ici)
```

Pourquoi utiliser un port hôte différent ?

Cas 1 — Le port est déjà pris sur la machine hôte

Si tu as déjà Apache ou IIS qui écoute sur le port 80 de ta machine, tu ne peux pas démarrer un deuxième service sur ce même port. Tu utilises un port libre :

```
1  # Apache utilise déjà le port 80 sur la machine hôte
2  docker run -p 8080:80 nginx    # Nginx accessible sur :8080
3  docker run -p 8081:80 apache   # Apache conteneur accessible sur :8081
```

Cas 2 — Tu fais tourner plusieurs instances du même service

```
1  # Trois serveurs PHP pour trois projets différents
2  docker run -p 8001:80 projet-alpha
3  docker run -p 8002:80 projet-beta
4  docker run -p 8003:80 projet-gamma
```

Chaque projet est accessible sur un port différent :

- http://localhost:8001 → projet-alpha
- http://localhost:8002 → projet-beta
- http://localhost:8003 → projet-gamma

Cas 3 — Le même port dans plusieurs conteneurs

Dans le conteneur, le port ne change pas (c'est toujours le port natif de l'application). Seul le port hôte doit être unique.

```
# Deux bases MySQL, chacune sur son port hôte
docker run -p 3306:3306 --name db-dev  mysql:8.0   # dev
docker run -p 3307:3306 --name db-test mysql:8.0   # test
```

La connexion à `localhost:3307` atteint le conteneur `db-test`, même si MySQL écoute toujours sur le port `3306` à l'intérieur.

Ce qui se passe en cas de conflit

Si tu essaies de mapper deux services sur le même port hôte, Docker refuse avec une erreur claire :

```
docker run -p 8080:80 nginx     # ✅ Premier service : OK
docker run -p 8080:80 apache    # ❌ Erreur !
```

```
Error response from daemon: driver failed programming external connectivity
on endpoint apache: Bind for 0.0.0.0:8080 failed:
port is already allocated
```

Solution : changer le port hôte du deuxième service.

Mapper plusieurs ports

Un même conteneur peut exposer plusieurs ports :

```
# Gitea : port web + port SSH Git
docker run -p 3000:3000 -p 2222:22 gitea/gitea
```

```
# Même chose en Docker Compose
services:
  gitea:
    image: gitea/gitea
    ports:
      - "3000:3000"   # interface web
      - "2222:22"     # SSH pour git push/pull
```

Écouter sur une interface spécifique

Par défaut, `-p 8080:80` expose le port sur **toutes les interfaces réseau** de ta machine (`0.0.0.0`). Cela signifie que les autres machines sur le réseau local peuvent aussi y accéder.

Pour restreindre l'accès **à ta machine uniquement** (loopback) :

```
# Accessible uniquement depuis localhost, pas depuis le réseau
docker run -p 127.0.0.1:8080:80 nginx
```

```
# En Compose
ports:
  - "127.0.0.1:8080:80"
```

C'est une bonne pratique pour les services sensibles (base de données, admin) qui ne doivent pas être accessibles depuis le réseau local ou Internet.

```
# ✅ Bonne pratique pour MySQL : jamais exposé sur le réseau
-p 127.0.0.1:3306:3306

# ⚠️ À éviter en production : MySQL visible sur tout le réseau
-p 3306:3306
```

`EXPOSE` dans le Dockerfile vs `-p` dans `docker run`

C'est une confusion très fréquente.

`EXPOSE` — Documentation seulement

```
FROM nginx
EXPOSE 80
```

`EXPOSE` dans un Dockerfile **ne publie pas le port**. Il sert uniquement de **documentation** : il indique aux utilisateurs de l'image sur quel port le service écoute.

C'est l'équivalent d'un commentaire : "Ce conteneur écoute sur le port 80."

`-p` — Ce qui ouvre réellement le port

```
docker run -p 8080:80 nginx
```

C'est `-p` (ou `ports:` dans Compose) qui crée réellement le mapping et rend le port accessible depuis ta machine.

`-P` majuscule — Mapping automatique

```
1  docker run -P nginx
```

Le `-P` majuscule publie automatiquement **tous les ports déclarés avec `EXPOSE`** dans le Dockerfile, sur des ports hôtes aléatoires (ex: `0.0.0.0:32768->80/tcp`).

Peu utilisé en pratique, mais pratique pour tester rapidement une image inconnue.

Résumé

	Publie le port ?	Utilisation
`EXPOSE 80` (Dockerfile)	❌ Non	Documentation uniquement
`docker run -p 8080:80`	✅ Oui	Mapping explicite (recommandé)
`docker run -P`	✅ Oui	Mapping automatique sur ports aléatoires

Les ports dans Docker Compose

Dans un fichier `docker-compose.yml`, les ports se déclarent sous `ports:` :

