🐬 Docker — Docker Compose

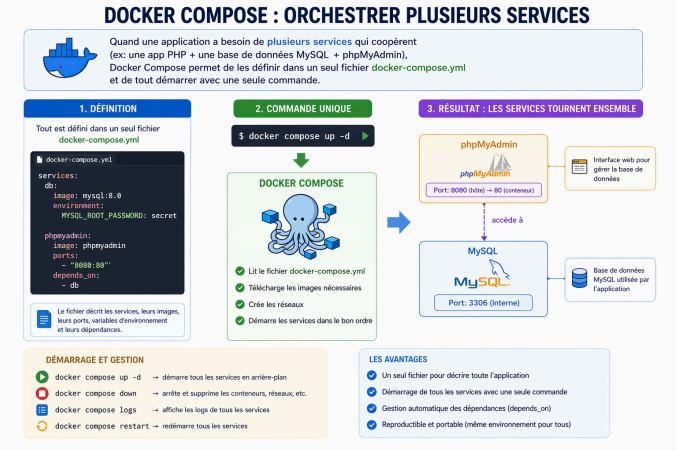
Objectifs

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

- Expliquer la structure d'un fichier `docker-compose.yml`
- Utiliser les directives essentielles : `image`, `build`, `ports`, `volumes`, `environment`, `networks`, `depends_on`, `restart`
- Gérer un projet multi-services avec les commandes `docker compose`
- Utiliser un fichier `.env` avec Docker Compose
- Déployer une application PHP + MySQL + phpMyAdmin fonctionnelle

5 notions-clés

1. **`services`** — La section qui liste chaque "brique" de l'application (web, db, cache, etc.)
2. **`build`** — Indique à Compose de construire l'image depuis un Dockerfile plutôt que de la télécharger
3. **`depends_on`** — Définit l'ordre de démarrage : ce service démarre **après** un autre
4. **`env_file`** — Charge les variables d'environnement depuis un fichier `.env`
5. **`docker compose up -d --build`** — La commande complète pour (re)construire et (re)démarrer tous les services



Pourquoi Docker Compose ?

Avec `docker run`, chaque conteneur est lancé séparément avec une longue commande. Dès qu'on a 2 ou 3 services, ça devient ingérable :

```
# ❌ Sans Compose : 3 commandes, des options à ne pas oublier, ordre à respecter...
docker network create appnet
docker run -d --name db --network appnet \
```

```
  -e MYSQL_ROOT_PASSWORD=secret -e MYSQL_DATABASE=app \
  -v db_data:/var/lib/mysql mysql:8.0

docker run -d --name app --network appnet \
  -p 8080:80 -e DB_HOST=db --build ... mon-app-php

docker run -d --name pma --network appnet \
  -p 8081:80 -e PMA_HOST=db phpmyadmin:latest
```

```
# ✅ Avec Compose : tout est dans un fichier, une seule commande
docker compose up -d
```

Docker Compose permet de **décrire toute l'infrastructure** d'un projet dans un fichier texte versionnable.

Structure du fichier `docker-compose.yml`

```
# Version du format (optionnel depuis Compose v2)
# services, volumes, networks sont les 3 sections principales

services:            # ← liste des services (conteneurs)

  nom-du-service:    # ← nom libre, utilisé comme hostname réseau
    image: ...       # ← image à utiliser
    build: ...       # ← ou: construire depuis un Dockerfile
    container_name: ...
    restart: ...
    ports:
      - "hôte:conteneur"
    environment:
      CLE: valeur
    env_file:
      - .env
    volumes:
      - nom_volume:/chemin/dans/conteneur
      - ./dossier_local:/chemin/dans/conteneur
    networks:
      - nom_reseau
    depends_on:
      - autre-service

volumes:             # ← déclaration des volumes nommés
  nom_volume:

networks:            # ← déclaration des réseaux nommés
  nom_reseau:
```

Les directives en détail

image vs build

```yaml
# Utiliser une image existante depuis Docker Hub
services:
  db:
    image: mysql:8.0       # télécharge l'image mysql version 8.0

  web:
    build: .               # construit l'image depuis ./Dockerfile
    # ou avec plus d'options :
    build:
      context: .           # dossier contenant le Dockerfile
      dockerfile: Dockerfile.prod   # nom du fichier si différent de "Dockerfile"
```

ports

```yaml
ports:
  - "8080:80"     # port 8080 sur ta machine → port 80 dans le conteneur
  - "3306:3306"   # même port des deux côtés
  - "127.0.0.1:8080:80"  # exposé seulement en local (pas sur le réseau)
```

environment

```yaml
# Forme bloc (lisible)
environment:
  MYSQL_ROOT_PASSWORD: secret
  MYSQL_DATABASE: apptest
  APP_DEBUG: "true"

# Forme liste
environment:
  - MYSQL_ROOT_PASSWORD=secret
  - MYSQL_DATABASE=apptest
```

env_file

```yaml
# Charger les variables depuis un fichier .env
env_file:
  - .env
  - .env.local    # on peut en charger plusieurs
```

