Struttura del Progetto

Panoramica

Il progetto utilizza Docker e Docker Compose per orchestrare tre servizi principali:

1. **database**: un contenitore MySQL personalizzato (con script di inizializzazione e backup)
2. **web-server**: un contenitore Ubuntu con Apache2 + PHP, che serve frontend e backend
3. **phpmyadmin**: interfaccia web per amministrare il database MySQL

I file di configurazione chiave sono:

- Dockerfile
- .env
- docker-compose.yml
- apache-conf/000-default.conf
- apache-conf/ports.conf
- frontend/.htaccess
- backend/ .htaccess

Dockerfile

```
FROM ubuntu:latest

ENV DEBIAN_FRONTEND=noninteractive
ENV TZ=Europe/Rome

RUN apt-get update && apt-get install -y \
   apache2 \
   php \
   libapache2-mod-php \
   php-mysql \
   python3 \
   python3-pip

RUN mkdir /var/www/api

RUN a2enmod rewrite
# Attiva il modulo mod_rewrite di Apache, necessario per gestire URL
# rewriting sia nel frontend che nel backend

COPY ./apache-conf/000-default.conf /etc/apache2/sites-available/000-default.conf
COPY ./apache-conf/ports.conf /etc/apache2/ports.conf

EXPOSE 80
EXPOSE 8080

CMD ["apache2ctl", "-D", "FOREGROUND"]
```

Il Dockerfile definisce l’immagine di base per il servizio web (web-server), ossia il container che ospita Apache, PHP. L’obiettivo è creare un’immagine che contenga:

- Un sistema operativo base (Ubuntu).

- Apache configurato con i moduli necessari per PHP e rewrite.
- PHP e le estensioni per connettersi a MySQL.
- Copia di file di configurazione di Apache.
- Esposizione delle porte e comando di avvio di Apache in foreground.

---

docker-compose.yaml

```
services:
  web-server:
    container_name: web-server
    build:
      context: .
      dockerfile: Dockerfile
    volumes:
      - ./frontend:/var/www/html
      - ./backend:/var/www/api
    ports:
      - "80:80"
      - "8080:8080"
    environment:
      - MYSQL_HOST=database
      - MYSQL_USER=${MYSQL_USER}
      - MYSQL_PASSWORD=${MYSQL_PASSWORD}
      - MYSQL_DATABASE=${MYSQL_DATABASE}
    depends_on:
      - database
    restart: always

  database:
    container_name: database
    image: mysql:latest
    environment:
      MYSQL_DATABASE: ${MYSQL_DATABASE}
      MYSQL_USER: ${MYSQL_USER}
      MYSQL_PASSWORD: ${MYSQL_PASSWORD}
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
      MYSQL_ALLOW_EMPTY_PASSWORD: "no"
    volumes:
      - db_data:/var/lib/mysql

      # Per ricreare un database vuoto
      # - ./db/init.sql:/docker-entrypoint-initdb.d/init.sql

      # Per inizializzare il database con un file SQL di backup
      - ./db/backup.sql:/docker-entrypoint-initdb.d/init.sql
    ports:
      - "${MYSQL_PORT}:3306"
    restart: always

  phpmyadmin:
    container_name: phpmyadmin
    image: phpmyadmin/phpmyadmin
    restart: always
```

```
    environment:
      - PMA_HOST=database
      - PMA_USER=root
      - PMA_PASSWORD=${MYSQL_ROOT_PASSWORD}
      - PMA_PORT=3306
    ports:
      - "8082:80"

volumes:
    db_data:
```

Definisce e avvia i servizi principali (web-server, database, phpMyAdmin) usando la rete predefinita di Docker Compose. Monta volumi per codice e dati, imposta variabili d'ambiente e dipendenze, e pubblica porte sul host.

Sezioni principali

1. **Servizio web-server**
   - **container_name: web-server**: nome del container.
   - **build**: utilizza il `Dockerfile` per costruire l'immagine
   - **volumes**:
     - `./frontend:/var/www/html`: monta il codice frontend nella directory servita da Apache.
     - `./backend:/var/www/api`: monta il codice PHP backend (API) nella directory servita da Apache.
   - **ports**:
     - "`80:80`": espone la porta 80 del container sulla porta 80 dell'host, per il frontend.
     - "`8080:8080`": espone la porta 8080 del container sulla porta 8080 dell'host, per le chiamate API.
   - **environment**:
     - `MYSQL_HOST`=database: istruisce il backend a connettersi al servizio database tramite il nome di servizio Docker Compose.
     - `MYSQL_USER`, `PASSWORD` e `DATABASE` letti dal file .env, fornendo credenziali al codice PHP.
   - **restart: always**: assicura il riavvio automatico in caso di crash o fermo del container.
2. **Servizio database**
   - **volumes**:
     - `db_data:/var/lib/mysql`: volume Docker nominato per persistere i dati del database, così i dati sopravvivono a riavvii o ricreazioni del container.
     - `./db/backup.sql:/docker-entrypoint-initdb.d/init.sql`: su prima esecuzione (database vuoto) MySQL importerà questo file, creando schema/dati iniziali.
   - **ports**:
     - "${MYSQL_PORT}:3306": mappa la porta interna 3306 del container sulla porta specificata in .env (3306) dell'host, consentendo l'accesso al database
3. **Servizio phpmyadmin**
   - Fornisce UI web per gestire MySQL.
   - **environment**:
     - `PMA_HOST`=database: punta al servizio database.
     - `PMA_USER=root`, `PMA_PASSWORD`: credenziali per connettersi come root.
   - **ports**:
     - "`8082:80`": espone l'interfaccia web di phpMyAdmin sulla porta 8082 dell'host.

