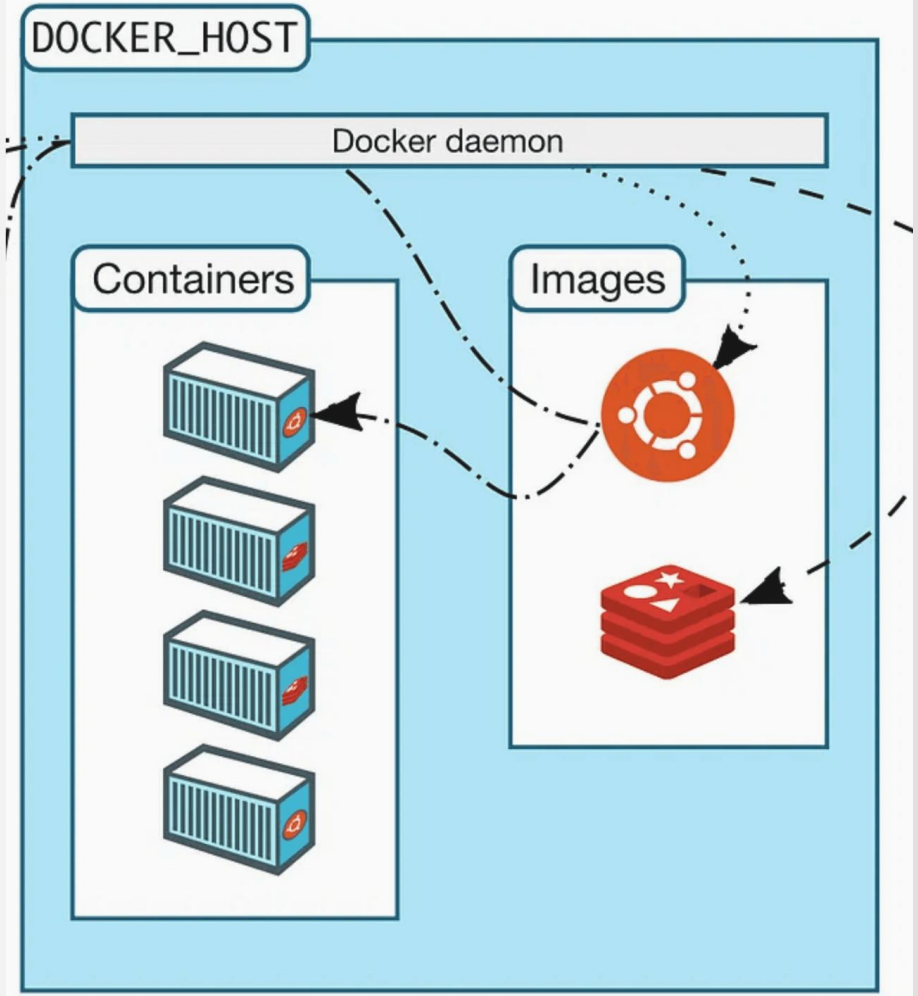
Spring Boot su Docker

Guida Pratica:

Dalla Teoria alla Produzione



Perché Docker per Spring Boot?

- **Consistenza Ambientale**

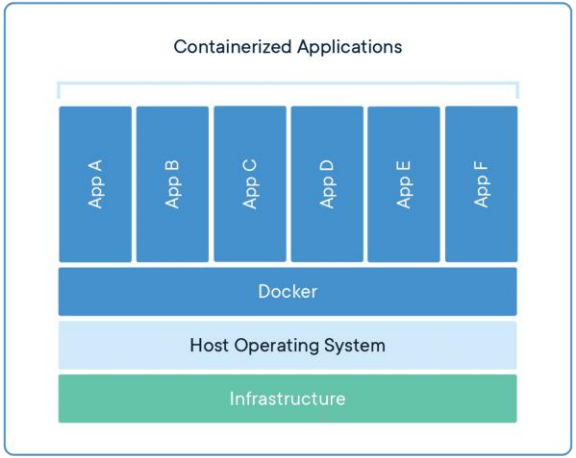
  Elimina il problema "funziona sulla mia macchina". L'ambiente è identico in sviluppo e produzione.