Docker Compose charge automatiquement `.env` pour **substituer les variables** dans le fichier `docker-compose.yml` lui-même (ex: `${DB_PASSWORD}`). L'option `env_file` les injecte aussi à l'intérieur du conteneur.

volumes

```yaml
volumes:
  # Volume nommé (géré par Docker)
  - db_data:/var/lib/mysql
```

```yaml

  # Bind mount (dossier local)
  - ./src:/var/www/html

  # Fichier unique
  - ./config/php.ini:/usr/local/etc/php/php.ini
```

depends_on

```yaml
services:
  app:
    depends_on:
      - db       # app démarre après db

  db:
    image: mysql:8.0
```

⚠️ **Attention** : `depends_on` garantit que le **conteneur** db est démarré, pas que MySQL est **prêt à accepter des connexions**. MySQL peut mettre quelques secondes à initialiser. Pour gérer ça, on ajoute une logique de retry dans l'application, ou on utilise `healthcheck` (avancé).

restart

```yaml
restart: no               # défaut : ne redémarre pas
restart: always           # redémarre toujours
restart: unless-stopped   # ✅ recommandé pour les services permanents
restart: on-failure       # uniquement en cas d'erreur
```

networks

```yaml
services:
  app:
    networks:
      - frontend
      - backend

  db:
    networks:
      - backend      # db n'est accessible que par le réseau "backend"

  nginx:
    networks:
      - frontend     # nginx expose vers l'extérieur

networks:
  frontend:
  backend:
```

Les commandes `docker compose`

Démarrage et arrêt

```
# Démarrer tous les services en arrière-plan
docker compose up -d

# (Re)construire les images ET démarrer
docker compose up -d --build

# (Re)construire uniquement un service spécifique
docker compose up -d --build app

# Arrêter les services (conserve les conteneurs et volumes)
docker compose stop

# Arrêter ET supprimer les conteneurs + réseaux
docker compose down

# Arrêter + supprimer conteneurs + réseaux + volumes
docker compose down -v
```

Supervision

```
# Voir les conteneurs du projet
docker compose ps

# Voir les logs de tous les services
docker compose logs

# Logs en temps réel
docker compose logs -f

# Logs d'un seul service
docker compose logs -f db

# Voir les logs des 50 dernières lignes
docker compose logs --tail=50
```

Interaction

```
# Ouvrir un shell dans un service
docker compose exec app bash
docker compose exec db mysql -u root -p

# Exécuter une commande ponctuelle
docker compose exec app php artisan migrate
docker compose exec db mysqldump -u root -p mabase > backup.sql
```

Mise à jour

```bash
# Télécharger les nouvelles versions des images
docker compose pull

# Redémarrer un service sans tout arrêter
docker compose restart app

# Reconstruire et redémarrer uniquement le service modifié
docker compose up -d --build app
```

Exercice — Application PHP + MySQL + phpMyAdmin

Objectif : Déployer une application complète à 3 services, avec volumes persistants, réseau interne, fichier `.env` et un Dockerfile personnalisé.

Ce que tu vas apprendre : structure Compose complète, `build` + `image`, résolution de noms entre services, `.env`, persistance, commandes de supervision

---

Structure du projet

```
compose-php-mysql/
├── docker-compose.yml
├── .env
├── .env.example
├── .gitignore
├── Dockerfile
└── src/
    ├── index.php
    ├── config.php
    └── pages/
        ├── ajouter.php
        └── supprimer.php
```

Étape 1 — Créer le dossier et les fichiers de base

Crée le dossier `compose-php-mysql/` sur ton Bureau.

