Tutorial

Guestbook: passo a passo verso Docker

L'applicazione *guestbook* è un esempio creato per introdurre Docker in modo progressivo e concreto.

Prerequisiti

- Docker installato
- Docker Compose disponibile
- Conoscenze di base su container e immagini

Passo 1 – Comprendere l'applicazione

Obiettivo didattico

- Separare il *funzionamento applicativo* dall'infrastruttura
- Capire cosa fa l'app prima di containerizzarla

Istruzioni

1. Accedere al repository dell'applicativo: https://github.com/geo-petrini/guestbook
2. Leggere il file README.md

Passo 2 – Build dell'immagine Docker

Obiettivo

- Capire la relazione tra Dockerfile e immagine

Attività

1. Creare l'immagine usando il dockerfile presente nel repository, osservare come docker build cerca automaticamente un dockerfile nella cartella indicata.

```
docker build -t guestbook:latest .
```

2. Provare argomenti aggiuntivi

```
docker build --pull --rm -f "dockerfile" -t "guestbook:latest" .
```

Comando / Flag	Descrizione
docker build	Il comando principale per istruire Docker a costruire un'immagine da un Dockerfile e un contesto specifico.

Comando / Flag	Descrizione
--pull	Forza l'aggiornamento dell'immagine base. Questo flag assicura che Docker scarichi sempre l'ultima versione dell'immagine base specificata nell'istruzione FROM del tuo Dockerfile (ad esempio FROM python:3.13-slim-buster) anche se ne ha già una versione cache localmente.
--rm	Rimuove i container intermedi. Durante il processo di build ogni istruzione del Dockerfile crea un container temporaneo. Questo flag assicura che una volta completato il layer tali container temporanei vengano automaticamente rimossi per evitare di lasciare residui sul sistema.
-f "dockerfile"	Specifica il nome del Dockerfile. Indica che il file delle istruzioni da usare si chiama ""dockerfile"" (con la 'd' minuscola) invece del nome predefinito Dockerfile (con la 'D' maiuscola). Se il tuo file è chiamato Dockerfile questo flag è superfluo ma se hai nomi personalizzati (es. Dockerfile.prod) è essenziale.
-t "guestbook:latest"	Tagga (Etichetta) l'immagine finale. Assegna il nome guestbook e il tag di versione latest all'immagine risultante rendendola facilmente identificabile.
.	Contesto di Build. Indica a Docker che il contesto (la directory che contiene il Dockerfile e tutti i file sorgente necessari) è la directory corrente (.). Tutti i file in questa directory verranno inviati al Docker Daemon.

Passo 3 – Esaminare l'immagine

Obiettivo

- Comprendere come è costruita un'immagine Docker

Comandi

```
docker images
docker image inspect guestbook:latest
docker history guestbook:latest
```

Concetti

- layer
- entrypoint e cmd
- dimensione dell'immagine

Passo 4 – Avvio senza persistenza

```
docker run -d \
  --name guestbook \
  -p 5000:5000 \
  guestbook:latest
```

Obiettivo

- Mostrare la natura *stateless* del container

Attività

- inserire messaggi nel guestbook
- riavviare il container: `docker restart [id_container]`
- verificare se ci sono ancora i dati
- eliminare il container: `docker rm [id_container]`
- avviare di nuovo il servizio
- verificare se ci sono ancora i dati

Concetti

- lifecycle del container
- filesystem effimero
- livello scrivibile

Azione	Livello scrivibile	Dati locali
`docker stop`	preservato	presenti
`docker start`	riutilizzato	presenti
`docker rm`	eliminato	persi
`docker run` (nuovo)	ricreato	assenti

Passo 5 – Avvio con persistenza tramite volume

```
docker run -d \
  --name guestbook \
  -p 5000:5000 \
  -v db:/app/instance \
  guestbook:latest
```

Obiettivo

- Introdurre i volumi come soluzione al problema dello stato

Concetti

- volume Docker
- separazione applicazione/dati
- mountpoint

Passo 6 – Analisi del volume

Attività

```
docker volume ls
docker volume inspect db
docker stop guestbook
docker rm guestbook
docker run ...
```

