docker-compose.yml の解説

この `docker-compose.yml` は、**Webアプリケーション（`web` サービス）と Postgres データベース（`db` サービス）をまとめて起動・停止するための設定ファイル**です。
さらに、それらをつなぐ **ネットワーク** と、DB のデータを永続化する **ボリューム** を定義しています。

`services:` セクション

```yaml
services:
  # Webサーバーの設定
  web:
    build: ./app
    ports:
      - "8000:8000"
    environment:
      - POSTGRES_USER=user
      - POSTGRES_PASSWORD=pass
      - POSTGRES_DB=mydb
    depends_on:
      - db
    networks:
      - app-network

  # データベースの設定
  db:
    image: postgres:15
    environment:
      - POSTGRES_USER=user
      - POSTGRES_PASSWORD=pass
      - POSTGRES_DB=mydb
    volumes:
      - db-data:/var/lib/postgresql/data  # データの永続化
    networks:
      - app-network
```

`web` サービス（Webアプリ）

- **`web:`**
  Web アプリケーション用コンテナの定義です。

- **`build: ./app`**
  - `./app` ディレクトリにある `Dockerfile` を使って **イメージをビルド** します。
  - `docker-compose up` または `docker compose up` を実行したときに、この設定に基づいてイメージが作られます（キャッシュがあるため毎回フルビルドではありません）。

- **`ports: - "8000:8000"`**
  - ホスト:コンテナ = `8000:8000` のポートマッピングです。

  - ホストマシンの `http://localhost:8000` へのアクセスが、コンテナ内のポート `8000` に届きます。
  - つまり、**ブラウザから Web アプリにアクセスする窓口**を開いています。

- **`environment:`（環境変数）**

```
environment:
  - POSTGRES_USER=user
  - POSTGRES_PASSWORD=pass
  - POSTGRES_DB=mydb
```

  - コンテナ内に設定される **環境変数** です。
  - おそらく `app/main.py` などのアプリコードが、これらの環境変数を読んで DB に接続する想定です。
  - 値は `db` サービス側とそろえておく必要があります（このファイルでは両方 `user/pass/mydb` で一致させています）。

- **`depends_on:`（起動順の制御）**

```
depends_on:
  - db
```

  - `web` コンテナを起動するとき、**先に `db` コンテナを立ち上げてから `web` を起動する**ように指示する設定です。
  - ただし「DB が完全に起動し、接続可能になるまで待つ」ことまでは保証しません（単に起動順を制御するだけです）。
    実際の本番環境では、アプリ側で「DB 接続をリトライする」仕組みを入れることが多いです。

- **`networks: - app-network`**

  - 後ろで定義している `app-network` というネットワークに参加させています。
  - これにより、`web` から `db` へ `db:5432` のように **サービス名でアクセス**できます（Docker の内部 DNS による名前解決）。

`db` サービス（Postgres データベース）

- **`db:`**
  データベース用コンテナの定義です。

- **`image: postgres:15`**

  - Docker Hub 上の公式イメージ `postgres` の **バージョン 15** を使う指定です。
  - `docker pull postgres:15` で取得できるイメージと同じものが利用されます。

- **`environment:`（Postgres 用の特別な環境変数）**

```
```

```
environment:
  - POSTGRES_USER=user
  - POSTGRES_PASSWORD=pass
  - POSTGRES_DB=mydb
```

  - Postgres 公式イメージが理解している特別な環境変数です。
    - `POSTGRES_USER`：初期ユーザー名
    - `POSTGRES_PASSWORD`：そのユーザーのパスワード
    - `POSTGRES_DB`：初期に作成するデータベース名
  - この設定により、コンテナ起動時に `user`/`pass` で `mydb` に接続できる状態が自動で用意されます。

- **`volumes:`（データの永続化）**

```
volumes:
  - db-data:/var/lib/postgresql/data
```

  - 左側 `db-data` は **ボリューム名**（後ろの `volumes:` セクションで定義）。
  - 右側 `/var/lib/postgresql/data` は **コンテナ内で Postgres がデータを保存するディレクトリ**です。
  - これによって、
    - コンテナを削除しても、DB のデータはボリューム `db-data` に残る
    - `docker-compose down -v`（`-v` を付けてボリュームも削除）しない限り、データは消えない
  - **「DB のデータの永続化」を実現する典型的な書き方**です。