Étape 2 — Le fichier `.env`

```bash
# .env — NE PAS COMMITTER CE FICHIER
DB_ROOT_PASSWORD=rootsecret
DB_NAME=gestion_app
DB_USER=appuser
DB_PASSWORD=appsecret
APP_PORT=8080
PMA_PORT=8081
```

Étape 3 — Le fichier `.env.example`

```
# .env.example — Copier en .env et remplir les valeurs
DB_ROOT_PASSWORD=
DB_NAME=
DB_USER=
DB_PASSWORD=
APP_PORT=8080
PMA_PORT=8081
```

Étape 4 — Le `.gitignore`

```
.env
```

Étape 5 — Le Dockerfile

```
FROM php:8.3-apache

# Extensions nécessaires pour la connexion MySQL
RUN docker-php-ext-install pdo pdo_mysql

WORKDIR /var/www/html

COPY src/ .

EXPOSE 80
```

Étape 6 — Le `docker-compose.yml`

```
services:

  # ── Base de données MySQL ──────────────────────────────
  db:
    image: mysql:8.0
    container_name: projet_db
    restart: unless-stopped
    environment:
      MYSQL_ROOT_PASSWORD: ${DB_ROOT_PASSWORD}
      MYSQL_DATABASE: ${DB_NAME}
      MYSQL_USER: ${DB_USER}
      MYSQL_PASSWORD: ${DB_PASSWORD}
    volumes:
      - db_data:/var/lib/mysql
    networks:
      - appnet

  # ── Application PHP ────────────────────────────────────
  app:
    build: .
    container_name: projet_app
```

```
    restart: unless-stopped
    ports:
      - "${APP_PORT}:80"
    environment:
      DB_HOST: db              # nom du service MySQL → résolu automatiquement
      DB_NAME: ${DB_NAME}
      DB_USER: ${DB_USER}
      DB_PASSWORD: ${DB_PASSWORD}
    volumes:
      - ./src:/var/www/html    # bind mount pour modifier le code en direct
    networks:
      - appnet
    depends_on:
      - db

  # ── phpMyAdmin ──────────────────────────────────────────────
  phpmyadmin:
    image: phpmyadmin:latest
    container_name: projet_pma
    restart: unless-stopped
    ports:
      - "${PMA_PORT}:80"
    environment:
      PMA_HOST: db             # même chose : "db" est résolu via le réseau Docker
      PMA_PORT: 3306
      MYSQL_ROOT_PASSWORD: ${DB_ROOT_PASSWORD}
    networks:
      - appnet
    depends_on:
      - db

# ── Volumes ─────────────────────────────────────────────────
volumes:
  db_data:

# ── Réseaux ─────────────────────────────────────────────────
networks:
  appnet:
```

Étape 7 — L'application PHP

`src/config.php` :

```php
<?php
$host     = getenv('DB_HOST')     ?: 'db';
$dbname   = getenv('DB_NAME')     ?: 'gestion_app';
$user     = getenv('DB_USER')     ?: 'root';
$password = getenv('DB_PASSWORD') ?: '';

try {
    $pdo = new PDO(
        "mysql:host=$host;dbname=$dbname;charset=utf8mb4",
        $user,
        $password,
        [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION]
```

```
        );
    } catch (PDOException $e) {
        // En dev, on affiche l'erreur. En prod, on la loggue discrètement.
        die('<p style="color:red">Connexion impossible : ' . $e->getMessage() . '</p>');
    }

    // Création de la table si elle n'existe pas encore
$pdo->exec("
        CREATE TABLE IF NOT EXISTS taches (
            id       INT AUTO_INCREMENT PRIMARY KEY,
            titre    VARCHAR(255) NOT NULL,
            statut   ENUM('en cours','terminée') DEFAULT 'en cours',
            cree_le  DATETIME DEFAULT CURRENT_TIMESTAMP
        )
");
?>
```

