Capitolo 4 – Gestione delle Immagini Docker e Dockerfile

In questo capitolo vedremo come:

- Cercare immagini Docker
- Scaricare e gestire immagini locali
- Creare immagini personalizzate
- Scrivere un Dockerfile
- Ottimizzare le immagini

4.1 Cosa sono le immagini Docker

Un'immagine Docker è un template immutabile utilizzato per creare container.

Le immagini:

- Sono composte da layer (livelli)
- Sono versionate tramite tag
- Possono essere archiviate in registry pubblici o privati

Il registry pubblico predefinito è Docker Hub:
https://hub.docker.com/

4.2 Ricerca di un'immagine

Per cercare un'immagine nel registry:

```
docker search nginx
```

L'output mostrerà:

- NAME
- DESCRIPTION
- STARS
- OFFICIAL

Le immagini ufficiali sono contrassegnate come `OFFICIAL`.

4.3 Scaricare un'immagine

Per scaricare un'immagine:

```
docker pull nginx
```

Per scaricare una versione specifica:

```
docker pull nginx:1.25
```

Se non si specifica un tag, Docker utilizza automaticamente `latest`.

4.4 Visualizzare le immagini locali

Per vedere le immagini presenti nel sistema:

```
docker images
```

Oppure:

```
docker image ls
```

L'output mostra:

- REPOSITORY
- TAG
- IMAGE ID
- CREATED
- SIZE

4.5 Eliminare un'immagine

Per rimuovere un'immagine:

```
docker rmi nginx
```

Oppure tramite ID:

```
docker rmi <image_id>
```

Per forzare la rimozione:

```
docker rmi -f nginx
```

4.6 Creare un'immagine personalizzata con Dockerfile

Un Dockerfile è un file di testo che contiene le istruzioni per costruire un'immagine personalizzata.

Esempio base:

```
FROM debian:12

RUN apt update && apt install -y nginx

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

Salvare il file con nome:

```
Dockerfile
```

4.7 Costruire un'immagine

Dalla directory dove si trova il Dockerfile:

```
docker build -t mio-nginx:1.0 .
```

Spiegazione:

- `-t` assegna nome e tag
- `.` indica il contesto di build (directory corrente)

Verifica:

```
docker images
```

4.8 Avviare un container dalla propria immagine

```
docker run -d -p 8080:80 --name webtest mio-nginx:1.0
```

Parametri utilizzati:

- `-d` esecuzione in background
- `-p 8080:80` mappatura porta host → container
- `--name` assegna un nome al container

Aprire nel browser:

```
http://localhost:8080
```

4.9 Comprendere i layer delle immagini

Ogni istruzione come:

- RUN
- COPY
- ADD

genera un layer.

Per visualizzare la cronologia:

```
docker history mio-nginx:1.0
```

Meno layer inutili significano immagini più leggere ed efficienti.

4.10 Ottimizzazione delle immagini

Unire i comandi RUN

Meglio scrivere:

```
RUN apt update && \
    apt install -y nginx && \
    apt clean && \
    rm -rf /var/lib/apt/lists/*
```

Usare immagini leggere

Esempi:

- alpine
- debian:slim

Multi-stage build (concetto base)

Permette di costruire in uno stage e copiare solo il necessario nell'immagine finale.

Esempio:

```
FROM golang:1.22 AS builder
WORKDIR /app
COPY . .
RUN go build -o app

FROM debian:12-slim
WORKDIR /app
COPY --from=builder /app/app .
CMD ["./app"]
```

Questo riduce la dimensione finale dell'immagine.

4.11 File .dockerignore

Permette di escludere file dal contesto di build.

Esempio:

```
.git
node_modules
*.log
```

4.12 Best Practice

- Specificare sempre il tag dell'immagine
- Evitare `latest` in produzione
- Ridurre il numero di layer
- Eliminare file temporanei durante la build
- Utilizzare immagini ufficiali quando possibile

Conclusione

Ora sai:

- Gestire immagini Docker
- Creare Dockerfile
- Costruire immagini personalizzate
- Ottimizzare le build

Nel prossimo capitolo vedremo la gestione avanzata dei container, inclusi volumi, reti e limiti di risorse.

Revision #1
Created 2026-02-27 16:35:31 UTC by Pe
Updated 2026-02-27 16:35:57 UTC by Pe