Verifica

- i dati persistono dopo ricreazione del container

Concetti

- ciclo di vita dei volumi
- indipendenza dal container

Passo 7 – Docker Compose: servizio standalone

```
version: '3.9'
services:
  guestbook:
    image: guestbook:latest
    container_name: guestbook
    volumes:
      - db:/app/instance
    ports:
      - '5000:5000'

volumes:
  db:
```

Obiettivo

- Passare da comandi imperativi a configurazione dichiarativa

Analisi

- volume creato automaticamente
- rete bridge generata da Compose
- naming dei container
- semplificazione operativa

Comandi

```
docker compose up
docker compose up -d
docker compose ps
docker compose down
docker compose down -v
```

Passo 8 – Integrazione con database esterno (MySQL / MariaDB)

Configurazione tramite variabili d'ambiente

```
services:
  web:
    image: guestbook:latest
    container_name: guestbook_web
    ports:
      - "5000:5000"
    environment:
      - DATABASE_HOST=db
      - DATABASE_USER=myuser
      - DATABASE_PASSWORD=mypassword
      - DATABASE_NAME=guestbook_db
    depends_on:
      db:
        condition: service_healthy

  db:
    image: mysql:8.0
    container_name: guestbook_mysql_db
    environment:
      MYSQL_ROOT_PASSWORD: root_password
      MYSQL_USER: myuser
      MYSQL_PASSWORD: mypassword
      MYSQL_DATABASE: guestbook_db
    volumes:
      - db_data:/var/lib/mysql
    ports:
      - "3306:3306"
    healthcheck:
      #NOT WORKING test: ["CMD", 'mysqladmin', 'ping', '-h', '127.0.0.1', '-u',
'root', '--password=$$MYSQL_ROOT_PASSWORD' ]
      test: ["CMD-SHELL", "mysql $$MYSQL_DATABASE -u $$MYSQL_USER -
p$$MYSQL_PASSWORD -h 127.0.0.1 -e 'select * from messages;' 2>&1"]
      interval: 10s
      timeout: 5s
      retries: 5

volumes:
  db_data:
```

Concetti didattici fondamentali

- DNS interno di Compose
- risoluzione dei nomi di servizio
- isolamento di rete
- configurazione a runtime

Verifiche

- accesso al DB via `docker exec`
- presenza dei dati nel database
- persistenza tramite volume DB

Istruzioni

1. collegarsi al container `docker exec -it guestbook_mysql_db /bin/bash`
2. collegarsi al db `mysql -u root -p` seguito da `root_password`
3. verificare che l'utente "myuser" esista

```
mysql> select user from mysql.user;
+------------------+
| user             |
+------------------+
| myuser           |
| root             |
| mysql.infoschema |
| mysql.session    |
| mysql.sys        |
| root             |
+------------------+
```

4. verificare che il db "guestbook_db" esista

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| guestbook_db       |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
```

5. verificare che PeeWee abbia creato le tabelle

```
use guestbook_db;
show tables;

+------------------------+
| Tables_in_guestbook_db |
+------------------------+
```

```
| messages               |
+------------------------+
```

Healthcheck (opzionale ma consigliato)

Perché introdurlo

- distinguere avvio da disponibilità
- rendere l'applicazione osservabile

Concetti

- stato healthy/unhealthy
- dipendenza tra servizi
- limiti di `depends_on`

Come verificare

1. `docker ps` mostra lo status con timestamp e la salute (healthy)
2. cercare la sezione "Health" in `docker inspect guestbook_mysql_db`

```
docker inspect my_mysql_db | jq '.[0].State.Health.Status'
```

Passi aggiuntivi consigliati

9. Variabili d'ambiente vs configurazione hard-coded

- `.env`
- override per ambienti diversi

10. Logging

- `docker logs`
- rotazione log con Compose
- separazione log/applicazione

11. Arresto corretto dell'applicazione

- `SIGTERM`
- `stop_grace_period`
- comportamento Flask/Gunicorn

12. Sicurezza di base

- utente non root
- filesystem read-only
- esposizione minima delle porte