`src/index.php` :

```
    <?php require 'config.php'; ?>
    <!DOCTYPE html>
    <html lang="fr">
    <head>
        <meta charset="UTF-8">
        <title>Gestionnaire de tâches — Docker</title>
<link rel="stylesheet"
      href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
    </head>
    <body class="bg-dark text-light py-5">
    <div class="container" style="max-width:700px">

        <h1 class="mb-1">🐬 Gestionnaire de tâches</h1>
        <p class="text-muted mb-4 small">
            PHP <?= PHP_VERSION ?> · MySQL sur <code><?= getenv('DB_HOST') ?></code>
        </p>

        <!-- Formulaire d'ajout -->
        <form method="POST" action="pages/ajouter.php" class="mb-4">
            <div class="input-group">
             <input type="text" name="titre" class="form-control bg-secondary text-light border-0"
                    placeholder="Nouvelle tâche..." required>
                <button class="btn btn-primary" type="submit">Ajouter</button>
            </div>
        </form>

        <!-- Liste des tâches -->
<?php
        $taches = $pdo->query("SELECT * FROM taches ORDER BY cree_le DESC")->fetchAll(PDO::FETCH_A
        if (empty($taches)):
?>
            <p class="text-muted">Aucune tâche pour l'instant.</p>
        <?php else: foreach ($taches as $t): ?>
       <div class="d-flex align-items-center justify-content-between
                    card bg-secondary border-0 mb-2 p-3">
                <span class="<?= $t['statut'] === 'terminée' ? 'text-decoration-line-through text-
                    <?= htmlspecialchars($t['titre']) ?>
```

```
            </span>
            <div class="d-flex gap-2">
                <span class="badge <?= $t['statut'] === 'terminée' ? 'bg-success' : 'bg-warnir
                    <?= $t['statut'] ?>
                </span>
            <a href="pages/supprimer.php?id=<?= $t['id'] ?>"
                    class="btn btn-sm btn-outline-danger"
               onclick="return confirm('Supprimer ?')">✕</a>
            </div>
        </div>
    <?php endforeach; endif; ?>

</div>
</body>
</html>
```

`src/pages/ajouter.php` :

```php
<?php
require '../config.php';

if ($_SERVER['REQUEST_METHOD'] === 'POST' && !empty($_POST['titre'])) {
    $stmt = $pdo->prepare("INSERT INTO taches (titre) VALUES (?)");
    $stmt->execute([trim($_POST['titre'])]);
}

header('Location: ../index.php');
exit;
?>
```

`src/pages/supprimer.php` :

```php
<?php
require '../config.php';

if (!empty($_GET['id']) && is_numeric($_GET['id'])) {
    $stmt = $pdo->prepare("DELETE FROM taches WHERE id = ?");
    $stmt->execute([$_GET['id']]);
}

header('Location: ../index.php');
exit;
?>
```

Étape 8 — Lancer le projet

```
docker compose up -d --build
```

Observe la sortie :

```
[+] Building 12.3s (8/8) FINISHED
 ✓ Container projet_db   Started
 ✓ Container projet_app  Started
 ✓ Container projet_pma  Started
```

Étape 9 — Tester

Service	URL
Application PHP	http://localhost:8080
phpMyAdmin	http://localhost:8081

Dans phpMyAdmin :

- Serveur : `db`
- Utilisateur : `root`
- Mot de passe : `rootsecret` (valeur de `DB_ROOT_PASSWORD` dans `.env`)

Ajoute quelques tâches dans l'application. Vérifie qu'elles apparaissent dans phpMyAdmin (table `taches` dans la base `gestion_app`).

Étape 10 — Superviser

```
# Voir l'état des 3 conteneurs
docker compose ps

# Logs de tous les services
docker compose logs

# Logs de l'appli PHP en temps réel
docker compose logs -f app

# Logs de MySQL seulement
docker compose logs -f db
```

Étape 11 — Modifier le code en direct

Ouvre `src/index.php` et change le `<h1>`. Recharge la page dans le navigateur.

La modification est **immédiate** — sans reconstruire l'image ! C'est grâce au bind mount `./src:/var/www/html` dans le Compose.

💡 En revanche, si tu modifies le **Dockerfile** (ajout d'une extension PHP par exemple), tu dois reconstruire l'image avec `docker compose up -d --build app`.

Étape 12 — Tester la persistance

```
# Arrêter et supprimer les conteneurs
docker compose down

# Relancer (l'image est déjà construite)
docker compose up -d

# Ouvrir http://localhost:8080 → les tâches sont toujours là ✅
```

Étape 13 — Nettoyage complet

```
# Tout supprimer, y compris le volume (les tâches seront perdues)
docker compose down -v

# Supprimer l'image construite
docker compose down --rmi local
```

Récapitulatif des commandes

Commande	Action
`docker compose up -d`	Démarrer les services
`docker compose up -d --build`	(Re)construire et démarrer
`docker compose down`	Arrêter et supprimer les conteneurs
`docker compose down -v`	Idem + supprimer les volumes
`docker compose ps`	Voir l'état des services
`docker compose logs -f`	Logs en temps réel
`docker compose logs -f [service]`	Logs d'un service précis
`docker compose exec [service] bash`	Shell dans un conteneur
`docker compose restart [service]`	Redémarrer un service
`docker compose pull`	Télécharger les nouvelles versions des images
`docker compose stop`	Arrêter sans supprimer les conteneurs
`docker compose start`	Redémarrer des conteneurs arrêtés