- **`networks: - app-network`**

  - `web` と同じネットワーク `app-network` に所属させることで、
    - `web` コンテナからホスト名 `db` でこの DB コンテナにアクセスできる
    - 例: `postgresql://user:pass@db:5432/mydb`

`networks:` セクション

```
networks:
  app-network:
    driver: bridge
```

- **`networks:`**

  - カスタムネットワークの定義です。

- **`app-network:`**

  - このファイル内で使っているネットワークの名前です。
  - `web` と `db` は両方ともこのネットワークに参加しています。

- `driver: bridge`
  - Docker の **ブリッジネットワーク** を使う指定です。
  - ブリッジネットワークは、**同じネットワーク上のコンテナ同士が名前解決できて通信できる**、一番よく使うモードです。
  - 省略した場合もデフォルトで `bridge` になりますが、ここでは明示的に書いています。

`volumes:` セクション

```
volumes:
  db-data:
```

- `volumes:`
  - 名前付きボリュームの定義です。
- `db-data:`
  - ボリューム名のみを書いていて、オプションは何も指定していません。
  - これで「Docker によって管理される専用ストレージ」が 1 つ作られます。
  - `db` サービスで `/var/lib/postgresql/data` にマウントされており、DB のデータがここに保存されます。
  - `docker volume ls` で一覧に表示され、`docker volume inspect db-data` で詳細を確認できます。

`docker-compose up` したときの挙動

この `docker-compose.yml` があるディレクトリで次を実行すると:

```
docker-compose up
# または
docker compose up
```

以下のような流れでコンテナが立ち上がります。

1. **`db` コンテナ起動**
   - イメージ `postgres:15` を取得（ローカルに無ければ pull）。
   - 環境変数に基づいて、ユーザー `user` / パスワード `pass` / DB `mydb` を作成。
   - データはボリューム `db-data` に保存される。
2. **`web` コンテナビルド & 起動**
   - `./app` の `Dockerfile` からイメージをビルド。
   - 環境変数 `POSTGRES_USER`, `POSTGRES_PASSWORD`, `POSTGRES_DB` を設定。
   - `app-network` に参加。

    - ホストの `8000` ポートがコンテナの `8000` に紐づく。

3. **アクセス**
    - ブラウザで `http://localhost:8000` にアクセス → `web` コンテナ内のアプリが応答。
    - アプリは内部で `db` へ接続（`db:5432` など）し、Postgres を利用する。

---

compose 学習用の観点からのポイント整理

- **サービス間通信**
    - `services.web.depends_on: db` と `networks: app-network` によって、
        - 起動順: `db` → `web`
        - 通信: `web` から `db` というサービス名で DB へ接続できる
- **イメージの種類**
    - `web` は `build:` を使って「自前の Dockerfile からビルド」。
    - `db` は `image:` で「既存の公式イメージをそのまま利用」。
- **状態を持つコンテナと永続化**
    - Postgres は状態（データ）を持つコンテナなので、`volumes:` を使って永続化している。
    - コンテナを消してもデータが残る、というのがボリューム利用の大きな利点。
- **環境変数の役割**
    - DB 接続情報を `environment:` でまとめて管理しておくことで、
        - コードを変更せずに設定だけで接続先を変えられる
        - `web` と `db` で設定値をそろえておきやすい

さらに学ぶためのヒント

- `./app/Dockerfile` や `./app/main.py` を読み、環境変数をどう使って DB 接続しているかを確認すると、
  「compose の設定」と「アプリケーションコード」がどう連携しているか がよりよく理解できます。
- `depends_on` にヘルスチェックを組み合わせて、「DB が完全に起動してから Web を起動する」ような高度な書き方もあります。
- 本番環境では、`.env` ファイルを使ってパスワードを管理したり、ボリュームのバックアップ戦略を考えたりするのが一般的です。