- **Scalabilità Semplice**

  Avvia multiple istanze in secondi per gestire picchi di traffico con orchestrazione orizzontale.

- **Isolamento Sicuro**

  Ogni applicazione ha le proprie dipendenze e runtime, riducendo conflitti e rischi di sicurezza.



Prerequisiti e Strumenti

Runtime

Docker Desktop

Essenziale per eseguire container locali. Include Docker Engine e CLI.

Java JDK

Versione 17 o 21 consigliata per nuove applicazioni Spring Boot.

Build

Maven

Standard de facto. Gestione dipendenze e plugin per build immagini.

Gradle

Alternativa performante. Sintassi Groovy/Kotlin e build incrementali.

Sviluppo

IDE Moderno

IntelliJ IDEA, VS Code o STS con plugin Docker installati.

Git

Per il versionamento del codice e dei file di configurazione Docker.

Dockerfile Semplice

Primo approccio alla containerizzazione

Dockerfile

```
FROM eclipse-temurin:21-jdk-alpine
ARG NOME_FILE=jarSpring/*.jar
COPY ${NOME_FILE} app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

FROM

Definisce l'immagine base (OpenJDK su Alpine Linux).

COPY

Copia il file JAR compilato dal sistema host al filesystem del container.

ENTRYPOINT

Specifica il comando esatto per avviare l'applicazione Java.

```
$ docker build -t myapp:latest .
 $ docker run -p 8080:8080 myapp:latest
```

Esecuzione del Container

Base

```
docker run -p 8080:8080 myapp:latest
```

-p Port Mapping

Mappa la porta del container (destra) sulla porta dell'host (sinistra).

```
-p 80:8080 (Host 80 -> Container 8080)
```

-d Detached Mode

Esegue il container in background e stampa l'ID del container.

```
docker run -d myapp:latest
```

--name Container Name

Assegna un nome mnemonico al container per una gestione più semplice.

```
--name my-spring-app
```

-e Environment Variables

Passa variabili d'ambiente al container (es. profili Spring).

```
-e SPRING_PROFILES_ACTIVE=prod
```

Gestione dei Container

Ciclo di Vita

```
docker stop <id>
```

Arresta gentilmente un container in esecuzione (SIGTERM).

```
docker start <id>
```

Avvia un container precedentemente arrestato.

```
docker restart <id>
```

Riavvia un container (Stop + Start).

Manutenzione

```
docker ps -a
```

Lista tutti i container (inclusi quelli fermi).

```
docker rm <id>
```

Rimuove definitivamente un container fermo.

```
docker exec -it <id> bash
```

Apre una shell interattiva dentro il container.

Ispezione

```
docker logs -f <id>
```

Visualizza e segue i log dell'applicazione in tempo reale.

```
docker inspect <id>
```

Mostra i dettagli JSON (IP, volumi, config).

```
docker stats
```

Monitora CPU, RAM e I/O di rete in tempo reale.

Docker Compose

Orchestrazione Multi-Container

Services

Definisce i container dell'applicazione (es. app, db) come servizi interconnessi.

Networking Automatico

I servizi possono comunicare tra loro usando il nome del servizio come hostname.

Depends On

Gestisce l'ordine di avvio dei servizi (es. il DB deve partire prima dell'app).

```
$ docker-compose up -d
 $ docker-compose down
```

compose.yml

```
version:'3.8’
services:
   nome-custom-service:
      container-name: unless-stopped
      build:
         #cartella dove si trova il Dockerfile
         context: ./nomeCartella
         #nome del Dockerfile
         dockerfile: nomeDockerFile
      ports:
         - "80:8080“
      #se per qualsiasi motivo si interrompe il container, riavvia
      restart: unless-stopped
```

Connessione Esterna

Per accedere a risorse all'esterno del container (ad esempio il db installato in locale) docker utilizza delle convenzioni a seconda dell'ambiente in cui ci troviamo

Ip Esposto

http://host.della.macchina:portaDellaMacchina

Macchina Windows

http://host.docker.internal: portaDellaMacchina

Macchina Linux

http://192.168.1.100:portaDellaMacchina