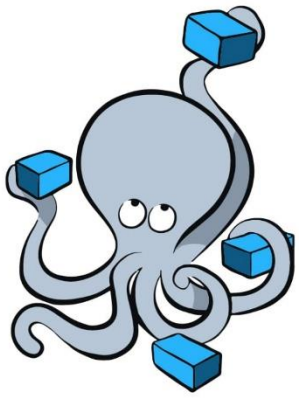
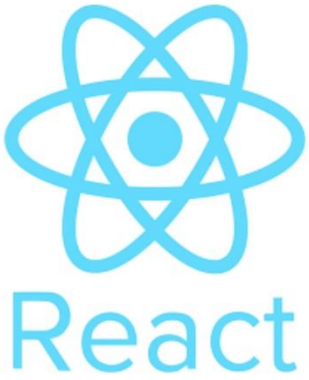


docker
Compose



Dockerizzazione di Web App con React-Vite e PHP

SISTEMI DI VIRTUALIZZAZIONE (A.A. 2023/2024)

Manuel R. Sciuto (547488) | 26/09/2024

Introduzione

Nella creazione delle moderne web app gli sviluppatori sono sempre alla ricerca di nuovi strumenti per semplificare e ottimizzare il processo di sviluppo e distribuzione, ***Docker*** è uno dei principali.

Docker è uno strumento sviluppato per rendere più semplice la creazione, lo sviluppo e l'utilizzo delle applicazioni utilizzando le ***immagini*** e i ***container***. Un'***immagine*** è un pacchetto che contiene file, eseguibili, librerie e configurazioni necessarie per far funzionare un ***container***, mentre un ***container*** è un processo che ci permette di utilizzare i componenti dell'app in ambienti sicuri e isolati dal resto, la natura di questi container li rende in grado di funzionare ovunque; infatti, un container funzionerà nello stesso modo sia sul computer per lo sviluppo sia in un data center o nel cloud.

Realizzazione

Per la realizzazione di questo progetto è stato utilizzato ***Docker Compose***, uno strumento sviluppato per gestire applicazione multi-container. Esso ci permette di creare un file .yml e di definire i servizi della nostra app al suo interno.

Nel caso di questo progetto il file docker-compose presenta 5 servizi principali:

1. **Frontend**: Il servizio frontend rappresenta la parte visibile all'utente dell'applicazione e viene generato a partire dal proprio Dockerfile che hosta la build del frontend con Nginx. Esso ha altre 2 istanze frontend-2 e frontend-3, generate dallo stesso Dockerfile ma con variabili d'ambiente diverse per poterle distinguere, ciò è stato fatto per permettere al servizio Nginx di fare da reverse proxy per poter dividere equamente il traffico sui 3 frontend.

2. **Backend**: Il servizio backend viene costruito a partire da un'immagine ed è composto da una serie di file PHP che mettono in comunicazione il frontend e il database MySQL ricevendo le chiamate dalla API Fetch di React.

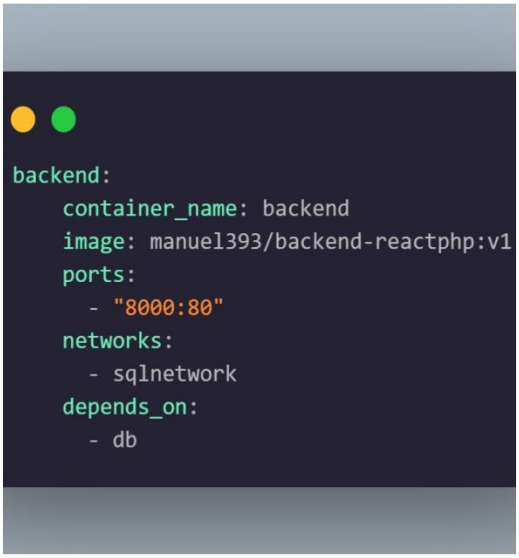
3. **Nginx:** Come specificato parlando del frontend la funzione di questo servizio è di fungere da reverse proxy, comunicando coi forntend attraverso il network "webnetwork", anche se a questo stadio del progetto tale compito non viene svolto in maniera corretta.

4. **DB:** Questo servizio permette al backend di accedere ai dati salvati nel volume sqldb attraverso sqlnetwork, esso presenta anche come variabili la root password e il nome del db che contiene i dati riguardanti il progetto.

5. **PhpMyAdmin:** Questo servizio permette l'accesso a un'interfaccia per la gestione del DB, comunica con esso attraverso sqlnetwork, e presenta delle variabili per la connessione con MySQL e l'autenticazione degli utenti.

