Dockerizzazione del progetto di basi di dati

Corso di Virtualizzazione e Integrazione di sistemi

Ingegneria e scienze informatiche
Università di Bologna

A.A. 2022-2023

Nome	Email	Matricola
Maicol Battistini	maicol.battistini@studio.unibo.it	0000988616

18 agosto 2023

Indice

1 | Introduzione

Il progetto si pone l'obiettivo di dockerizzare il progetto di basi di dati.

1.1 | Introduzione al progetto di basi di dati

Il progetto mira alla realizzazione di un software che faciliti la gestione di una catena di negozi che vende ciambelle. In particolare si vuole realizzare la possibilità di gestire ordini ricevuti tramite sito web dai clienti, che possono scegliere tra consegna a domicilio o takeaway, e dai rivenditori terzi, tenendo uno storico delle vendite. Il sito web, compresa l'effettuazione di ordini da parte degli utenti, con relativo sistema di accesso e registrazione degli utenti è già stata sviluppata in precedenza da terzi. Inoltre saranno gestiti i dipendenti e manager di ogni negozio, con i dati a loro relativi, gli ingredienti e le ciambelle disponibili in ogni punto vendita

1.1.1 | Architettura del software

L'applicazione è stata realizzata in PHP e il framework Laravel, utilizzando anche Typescript e MithrilJS (framework per gestire le views). L'applicazione è quindi così divisa:

- Backend: Laravel + PHP. Librerie di rilievo:
  - Laravel Eloquent: ORM per le query lato backend
  - Restify: API per fornire i dati di ogni Model al frontend seguendo lo standard JSONAPI
  - InertiaJS: Ponte che collega backend e frontend per quanto riguarda la gestione del routing e delle views
- Frontend: Typescript + Mithril. Altre librerie di rilievo:
  - Material Web: componenti web utilizzati per costruire l'interfaccia grafica
  - Coloquent: Interfaccia ORM-like per comunicare con l'API del backend

Si utilizza anche SCSS per i fogli di stile.

2 | Dockerizzazione

Sono stati creati due possibiilità di utilizzo del progetto attraverso Docker: ambiente di produzione e ambiente di sviluppo, così da dare la possibilità al developer del software di utilizzare un container simile a quello di produzione (più leggero) durante lo sviluppo del software

2.1 | Struttura e analisi

Analizziamo ora ogni file riguardante la dockerizzazione del progetto:

2.1.1 | .dockerignore

```
node_modules
vendor
.env
```

Questo file elenca i file e le directory da escludere quando si effettua la copia del progetto nel container tramite il comando `COPY`. In particolare, si vogliono escludere:

- La directory `node_modules` che contiene le dipendenze del frontend dell'app (principalmente Javascript e CSS installate tramite npm)
- La directory `vendor` che contiene le dipendenze PHP del backend dell'app installate tramite composer
- Il file `.env` che contiene le variabili di configurazione utilizzate dall'app

2.1.2 | Dockerfile

```
# syntax = edrevo/dockerfile-plus
# Dockerfile for development environment
FROM litespeedtech/openlitespeed:1.7.18-lsphp81

INCLUDE+ Dockerfile.common
INCLUDE+ Dockerfile.entrypoint
```

Il Dockerfile utilizzato per l'ambiente di sviluppo. Utilizza una sintassi aggiuntiva per i Dockerfile, ottenuta attraverso BuildKit (attivato di default da Docker v23+), per poter "modularizzare" il Dockerfile e permettere così di importare un Dockerfile in un altro:

```
# syntax = edrevo/dockerfile-plus
```

In particolare, il commento indica a BuildKit di utilizzare l'estensione della sintassi di Docker fornita dal repo Github `edrevo/dockerfile-plus` (che aggiunge la nuova istruzione INCLUDE+).
Successivamente, si parte da una immagine di OpenLiteSpeed (un webserver alternativo a Apache/Nginx e più performante), basata su Ubuntu 22.04, contenente PHP 8.1 (lsphp81 è un pacchetto che include una versione ottimizzata di php per litespeed).
Infine, si includono i due dockerfile common e entrypoint.