Collaborazione tra i servizi

- **web-server**: Apache serve il frontend su porta 80 e il backend API su porta 8080; il codice PHP utilizza MYSQL_HOST=database per connettersi al container MySQL.
- **database**: MySQL persiste sul volume db_data; si inizializza con il file .sql se non esiste database. Il servizio è accessibile internamente dal web-server e tramite la porta mappata
- **phpmyadmin**: si connette a MySQL tramite hostname database, rendendo possibile amministrare il DB da http://trekkigram.com:8082.

- **Esposizione porte**:
  - Frontend accessibile su `http://trekkigram.com` (porta 80).
  - Backend API su `http://trekkigram.com:8080.`
  - phpMyAdmin su `http://trekkigram.com:8082.`

Configurazioni di Apache

000-default.conf

```
<VirtualHost *:80>
   ServerName trekkigram.com

   DocumentRoot /var/www/html
    <Directory /var/www/html>
      AllowOverride All
      Require all granted

   </Directory>
   ErrorLog ${APACHE_LOG_DIR}/error_80.log
   CustomLog ${APACHE_LOG_DIR}/access_80.log combined
</VirtualHost>

<VirtualHost *:8080>
   ServerName trekkigram.com

   DocumentRoot /var/www/api
   <Directory /var/www/api>
      Options Indexes FollowSymLinks
      AllowOverride All
      Require all granted
   </Directory>
   ErrorLog ${APACHE_LOG_DIR}/error_8080.log
   CustomLog ${APACHE_LOG_DIR}/access_8080.log combined
</VirtualHost>
```

Imposta due VirtualHost in Apache:

1. *VirtualHost :80*
   - **DocumentRoot /var/www/html**: serve i file del frontend montati dal container.
   - **<Directory /var/www/html>**: consente override (AllowOverride All) affinché .htaccess nel frontend abbia effetto, e autorizza accessi.
   - **ErrorLog / CustomLog**: file di log su porte 80, separati per facilità di debug.

2. *VirtualHost :8080*
    - Serve su porta 8080 lo **schema API**:
    - **DocumentRoot /var/www/api**: directory del backend PHP.
    - Simile direttiva <Directory> con AllowOverride All per abilitare rewrite tramite .htaccess.
    - Log separati per errori e accessi su porta 8080.

Un **Virtual Host** in Apache è una configurazione che permette di **servire siti web diversi dallo stesso server**, distinguendoli in base al **nome di dominio** o alla **porta** (es. :80, :8080);
Quindi
separano il frontend (porta 80) e il backend API (porta 8080) pur girando sullo stesso container/server.

ports.conf

```
Listen 80
Listen 8080
```

- **Listen 80** e **Listen 8080**: istruisce Apache ad aprire entrambe le porte. È necessario per far vedere entrambi i VirtualHost.

Relazione con Dockerfile e docker-compose

- Nel Dockerfile si copiano questi file di configurazione in /etc/apache2/. Quando il container parte e avvia Apache, questi file stabiliscono:
    - Su quali porte ascoltare.
    - Quali directory servire per frontend e backend.
    - Attivano URL rewriting tramite .htaccess.
- In docker-compose, le porte del container vengono esposte e mappate su porte dell'host, rendendo il servizio accessibile

---

File .htaccess

Frontend (.htaccess in ./frontend)

```
RewriteEngine On
RewriteBase /
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule . /index.html [L]
```

Questo file abilita il client-side routing; Questa abilitazione è importante in app come Trekkigram in quanto è una `SPA - Single Page Application` ovvero in cui la navigazione viene gestita tramite javascript.

Funzionamento: se viene aperto un link (es: trekkigram.com/pagina) Apache controlla se esiste un file o una cartella chiamato *pagina*; se non esiste si viene reindirizzati al file `index.html` e verrà lasciato a Javascript il compito di decidere dove portare l'utente (→ */pagina*)

Backend (.htaccess in ./backend)

```
RewriteEngine on
RewriteRule . index.php
```

Questo file permette di gestire tutte le richieste a una API dell'applicazione per reindirizzarle tutte al file `index.php` ; Il file index.php poi legge l'url arrivato, comprende il tipo di richiesta e invia la richiesta all'endpoint corretto.

File .env

```
MYSQL_DATABASE=TrekkigramDB
MYSQL_USER=trekking_user
MYSQL_PASSWORD=trekking_pass
MYSQL_ROOT_PASSWORD=root_password
MYSQL_PORT=3306
```

Ruolo generale

- Contiene le variabili d’ambiente necessarie per configurare il database MySQL e per collegare il backend PHP.

Relazioni e Flusso Complessivo

1. **Fase di build**:
   - Il Dockerfile crea l’immagine del web server con Apache, PHP, moduli e configurazioni personalizzate.
2. **Avvio dei servizi**:
   - Docker Compose accende tutti i container:
     - MySQL si avvia e importa il database (se vuoto).
     - Apache parte e monta i file del frontend e backend.
     - phpMyAdmin si collega al database per la gestione.
3. **Flusso di una richiesta**:
   - L’utente apre http://trekkigram.com → Apache serve index.html dal frontend.
   - Il JavaScript del frontend gestisce la navigazione e chiama le API su http://trekkigram.com:8080.
   - Le richieste API arrivano su porta 8080 → Apache le manda tutte a index.php → PHP esegue la logica e interroga MySQL.