**🌊 1. À quoi sert Docker Compose ?**

Quand tu fais du développement web, ton projet a souvent **plusieurs parties** :

* ton **serveur web** (ici : Express)
* une **base de données** (ou Redis)
* parfois d’autres services (frontend, API, etc.)

👉 **Docker Compose** sert à **démarrer et gérer tous ces conteneurs ensemble**, facilement, avec **un seul fichier de configuration**.

Ce fichier s’appelle docker-compose.yml.

**🧱 2. Le principe de base**

Imagine que chaque service (ex: ton serveur Express, ton Redis, ton Postgres, etc.) soit comme une **petite boîte indépendante** (un conteneur Docker).

Le fichier docker-compose.yml dit à Docker :

“Crée ces boîtes, connecte-les entre elles, et fais-les communiquer.”

**📄 3. Structure d’un fichier docker-compose.yml**

Regarde cette structure simple :

version: "3.9"

services:

web:

...

redis:

...

volumes:

...

* version: la version du format Docker Compose (3.9 = moderne).
* services: la liste de tes applications (tes conteneurs).
* volumes: pour stocker des données qui doivent persister (même après redémarrage).

**🧩 4. Construisons ton fichier pas à pas**

Tu veux deux services :

1. web → ton serveur **Express**
2. redis → ta base **Redis**

**Étape 1 : le squelette**

version: "3.9"

services:

web:

redis:

**Étape 2 : le service web**

Ton app Express est ton code, donc on veut **la construire depuis ton Dockerfile** :

web:

build: . # On construit à partir du Dockerfile dans le dossier actuel

ports:

- "8000:8000" # On ouvre le port 8000 (pour accéder dans le navigateur)

**Étape 3 : on connecte web à redis**

Ton app Express doit parler à Redis via une variable d’environnement (REDIS\_URL).

On l’ajoute dans environment :

environment:

- REDIS\_URL=redis://redis:6379

⚠️ Ici, “redis” dans l’URL (redis://redis:6379) n’est pas magique — c’est le **nom du service** défini plus bas. Docker Compose crée automatiquement un petit réseau entre les services.  
Donc ton serveur Express pourra contacter Redis simplement avec le nom redis.

**Étape 4 : indiquer que web dépend de redis**

On veut que Redis démarre avant web :

depends\_on:

- redis

**Étape 5 : le service Redis**

Redis n’est **pas ton code** : tu peux directement utiliser l’image officielle.

redis:

image: redis:7-alpine # On utilise une image toute prête

volumes:

- redis-data:/data # On garde les données (persistantes)

**Étape 6 : le volume**

Et enfin, on déclare ce volume en bas du fichier :

volumes:

redis-data:

**✅ 7. Résultat final (complet et propre)**

Voici ton fichier docker-compose.yml **final**, commenté :

version: "3.9"

services:

web:

build: . # Construit ton image à partir du Dockerfile

ports:

- "8000:8000" # Ouvre le port 8000

environment:

- REDIS\_URL=redis://redis:6379 # Variable d'environnement pour Redis

depends\_on:

- redis # Redis doit démarrer avant le serveur web

restart: unless-stopped # Redémarre si le conteneur plante

redis:

image: redis:7-alpine # Image officielle Redis

volumes:

- redis-data:/data # Stocke les données Redis

restart: unless-stopped # Redémarre si planté

volumes:

redis-data: # Volume nommé (données persistantes)

**🚀 8. Comment l’utiliser**

1. **Construis et démarre tes conteneurs :**
2. docker compose up -d
3. **Ouvre ton navigateur :**  
   👉 <http://localhost:8000>
4. **Actualise plusieurs fois :**  
   Le nombre de visites augmente (Redis garde la valeur).
5. **Teste la persistance :**
6. docker compose down
7. docker compose up -d

➜ Le compteur reste le même grâce au volume redis-data.

**🧠 9. À retenir**

| **Élément** | **Rôle** |
| --- | --- |
| version | Indique la version du format |
| services | Liste des conteneurs à lancer |
| build | Construit à partir du Dockerfile |
| image | Utilise une image toute faite |
| ports | Rend ton app accessible depuis ton ordi |
| environment | Variables d’environnement |
| depends\_on | Démarrage dans le bon ordre |
| volumes | Sauvegarde des données même après down |