2.1.3 | Dockerfile.common

Dockerfile parziale che effettua operazioni comuni ai due ambienti di sviluppo:

- Accetta come build argument `PHP_VERSION` con cui è possibile impostare la versione della CLI di PHP da installare e la imposta come variabile d'ambiente
- Effettua un aggiornamento dei pacchetti, aggiunge i ppa di fish e PHP e installa i pacchetti necessari
- Cambia la shell predefinita in fish (che preferisco rispetto alla bash)
- Installa composer (il gestore delle dipendenze di PHP) e lo aggiunge al path
- Scarica uno script per permettere un editing semplice del file .env

2.1.4 | Dockerfile.entrypoint

```
# Esegue eventuali script di avvio o di configurazione necessari
ADD https://raw.githubusercontent.com/litespeedtech/ols-dockerfiles/master/template/entrypoint.sh /
RUN chmod +x /ols-entrypoint.sh
COPY ./entrypoint.sh /entrypoint.sh
RUN chmod +x /entrypoint.sh
ENTRYPOINT ["/entrypoint.sh"]
```

Dockerfile parziale che effettua le operazioni finali della build, comuni ai due ambienti di sviluppo:

- Scarica l'entrypoint di OpenLiteSpeed per utilizzarlo poi in entrypoint.sh
- Copia, imposta i permessi e imposta come entrypoint dell'immagine il file entrypoint.sh

2.1.5 | Dockerfile.prod

Il Dockerfile utilizzato per l'ambiente di produzione. Oltre al dockerfile dell'ambiente di sviluppo, questo dockerfile si occupa di:

- Copiare l'app nella directory utilizzata dal webserver
- Copiare la configurazione del webserver all'interno dell'immagine nella sua directory
- Impostare la directory dell'app all'interno dell'immagine come directory di lavoro (in questo modo la shell verrà sempre aperta di default in questa directory)
- Impostare la properietà di ogni file e directory dell'app al webserver

- Installare le dipendenze di Composer (evitando quelle di sviluppo)
- Creare un nuovo file .env se non già esistente
- Generare la chiave di crittografia dell'app (richiesta da Laravel)
- Installare Node, pnpm e le dipendenze JS/CSS
- Imposta il file .env per indicare all'app di essere in ambiente di produzione

2.1.6 | docker-compose.prod.yml

Come per il Dockerfile, anche il compose file è stato, in un certo senso, "modularizzato". Questo file, oltre a creare un volume per il database, elenca tutti i servizi estendendoli dal file docker-compose.shared.yml, oltre ad indicare anche eventuali particolarità del servizio come i volume e le dipendenze

2.1.7 | docker-compose.shared.yml

Il file "comune" a tutti e due gli ambienti. Vengono elencati i servizi utilizzati:

- mysql, il database che utilizza MariaDB
- PHPMyAdmin, applicazione web per accedere al DB
- app: Servizio che punta al Dockerfile e rappresenta DonutShopApp

2.1.8 | docker-compose.yml

Compose file per l'ambiente di sviluppo. La differenza rispetto a quello dell'ambiente di produzione è che vengono specificati i volume per il servizio app (infatti non viene effettuata nessuna copia nell'immagine per l'ambiente di sviluppo)

2.1.9 | entrypoint.sh

```
#!/bin/bash
cd /var/www/vhosts/localhost/html || exit
dotenv set DB_HOST="${MYSQL_HOST}" DB_USERNAME="${MYSQL_USER}" DB_PASSWORD="${MYSQL_PASSWORD}" DB_

php artisan cache:clear
php artisan config:cache
php artisan migrate
/ols-entrypoint.sh
```

Entrypoint che si occupa di impostare le variabili d'ambiente nel file .env per la connessione al database, ottimizzare la cache della configurazione, migrare il database (creare le tabelle mancanti) e infine avviare l'entrypoint di OpenLiteSpeed (la cui funzione principale è avviare il webserver e mantenerlo in foreground).

3 | Utilizzo

Per avviare tutti i container dell'ambiente di sviluppo utilizzare il comando:

```
docker compose up -d
```

Per invece avviare l'ambiente di produzione specificare il composefile con il comando:

```
docker compose -f docker-compose.prod.yml up -d
```

Nota: è possibile buildare l'immagine forzatamente aggiungendo al comando il flag `--build`

4 | Scalabilità

Per quanto riguarda la scalabilità dell'applicazione, la scalabilità verticale è sicuramente possibile senza troppe difficoltà mentre per la scalabilità orizzontale può essere scalato il container dell'app, ma per quanto riguarda il database, che deve essere condiviso tra le varie istanze dell'app, bisogna ripiegare su un singolo container, un database esterno alle macchine in cui viene deployata l'app oppure instanze del database multiple con un sistema di sincronizzazione per mantenerli aggiornati.