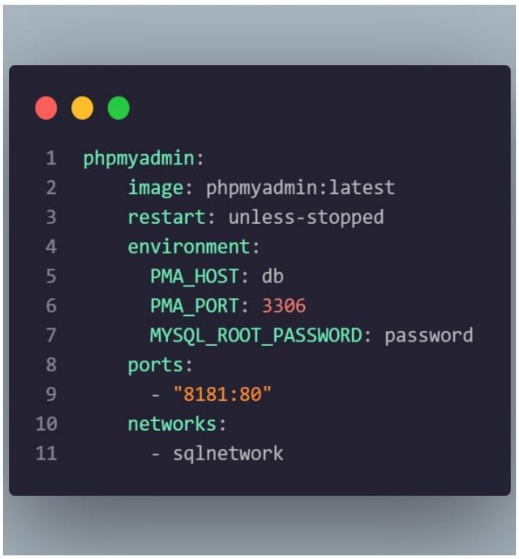
Di seguito si può osservare uno snippet del file docker-compose.

```yaml
frontend-1:
    container_name: frontend-1
    build:
      context: React/
      args:
        REACT_APP_REPLICA_NUM: 1
    expose:
      - "80"
    ports:
      - "3001:80"
    environment:
      - NUM_FRONTEND=1
    depends_on:
      - backend
    networks:
      - webnetwork
```



```yaml
backend:
    container_name: backend
    image: manuel393/backend-reactphp:v1
    ports:
      - "8000:80"
    networks:
      - sqlnetwork
    depends_on:
      - db
```

```yaml
db:
    image: mysql:latest
    restart: unless-stopped
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: react-php
    volumes:
      - sqldb:/var/lib/mysql
    ports:
      - "3306:3306"
    networks:
      - sqlnetwork
```



```yaml
phpmyadmin:
    image: phpmyadmin:latest
    restart: unless-stopped
    environment:
      PMA_HOST: db
      PMA_PORT: 3306
      MYSQL_ROOT_PASSWORD: password
    ports:
      - "8181:80"
    networks:
      - sqlnetwork
```

Come si può osservare questi servizi sono generati in 3 modi differenti, il backend viene generato a partire da un'immagine personalizzata che contiene tutte le dipendenze, gli import e le configurazioni necessarie per PHP, il frontend viene costruito a partire da un Dockerfile che utilizza 2 immagini standard di Node.js e Nginx per creare il servizio, mentre db e PhpMyAdmin vengono generati da immagini standard di cui viene presa l'ultima versione su DockerHub.

Differenze nello sviluppo con e senza Docker

```php
<?php
require 'vendor/autoload.php';
header("Access-Control-Allow-Origin: *");
header("Access-Control-Allow-Methods: GET, POST, OPTIONS");
header("Access-Control-Allow-Headers: Content-Type");
header("Content-Type: application/json");

if ($_SERVER['REQUEST_METHOD'] === 'OPTIONS') {
    exit(0);
}

// PER DOCKER $dbhost = "db", PER LOCAL DEV $dbhost = "localhost";
$dbhost = "localhost";
$dbport = "3306";
$dbuser = "root";
// PER DOCKER $dbpass = "password", PER LOCAL DEV $dbpass = "";
$dbpass = "";
$dbname = "react-php";

$conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname, $dbport);
```

Come si può vedere in questo snippet, anche la connessione al DB attraverso PHP differisce se si utilizza Docker, la variabile $dbhost assume valori diversi: “db” se si è in un ambiente Docker dato che il nome del servizio corrispondente del docker compose è db mentre se si è in un ambiente di sviluppo locale si utilizza localhost.

Variano anche le password, quella per Docker sarà quella specificata nella variabile d’ambiente MYSQL_ROOT_PASSWORD oppure se non si vuole una root password si può utilizzare MYSQL_NO_ROOT_PASSWORD = “true”.

Nel caso del frontend React invece possono esserci più problemi, il più evidente agli occhi dello sviluppatore è la mancanza di “real-time changes” quando viene modificato il codice sorgente dell’app. Una scappatoia a questo problema è l’utilizzo di un Docker Volume dove copiamo la directory contenente l’app, ciò permette di sviluppare e testare l’app in un ambiente isolato senza dover di nuovo costruire da zero l’immagine.

Conclusione

Nonostante l'iniziale complessità nella configurazione, Docker offre vantaggi significativi che possono migliorare notevolmente il processo di sviluppo e distribuzione delle applicazioni. Le comodità fornite da Docker, come la possibilità di creare ambienti di sviluppo isolati e replicabili, sono inestimabili, specialmente quando si lavora su progetti che devono essere eseguiti su dispositivi diversi o in ambienti non compatibili.

La sicurezza, l'isolamento e la scalabilità sono tre dei principali punti di forza di Docker. L'isolamento dei container garantisce che le applicazioni non interferiscano l'una con l'altra, mentre la scalabilità consente agli sviluppatori di gestire facilmente il carico di lavoro aumentando o diminuendo il numero di istanze dei container in base alle esigenze. Questi fattori rendono Docker uno strumento stabile e affidabile, capace di supportare applicazioni moderne in continua evoluzione.

Docker Compose, invece, permette di creare applicazioni multi-container e di far comunicare essi tra di loro senza problemi, di scalare virtualmente all'infinito le applicazioni aggiungendo sempre più servizi e di poter effettuare modifiche ai servizi senza la necessità di dover alterare lo stato dell'intera applicazione.