```
1  services:
2    web:
3      image: nginx:1.25
4      ports:
5        - "8080:80"             # hôte:conteneur (avec guillemets recommandés)
6        - "127.0.0.1:443:443"   # restreint à localhost
```

💡 **Pourquoi les guillemets ?** En YAML, `80:80` pourrait être interprété comme un nombre (base 60 en vieux YAML). Les guillemets évitent toute ambiguïté.

La communication interne entre services

Quand deux services sont dans le même réseau Docker Compose, ils se contactent **directement par leur port interne**, sans passer par le port hôte.

```
1   services:
2     app:
3       image: php:8.3-apache
4       ports:
5         - "8080:80"        # exposé vers l'extérieur
6
7     db:
8       image: mysql:8.0
9       # Pas de "ports:" ici → MySQL n'est PAS accessible depuis ta machine
10      # Mais "app" peut le contacter via le réseau Docker interne
```

Dans le code PHP du conteneur `app` :

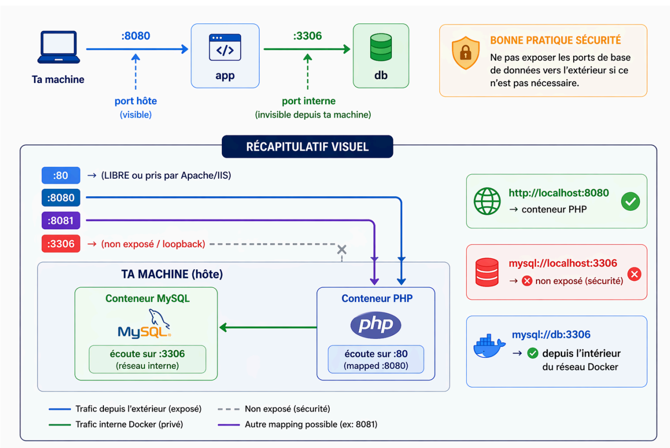
```
1  // On utilise le port INTERNE du conteneur db (3306)
2  // pas le port hôte
3  $pdo = new PDO("mysql:host=db;port=3306;dbname=...", ...);
```

© 2025 Mr. Lambrechts

C'est une bonne pratique de sécurité : **ne pas exposer les ports de base de données vers l'extérieur** si ce n'est pas nécessaire.

```
Ta machine  ──:8080──►  app  ──:3306──►  db
                ↑                  ↑
           port hôte          port interne
           (visible)         (invisible depuis ta machine)
```

Récapitulatif visuel




```
                    TA MACHINE (hôte)
┌─────────────────────────────────────────────────────────┐
│                                                         │
│  :80   → (LIBRE ou pris par Apache/IIS)                 │
│  :8080 ──────────────────────────────────────────┐      │
│  :8081 ────────────────────────────────┐         │      │
│  :3306 → (non exposé / loopback)       │         │      │
│                                        │         │      │
│      ┌────────────────────────┐    ┌─────────────────┐  │
│      │  Conteneur MySQL       │    │  Conteneur PHP  │  │
│      │                        │    │                 │  │
│      │  écoute sur :3306  ◄───┼────│  écoute sur :80 │  │
│      │  (réseau interne)      │    │  (mapped :8080) │  │
│      └────────────────────────┘    └─────────────────┘  │
│                                                         │
└─────────────────────────────────────────────────────────┘

http://localhost:8080  → conteneur PHP  ✅
mysql://localhost:3306 → ❌ non exposé (sécurité)
mysql://db:3306        → ✅ depuis l'intérieur du réseau Docker
```

Tableau de référence rapide

Syntaxe	Signification
`-p 8080:80`	Port 8080 hôte → port 80 conteneur
`-p 3306:3306`	Même port des deux côtés
`-p 127.0.0.1:8080:80`	Accessible uniquement depuis localhost
`-P`	Publie tous les `EXPOSE` sur ports aléatoires
`ports: ["8080:80"]`	Équivalent Compose de `-p 8080:80`

© 2025 Mr. Lambrechts

Syntaxe	Signification
Pas de `ports:`	Le service n'est accessible que depuis le réseau Docker interne